

TESTIRANJE SOFTVERA



TESTIRANJE SOFTVERA

Предиспитне обавезе	Обавезна	Поена
Одбрана пројекта	Да	70.00

Завршни испит	Обавезна	Поена
Усмени део испита	Да	30.00

TESTIRANJE SOFTVERA

Aktivnost koja je važna i bitna u celom procesu razvoja softvera.

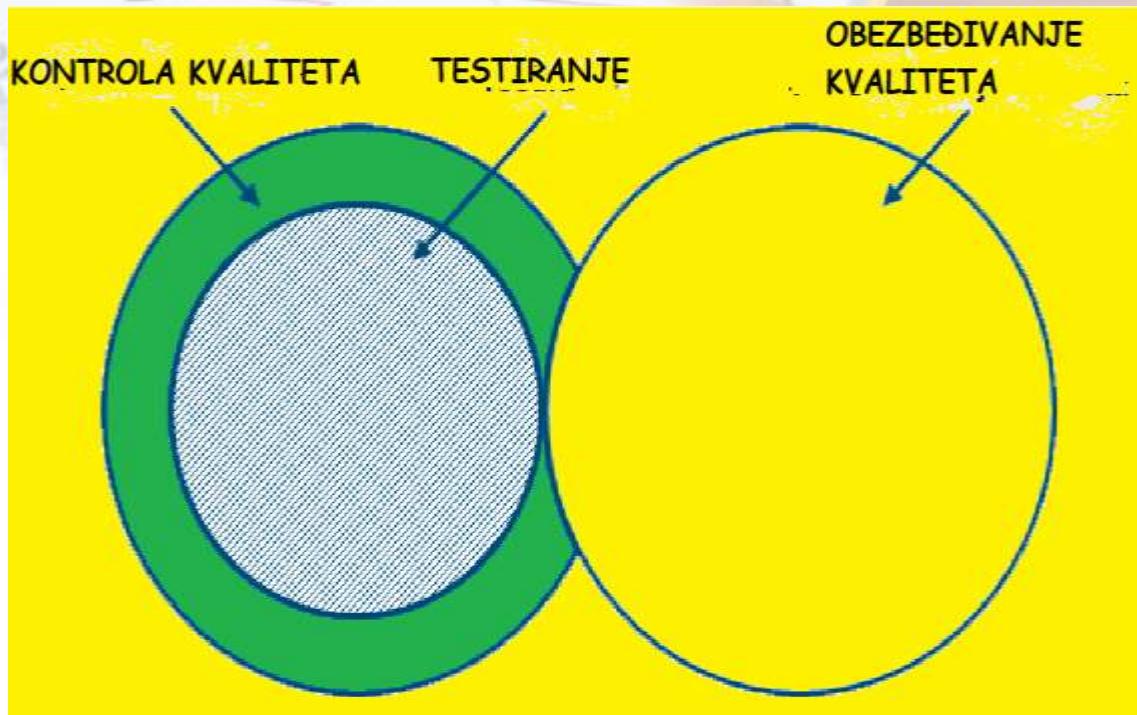
- ✓ Šta se mora testirati u softveru ?
- ✓ Ko bi trebalo da testira softver?

Neke od glavnih aktivnosti testiranja softvera su:

- ✓ Planiranje,
- ✓ Analiza i dizajn testova,
- ✓ Implementacija
- ✓ Izvršavanje test slučajeva
- ✓ Dokumentacija.

Jedan od glavnih ciljeva u testiranju softvera je detektovanje grešaka.

TESTIRANJE SOFTVERA



- KVALITET SOFTVERA
- TESTIRANJE SOFTVERA

TESTIRANJE SOFTVERA

Veliki su gubici kompanije usled grešaka u softveru koji su uslovili razvoj oblasti krivične odgovornosti proizvođača softvera i zaštite korisnika zbog neadekvatnog kvaliteta softvera.

Softver nije adekvatno testiran.

TESTIRANJE SOFTVERA

Neki primeri softverskih otkaza

- ❖ Zamalo nuklearni rat (1983)
- ❖ Prvi internet crv - program koji se sam umnožava i širi, 1988 zaraženo nekoliko hiljada računara, bug u virusu prouzrokovao zagušenje računara,
- Autor R.T.Moris sada profesor na MIT-u

TESTIRANJE SOFTVERA

- ❖ Therac 25 - računarski kontrolisani uređaj za terapijsku radijaciju: (1985. i 1987)
- ❖ AT&T - prekid u pozivima na velikim rastojanjima (1990)
- ❖ Aerodrom u Denveru (1994)
- ❖ Deutsche Telekom: pogrešan proračun cene telefonskih impulsa za 1996 (softverska greška)
- ❖ Mariner I, prva svemirska raketa za Veneru: 1962
- ❖ Ariane 5 let 501, (1996)
- ❖ NASA - Mars Climate Orbiter (1998)

TESTIRANJE SOFTVERA

- ❖ Milenijumski bag
- ❖ Greška fleš memorije skoro sprečava šetanje Mars Spirita, (2004)
- ❖ Greška alokacije memorije prekida produženu misiju Mars Global Surveyora, (2006)
- ❖ Windows XP SP1
- ❖ Windows XP SP2
- ❖ Windows XP SP3
- ❖ Amazon prodaja u iznosu od jednog centa, (2014)

TESTIRANJE SOFTVERA

- ❖ Ranjivost koja se tiče Intel-ovih CPU-ova, (2018)
- ❖ Boing 737 MAX, (2019)
- ❖ WINDOWS 10, (2020)
- ❖ Minecraft, (2021)
- ❖ Log4j, (2021)
- ❖ Grčka željeznica, (2023)
- ❖ WordPress, greške u plugin

GREŠKE U KRITIČNIM SOFTVERIMA

TESTIRANJE SOFTVERA

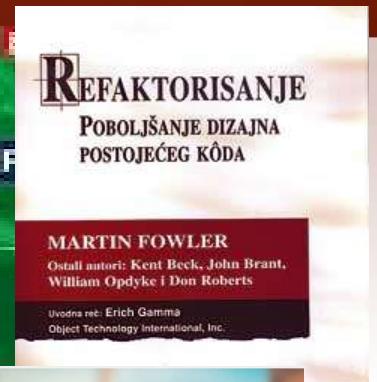


TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA

- Koncepti testiranja softvera
- Životni ciklus razvoja softvera i zašto je testiranje bitno ?
- Kada početi sa testiranjem softvera ?
- Metode testiranja softvera
Testni slučaj, testni plan, pravilno pisanje testnih slučajeva



TESTIRANJE SOFTVERA

C.A.R. Hoare

"Jedan od mojih najproduktivnijih dana je bio kada sam bacio 1000 linija koda."

Ken Thompson

"Izbrisani kod je debagovan kod. "



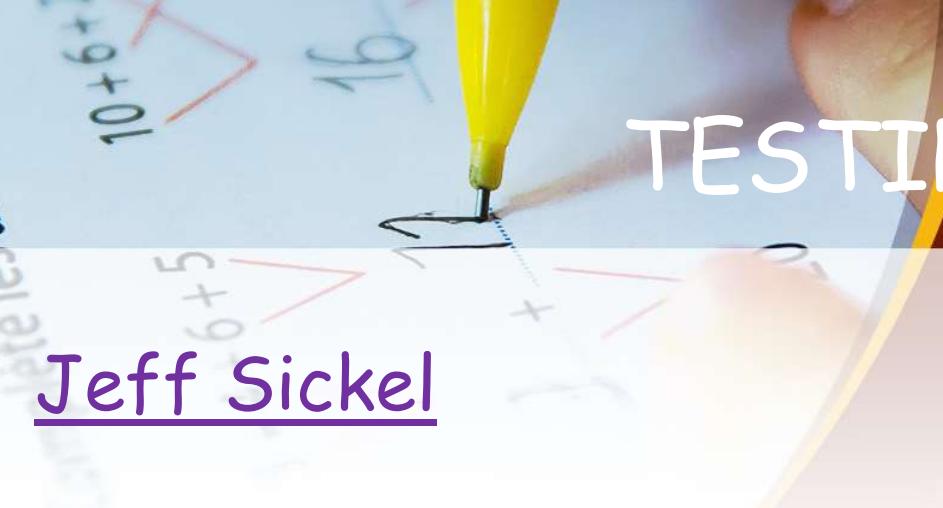
Doug Gwyn

"Program koji dvostruko brže daje netačan rezultat je beskonačno sporiji."



Linus Torvalds

"U kratkoj istoriji programiranja niko nikada nije napisao savršen komad koda. Male su šanse da čete biti prvi."



TESTIRANJE SOFTVERA

Jeff Sickel

"Debagovanje je dvostruko teže od pisanja koda.
Ako si napisao kod najpametnije što možeš, onda,
po definiciji, nisi dovoljno pametan da ga
debaguješ."



TESTIRANJE SOFTVERA

Brian W. Kernighan i P. J. Plauger u
The Elements of Programming Style.

“Kontrola kompleksnosti je suština
kompjuterskog programiranja”



TESTIRANJE SOFTVERA

- E. W. Dijkstra

"Postoje dva načina izgradnje softvera:
Jedan je napraviti ga toliko prostim da očigledno
nema nedostaka, a drugi je napraviti ga toliko
komplikovanim da nema očiglednih nedostataka.
Prva metoda je puno teža. "



Charles M. Strauss

"Ne možeš verovati kodu koji nisi u potpunosti sam napisao. "

Ken Thompson

"Jednostavnost je vrhunska prefinjenost. "



TESTIRANJE SOFTVERA

E. W. Dijkstra

“Najveći izazov za programera je da se ne zbuni pred kompleksnošću sopstvene tvorevine.”

· Brian Kernighan

“Kontrola kompleksnosti je suština kompjuterskog programiranja.”



TESTIRANJE SOFTVERA

Daniel Geer

"Glavni neprijatelj pouzdanosti je složenost."

John Osterhout

"Program koji dvostruko brže daje netačan rezultat je beskonačno sporiji."

CHROS Wenham

"Vreme debagovanje se povećava kvadratno u odnosu na veličinu programa."



TESTIRANJE SOFTVERA

Edsger W. Dijkstra

"Ceniti programe po broju linija koda je kao ceniti avione po težini."

Bill Gates

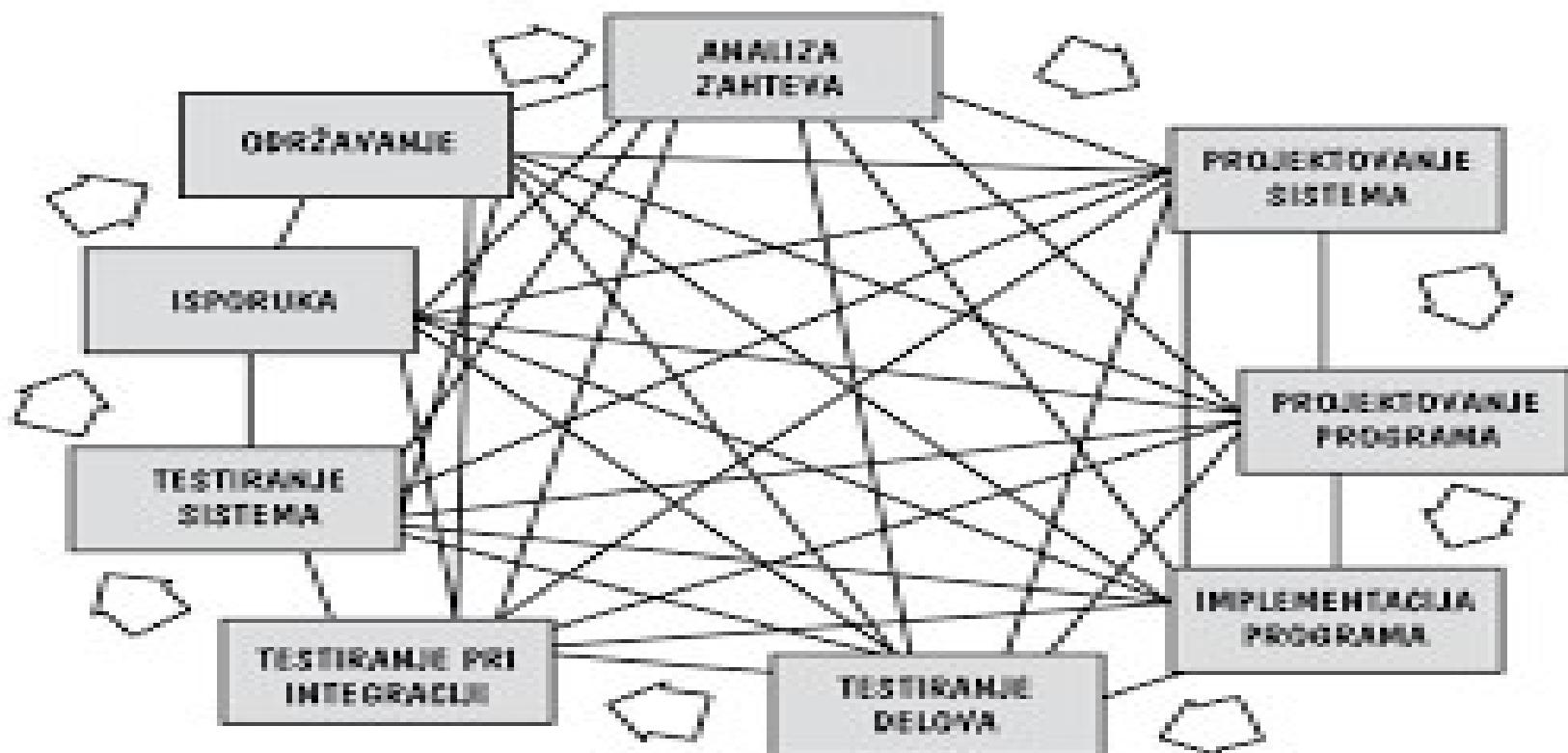
"Prvo, reši problem. Onda, piši kod."



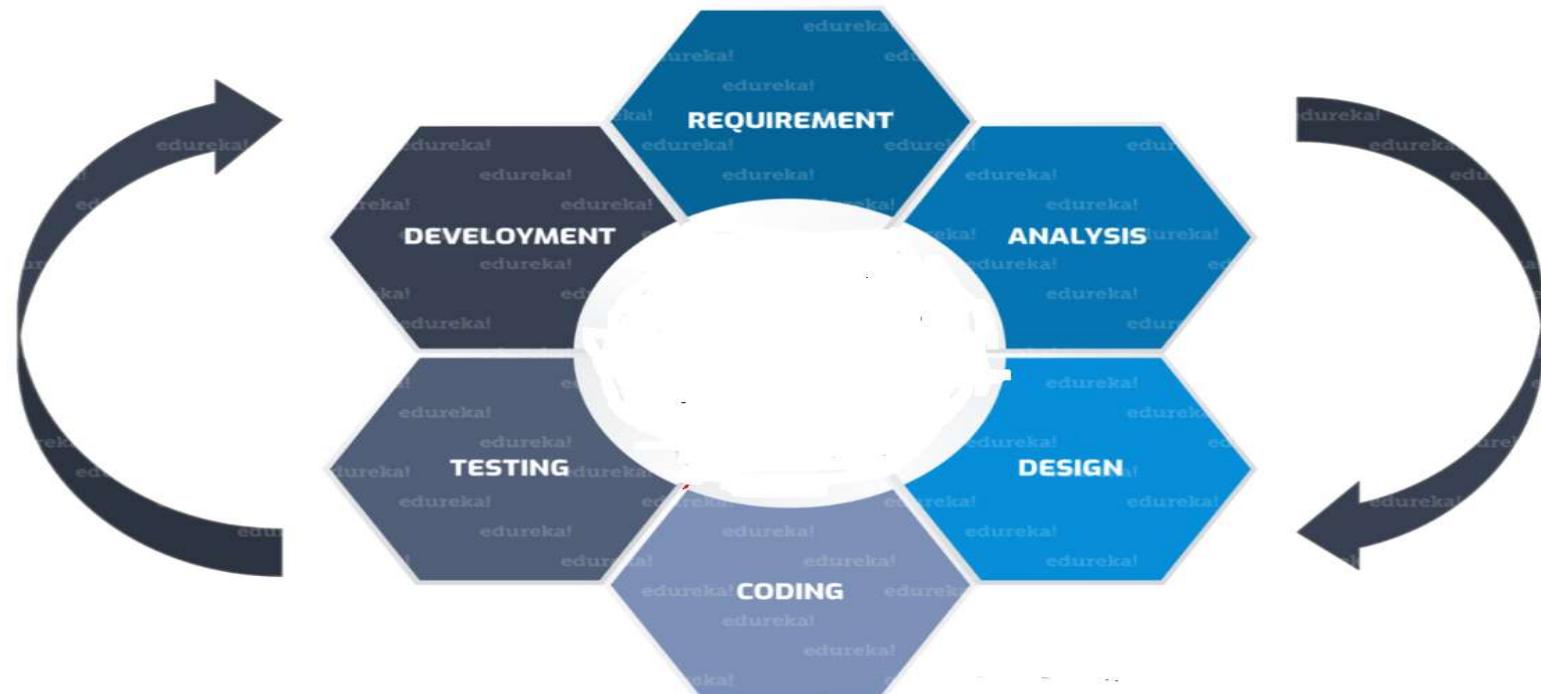
TESTIRANJE SOFTVERA

- ✓ Testiranje, kao jedna od najznačajnijih faza u procesu razvoja softvera, zahteva posebnu pažnju, koncentraciju i vreme.
- ✓ Testiranje kao proces, za cilj ima **pronalaženje i uočavanje** svih, pa i najsitnijih, grešaka (bagova) koje se mogu pojaviti pre i u toku korišćenja određenog softvera.

TESTIRANJE SOFTVERA



TESTIRANJE SOFTVERA





- ✓ Kako softver postaje sve kompleksniji i kako (razumni) zahtevi korisnika postaju sve veći stvara se sve veća potreba da se dobije "Kvalitetan softver" u smislu pouzdanosti, da je jednostavan za održavanje i dodavanje novih fukcionalnosti , da je otporan na greške (bagove)...
- ✓ U razvoju softvera greške (bagovi) se mogu pojaviti u bilo kojoj fazi životnog ciklusa.
- ✓ **NAJVAŽNIJE JE :** Da se Testiranjem softvera želi postići da softver bude pouzdan i siguran za korištenje.



TESTIRANJE SOFTVERA

- Kvalitetno testiranje znači kvalitetniji softver-

A šta može da se desi ako softver nije detaljno i kvalitetno testiran?

NAJBITNIJE:

Poverenje klijenata

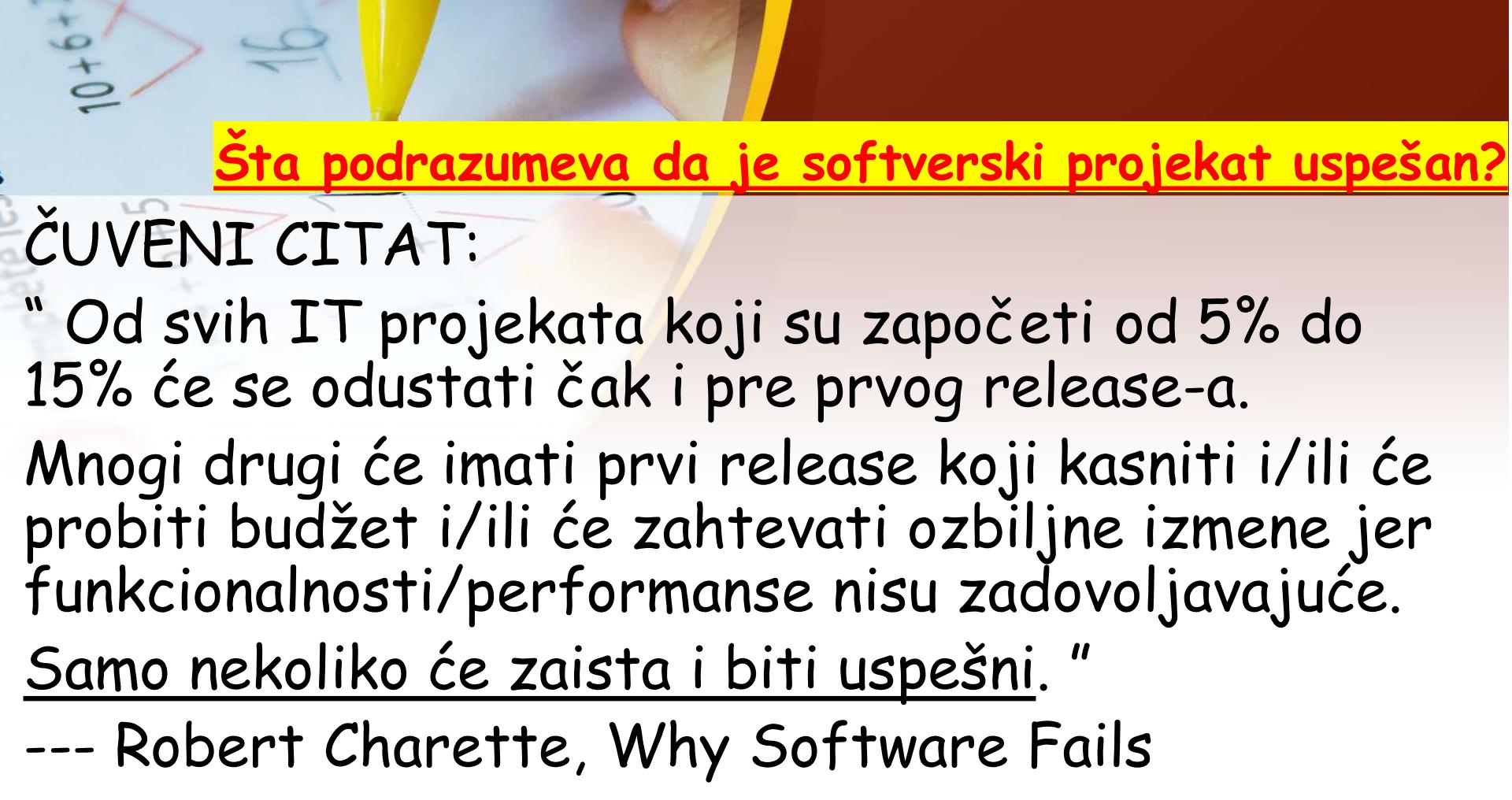


TESTIRANJE SOFTVERA

- ✓ Testiranje svih mogućih kombinacija ulaza i preduslova nije moguće u praksi.
- ✓ Radi se analiza rizika i testovi se izvršavaju prema prioritetu



- ✓ "Odsustvo grešaka ne garantuje da sistem radi kako treba - pronalaženje i ispravljanje defekata ne pomaže ako je sistem neupotrebljiv ili ako ne ispunjava zahteve korisnika."
- ✓ Citat od Wirth-a



Šta podrazumeva da je softverski projekat uspešan?

ČUVENI CITAT:

" Od svih IT projekata koji su započeti od 5% do 15% će se odustati čak i pre prvog release-a.

Mnogi drugi će imati prvi release koji kasniti i/ili će probiti budžet i/ili će zahtevati ozbiljne izmene jer funkcionalnosti/performanse nisu zadovoljavajuće.

Samo nekoliko će zaista i biti uspešni."

--- Robert Charette, Why Software Fails



Šta podrazumeva da je softverski projekat uspešan?

✓ Ako je Softver isporučen na vreme

- Troškovi razvoja su u okviru planiranog budžeta
- Softver je zadovoljio razumne zahteve korisnika po pitanju:

Funkcionalnosti softvera

Kvaliteta softvera



TESTIRANJE SOFTVERA

- Kada testirate softver da li se
Vi zapitate: "**Koliko košta moj
(propušteni) bug?**"



- Zašto softver nakon nekog vremena prestaje da se koristi ?

ILI

- Zašto softver nakon nekog vremena prestaje da radi ?

TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA

- Po nekim radovima i istraživačima, velika većina (autora) tvrdi da ovaj deo Sofverske industrije (IT industrije) ubedljivo najbrže raste ili bolje reći da je to jedna od najperspektivnijih oblasti.
- Po njima testeri igraju jednu od ključnih ulogu u Sofverskoj industriji jer je njihov zadatak da utvrde da li softverski proizvod odgovara unapred postavljenoj specifikaciji ili ne.
- Jednostavno - Greška, error, bug...

To je ono što ugrožava „život“ sajtova, web ili mobilnih aplikacija, igrica, različitih programa i softvera,



TESTIRANJE SOFTVERA

- Svi projekti bez izuzetka imaju definisane finansijske limite i rokove, odnosno definisani budžet za sam projekat i NARAVNO vremenske rokove kada sistem mora biti instaliran i potpuno operativan.
- KVALITET SOFTVERA



TESTIRANJE SOFTVERA

- U literaturi iz oblasti softverskog inženjerstva opisani su mnogi modeli procesa izrade softvera.
- Proces razvoja softvera, kroz uključenje potprocesa koji doprinose boljem razumevanju, pomaže u procesu kontrolisanja nedostataka.
- Izrada prototipova je jedan takav potproces. Prototip predstavlja delimično razvijen proizvod koji omogućava naručiocima i projektantima da ispitaju neke aspekte predloženog sistema i odluče da li je on pogodan ili potreban u sklopu gotovog proizvoda.



Održavanje softvera

Održavanje softvera se baziran na procesu modifikacije softvera gde se on može nadograđivati i prilagođavati potrebama korisnika (kroz duži vremenski period.)

PREVIŠE NOVIH VERZIJA ZA KRATAK VREMENSKI ROK NIJE DOBRO (SA ASPEKTA KORISNIKA)

Klasifikacija Swansona i Lientza (Swanson & Lientz, 1980) je jedna od najcitanijih, koja kaže da se održavanje softvera može podeliti na 4 kategorije: **Korektivno, Adaptivno, Perfektivno i Preventivno održavanje**



Održavanje softvera

❖ Adaptivno održavanje

Modifikacija softverskog proizvoda koja se izvodi nakon isporuke, a sa ciljem da se tom softverskom proizvodu sačuva upotrebna vrednost u promenjenoj sredini ili sredini koja se upravo menja.

❖ Korektivno održavanje

Reaktivna modifikacija softverskog proizvoda koja se vrši nakon isporuke, radi popravke otkrivenih grešaka.



Održavanje softvera

Prema standardu ISO/IEC 14674(...)

❖ Perfektivno održavanje

Modifikacija softverskog proizvoda nakon isporuke radi unapređenja performansi ili održivosti .(takođe uključuje i dodavanje novih karakteristika).

❖ Preventivno održavanje

Modifikacija softverskog proizvoda nakon isporuke, sa ciljem da se detektuju i isprave skrivene greške u tom softverskom proizvodu pre nego da postanu delotvorne.



Održavanje softvera

- Održavanje softvera u softverskom inženjerstvu predstavlja modifikacija softverskog proizvoda nakon isporuke sa ciljem da ispravi greške, da bi poboljšali pre svega performance.
- Održavanje softvera obuhvata sve modifikacije na softveru nakon njegove isporuke, sve do trenutka njegovog povlačenja iz upotrebe.

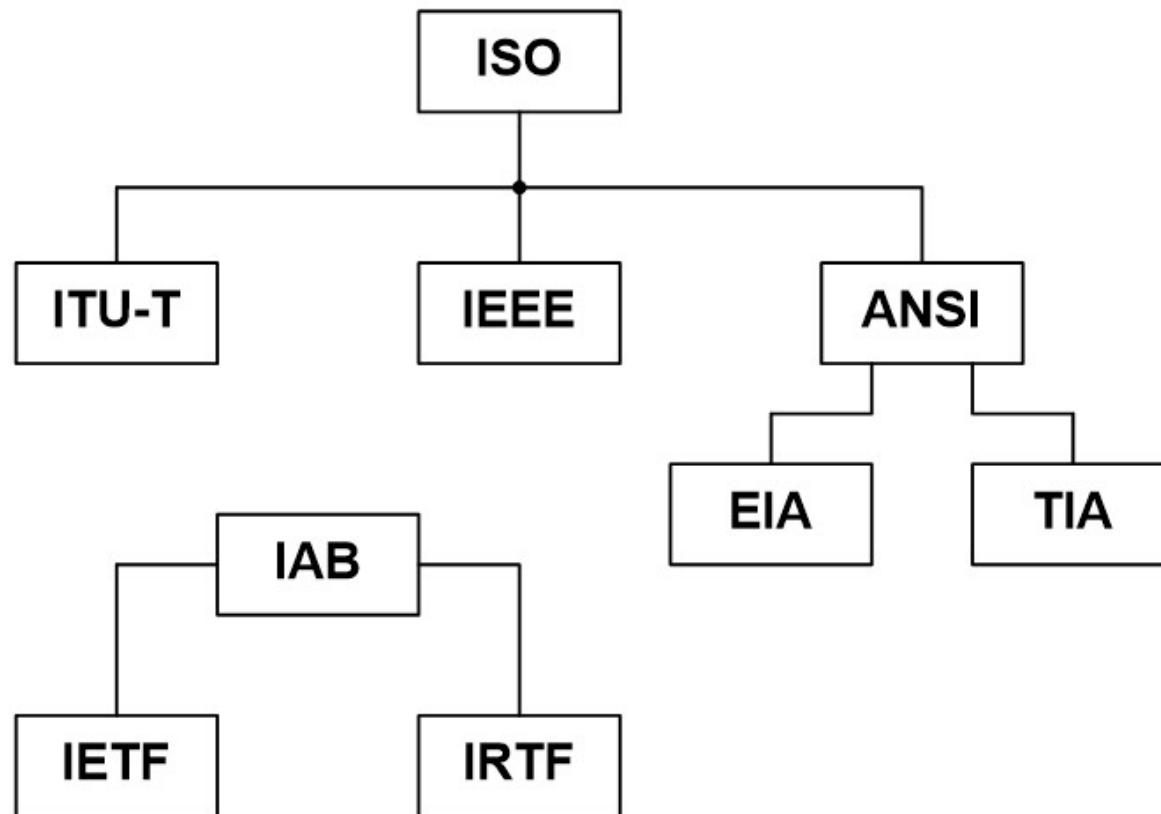


Održavanje softvera

Vrlo često kvaliteta i održivost softvera njegovim korišćenjem (sa godinama) opadaju.

Održavanje ne bi trebalo da uključuje velike promene (po pravilu) na arhitekturi samog sistema. Promene se implementiraju modifikovanjem postojećih komponenti i dodavanjem novih komponenti u sam system.

Organizacije zadužene za donošenje standarda





STANDARDI

Proces modifikacije softverskog sistema ili komponente nakon isporuke radi ispravljanja grešaka, poboljšanja performansi ili drugih atrubuta ili prilagodjenja novoj sredini

[IEEE Std 610.12-1999]

Softverski proizvod podleže modifikaciji koda i odgovarajuće dokumentacije usled nekog problema ili potrebe za poboljšanjem.

Cilj toga je da se modifikuje postojeći softverski proizvod a da se u isto vreme sačuva njegov integritet.

[ISO Std 12207]



STANDARDI

International Standards Organization (ISO)- je internacionalna organizacija za standardizaciju širokog delokruga.

Serija standarda ISO 9000 nastala je sa ciljem da se kreira skup opštih standarda za upravljanje kvalitetom i da se obezbedi kvalitet sofvera.



STANDARDI

JEDAN OD JAKO BITNIH STANDARDA je:

ISO standard 9126

On obezbeđuje hijerarhijski okvir za definiciju kvaliteta, organizovanu kao karakteristike i podkarakteristike kvaliteta.



ISO standard 9126

Prema ovom standardu postoji šest karakteristika:

- **Funkcionalnost**

postojanost funkcija softvera

- **Pouzdanost**

sposobnost softvera da održava nivo performansi pod navedenim uslovima, u toku navedenog vremenskog perioda

- **Upotrebljivost**

pogodnost za korišćenje softvera, odnosno kompleksnost za krajnjeg korisnika



ISO standard 9126

Prema ovom standardu postoji šest karakteristika:

- **Efikasnost**

odnos između nivoa performansi softvera i količine resursa koji se koriste, pod navedenim uslovima

- **Održivost**

sposobnost softvera da jednostavno podrži buduće modifikacije

- **Prenosivost**

sposobnost softvera da bude prenesen iz jednog u drugo radno okruženje



ISO standard 9126

- **SUBLIMIRAMO!**

Prema ISO/IEC 9126 standardu moguće je utvrditi kvalitet softverske aplikacije (Model kvaliteta, eksterna metrika, interna metrika, pokazatelj kvaliteta u samoj upotrebi softvera)



STANDARDI

ANSI (American National Standards Institute) – Američki nacionalni zavod za standardizaciju, je neprofitna organizacija koja nadgleda razvoj standarda za proizvode, servise, procese i sisteme u SAD.

Ova organizacija takođe usklađuje standarde sa međunarodnim standardima, tako da se američki proizvodi mogu koristiti širom sveta.

ANSI je nacionalni standard Sjedinjenih Američkih Država koji propisuje Nacionalni zavod za standardizaciju (American National Standards Institute)



STANDARDI

- The *Institute of Electrical and Electronics Engineering* (IEEE) - internacionalna profesionalna organizacija osnovana u USA, a čine je inženjeri iz oblasti elektronike, računarstva, komunikacije i energetike.
- IEEE je trenutno najveće profesionalno udruženje.
- IEEE usko saradjuje sa ANSI u razvoju standarda za komuniciranje i procesiranje informacija sa osnovnim ciljem da se ostvari napredak na polju teorije, kreativnosti, i povećenju kvaliteta u bilo kojoj oblasti elektrotehnike

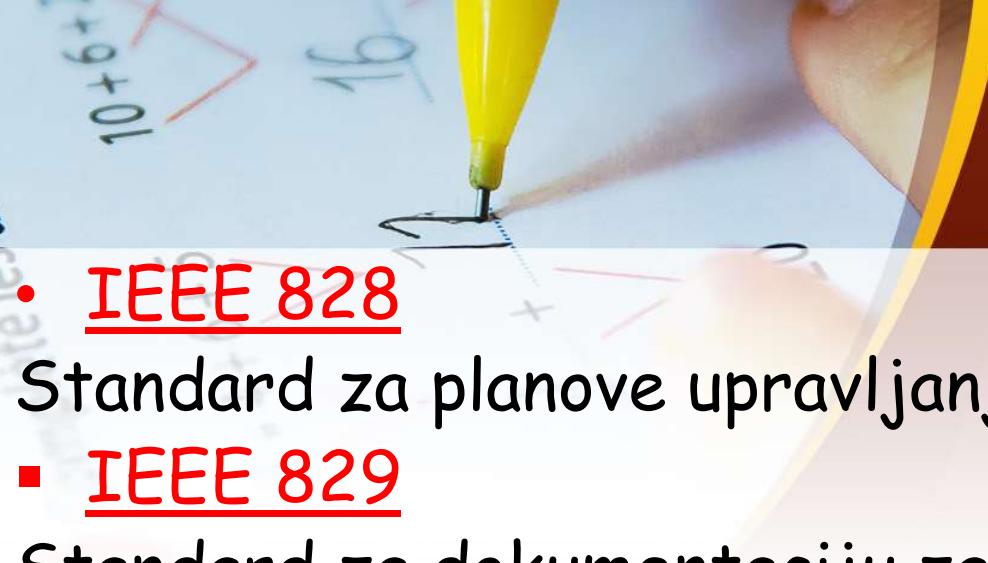


STANDARDI

- IEEE 730

Standard za planove obezbeđenja kvaliteta softvera:
Specifikacije zahteve softvera

- ✓ Opisa projekta softvera
- ✓ Plana softverske verifikacije i validacije
- ✓ Izveštaja o softverskoj verifikaciji i validaciji
- ✓ Korisničke dokumentacije
- ✓ Plana upravljanja softverskom konfiguracijom



STANDARDI

- **IEEE 828**

Standard za planove upravljanja dokumentacijom

- **IEEE 829**

Standard za dokumentaciju za softversko testiranje

- **IEEE 830**,

Preporučena praksa za zahteve softverske specifikacije

- **IEEE 1008**

Standard za jedinično (unit) testiranje softvera



STANDARDI

- **IEEE 1012**

Standard za planove softverske verifikacije i validacije

- **IEEE 1016**

Vodič za opis softverskog projekta

- **IEEE 1028**

Standard za softverske kontrole i audio

- **IEEE 1042**

Vodič za planove upravljanja softverskim konfiguracijama

- **IEEE 1044**

Standard klasifikacije softverskih anomalija



STANDARDI

- IEEE 1045,

Standard za metriku softverske produktivnosti

- IEEE 1058.1

Standard za planove upravljanje softverskim projektima

- IEEE 1059

Vodič za planove softverske verifiakcije i validacije

- IEEE 1061

Standardi za metriku metodologije kvaliteta softvera

- IEEE1063

• Standard za korisničku dokumentaciju softvera



STANDARDI

- **IEEE 1074**

Standard za razvoj SDLC Connected Limited Device Configuration (CLDC) procesa

- **IEEE 1219**

Standard za softversko održavanje

- **IEEE 1233**

Vodič za razvoj specifikacije zahteva sistema



Održavanje softvera

Starenje softvera

"Moramo naučiti kako da poništimo efekte(posledice) starenja softvera"- **PARNAS.**

"Programi, kao i ljudi, stare. Ne možemo da sprečimo zastarevanje, ali možemo razumeti njegove razloge, preduzeti korake da ograničimo njegove posledice, sa vremena na vreme poništiti neke od oštećenja koje je prouzrokovalo i pripremiti se za dan kada taj softver neće više biti održiv." **(Parnas, 1994.)**



Održavanje softvera

- Modifikacije softvera postaje sve teže sa povećanjem veličine softvera
 - Raste veličina koda koji treba promeniti
 - Sve je teže naći delove koje treba promeniti
 - Nemogućnost dodavanja nove funkcionalnosti
 - Nema dokumentacije
 - (Nema koda ili se ne zna kod koga je) / nemogućnost rada bilo šta
- Kao rezultat dešava se to da se korisnik prestane da koristi takav softver



Održavanje softvera

- **NEKI PROBLEMI KOD ODRŽAVANJA SOFTVERA**

- Iako se smatra jednim od ključnih delova životnog ciklusa softvera, postupak nije tako jednostavan
- PROBLEM : trošak samog održavanja
- Nedostatak informacija u analizi između korisnika i programera takođe može postati veliki izazov u održavanju softvera.
- Ako softver nije implementiran za jednostavne izmene, održavanje takvog softvera može da prouzrokuje puno problema.



Održavanje softvera

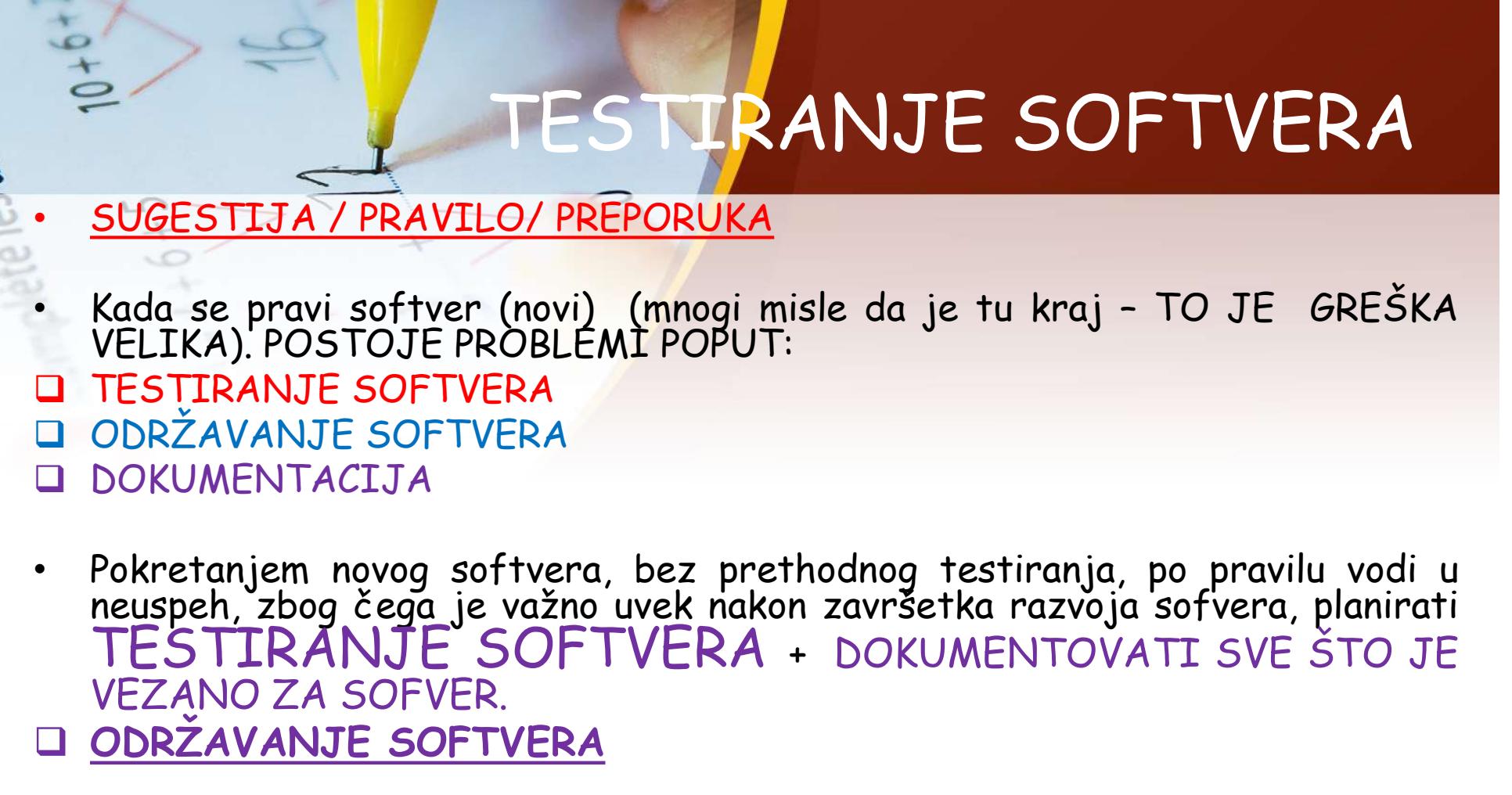
- NEKI PROBLEMI KOD ODRŽAVANJA SOFTVERA
- U zavisnosti od potrebe klijenta, često se susrećemo i sa najrazličitijim operativnim sistemima na kojima je potrebno izvršiti testiranje.
- Ne često je potrebno da se dobro poznaju što veći broj operativnih sistema
- Takođe od softverskog inženjera u oblasti testiranja očekuje poznavanje što većeg broja programski jezika. .



TESTIRANJE SOFTVERA

Softverska industrija danas se veoma brzo razvija i može se tvrditi da je potreba za novim aplikacijama sve veća.

Često zbog raznih faktora se softver bez adekvatnog i dobrog testiranja pre vremena stavlja na tržište i tu nastaju veliki problem za kompanije.



TESTIRANJE SOFTVERA

- SUGESTIJA / PRAVILA / PREPORUKA
- Kada se pravi softver (novi) (mnogi misle da je tu kraj - TO JE GREŠKA VELIKA). POSTOJE PROBLEMI POPUT:
 - TESTIRANJE SOFTVERA
 - ODRŽAVANJE SOFTVERA
 - DOKUMENTACIJA
- Pokretanjem novog softvera, bez prethodnog testiranja, po pravilu vodi u neuspeh, zbog čega je važno uvek nakon završetka razvoja softvera, planirati **TESTIRANJE SOFTVERA + DOKUMENTOVATI SVE ŠTO JE VEZANO ZA SOFVER.**
 - ODRŽAVANJE SOFTVERA



TESTIRANJE SOFTVERA

- Za softver se podrazumeva da on mora biti kvalitetan, siguran, bezbedan

KARAKTERISTIKE

- ✓ Mogućnost održavanja (Softver se mora menjati u skladu sa razumnim očekivanjima korisnika)
- ✓ Pouzdanost i sigurnost. (Softver se mora ponašati na predvidiv način)
- ✓ Efikasnost (Softver mora imati zadovoljavajuće performanse)
- ✓ Upotrebljivost. (Softver treba da radi ono što korisnici od njega očekuju)
- ✓ Dokumentacija (mora da postojati u pisanim obliku - što detaljnije napisana)



TESTIRANJE SOFTVERA

- ✓ Korisnički interfejsi
- ✓ Softverski interfejsi
- ✓ Hardverski interfejsi
- ✓ Komunikacioni interfejsi



SPECIFIKACIJA ZAHTEVA

MORA DA POSTOJI u pisnom obliku : (specifikacija softverskih zahteva). (Ova DOKUMENTACIJA JE NEOPHODNA)

Menadžer projekta

Koristi specifikaciju zahteva kako bi se uverio da su obuhvaćen svi zahtevi od strane korisnika i da razvojnom timu obezbedi opis sistema koji treba da razviju, kao i da prati sam razvoj sistema.

MORA DA POSTOJI u pisnom obliku : (specifikacija hardverskih zahteva). (Ova DOKUMENTACIJA JE NEOPHODNA)



SPECIFIKACIJA ZAHTEVA

- "Jedna od ideja samog kvalitet proizvoda je težnja da taj proizvod bude izrađen u skladu sa specifikacijom." (**Crosby 1979**).
- Kada testiramo program treba da koristimo i nespecifične ulazne podatke.

OBAVEZNA SUGESTIJA

- Unesite iste ulazne podatke (ili niz ulaza) veliki broj puta.
Ne retko postoji mogućnost da dobijamo različite rezultate.
- U takvoj situaciji-veliki problem za Testere



SPECIFIKACIJA ZAHTEVA

- Analiziraju se zahtevi korisnika. (Razumni zahtevi)
- Vrši se projektovanje koje određuje kako će softver raditi.
- Implementacija softvera
- Verifikacija i validacija. Proverava se da li softver radi prema zadatoj specifikaciji, odnosno da li radi ono što korisnik želi.
- Održavanje softvera
- Dokumentacija softvera

NEOPHODNO JE :

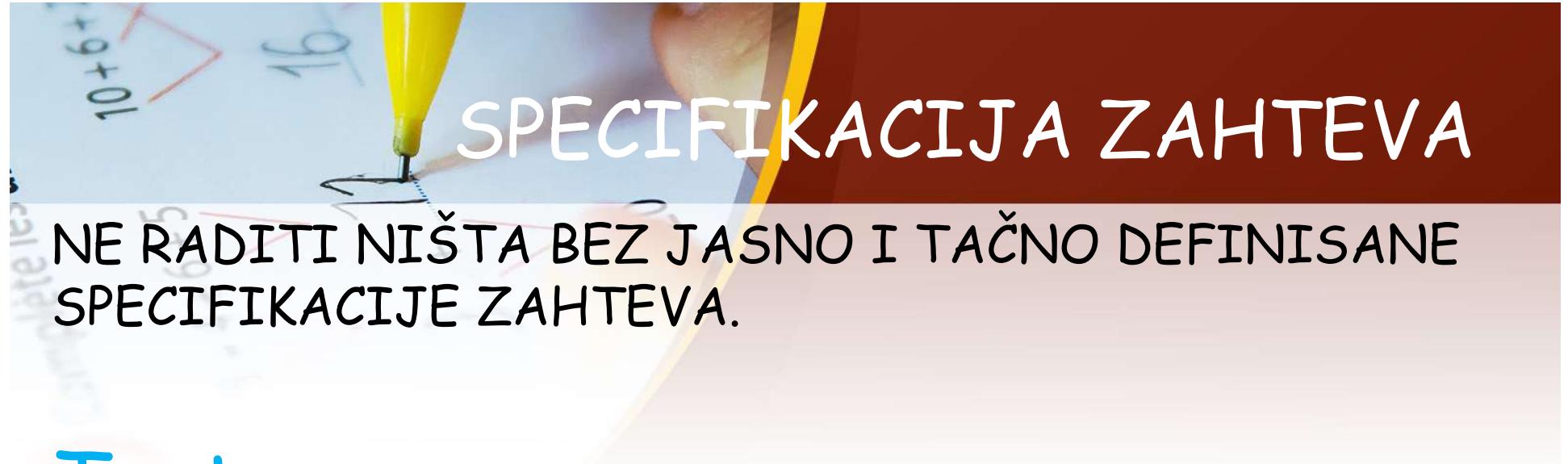
TESTIRATI I SAMO TESTIRATI U SVAKOJ FAZI



SPECIFIKACIJA ZAHTEVA

Faktori koji utiču na kreiranje pouzdane specifikacije pre samog razvoja (implementacije) softvera

- Korisnici nisu sigurni šta tačno žele
- Imaju problem da iskažu ono šta znaju i šta očekuju
- Mnogi detalji zahteva će se obelodaniti tek tokom razvoja
- Detalji su često jako komplikovani korisnicima
- Kako se softver razvija korisinci menjaju mišljenje
- Spoljašnji uticaj na promeni zahteva (promene u tržištu, konkurenciji, ...)



SPECIFIKACIJA ZAHTEVA

NE RADITI NIŠTA BEZ JASNO I TAČNO DEFINISANE
SPECIFIKACIJE ZAHTEVA.

Tester:

Koristi specifikaciju za verifikaciju i
validaciju traženih funkcija.



ANALIZA ZAHTEVA

- **Analiza zahteva** obuhvata one zadatke koji se izvode pri utvrđivanju potreba ili uslova da bi se ostvario novi ili izmenjeni proizvod ili projekat, uzimajući u obzir eventualno konfliktne zahteve različitih karaktera, analiza, dokumentovanja, procena i upravljanja softverom ili sistemske zahteve.



Testiranje nije aktivnost koja počinje samo nakon kompletiranja faze kodiranja.

Softversko testiranje se danas vidi kao aktivnost koja obuhvata ceo proces razvoja i održavanja i predstavlja važan deo kompletne konstrukcije softvera.

Planiranje testiranja treba da počne sa ranom fazom procesa uzimanja zahteva; i test planovi i procedure moraju biti sistematski i kontinualno razvijani i po potrebi redefinisani. Pravi stav prema kvalitetu je prevencija, mnogo je bolje izbeći probleme nego ih ispravljati.

TESTIRANJE SOFTVERA /NEKE DEFINIJE

“Testiranje softvera je organizovani proces u okviru razvoja softvera u kojem se proverava ispravnost, kvalitet i performanse softvera.”

“Testiranje softvera je aktivnost ili proces kojim se pokazuje da program ili sistem izvršava sve predviđene funkcije korektno.”

TESTIRANJE SOFTVERA /NEKE DEFINIJE

"Testiranjem softvera može samo da otkrije greške u samom kodu ali ne i da dokaže njihovo odsustvo."

"Teško je znati kada stati sa testiranjem i koliko testiranja je dovoljno za koji sistem."



TESTIRANJE SOFTVERA /NEKE DEFINIJE

“Testiranje predstavlja proces analiziranja stavki softvera radi otkrivanja razlika između postojećih i potrebnih uslova i procena same karakteristike softvera”

“Testiranje softvera je aktivnost koja se izvodi identifikacijom defekata i problema radi procene kvaliteta proizvoda, kao i za njegovo unapređenje.”



TESTIRANJE SOFTVERA /NEKE DEFINIIJE

“Testiranje softvera je proces koji se koristi da bi se utvrdila ispravnost, potpunost i kvalitet razvijenog softvera. Imajući to u vidu, testiranje nikada ne može u potpunosti da utvrdi ispravnost datog softvera.”

“Testiranje softvera je način da se obezbedi manji broj grešaka, manji trošak održavanja i na kraju sveukupne cene softvera.”



TESTIRANJE SOFTVERA /NEKE DEFINIIJE

"Efikasno testiranje softvera doprinosi isporuci kvalitetnog softverskog proizvoda koji zadovoljava potrebe, očekivanja i zahteve korisnika"

" To je proces izvršavanja programa sa ciljem nalaženja grešaka (G. Myers) "



TESTIRANJE SOFTVERA /NEKE DEFINIJE

"Paralelno sa testiranjem softvera potrebno je vršiti i proveru kvaliteta softvera, odnosno testa kroz koji će se krajnji korisnici uveriti da dobijaju softver koji odgovara kako njihovim, tako i generalnim zahtevima koji se odnose na kvalitet softvera."

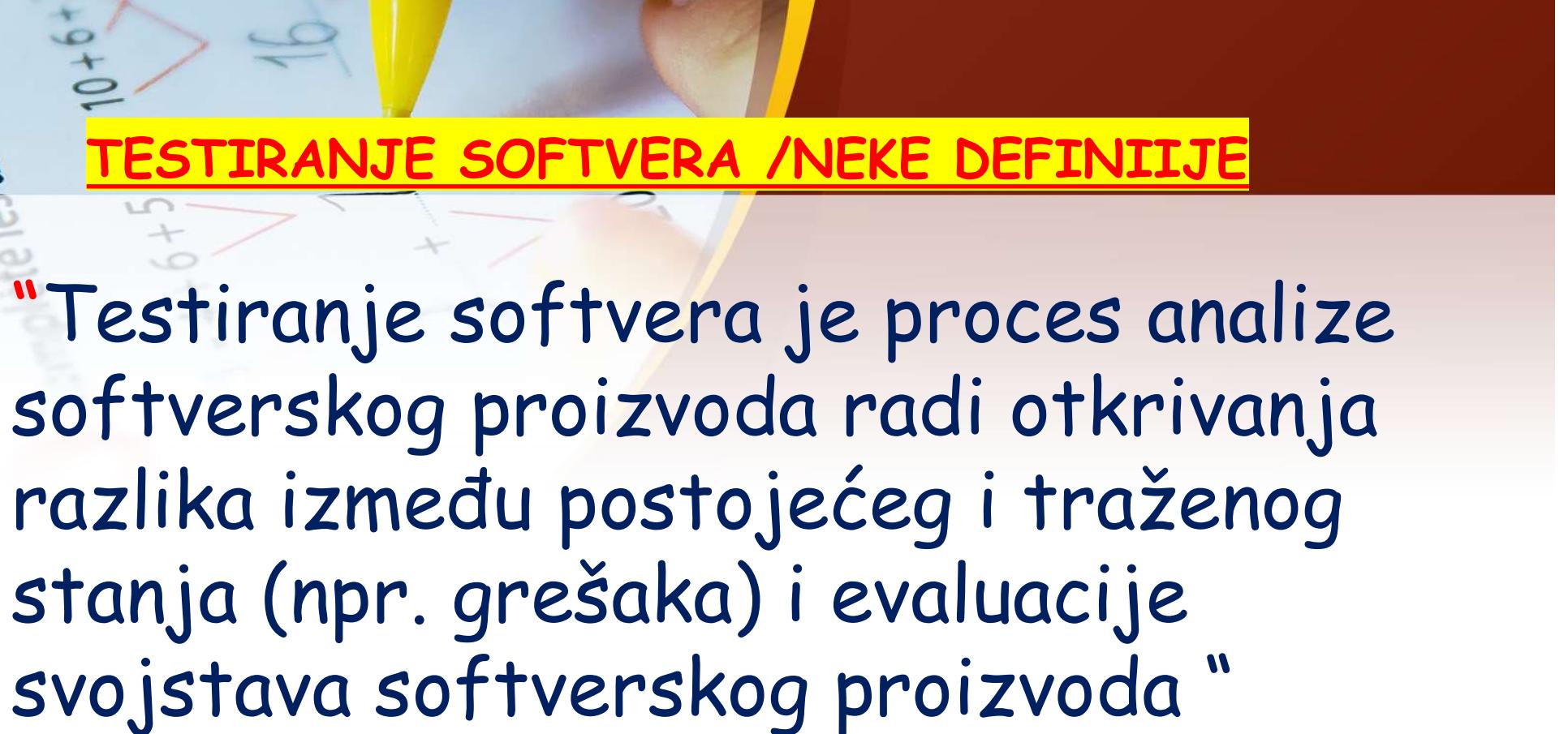
TESTIRANJE SOFTVERA /NEKE DEFINIIJE

"Testiranje softvera je simulacija načina rada sistema."

"Testiranje softvera je proces provere da li sistem radi ono za šta je predviđen."

"Testiranje softvera je analiza programa sa ciljem pronalaženja problema i grešaka."

"Testiranje softvera je merenje funkcionalnosti i kvaliteta sistema."



TESTIRANJE SOFTVERA /NEKE DEFINIIJE

“Testiranje softvera je proces analize softverskog proizvoda radi otkrivanja razlika između postojećeg i traženog stanja (npr. grešaka) i evaluacije svojstava softverskog proizvoda ”



"Testiranje softvera je procenjivanje svih atributa i mogućnosti programa, radi utvrđivanja da li oni daju zadovoljavajuće rezultate."

"Testiranje softvera obuhvata proveru samih zahteva projekta i da li dati kod radi ispravno."



"Testiranje softvera je obavezan korak u kreiranju svakog softverskog proizvoda "

"Testiranje softvera je postupak provere svih funkcionalnosti aplikacije radi utvrđivanja zadovoljava li navedene zahteve ili ne."

"Testiranje softvera je postupak pronašnja nedostataka u aplikaciji i provere gde aplikacija "radi" u skladu sa zahtevima krajnjeg korisnika."



"To je tehnika za kvalitet procene softverskog proizvoda."

"Testiranje softvera obezbeđuje kompatibilnost softverskog proizvoda sa, funkcionalnim i korisničkim zahtevima."

"Prvi utisci su važni, ne želite da rizikujete reputaciju tako što imate proizvod (proizvode) sa greškom."



TESTIRANJE SOFTVERA

- ✓ Predstavlja pokušaj da se pronađu greške u softveru koji je napravljen.
- ✓ Softver je implementiran prema korisničkim zahtevima kojima se rešava neki realni problem ili se kreira neka korisna funkcionalnost koja predstavlja nešto što je potrebno krajnjim korisnicima.



TESTIRANJE SOFTVERA

Software Testing and Quality Assurance predstavlja proces otkrivanja grešaka (bagova) u računarskim sistemima ili softverima.

Testiranje je veoma važna aktivnost u razvoju softvera koja je posebno dobila na značaju kada je vrednost softvera počela da raste.

TESTIRANJE SOFTVERA

- TESTERI -CILJ /Detaljno isplanirati ceo proces testiranja





TESTIRANJE SOFTVERA

- **Quality Assurance - QA Engineer**
- Obezbeđenje Kvaliteta (Quality Assurance) bavi se svim aspektima procesa proizvodnje softvera, osmišljavanjem metodologije i neophodnih procedura razvoja koje treba da dovedu do dogovorenog kvaliteta softverskog proizvoda.
- QA tester je jedna od ključnih i veoma važnih uloga u svakom delu razvojnog tima koji se bavi razvojem softvera.



TESTIRANJE SOFTVERA

SUGESTIJA

- Naučite tehnike i principe, a ne samo određene softverske alate sa kojima se nešto testira
- Testiranje podrazumeva **traženje grešaka**.
- Testerski proces je veoma dinamičan i ne retko zahteva logičko razmišljanje i brzo odlučivanje.
- Proces testiranja softvera ima za cilj ne samo pronalaženje grešaka u postojećem softveru, već i pronalaženje mera za poboljšanje softvera u pogledu efikasnosti, tačnosti i upotrebljivosti.
- Glavni cilj je **merenje specifikacija, funkcionalnosti i performansi** softverskog programa ili aplikacije



TESTIRANJE SOFTVERA

Metodologije su se međusobno smenjivale i menjale zbog porasta kompleksnosti programa, prvenstveno zbog naglog razvoja hardvera koji je pružao svoj maksimum tek kada je bio praćen odgovarajućim softverom.

Kompleksnost programa nije tako lako rešiti.



TESTIRANJE SOFTVERA

Sa rastom kompleksnosti realizovanih funkcija i primena posebno je narastao zahtev za kvalitetom softvera u pogledu pouzdanosti (pogotovo kod softvera sa kritičnom misijom), pogodnosti za testiranje i održavanje, ponovne upotrebljivosti, otpornosti na greške i drugih faktora kvalitetnog softvera



TESTIRANJE SOFTVERA

Razvoj i isporuka softvera mogu da budu komplikovani procesi, posebno kako aplikacije, timovi i infrastruktura postaju sve složeniji, kako projekti rastu.

Da bi se softver razvijao, testirao i isporučio na brz način i bez greške, programeri i organizacije su kreirali tri povezane ali različite strategije za upravljanje i automatizaciju ovih procesa a to su:

- kontinuirana integracija,
- kontinuirana isporuka i
- kontinuirano raspoređivanje.



TESTIRANJE SOFTVERA

- Potrebno je znati da pri izboru NEADEKVATNE METODOLOGIJE tok razvoja softvera može otići u potpuno "pogrešnom" smeru, te se i tekako može ugroziti završavanje softvera / projekta.
- KVALITET softverskog proizvoda određen je brojnim faktorima, kao što su na primer pre svih :
- saglasnost sa funkcionalnom i programskom specifikacijom,
- održavanje,
- skalabilnost,
- portabilnost,
- dokumentacija i dr
- Pouzdanost softvera je bitna karakteristika kvaliteta



Koncept apstrakcije

- Koncept apstrakcije je fundamentalan u programiranju i razvoju softvera.
- Apstrakcije koje podržavaju programske jezice mogu se klasifikovati u sledeće apstraktne nivoe
- Bazne apstrakcije
- Strukturne apstrakcije
- Proceduralne apstracije
- Apstracije podataka



TESTIRANJE SOFTVERA

Testiranje Softvera je skup proces, jer u proseku oko 30%, neki autori tvrde 50% ukupnog budžeta na razvoj softverskog proizvoda se troši na testiranje softvera dok je u nekim oblastima primene čak i preko 80%.

Predstavlja ponašanja programa na bazi konačnog skupa testova, odabralih na pogodan način iz beskonačnog skupa mogućih načina izvršavanja programa, a prema specificiranom očekivanom ponašanju softvera u razvijanoj aplikaciji.



TESTIRANJE SOFTVERA

Kako na tržištu rada vlada sve veća potražnja za developerima, istovremeno se javlja i nedostatak stručnih kadrova spremnih da kvalitetno testiraju softvere koji se razvijaju.

Testeri imaju veliki značaj , jer se nakon faze testiranja softver isporučuje naručiocima



TESTIRANJE SOFTVERA

Testiranje je aktivnost izvedena radi evaluacije kvaliteta proizvoda i njegovog poboljšanja, putem identifikovanja defekata i problema.

Ne treba zaboraviti i važnost hardvera. Primer mobilnih uređaja

- Svaki mobilni uređaj koji se proizvede na svetu prolazi kroz više faza testiranja pre nego što dođe do krajnjih korisnika. Bez testiranja bismo imali uređaje koji polovično rade, imaju nestabilan softver ili ne komuniciraju sa mrežom.
- Testiranje softvera se sastoji od dinamičke verifikacije ponašanja programa na konačnom skupu test slučajeva, prikladno izabranih iz obično beskonačnog domena izvršavanja, prema očekivanom ponašanju



TESTIRANJE SOFTVERA

Tržište zahteva od kompanija koje razvijaju softver sve kraće vreme razvoja softvera koje zbog toga vode bespoštenu borbu u izvršavanju poznate manre: **brže, bolje jeftinije.**

Testiranje softvera obuhvata različite vrste testiranja kako bi se osiguralo da softverski proizvod neće imati funkcionalne i nefunkcionalne nedostatke, a sve u cilju smanjenja ukupnih troškova razvoja softvera, i poboljšanju njegovog kvaliteta



ŠTA SU TESTERI?

Tester proverava da li je tim napravio dobar prozivod i da li ga tim pravi onako kako bi trebalo.

Osoba koja obavlja ovu ulogu je neko ko dosledno dovodi u pitanje sve delove procesa kako bi bio siguran da tim proizvodi željeni rezultat.

Tester poboljšava kvalitet korisničke aplikacije.

ZAŠTO SU NAM POTREBNI TESTERI ?

Softverski timovi često ne shvataju proces testiranja i složenost uloga koje postoji u procesu testiranja.

Zato što su veliki gubici kompanija koje razvijaju softver upravo zbog velikog broja defekata u isporučenom softveru.

Prvenstveni zadatak testera je otkrivanje problema u softveru sa ciljem da se oni otklone pre predaje softverskog proizvoda kupcu.



TESTIRANJE SOFTVERA

Tester pokušava da putem uobičajenih i neuobičajenih načina korišćenja softvera dođe do informacija o eventualnim greškama u softveru.

U literature se navodi:

Testeri nisu samo "čuvari kvaliteta", već su integralni deo procesa testiranja softvera.

To je timska aktivnost.

PREPORUKA:

Testeri učestvuju u pravljenju softvera od samog početka

ZAŠTO SU NAM POTREBNI TESTERI ?

Od testera se zahteva da otkrije što je moguće više problema i to što više onih vrlo ozbiljnih čije posledice mogu biti katastrofalne sa materijalnog i bezbednosnog aspekta.

Zato je sa svih aspekata potrebno da se proces testiranja softvera učini što efikasnijim i uz što manje troškove ukoliko je to moguće.

Proces testiranja softvera ima za cilj ne samo pronalaženje grešaka u postojećem softveru, već i pronalaženje mera za poboljšanje softvera u pogledu efikasnosti, tačnosti i upotrebljivosti.



Kako programerski tim vidi testera?

Developeri rade na samoj implementaciji taskova.

Testeri softvera moraju da osiguraju da se svi zahtevi naručioca neometano izvršavaju.

- ✓ Testiranje se često posmatra kao destruktivna aktivnost.
- ✓ Tako da, uloga testera nekada može biti vrlo specifična.
- ✓ U nekim slučajevima, tester se neće lako uklopiti u tim.



- ✓ Programeri ponekad misle da tester "ne saradjuje sa njima", i da bi to moglo da izazove razne probleme.

TESTER:

Stalno nalazi bagove i stalno je nezadovoljan ponuđenim rešenjima.

- ✓ Tester mora da ubedi ostatak tima da je njegova/njena uloga zaista bitna, da ima dobru komunikaciju sa njima i da na lep način dokaže da on nije protiv programskog tima.
- ✓ Tester igra u istom timu, samo sa razlicitim dužnostima.



Šta čini dobrog testera

Testiranje softvera se ne radi nasumično, već je važno poznavati metodologiju, procese i principe testiranja.

Dobar Tester mora da ima nephodno znanje :

- ✓ Kako projekat funkcioniše
- ✓ Kakva je saradnja izmedju naručioca softvera i tima koji razvija softver (kako se ti novi zahtevi implementiraju)
- ✓ Da razume tehnologije u kome je dati softver rađen
- ✓ Da razume tehnike testiranja i da izabere najbolje tehnike
- ✓ Najbolja praksa testiranja
- ✓ Da bude u mogućnosti da razmišlja unutar i izvan sistema
- ✓ Vremenski okvir za pravljenje softvera a samim time i za testiranje softvera
- .

TESTER i Komunikacija sa programerskim timom

- Što se tiče komunikacije sa programerima, dobar odnos je veoma važan.
- Takođe, testeri moraju informisati programere o greškama, putem dnevnika testiranja, kako bi se omogućile ispravke istih u najbržem mogućem roku.



TESTER i Komunikacija sa programerskim timom

Testiranje softvera i kontrola kvaliteta su obvezni segmenti razvoja softverskog proizvoda.

Progamer nikada ne moze biti dobar tester ako testira svoj kod. Tj. onaj ko je pisao kod nije objektivan tester.

Za testera nije neophodno da zna da razvija softver, ali suštinsko razumevanje kako softver radi i kako se razvija je od izuzetnog značaja

TESTIRANJE SOFTVERA





Testiranje softvera

Kada bi se iz ŽIVOTNOG ciklusa softvera nestalo Testiranje softvera, krajnji proizvod bi imao puno više nepravilnosti i grešaka što bi naravno rezultovalo velikim nezadovoljstvom samog korisnika, pa je upravo jedan od razloga što TESTIRANJE ima veliku primenu (svrhu) u procesu razvoja softvera.

Testiranje je kombinacija višestrukih aktivnosti životnog ciklusa softvera povezanih sa planiranjem, dizajniranjem i procenom softverskog proizvoda kako bi se što pre otkrile greške i utvrdilo da li softver ispunjava navedene zahteve ili ne.



Testiranje softvera

Jedna od Definicija za Testiranje softvera je:

"Testiranje je proces ocenjivanja celokupnog sistema ili njegovog dela tj. komponente sa namerom da se utvrди da li su definisani korisnički zahtevi zadovoljeni ili nisu."

- Testiranje softvera takođe uključuje implementaciju modula delova sistema za procenu svih (ili nekih) osobina kroz testiranje softvera.



Testiranje softvera

Testiranje softvera prestavlja važnu osobinu za kvalitetan softverski proizvoda koji zadovoljava potrebe, očekivanja i zahteve samog korisnika.

Testiranje treba da započne još u ranoj fazi razvoja softvera i da se obavlja veoma često.

Treba da se na neki način integriše razvoj aplikacije i životni ciklus testiranja softvera.

Treba formalizovati metodologiju testiranja- a to znači treba sve testitati na identičan način i tada se može očekivati da će se dobiti uniformisani rezultat.



Testiranje softvera

Prema istraživanjima pojedinih autora, najviše grešaka nastane u fazi implementacije, čak 50%, zatim 40% grešaka se dogodi u fazi dizajna, dok samo 10% u fazi specifikacije.

Po velikom broju autora i clanaka u naučnim časopisima najviše grešaka dogodi se u fazi implementacije.

Sugestija/Pravilo

Vrlo važno je Testirati i samo Testirati u svakoj fazi razvoja softvera.



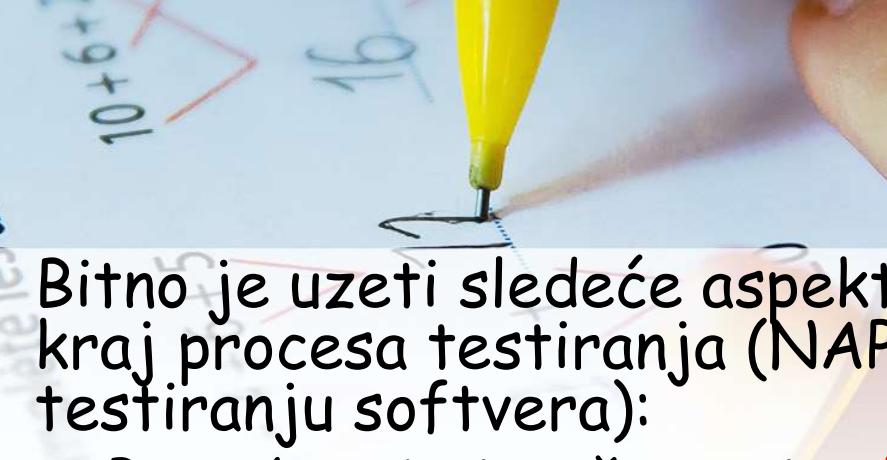
Testiranje softvera

"Testiranje softvera je mnogo više od običnog traženja grešaka (bugova)."

Kontrolu kvaliteta softvera vrše i testeri.

Tester je u velikoj meri zaslužan za zadovoljstvo klijenta i krajnjeg korisnika

Zadatak testera je da pronađe što veći broj bugova koje će zatim programmer ili programerski tim ispraviti.



Testiranje softvera

Bitno je uzeti sledeće aspekte u obzir, koje bi mogle označiti kraj procesa testiranja (NAPOMENA: Nikad nema kraju testiranju softvera):

- Potrebno je ispoštovati **rokove testiranja**,
- Završetak izvršavanja **test slučaja**
- Dovršavanje funkcionalnog i kodnog testiranja(do određene tačke/po specifikaciji)
- Stopa grešaka pada ispod određenog nivoa i nisu identifikovane greške visiokog prioriteta
- **Odluka** uprave ili menadžera projekta.
- Vremenski rokovi za implementaciju softvera su istekli



Testiranje softvera

Preduslov da bi programer mogao da preda kod na glavnu razvojnu granu je da se njegov kod uspešno bilduje.

Ako određeni zahtev može biti izvršen i dobijen je zadovoljavajući rezultat, test slučaj je u redu.

Ako su test slučajevi odabrani tako da je svakom zahtevu dodeljen bar jedan test slučaj, program se smatra testiranim i trebalo bi da radi kada svi test slučajevi pokažu zadovoljavajuće rezultate.



Testiranje softvera

Testiranje softvera se može podeliti na :

- ❖ Dinamičko
- ❖ Statičko

Podela se vrši na osnovu toga da li se u toku testiranja kod izvršava ili ne.

Statičko testiranje je testiranje bez izvršavanja koda.
Ono se obično vrši kroz razne vrste pregleda i revizija.

Predmet tih pregleda i revizija može biti dokumentacija ili sam kod.



STATIČKO testiranje softvera

Statičko testiranje se svodi na testiranje softvera bez njegovog izvršavanja i pokretanja, bilo ručno bilo putem alata. Moguće je da se izvršava vrlo rano u samom razvoju softvera.

Statičko testiranje nije alternativa dinamičkom testiranju. Ako koristite obe vrste testiranja možete pronaći različite greške.

Statičko testiranje je tehnika koja nam omogućava da prepoznamo nedostatke u softveru, a da ovaj softver zapravo ne pokrećemo.

Statičkim testiranjem možemo pronaći greške koje ne možemo pronaći kod dinamičkog testiranja.

U literature se statičko testiranje može pronaći pod nazivom statična analiza.



Statičko testiranje

Statičko testiranje je skup tehnika kojim se poboljšava kvaliteta softvera, kao i efikasnost i produktivnost samog procesa razvoja softvera.

Ovaj tip testiranja se bazira na procesu statičkog pregleda (review), sa ciljem da se defekti pronađu što ranije u procesu razvoja softvera.

CILJ: Svako testiranje tako i statičkog je detektovanje što većeg broja grešaka.



Statičko testiranje

- je testiranje softvera bez njegovog pokretanja ili izvršavanja. Statičko testiranje se može vršiti na kodu, dizajnu, modelima, funkcionalnim zahtevima, specifikaciji zahteva i arhitekturi samog softvera

Tipovi statičkog testiranja su:

- ✓ Neformalni pregled
- ✓ Formalni pregled
- ✓ Walkthrough
- ✓ Tehnički pregled
- ✓ Inspekcija



Neformalni pregled

Pregled sam po sebi može biti formalan ili neformalan.

Pregled je generalni termin koji se koristi za sve tehnike manuelnog statičkog testiranja

- u praksi se najčešće koristi *review* ili *inspection*
- u pregledu najčešće učestvuje više ljudi
 - *peer review*

Ako se govori o neformalnom pregledu tada se najčešće radi o prezentaciji dokumenata projektnom timu (specifikacije zahteva, detaljni dizajn, tehnička /e specifikacije).

- Svrha pregleda je davanje informacija svim članovima tima o napretku projekta.



Neformalni i formalni pregled

Neke od faza tokom formalnog pregleda su planiranje, kick-off, priprema, formalni sastanak, korekcije i follow-up. Za razliku od neformalnog pregleda, formalni pregledi se obavezno moraju dokumentovati u obliku zapisnika.

- **Prednosti pregleda**
- Poboljšava se dalji proces razvoja softvera,
- Ideja je da ako je to moguće (a jeste) da se ti nedostaci otkriju, što pre
 - ✓ u razvoj rešenja uključeno više ljudi a samim time je kvalitetnije rešenje
 - učesnici uče jedni od drugih
 - ✓ formalizacija i dokumentovanje rešenja
 - ✓ autor mora da iznese rešenje tako da drugim učesnicima bude čitljivo i razumljivo, kao i da obrazloži odluke donete pri razvoju
 - ✓ odgovornost se raspoređuje na sve učesnike
- Iako troši resurse, pregled je neizostavan u razvoju jer značajno smanjuje broj kasnijih nedostataka i troškove otklanjanja



Neformalni i formalni pregled

- **Na generalnom nivou**

- na nivou projekta se mora doneti odluka koji dokumenti se pregledaju
- mora se u plan projekta uvrstiti i dinamika i troškovi tih pregleda

- **Na nivou jednog pregleda**

- pripremiti dokumente koji su predmet pregleda
- izabrati učesnike u pregledu
- obezbediti da je dokument spremán za pregled
 - da je završen na nivou predviđenom za tu fazu projekta
- odrediti mesto i vreme sastanka i druge detalje potrebne za održavanje pregleda



TEHNIČKI pregled

Tehnički pregled je fokusiran na tehničke aspekte sistema

- usklađenost sa specifikacijom
- ispunjenost svrhe zbog koje se razvija
- usklađenost sa standardima
- Osnov za pregled je isključivo zvanična specifikacija zahteva
- Recenzenti poseduju tehničko znanje
- Procedura može biti više ili manje formalno sprovedena
 - kod formalnog tehničkog pregleda, svi rezultati sve formalno evidentiraju



Inspekcija softvera

- Uključuje ljude koji proveravaju source code aplikacije sa ciljem utvrđivanja anomalija i defekata.
- ✓ Ne zahteva izvršenje aplikacije, pa se može koristiti i pre implementacije.
- ✓ Može se primeniti na bilo koji element sistema - zahteve, dizajn konfiguracione podatke,.....
- ✓ Pokazalo se da su vrlo efikasna tehnika za utvrđivanje programskih grešaka



Prednosti inspekcija

- U toku testiranja, greške mogu maskirati (sakriti) druge greške. Zbog toga što je inspekcija staticki proces, druge greške koje se pojavе ne mogu uticati ili sakriti greške.
- Nekompletne verzije sistema se mogu proveravati bez dodatnih troškova. Ako je aplikacija nekompletna potrebno je razviti specijalizovan test koji će testirati dostupne delove
- Osim pronalaženja programskih defekata, inspekcija može posmatrati i šire atribute kvaliteta aplikacije, kao što su poštovanje standarda, portabilnost i mogućnost održavanja.

PROVERA SOFTVERA (SOFTWARE INSPECTIONS)

Odnosi se na „statičku“ proveru, pre implementacije kompletног softvera, za razliku od testiranja koje podrazumeva „dinamičku“ proveru prilikom izvršavanja.

Uključuje ljudе koji mogu da pregledaju izvorni kod kako bi našli neke anomalije u softveru, greške i delove koda koji bi mogli izazvati neželjeno ponašanje prilikom samog izvršavanja softvera.



PREDNOSTI inspekcija

Osim pronalaženja programskih defekata, inspekcija može posmatrati i šire atribute kvaliteta aplikacije, kao

što su :

- ✓ poštovanje standarda,
- ✓ portabilnost i
- ✓ mogućnost održavanja.

PROVERA SOFTVERA (SOFTWARE INSPECTIONS)

- Može biti primenjeno na bilo koji segment sistema (na primer na zahteve, na dizajn softvera)
- Koristi se znanje o čitavom sistemu, UML modeli, šeme baze podataka, domen aplikacije.
- Ne može zameniti testiranje softvera.

PROVERA SOFTVERA (SOFTWARE INSPECTIONS)

Tokom testiranja softvera greške koje se pronalaze mogu da "sakriju" druge (velike) greške (bugove). Kako je inspekcija staticki proces, često se ističe da možda nema razloga za brigu o „vezanim“ greškama.

(UVEK SE MOGU NAĆI bugovi i nekad jedan bug kad se ispavi prouzrokuje drugi bug, to su takozvani vezani bugovi).

Može se primeniti na takozvane nepotpue ili nedovrše sisteme, pre konačne implementacije softvera.

PROVERA SOFTVERA (SOFTWARE INSPECTIONS)

Pored pronalaženja grešaka, tokom inspekcije mogu biti analizirane i druge važne karakteristike sistema, kao što je na primer saglasnost(cookies), prenosivosti, održivost samog softvera.

Inspekcija ne može da proveri nefunkcionalne karakteristike sistema kao što su na primer performance (nekog sistema, uređaja), ..



Faze pregleda

- Postoje različiti tipovi pregleda, obično postoje sledeće faze (implicitno ili eksplicitno izražene)
 - ✓ Planiranje
 - ✓ Informisanje
 - ✓ Samostalna priprema
 - ✓ Sastanak
 - ✓ Ispravke
 - ✓ Revizija



Walkthrough

Walkthrough spada u neformalni proces. Autor/i sa svim učesnicima sastanka prolazi kroz ceo dokument. Ideja je da se da se prikupe dodatne povratne informacije.

Cilj walkthrough procesa je:

- ✓ Prezentacija dokumenta da bi se dobile povratne informacije o dokumentaciji
- ✓ Prenošenje znanja
- ✓ Evaluacija sadržaja dokumenta
- ✓ Diskusija o predloženim rešenjima



Revizija

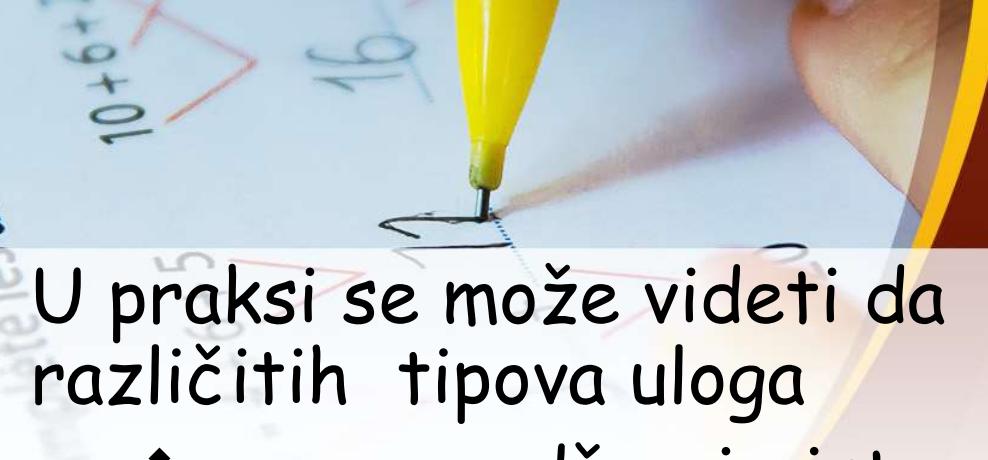
- Nakon ispravki neophodno je izvršiti reviziju nove verzije dokumenta.
- Mora se obaviti ŠTO PRE novi pregled
 - procedura drugog pregleda može biti nešto kraća sa fokusom samo na modifikovane delove (koji su promenjeni)
- Potrebno je evaluirati sam proces pregleda kako bi naredni pregled ispravio nedostatke u proceduri
 - Treba da se identifuju novi faktori koji mogu dovesti do toga da neki naredni pregled bude efikan.



Uloge učesnika

Mogu se identifikovati nekoliko klasa učesnika pregleda:

- Menadžer
- Moderator
- Autor
- Recenzenti
- Beležnik
 - zadužen da evidentira informacije iz pregleda (probleme, stavove, odluke, zaključke)



Uloge učesnika

U praksi se može videti da isti ljudi preuzimaju više različitih tipova uloga

- ❖ npr. menadžer je istovremeno i moderator, ali i recenzent koji aktivno učestvuje u doноšењу odluka u pregledu
- ❖ takođe, za vođenje beležaka može biti zadužen neko od članova tima
 - Veoma je bitno da svi imaju uvid u zaključke pregleda



Tipovi pregleda

Postoje dve grupe pregleda i to:

- pregledi koji su namenjeni analizi karakteristika samog projekta
 - da li su zahtevi (funkcionalni i nefunkcionalni) ispunjeni
- pregledi koji su namenjeni analizi samog procesa razvoja projekta
 - Na primer analiziraju se kvalitet upravljanja projektom, ispunjenost plana projekta i drugo.



Izbor tipa pregleda

To zavisi od konkretnog slučaja

- Preporuke i ideje:
 - ❖ Formalno dokumentovani rezultati pregleda trebaju biti
 - ❖ Koliko je komplikovano organizovati grupni lični sastanak sa učesnicima
 - ❖ Koliko tehničkog znanja je potrebno da učesnici poseduju
 - ❖ Kakav je odnos resursa potrebnih za pripremu pregleda i koristi od rezultata pregleda
 - ❖ Da li je predmet pregleda formalno specificiran da bi mogao biti analiziran programski
 - ❖ Koliko je menadžment projekta obezbedio resursa za preglede



STATIČKO testiranje softvera

Vrste grešaka koje je lakše pronaći tokom statičkog testa:

- ✓ Odstupanja od standarda
- ✓ Loš kod
- ✓ Greške u dizajnu
- ✓ Problemi sa specifikacijom

IDEJA: Koristiti statičko testiranje da biste rano pronašli greške (bagove).



STATIČKO testiranje softvera

Statičko testiranje obuhvata testiranje softverskog zahteva. Planirane aplikativne funkcionalnosti se analiziraju na primer da li postoje neke nejasnoće, neka dvosmislenosti, kao i da li se uopšte može testirati softver.

Svaki zahtevani detalj koji može dovesti do pogrešnog tumačenja mora biti nedvosmeleno razjašnjen.

Statičko testiranje podrazumeva verifikaciju test objekta analizom njegovog sadržaja.



Testiranje softvera

Testiranje softvera može da se izvrši na različitim nivoima.

U zavisnosti od samog sistema, kao i od raspoloživih resursa, bitno je odabrati odgovarajuću metodologiju testiranja softvera.

-Metodologiju testiranja softvera -

Softverski testeri igraju ključnu ulogu u IT industriji jer je njihov zadatak da utvrde da li softverski proizvod odgovara unapred postavljenim zahtevima.



Testiranje softvera

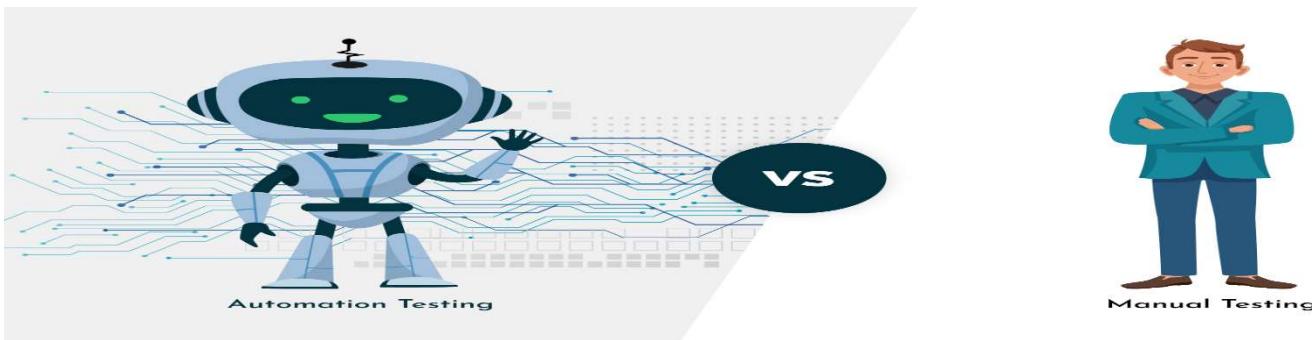
Tester je član tima koji mora da se postavi u ulogu samog klijenta ili ciljne grupe korisnika, ali isto tako da se postavi u ulogu programera koji kreira neku softversku aplikaciju.

Testiranje u bilo kojoj od faza razvojnog životnog ciklusa softvera se obavlja na specifičan način, na primer u fazi prikupljanja se razmatraju analize i verifikacija zahteva testiranja.



Testiranje softvera

Testiranje može da se radi manuelno ili automatski uz pomoć alata i tehnologija za pisanje automatskih testova. Zbog toga postoji podela na automatsko i manuelno testiranje. manuelno testiranje obavlja čovek, dok automatsko izvršava program.





Testiranje softvera

Da bi garantovao kompletnost testiranja, tester često sledi pisani plan ispitivanja koji ih vodi kroz niz važnih test slučajeva.

Svrha manuelnog testiranja je da pokrije što više scenarija koje **nije moguće automatizovati** i na taj način da se dobije na kvalitetu samog softvera.

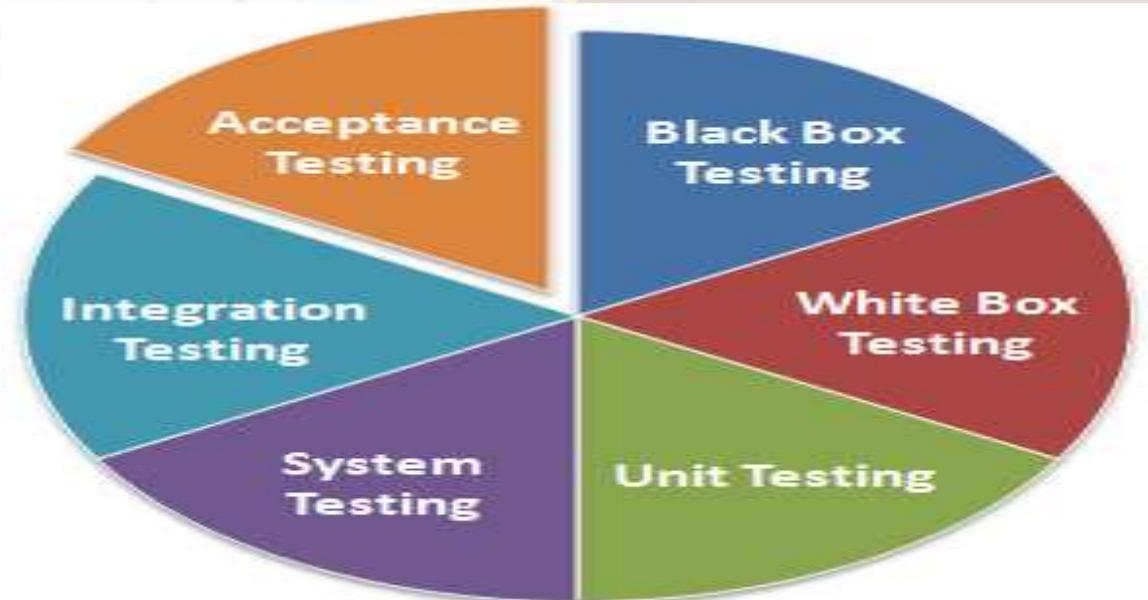
Ručno (manuelno) testiranje softvera

Glavni koncept ručnog testiranja je da aplikacija radi bez grešaka.

Ovaj način testiranja softvera predstavlja osnovnu tehniku svih tipova testiranja i pomaže u pronalaženju kritičnih grešaka u softverskoj aplikaciji.

Svrha manuelnog testiranja jeste da se osigura da aplikacija nema grešaka i da radi u skladu sa specificiranim funkcionalnim zahtevima

Testiranje softvera





Ručno (manuelno) testiranje softvera

Manuelno testiranje je takav način testiranja gde se ne koristi ni jedan automatizovan alat ili bilo koja skripta. U ovom načinu testiranja ispitivač preuzima ulogu krajnjeg korisnika i testira softver sve dok ne identificuje neko neočekivano ponašanje ili grešku. Prilikom testiranja, obično su na raspolaganju ograničeni resursi, čime se onemogućava efikasno izvršavanje manuelnih testova

Postoje različite faze manuelnog testiranja :

- Integraciono testiranje i
- Prihvatljivo testiranje

Postoje i različite tehnike analize

- razlikuju se po intenzitetu, formalnosti, potrebnim resursima i ciljevima
- ne postoji jedinstvena terminologija za ove različite tehnike analize

Testeri koriste plan testiranja, testove ili scenarija za testiranje softvera i time oni osiguravanju da proces testiranja bude kompletan.

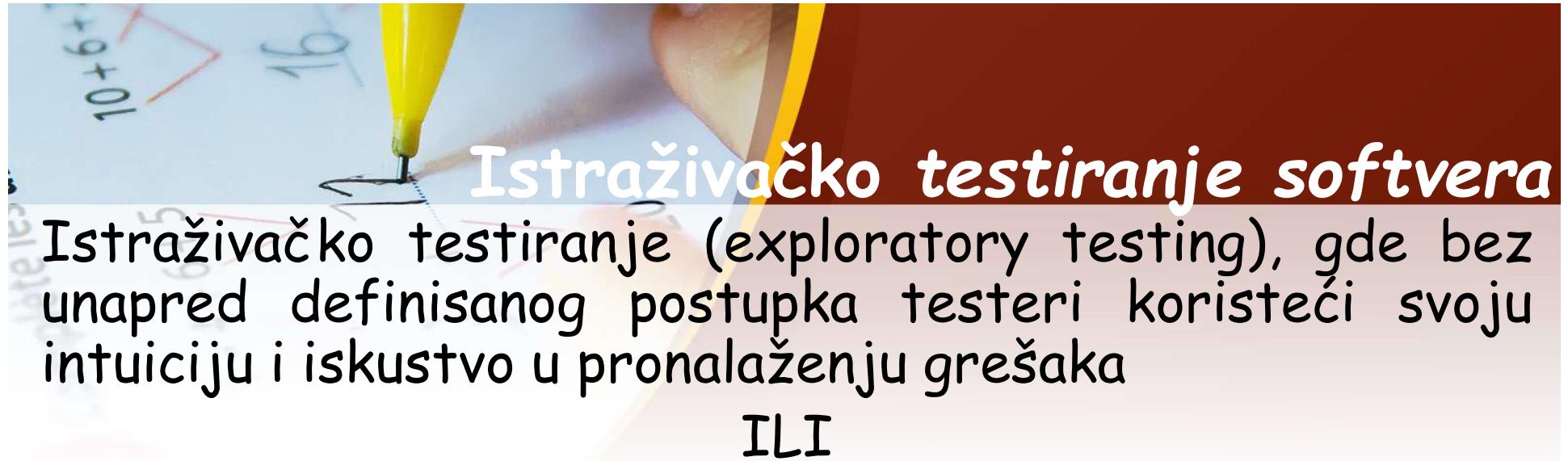


Ručno (manuelno) testiranje softvera

Manuelno testiranje takođe uključuje i istraživačko testiranje za cilj da treba istražiti softver kako bise identifikovala greška u njemu.

Manuelno testiranje zahteva angažovanje većeg broja ljudi unutar test timova za izvršenje testa.

Za manuelan način testiranja nije potrebno poznavanje nijednog alata za testiranje



Istraživačko testiranje softvera

Istraživačko testiranje (exploratory testing), gde bez unapred definisanog postupka testeri koristeći svoju intuiciju i iskustvo u pronalaženju grešaka

ILI

Istraživačko testiranje je vežbanje testiranja, u kojem se testerima dodeljuje neprecizno definisan zadatak, uz korišćenje softvera koji se testira.

Istraživačko testiranje nije nasumično, ali nije ni skriptovano kao manuelno testiranje.



Istraživačko testiranje softvera

Istraživačko testiranje je metoda testiranja u kojoj kao i u skriptovanim testovima postoje test procedure ali ne moraju striktno da se prate.

Tokom ove metode testiranja testeri otkrivaju i proveravaju alternativne pravce korišćenja sistema, što znači da se u istraživačkom testiranju objedinjene aktivnosti identifikacije, dizajna i izvršavanja test slučajeva.



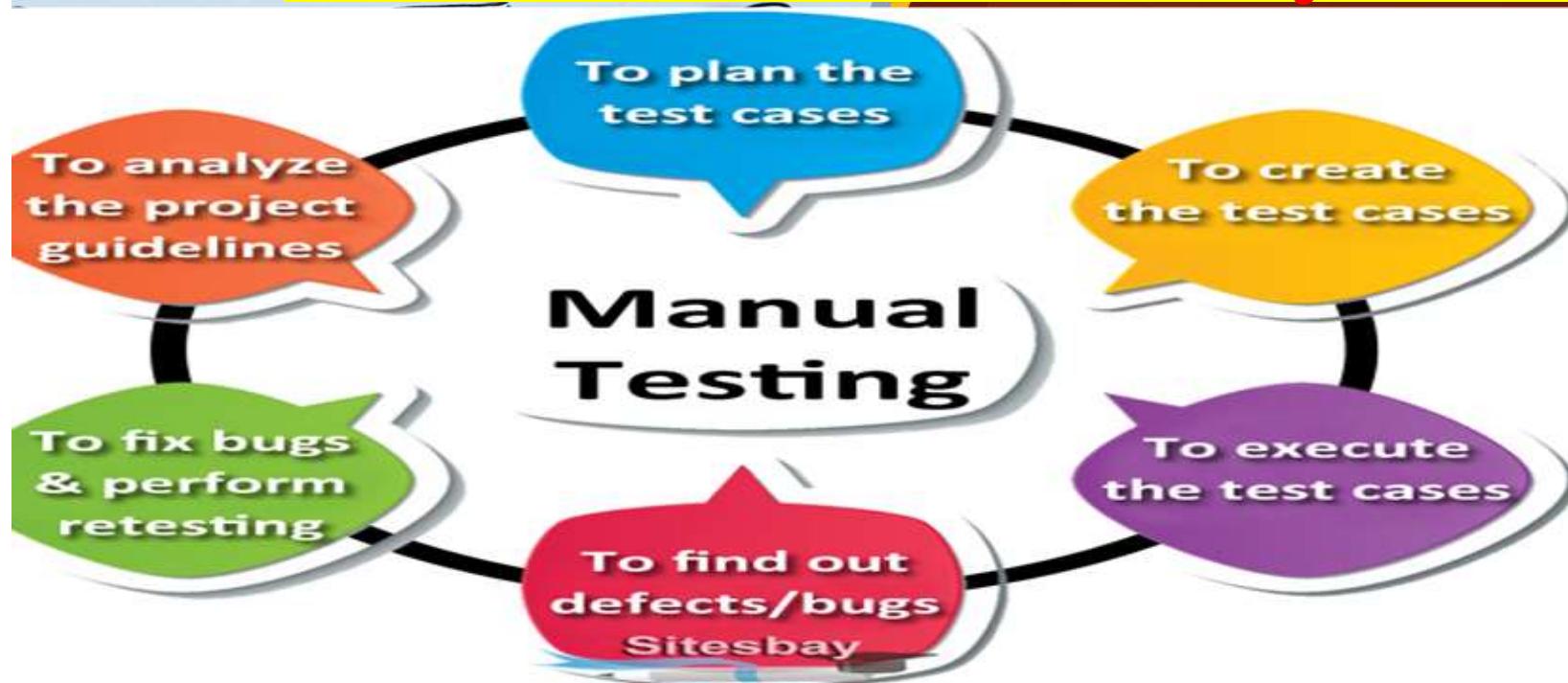
Istraživačko testiranje softvera

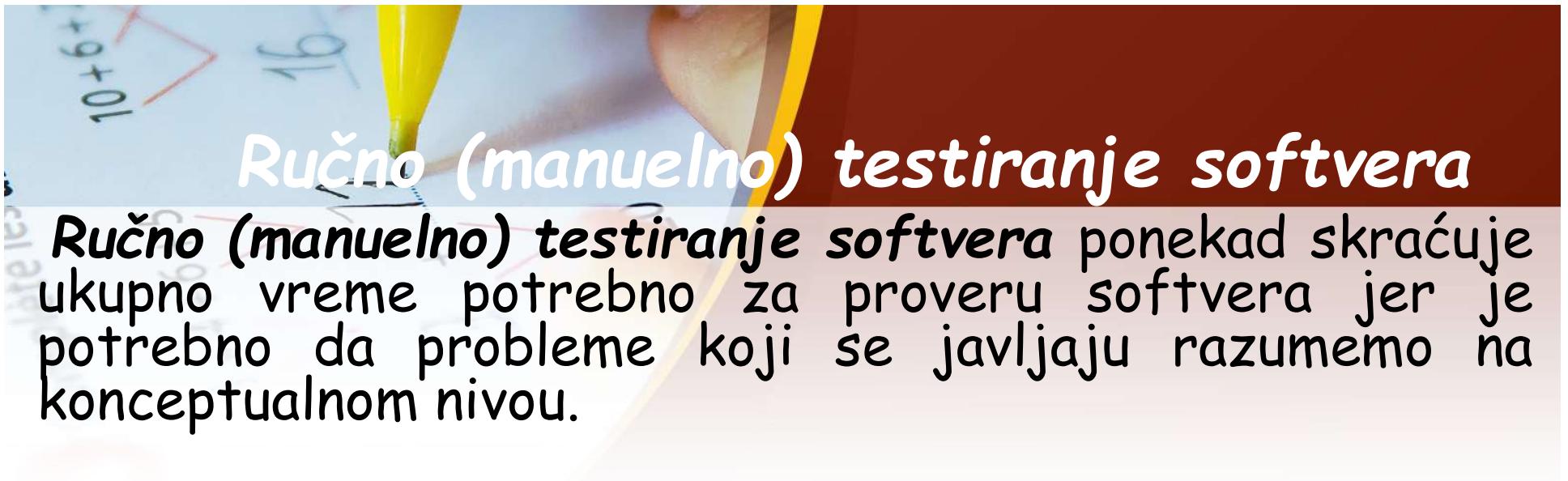
Jedna od mana ovog testiranja je da je ovo testiranje neregulisano i da nema realnog plana da se odredi koje delove softvera treba da se testira.

Neki od problema koji se javljaju jeste da se neke greške ili defekti ne mogu reprodukovati.

Ovo je neformalni proces testiranja.

Ručno (manuelno) testiranje softvera





Ručno (manuelno) testiranje softvera

Ručno (manuelno) testiranje softvera ponekad skraćuje ukupno vreme potrebno za proveru softvera jer je potrebno da probleme koji se javljaju razumemo na konceptualnom nivou.

Postoje radovi gde neki autori tvrde da se u velikoj kompaniji poput Microsoft pronađe 10-20 % grešaka u njihovim projektima (a potom uspešno otkloni) na ovakovom vidu testiranja softvera.



Ručno (manuelno) testiranje softvera

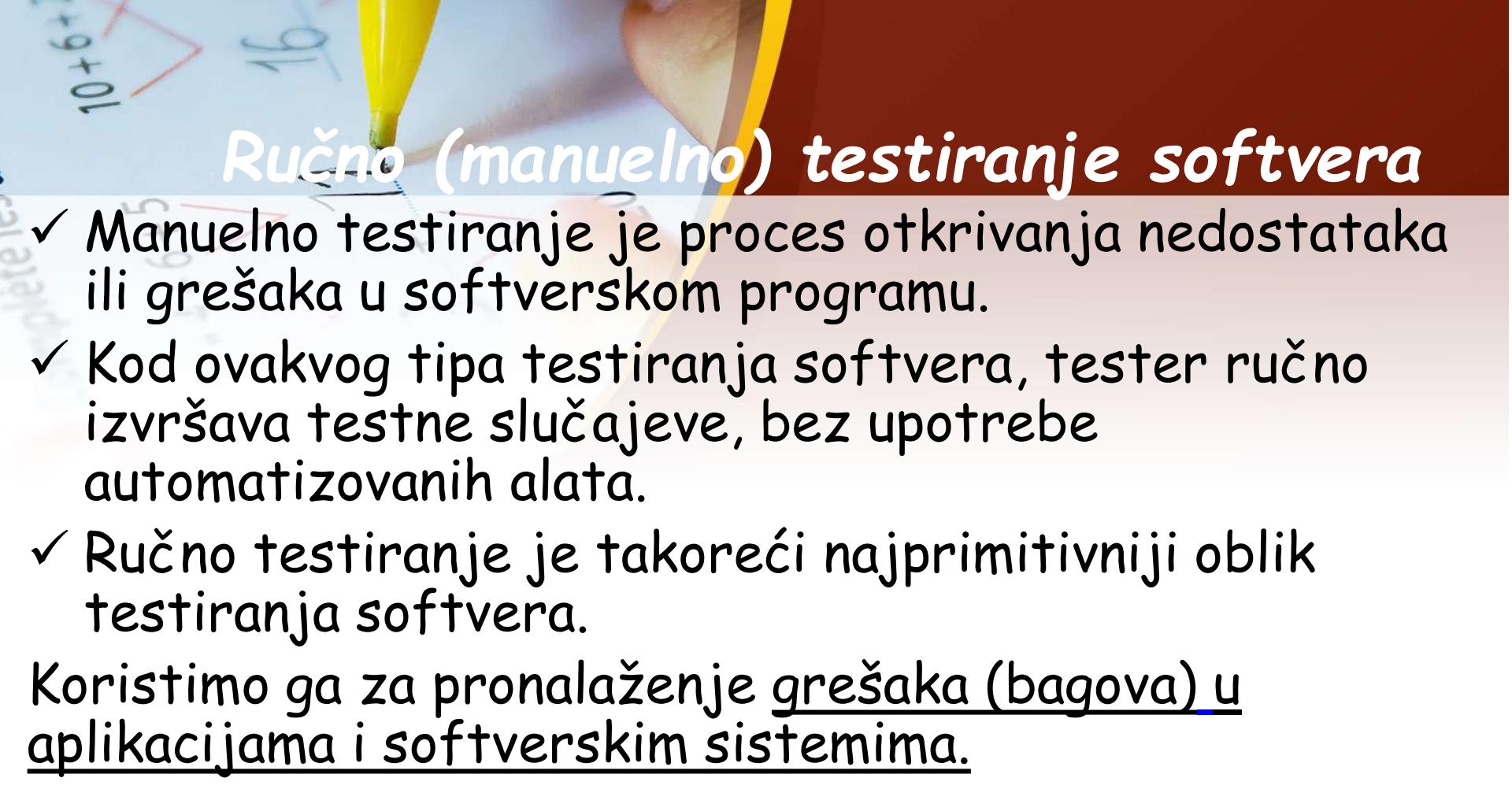
Oni mogu da budu neformalni, gde tester proverava da li je implementirana funkcionalnost u skladu sa dokumentacijom, ili formalni, gde se test sprovodi po unapred definisanom setu koraka, koji simuliraju ponašanje korisnika.



Ručno (manuelno) testiranje softvera

Manuelno testiranje ne zahteva poznavanje programskih jezika i predstavlja samostalnu celinu.

Manuelno tesiranje takođe može biti i uvod u automatsko testiranje, jer se najčešće manuelni testeri odlučuju da kasnije uče i automatsko testiranje, koje zahteva osnovno poznavanje bar jednog programskog jezika.



Ručno (manuelno) testiranje softvera

- ✓ Manuelno testiranje je proces otkrivanja nedostataka ili grešaka u softverskom programu.
- ✓ Kod ovakvog tipa testiranja softvera, tester ručno izvršava testne slučajeve, bez upotrebe automatizovanih alata.
- ✓ Ručno testiranje je tako reći najprimitivniji oblik testiranja softvera.

Koristimo ga za pronalaženje grešaka (bagova) u aplikacijama i softverskim sistemima.

Ručno (manuelno) testiranje softvera

Odnosi se na analiziranje dokumenata u projektu od strane učesnika

- dokument je najčešće sam kod, ali može biti i specifikacija zahteva, model sistema, model podataka, plan aktivnosti, ...

Različite tehnike analize

- razlikuju se po intenzitetu, formalnosti, potrebnim resursima i ciljevima
- ne postoji jedinstvena terminologija za ove različite tehnike analize



Ručno (manuelno) testiranje softvera

- ✓ Manu ručno (manuelno) testiranje softvera je dugotrajnost celog procesa i mogućnost ljudske greške.
- ✓ Ljudski faktor u ovom načinu testiranja često je ključan kod što bolje evaluacije.
- ✓ U ovakvom vidu testiranja softvera je veoma bitno iskustvo kao jedan od bitnih faktora za sam kvalitet softvera.



Ručno (manuelno) testiranje softvera

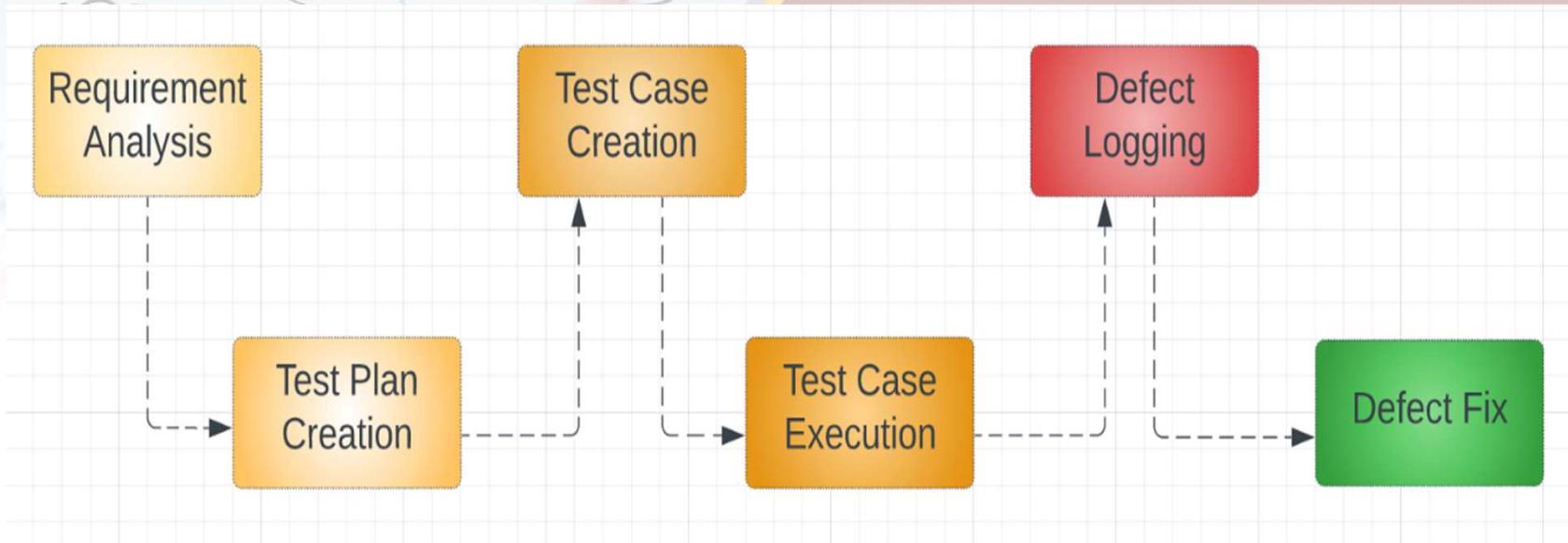
✓ Manuručno (manuelno) testiranje softvera je:

Puno zavisi od samog testera (njegovog iskustva i njegovog poznavanja programskog jezika, koda i samog domena problema)

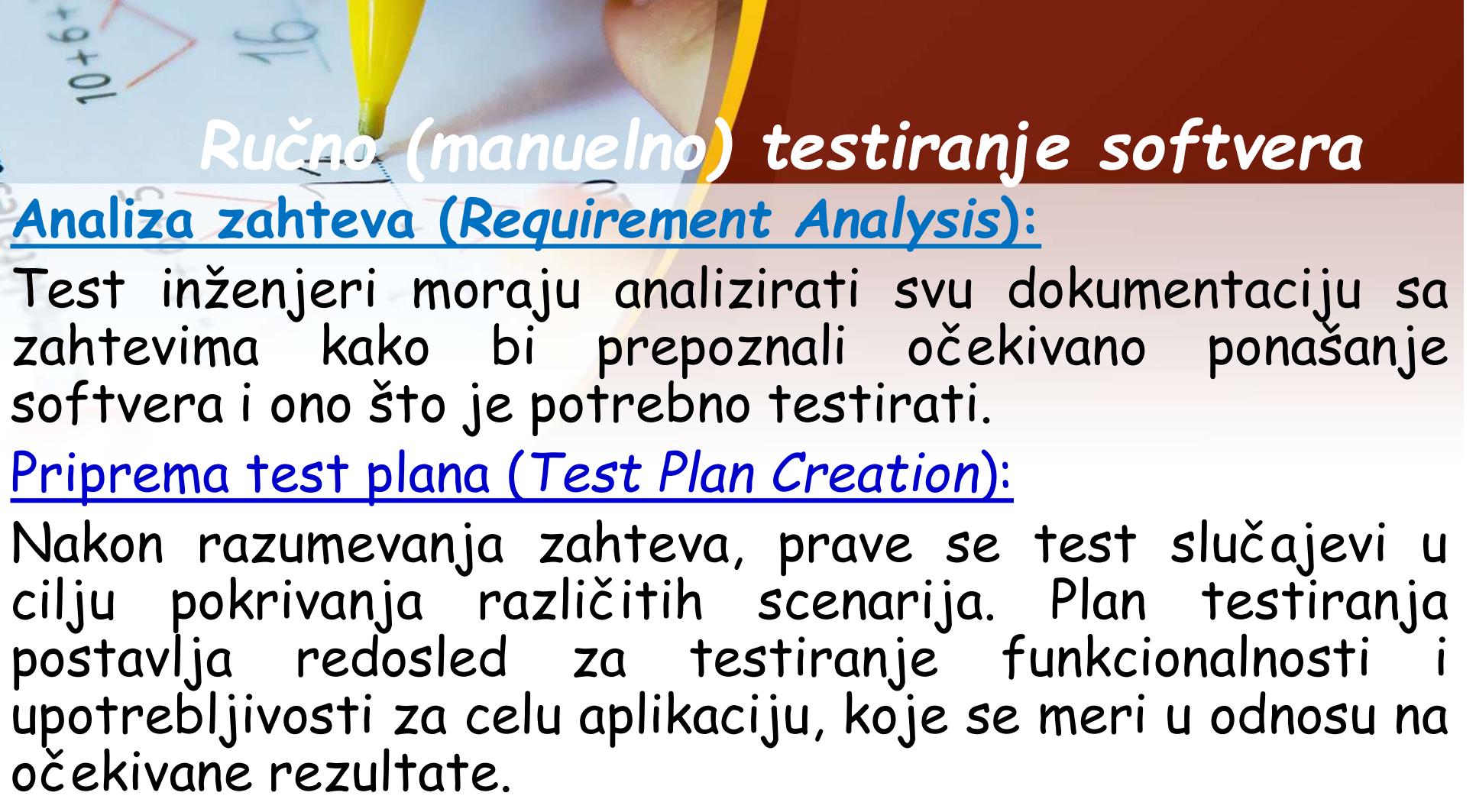
Ako se u međuvremu promeni deo koda, neophodno je ponovo testiranje (dugotrajnost celog procesa)

Slabi kriterijumi pokrivenosti

Ručno (manuelno) testiranje softvera



Šest osnovnih faza za izvođenje manuelnog testiranja



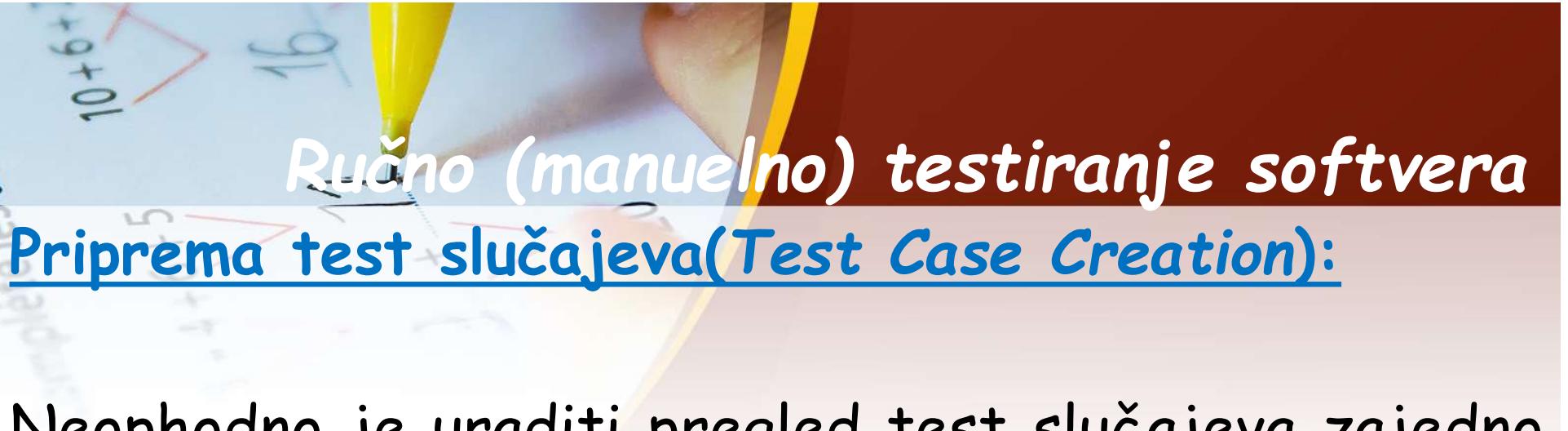
Ručno (manuelno) testiranje softvera

Analiza zahteva (Requirement Analysis):

Test inženjeri moraju analizirati svu dokumentaciju sa zahtevima kako bi prepoznali očekivano ponašanje softvera i ono što je potrebno testirati.

Priprema test plana (Test Plan Creation):

Nakon razumevanja zahteva, prave se test slučajevi u cilju pokrivanja različitih scenarija. Plan testiranja postavlja redosled za testiranje funkcionalnosti i upotrebljivosti za celu aplikaciju, koje se meri u odnosu na očekivane rezultate.



Ručno (manuelno) testiranje softvera

Priprema test slučajeva (Test Case Creation):

Neophodno je uraditi pregled test slučajeva zajedno sa celim timom, pa i klijentom, kako bi se utvrdilo i proverilo da li je sistem pokriven testovima, kao i da li je potrebno napraviti bilo kakve izmene pre početka izvršenja testova.



Ručno (manuelno) testiranje softvera

Izvršavanje test slučajeva (Test Case Execution):

Manuelno testiranje se sada može izvršiti, koristeći bilo koju od gore navedenih tehnika.

Osim pronalaženja grešaka, cilj je identifikovati potencijalne greške do kojih bi mogli da dođu korisnici, kao i „rupe u kodu“ koje bi hakeri mogli iskoristiti.

Test inženjeri izvršavaju test slučajeve jedan po jedan, ponekad koristeći alate za praćenje grešaka kao što je Jira alat.



Ručno (manuelno) testiranje softvera

Prijavljivanje grešaka (Defect Logging):

Kada se identifikuju greške, tim za testiranje prosleđuje rezultate testiranja razvojnom timu u obliku izveštaja o testiranju.

Ovo sadrži detalje o tome koliko je nedostataka ili grešaka pronađeno, koliko test slučajeva nije uspelo i koje je potrebno ponovo pokrenuti.



Ručno (manuelno) testiranje softvera

Ispravljanje grešaka (Defect Fix):

Kada se okrije greška u kodu, razvojni tim dobija zadatak da ispravi greške.

Kada razvojni tim popravi greške, softver se ponovo vraća test inženjerima.

Ponovo se pokreće isti test slučaj koji je prvobitno rezultirao grešku, kako bi proverili je li problem rešen.



Ručno (manuelno) testiranje softvera

DA SUBLIMIRAMO!

- Procedura koja se prati prilikom manuelnog testiranja:
- Kreiranje plana testa → treba da se Dokumentuje plan testiranja i nove funkcije.
- Stvaranje predmeta testa → Uključuju sve slučajeve vezane za testiranje zadate funkcije.
- Razne kombinacije test scenarija.
- Ispitivanje slučaja → Izvršiti sva testiranja i sve moguće scenarije.
- Logovanje defekta → Kada se nađe greška, treba je dokumentovati u alatu za praćenje i sve zapisati u dnevnik testiranja
- Popravljanje defekata i ponovna verifikacija → Programerski tim ispravlja grešku, testerski tim ponovo testira.
- TESTIRATI I SAMO TESTIRATI

TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA

Pojam softver-a prvi put se pominje u radovima John W. Tukey, 1957. godine.

Testiranje softvera se danas vidi kao aktivnost koja obuhvata ceo proces razvoja i održavanja i predstavlja važan deo kompletne konstrukcije softvera.

Kompletno testiranje softvera je nemoguće zbog ograničenja u resursima, budžetu, vremenu i zahtevima kvaliteta softvera.

Uvek postoji neki skriveni "bug"

Podrazumeva detaljna testiranja svih delova programa.

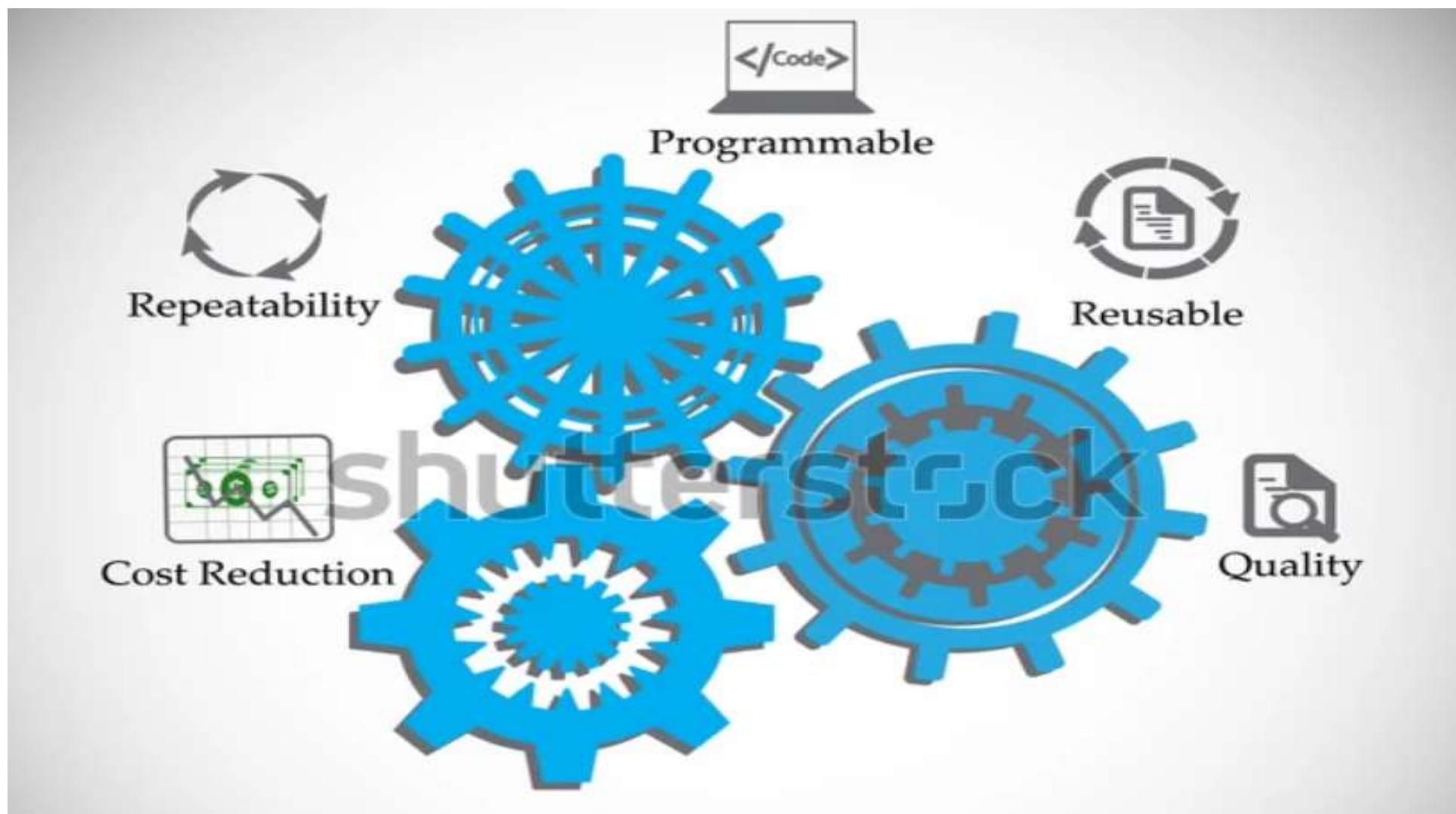
TESTIRATI[<] TESTIRATI I SAMO TESTIRATI SVE DELOVE SOFTVERA



Automatsko testiranje softvera

Bitna pitanja kod automatskog testiranja softvera su:

- ❖ Šta automatizovati?
- ❖ Kada automatizovati?
- ❖ Kako automatizovati?





Automatsko testiranje softvera

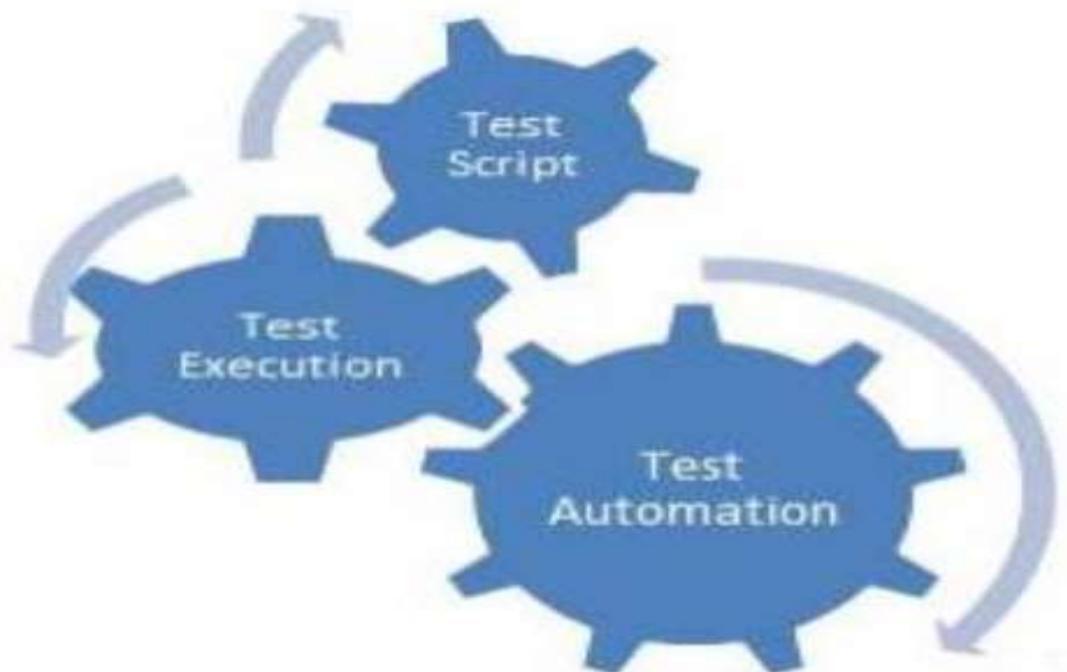
ISO/IEC/IEEE 29119-1

Neke od tema u ovom standardu su uvod u testiranje softvera, testni procesi i potprocesi, prakse u testiranju, **automatsko testiranje**.

Mark Fewster i Dorothy Graham navode kako je automatsko testiranja skuplje od manuelnog (ručnog) testiranja. Kao najvažniji razlog su naveli potrebu za pažljivim izborom testova koji se mogu automatizovati.

Automatsko testiranje je proces kontrole i brze provere velikog skupa funkcionalnosti sistema te nam daje izveštaje o razlici izlaznih i ulaznih parametara.

Automatsko testiranje softvera





Automatsko testiranje softvera

Neki od pristupa automatskog testiranja su:

- **Testiranje API-ja** (application programming interface) - sa ulaznim podacima (gde prilikom testiranja softvera vidimo da li dobijamo na neki način očekivane izlazne podatke.)
- **Testiranje GUI-ja** (graphical user interface) - testiramo događaje na grafičkom korisničkom interfacu.
- **Testiranje web aplikacija** - testiramo web aplikacije.

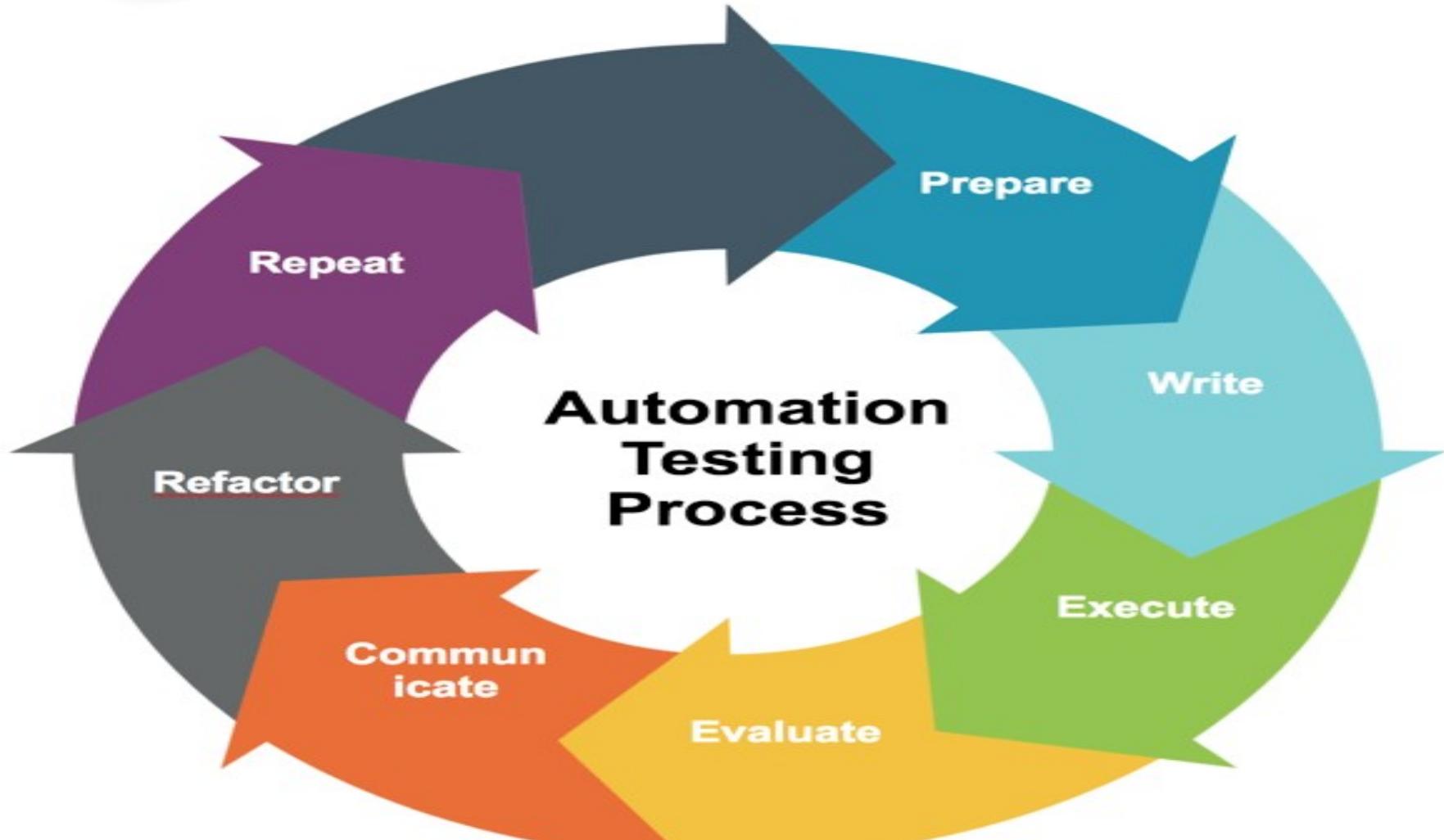
Na primer. Selenium je alat za testiranje web aplikacija, jedan je najzastupljenijih i dostaje jednostavan za korišćenje.

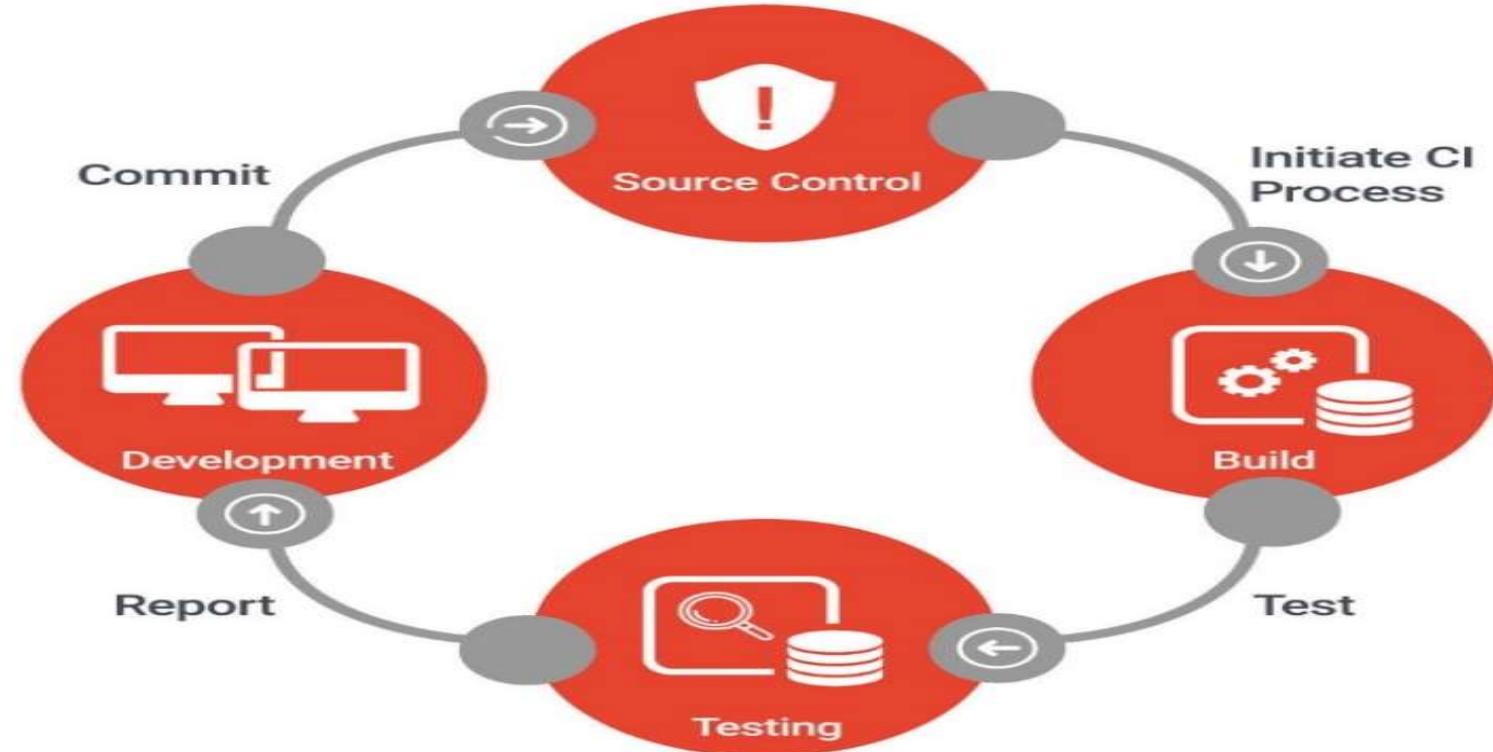


Automatsko testiranje softvera

Kako bi se odredilo koji testovi se mogu automatizovati potrebno je proveriti testove odnosno koji su od napisanih testova najbolji odnosno najpodložniji automatizaciji.

Danas automatsko testiranje softvera je sve češća pojava te skoro sve organizacije imaju bar deo funkcionalnosti pokriven sa automatskim testovima







Automatsko testiranje softvera

Automatsko testiranje softvera je posebno izraženo u radovima (Fernandes, Di Fonzo, 2017) kod takozvanog regresijskog testiranja (jednom pripremljeni regresijski testovi mogu se izvoditi svaki put kad je to potrebno)

Zatim u testiranju osnovnih funkcionalnosti (Smoke Testing) koje omogućuje kvalitetnu brzu procenu softvera na osnovu koje se zatim donosi odluka o detaljnijem testiranju; staticnih testova (mogućnost češćeg izvođenja većeg broja testova, što donosi pouzdanosti sistema);



Automatsko testiranje softvera

Nekoliko razloga da koristimo automatsko testiranje softvera:

- Ako postoje automatski testovi oni daju mogućnost da se češće pokrenu ti testovi.
- Automatskim testiranjem softvera možemo izvršavati testove koje je teško ili nemoguće realizovati ručno.

Šta ako u sistemu imamo nekoliko hiljada korisnika.
Kako da manuelno simuliramo tu zadatu situaciju ?



Automatsko testiranje softvera

Nekoliko razloga da koristimo automatsko testiranje softvera:

- Automatskim testiranjem softvera omogućuje se bolja raspodela resursa
- Testove koji su automatizovani moguće je pokrenuti više puta. (videte da li postoji razlika u ulaznim i izlaznim parametrima)
- Automatskim testiranje softvera / smanjuje se vreme potrebno za testiranje



Automatsko testiranje softvera

Nekoliko razloga da koristimo automatsko testiranje softvera:

- Ukoliko je dobro definisan skup automatskih testova, povećava se sigurnost da u proizvodu neće biti "neugodnih iznenađenja nakon isporuke."
- Može da se pokrije i više test scenarija, čak i onih manje verovatnih koje mogu da se dese.
- Moguće je ponovno korištenje automatskih testova.



Automatsko testiranje softvera

Jedina strategija koja nam garantuje dugoročno održivi tempo razvoja je nezavisnost od manuelnih testera i istovremeni razvoj automatskih testova uporedo sa razvojem novih funkcionalnosti.

Kratkoročni cilj:

Brzina razvoja će se očigledno smanjiti jer osim aplikacijskog moramo napisati i testni kod, ali će zato **kvalitet koda biti dugoročno obezbeđen.**



Automatsko testiranje softvera

Test "automatizacije" je proces kada testerski tim piše skripte i koristi drugi softver za testiranje softvera.

Automatsko testiranje se koristi za ponovno pokretanje test scenarija koji se izvodi ručno, brzo i više puta.

Automatsko testiranje se koristi i za testiranje aplikacije sa stanovišta opterećenja, performansi i stresa.

Povećava se pokrivenost testom, poboljšava tačnost, štedi vreme i budžet u poređenju sa manuelnim testiranjem.



Automatsko testiranje softvera

Automatsko testiranje predstavlja tehniku testiranja softvera kojom se upoređuje stvarni ishod (rezultat) sa očekivanim ishodom (rezultatom) pisanjem automatskih testova.

Automatsko testiranje softversa se koristi sa test skriptima ili korišćenjem nekog alata za automatizaciju testova.

Ovaj način testiranja se koristi kako bi se automatizovali testovi koji se često ponavljaju pa i oni testovi koje je teško izvršiti ručno.



Automatsko testiranje softvera

- **Procedura koja se prati prilikom automatskog testiranja:**
 - Donesite odluku
 - Pribavite alate
 - Uvedite testiranje automatizacije
 - Planirajte dizajn i razvijajte skripte
 - Izvršite testove
 - Procenite rezultate testa
 - Sve dokumentovati



Automatsko testiranje softvera

Automatsko testiranje treba koristiti kada su :

Veliki i kritični projekti, kada postoje česti (promenljivi) zahtevi koje naručilac softvera želi da se implementira , pristup aplikaciji za opterećenje i preformanse kod mnogih virtuelnih korisnika.

Faktor vremena.

Planovi **za automatsko testiranje** se razvijaju zajedno sa redovnim razvojem specifikacija softverskih proizvoda, a zatim se automatski izvršavaju pomoću alata za kontinuiranu integraciju softvera.



Automatsko testiranje softvera

- Pisanje koda zahteva stručnost testera i poznavanje nekog programskog jezika i određenih alata.
- Složeni testovi i funkcionalnosti mogu biti jako teški za automatizaciju.
- Napomena: Programi i alati koji se koriste vrlo često mogu biti skupi.
- Potrebno je održavati automatske testove da bi imali validne rezultate, što će reći potrebno je i dodatno vreme za to.

Automatsko testiranje softvera

Neke od prednosti automatskog testiranja.

Automation Testing

- Fast
- Reliable
- Reusable
- Improves Accuracy
- Saves time and money
- Reduces Human-generated error
- Supports the execution of repeated test cases





Automatsko testiranje softvera

Povećana preciznost izvođenja testova:

Jedna od glavnih prednosti automatskog testiranja jeste **povećanje preciznosti izvođenja testova**.

Ljudi greške prilikom izvršavanja manuelnih testova mogu biti problematične.

- Za razliku od izvršavanja manuelnih testova, kod izvođenja automatskih testova manja je verovatnoća da će ljudska greška uticati na automatizovano testiranje.
- Kada su testovi automatizovani, oni se pokreću češće i sa većom preciznošću izvođenja nego kada se testovi pokreću ručno.
- Automatizacija testova je korisna kada se radi sa velikom aplikacijom koja ima dosta funkcionalnosti ili kada se dodaju nove funkcije.
- Pored toga, pomaže da se sve greške ili defekti u kodu identifikuju i poprave što je pre moguće.



Automatsko testiranje softvera

Brže izvršenje

- Automatsko testiranje takođe omogućava brže izvršavanje testova.
- To je zato što se testovi izvode istovremeno, paralelno umesto serijski.
- Paralelno pokretanje testova znači da se više testova pokreće istovremeno za kraće vreme.

Pouzdan rezultati testiranja

- Još jedna prednost automatskog testiranja je ta što obezbeđuje pouzdane rezultate testiranja.
- Ovo dolazi kao rezultat toga jer se testovi pokreću često i automatski.
- Automatsko testiranje softvera pomaže u brzoj identifikaciji grešaka i nedostataka, što olakšava timu da rešavanje problema onda kada se problemi otkriju.



Automatsko testiranje softvera

Smanjeni troškovi

- Automatsko testiranje takođe može dovesti do smanjenja troškova.
- Kada su testovi automatizovani, smanjena je potreba za manuelnim testiranjem.
- Pored toga, vreme potrebno za izvođenje testova je smanjeno, što dovodi do uštede u vremenu i novcu.
- Štaviše, automatski testovi mogu pomoći u smanjenju troškova razvoja softvera otkrivanjem i ispravljanjem grešaka ranijim fazama razvoja softvera.
- Oni takođe mogu pomoći u smanjenju troškova podrške aplikacije, jer će automatizovanim testovima biti potrebno manje vremena za pronađenu grešaku.



Automatsko testiranje softvera

Povećana efikasnost

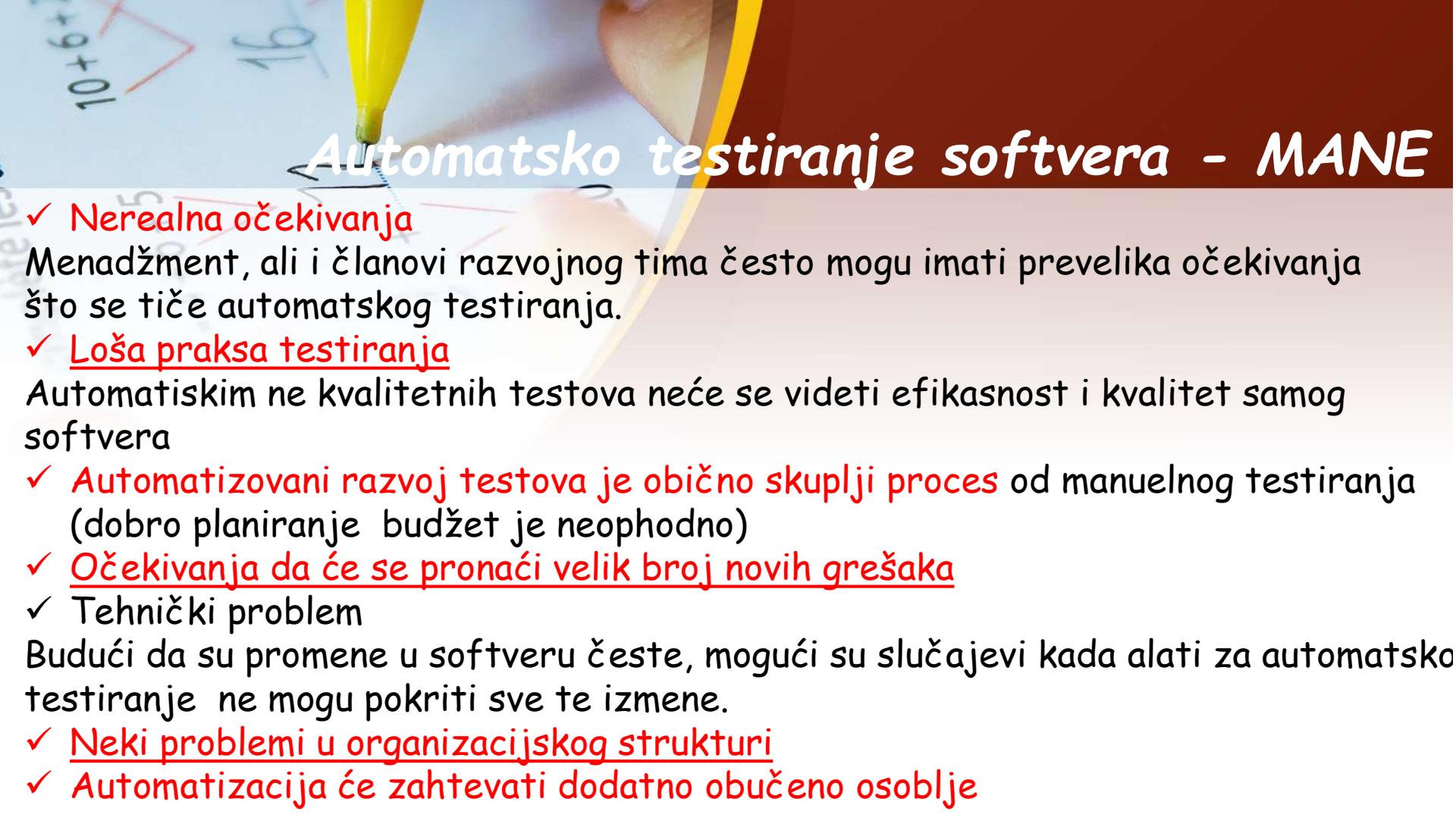
Automatsko testiranje može pomoći u poboljšanju produktivnosti test inženjera automatizacijom testova koji bi inače morali da se izvršavaju manuelno.

Povećana saradnja između programera

Automatsko testiranje može pomoći da se poboljša saradnja između programera. Kada postoji napisani testovi za neku aplikaciju, programeri mogu da se oslove na testove prilikom implementacije novih promena ili funkcija. Ovim se obezbeđuje visok nivo pokrivenosti koda i smanjuje verovatnoća grešaka u novom kodu.

Poboljšana skalabilnost

Automatski testovi se mogu koristiti na mnogim uređajima i konfiguracijama, što olakšava testiranje više stvari istovremeno. Na primer, automatski testovi se mogu pisati za merenje performansi aplikacije na različitim uređajima ili browserima. Ovo omogućava lakše testiranje i proveru rada aplikacije na različitim uređajima.



Automatsko testiranje softvera - MANE

✓ **Nerealna očekivanja**

Menadžment, ali i članovi razvojnog tima često mogu imati prevelika očekivanja što se tiče automatskog testiranja.

✓ **Loša praksa testiranja**

Automatiskim ne kvalitetnih testova neće se videti efikasnost i kvalitet samog softvera

✓ **Automatizovani razvoj testova je obično skuplji proces** od manuelnog testiranja (dobro planiranje budžet je neophodno)

✓ **Očekivanja da će se pronaći velik broj novih grešaka**

✓ **Tehnički problem**

Budući da su promene u softveru česte, mogući su slučajevi kada alati za automatsko testiranje ne mogu pokriti sve te izmene.

✓ **Neki problemi u organizacijskog strukturi**

✓ **Automatizacija će zahtevati dodatno obučeno osoblje**

ALATI ZA AUTOMATSKO TESTIRANJE SOFTVERA

Softverski alati za automatsko testiranje softvera se mogu podeliti u sledeće tri kategorije:

- Open Source alat
- Komercijalni testni alati
- Prilagođeni alati



ZAP TEST
Unlimited Software Automation

Testiranje softvera

cypress

BlazeMeter

APACHE
JMeter™

 Selenium

 Katalon Studio

 Unified
Functional
Testing

 TestComplete

 watir

 **Ranorex®**
An Idera, Inc. Company



Selenium

- Prva verzija Selenium-a nastala je 2004. godine kada je Džejson Hagins (Jason Huggins), za potrebe testiranja aplikacije u firmi ThoughtWorks. Uspeo je da napiše Javascript biblioteku koju je integrisao sa Web stranama, što mu je omogućilo automatsko izvršavanje testova u različitim browserima.



Selenium

- Kada je Jason Huggins, uvideo potencijal ove ideje da pomogne automatizaciji drugih veb aplikacija, razvijao je kreiranu skriptu i kreirao Selenium Core.
- Prilikom jednog problema u timu sa kompanijom Mercury Interactive, Jason je došao na ideju da ovaj alat dobije naziv Selenium.



Selenium

- Selenium je najzastupljenije open-source rešenje automatskog testiranja softvera, koje omogućava upravljanje web browserima, kontrolisanje njegovog ponašanja i pomoću kojeg je moguće inicirati određene akcije nad web aplikacijom. Namenjen je pisanju TestCase, koji su jako bitni u Agilnoj metodologiji rada, (na kojoj se baziraju ili koriste vodeće kompanije u oblasti sofvera)



Selenium

Selenium je skup alata za automatizovano testiranje korisnickog interfejsa Web aplikacija.

Može da se koristi za više programskih jezika (Java, C#, Python, Ruby, Perl ...) za kreiranje Selenium test skripti. Može se koristiti na Windows, Macintosh ili Linux platformama.

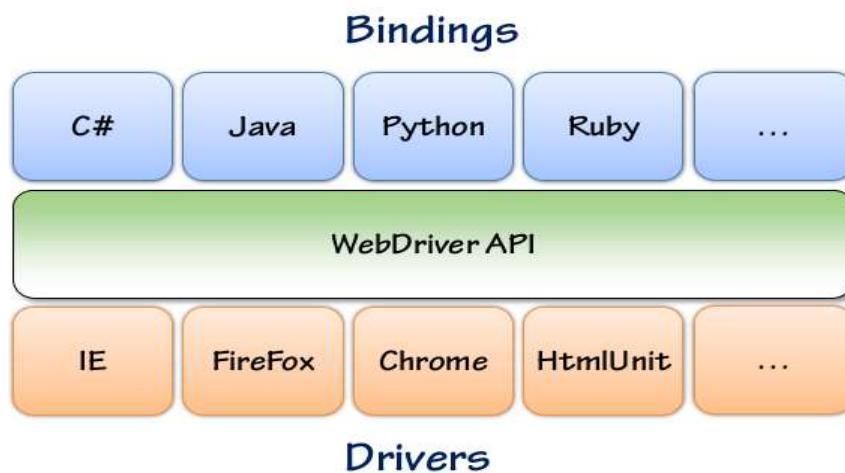
- U pitanju je softver otvorenog koda
- Lako se integriše sa drugim framework platformama i alatima
- Podržava sve zastupljenije operativne sisteme
- Podržava sve bitnije internet pretraživače
- Omogućuje pisanje testova na velikom broju programskih jezika

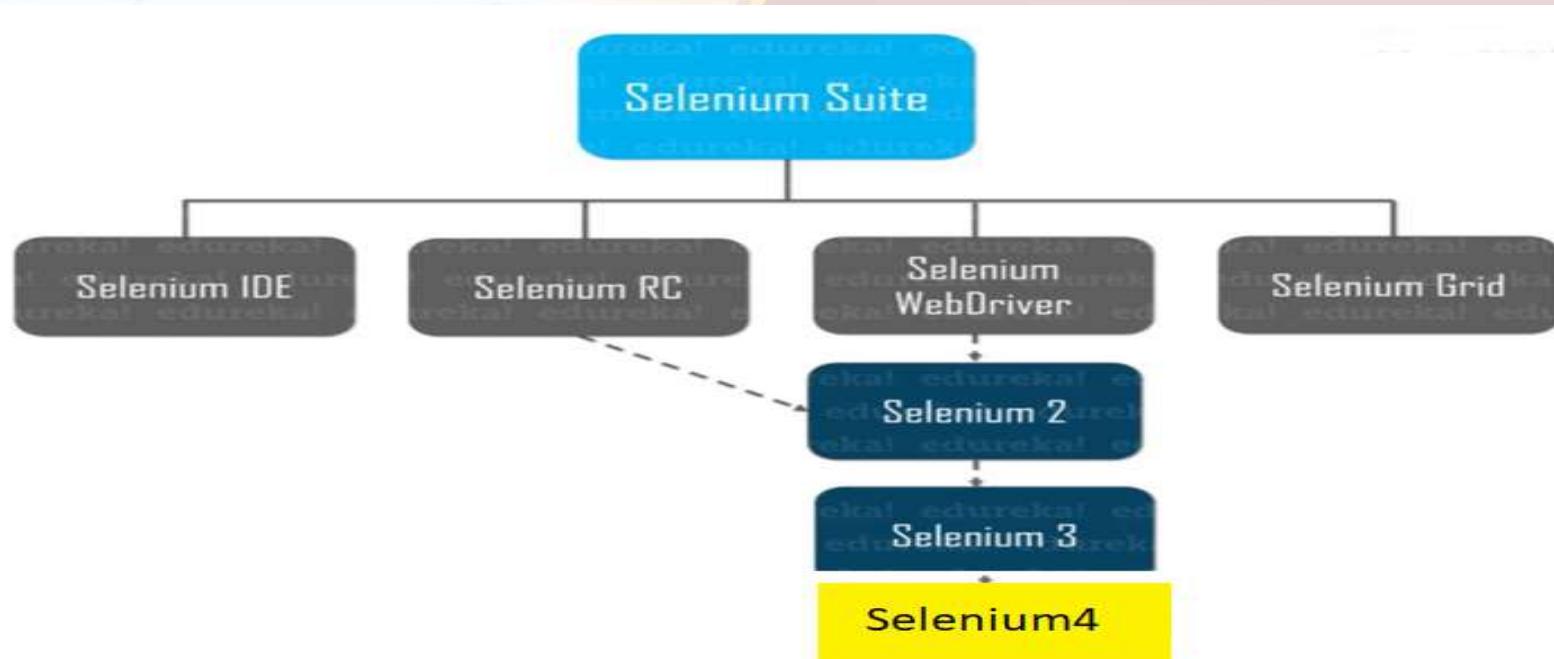


Selenium

Softver se sastoji iz četiri nezavisne komponente: Selenium IDE, Selenium Remote Control (RC), Selenium Webdriver i Selenium-Grid.

Selenium WebDriver arhitektura







- ✓ **Selenium IDE** je dodatak za Firefox koji se može koristiti samo u stvaranju relativno jednostavnih test slučajeva i programskih paketa.
 - ✓ **Selenium RemoteControl** je takođe poznat kao Selenium 1 jer predstavlja prvi alat Seleniuma koji je korisnicima omogućio da koriste programske jezike u kreiranju složenih testova.
 - ✓ **WebDriver**, predstavlja najnoviji paker koji omogućava test skriptama da komuniciraju direktno sa browserima.
 - ✓ **Selenium Grid** je alat koji koristi sa Selenium RC za izvršavanje paralelnih testova u različitim browserima i operativnim sistemima.
-
- **Selenium RC** i **WebDriver** su objedinjeni da bi se formirao Selenium 2.



Selenium

Kod pisanja automatskih testova najbitniji aspekt je interakcija sa HTML elementima veb stranice.

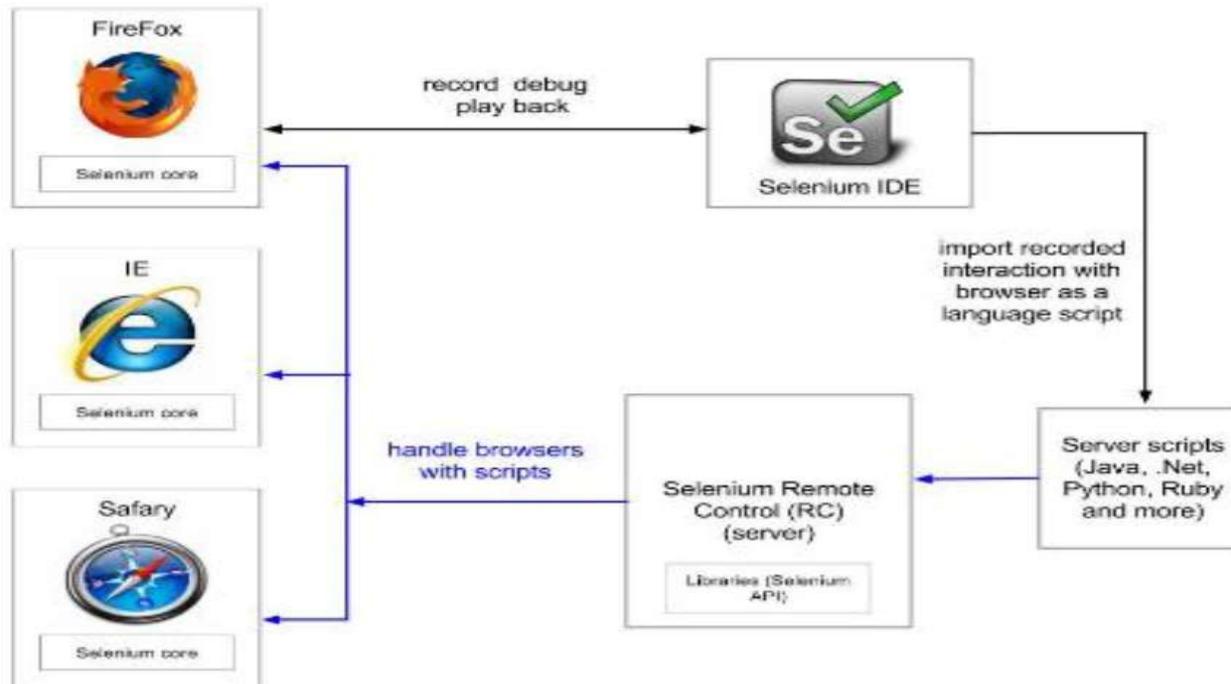
Da bi Selenium test mogao da funkcioniše potrebno je identifikovati elemente HTML stranice sa kojima test treba da komunicira.

Test skripte su niz komandi koje se snimaju automatski ili se pak mogu i ručno unositi.

Rezultate testiranja možemo proveriti u "Logovima". U slučaju da test skripta nije uspešno izvršena tu ćemo moći detaljnije da vidimo i gde se javio problem i da izvršimo detaljno analiziranje.

<https://selenium.dev/>

Selenium



Lociranje elemenata HTML stranice

- Putem
 - imena taga
 - teksta u linku
 - XPath upita
 - vrednosti atributa id
 - vrednosti atributa name
 - dela teksta u linku
 - imena CSS klase
 - CSS selektora





Apache JMeter

Apache JMeter je softver otvorenog koda. Java aplikacija dizajnirana za učitavanje test funkcionalnog ponašanja i merenje performansi.

U prvo vreme je bila implementirana za testiranje web aplikacija, ali se vremenom razvijala i na druge funkcije testiranja softvera.

Dobre karakteristike su na primer:

- ✓ Licenca otvorenog koda
- ✓ Odličan GUI
- ✓ Može da se koristi na više različitih operativnih Sistema
- ✓ Odlična simulacija
- ✓ Podržava više protokola kao što su FTP, HTTP, JDBC
- ✓ Vizuelizacija testa

<http://jmeter.apache.org/>



Cypress

- Cypress predstavlja jedan od mnogobrojnih softvera za testiranje. Tačnije, on je framework koji izvršava testove nad nekom aplikacijom u isto vreme komunicirajući i sa front-end i back-end delom softvera.
- Cypress predstavlja framework koji se sastoji od **Node.js** koji radi u pozadini i izvršava serverske procese i **JavaScript** koja radi sa svim elementima aplikacije kojima pristupa iz veb browsera.



Cypress

- Većina alata za testiranje deluje tako što se pokreće izvan veb browser i izvršava daljinske komande širom mreže, Cypress je upravo suprotno od toga. Cypress testove izvršava u istoj petlji kao i aplikacija koja se testira. U pozadini Cypress-a je Node.js serverski proces, oni neprestano međusobno komuniciraju, sinhronizuju se i izvršavaju zadatke uporedno jedan sa drugim.
- Pristup u oba dela softvera (*front and back*) pruža nam mogućnost da testiramo i odgovaramo na događaje aplikacije u realnom vremenu dok istovremeno radimo van veb browser na zadacima koji zahtevaju više pažnje.



Ranorex

- Ranorex se smatra alatom za automatsko testiranje softvera i on koristi Selenium WebDriver.

Koristi se kod web, desktop i mobilnih aplikacija.

Dosta je jednostavan za korištenje UI (*User interface*)

Osim tehničkih plus bodova, Ranorex nudi dobru korisničku podršku putem online foruma i putem e-pošte

<https://www.ranorex.com/>



Watir

- To je alat za automatsko testiranje softvera - softver otvorenog koda.
- Ruby
- Automatsko testiranje softvera web aplikacija u Rubyju.
- WatirCraft framework koji je razvijen kako bi se olakšalo i ubrzalo pisanje Watir testova
- Watir podržava vašu web aplikaciju bez obzira koju ste tehnologiju koristili za vašu aplikaciju.
- Neke kompanije koje koriste ovaj alatu su SAP, Oracle, Facebook
- <http://watir.com/>



Lambda test

- Pomoću LambdaTest-a može se izvesti i ručno i automatsko testiranje softvera.
- Neke od dobrih osobina LambdaTest-a:
- LambdaTest daje mogućnost da izvedete interaktivno i automatizovano testiranje unakrsnih browsera (na velikom broju 2000)
- ✓ Podržava paralelno izvršavanje
- ✓ Cloud alat za testiranje unakrsnih browsera i za sve aktivnosti daje akcenat na sam kvalitetet softvera
- ✓ Neograničena veb automatizacija sa Selenom
- ✓ Testiranje geolokacije
- ✓ Integrисано отklanjanje grešaka
- <https://www.lambdatest.com/>



TestComplete

- To je softverski alat za automatsko testiranje. Vlanik je softvera kompanija SmartBear.
- TestComplete nudi sveobuhvatno testiranje mobilnih, i web aplikacija.
- JavaScript, Python, VBScript i C ++
- Može obaviti testiranje ključnih reči i podataka.
- <https://smartbear.com/product/testcomplete/overview/>



Subject7

Subject7 je komercijalna platforma koja je na Cloudu.

Koristi otvorene izvore kao što su Selenium i Appium
Jednostavan za upotrebu za ne-tehničke ljudе

<https://www.subject-7.com />



ZAPTEST

- Besplatni i poslovni alat za automatizaciju bilo koje Windows aplikacije.
- Testove možete izvoditi brzo i jednostavno bez dugog čekanja na izvođenje.
- Dosta je jednostavan za korištenje UI
(User interface)
- <https://www.zaptest.com>



To je platforma za testiranje opterećenja za simulaciju bilo kog korisničkog scenarija za web aplikacije, web stranice, mobilne aplikacije ili web usluge

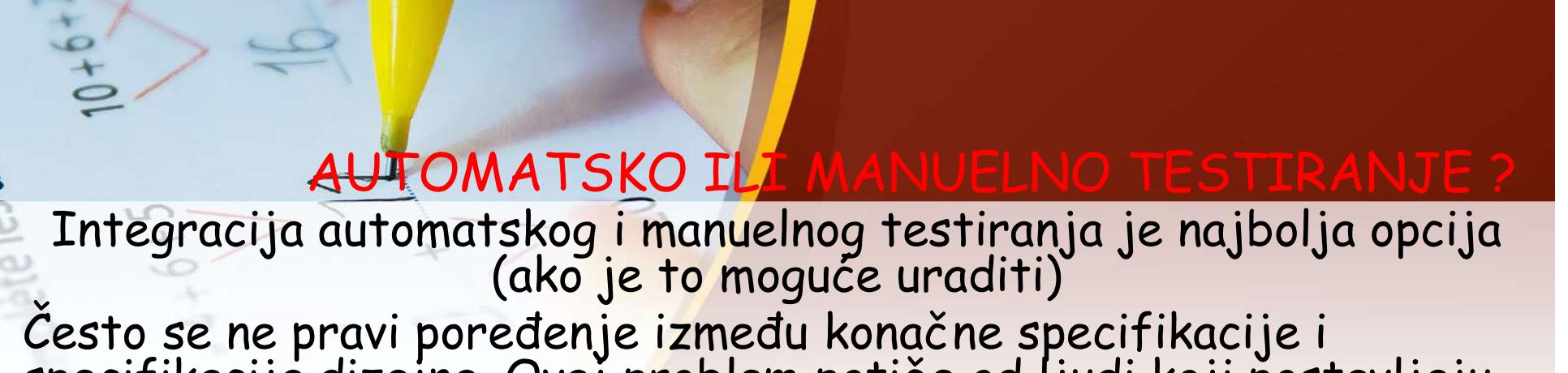
Dobar je za testiranje testova opterećenja i performansi.

Kompatibilan je sa alatom JMeter. Bilo koji JMeter test dobro funkcioniše i na BlazeMeteru.

To je licencirani alat.

Na raspolaganju je i njegovo free probno testiranje koje omogućuje 50 istovremenih korisnika, i 10 besplatnih testova.

<https://www.blazemeter.com/>



AUTOMATSKO ILI MANUELNO TESTIRANJE ?

Integracija automatskog i manuelnog testiranja je najbolja opcija
(ako je to moguće uraditi)

Često se ne pravi poređenje između konačne specifikacije i specifikacija dizajna. Ovaj problem potiče od ljudi koji postavljaju zahteve.

- Ispravnost softvera je moguće proveravati i bez njegovog izvršavanja, samo na osnovu analize izvornog koda, korišćenjem tehnika statičke analize.
- Prednost automatskog testiranja je u povećanoj preciznosti, jer automatski test će dati precizan rezultat bez obzira na broj ponavljanja testa, dok kod manuelnog testiranja postoji faktor zamora i ljudske greške.
- Statička analiza koda može biti manuelna, i podrazumeva ručne provere i pregledе koda, ili može biti automatizovana.
- Automatizacija procesa generisanja test primera i provera rezultata testiranja posebno je važna jer olakšava i ubrzava proces testiranja.



■ Manuelno testiranje:

- Proces nalaženja defekata unutar samog softvera
- Tester ponaša kao krajnji korisnik i manuelno izvršava test slučajeve bez korišćenja automatizovanih alata
- Rezultat izvršavanja koji nije očekivan je zapisan i analiziran u formi test izveštaja

■ Automatsko testiranje:

- Proces u kojem se izvršavaju pripremljeni automatski test scenariji kako bi se pronašli defekti
- Cilj: smanjenje ukupne cene manuelnog testiranja, sprečavanje ljudske greške



AUTOMATSKO ILI MANUELNO TESTIRANJE ?

- Svaki automatski test se sastoji iz nekoliko procesa:
- Priprema testova,
- Pisanje testova,
- Izvršavanje testova,
- Procena,
- Komunikacija sa developerima,
- Refaktorisanje testova,
- Ponavljanje testiranja.



AUTOMATSKO ILI MANUELNO TESTIRANJE ?

Oba pristupa imaju svoje prednosti i mane.

Testiranje je neizbežan proces ukoliko želimo implementirati uspešan i kvalitetan softver.

Softver možemo testirati ručno (manuelno) ili automatski pomoću specijalnih alata za testiranje.

Ako se može napraviti kombinacija automatskog i ručnog (manuelnog) testiranja ona može biti uspešna za neki softver, dok za neki drugi ne mora.



AUTOMATSKO ILI MANUELNO TESTIRANJE ?

- Kod manuelnog testiranja softvera tester ručno obavlja sve potrebne korake, dok kod automatskog testiranja, testovi se pokreću automatski i izvršavaju od strane programa.
- Program za testiranje piše osoba koja se bavi automatskim testeiranjem.
- Programi za testiranje se pišu u određenim alatima,
- Na taj način se pronalaze greške u softveru koje će biti naknadno ispravljene od strane programerskog tima.

AUTOMATSKO ILI MANUELNO TESTIRANJE ?

Automatizacija testova može da se izvrši imitirajući prave akcije klijenta tokom testiranja razvijene aplikacije ili kroz testiranje ne grafičkih nivoa aplikacije (API funkcionalno testiranje).

Otežavajući faktor za automatizaciju je činjenica da se pojedine funkcionalnosti menjaju često i samim tim nisu pogodne za automatizaciju, a održavanje takvih test scenarija bi oduzimalo previše vremena i resursa.







Zašto testiranje softvera ?

Do danas nije razvijena metodologija testiranja softvera, kao najvažnija aktivnost u procesu obezbeđenja kvaliteta softvera, koja garantuje da će program biti bez grešaka, ali sistematiziran i planirani postupak testiranja softvera, kroz izvršavanje programa sa pažljivo odabranim skupom ulaznih podataka, značajno povećava poverenje u korektnost programa.

- Testiranje softvera odavno nije samo faza u procesu razvoja softvera već paralelni podproces.

Pošto je stalan pritisak, u raznim poslovnim primenama, za upotrebu novih, produktivnijih programskih jezika i razvojnih alata, sve obimniji programski kod se proizvodi u vrlo kratkom vremenskom periodu



TESTIRANJE SOFTVERA

Zašto testiranje softvera?

Zato što su veliki gubici kompanija koje razvijaju softver upravo zbog velikog broja defekata u isporučenom softveru. Prvenstveni zadatak test inženjera je otkrivanje problema u softveru sa ciljem da se oni otklone pre predaje softverskog proizvoda kupcu.

Od test inženjera se zahteva da otkrije što je moguće više problema i to što više onih vrlo ozbiljnih čije posledice mogu biti katastrofalne sa materijalnog i bezbednosnog aspekta.

Zato je sa svih aspekata potrebno da se proces testiranja softvera učini što efikasnijim i uz što manje troškove ukoliko je to moguće.



TESTIRANJE SOFTVERA

- Bez softvera danas je nemoguće zamisliti savremeni svet. Softveri su postali nezamenljiva komponenta u donošenju poslovnih odluka, naučnim istraživanjima i radu mnogih inženjera. Kompanije koje se bave razvijanjem softvera najveće gubitke beleže upravo ukoliko taj isporučeni softver sadrži određene greške.
- Kako bi se tako nešto izbeglo, osnovni zadatak test inženjera jeste da što pre otkriju problem i da ga otklone pre nego što proizvod stigne do potrošača.
- Od test inženjera očekuje se da proces testiranja softvera učine što efikasnijim i uz što manje troškova obezbede normalan rad određenog programa.

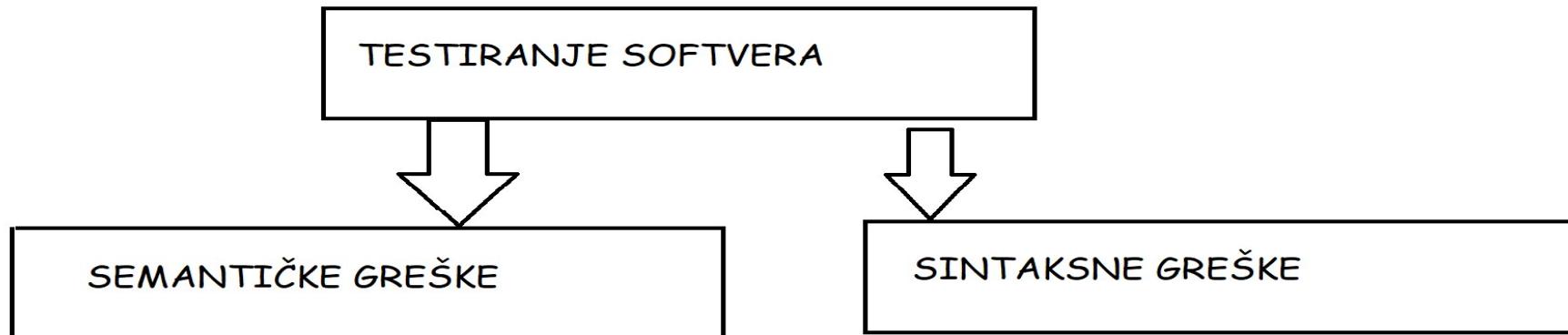


TESTIRANJE SOFTVERA

- Softversko testiranje **danas predstavlja aktivnost** koja obuhvata kompletan proces razvoja i održavanja i kao takva čini veoma važan deo cele konstrukcije softvera.

Testiranje softvera nije aktivnost koja počinje samo nakon kompletiranja faze kodiranja, već aktivnost koja se izvodi zbog evaluacije kvaliteta proizvoda i njegovog poboljšanja, putem identifikovanja defekata i problema.

Semantičke greške



Semantičke greške obuhvataju:

- Logičke greške (greške nastale zbog primene pogresne logike u programiranju)
- Greške u kodiranju (nprimer: zamena slova koja dovodi do promene imena promenljivih)



Sintaksne greške

Sintaksa u programskom jeziku uključuje skup dopuštenih fraza jezika, dok semantika izražava povezano značenje tih fraza.

Sintaksa definiše stroga pravila pisanja reči nekog jezika.

To su greške u zapisu programa koje nisu u skladu sa pravilima programskog jezika



Propusti u razvoju softvera se često nazivaju greškama ili bagovima.

"Greška (Bag) može izazvati neuspeh - tj. da dobijeni rezultat odstupa od ocekivanog." - MAYERS

Kada se pogleda literature onda postoje sledeći termini za testiranje softvera koji se odnose za pojam greške:

- ✓ Greška (eng. error),
- ✓ Defekt (eng. fault),
- ✓ Otkaz (eng. failure).



Greške / Bag

Potencijalni izvori grešaka:

Greške se mogu generisati tokom razvoja projekta, komponenti, koda i dokumentacije.

Obično se nalaze tokom izvršavanja ili održavanja softvera na najneočekivanim i najrazličitim tačkama u njemu.

Greške su uzrokovane nerazumevanjem zahteva korisnika; netačna specifikacija zahteva u projektnoj dokumentaciji i drugo



Softverska greška

- Šta je softverska greška?

Prva kompjuterska „bubica“ - HARVARD 1945 godine

Greškom se još može smatrati i pogrešan potez koji korisnik sistema može da napravi, nakon čega sistem proizvodi pogrešan rezultat.

Grešku ili nedoslednost u funkcionisanju softverskog sistema ili samog računara nazivamo **bug**.

Potrebno je razlikovati vrste grešaka koje e mogu pojaviti (error, bug, defect, failure).

Greška (Error)

Ljudske prirode koje pravi čovek , prilikom specifikacije zahteva ili implementacije programa.



- **GREŠKE**

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

- **OPET- VREMENSKI FAKTOR JE VEOMA BITAN!!!**



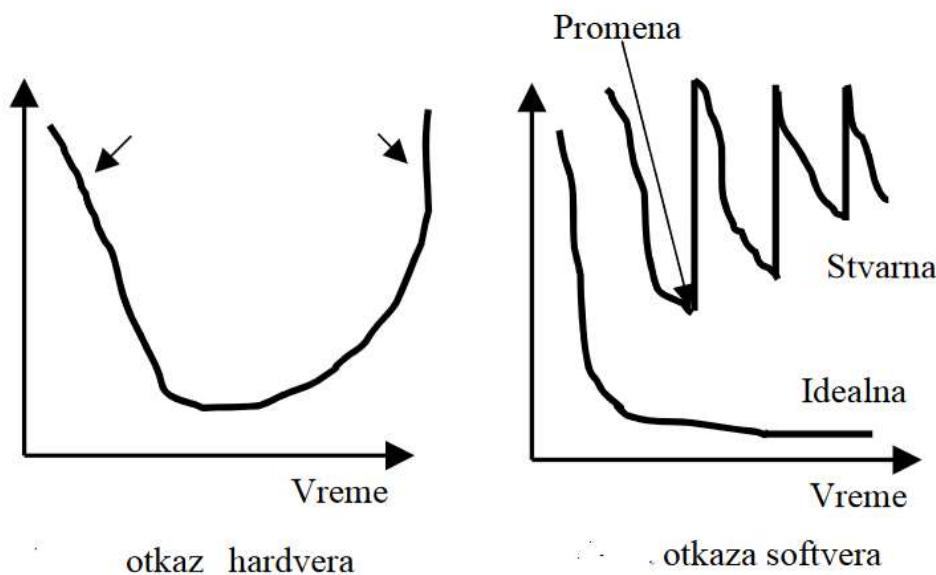
Softverska greška

Error / Mistake	Defect / Bug/ Fault	Failure
Found by	Found by	Found by
Developer	Tester	Customer

Softverska greška

Promene

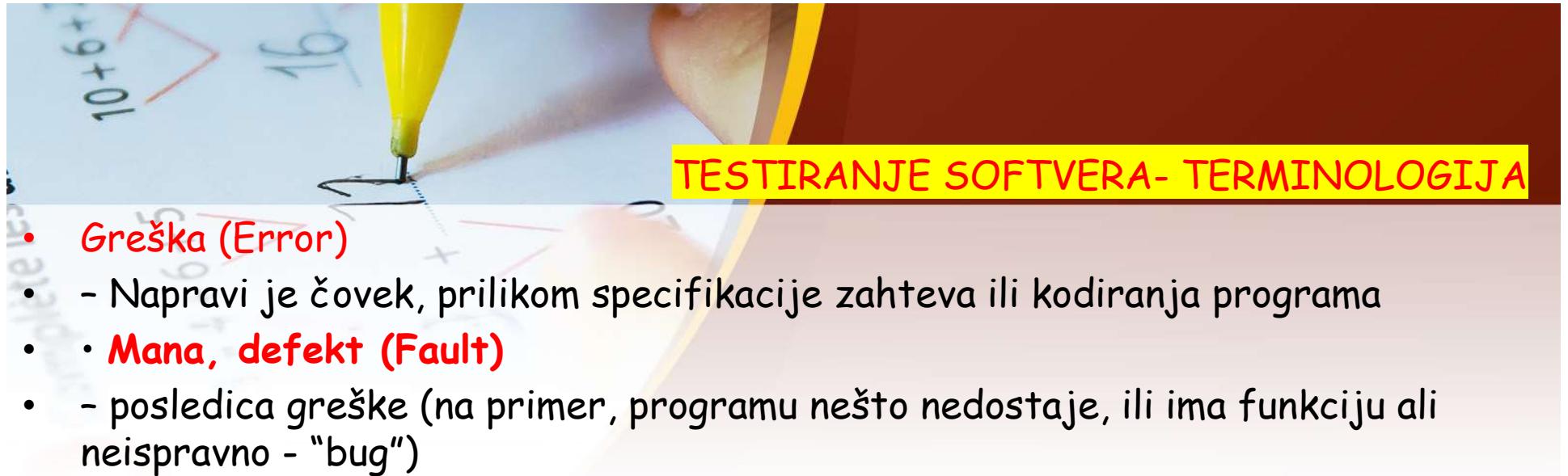
- zahtevi se menjaju
- optimizacija





Greške zbog nepoštovanja standarda

- ✓ Postavljaju se standardne procedure za izradu softvera (greške nastaju zbog nepoštovanja i nepredržavanja zadatih procedura)
- ✓ Nekada mogu da utiču na rad programa, ali nekad i ne moraju
- ✓ Iako program radi, postoji mogućnost da softverske greške stvaraju probleme pri testiranju softvera i njegovom održavanju (logika rada)



TESTIRANJE SOFTVERA- TERMINOLOGIJA

- **Greška (Error)**
- - Napravi je čovek, prilikom specifikacije zahteva ili kodiranja programa
- • **Mana, defekt (Fault)**
- - posledica greške (na primer, programu nešto nedostaje, ili ima funkciju ali neispravno - "bug")
- • **Otkaz (Failure)**
- - Nemogućnost sistema da obavi zahtevanu funkciju - javlja se aktiviranjem (izvršavanjem) defektnog koda..



Softverska greška (error)

Šta je softverska greška?

Greškom se još može smatrati i pogrešan potez koji korisnik sistema može da napravi, nakon čega sistem proizvodi pogrešan rezultat.

Softverska greška (error) je deo koda koji je delimično ili totalno pogrešan najčešće kao rezultat gramatičke (sintaksne) greške.

Logičke greške (greske nastale zbog primene pogresne logike u programiranju)



Softverska greška (error)

Postoje više podela bagova, [Jorg, 1995]: **GREŠKA**, (Kada ih načine prilikom kodiranja, greške nazivamo "bagovi". Greške imaju težnju širenja) Po Jorg-u napoznatija podela je na:

- **hardverske**
- **softverske.**

Kada se testiranjem programa utvrdi da se on ne ponaša kao što se očekuje, pristupa se analiziranju i lociranju bagova.

Detaljan postupak identifikaciji bagova omogućuje se metodičkim pristupom testiranju programa.

Potraga za bagovima olakšava testiranje programa, ali ne može adekvatno da ga zameni.

Poznato je takođe da nikakvo testiranje ne može da otkrije sve bagove.

U vezi sa testiranjem programa često se rade statičke analize (istražuju se osnovni programi, pri tome traže se osnovni problemi i prikupljaju podaci bez izvršavanja programa) i dinamičke analize (istražuje se ponašanje programa u izvršenju i tako se dobijaju podaci vezani za puteve izvršavanja, vremenske profile i pokrivenosti samog testiranja programa).



- **Šta je softverska greška? (1 ili 5 metara!)**

Grešku možemo posmatrati kroz nekoliko primera.

Ako uzmemo npr. razliku od 5 metara između izračunatog rastojanja dve tačke i tačnog rastojanja, to predstavlja grešku koju možemo pripisati loše prikupljenim zahtevima, odnosno podacima.

Ako uzmemo npr. razliku od 1 metara između izračunatog rastojanja dve tačke i tačnog rastojanja, to predstavlja grešku koju možemo pripisati loše prikupljenim zahtevima, odnosno podacima.

Isto tako pogrešno odabrana instrukcija ili pogrešan rezultat neke operacije u programu predstavlja grešku koja može dovesti do pogrešnih podataka u sistemu.



Zašto je bitno praviti izveštaje o greškama?

1. Proces izveštavanja o greškama i preduzetim akcijama mora biti precizno definisan.
2. Česti su slučajevi otežanog identifikovanja grešaka usled nerazumevanja zahteva.
3. Postoje i problemi kada neke izmene u kodu (nove opcije) nisu pravilno definisane.
4. Sam proces prihvatanja zahteva i ispravljanja mora biti vođen po unapred definisanoj proceduri.
5. Moraju se dati precizan odgovor na pitanja ko prijavljuje grešku, kako je prijavljuje, kako je opisuje, kakvog je tipa, kakav je status greške, ko prihvata i dalje delegira zahtev itd...



U literaturi u nekim knjigama se navodi kao termin Nedostatak (fault, defect, bug)

Softverski defekti (faults) su softverske greške koje uzrokuju nepravilno funkcioniranje softvera tokom specifične primene.

Fault je softverski defekt koji prouzrokuje failure, Failure je neprihvatljiva programska operacija u odnosu na programske zahteve.

DEFEKT, (Defekt je rezultat greške.

Preciznije rečeno, defekt je prikaz greške, pri čemu je prikaz način izražavanja, kao što je opisni tekst, dataflow dijagrami graf hiperarhije, osnovni kod itd.)



Softverski defekti

Nedostatak (fault, defect, bug)

- ✓ mana u programu koja može da onemogući program da izvrši namenjenu funkcionalnost
 - ✓ ukoliko se pri izvršavanju programa izvrši ovaj deo koda

Nedostatak se manifestuje pojavom otkaza

- u toku testiranja ili
- Kada se softver koristi



Softverski defekti

Statistike pokazuju da određeni broj softverskih defekata se preokrene u softverske otkaze u ranim ili kasnijim fazama primene aplikacije.

Ostali softverski defekti ostaju skriveni, nevidljivi za korisnike softvera, ali uvek postoji mogućnost da se aktiviraju ako se situacija izmeni.



- Defekt
 - greška u softveru koja uslovljava da se softver ne ponaša kao što je očekivano. Defekti u programskom kodu često se nazivaju bagovi (engl. Bug).
- Pronalaženje defekata
 - proces analize softvera kako bi se pronašlo šta uzrokuje pronađenu programsku grešku. Često se naziva debagovanje (engl. Debugging). Pošto se programska greška pronađe i ispravi, sistem postaje stabilniji.



Reliability

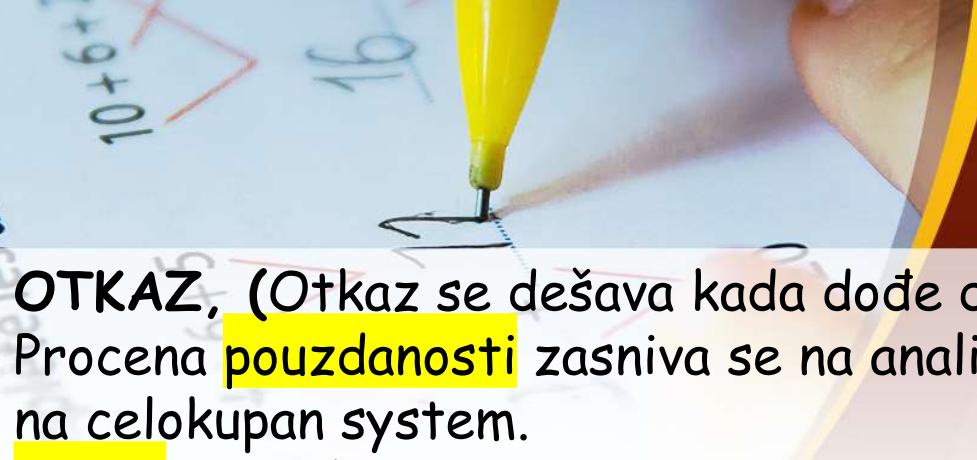
- Pouzdanost softvera (*reliability*) se definiše kao verovatnoća izvršavanja operacija softverskog sistema bez greške (*failure-free operation*) za specifično vreme u specifičnom okruženju.

Pouzdanost se definiše kao niz atributa koji predstavljaju mogućnost softvera da održi svoj nivo performansi pod određenim uslovima u određenom periodu vremena i ona je jedan od ključnih faktora kvaliteta softvera.(ISO 9126)



Reliability

- ✓ IEEE 982.1-1988 definiše upravljanje pouzdanošću softvera kao proces optimizacije softvera kroz tri aktivnosti u kontekstu ograničenja u projektu kao što su resursi, vreme i performanse:
- ✓ Prevencija greške (error)
- ✓ Detekcija i otklanjanje fault-ova
- ✓ Merenje da bi se dobila maksimalna pouzdanost za dve aktivnosti (prevencije greške i Detekcija i otklanjanje fault-ova)



OTKAZ

OTKAZ, (Otkaz se dešava kada dođe do defekta.)

Procena **pouzdanosti** zasniva se na analizi otkaza softvera i njihovog uticaja na celokupan system.

Otkaz se može definisati i kao odstupanje ponašanja, softvera u odnosu na zahteve ili specifikaciju softverskog proizvoda.

Otkaz softvera je evidentan događaj postojanja greške u softveru tj. posledica, manifestacija greške u softveru.

Otkaz softvera može biti posledica grešaka u kodu, pogrešnog korišćenja, pogrešne interpretacije specifikacije koju softver treba da zadovolji ili nestrupnosti programera, primene neodgovarajućih testova ili drugih nepredviđenih problema.

INCIDENT, (Kada se desi otkaz, korisnik to možda neće ni primetiti.

Incident je simptom praćen otkazom ko i obaveštava korisnika o dešavanju



TESTIRANJE SOFTVERA

- Potencijalni bagovi, u prošlosti, sadašnjosti i budućnosti

- Aritmetički bagovi
- Deljenje sa nulom
- Gubitak preciznosti zbog zaokruživanja
- Logički bagovi
- Beskonačna petlja
- Beskonačna rekurzija

TESTIRANJE SOFTVERA - TERMINOLOGIJA

✓ Test slučaj

- stanje opisano akcijom koju treba izvršiti, podacima koje treba poslati sistem, kao i očekivanim odgovorom od strane softvera. On opisuje šta će biti testirano (stavke), korake i podatke koji će biti korišćeni u testiranju softvera. Umesto ovog termina u literaturu se koristi i termini test primer ili slučaj testiranja.

✓ Test scenario

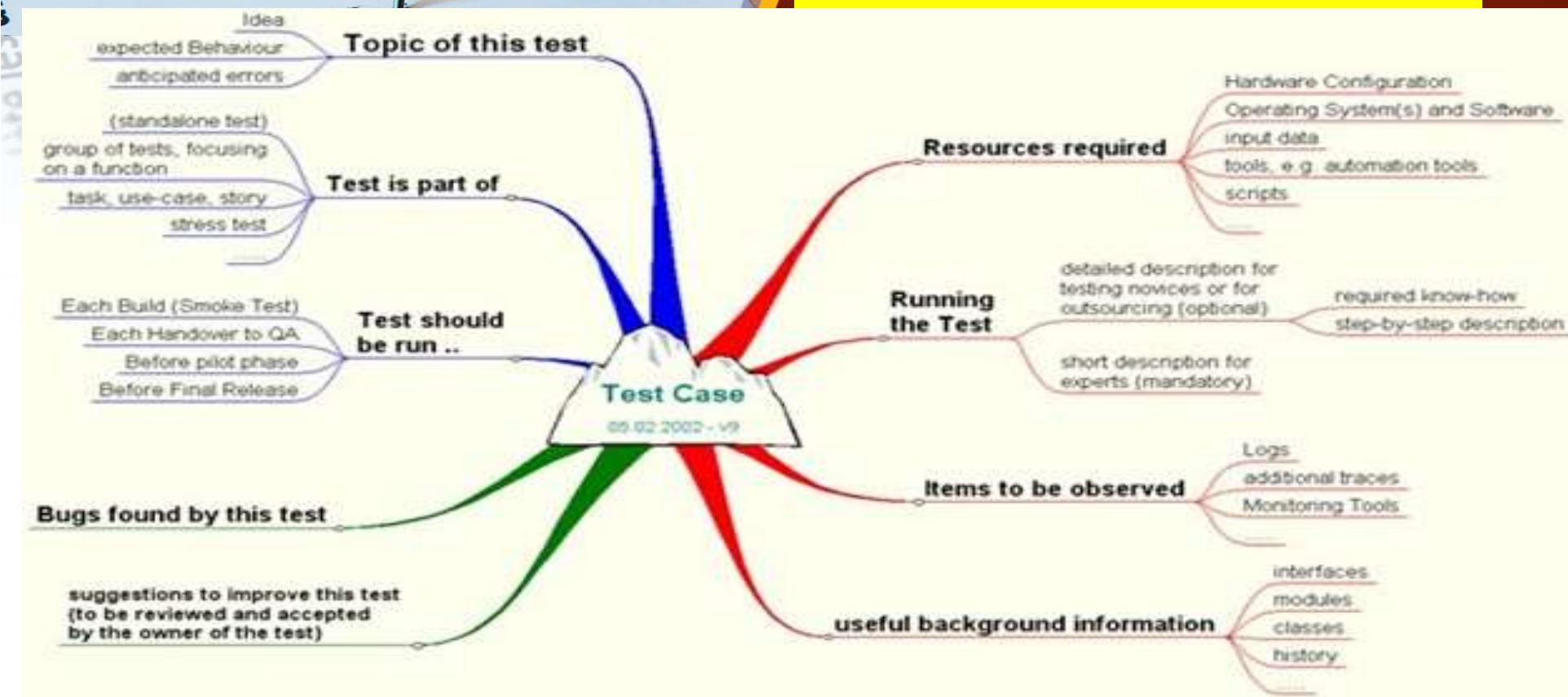
- sekvenca akcija koje treba izvršiti tokom testiranja sistema kako bi se proverili jedan ili više test slučaja.

Često se umesto termina test scenario koristi termin test procedura ili test skript



- ✓ Faza gde se određuju softverski i hardverski uslovi testiranja određenog proizvoda.
- ✓ Identificiranje test slučajeva može potrajati i ponekad moramo ponoviti test i po nekoliko puta.
- ✓ Testni slučajevi sadrže niz radnji koje izvodimo kako bismo poboljšali određenu funkciju ili funkcionalnost.

TEST SLUČAJEVI





- Testni slučajevi su jedan su od najvažnijih delova životnog ciklusa razvoja softvera koji je odgovoran za razvoj programa. (i na kraju da li on radi ispravno ili ne).
- Ovde postoje i takozvani test scenariji - To je skup skup testnih slučajeva, koji određuju pozitivne i negativne aspekte projekta kako bi se dala ocena softvera a u isto vreme identifikovali potencijalni nedostaci u program.

TEST CASE VERSUS TEST SCENARIO

Test Case	Test Scenario
<p>It's a set of variables or conditions which determine the viability of a software application.</p>	<p>It's a series of test cases executed one after the other to determine the functionality of the system or application.</p>
<p>It is a detailed document consisting of application requirements, preconditions, test data, post conditions and expected results.</p>	<p>It is a detailed test procedure consisting of test cases which help find problems in the system and evaluating results.</p>
<p>QA team and development team write test cases.</p>	<p>Reviewed by business analyst/ business manager.</p>
<p>It is important when development is done onsite and testing is done off-shores.</p>	<p>It is beneficial when time to build test cases is less.</p>
<p>More resources are required for writing test cases which is a waste of time and money.</p>	<p>It's a collaborative effort which reduces complexity thereby saving time and money.</p>



Test objekat

Test objekat

- Komponenta (Softverska) koja se testira

Testiranje je svako izvršavanje ili analiziranje test objekta kako bi se utvrdilo njegovo ponašanje

- porede se stvarno i očekivano ponašanje da bi se utvrdilo da li test objekat zadovoljava zahteve

Uслови testa (test conditions)

- koje osobine ili funkcionalnosti test objekta treba verifikovati
- verifikacija se vrši kroz test slučajeve



- Česti bagovi - neki primeri

Sintaksni bagovi

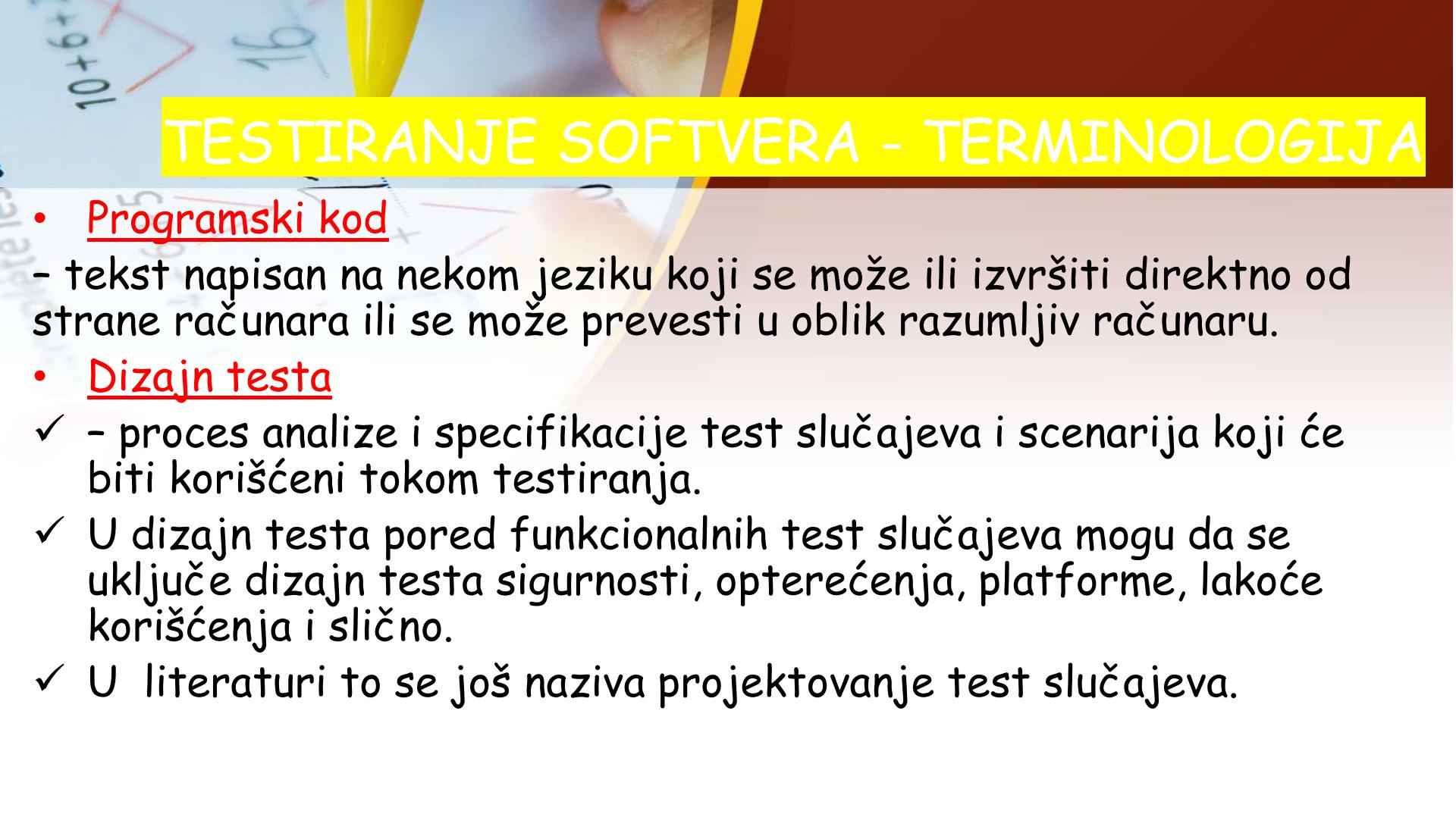
- Korišćenje pogrešnog operatora, na primer operatora dodele vrednosti umesto operatora poređenja jednakosti
- **Resursni bagovi**

Dereferenciranje NULL pokazivača

Korišćenje neinicijalizovane promenljive

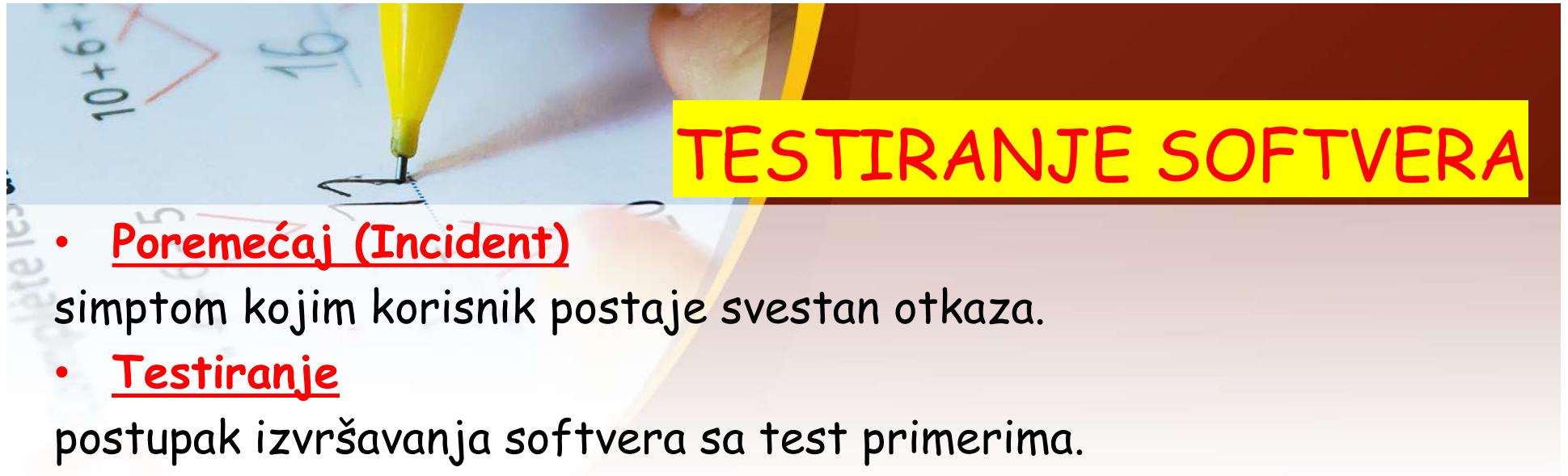
Pristup nedozvoljenom području memorije (segmentation fault)

Preduboka rekurzija, koja ima za posledicu prepunjavanje steka



TESTIRANJE SOFTVERA - TERMINOLOGIJA

- Programski kod
 - tekst napisan na nekom jeziku koji se može ili izvršiti direktno od strane računara ili se može prevesti u oblik razumljiv računaru.
- Dizajn testa
 - ✓ - proces analize i specifikacije test slučajeva i scenarija koji će biti korišćeni tokom testiranja.
 - ✓ U dizajn testa pored funkcionalnih test slučajeva mogu da se uključe dizajn testa sigurnosti, opterećenja, platforme, lakoće korišćenja i slično.
 - ✓ U literaturi to se još naziva projektovanje test slučajeva.



- Poremećaj (Incident)

simptom kojim korisnik postaje svestan otkaza.

- Testiranje

postupak izvršavanja softvera sa test primerima.

Dva su različita cilja testiranja: da se nađu otkazi, ili da se demonstrira ispravno izvršavanje

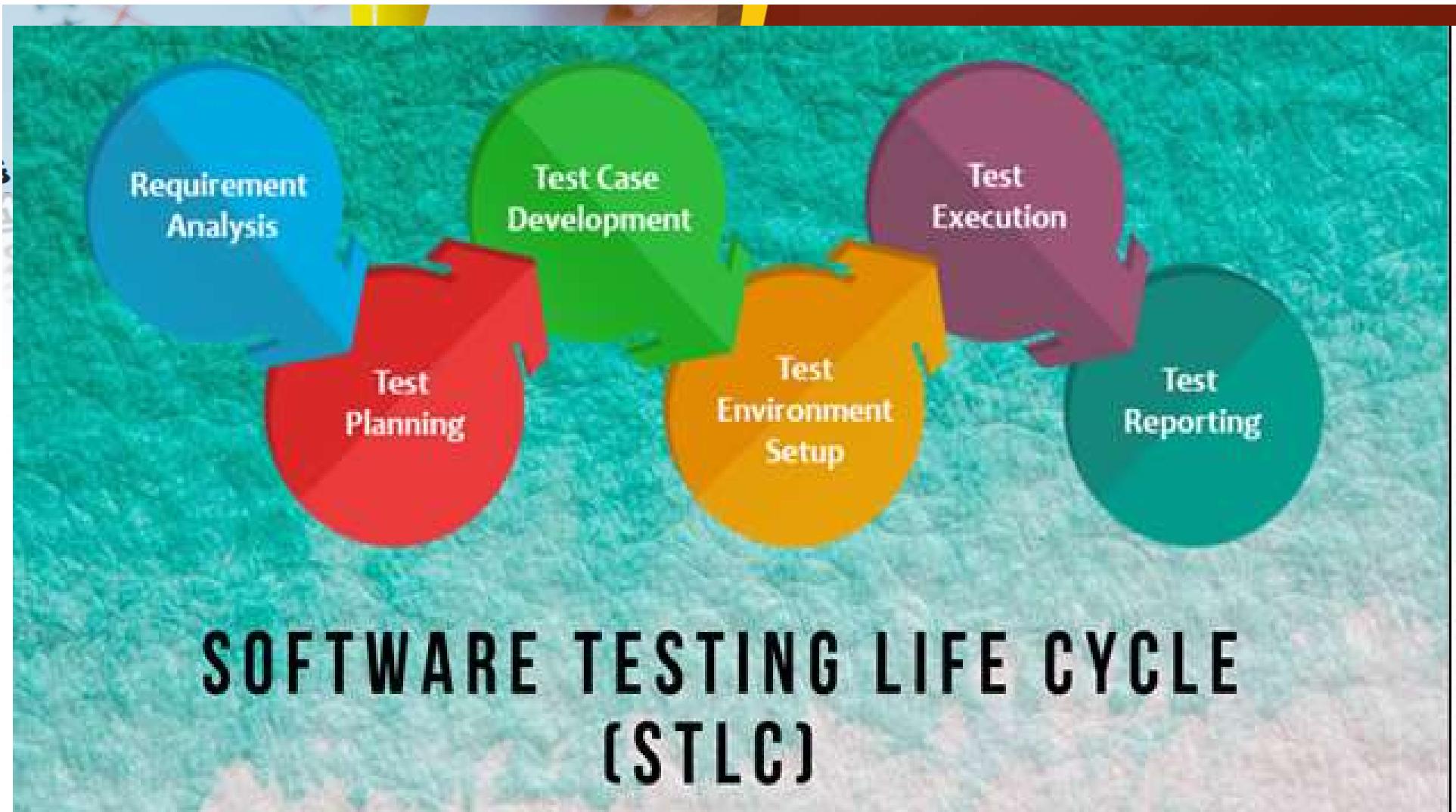
Testiranje može otkriti otkaze, ali potom treba eliminisati defekte debagovati program, što je posebna aktivnost

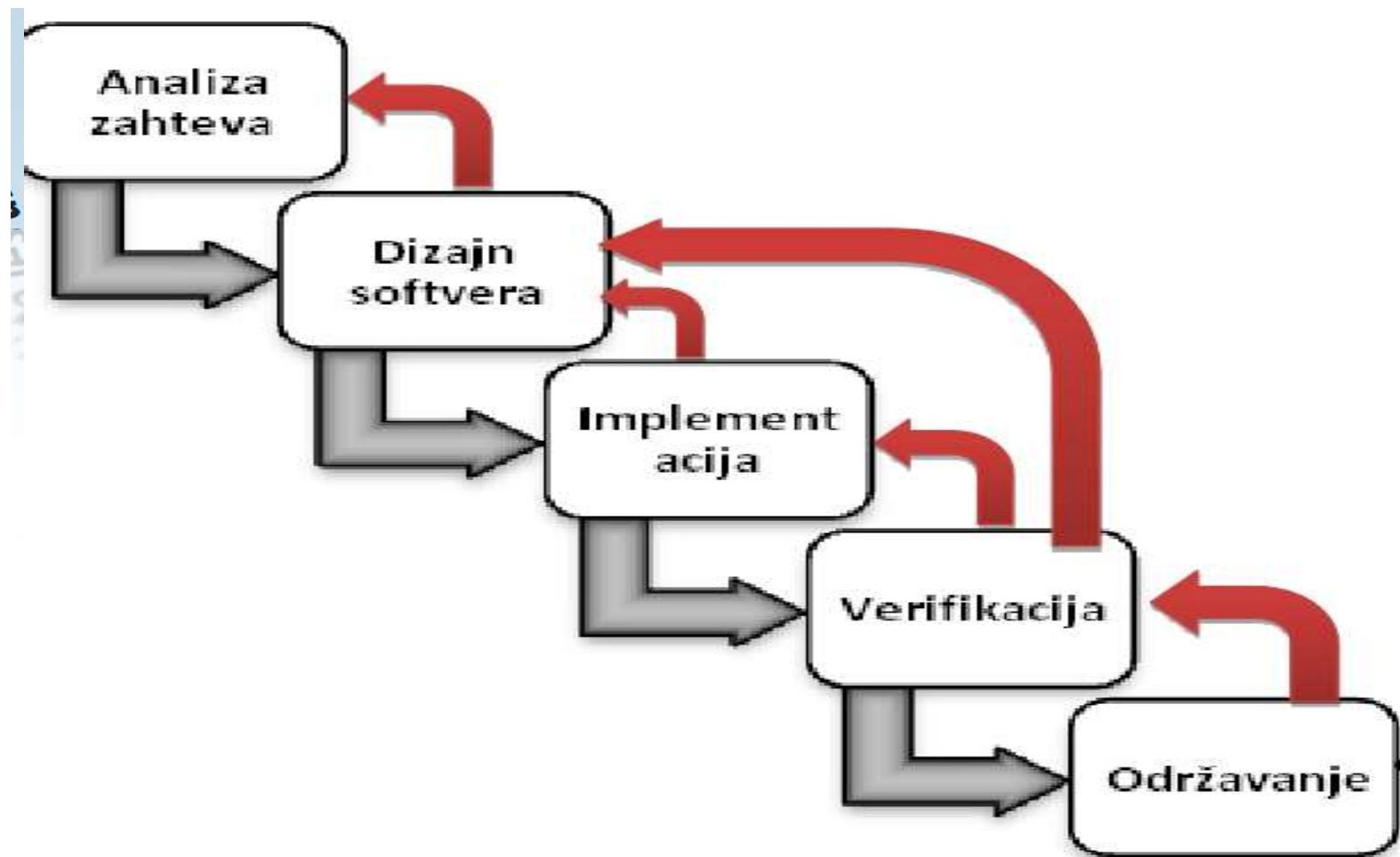


MODEL I ŽIVOTNOG CIKLUSA SOFTVERA

Životni ciklus softvera obuhvata period od definicije zahteva do prestanka njegovog korišćenja.

- ❖ Model životnog ciklusa opisuje procese iz razvoja, korišćenja i održavanja softverskog proizvoda u toku njegovog životnog ciklusa.
- ❖ Za konkretan proizvod potrebno je izabrati konkretan model (implementirati standard).
- ❖ Ne postoji jedinstveni optimalan model za sve softverske proizvode, obično se primenjuje neki od standardnih modela ili neka kombinacija istih







Faze se sekvensijalno izvršavaju dok se ne završi projekat. Faze u modelu vodopada su:

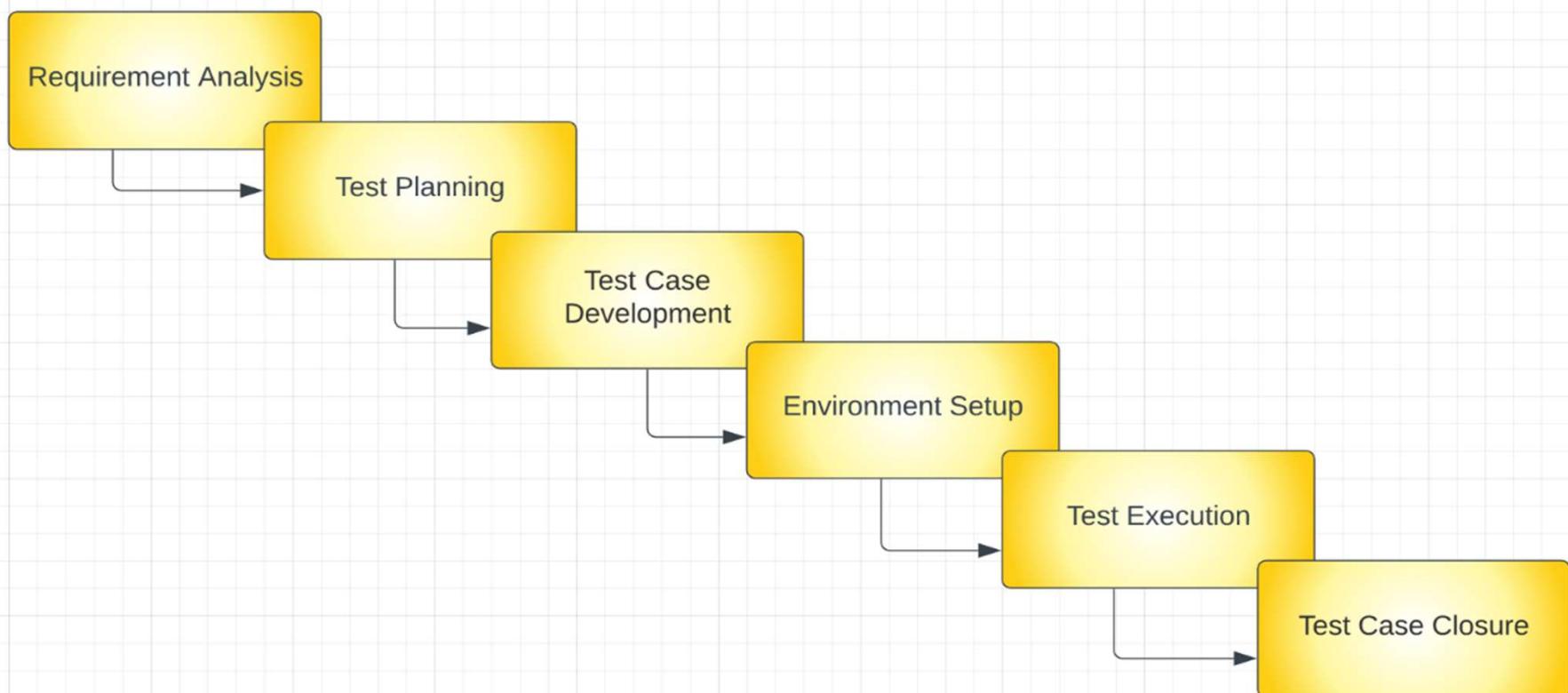
- Analiza zahteva -faza gde se sakupljaju zahtevi od krajnjih korisnika, analizira šta je potrebno raditi, kreiraju ugovori kojima se definiše šta će biti urađeno i potvrđuju zahtevi koji će biti implementirani.
- Dizajn softvera -faza gde se detaljnije analiziraju zahtevi prikupljeni tokom analize, specificira kako će se implementirati softver, kreira tehnička specifikacija i dizajn softvera. Dizajn softvera predstavlja konkretan plan kako će biti implementiran sistem od generalne arhitekture softvera do detaljnog opisa implementacije pojedinih komponenti i algoritama. Na kraju ove faze je poznato kako će se sistem implementirati



MODEL VODOPADA

- Implementacija - faza gde se projektovani softver implementira u određenom programskom jeziku i platformi. Na kraju ove faze softver je završen i spreman za upotrebu.
- Verifikacija - faza u kojoj se planira testiranje, testira sistem koji je implementiran u prethodnoj fazi, prijavljuju i otklanjaju problemi nađeni u softveru. Na kraju ove faze softver je testiran i spreman na predaju krajnjim korisnicima.
- Održavanje - tokom faze održavanja softver je predat krajnjim korisnicima i vrše se eventualne dorade u skladu sa izmenama traženim od korisnika. Ova faza traje sve dok korisnici upotrebljavaju softver.

Životni ciklus razvoja softvera





Test Planning

Kada se završi faza analize zahteva može preći na fazu planiranja testiranja (Test Planning).

Faza planiranja podrazumeva kreiranje test plana - dokumenta plana testiranja koji služi kao strateški plan za testiranje.

Dokument sadrži detaljnu procenu vremena i troškova, potrebne resurse, podelu odgovornosti i određivanje okruženja za testiranje

Faza analize zahteva (*Requirement Analysis*)

Podrazumeva analizu funkcionalnih i nefunkcionalnih zahteva, kao i pregled korisničkih zahteva pri čemu se definišu test zahtevi.

U ovoj fazi se definišu tipovi testova koji će se pisati, definišu prioriteti zahteva, priprema se dokument za praćenje korisničkih zahteva pomoću test slučajeva, određuju detalji test okruženja (*Test Environment*) u kojem će se izvoditi testovi i ukoliko je moguće kreira izveštaj o izvodljivosti automatizacije.



Test slučajevi se kreiraju u fazi razvoja test slučajeva (Test Case Development).

Testovi se zatim proveravaju tako da svaki deo softvera radi kako je predviđeno za krajnjeg korisnika.

Razvoj test slučajeva zavisi od obima zahteva i kriterijuma testiranja.

U ovoj fazi se, ukoliko je to moguće, kreiraju skripte za automatizaciju.

Environment Setup i Test Execution

- Postavka test okruženja (*Environment Setup*) je važna jer predstavlja uslove pod kojima će se izvršavati testovi. Ova faza uključuje stvaranje test okruženja koje blisko simulira okruženje iz stvarnog sveta. Tim za testiranje koristi ovo okruženje za testiranje cele aplikacije.
- Faza implementacije testova (*Test Execution*) je faza izvršavanja napisanih test slučajeva pri čemu se proizvod validira i pronalaze greške. Ukoliko greške postoje one se prijavljuju razvojnom timu koji dalje rešava problem nakon čega se on ponovo testira.



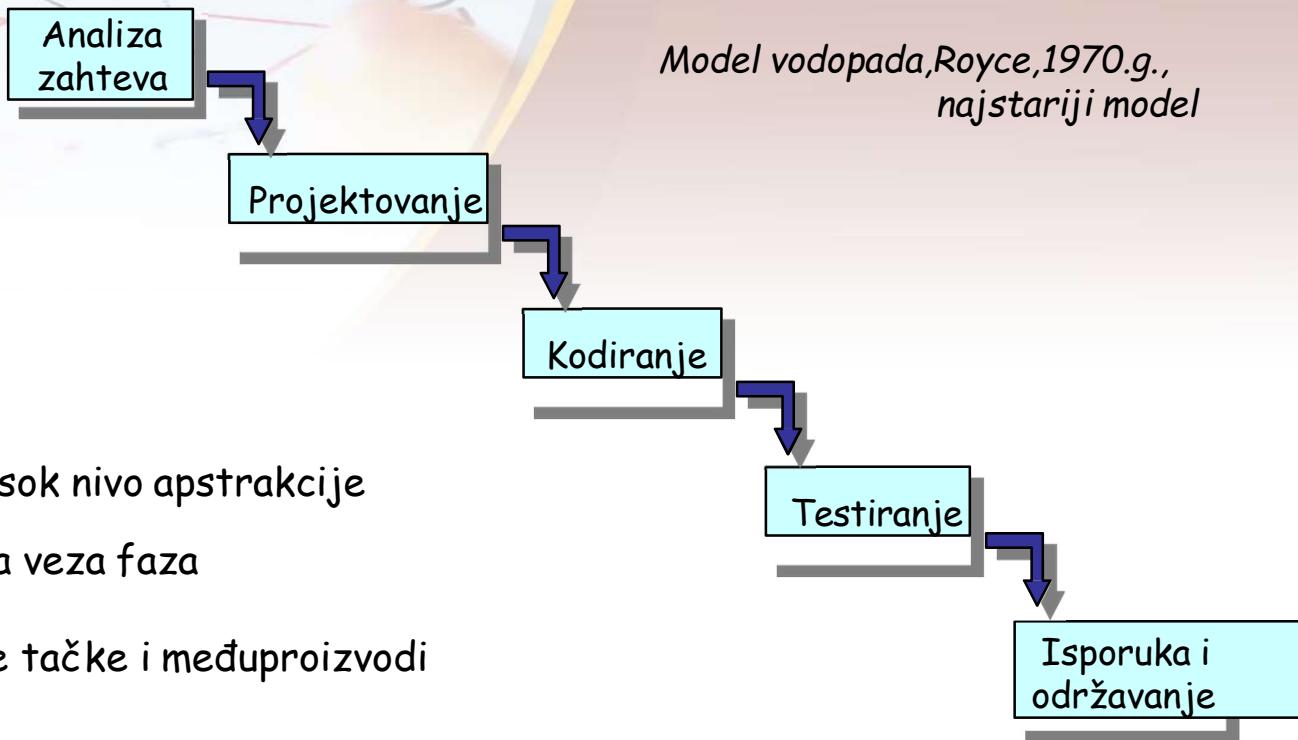
Test Case Closure

- Nakon faze implementacije testova sledi faza završetak životnog ciklusa testiranja softvera (*Test Case Closure*).
- U ovoj fazi tim se okuplja i analizira ceo tok trenutnog ciklusa testiranja i dolaze do zaključka koje će im u budućim projektima koristiti za poboljšanje načina testiranja.

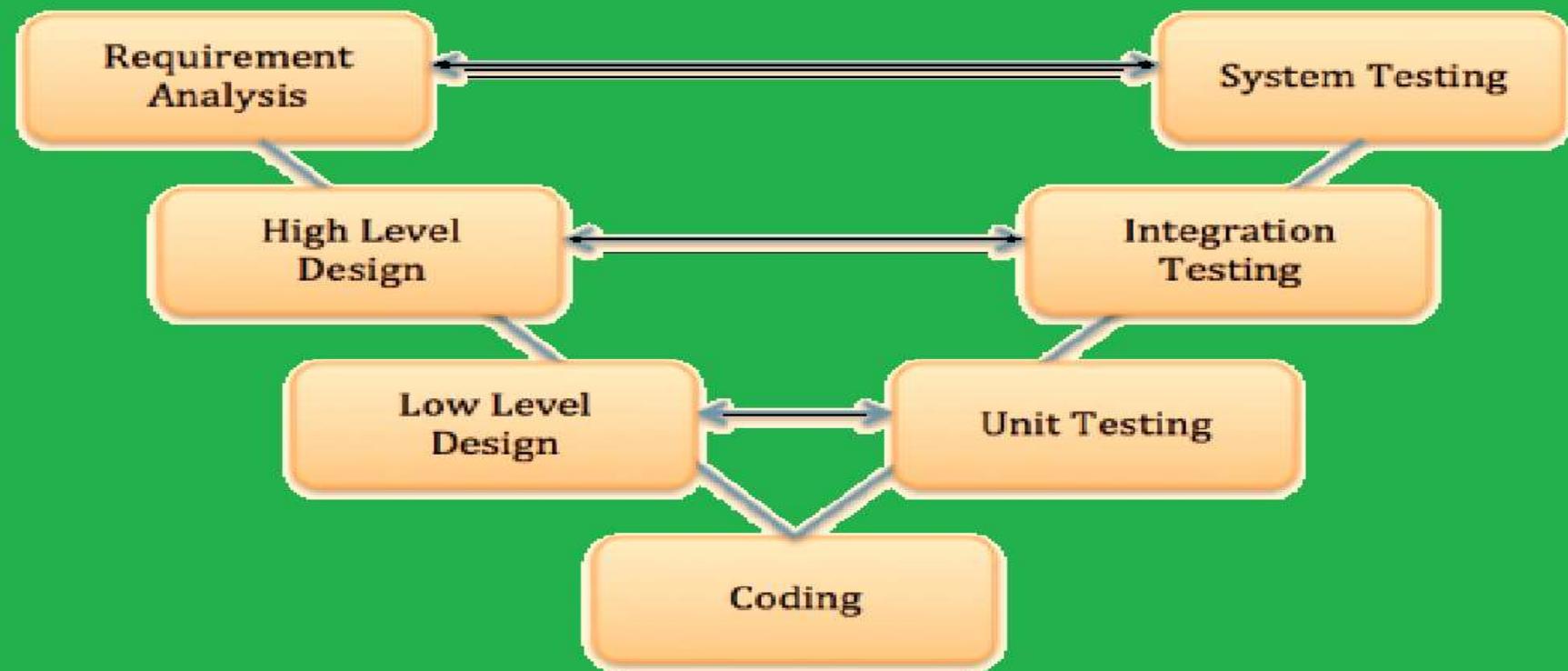
Životni ciklus razvoja softvera (Software development life cycle, SDLC)



KASKADNI MODEL

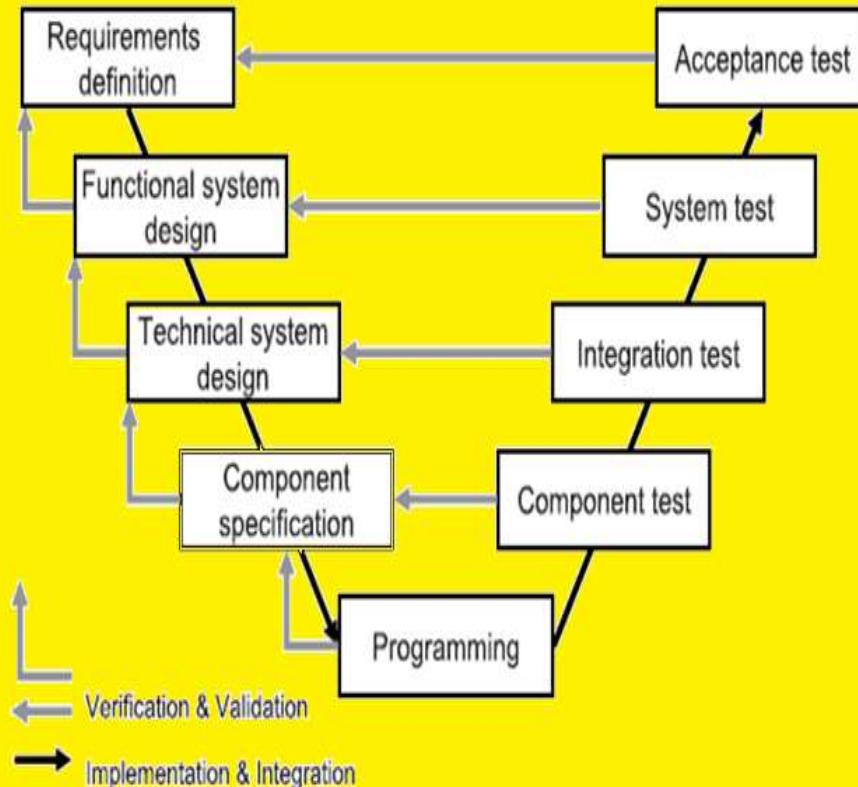


V model životnog ciklusa razvoja softvera



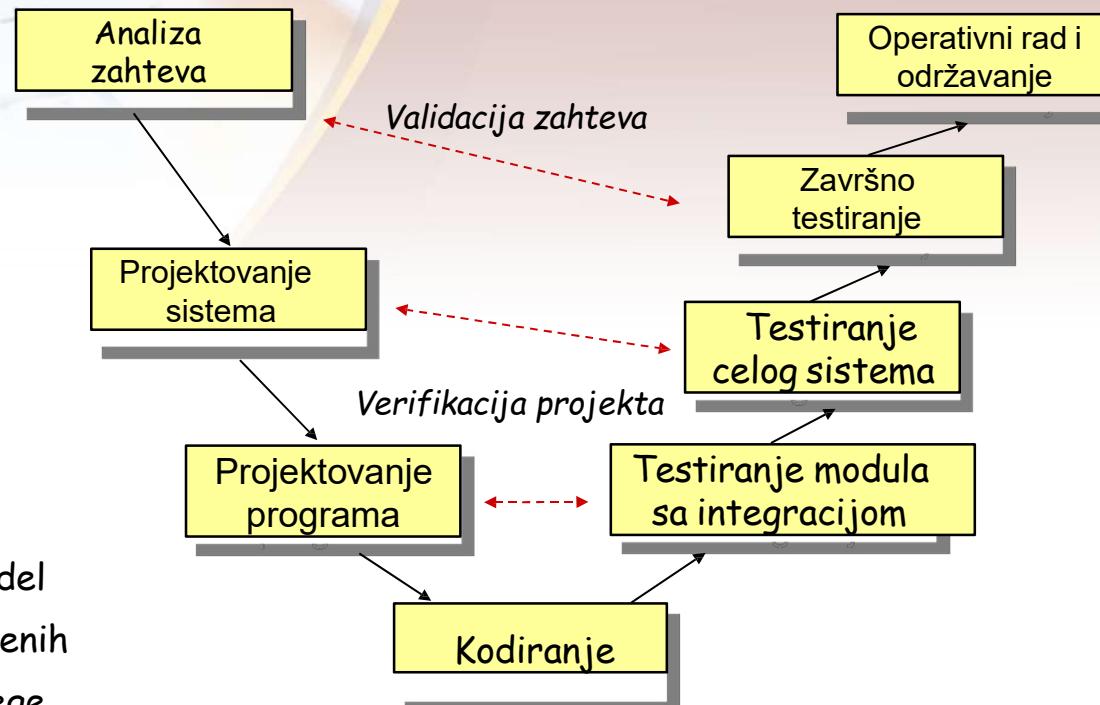
V model

Iz ovoga se vidi koliki je značaj testiranju softvera a koliki je značaj razvoja softvera.



- ✓ Razvoj funkcionalnosti softvera je prikazan na levoj strani slike
- ✓ Proces testiranja i integracije je prikazan na desnoj strani slike
- Svaka faza u testiranju softvera verifikuje jednu od faz razvoja softvera

V MODEL)



Osobine

- modifikovan kaskadni model
- jedan od najčešće korišćenih
- eksplicitne povratne sprege

Nemačko Ministarstvo odbrane, 1992.g

Aktivnosti vezane za testiranje V Model

Testiranje komponenti

Component testing je testiranje komponenti koje nastaju spajanjem više jedinica koda

- zove se i jedinično testiranje
- nezavisno testiranje svake pojedinačne komponente da se utvrdi da li zadovoljava zadatu specifikaciju ili ne

Integraciono testiranje

- otkriti potencijalne probleme i greške u interakciji između integrisanih celina.
- Integration testing se koristi da bi se proverile veze između različitih komponenti koje zajedno čine neki deo sistema

Aktivnosti vezane za testiranje V Model

Sistemsko testiranje

- verifikacija da li kompletan sistem zadovoljava funkcionalnosti kada se svi njegovi delovi integrišu u jednu celinu

Test prihvatljivosti (eng. acceptance test)

- provera da li sistem zadovoljava korisnikove zahteve i očekivanja



V-modelu

U V-modelu postoje pravila kojima se definišu preduslovi za prelazak u sledeću fazu:

- ✓ Iz faze analize zahteva se može preći u fazu specifikacije samo ako su analizirani zahtevi i ako je definisano kako će se ti zahtevi testirati tokom testa prihvatljivosti.
- ✓ U fazu dizajna sistema se može preći ako je završena faza specifikacije sistema i definisano kako će se testirati kompletan sistem.
- ✓ U fazu dizajna pojedinih modula se može preći ako je dizajnirana arhitektura sistema i definisano kako će se testirati komponente tokom integracije. U fazu kodiranja se može preći ako su dizajnirani moduli koji će se kodirati i ako je definisano kako će se ti moduli testirati.



V-modelu

Druga značajna izmena je definisanje više nivoa testiranja kojima se proveravaju različiti delovi sistema. Nivoi testiranja po V-modelu su:

- ✓ Jedinično testiranje kojim se testiraju pojedini delovi sistema (moduli, komponente, forme).
- ✓ Integraciono testiranje kojim se testira komunikacija, povezivanje i tokovi među modulima.
- ✓ Sistemsко testiranje kojim se testira sistem u celini.
- ✓ Test prihvatljivosti kojim krajnji korisnici potvrđuju da aplikacija radi upravo ono što im treba



- Postoji veliki broj različitih modela razvoja softvera kojima se uklapaju pomenute aktivnosti. Modeli razvoja softvera mogu se podeliti na sledeće grupe:
 - ✓ 1. **Fazni modeli** - modeli u kojima se aktivnosti razvoja dele po grupama koje se rade sekvencialno po fazama.
 - ✓ 2. **Iterativni modeli** - u kojima se projekat deli na manje periode (iteracije) u kojima se vrše sve aktivnosti u razvoju.
 - ✓ 3. **Agilne metode** - slične iterativnim ali se na manje formalan način kombinuju aktivnosti razvoja softvera.



FAZNI MODELI

- Fazna priroda modela vodopada zasniva se na „zdravo razumskoj“ podeli posla.
- Koji god posao je potrebno izvršti logično je da se najpre analizira šta će biti urađeno, potom da se odredi kako će to biti urađeno, a nakon izvršenja posla potrebno je proveriti da li je posao stvarno urađen kako treba.
- Model vodopada je samo primena ovog načina rada u softverskim projektima



Iterativni model

U iterativnom procesu razvoja potrebno je napraviti strategiju rešavanja problema.

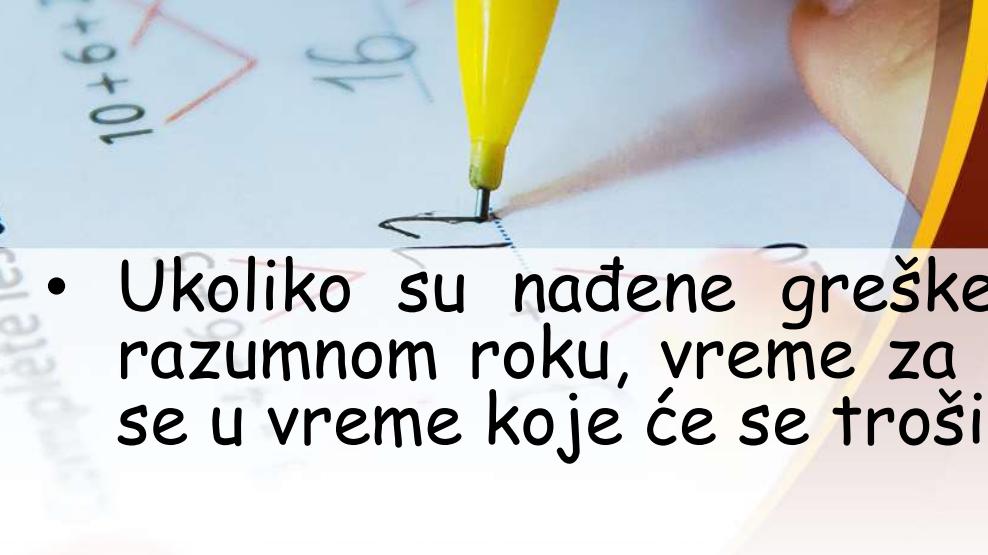
Tokom testiranja pronalaze se programske greške, koje je potrebno rešiti, što zahteva određeno programersko vreme koje se troši na uštrb vremena planiranog za razvoj funkcionalnosti.

U slučaju da je programski kod napravljen tokom iteracije suviše nestabilan, neće biti dovoljno vremena da se završe sve planirane funkcionalnosti do roka.



Iterativni model

- Pošto je kraj iteracije fiksan bez obzira na implementirane funkcionalnosti i probleme koji se dese potrebno je odlučiti šta treba raditi sa greškama koje se nađu.
- Neke od mogućih strategija su:



Iterativni model

- Ukoliko su nađene greške koje se mogu rešiti u razumnom roku, vreme za rešavanje grešaka uklapa se u vreme koje će se trošiti u okviru iteracije
- Kreiranje scenarija za sledeću iteraciju
- U slučaju da nije moguće rešiti sve greške u okviru ineracije i kompletirati planirane funkcionalnosti, iteracija se proglašava za neuspešan i svi zahtevi se vraćaju u glavni log.

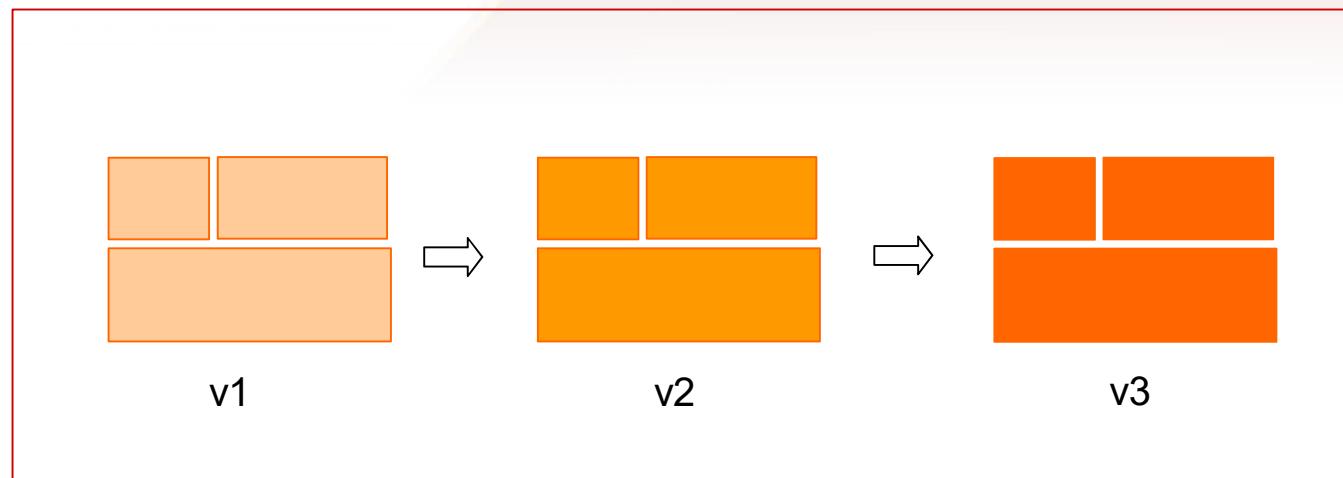


Iterativni model

- ❖ Specifikacija je precizno definisana i potpuno jasna.
- ❖ Glavni zahtevi moraju biti definisani; međutim, neke funkcionalnosti ili zahtevana poboljšanja mogu se razviti (potrebno je vreme).
- ❖ Vremenski rok je precizno definisan (i on je jako bitan faktor za distribuiranje na tržište)
- ❖ Koriste se nove tehnologija
- ❖ Često postoji slučaj da Programeri nisu dostupni u potrebnom broju.
- ❖ Postoje visok rizik da se neke funkcije softvera i ciljevi mogu promeniti u budućnost

ITERATIVNI RAZVOJ)

- ❑ podela sistema na podsisteme (faze, verzije) prema funkcijama
- ❑ u svim verzijama se isporučuje ceo sistem, uz menjanje funkcija svakog podistema





Agilne metode

Ovi modeli razvoja softvera predstavljaju novi pristup razvoju softvera, koji se bori protiv formalizacije i birokratizacije procesa razvoja softvera krutim modelima razvoja, u kojima se precizno moraju definisati sve potrebne faze i aktivnosti kako bi se kompletirao projekat

- Agilni model je pokušaj da se ukloni formalizacija procesa razvoja softvera, minimizuju prateće aktivnosti koje nisu direktno vezane za razvoj softvera i tako dobije maksimalna efikasnost



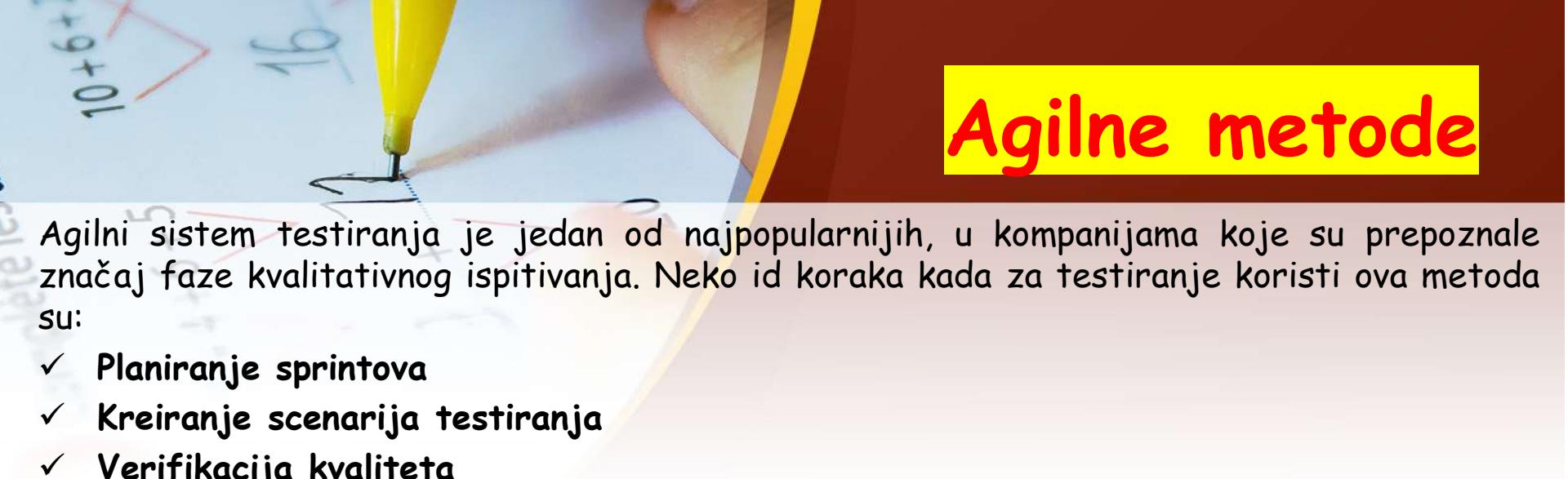
Agilne metode

- Osnovna karakteristika agilnog razvoja softvera je iterativno/spiralni razvoj u kome se projekat, umesto na dugačke faze ili verzije kao u prethodnim modelima, deli na kratke iteracije (trajanja od dve do tri nedelje), u okviru kojih se vrše sve potrebne aktivnosti analize, dizajna, kodiranja i testiranja kao i u ostalim modelima



Agilne metode

- U okviru svake iteracije, koja može trajati dan, nedelju ili par nedelja, ponavljaju se aktivnosti analize, dizajna, implementacije i testiranja gde je fokus samo na funkcionalnostima koje se implementiraju u trenutnoj iteraciji.
- Po završetku jedne iteracije kreće se u novu po istom šablonu. Ideja agilnih procesa je napuštanje formalizacije nastale modelom vodopada i fokusiranje na ljude, efikasnost i komunikaciju, a ne na procese, dokumentaciju i planove kao u formalnim metodama.



Agilne metode

Agilni sistem testiranja je jedan od najpopularnijih, u kompanijama koje su prepoznale značaj faze kvalitativnog ispitivanja. Neko id koraka kada za testiranje koristi ova metoda su:

- ✓ Planiranje sprintova
- ✓ Kreiranje scenarija testiranja
- ✓ Verifikacija kvaliteta
- ✓ Ispitivanje stabilnosti
- ✓ Ispravljanje, testiranje i isporuka softvera

Odlika agilne metodologije jeste organizacija posla u sprintovima. U praksi to izgleda tako što se veliki projekti razbiju na manje celine, i svaki od tih delova se "daju" određenom timu. Sprintovi su uglavnom vremenski ograničeni i pre samog početka se definiše cilj koji se u tom roku mora izvršiti.

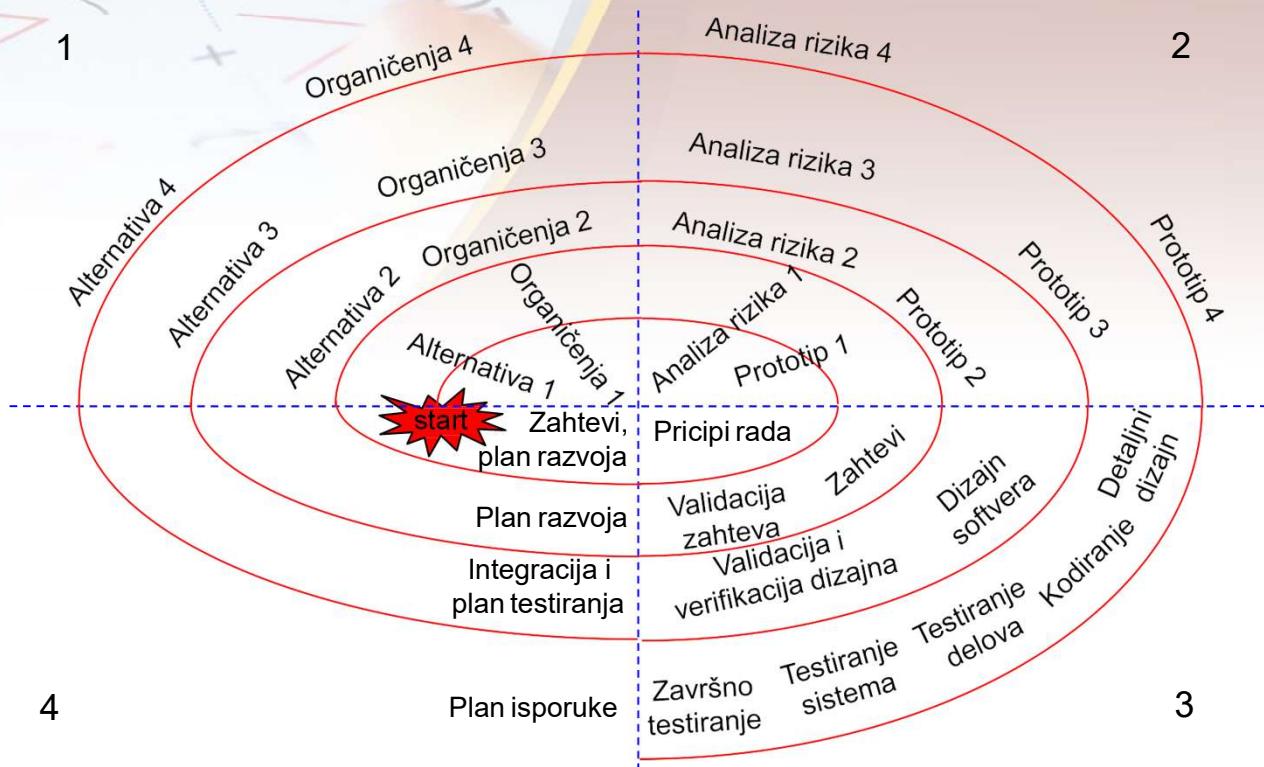


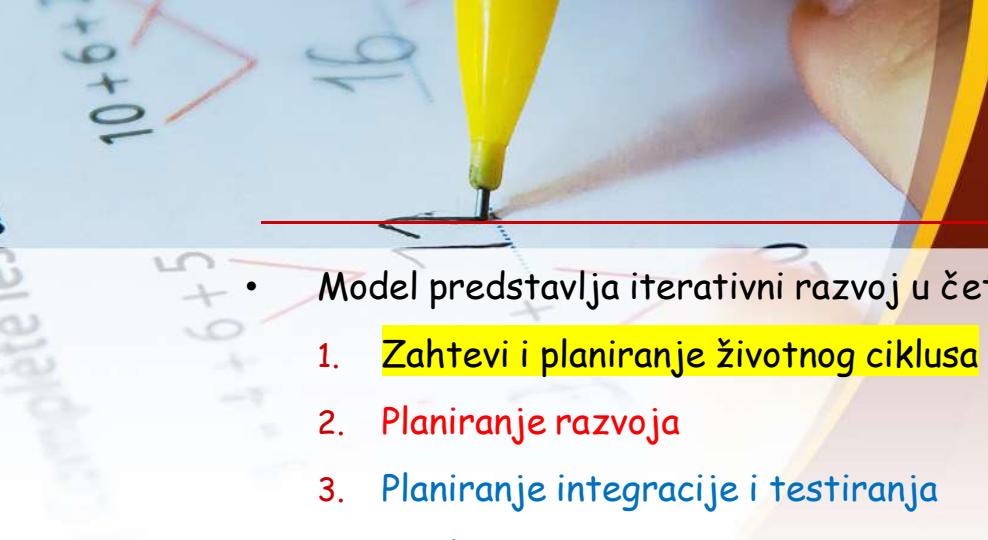
Agilne metode

U agilnim metodama, korisnik je deo razvojnog tima i odgovoran je za donošenje odluka o prihvatljivosti sistema.

- ✓ Testovi su definisani sa strane korisnika i integrисани su sa drugim testovima na taj način da se izvršavaju automatski kada se naprave promene.
- ✓ Ne postoji odvojen proces testiranja prihvatljivosti.
- ✓ Osnovna dilema je da li je uključeni korisnik radi po pravilima ili ne radi.

SPIRALNI MODEL





SPIRALNI MODEL

- Model predstavlja iterativni razvoj u četiri koraka:
 1. Zahtevi i planiranje životnog ciklusa
 2. Planiranje razvoja
 3. Planiranje integracije i testiranja
 4. Implementacija
- Svaka iteracija obuhvata pun krug i prolazi kroz četiri kvadranta:

kvadrant 1: određivanje ciljeva, alternativa i ograničenja	kvadrant 2: evaluacija alternativa, identifikacija i procena rizika
kvadrant 4: planiranje sledeće iteracije	kvadrant 3: razvoj i verifikacija putem testiranja



TESTIRANJE SOFTVERA

Proces testiranja mora se odvijati tokom svih faza životnog ciklusa.

Testeri će izvoditi testove na osnovu planova ispitivanja i pripremljenih test slučajeva.

Za ovaj process moraju se realizovati standardizovani propisi i procedure koje definišu nacin rada i aktivnosti test tima, kao i svih ucesnika u razvoju.



Kako poboljšati sam proces testiranje softvera?

- Poboljšanja procesa testiranja je konstantan proces zajedno sa svim drugim elementima razvoja softvera. Angažuju se sve veći resursi kako velikih tako i malih kompanija u proces testiranja i njegovo unapredjenje. Postoji više aspekata koji doprinose poboljšanju procesa testiranja:
 - obuka kadrova za proces testiranja
 - automatizacija procesa testiranja

Kako poboljšati sam proces testiranje softvera?

- automatizacija procesa testiranja
- razvoj novih alata
- razvoj novih modela testiranja
- integracije procesa merenja parametara efikasnosti i efektivnosti procesa testiranja
- analize slabih i jakih strana postojećeg procesa testiranja
- identifikacije rizika i njihovih posledica na uspeh procesa razvoja



Šta danas predstavlja testiranje softvera?

Testiranje predstavlja važan deo životnog ciklusa razvoja softvera.

Softver se implementira prema zahtevima korisnika sa ciljem rešavanja realnog problema ili kreiranja potrebne funkcionalnosti.

Nakon implementacije, softver može u manjoj ili većoj meri odgovarati zahtevima.

Svako ponašanje softvera koje se ne slaže sa zahtevima predstavlja grešku koju je poželjno detektovati i eliminisati.



- **Validacija i verifikacija** su izrazi koji se najčešće povezuju sa testiranjem programa. Verifikacija predstavlja proveru ili testiranje objekata (ili programa) u cilju utvrđivanja koliko odgovaraju predviđenim karakteristikama.
- **Verifikacija** obuhvata analize, inspekciju, isprobavanje, a jedan od njenih oblika je i testiranje programa.



TESTIRANJE SOTVERA

Izrazi "verifikaciju" i "validacija" se vrlo često koristi u stručnoj literaturi i odnose se na kvalitet i analize bilo kog softvera.

Verifikacija programa se može osloniti na tehnike automatskog dokazivanja teoreme.

Ove tehnike imaju principe deduktivnog zaključivanja, iste one koje koriste i programeri prilikom samog konstruisanja programa.

Testiranje softvera se posmatra kao validacioni proces.



Zašto ne bi koristili iste principe u sistemu za automatsku sintezu, koji može da konstruiše program umesto da samo dokazuje njegovu ispravnost?

Svakako, konstruisanje programa zahteva više originalnosti i kreativnosti nego dokazivanje njegove ispravnosti, ali oba posla zahtevaju isti način razmišljanja.

Testiranje nove verzije softvera koja se pušta u produkciju

Kada se nova aplikacija pušta u produkciju može se reći da je to samo početak nove faze u životnom ciklusu softvera (proizvoda).

Tu se posebno ističe termin Održavanje softvera (proizvoda).

Naravno tu se mora voditi računa o novim izmenama u svakom softveru (proizvoda).

Postoje :

- Implementacija novih zahteva
- Prilagođavanje softvera novom okruženju (ili zadatim uslovima)
- Ispravljanje uočenih grešaka (bagova)

Ograničavajući faktor je vremenesci okvir jer pri svakoj izmeni softvera je potrebno da se obavi kompletan ciklus testiranja

Kada se govoro o Novim verzijama softvera one mogu biti:

- neplanirane, izazvane ispravkama nedostataka (*hot fixes*)
 - planirano izdavanje nove verzije
- Prestanak korišćenja softvera zahteva testiranje arhiviranja i migracije podataka



Jedno od najbitnijih stavki - u testiranju softvera.

U literaturu se navodi kao **završna faza procesa testiranja**.

Istalacioni testovi se radi zajedno sa **korisnicima**. (Ne može se testirati bez prisustva samog korisnika softvera.)

Ovaj tip testiranja se radi kako bi se osiguralo da su sve mogućnosti softvera i opcije koje on pruža pravilno instalirani.

Treba proveriti da li su sve komponente aplikacija pravilno instalirane.

Poslednja provera - Da li je sve u redu sa dokumentacijom.

Sugestija:

testira instalacije softvera na različitim sistemima i u različitim situacijama (npr. prekid napajanja ili nedovoljno prostora na disku).

Uvek treba pokušati instalaciju na veoma slabim računarima koji ima manje memorije (non-compliant),



Preporuka a i sugestija:

Ako se želi zameniti postojeći sistem, instalacija novog sistema se vrši u paralelnom modu.

Paralelno testiranje

Paralelno testiranje se koristi tokom razvoja, kada jedna verzija softvera zamenjuje drugu ili kada novi sistem treba da zameni stari

To znači da će oba sistema raditi istovremeno u jednom periodu.

Programeri moraju da svako dnevno prate rad korisnika na ovakovom sistemu i kad se steknu uslovi (kada se postigne komforan i olakšan rad korisnika novog sistema), stari sistem se tek tada eliminiše.

Mnoge kompanije postavaljaju nekoliko servera (multi - server) na jedan sistem, koji su nezavisni jedan od drugog.



Prilikom instaliranja, sistem se konfiguriše u skladu sa okruženjem.
Ukoliko je potrebno, sistem se povezuje sa spoljnim uređajima i sa njima
uspostavlja komunikaciju

Jedan dobar način za instaliranje sistema pre puštanja u takozvanu
produkciju je probna instalacija na jednom odabranom serveru, posle
provere dužine trajanja, instalirati ga na drugom severu.

Sve mora da se pravilno dokumentuje i da se zapiše u dnevnik testiranja
softvera.

Za to je potrebno vreme i sve što se radi se mora zapisivati u dnevnik
testiranja softvera.

VRSTE i Metode testiranja softvera

Zavisno od toga na koje se rizike i karakteristike proizvoda test tim mora fokusirati, može se definisati više vrsta testiranja.

- Funkcionalno testiranje softvera
- Ne funkcionalno testiranje softvera





Funkcionalno testiranje softvera

Funkcionalno testiranje softvera obuhvata proveru da li je softver razvijen u skladu sa funkcionalnom specifikacijom.

Funkcionalno testiranje (funkcionalnost aplikacije) obuhvata testiranje na različitim nivoima, testove prihvatljivosti,

Formiraju se detaljni test scenariji i koriste se kao osnova za čitav proces funkcionalnog testiranja - najčešće BlackBox metodologijom.

Rezultat testiranja se dostavlja u formi izveštaja koji tačno navodi koje funkcije ne rade u skladu sa aplikativnim zahtevom.



Tipovi nefunkcionalnog testiranja

- **Test opterećenja (load test)**
 - merenje ponašanja sistema u uslovima povećanog opterećenja (npr. broja korisnika ili količine pozvanih operacija) kako bi se utvrdilo koji stepen opterećenja sistem može da izdrži
- **Test performanse (performance test)**
 - merenje performanse softvera (npr. brzina obrade ili vreme odziva) za određenu fukcionalnost (obično se meri zavisno od opterećenja)
- **Test velike količine podataka (volume test)**
 - testiranje ponašanja sistema kada upravlja velikom količinom podataka



TESTIRANJE OPTEREĆENJA

- To je proces testiranja softvera u kom se performanse ispituju pod određenim opterećenjem.
- Softver za testiranje opterećenja prvenstveno testira robusnost ili dostupnost softvera u normalnim i ekstremnim radnim uslovima.



TESTIRANJE OPTEREĆENJA

Postoji puno alata koji mere performanse u uslovima opterećenja:

- Jmeter <https://jmeter.apache.org/>

Podržava više platformi. Napisan je u Scala, podržava veliki broj veb protokole.

- Gatling <https://gatling.io/open-source/>

Napisan je u Scala, po završetku testiranja generiše puno izveštaja.



Locust <https://locust.io/>

Prikazuje rezultate testiranja u realnom vremenu.

Taurus <https://gettaurus.org/>

Pomoći njega testirate kod u razvoju.



Testiranje performansi

Deo testiranja proizvoda može obuhvatiti testiranje bitnih karakteristika sistema, kao što su performanse i pouzdanost.

- ✓ Testovi treba da reflektuju profil upotrebe sistema.
- ✓ Testovi performansi obično obuhvataju planiranje serija testova kod kojih se postepeno povećava opterećenje sve dok performanse sistema ne postanu neprihvatljive.

"Stres" testiranje je oblik testiranja performansi gde se namerno testira sistem pod maksimalnim opterećenjem kako bi namerno doveli do otkaza.



- Kod testiranje performansi posmatraju se:
- Brzina / proverava se brzina odziva softvera
- Skalabilnost / testira se maksimalno korisničko opterećenje pod kojim softver može da rad
- Stabilnost / proverava se stabilnost softvera pod različitim opterećenjima



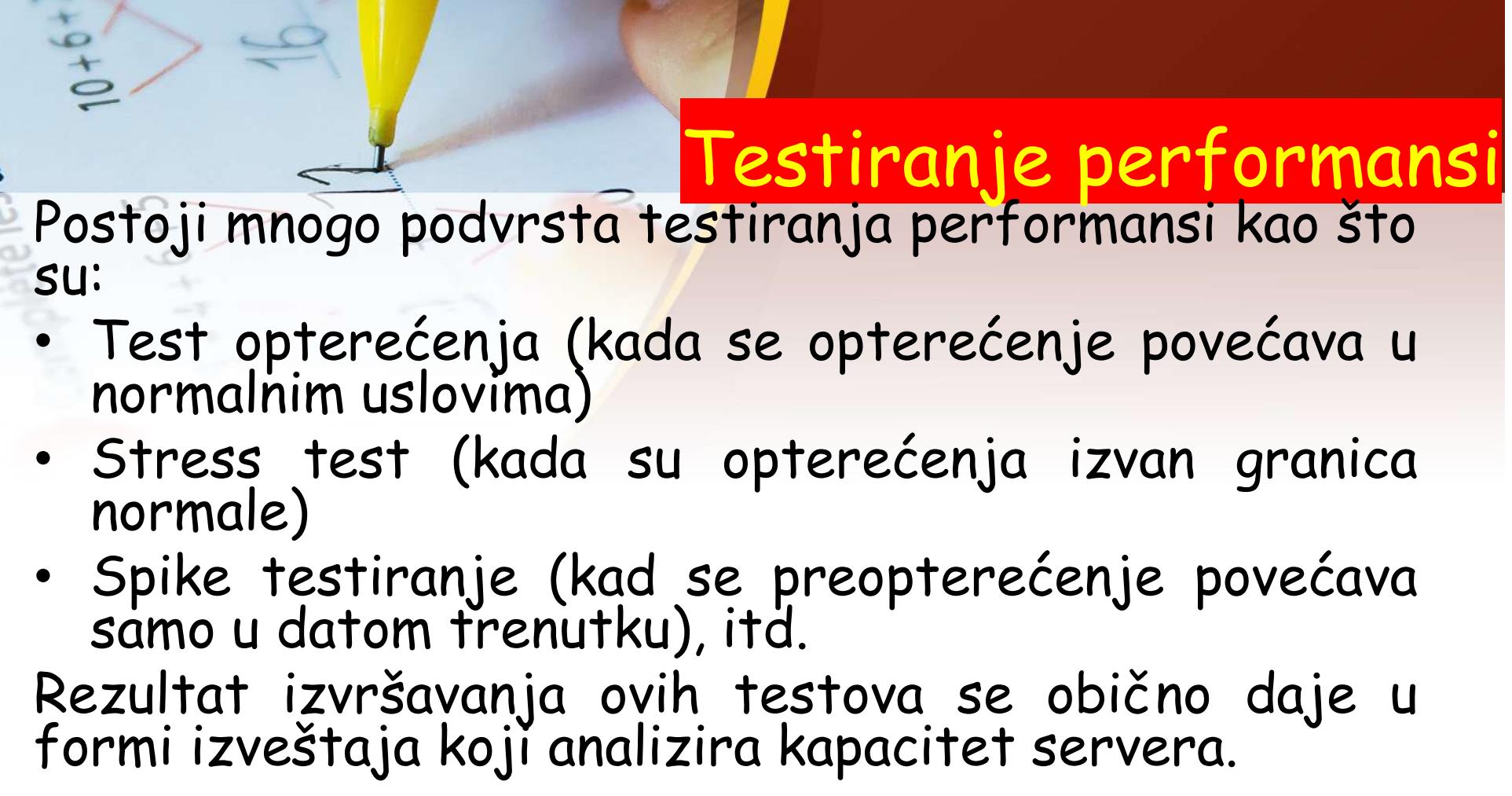
Testiranje performansi

- Testiranje performansi je proces utvrđivanja brzine, odziva kao i stabilnosti kako same aplikacije tako i sistema na kome se izvršava.
- Kada se gleda na nivo operacija razlika od jedne milisekunde je beznačajna, ali ako se ta operacija ponavlja recimo hiljda puta to već predstavlja ozbiljnu razliku.



Testiranje performansi

- Performanse obuhvataju vreme odziva softvera, pouzdanost, upotreba resursa i skalabilnost.
- Testiranje performansi je tip testiranja koji verifikuje da se sistemski softver ponaša na odgovarajući način pod nekim očekivanim opterećenjem.
- Cilj testiranja performansi nije pronađak novih defekata, već da se eliminišu potencijalni problemi koji utiču na performanse sistema.



Testiranje performansi

Postoji mnogo podvrsta testiranja performansi kao što su:

- Test opterećenja (kada se opterećenje povećava u normalnim uslovima)
- Stress test (kada su opterećenja izvan granica normale)
- Spike testiranje (kad se preopterećenje povećava samo u datom trenutku), itd.

Rezultat izvršavanja ovih testova se obično daje u formi izveštaja koji analizira kapacitet servera.



Test prihvatljivosti

Test prihvatljivosti je formalni opis ponašanja softverskog proizvoda.

Test prihvatanje generalno ima binarni rezultat, da li proizvod radi ili ne.

Testiranje prihvatljivosti ima mnogo prednosti, ali i nedostataka (skoro nikada se ne testira softver posle redovnog korišćenja, a i ne testiraju se sve funkcije interfejsa), tako da ga treba dopuniti nekom od navedenih metoda evaluacije



Testiranje performansi

Najčešće obuhvata iskorišćenje procesorskih resursa, protok podataka i vreme odziva.

Tipični resursi koji se proveravaju su: propusni opseg, brzina procesora, iskorišćenje memorije, zauzetost prostora na disku.

U real time sistemima i ugrađenim sistemima, softver koji pruža potrebne funkcije ponekad nije prihvatljiv u skladu sa performansama.

Testiranje performansi dizajnirano je za izvođenje run-time testa performansi u softverskom okruženju jednog integrisanog sistema.

Tokom testiranja performansi, izvršavaju se testovi konfiguracije.

Testiranje performansi se pojavljuje u svim procesima testiranja.

Testiranja performansi i testiranja opterećenja

Razlika između testiranja performansi i testiranja opterećenja je u tome što testiranje performansi utvrđuje jesu li performanse Sistema normalne, dok testiranje opterećenja proverava radni kapacitet softverske aplikacije.



Peak testiranje

Unapred definisani broj korisnika pristupa aplikaciji i koristi je neki vremenski interval, i nakon tog isteka intervala, aplikaciji pristupaju novi korisnici koji posle nekog vremenskog intervala više ne koriste aplikaciju.

Tako imate nove stres tačke sa velikim brojem korisnika.

Ideja je da se vidi da li je zadata aplikacija (koja je već opterećena sa velikim brojem korisnika) u stanju da na neki način sa novim dodatnim korisnicima može da kvaliteno radi.

Istraživačko testiranje softvera

(Exploratory)

- Istraživačko testiranje je softverska tehnika testiranja koja ne koristi bilo koji određeni dizajn, plan ili pristup ispitivanja.
- To je tehnika testiranja softvera u kojoj ispitivači istražuju i identificiraju različite načine ocjenjivanja i poboljšanja kvalitete softvera.



Istraživačko testiranje softvera

Istraživačko testiranje (Exploratory testing), gde bez unapred definisanog postupka testeri koristeći svoju intuiciju i iskustvo u pronalaženju grešaka.

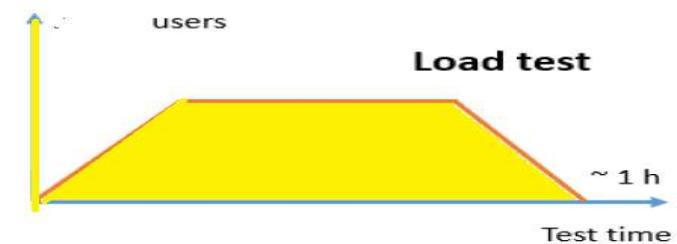
Ideja istraživačkog testiranja je da testeri sami tokom testiranja aplikacije pronađe alternativne scenarije za testiranje koji ne mogu biti unapred planirani, na ovaj način se podstiče kreativnost testera.

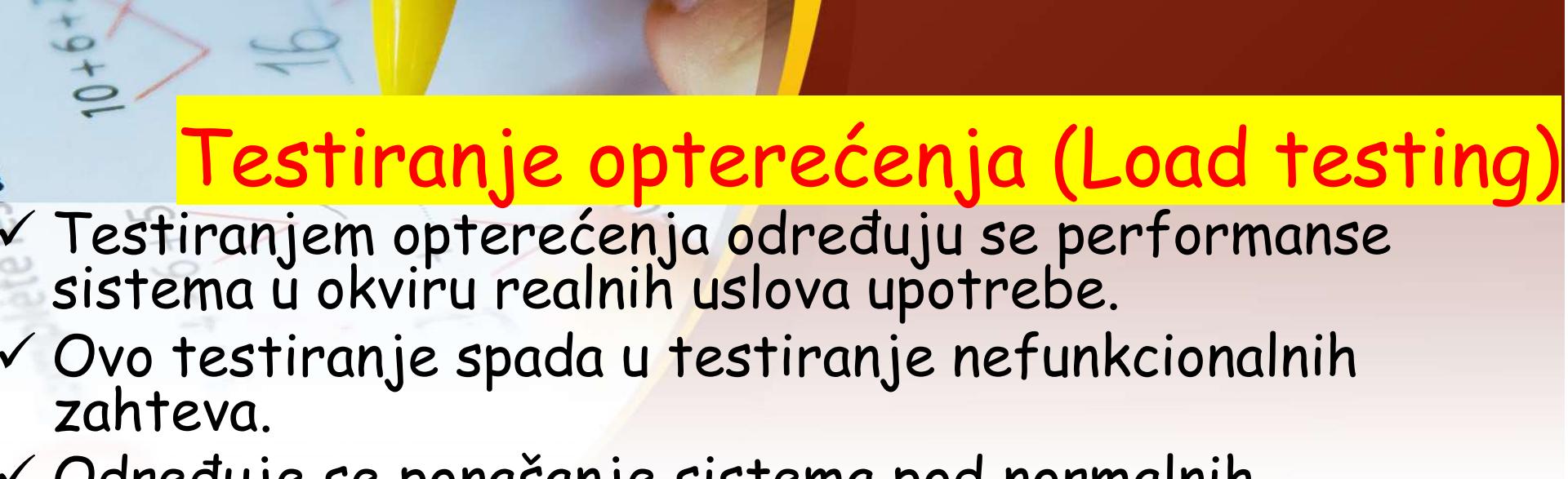
Load testiranje



Odredi se maksimalni definisani broj korisnika koji mogu da rade na softveru (aplikaciji).

- Najpre jedan korisnik koristi aplikaciju, pa drugi korisnik, pa treći korisnik i tako dalje dok ne dodjete do maksimalnog definisanog broja korisnika.
- Zatim jedan po jedan korisnik napušta aplikaciju.
- Ideja je da se vidi kako taj maksimalni definisani broj korisnika može da pristupi aplikaciji u isto vreme, odnosno šta će biti sa aplikacijom kada njoj pristupa taj maksimalni definisani broj korisnika u isto vreme.





Testiranje opterećenja (Load testing)

- ✓ Testiranjem opterećenja određuju se performanse sistema u okviru realnih uslova upotrebe.
- ✓ Ovo testiranje spada u testiranje nefunkcionalnih zahteva.
- ✓ Određuje se ponašanje sistema pod normalnih opterećenjima, kao i pod najvećim očekivanim opterećenjem.
- ✓ Identificuje se maksimalni operativni kapacitet, uska grla ako postoje i koja komponenta izaziva opadanje performansi

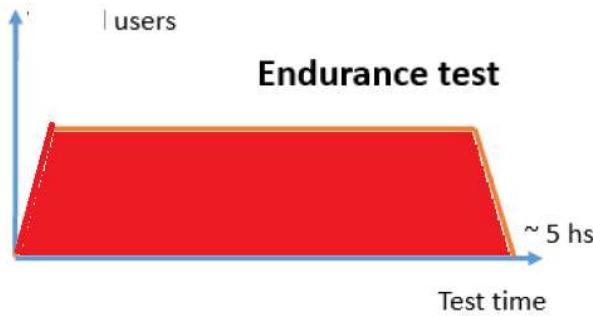
Testiranje opterećenja -Load testing

- U slučaju kada se opterećenje podigne iznad razumnog nivoa, test opterećenja postaje stres test.
- Ovaj oblik testiranja najčešće se primjenjuje za klijent/server web sisteme.

Testiranje opterećenja (Load testing)

Test opterećenja određuje:

- ✓ Maksimalni operativni kapacitet sistema
- ✓ Da li postojeća infrastruktura zadovoljava potrebe sistema
- ✓ Održivost sistema u slučaju povećanog korisničkog opterećenja (eng. peak load)
- ✓ Broj konkurenčnih korisnika koje sistem može da izdrži

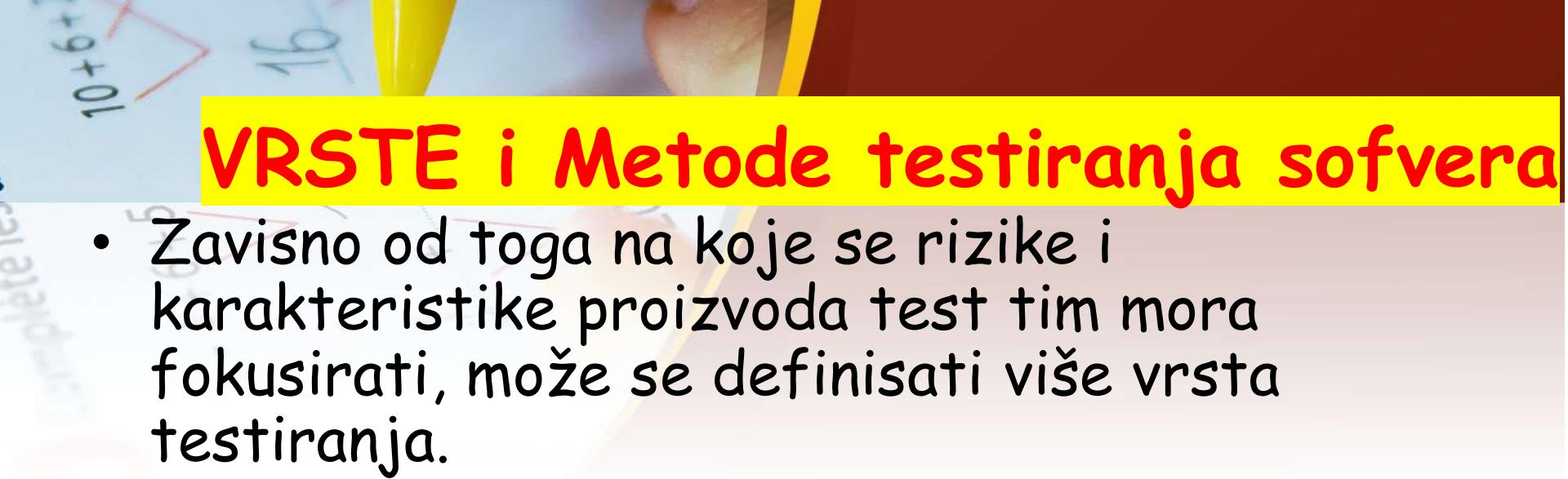


Endurance testiranje softvera

Ova vrste testiranja softvera vrlo je slična *load* testiranju. Razika je u tome da je vreme simultanog korištenja aplikacije od strane svih korisnika dosta duže nego kod *load* testiranja (može da bude dosta dugo od nekoliko dana ili nekoliko sati u zavisnosti za šta je namenjena aplikacija).

Ideja je da se testiraju da li svi ti korisnici mogu da pristupe aplikacije ili samo zadati unapred broj korisnika.

Kako se aplikacija ponaša sa velikim brojem korisnika (pre svega njena funkcionalnost)



VRSTE i Metode testiranja sofvera

- Zavisno od toga na koje se rizike i karakteristike proizvoda test tim mora fokusirati, može se definisati više vrsta testiranja.
- Testiranje korisničkog interfejsa
- Jedinično testiranje
- Integraciono testiranje
- Sistemsko testiranje



- Zašto je bitno praviti izveštaje o greškama?
- PROBLEM AKO SE GREŠKA NE MOŽE REPRODUKOVATI,
- KAD SE DESILA I POD KOJIM USLOVIMA
 - DNEVNIK TESTIRANJA





U osnovi svih definicija testiranja programa je težnja da se odgovori na pitanje:

Da li se program ponaša onako kako je zahtevano?

U bilo kom upotrebljivom softveru u praksi postoji beskonačan broj mogućih testova koje je praktično nemoguće sve izvesti, te je nemoguće u određenom vremenskom periodu, sa ograničenim resursima izvršiti totalno testiranje koje bi otkrilo sve greške u softveru.



Razvoj i implementacija softvera je zahtevan i dugotrajan proces.

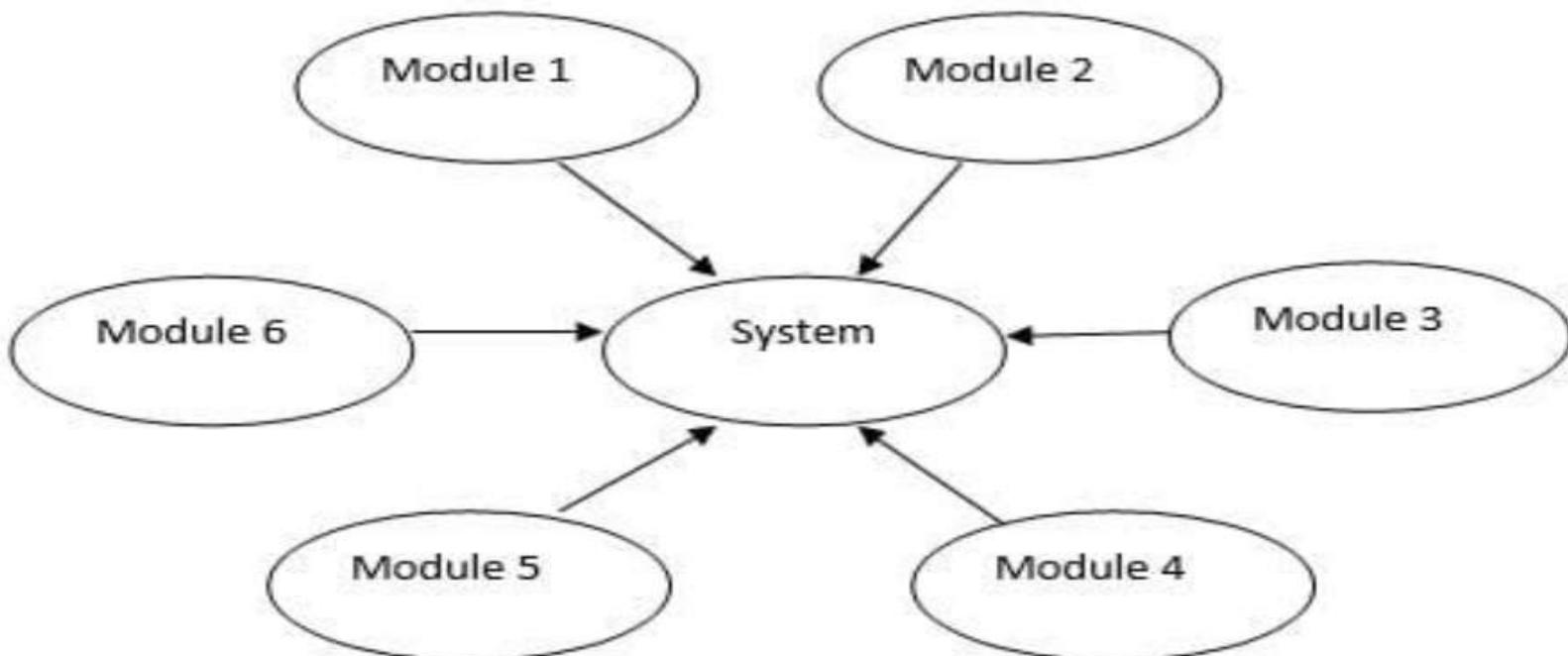
Podrazumeva korištenje raznih tehnologija i metoda.

Razvoj softvera svakim danom postaje sve kompleksniji.

Softver mora biti isporučen na vreme sa što većim kvalitetom i sa što manje grešaka.

Kao najvažniji ciljevi u Testiranju softvera ističu se prevencija, otkrivanje grešaka, zadovoljstvo krajnjeg korisnika i kvalitet samog softvera.

TESTIRANJE SOFTVERA





PROCES OTKLANJANJA BUGOVA

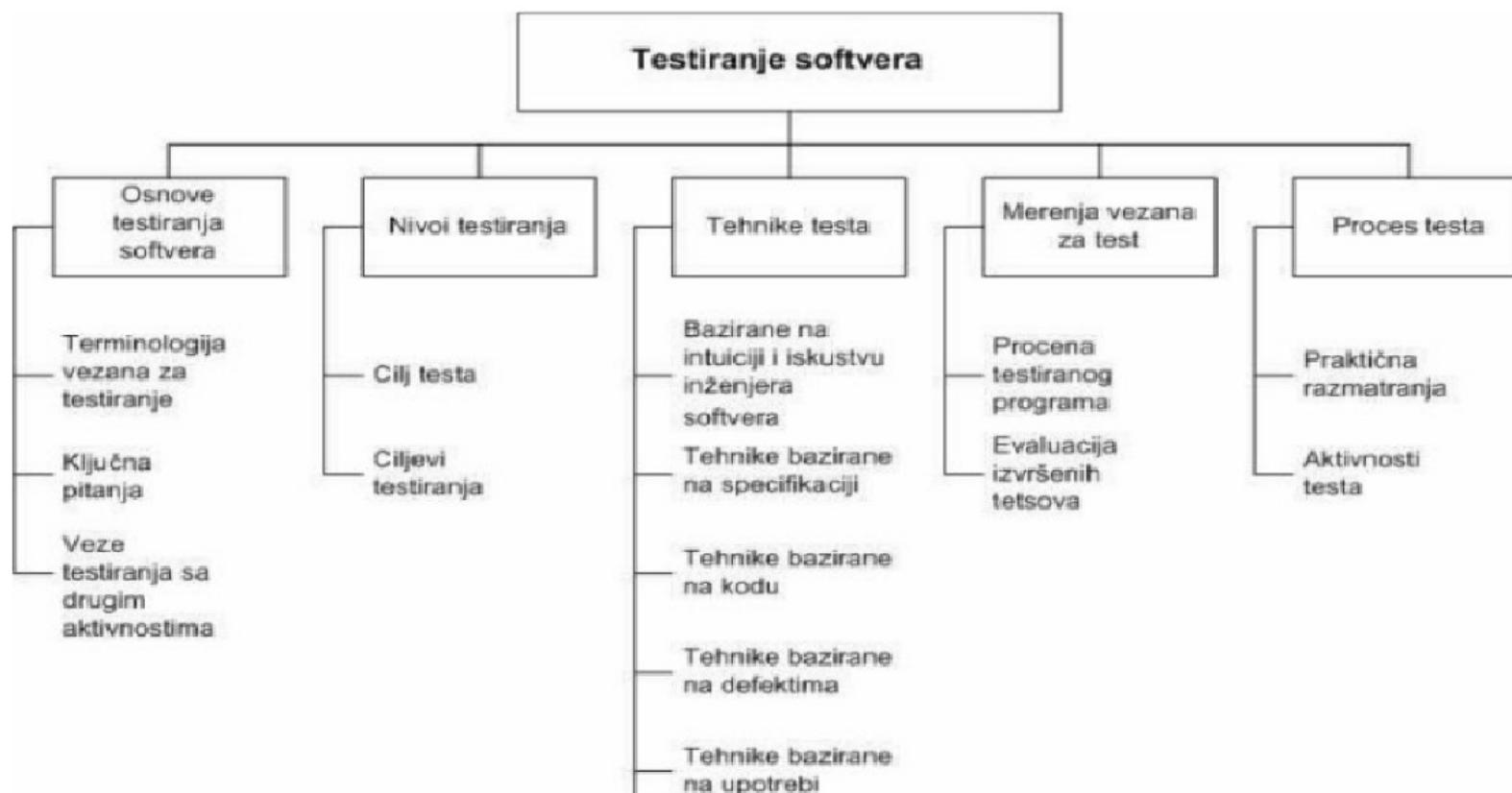
- Identifikacija bugova
- Razumevanje bugova
- Lociranje bugova
- Ispravljanje bugova
- Ponovo testiranje svega
- VREMENSKI OKVIR + RESURSI
- TESTIRANJE I SAMO TESTIRANJE



Testiranje sistema obuhvata različite procedure, metodologije i strategije koje nude različite pristupe. Efikasno testiranje softvera doprinosi isporuci kvalitetnog softverskog proizvoda koji zadovoljava potrebe, očekivanja i zahteve korisnika. Ukoliko radi loše, to vodi visokim troškovima održavanja i nezadovoljstvu korisnika.



TESTIRANJE SOFTVERA





Proces testiranja

- ✓ Prikupljanje nefunkcionalnih zahteva
- ✓ Izgradnja okruženja za testiranje performansi
- ✓ Programiranje slučajeva korišćenja
- ✓ Izgradnja scenarija testiranja performansi
Sprovođenje testiranja i analiza rezultata
- ✓ Naknadna analiza rezultata, izveštavanje i sama dokumentacija



- Strategija testiranja- koja pokazuje način, tj.metod testiranja-koje tipove i koju količinu testova treba upotrebiti da bi se na najbolji način otkrile greške koje su skrivene u softveru.
- Plan testiranja- koji sadrži konkretne zadatke koji će omogućiti ostvarenje strategije testiranja.
- Slučajevi testiranja - koji su pripremljeni u formi detaljnih primera i koji se koriste da bi se proverilo da li softver odgovara zahtevima.
- Podaci za testiranje- koji se sastoje i od ulaznih podataka i baze podataka, koji se koriste dok se izvršavaju test slučajevi,
- Okruženje testiranja- softversko i hardversko okrujenje (operativni sistem i druga softverska rešenja)u kome se celo testiranje obavlja.



IDEALNO TESTIRANJE SOFTVERA

U idealnom slučaju!!!!

- ✓ Da koristimo što manje resursa, a da pronadjemo što više bagova (uz što manje vremena)
- ✓ Da se naprave mali broj testova a da otkriju sve bagove
- ✓ Da ne koramo da izvršavamo testove svakodnevno
- ✓ Da svaki propušteni test daje reprezentativne rezultate
- ✓ da su testovi povezani sa Git-om ili nekim drugim alatom za verzionisanje
-

OVO ne postoji, samo testirati, testirati, testirati i samo testirati.



Testiranje razvoja

Sprovodi niz sinhronizovanih strategija za otkrivanje i sprečavanje defekata.

Uključuje staticku analizu koda, pregledе peer kodova i analizu metrika.

Cilj je smanjiti rizik i same troškove gde god je to moguće u svakom delu razvoja sofvera.

Testiranje i samo testiranje!!!!



Kvalitet softvera

Kvalitet softvera se vrši primenom pravila, standarda, i na kraju nekih zakonskih obaveza (o kojima se mora voditi posebno računa)

- Primeri dobre prakse
 - široko prihvaćene metode rada u određenoj oblasti
- Kompanijski standardi
 - interne odredbe, procedure i smernice
 - npr. dokument o upravljanju kvalitetom, šabloni za test dokumentaciju, konvencije programskog koda



Kvalitet softvera

- Standardi upravljanja kvalitetom
 - generalni standardi koji se odnose na različite industrijske grane
 - specificiraju minimalne zahteve kvaliteta
 - ne bave se metodama implementacije kojima se kvalitet dostiže
- Standardi za specifičnu industrijsku granu
 - svaka industrija ima svoje standarde kvaliteta
- Standardi za testiranje softvera
 - odnose se na bilo koji tip softvera
 - bave se dokumentacijom i procesom testiranja

Kvalitet softvera





❖ Master test plan

- sadrži generalni plan testiranja za ceo projekat
- za svaku fazu životnog ciklusa testiranja predviđa konkretne aktivnosti, resurse i učesnike

❖ Level test plan

- plan testiranja za određeni nivo testiranja
- tako imamo *component test plan*, *integration test plan*, *system test plan*, ...
- za svaki nivo testiranja se definiše
 - pristup, resursi i raspored aktivnosti
 - šta će biti testirano
 - konkretne aktivnosti kojima će se vršiti testiranje
 - odgovorne osobe za svaku aktivnost



TESTIRANJE SOFTVERA

Prema QA TestLab-u postoji 7 vrsta testiranja:

- Skeniranje ranjivost (engl. Vulnerability scanning)
- Sigurnosno skeniranje (engl. Security scanning)
- Penetracijski testovi (engl. Penetration testing)
- Procena rizika (engl. Risk assessment)
- Revizija ranjivosti (engl. Security Auditing)
- Procena sigurnosnog držanja (engl. Posture Assessment)
- etičko hakiranje (engl. Ethical hacking)



TESTIRANJE SOFTVERA

Kada se vrši pravljenje nekog softvera svaka kompanija mora da testira taj softver.

Ali, i pored toga taj softver ima određene mane ili bagove. Početna greška može da naraste tokom projektovanja a potom da se višestrukoj poveća tokom kodiranja.

Kada se testiranjem programa utvrди da se on ne ponaša kao što se očekuje, pristupa se analiziranju i lociranju bagova. Detaljan postupak identifikacije bagova omogućuje se metodičkim pristupom testiranja programa.

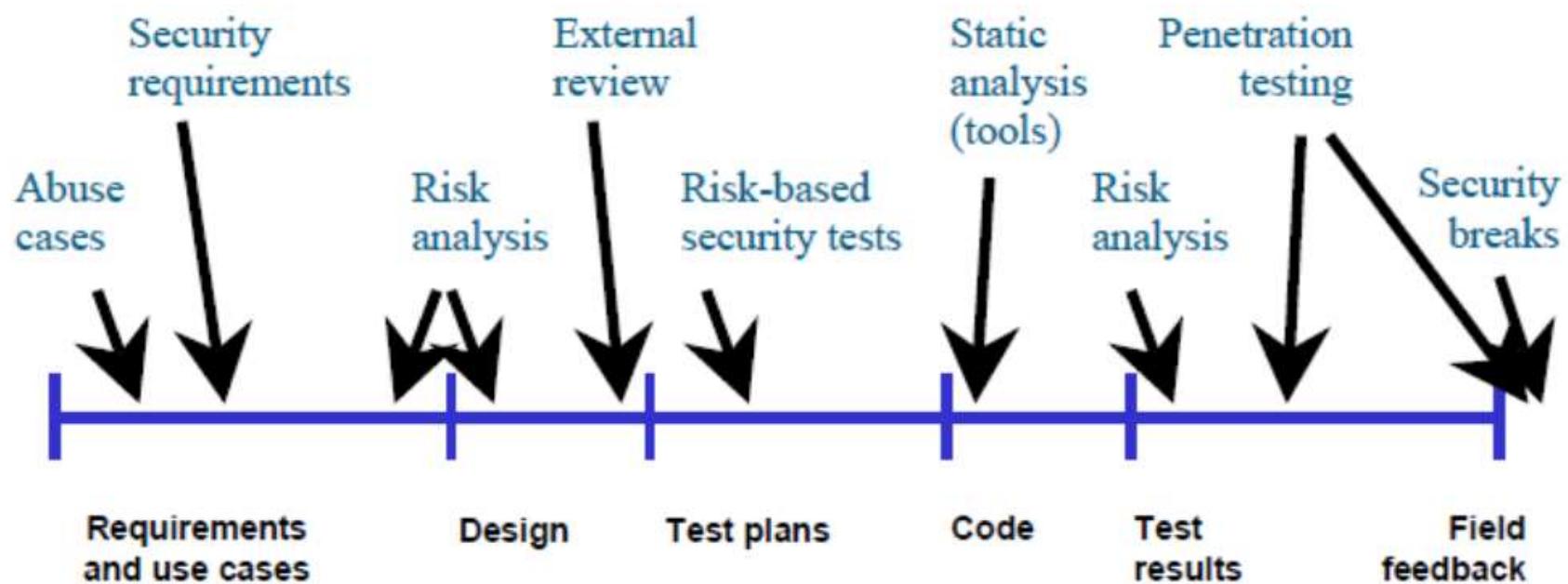
Poznato je takođe da nikakvo testiranje ne može da otkrije sve bagove.



Test evaluation

- Evaluacija testova (Test evaluation) predstavlja kreiranje izveštaja kojim se opisuje šta je testirano i potvrđivanje da je implementirani sistem spreman za korišćenje u skladu sa korisničkim zahtevima.
- Proces evaluacija uključuje i pregled rezultata dobijenih analizom izlaza test slučajeva.

TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA

U vezi sa testiranjem programa često se rade statičke analize (istražuju se osnovni programi, pri tome traže se osnovni problemi i prikupljaju podaci bez izvršavanja programa) i dinamičke analize (istražuje se ponašanje programa).

Postoji mnogo pristupa softverskom testiranju, koji teže kontroli kvaliteta procesa testiranja da bi obezbedili kvalitetne informacije o kvalitetu testiranja softvera.

Iako je većina istraživanih testova koncentrisana na pronalaženje efektivne tehnike testiranja, takođe je veoma bitno napraviti softver koji se lako testira.

Softver je lak za testiranje ukoliko greške uzrokuju rušenje programa, pa prema tome te greške će biti uočavane tokom samog testiranja.



Kada se govori o testiranju softvera neki preduslovi moraju biti ispunjeni da bi se uopšte moglo izvršiti testiranje.

- Nije dobro testirati softver koji nije spreman (resursi)
- Kriterijumi početka testiranja
 - Test alati su spremni za korišćenje
 - Test okruženje mora biti spremno
 - Test objekti postoje i nalaze se u takvom stanju razvoja da ih ima smisla testirati
 - Test podaci (ulazni) su dostupni



Kriterijum za završetak testiranja

Kada treba prestati sa testiranjem?

Dva su odgovora na ovo pitanje:

- Nikad. Korisnik nastavlja da testira nakon preuzimanja programa.
- Kada se ostane bez resursa za dalje testiranje (vremenski okvir /budžet)

Potrebno vreme testiranja bi mogao biti relevantan parametar

na osnovu kojega bi se donela odluka kada prestati sa testiranjem softvera.



TESTIRANJE SOFTVERA

Proizvođači softvera bi želeli da predvide broj grešaka u softverskim sistemima pre nego što ih primene, kako bi mogli da procene kvalitet kupljenog proizvoda i teškoće koje se javljaju u samom procesu održavanja.

Testiranje softvera je veoma **kompleksan proces**.

Cilj testiranje je da se ustanovi da li se softver ponaša na način kako se to od njega očekuje.

Prema tome, primarni cilj testiranja softvera je otkrivanje gešaka.



TESTIRANJE SOFTVERA

Aktivnost testiranja pokazuje da li je dati softver usklađen sa specifikacijom.

Specifikacija je ključna stvar u testiranju.

Stoga, kako se sakupljaju rezultati testiranja, pojavljuju se dokazi o nivou kvaliteta i pouzdanosti programa.

Ako testiranje često otkriva važne greške, kvalitet i pouzdanost programa mogu se smatrati nedovoljnim i zahteva se dalje testiranje. Sa druge strane ako su greške minorne i luke za ispravljanje, onda je nivo kvaliteta i pouzdanosti programa prihvatljiv.

Testiranje nikada ne može da definitivno kaže da li je program ispravan, jer neotkrivene greške mogu ostati u programu čak i posle najobimnijeg testiranja.



Stoga je uobičajeno gledište po kome je uspešan test onaj koji ne otkrije nijednu grešku netačno, što naglašavaju navedeni ciljevi testiranja a to su:

- ✓ Testiranje je proces izvršavanja programa u namjeri da se nađu greške
- ✓ Dobra test stavka ima visoku mogućnost pokrivanja greške
- ✓ Uspešan test otkriva dotle neotkrivenu grešku.



TESTIRANJE SOFTVERA

Uspeh jednog skupa test podataka je jednak uspešnom izvršenju iscrpnog testiranja programa. Jedno od velikih pitanja koje se javlja kod testiranja programa jeste reprodukcija greške (testeri otkrivaju greške a programeri otklanjaju bagove).

Očigledno je da mora postojati koordinacija između testera i programera.

Reprodukcijska greška je slučaj kada bi bilo najbolje da se problematicni test izvede ponovo i da znamo kad, i tačno gde se greška desila.

Dakle idealnog testa i idealnog proizvoda nema.



U poslednje vreme ulaze se dosta napora da se konstruktivni pristup primeni parcijalno, tj. na određene delove programa, ako već ne može u celini.

Jedan od pristupa je matematički dokaz korektnosti zasnovan na radovima Dijkstre, Hoare-a, Floyda.

Izvođenja u formalnim teorijama predstavljaju opšti okvir za razvoj deduktivnih metoda utvrđivanja korektnosti programa. Iz tog okvira izviru dve osnovne metode (pobijanje pridružene predikatske formule, odnosno korišćenje pravila programske logike) kao i njihove modifikacije. Rad sa formulom koja je pridružena datom programu podrazumeva prisustvo dodatnih aksioma, bez kojih pobijanje nije ostvarljivo.



TESTIRANJE SOFTVERA

Druga tehnika dokazivanja je zasnovana na pojmu simboličnog izvršenja. Programske linije se obrađuju istim redom kao što će biti izvršavane, za razliku od Dijkstrinog redosleda "unatrag".

Obe tehnike su jednako snažne i logički ekvivalentne.

Jedna od najstarijih i najpoznatijih metoda koja se primenjuje za konstruktivno testiranje manjih programa je simboličko izvršavanje programa.

Jedan od načina njene primene je putem računara što sa druge strane nije osnovni uslov.

Samo donekle podseća na metodu simulacije rada računara, ali se od nje bitno razlikuje po cilju.

Osnovni cilj ove metode je konstruktivno testiranje.



TESTIRANJE SOFTVERA

1. Beskonačno i detaljno testiranje softvera nije moguće - (vremenski rokovi i budžet)
2. Testiranje povezanih grešaka (Defect clustering) - grupiranje grešaka odnosno (u teoriji) da mali broj modula sadrži većinu otkrivenih nedostataka.
3. Povezanost grešaka u modulima (Pesticide paradox)
4. Testiranjem dokazujemo postojanost grešaka
5. Velika zabluda je da ne postoje greške u softveru
6. Testiranje u ranoj fazi razvoja
7. Pristup testiranja prema vrsti projekta



TESTIRANJE SOFTVERA

Proces testiranja ima cilj da potvrdi funkcionalne zahteve, pouzdanost, otpornost na greške i ostale faktore kvaliteta softvera.

Testiranje softvera je aktivnost koja je važna u celom procesu razvoja softvera.

Testiranje softvera koristi određenu strategiju za izbor testova koji su izvodljivi za raspoloživo vreme i resurse.

Opadanje kvaliteta softvera - kvalitet softvera će opadati ukoliko se ne modifikuje.



Planiranje testiranja

Za process testiranja softvera vrši se pravljenje različitih planova za sam process testiranja.

Planovi obuhvataju sledeće aktivnosti:

- Ko radi testiranje
- Šta se testira
- Kada se vrši testiranje
- Kako se vrši testiranje
- Da li postoje neke dodatne aktivnosti koje su neophodne da bi testiranje bilo urađeno
- Ko obezebeđuje te dodatne aktivnosti



- ✓ Plan testiranje se isključivo vezuje za određeni projekat
- ✓ Test plan je povezan sa specifikacijom (detaljno opisuje koje tehnike testiranja se koristi, vreme testiranje, role, neophodni alati....)
- ✓ Analizira rizika (šta se dešava ako test ne prođe)
- ✓ Šta nije testirano



TESTIRANJE SOFTVERA

Razvojno okruženje - je okruženje u kom se projekat razvija i u kom se ukljanjaju pronađeni bug-ovi.

Programer je odgovoran da konfiguriše sve potrebne promene u okruženju da bi funkcija radila na očekivani način.

Test okruženje- odlučuje pod kojim će uslovima softvera i hardvera projekat biti testiran.

Production okruženje- je okruženje u kom je softver puštena za krajnje korisnike.



TESTIRANJE SOFTVERA

Kontinuirano testiranje, koje se izvodi istovremeno sa procesom razvoja, je efikasna tehnika testiranja koja se sprovodi radi automatizacije procesa testiranja. Pomaže u postizanju više ciljeva kao što su:

- Pronalaženje grešaka u ranijim fazama,
- Automatsko izvršavanje testova da bi se otkrili problemi,
- Smanjeno vreme za isporuku softvera,
- Smanjenje poslovnog rizika,
- Obezbeđivanje kontinuiranog procesa isporuke softvera,
- Poboljšava stopu izdavanja verzije softvera.



TESTIRANJE SOFTVERA

Metodologije testiranja softvera su razne strategije ili pristupi koji se koriste za testiranje softvera kako bi se obezbedilo da se softver ponaša i izgleda onako kako se očekuje.

Ne možemo znati da li u nekom delu softvera postoji neka greška ako ga ne pokrenemo i ako ga ne testiramo.



TESTIRANJE SOFTVERA

- Testiranje softvera (dinamika,konačnost, selektivnost, očekivanost)
- Veza testiranja sa drugim aktivnostima razvoja softvera.
- Nivoi testiranja
- Predmet testiranja.
- Ciljevi testiranja softvera
- Tehnike testiranja
- Kombinovanje tehnika
- Proces testiranja softvera
- Evaluacija programa koji se testiraju.
- Evaluacija testova



TESTIRANJE SOFTVERA

PONOVNA UPOTREBA TEST INFORMACIJA

Kad imamo dat tip grešaka, možemo odabratи metod testiranja (funkcionalni ili strukturni) koji će najverovatnije otkritи greške tog tipa. Ovde se postavlja pitanje kada prekinuti testiranje? Evo nekih mogućih odgovora:

- Kad nestane vremena,
- Kad dalje testiranje ne izaziva nove otkaze,
- Kad dalje testiranje ne otkriva nove greške,
- Kad ne može da se smisli nijedna nova test stavka,
- Kad se postigne zahtevana pokrivenost,
- Kad su sve greške uklonjene.



TESTIRANJE SOFTVERA

TESTIRANJE TOKA PODATAKA

Testiranje toka podataka odnosi se na oblike struktturnog testiranja koji se usmeravaju na tačke u kojima promenljive dobijaju vrednosti i na tačke u kojima se te vrednosti koriste (ili pominju).

Testiranje toka podataka služi kao "provera u stvarnosti" za testiranje putanja; zaista, mnogi zagovornici i istraživači testiranja toka podataka smatraju ovaj pristup za oblik testiranja putanje.

Tu postoje dva glavna oblika testiranja toka podataka: jedan pruža skup osnovnih definicija i jedinstvenu strukturu mera pokrivenosti testa, a drugi se zasniva na konceptu "programskog adreska".

Oba formalizuju intuitivno ponašanje (i analizu) testera, i mada oba počinju od programskog grafa, oba se vraćaju u pravcu funkcionalnog testiranja.



TESTIRANJE SOFTVERA

Promenljive koje predstavljaju podatke nekako zadobijaju vrednosti, i te vrednosti se koriste za izračunavanje vrednosti drugih promenljivih.

Još od početka šezdesetih godina, programeri analiziraju početne programe u smislu tačaka (iskaza) u kojima promenljive dobijaju vrednost i tačaka u kojima se te vrednosti koriste.

Te analize su se često zasnivale na usklađenosti spisaka broja iskaza u kojima se pojavljuju imena promenljivih. Rane analize toka bile su usmerene na skup grešaka koje su danas poznate kao anomalije definisanja/pozivanja:

-



TESTIRANJE SOFTVERA

- Promenljiva koja se definiše, ali se nikada ne koristi (niti pominje)
- Promenljiva koja se koristi, ali se nikada ne definiše
- Promenljiva koja se definiše dva puta pre nego što se upotrebi.

Svaka od ovih anomalija može se otkriti na osnovu usklađenosti programa odnosno ove anomalije se mogu otkriti onim što sa naziva "statička analiza" tj. nalaženje grešaka u programu bez njegovog izvršenja.



TESTIRANJE SOFTVERA

Test plan je u suštini dokument koji u sebi sadrži podatke koji su vezani za planiranja testiranja softvera

- Podaci a vezani za akciju aktivnosti planiranja su:
 - Izbor i način strategije testiranja
 - Izbor test okruženja i načina automatizacije testova
 - Izbor nivoa testiranja i veza između njih
 - Integracija testiranja sa drugim aktivnostima u okviru projekta
 - Načina evaluacije rezultata testiranja
 - Metrike za kriterijum završetka testiranja



NEKI nedostaci u procesa testiranja (Shortcoming of the testing process)

Nedostaci u procesu testiranja utiču da mnoge greške ostanu ne otkrivene prilikom testiranja.

Ovi nedostaci nastaju zbog sledećih uzroka:

Nepotpuni testni plan.

Nepostoji dokumentovanje otkrivenih grešaka.

Odlaganje ispravljanja otkrivenih softverskih nedostataka.

Nepotpuna korekcija ili detekcija greška zbog nemarnosti ili vremenskih ograničenja.



Refaktoring (Refactoring)

Ključna strategija u postizanju kardinalnih pravila softverske evolucije je refaktoring (eng. refactoring) koji je Martin Fowler naveo da je to "promena interne strukture softvera da bi bio lakši za razumevanje i jeftiniji za modificiranje bez promene njegovog ponašanja" (Fowler).

Ponekad se kod drastično promeni za vreme održavanja, a ponekad kod nije dovoljno dobar nakon samog procesa kodiranja.



Refactoring (Refactoring)

Refactoring je postupak poboljšanja softverskog programa tako što se radi interno restrukturiranje koda bez izmene ili dodavanja funkcionalnosti.

- ✓ Razvoj podrazumeva postojanje testova. Prosto, nema izmene koda ukoliko nema testa.
- ✓ Razvoj podrazumeva da se kod menja od prvog koraka.
- ✓ Da bi dodalo nešto novo u sledećem koraku.
- ✓ TESTIRATI U SVAKOM KORAKU



Fazi testiranje

Tehniku je konstruisao prof. Barton Miller sa Univerziteta u Wisconsinu 1988.

Osnovna ideja testiranja fazi metodom je generisanje slučajnih ili pseudo-slučajnih podataka kao ulaz u sistem.

Ukoliko sistem prestane da funkcioniše nakon nekog ulaza onda se taj niz podataka zapamti za dalju analizu.



Fazi testiranje

U literaturi se koristi termin negativno testiranje Negativno testiranje predstavlja efikasnu metodu za otkrivanje sigurnosnih grešaka.

Ova vrsta testiranja softvera predstavlja:

- automatizovano
- delimično automatizovano testiranje

Pre svega se onosi na granične slučajeva u softveru,

Ulagni podaci se biraju:

- nasumično,
- delimično nasumično
- ne validni podaci
- Delimično ne validni podaci



Fazi testiranje

- faze (fuz-zer).

Ovde se insistira na velikoj automatizaciji i testovima.

Ne zahteva mnogo ekspertskega vremena.

Testiranje se sastoji od generisanja test slučajeva ulaznih podataka i praćenja izvršenja testiranog softvera u toku obrade.

Postoji više različitih pristupa faz testiranju.

Jedna od manje ove vrste testiranja je mogućnost da ima više false nego pozitivnih rezultata.



Fazi testiranje

Prednosti

- Može otkriti greške propuštene pri ručnoj proveri.
- Daje opštu sliku robusnosti ciljnog softvera.

Mane

Programi sa složenim ulazima mogu zahtevati mnogo više rada da bi se proizveo dovoljno dobar fazi test da bi se postigla što veća pokrivenost koda.



Continuous integration

Neprekidna integracija ili Continuous integration je praksa čestog spajanja (merge) radnih verzija koda u toku razvojnog procesa.

- ✓ Postupak pisanja koda se deli na manje cikluse.
- ✓ Sofverski alati za neprekidnu integraciju.

BENEFIT:

Stalna raspoloživost kompletног izgrađenog softvera.



Sistemi mogu obuhvatati i poverljive informacije i time su potencijalno mete neovlašćenog korištenja. Ono može uključivati:

- Pokušaj probroja u sistem sa strane sa ciljem zabave ili koristi.
- Nanošenje lične štete legalnim korisnicima sistema ili ljudima čiji se podaci nalaze u sistemu.



Testiranje softvera obuhvata različite vrste testiranja kako bi se obezbedilo da softver neće imati funkcionalne i nefunkcionalne nedostatke.

Bezbednosno testiranje podrazumeva izvršavanje nefunkcionalnih tipova testova kako bi se otkrili sigurnosni nedostaci.

Ono se radi kako bi se spriječili različiti hakerski napadi.

Aplikacije koje se nalaze na Internetu.



Prilikom ovakvog testiranja, Tester (ili testierski tim) pokušava da se stavi u ulogu sa "druge strane" to jest osoba koja pokušava da nelegano uđe u u sistem."

Metode koje se mogu koristiti su:

- Pokušaj da se dođe do lozinke
- Korišćenje specijalizovanog softvera za napad na sistem
- Prebukiranje sistema zahtevima
- Otkrivanje grešaka u sistemu i njihovo korištenje za napad
- Upad do neobezbeđenih podataka (ako ih ima)

U literature se može naći :

"Uz dovoljno velike materijalne i vremenske resurse svaki se sistem može uspešno napasti."

TESTIRANJE SOFTVERA





PROBLEM: Kada softver nije adekvatno testiran.

"Testiranje softvera je proces izvršavanja programa/funkcije sistema sa ciljem da se otkrije što više grešaka."

Naglašena je posvećenost aktivnosti otkrivanja bigova (grešaka).

Jedan od ciljeva testiranja softvera je da se izbegnu bilo kakvi incidenti i problemi prilikom korišćenja softvera i njihove posledice.



TESTIRANJE SOFTVERA

Testiranje softvera je već više godina, pa i decenija, ustanovljeno kao naučna, odnosno inženjerska disciplina.

"Testiranje softvera se sastoji od aktivnosti dinamičke verifikacije ponašanja programa na bazi konačnog skupa testova, odabralih na pogodan način iz beskonačnog skupa mogućih načina izvršavanja programa, a prema specificiranom očekivanom ponašanju softvera u razvijanoj aplikaciji tj. zahtevanog kvaliteta softvera".



Bitna razlika :

- ✓ Da li program ili
- ✓ Sistem radi.

Ne postoji savršen softver.

Zato što je jasno da je nemoguće otkriti sve bugove (greške) u softveru.

Od testera se očekuje da proces testiranja softvera učine što efikasnijim i uz što manje troškova obezbede normalan rad određenog softvera.



TESTIRANJE SOFTVERA

U radovima DeMilla - kompleksne greške su povezane sa jednostavnijim greškama.

Kako testiranje skoro nikada ne može da bude kompletno urađeno, ono ne može da pokaže nepostojanje grešaka.

Odatle se nameće zaključak da je cilj testiranja detektovanje što većeg broja grešaka.

Neke greške su kompleksnije od drugih i njihov efekat može biti ozbiljniji.

Testiranje obično ne mora da se fokusira na neki određenu vrstu grešaka zbog postojanja takozvanog grupisanja defekata.



- Veoma je važno detaljno isplanirati ceo proces testiranja, kako bi se slučajno ne bi prikrio neki veliki bug koji može da ugrozi krajnjeg korisnika.**
- U mnogim knjigama i časopisima se tvrdi da testiranje softvera odavno nije samo faza u procesu razvoja softvera već paralelni pod-proces.
- U bilo kom softveru u praksi postoji beskonačan broj mogućih testova koje je praktično nemoguće sve izvesti, te je nemoguće u određenom vremenskom periodu, sa ograničenim resursima (ljudi, oprema i alati) izvršiti totalno testiranje koje bi otkrilo sve greške u softveru.



Prema Snedakeru , postoje sledeće vrste IT projekata:

- ❖ Strategijski ili operativni
- ❖ Dugoročni ili kratkoročni
- ❖ Hardver ili softver
- ❖ Razvojni ili implementacioni

Postoje još neke podele vezano za IT projekte:

Kriterijum vremena: dugoročni i kratkoročni

Karakter izvođača: interni, eksterni i kombinovani



- ✓ Pionirski projekti,
- ✓ Repetitivni projekti,
- ✓ Standardni projekti
- ✓ Potencijalni projekti

Prema karakteru ciljeva :

- (otvoreni, zatvorenii) i
- složenosti ciljeva (niska, visoka)



KOMPONENTE SOFTVERA

Neki od razloga potencijalnog (nastanka) neuspešnog softvera se mogu podeliti na :

- ✓ Složenost problema koji se rešavaju + vremenski okvir koji utiču na razvoja softvera
- ✓ Saradnja među članovima razvojnog tima
- ✓ Česte promene koje "odudaraju od zadate specifikacije".

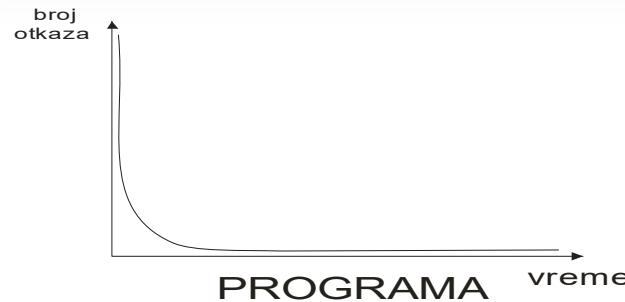
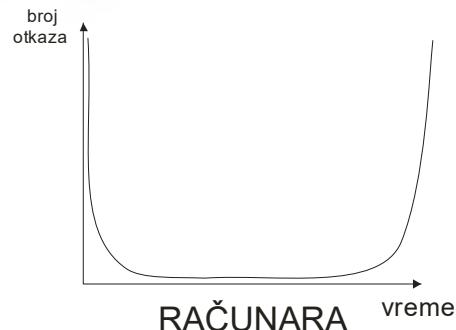
PROGRAM KAO PROIZVOD treba da zadovoljava:

FUNKCIONALNOST

Podrazumeva da program mora odgovarati zahtevima koji proističu iz prirode problema za koji je pisan program. Program je funkcionalan ako zadovoljava razumna očekivanja korisnika

POUZDANOST

Pod pouzdanosti se podrazumeva broj otkaza u jedinici vremena.



PRENOSIVOST

Sposobnost izvršavanja na što više različitih sistema.



TESTIRANJE SOFTVERA

Zašto se onda tolika pažnja u poslednje vreme posvećuje testiranju softvera kada ima toliko ograničavajućih faktora?

Zato je prvenstveni zadatak testera (ili testerskog tima) otkrivanje problema u softveru sa ciljem da se oni otklone i pronađu pre produkcije.



Prema nameni dokumenta se mogu podeliti na :

- Interna dokumenta** - koji nastaju u kompanijama i za potrebe istih
- Eksterna dokumenta** - namenjeni za razmenu sa drugima

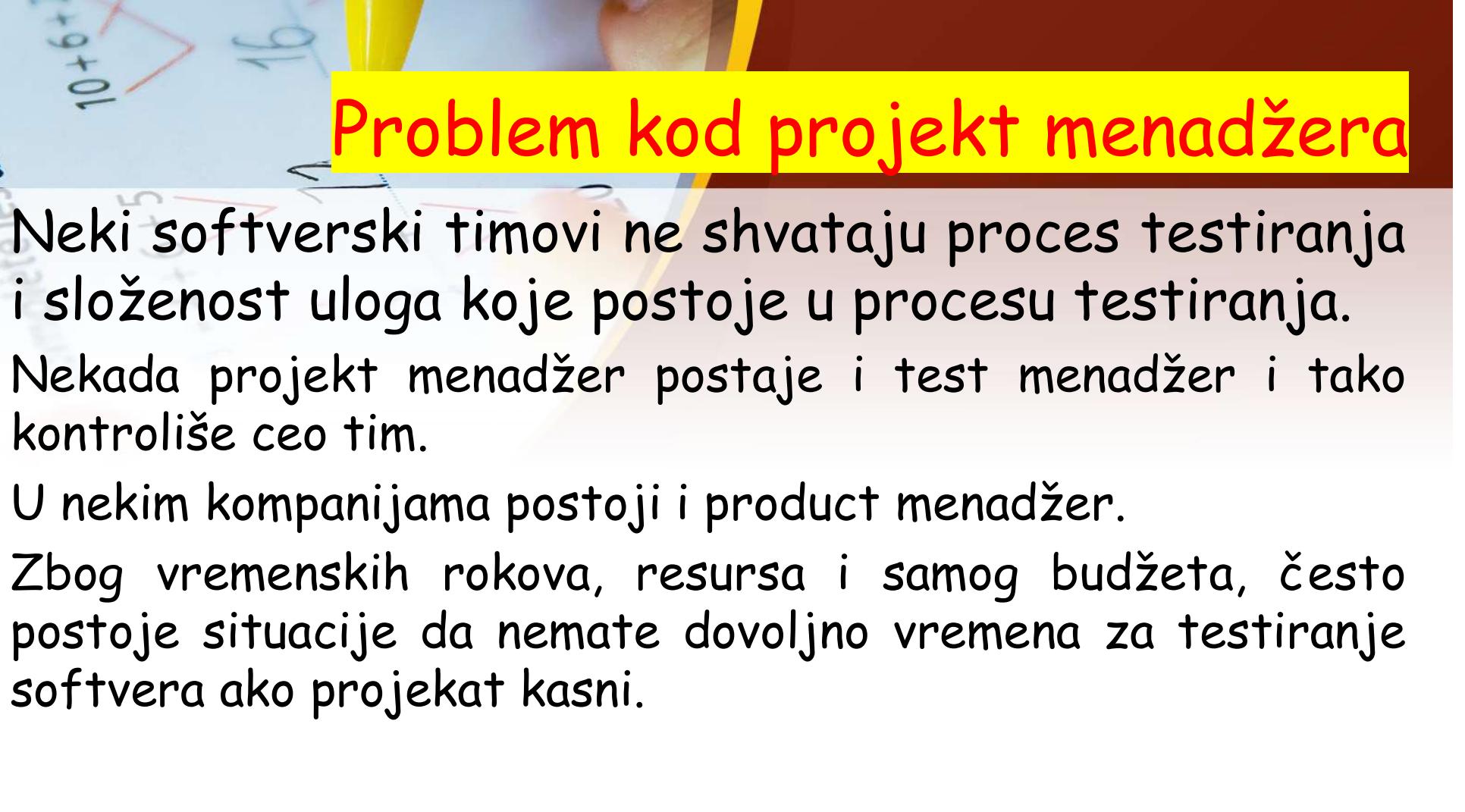
Razvojna dokumentacija (izveštaji o specifikaciji i dizajnu, opisi programa,...)

Podržava efikasnu kooperaciju i koordinaciju između članova razvojnog tima i efikasan pregled i inspekciju dizajna i programskog proizvoda.

Korisnička dokumentacija daje jasan opis kako da se koristi zadatana aplikacija.

Dokumentacija za održavanje:

- ✓ Prikuplja potrebne informacije o kodu, strukturi i zadacima svakog člana tima
 - ✓ Veoma je bitna za lociranje uzroka softverskih grešaka, izmene i korekcije postojećeg softvera.
-
- ✓ **DNEVNIK TESTIRANJA!**



Problem kod projekt menadžera

Neki softverski timovi ne shvataju proces testiranja i složenost uloga koje postoji u procesu testiranja.

Nekada projekt menadžer postaje i test menadžer i tako kontroliše ceo tim.

U nekim kompanijama postoji i product menadžer.

Zbog vremenskih rokova, resursa i samog budžeta, često postoji situacija da nemate dovoljno vremena za testiranje softvera ako projekat kasni.

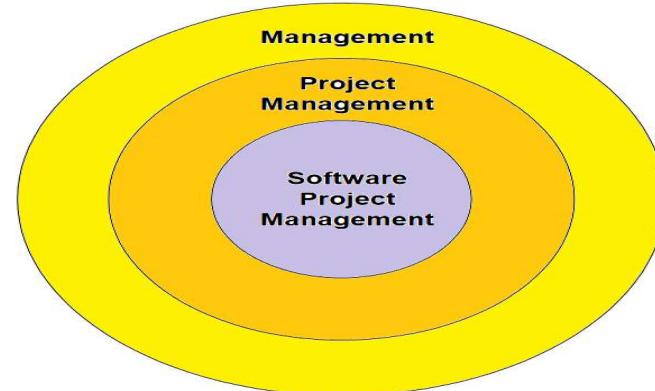
Problem kod projekt menadžera

- Potencijalni problem - mali broj project menadžera adekvatno tretira informacije dobijene tokom procesa Testiranja Softvera.

Neki od zadataka project menadžera su:

- Planirajanje projekta
- Propisuju pravila
- Određuju rokove
- Raspored obavljanja posla
- Kontrolišu rad

Software project management





TESTIRANJE SOFTVERA

Dosadašnji pristup problemu testiranja softvera je bio takav da se kroz čitav repertoar međusobno nezavisnih tehnika i strategija testiranja softvera oceni nivo kvaliteta softverskog proizvoda. Takav pristup u literaturi se naziva pasivan pristup.

Pasivan pristup je kao rezultat davao samo konstatataciju da softverski proizvod nije korektan jer su detektovane (otkrivene) greške i to najčešće u kasnijim fazama razvoja softvera t.j. u fazi testiranja dok su one pravljene mnogo ranije. Može se reći da je život softverskih grešaka bio dug od momenta generisanja do momenta otkrivanja.



Najveće softverske kompanije i grupe svakodnevno objavljaju sve bolje i bolje arhitekture i komponente koje omogućavaju da se napravi odličan softver.

Kompleksan kod se lako generiše brojnim alatima i generatorima koda

U dobrom kompanijama testeri su:

Zaduženi za kontrolu kvaliteta.

Takve kompanije se lako prepoznaju po tome što imaju stabilne softverske proizvode koje koristi veliki broj zadovoljnih korisnika.



TESTIRANJE SOFTVERA

- Faza Optimizacije
- Prevencije grešaka i
- Kontrola kvaliteta



Optimizacija

Šta optimizujemo?

Pod pojmom optimizacijom koda se obično misli na:

- ❖ Poboljšanje čitljivosti koda i kultura programiranja
(da neko može da nastavi brzo da radi posle Vas - komentari, dokumentacija)
- ❖ Optimizaciju koda radi ubrzanja rada
- ❖ Optimizaciju koda radi smanjenja veličine programa



TESTIRANJE SOFTVERA

Testiranje softvera je aktivnost za koju se vezuje negativan prizvuk jer je operativno usmerena na "slabe tačke" procesa razvoja kao i ka kvalitetu samog softverskog proizvoda

Drugi problem je, da skoro nikad nema dovoljno vremena za testiranje softvera

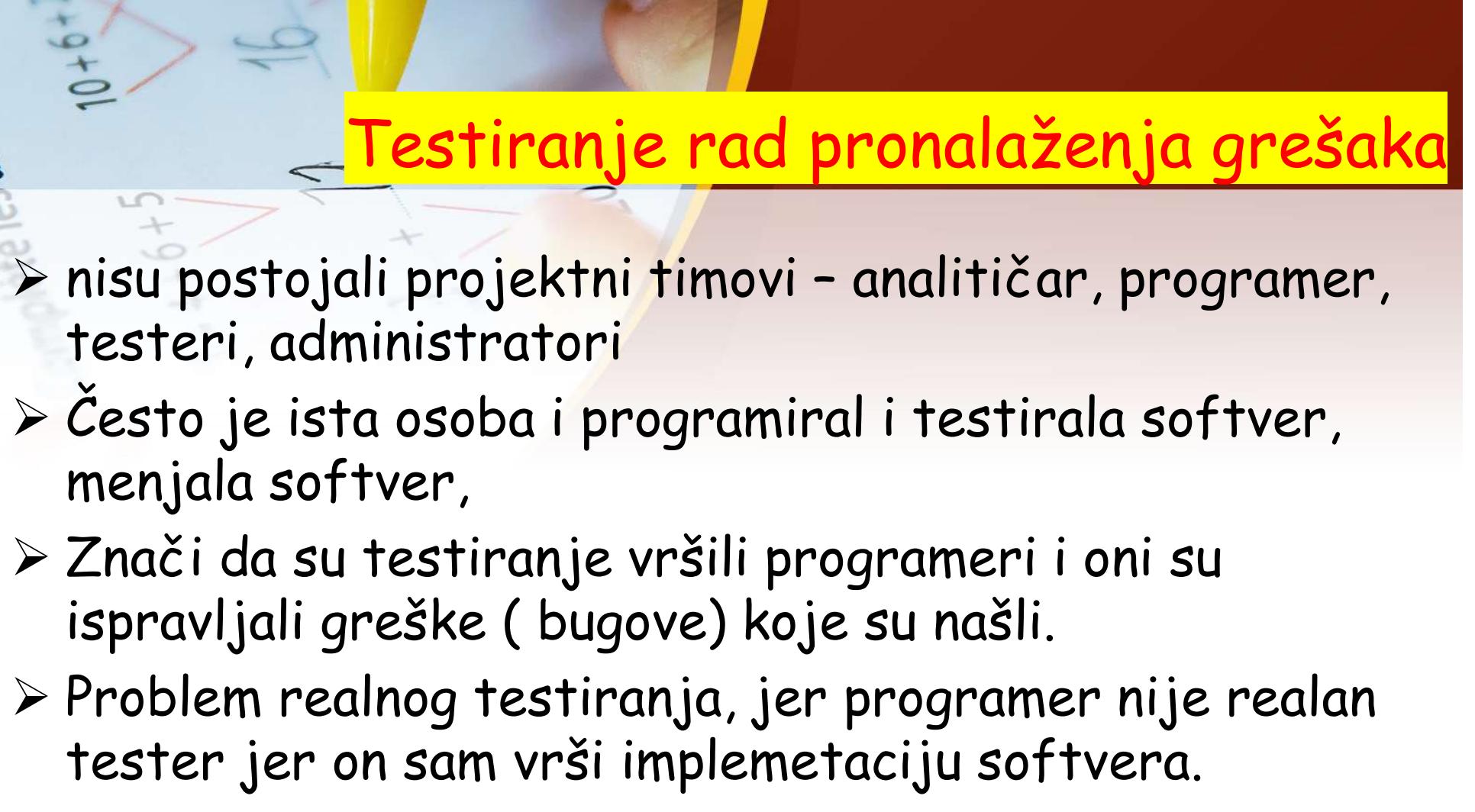
Zato treba da menjamo odnos prema aktivnosti testiranja softvera i vremena koje je planirano za testiranje,

Testiranje softvera u ranim fazama razvoja softvera



Testiranje se menjalo kroz istoriju:

- ✓ Testiranje radi pronalaženja bugova (grešaka)
- ✓ Demonstraciono testiranje
- ✓ Destrupciono testiranje
- ✓ Evaluaciono testiranje
- ✓ Preventivno testiranje



Testiranje rad pronalaženja grešaka

- nisu postojali projektni timovi - analitičar, programer, testeri, administratori
- Često je ista osoba i programiral i testirala softver, menjala softver,
- Znači da su testiranje vršili programeri i oni su ispravljali greške (bugove) koje su našli.
- Problem realnog testiranja, jer programer nije realan tester jer on sam vrši implementaciju softvera.



➤ Testiranje je bilo proces kojim se demonstriralo da softver radi upravo ono za šta je originalno namenjen.

Svako odstupanje je predstavljalo grešku koju je trebalo ispraviti.

➤ Pronalaženje grešaka je predstavljalo proces koji je bio povezan samo sa funkcionalnosti softvera.



- Testiranje je proces otkrivanja grešaka u programu.
- Pronalaženje grešaka je proces otkrivanja gde se greške nalaze u programskom kodu

Testiranje Softvera: Pronalaze se test slučajevi kojim se najlakše dokazuje da stvarno postoje bugovi (greške) u programu.

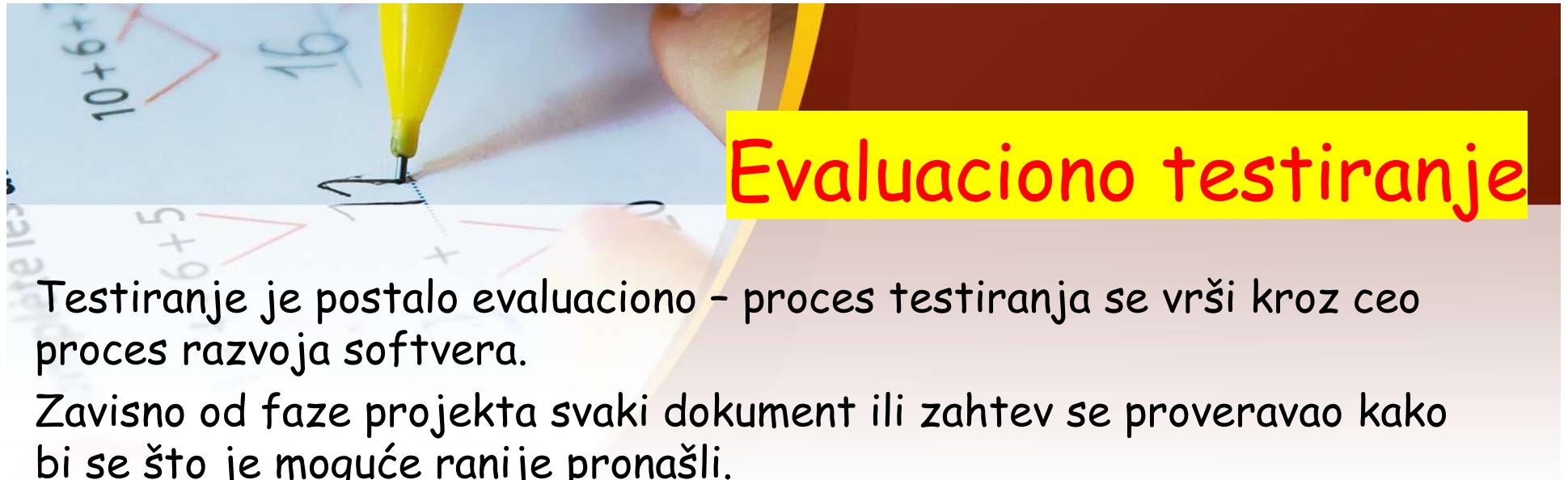


TESTIRANJE SOFTVERA

Aktivnosti testiranja softvera je:

- Programski kod visoke pouzdanosti,
- Velike otpornosti (robustan),
- Vrlo stabilan
- da potvrди da softver zadovoljava skoro sve zahteve krajnjeg korisnika ili one koje je obavezno zahtevao.

Zato se aktivnost testiranja softvera smatra destruktivnom prema programskom kodu, mada se na kraju pokazuje da je ta aktivnost vrlo konstruktivna



Testiranje je postalo evaluaciono - proces testiranja se vrši kroz ceo proces razvoja softvera.

Zavisno od faze projekta svaki dokument ili zahtev se proveravao kako bi se što je moguće ranije pronašli.

U agilnim metodama testiranje je istovremeno i preventivno i evaluaciono.

Na početku svake iteracije testeri evaluiraju zahteve pre nego što se počne sa implementacijom kako bi se obezbedilo da se samo validan zahtev predstavi timu na implementaciju.



Preventivno testiranje

Testiranje softvera je aktivnost ili proces kojim se pokazuje ili demonstrira da program ili sistem izvršava sve predviđene funkcije korektno.

Za razliku od evaluacionog testiranja kojim se testira svaka komponenta pošto se napravi, u prevencionom testiranju se testovi prave i pre nego što se komponente naprave.

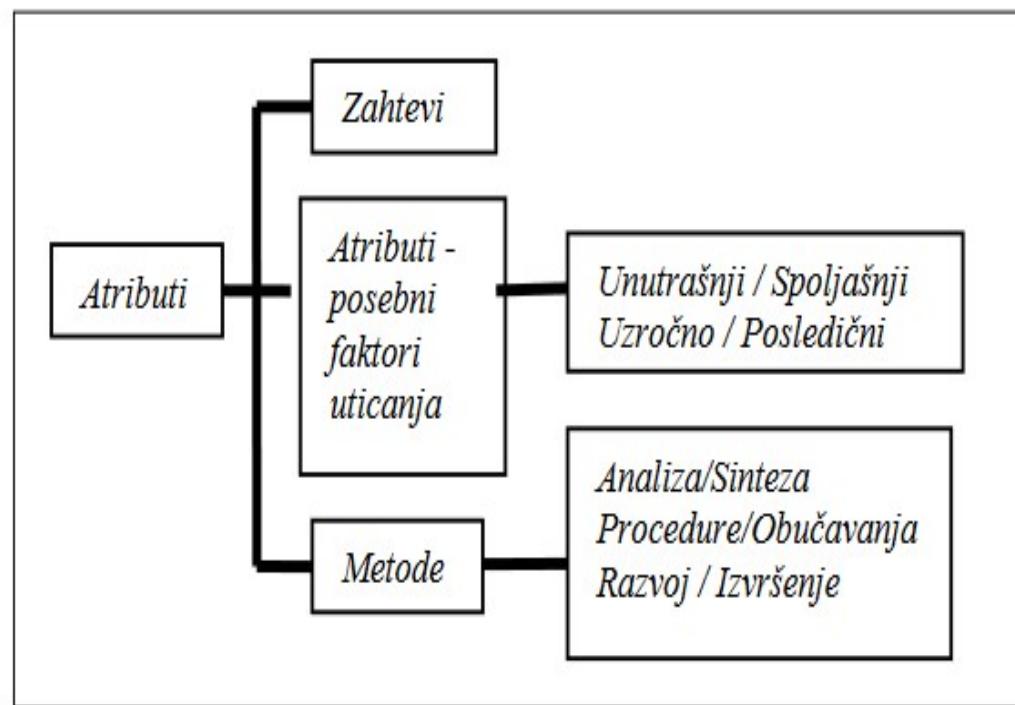
Stav prema kvalitetu softvera je prevencija, mnogo je bolje izbeći probleme nego ih ispravljati.

Troškovi proizvodnje razvoja softvera se smanjuju sa ranim otkrivanjem i ispravkom grešaka. ODNOSNO:

Preventivni troškovi sastoje se od akcija koje se preduzimaju kako bi se sprečili defekti.



OPŠTA KLASIFIKACIJA ATRIBUTA KVALITETA





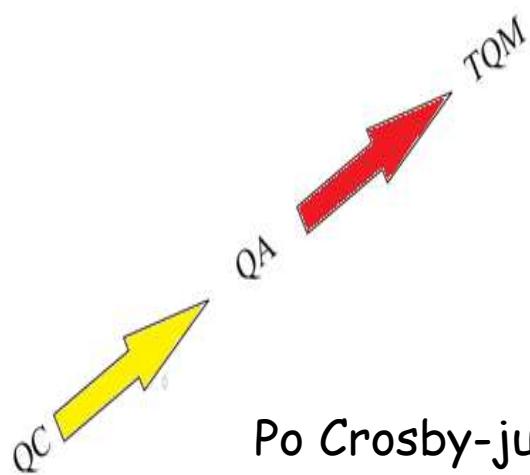
Ključni aspekti kvaliteta softvera odnose se na:

- KVALITET PROCESA RAZVOJA
- STRUKTURNI KVALITET SOFTVERA
- FUNKCIONALNI KVALITET SOFTVERA

PROBLEM kod KVALITETA SOFTVERA:

Reputacije kompanije

Gubitka značajnih klijenata



KVALITET SOFTVERA

- ✓ Kontrola kvalitete - QC
(Quality Control)
- ✓ Osiguranje kvaliteta - QA
(Quality Assurance)
- ✓ Potpuno upravljanje kvalitetom - TQM (Total Quality Management),

Po Crosby-ju kvalitet je "prilagođavanje zahtevima"



KVALITET SOFTVERA

Testiranje softvera može se posmatrati kao jedan od metoda koji se koriste za kontrolu kvaliteta softvera.

Kontrola kvaliteta je process koji se sprovodi na kraju izrade softvera i kontrolom se upoređuje kvalitet urađenog softvera sa polaznom specifikacijama.

Greške koje se prave tokom rada su normalna i svakodnevna stvar i nisu direktno vezane za izradu softvera.

Greške se dešavaju u svim oblastima rada i uzrokovane su kako ljudskim, tako i drugim faktorima.

Imajući u vidu da su greške realna pojava, u svim oblastima rada se uvodi sistem kontrole kvaliteta softvera kojim se greške identifikuju i otklanjaju pre nego što izazovu veće probleme u radu.



Kvalitet softvera podrazumeva usklađenost sa operativnim i efektivnim potrebama",

Kvalitet softvera se može definisati na različite načine :

- ✓ Usaglašenost sa zahtevima i potrebama korisnika
- ✓ Dobri atributi proizvoda (brzina rada, pokretanja...)
- ✓ Lakoća održavanja i promena u samom softveru
- ✓ Usaglašenost sa standardima
- ✓ Rad sa огромном količinom podataka



KVALITET SOFTVERA

"Upravljanje kvalitetom softvera u najjednostavnijoj mogućoj formi je jedan proces kroz koji možemo definisati kvalitet softvera, ali i planirati, i i proveriti koliko je softver u razvoju u skladu sa inicijalnim zahtevima."



Šest generalnih karakteristika kvalitetnog softvera:

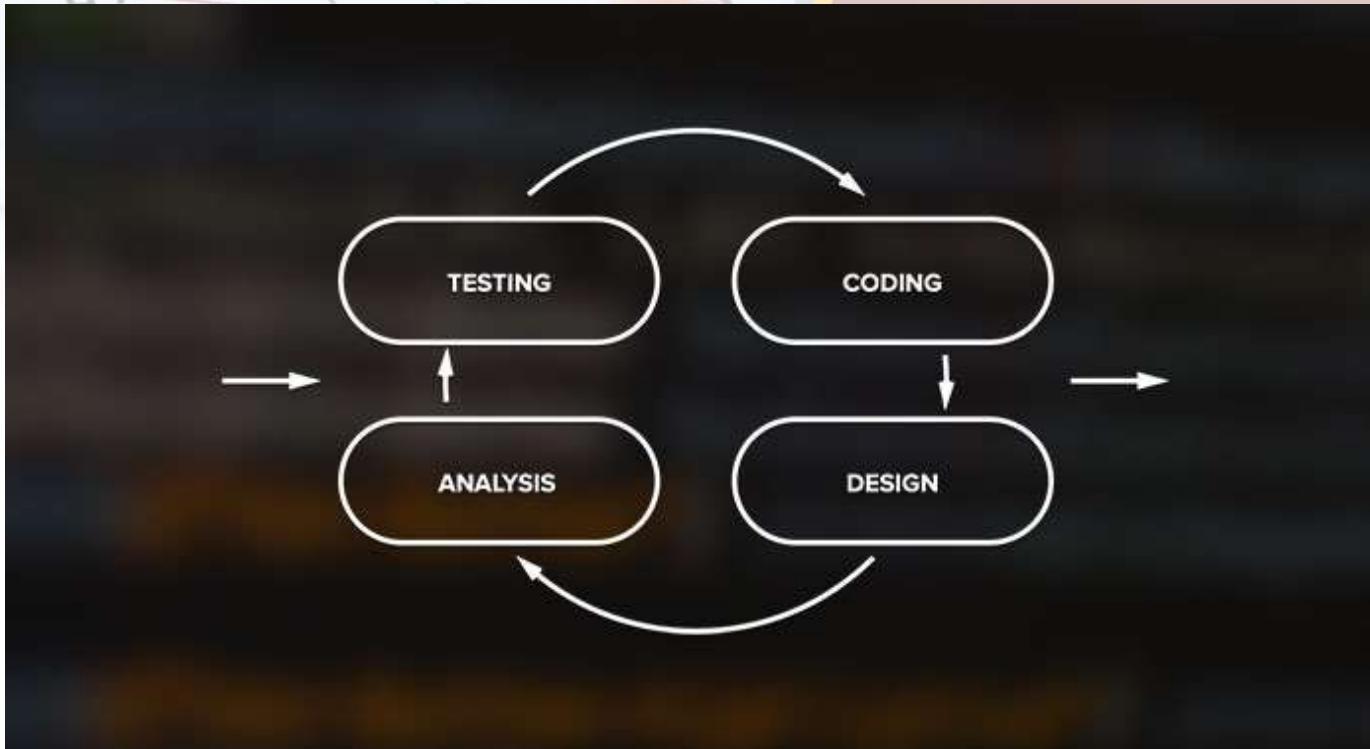
- ✓ 1. **funkcionalnost** - postojanost funkcija softvera ili njenih naznačenih osobina
- ✓ 2. **pouzdanost** - sposobnost softvera da održava nivo performansi pod navedenim uslovima, u toku navedenog vremenskog perioda
- ✓ 3. **upotrebljivost** - pogodnost za korišćenje softvera, odnosno kompleksnost za krajnjeg korisnika
- ✓ 4. **efikasnost** - odnos između nivoa performansi softvera i količine resursa koji se koriste, pod navedenim uslovima
- ✓ 5. **održivost** - sposobnost softvera da jednostavno podrži buduće modifikacije
- ✓ 6. **prenosivost** - sposobnost softvera da bude prenesen iz jednog u drugo radno okruženje



TESTIRANJE SOFVERA

- Kao što se vidi od strategije rešavanja zavisi kako će teći projekat

TESTIRANJE SOFTVERA





ZAŠTO TESTIRAMO PROGRAM?

- Da bi lakše pokazali korisnicima da softver zadovoljava željene zahteve i funkcionalnosti.
 - Koristimo testove validacije
 - Tada očekujemo od sistema da za skup test primera radi korektno i da daje isti rezultat
 - Uspešni testovi pokazuju da sistem pruža željene funkcionalnosti.



ZAŠTO TESTIRAMO SOFTVER?

- ✓ Da bi lakše primetili ili locirali greške i probleme u samom programu.
- ✓ Koristimo defect testiranje
- ✓ Test primeri korišćeni pri ovakovom testiranju su pravljeni kako bi izazvali pojavljivanje grešaka (ukoliko one postoje) i pokazali eventualno neželjeno ili neočekivano ponašanja sistema.



NEKI KORACI U PROCESU TESTIRANJA PROGRAMA

- ✓ Ponovite testove za iste ulazne podatke ili niz ulaznih podataka veliki broj puta.
- ✓ Pronaći ako je moguće da softver generiše neispravane izlazne vrednosti.
- ✓ Treba postići da rezultati izračunavanja budu preveliki ili premali.



❖ Analiza i definisanje zahteva

U ovoj fazi se utvrđuju zahtevi koje sistem treba da zadovolji, što se postiže saradnjom između razvojnog tima i korisnika sistema. Rezulat ove faze je lista korisničkih zahteva.

❖ Projektovanje sistema (dizajn)

U ovoj fazi se generiše projekat sistema koji daje plan rešenja odnosno arhitekturu sistema.

❖ Implementacija softvera

Faza u kojoj developeri pišu kod softvera.

Razvoj softvera je podeljen u nekoliko faza

- Testiranje softvera

Faza u kojoj se otkrivaju i ispravljaju greške u sistemu.

- Isporuka sistema

Sistem se isporučuje naručiocu, softver se instalira u radnom okruženju i vrši se obuka korisnika.

- Održavanje

Dugotrajna faza u kojoj se ispravljaju greške u sistemu, otkrivenе nakon njegove isporuke.

Radi se na dalnjem unapređenju delova sistema prema zahtevima korisnika.



TESTIRANJE SOFTVERA

- Development testing

Testovi u toku razvoja

- Test-driven development

Test driven development (TDD) je proces razvoja softvera nastao kao deo Ekstremnog programiranja, da bi ga kasnije usvojile sve agilne metodologije

- Release testing

- User testing



- Development testing - to je testiranje softvera u fazama razvoja kako bi greške bile otkrivene i otklonjene iz programa.
- Release testing - testiranje čitavog sistema od strane posebnog tima za testiranje pre nego što se sistem konačno pusti u produkciju.
- User testing - kada korisnici ili potencijalni korisnici testiraju sistem u svom okruženju.



DEVELOPMENT TESTING Predstavlja sva testiranja izvršena od strane razvojnog tima na sistemu.

Dele se na:

Unit tests

Testiranje jedinica koda, gde je svaka jedinica koda zasebno testirana, pri čemu jedinica koda može biti funkcija, metod, klasa, operacija, struktura podataka.



Component testing

Testiranje komponenata, pri čemu komponenta predstavlja integrisane jedinice koda ko je zajedno pružaju neku funkcionalnost, tako da se ovo testiranje uglavnom (a ne uvek) bavi testiranjem interfejsa te komponente.



System testing

□ System testing

Testiranje sistema kao celine.

Sistemsko testiranje se zasniva na testiranju sistema kao celine, nakon što je završena kompletna integracija.

Definisano je funkcionalnim zahtevima i specifikacijama sistema.

Po nekim statistikama tvrdi se da najviše grešaka nastane u fazi implementacije.

Tada je neophodno obavljati testiranje sistema tokom i nakon implementacije.

Na ovaj način se smanjuje mogućnost grešaka.



Faze u testiranju softvera

- Prva faza (eng. Red Phase)
- Druga faza (eng. Green Phase)
- Treća faza (eng. RefactorPhase)



Faze u testiranju softvera

- Prva faza (eng. Red Phase)

To je faza koja programerima koji tek počinju sa primenom ove metode stvara najviše problema.

Ona podrazumeva pisanje testova, za klase i metode koje još nisu implementirane.

Posledica je da napisani testovi u ovoj fazi možda neće proći.



Druga faza (eng.Green Phase)

Sve dok testovi ne prolaze, traje prva faza.

Da bi prešli u drugu fazu potrebno je implementirati kod potreban za testove da bi prošli.

Dobra praksa da se napiše najjednostavniji kod da bi test prošao.



Treća faza (eng. Refactor Phase)

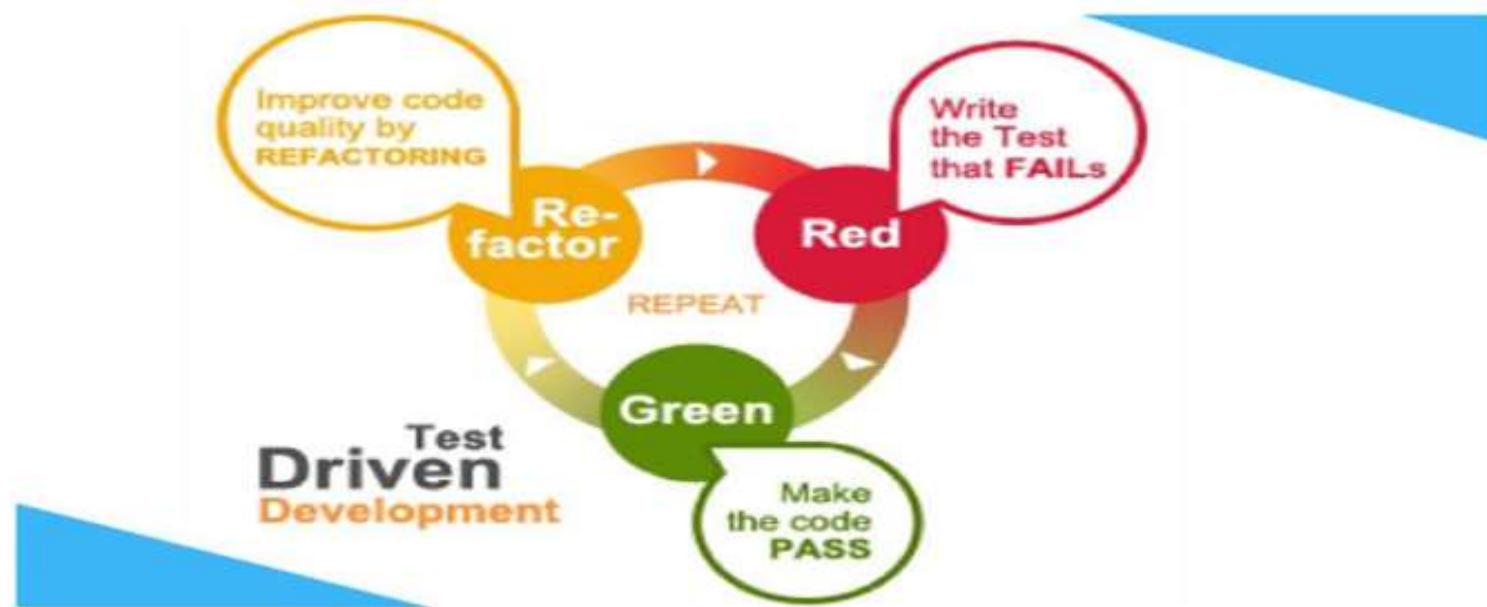
U ovoj fazi se vodi računa o kvalitetitu koda koja je napisan, jednostavnosti održavanja

Refaktorizacija podrazumeva menjanje

Testovi u ovoj fazi služe da bi developeri videli da promene na kodu ne menjaju dosadašnju prolaznost testova.

Dok testovi prolaze znamo da je kod koji je napisan u redu.

Faze u testiranju softvera





- Pišu se testovi koje prolaze.
- Prave se jednostavniji programi i korisnički interfejsi (eng. Application Interface - API).
- Programer koji piše program i pravi korisnički interfejs je prva osoba koja ih koristi, te na osnovu toga može da vidi da li imaju smisla, nerealno testiranje ili je potrebno poboljšanje.



- U finalnim verzijama skoro da nema ne korišćenog koda.
- Testovi koji se pišu omogućavaju da sve (eventualne) promene u kodu ili dodavanje novih metoda neće narušiti već postojeće funkcionalnosti.
- Ova metoda ima vrlo malo defekata.
Ako se uoči neki problem ili bug, kod se ispravlja i piše se novi test za taj bug da bi se omogućilo da se taj problem neće više pojavljivati.





"Testiranje programa je proces izvršavanja programa i upoređivanje opserviranih ponašanja prema željenom ponašanju."

"Primarni cilj testiranja je otkrivanje grešaka u softveru sa sekundarnim ciljem građenja samopouzdanja kod testera u slučaju kada test ne otkriva grešku ". / Myers G.



- Programi se mogu testirati u raznim fazama razvoja i različitim stepenom strogosti.
- Kao i svaki deo razvoja testiranje zahteva rad. Testiranje programa je jedna od najsukupljih pa, prema tome, i najvažnijih aktivnosti u toku njegovog životnog ciklusa.
- Sprovodi se, kako u toku prvobitne realizacije tako i u fazi eksploracije i održavanja i to prilikom svake modifikacije programa.



TESTIRANJE SOFTVERA

Konflikt između ova dva cilja se uočava kada proces testiranja ne otkriva grešku.

U odsustvu drugih informacija, ovo može značiti ili da je softver visokog kvaliteta ili da je veoma niskog kvaliteta.

Postavljeno davne 1989 godine.

[Mye,1989] Myers G. *The Art of Software Testing*, još 1989 godine.



Direktni ciljevi:

- ❖ Identifikacija i otkrivanje grešaka koliko je moguće pre produkcije.
- ❖ Postizanje što većeg nivoa kvaliteta softvera

Indirektni ciljevi :

- Skupljanje informacija o softverskim greškama
(da bi se one ispravile u novim verzijama softvera)



TESTIRANJE SOFTVERA

Da bi se izvelo testiranje velikih i složenih programa ono se mora izvesti što sistematičnije, kako bi se ostvarila.

Ovo, pak, znači da od svih metoda za testiranje jedina koja se ne sme primenjivati je "**ad hoc**" metoda testiranje jer ona ne može da utvrdi kvalitet i ispravnost, ni prema specifikaciji, ni prema konstrukciji, niti prema primeni.

STRATEGIJA TESTIRANJA SOFTVERA

Strategija testiranja predstavlja celokupan pristup testiranju, pri čemu se definišu nivoi testiranja, metode, tehnike i alati.

Ukoliko bi se radilo u idealnim uslovima, strategija testiranja bila bi zajednička za celu organizaciju, kao i za sve programe unutar nje.

Primena i razvoj strategije vrlo su bitni za postizanje određenog kvaliteta programa a samim tim i realizaciju projekta.



TESTIRANJE SOFTVERA

Planiranje testiranja trebalo bi počne sa **ranom fazom procesa uzimanja zahteva.**

To znači da test planovi i procedure moraju biti sistematski i kontinualno razvijani i po potrebi redefinisani.

Pravi stav prema **kvalitetu** je prevencija, mnogo je bolje izbeći probleme nego ih ispravljati.

Zbog svega navedenog, jasno je zašto je softversko testiranje tesno povezano sa drugim oblastima softverskog inženjerstva

STRATEGIJA TESTIRANJA SOFTVERA

- Prvi korak pri izboru strategije testiranja programa jeste

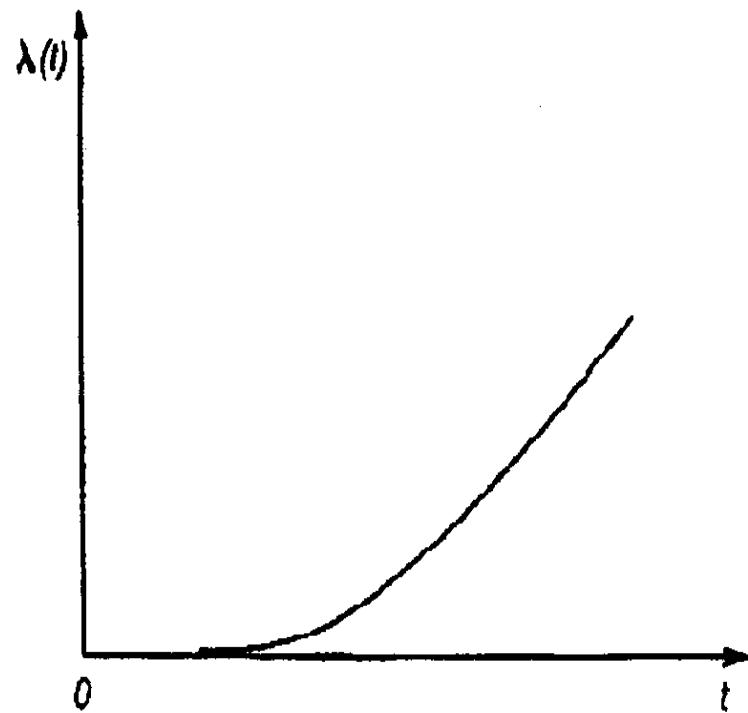
Specifikacija.

OPŠTI MODEL RASPODELE GREŠAKA U SOFTVERU

Postoji takozvana **Vejbulova raspodela grešaka** koja se može podeliti:

- Testiranje pokazuje kako će se program ponašati u eksploataciji.
- Sve greške imaju istu verovatnoću pojavljivanja
- Greške su nezavisne jedna od druge.
- U početku samog procesa testiranja postoji fiksni broj grešaka u softveru tj. staticna pretpostavka generisanja grešaka.
- Vremenska raspodela otkrivanja grešaka podleže Vejbulovoj raspodeli.
- Broj otkrivenih grešaka u nekom vremenskom intervalu je nezavisan od broja otkrivenih grešaka u drugom vremenskom intervalu.

OTKAZI



U praksi, otkazi u zoni tzv. negativnih vremena označavali bi otkaze koji su se desili pre nego što je sistem pušten u korišćenje.
Za svaki parametar su definisane granice ispravnog rada

Intezitet otkaza
Deli se na slučajne i sistemske.



Prednosti

- Nezavisni testeri su nezavisni i nepristrasni
 - Mogu da pronađu više nedostataka (greške, bagovi) nego sami programeri
 - Imaju drugačije principe i nalaze drugačije nedostatke nego programeri

Najčešće programer implicitno podrazumeva određeno (stanje, ponašanje) programa

- Testerski tim ima neke druge principe i ne polazi od istih pretpostavki (ili sam tester, ako nije u testerskom timu)
- Ne znači da nema svoje pretpostavke i ponašanje koje on implicitno podrazumeva
- Implicitno razumevanje testera je bliže nego razumevanju krajnjeg korisnika





Prvi princip

Svaka aplikacija ili proizvod pušta se u produkciju nakon "dovoljnih metoda i tehnika testiranja" od strane različitih timova.

Da li je treba prestati sa **TESTIRANJEM SOFTVERA** ikada ?

NIKAD NE TREBA PRESTATI!

Testiranje softvera se ne može dokazati da softver ne sadrži greške ili nedostatke. Postoje 2 slučaja:

- Ako test ne prolazi, to znači da nedostatak postoji
- Ako svi testovi prolaze, to i dalje ne dokazuje da nedostataka nema

Što više testiramo softver smanjuje se verovatnoća da određeni nedostaci postoje, a to ne znači da nema nedostataka u samom softveru.

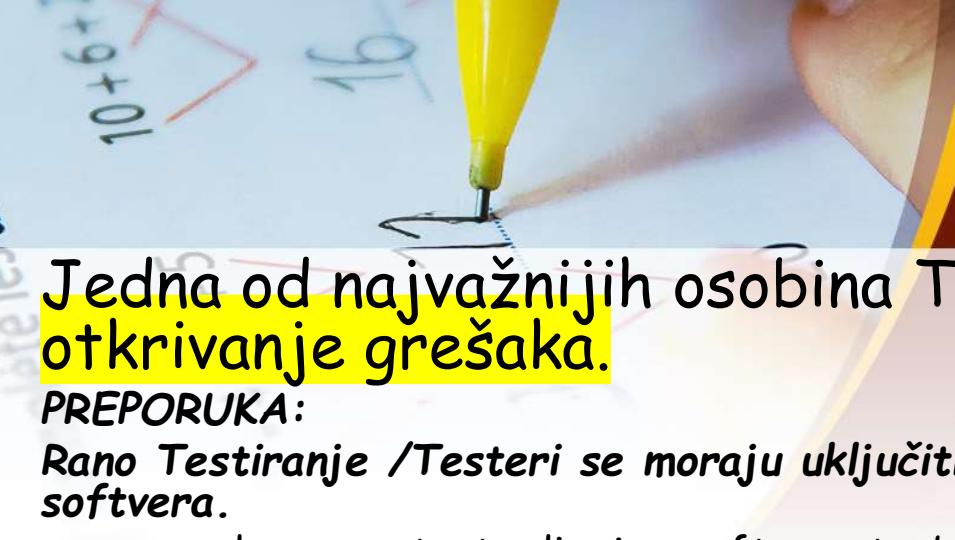


Drugi princip

Takozvano "**Potpuno testiranje je nemoguće**"

Testiranjem se identificuje što je više moguće nedostataka. Test se smatra uspešnim ukoliko je ponašanje sistema pri njegovom izvršavanju u skladu sa korisničkim zahtevima.

- ✓ Za sve ulazne vrednosti (nije moguće da se izvrše i kreiraju svi testovi koji bi pokrili sve kombinacije ulaznih vrednost zajedno sa kombinacijama svih različitih preduslova).
- ✓ U zavisnosti od procenta celokupnog izvornog koda kome se može pristupiti, pokrivenost koda testovima može biti ograničena.
- ✓ Može se reći da Test slučaj samo uzorak iz skupa mogućih podataka
 - ✓ potrebno je pronaći podskup mogućih test slučajeva tako da u granicama dostupnih resursa obezbede što veću verovatnoću ispravnosti programa



Treći princip

Jedna od najvažnijih osobina Testiranja softvera je rano otkrivanje grešaka.

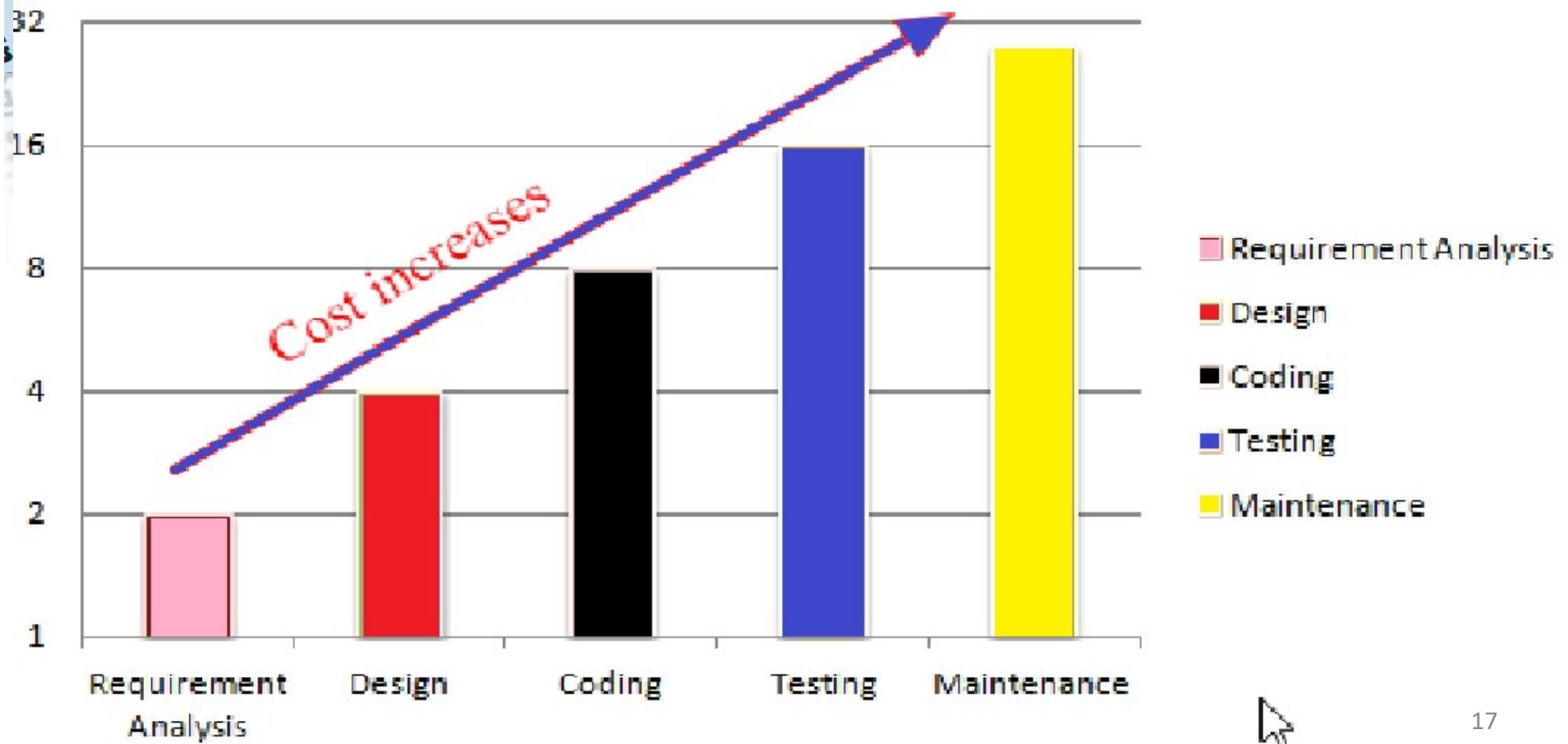
PREPORUKA:

Rano Testiranje / Testeri se moraju uključiti u ranoj fazi životnog ciklusa razvoja softvera.

- od samog starta dizajna softvera treba uračunati i testiranje i uključiti ga u sve faze razvoja softvera
- *Rano otkrivene greške se lakše otklanjaju.*

Vrlo je bitan trenutak kada se testovi pripremaju. Što se ranije testovi pripreme to će se bolje razumeti zahtevi projekta. Ta rana priprema testova pomaže da se problemi otkriju i preduprede. Rano pripremljeni testovi povećaju kasniju efikasnost.

Defect fix cost





Četvrti princip

- Grupisanje nedostataka

Grupisanje i sistematizovanje podataka. Treba analizirati i dati proporciju između broja očekivanih i pronađenih defekata po modulu.

- ✓ Moduli koji sadrže najviše defekata su najkritičniji i sadrže implementaciju bitnih funkcionalnosti sistema.
 - Nedostaci nisu ravnomerno raspoređeni po programu
 - Većina nedostataka se nalazi u pojedinim delovima test objekta
 - Ako se u određenom delu programa pronađu nedostaci, testirati i dalje jer najverovatnije postoji mogućnost da postoje još u tom delu programa



Peti princip

- Paradoks pesticida

- Kao što insekti postaju otporni na pesticide, tako da ukoliko više puta testove stalno ponavljamo u svakoj iteraciji testiranja, ti testovi više neće moći da otkriju nove defekte. Dakle, ako ponavljamo istu seriju testova, metoda testiranja softvera beskorisna je za otkrivanje novih nedostataka.

- Skup test slučajeva treba redovno pregledati i revidirati
 - novi nedostaci se verovatno nalaze u delovima koda i scenarijima koje postojeći test slučajevi ne pokrivaju



Šesti princip

Testiranje zavisi od konteksta

Testiranje se prilagođava funkcionalnostima sistema

- Testiranje mora biti prilagođeno nameni aplikacije, okolnostima u kojima se koristi i mogućem riziku koji otkaz izaziva
- Sugestija (a neki smatraju da je to i pravilo)
- Različiti sistemi se ne mogu testirati na isti način
 - strategija testiranja, intenzitet testiranja, kriterijum završetka itd. moraju biti prilagođeni kontekstu konkretnog sistema



Sedmi princip

Pogrešna je ideja ili pretpostavka je da je sistem bez nedostataka upotrebljiv.

Važno je da se dobiju optimalni rezultati testa (testiranjem softvera) bez odstupanja od testnog cilja.

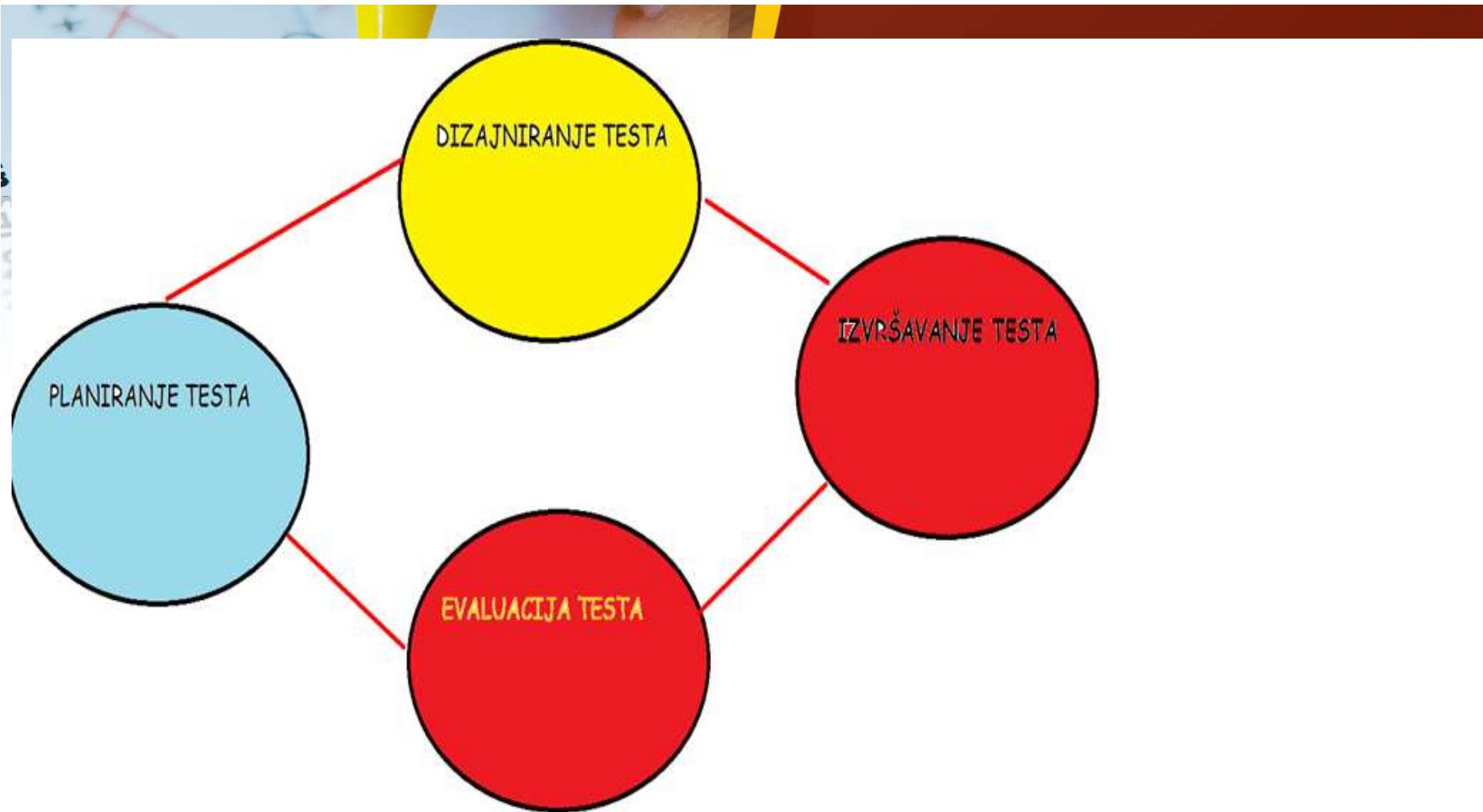
- Odsustvo grešaka ne garantuje da sistem radi kako treba – pronalaženje i ispravljanje defekata ne pomaže ako je sistem neupotrebljiv ili ako ne ispunjava zahteve korisnika
- Korisnik ne očekuje samo ispravan sistem nego i sistem koji je vizuelno i funkcionalno za njega intuitivan
 - intuitivnost je dosta subjektivna kategorija, pa ju je teško ugraditi u specifikaciju zahteva
 - treba uvesti korisnika u rane faze razvoja aplikacije i izradom ranih prototipa dobiti korisne povratne informacije u pogledu njegovih očekivanja



PROTOTIP

Prototip je (po literaturi ili nekim autorima) ogledna verzija ili simulacija samog (IT) proizvoda.

- ✓ Kada su od strane korisnika samo uopšteno definisani ciljevi razvoja proizvoda, (ali ne i detalji u pogledu ulaza, procedura i izlaza)
- ✓ Preporuka: Simulirati rad softvera da bi korisnik mogao videte kako će budući softverski proizvod raditi
- ✓ Testiranjem prototipova zajedno sa krajnjim korisnicima.
Ishod: Tada UX timovi mogu poboljšati korisničko iskustvo.
- ✓ Kada kompanija želi proveriti efikasnost algoritama ili samog sistema.
- ✓ Greške u zahtevima i dizajnu otkrivaju se u ranoj fazi.

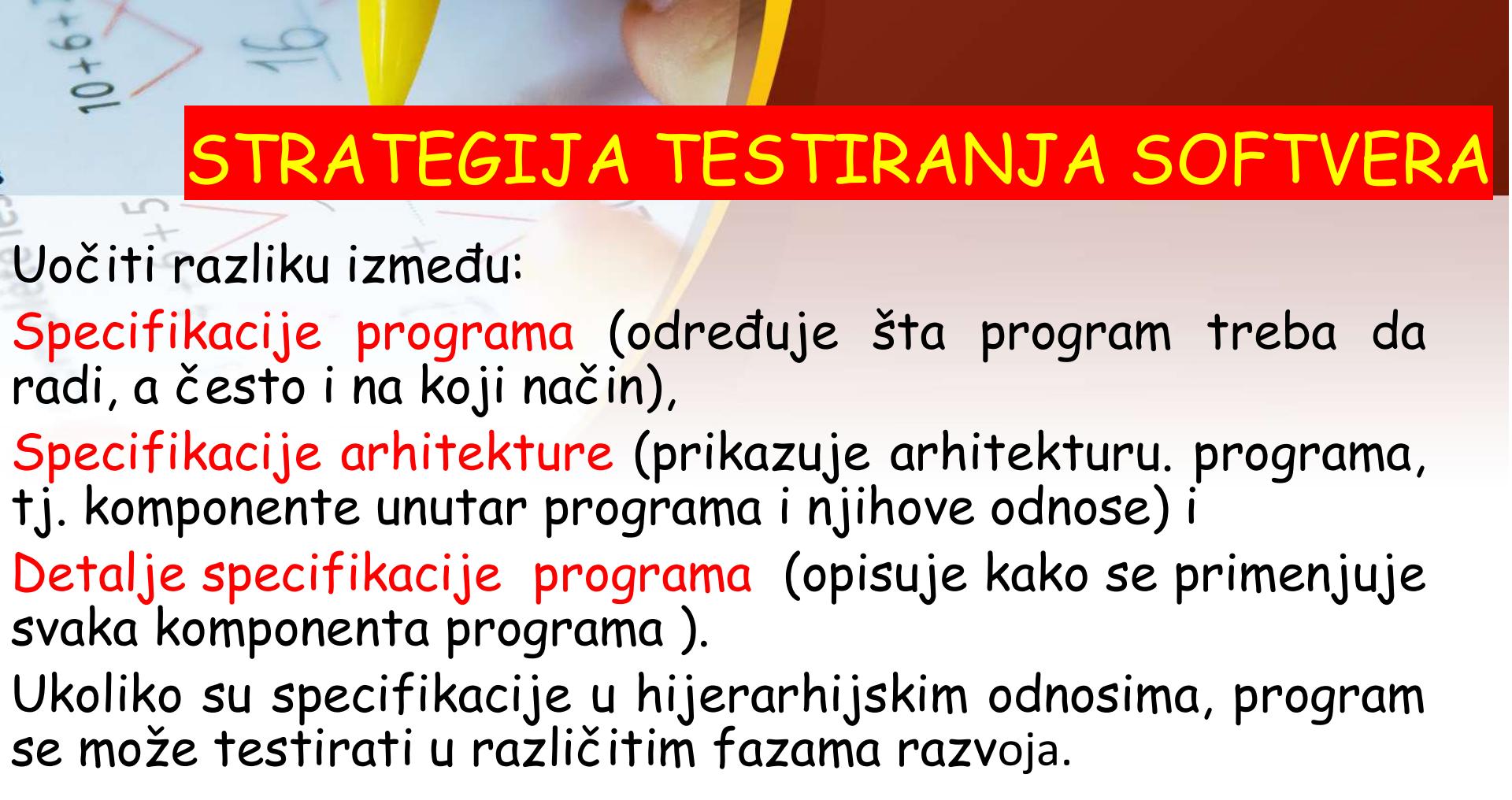


STRATEGIJA TESTIRANJA SOFTVERA

Specifikacija predstavlja osnovnu komponentu svih definicija.

Programi mogu biti veoma jednostavni ili složeni, dok specifikacija (zavisno od veličine i применjenih programerskih metoda) može da bude u obliku jednog dokumenta ili da predstavlja složenu hijerarhiju dokumenata.

Ukoliko se posmatraju različite hijerarhije programa specifikacija može se zaključiti da se one sastoje od tri ili više nivoa dokumenata.



STRATEGIJA TESTIRANJA SOFTVERA

Uočiti razliku između:

Specifikacije programa (određuje šta program treba da radi, a često i na koji način),

Specifikacije arhitekture (prikazuje arhitekturu programa, tj. komponente unutar programa i njihove odnose) i

Detalje specifikacije programa (opisuje kako se primenjuje svaka komponenta programa).

Ukoliko su specifikacije u hijerarhijskim odnosima, program se može testirati u različitim fazama razvoja.



STRATEGIJA TESTIRANJA SOFTVERA

Na osnovu hijerarhije programa specifikacije, evidentni su različiti nivoi testiranja:

- Testiranje jedinice (podrazumeva testiranje svake jedinice kao osnovne komponente u cilju određivanja korektnosti)
- Testiranje integracije programa korak po korak (sa sve većim i većim komponenata programa koje se integrišu ali i testiranjem sve do onog momenta kada funkcioniše kao celina)

TESTIRANJE INTEGRACIJE PROGRAMA

Cilj je :

Pronaći nedostatke u komunikaciji i konflikte između komponenti koje se integrišu.

Moguće je integrisati sve komponente odjednom i onda tek pokrenuti testiranje.

Po nekim autorima - Ovo nije baš dobar pristup za velike sisteme zato što se greška (ako postoji) teško pronalazi, ali je dobar za male sisteme, koji imaju mali broj komponenti.

TESTIRANJE INTEGRACIJE PROGRAMA

Cilj je da se nedostaci u integraciji pronađu što je ranije moguće.

Cilj je da se integracioni test sprovede čim se komponente integrišu.

- ✓ Ako postoje da se nedostaci lakše lociraju
 - ✓ pouzdano se zna da se nedostatak nalazi u komunikaciji komponenti čija se integracija testira

TESTIRANJE INTEGRACIJE PROGRAMA

Testiranje integracije bitan je aspekt testiranja softvera.

Cilj integracionog **testiranja** je da se moduli, koji su jedinično testirani, integrišu, zatim da se pronađu greške, da se te greške uklone i da se izgradi celokupna struktura sistema, kao što je predviđeno dizajnom.

TESTIRANJE INTEGRACIJE PROGRAMA

Integraciono testiranje (ponegde se zove integracija i testiranje i piše se kao oznaka -I&T) je faza u testiranju softvera u kojoj se pojedinačne komponente kombinuju i testiraju kao grupa.

Aplikacija se obično sastoji od nekoliko softverskih modula.

Ove module kodiraju različiti programeri.

Ali kako da znamo da li moduli dobro funkcionišu zajedno?



TESTIRANJE INTEGRACIJE PROGRAMA

- Integracijsko testiranje obično se odvija nakon jediničnog testiranja, koje uključuje testiranje pojedinačnih modula i jedinica.
- Nakon što se utvrdi da svaka jedinica radi zasebno, testiranje integracije procenjuje kako sve jedinice rade u kombinaciji.

TESTIRANJE INTEGRACIJE PROGRAMA

Testiranje integracije je u suštini inkrementacioni proces, koji obično zahteva od testera da integrišu module jedan po jedan i da vrše testiranje u svakom koraku.

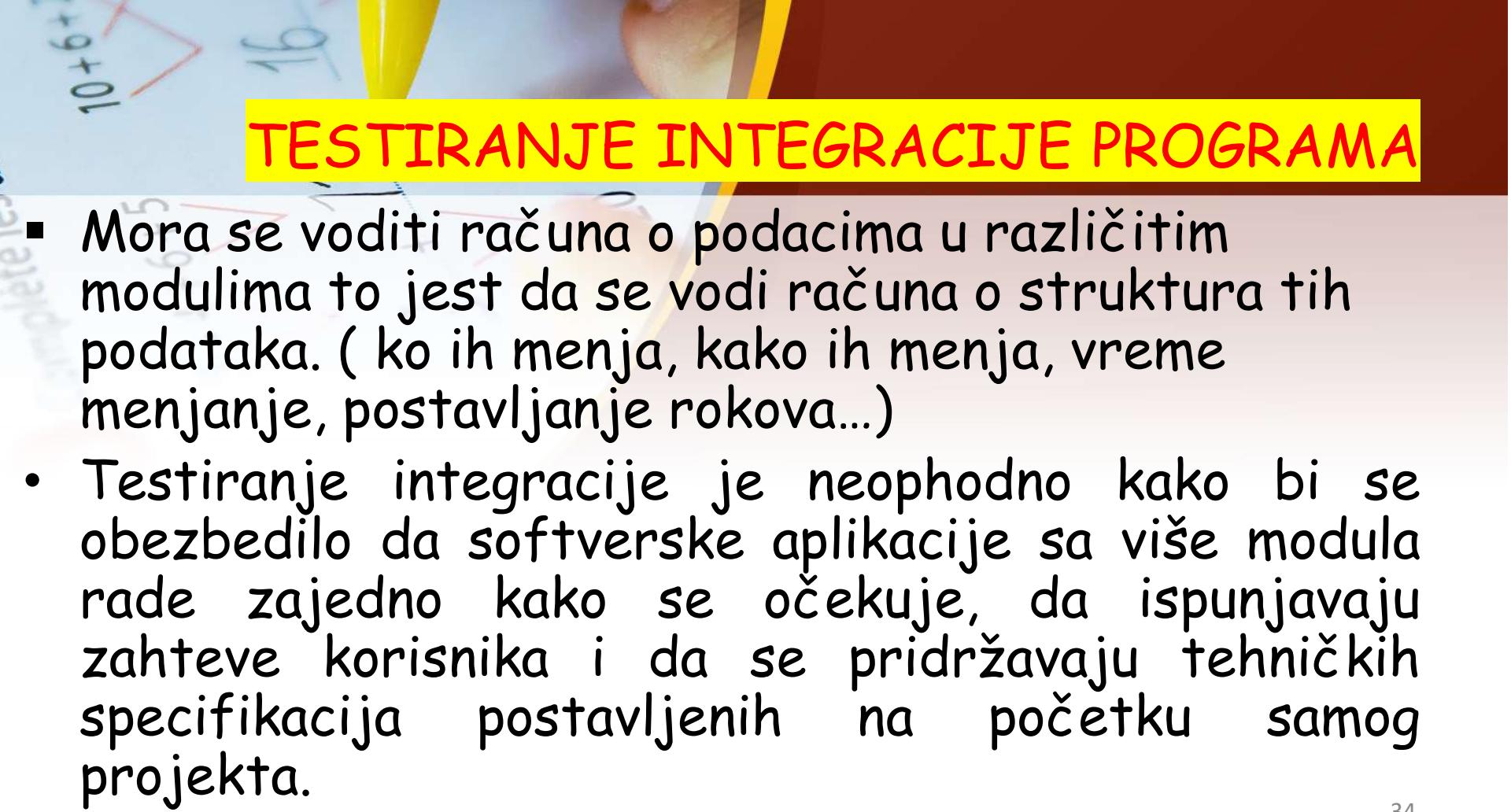
Može da se koristiti i neki dodatni alati za praćenje komunikacije između komponenti

- npr. browser sadrži alat za prikaz podataka koji se razmenjuju koristeći HTTP protokol

TESTIRANJE INTEGRACIJE PROGRAMA

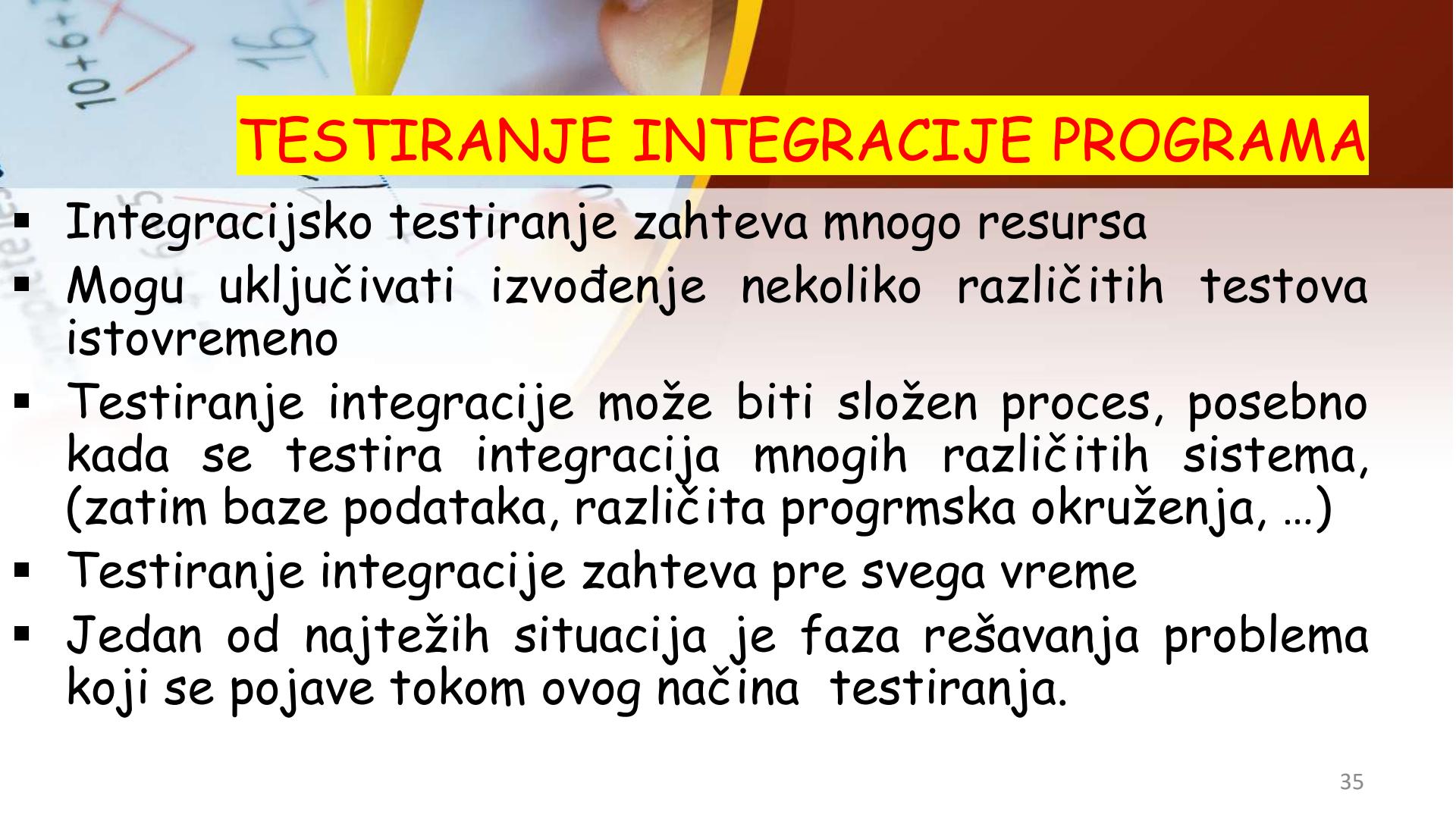
Neki od razloga zašto se testiranje integracije danas koristi su:

- Različiti programeri koriste različitu logiku kada razvijaju module čak i za istu softversku aplikaciju.
- Integracijsko testiranje je jedini način da se zasebni moduli rade zajedno kako se od njih (potencijalno) očekuje .



TESTIRANJE INTEGRACIJE PROGRAMA

- Mora se voditi računa o podacima u različitim modulima to jest da se vodi računa o struktura tih podataka. (ko ih menja, kako ih menja, vreme menjanje, postavljanje rokova...)
- Testiranje integracije je neophodno kako bi se obezbedilo da softverske aplikacije sa više modula rade zajedno kako se očekuje, da ispunjavaju zahteve korisnika i da se pridržavaju tehničkih specifikacija postavljenih na početku samog projekta.



TESTIRANJE INTEGRACIJE PROGRAMA

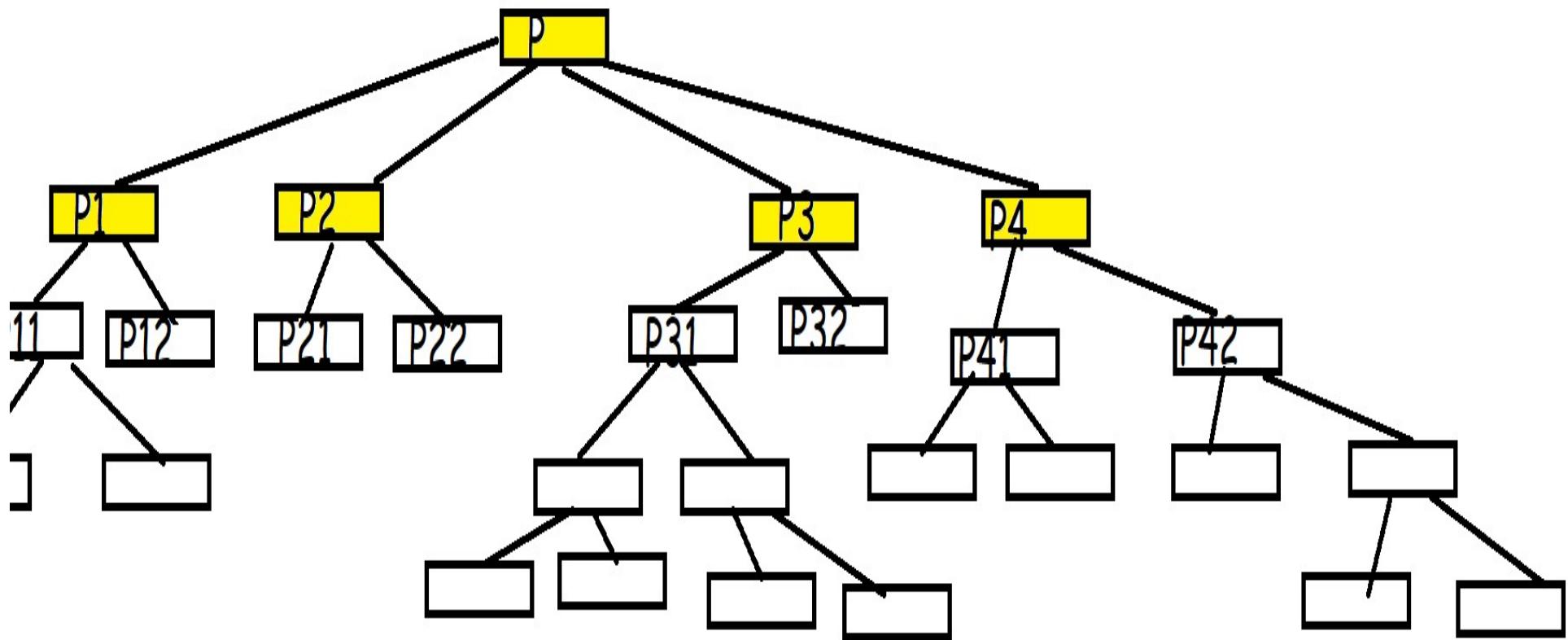
- Integracijsko testiranje zahteva mnogo resursa
- Mogu uključivati izvođenje nekoliko različitih testova istovremeno
- Testiranje integracije može biti složen proces, posebno kada se testira integracija mnogih različitih sistema, (zatim baze podataka, različita programska okruženja, ...)
- Testiranje integracije zahteva pre svega vreme
- Jedan od najtežih situacija je faza rešavanja problema koji se pojave tokom ovog načina testiranja.

TESTIRANJE INTEGRACIJE PROGRAMA

Testiranje integracije može se izvršiti:

- ✓ **Prvi pristup:** Primenom pristupa odozgo nadole.
(Bottom-up testing)
- ✓ **Drugi pristup:** Pristup odozdo prema gore, koji se izvodi sa dna kontrolnog toka. (Top-down testing)
- ✓ **Treći pristup** je Mešovita (Sandwich testing) integracija

TESTIRANJE INTEGRACIJE PROGRAMA



TESTIRANJE INTEGRACIJE PROGRAMA

Integracija od vrha ka dnu.

Prepoznaju se dve podstrategije u dubinu ili u širinu.

Moduli se analiziraju od vrha na kraju.

U top-down testiranju, prvo se testira glavni (main) modul tj. modul na najvišem nivou u softverskoj strukturi; zadnji moduli koji se testiraju su moduli na najnižem nivou.

Glavni modul se koristi kao početni za sve podmodule.

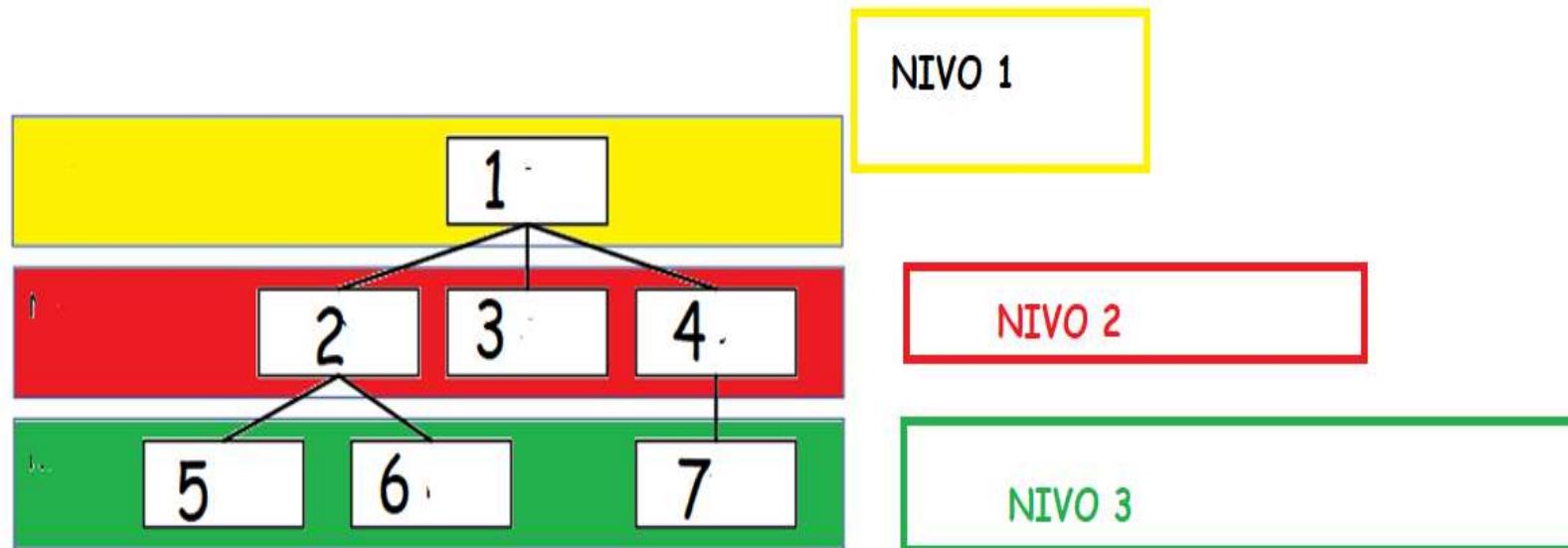
2. Ubacuju se moduli jedan po jedan.
3. Testiranje se vrši posle ubacivanja svakog novog modula.
4. Primjenjuje se i regresiono testiranje (obuhvata neka ili sva prethodna testiranja) da bi se proverilo da li dolazi do novih grešaka

Njihovo rano otkrivanje je vrlo bitno.

TESTIRANJE INTEGRACIJE PROGRAMA

- Integracija odole na gore
 - Integracija kreće sa komponentama na dnu hijerarhije
 - Komponente na nižem nivou hijerarhije koje još nisu implementirane se simuliraju
 - Kako se niže komponente implementiraju tako se verificuje njihov rad stavljanjem u test umesto simuliranih objekata
 - prvo se testiraju moduli na najnižem nivou, a glavni (main) modul se testira zadnji.

Mešovita (Sandwich testing)





- Kombinacija prva dva pristupa (od vrha ka dnu i od dna ka vrhu)
- Test primeri se relativno mogu lako uraditi (za testiranje)
- Komponente se mogu integrisati čim se implementiraju

Mana:

- Greške se teže otkrivaju



Bing bang testiranje

Radi se kada se spoje svi razvijeni moduli.

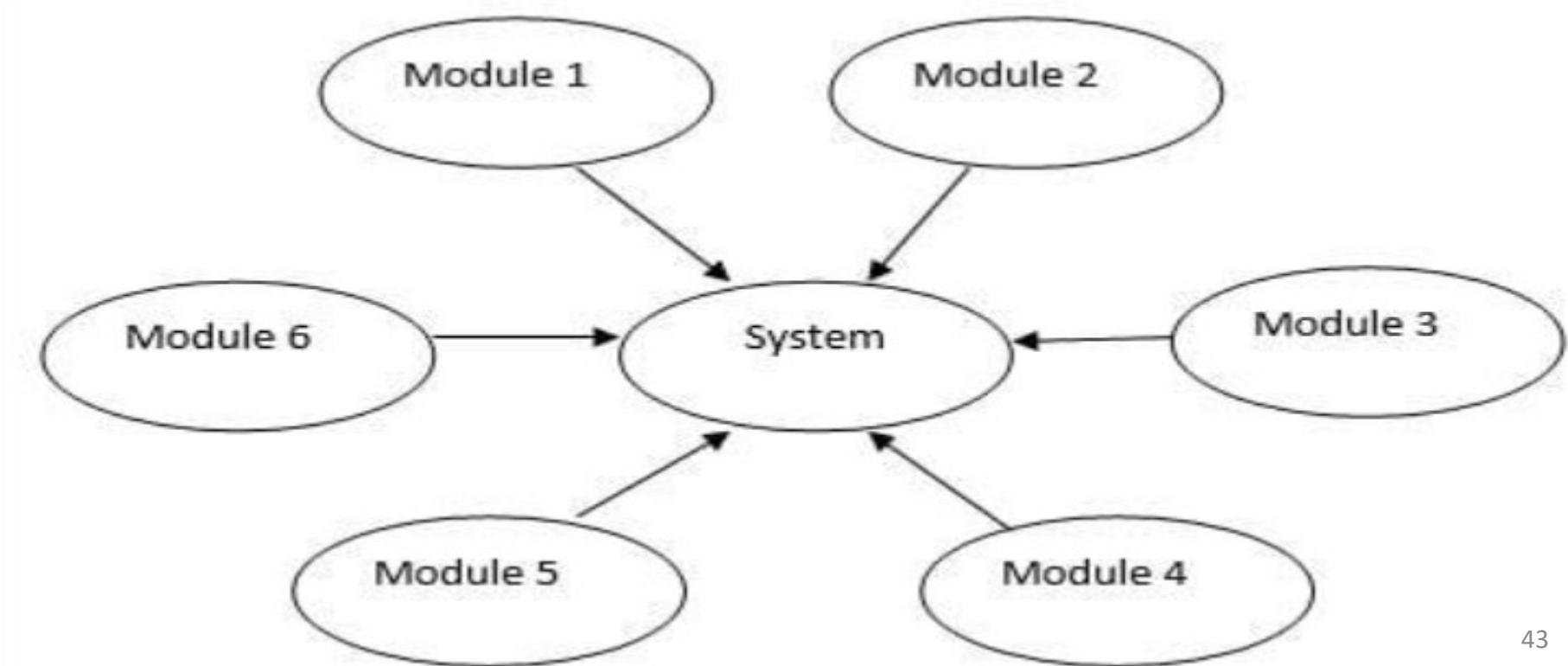
Ako se testni slučajevi i njihovi rezultati ispravno ne zabeleže, proces integracije će biti mnogo komplikovaniji i može čak sprečiti tim za testiranje da ostvari cilj testiranja integracije.

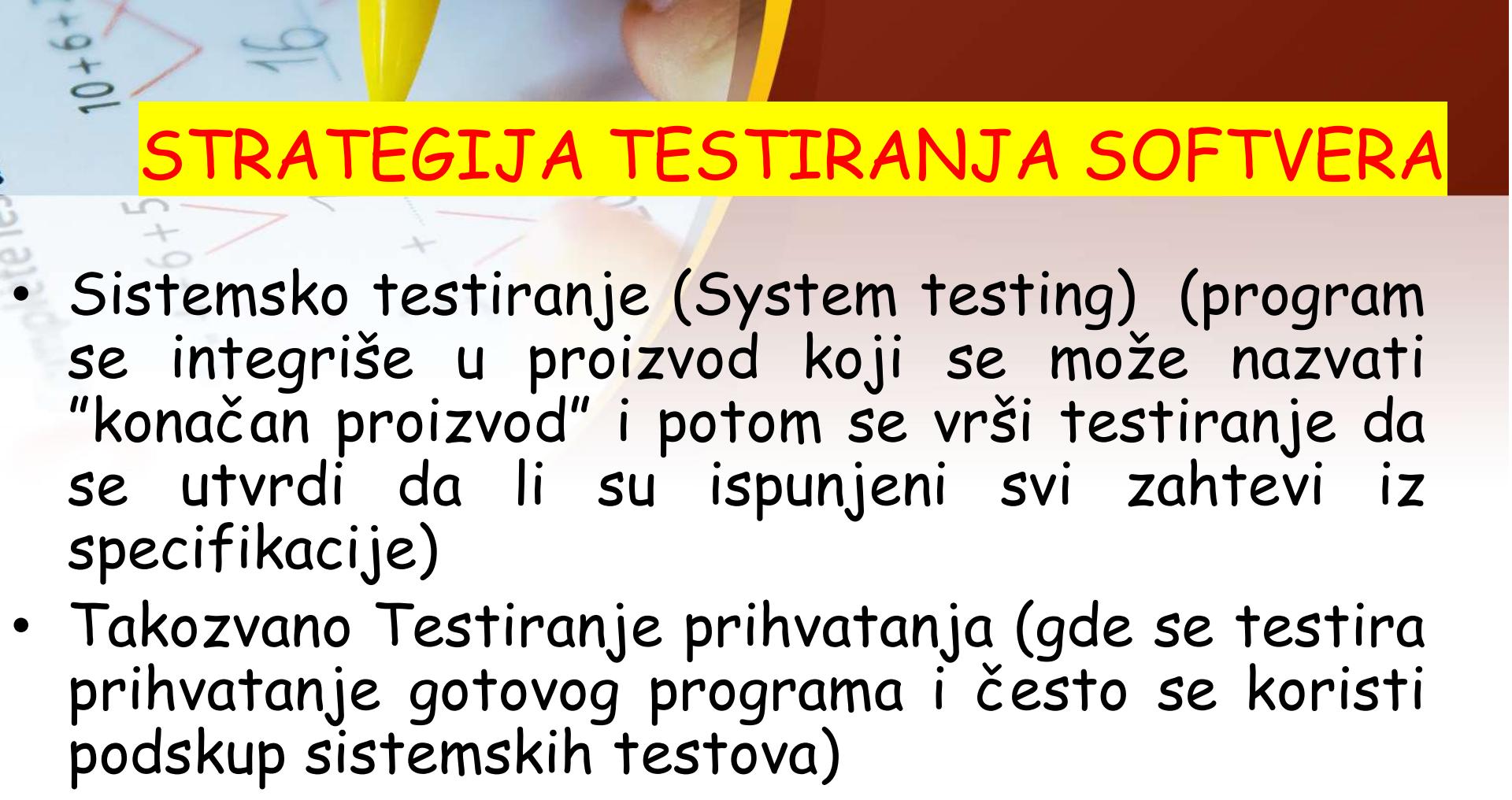
Jedan tip Big Bang testiranja zove se :

Usage Model testiranje.

Usage Model testiranje se može koristiti u testiranju i softvera i hardvera

Bing bang testiranje





STRATEGIJA TESTIRANJA SOFTVERA

- Sistemsko testiranje (System testing) (program se integriše u proizvod koji se može nazvati "konačan proizvod" i potom se vrši testiranje da se utvrди da li su ispunjeni svi zahtevi iz specifikacije)
- Takozvano Testiranje prihvatanja (gde se testira prihvatanje gotovog programa i često se koristi podskup sistemskih testova)



STRATEGIJA TESTIRANJA SOFTVERA

Strategija testiranja predstavlja celokupan pristup testiranju, pri čemu se definišu nivoi testiranja, metode, tehnike i alati.

Ukoliko bi se radilo u idealnim uslovima, strategija testiranja bila bi zajednička za celu kompaniju, kao i za sve programe koje se koriste u okviru same kompanije. Primena i razvoj strategije vrlo su bitni za postizanje određenog kvaliteta programa a samim tim i realizaciju projekta.



SISTEMSKO TESTIRANJE SOFTVERA

Ispituje se da li je ponašanje sistema u skladu sa specifikacijom zadatom od strane klijenta.

Ova vrsta testiranja stavlja naglasak na nefunkcionalne zahteve sistema kao što su brzina, efikasnost, otpornost na otkaze, uklapanje u okruženje u kojem će se sistem koristiti.

Testiranje sistema obavlja se u drugačijim uslovima u odnosu na testiranje jedinica koda i testiranje prihvatljivosti.

U proces testiranja sistema uključuje se ceo razvojni tim pod nadzorom menadžera (rukovodioca) projekta.



SISTEMSKO TESTIRANJE SOFTVERA

- Nakon što se u toku sistemskog testiranja uoče i isprave većina defekata (nikad se mogu otkriti svi), sistem je spreman za isporuku klijentu.
- Sistemsko testiranje je obično dobro za poređenje sistema sa nefunkcionlanim sistemskim zahtevima, kao što su bezbednost, brzina, preciznost ili pouzdanost.
- Klijent onda najčešće inicira Acceptance Testing, odnosno User Acceptance Testing (UAT).

User Acceptance Testing (UAT).

Testirajte softver sa stvarnim korisnicima.

Bez sveobuhvatnog testiranja prihvatljivosti korisnika, tražene promene mogu izgledati gotove kada je stvarni rad još uvek neophodan.

Jedina dostupna metoda za prepoznavanje takvih problema je nakon implementacije kada je klijenti pokušavaju da sami otkriju neke defekte.

Korisnički test (User Acceptance Test)

Acceptance test

Koristi se za identifikaciju nekih bugova u ovoj fazi testiranja softvera.

Koristi se kada naručilac i korisnik nisu iste osobe.

Krajnji korisnik verifikuje da li je sistem upotrebljiv ili nije.

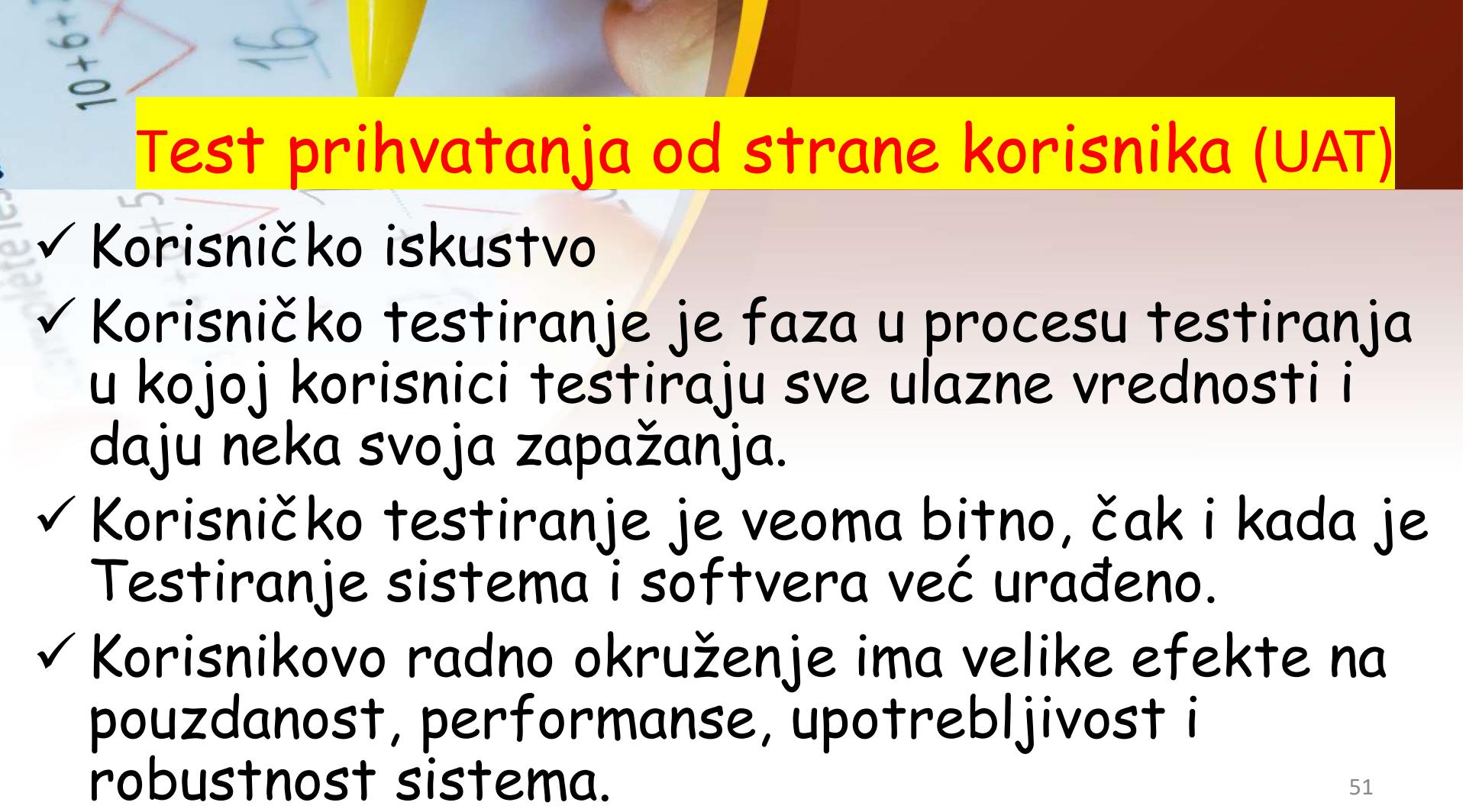
Vrlo često najteži test jer:

Strah od novog i nepoznatog

- Krajnji korisnik ima najmanju sposobnost i interes da svoja očekivanja i zahteve izrazi i formalizuje
- Bez prihvatanja od strane krajnjeg korisnika sistem će teško biti uveden u upotrebu
- Test se najčešće vrši u kasnoj fazi razvoja kada je svaka izmena komplikovana i skupa.

Test prihvatanja od strane korisnika (UAT)

- Test prihvatanja od strane korisnika (User acceptance testing ili Acceptance testing) je testiranje koje vrši klijent kada je sistem spreman za isporuku, nakon što se ispravila većina defekata pronađena u sistemskoj fazi testiranja.
- Cilj ovog testiranja je da klijent stekne poverenje u sistem koji je implementiran.
- Proveravaju se funkcionalnosti prema specifikaciji zahteva i određuje se da li sistem ispunjava potrebe krajnjih korisnika.



Test prihvatanja od strane korisnika (UAT)

- ✓ Korisničko iskustvo
- ✓ Korisničko testiranje je faza u procesu testiranja u kojoj korisnici testiraju sve ulazne vrednosti i daju neka svoja zapažanja.
- ✓ Korisničko testiranje je veoma bitno, čak i kada je Testiranje sistema i softvera već urađeno.
- ✓ Korisnikovo radno okruženje ima velike efekte na pouzdanost, performanse, upotrebljivost i robustnost sistema.

Test prihvatanja od strane korisnika

User Acceptance Testing

Test prihvatanja od strane korisnika je obavezan zbog:

- ✓ U slučaju da su developeri pogrešno protumačili zahteve i samim tim pogrešno implementirali funkcionalnosti
- ✓ Izmene u samoj specifikaciji ne budu iskомуunicirane sa developerima kako treba na pravi način, mogu izazvati velike probleme (posebno ako imamo vremenske rokove i probleme u samom budžetu projekta)

TEST prihvatanja od strane korisnika



User Acceptance Testing (UAT).

Nepisano pravilo:

Potrebe samog korisnika: Treba obratiti pažnja da je akcenat dat na određenom nivou kvaliteta softvera, a ne samo na funkcionalnosti.

Preporuka: Korisnici treba da isprobaju različite uređaje kako bi bili sigurni da se test slučajevi ponašaju isto.

To je faza u procesu testiranja u kojoj sami korisnici aktivno učestvuju.

Potencijalni problem: Odabir tima za testiranje: Tim za testiranje sastoji se od krajnjih korisnika koji vrše testiranje u realnom vremenu i realnom okruženju.

Izvođenje testnih slučajeva i sve se mora dokumentovati.

UAT je jedan od poslednjih i (mnogi autori navode kao) jedan od najkritičnijih postupaka softverskog projekta. On se mora testirati pre nego što se novo razvijeni softver - bude pušten u produkciju.



Alfa testiranje

Alfa testiranje je vrsta testiranja prihvatljivosti softvera.

Ono se koristi kako bi se identifikovali svi mogući problemi pre objavljanja proizvoda.

Fokus ovog testiranja je simuliranje rada stvarnih korisnika.

Tada se testiraju sve opcije u softveru koje tipični korisnik može uraditi u stavrnom okruženju i samom načinu rada.



ALFA I BETA TESTIRANJE SOFTVERA

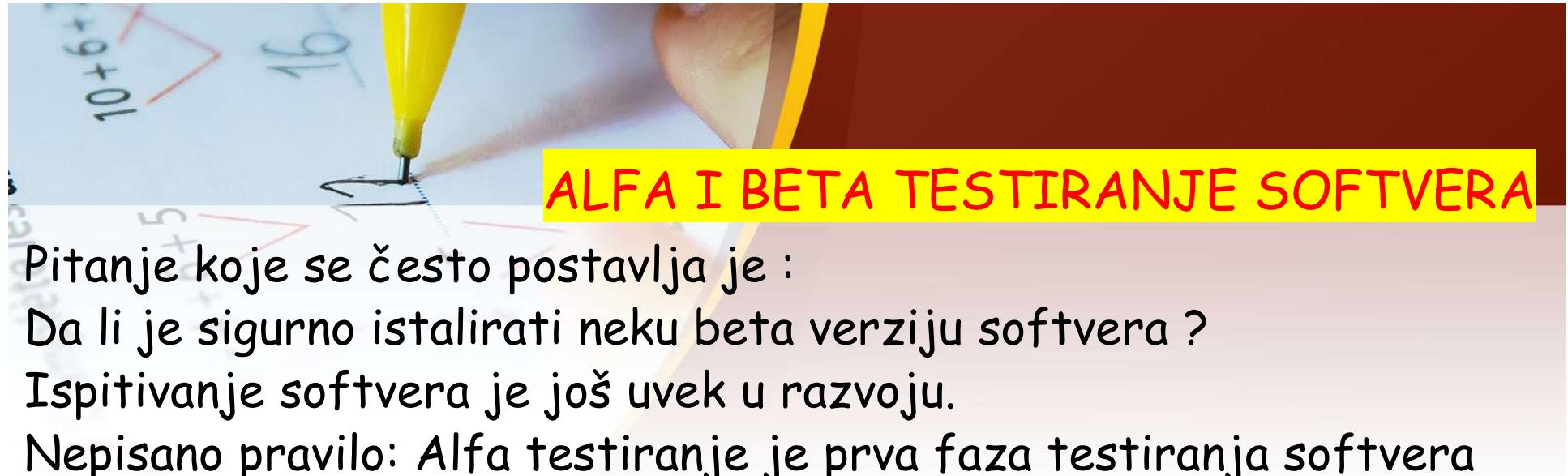
Alfa testiranje se izvršava u stvarnom okruženju i obično su testeri interni članovi. Ova vrsta testiranja naziva se alfa jer se koristi relativno rano, pri kraju razvoja programskog rešenja i pre beta testiranja.

Po nekom nepisanom pravilu, alfa testiranje se vrši u najranijoj fazi razvoja softvera, međutim, u nekim se slučajevima može primeniti na završenu ili blizu završetka softvera.

Beta testiranje obavljaju stvarni korisnici programskog rešenja u "stvarnom okruženju" i može se smatrati oblikom eksternog korisničkog testiranja prihvatljivosti.

Drugi naziv Field Testing - beta testera

Beta verzija programskog rešenja objavljuje se sa ograničenim brojem krajnjih korisnika kako bi se dobila povratna informacija o kvalitetu samog projekta (proizvoda.). To je konačni test pre slanja, (postavljanja na web sajt) ka krajnjim korisnicima



Pitanje koje se često postavlja je :

Da li je sigurno instalirati neku beta verziju softvera ?

Ispitivanje softvera je još uvek u razvoju.

Nepisano pravilo: Alfa testiranje je prva faza testiranja softvera nakon razvoja.

Beta testiranje se vrši nakon što softver prođe alfa testiranje.

Beta verzija softvera se isporučuje korisnicima, koji ga instaliraju i koriste u realnim uslovima.

Obično se beta testiranje vrši u dve faze: zatvoreni beta test i otvoren beta test.



- Cilj: Testirati sve delove softvera,
 - Pronaći sto je više grešaka pre same produkcije,a to je i promarni cilj testiranja softvera
- Softver dobijaju stvarni korisnici pre produkcije.
- Korisnici u toku beta perioda mogu testirati aplikaciju i slati feedback,
- Beta testeri često pronađu greške koje nisu do tada primećene (naročito iz korisničke perspektive), i moguće je popraviti te probleme pre produkcije. (ako su te primedbe opravdane)
- Što se više ovakvih grešaka ispravi, veći je kvalitet softvera nakon produkcije
- Kranji cilj: Zadovoljstvo korisnika (customer satisfaction)

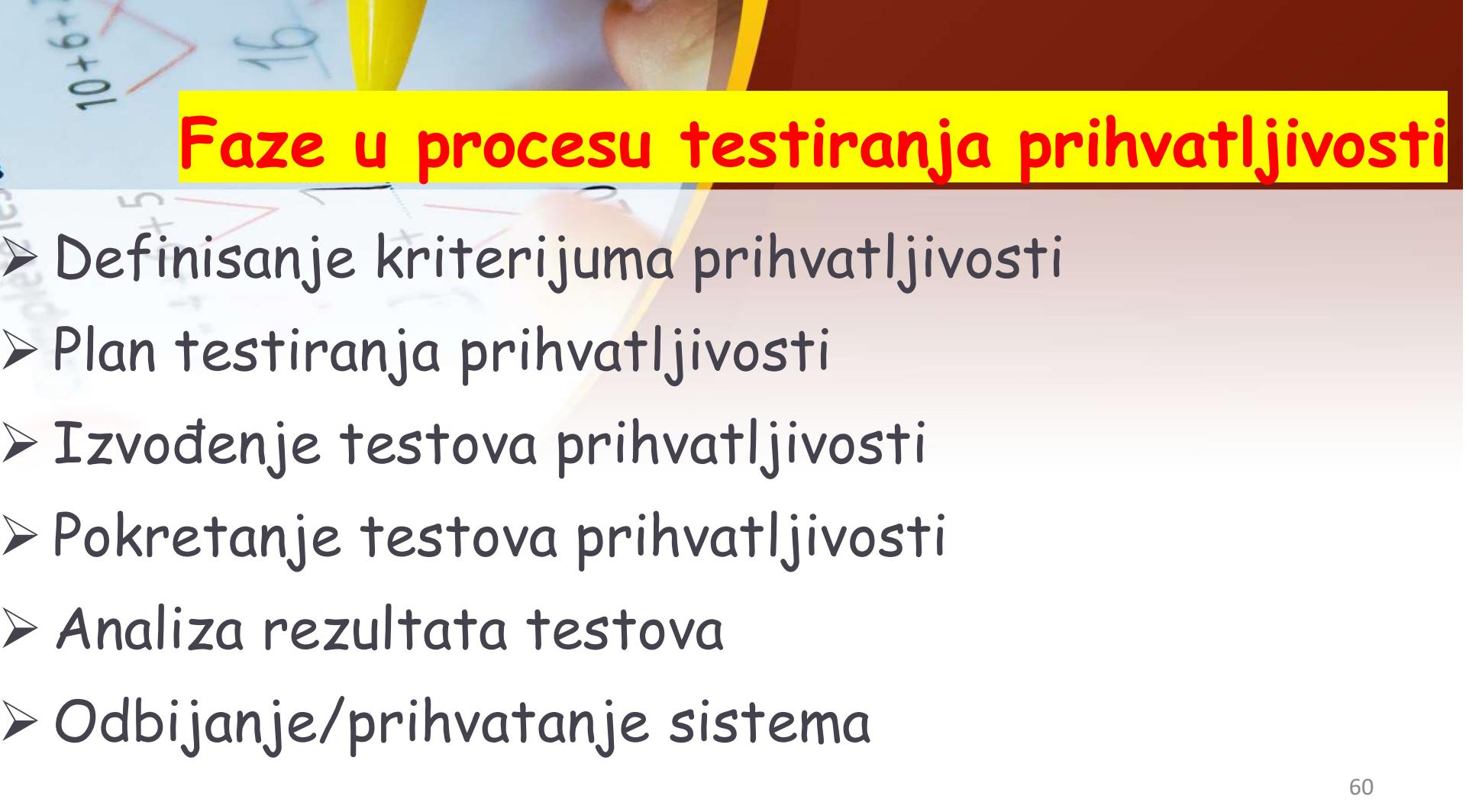


Buddy testiranje

Buddy testiranje je tehnika testiranja na kojem su angažovana obično dva člana tima koji rade na istom programskom rešenju na istom uređaju.

Jedan od članova tima će raditi sa raznim scenarijima da li nešto radi ili ne a drugi će sve to unositi u dnevnik testiranja.

Jedan član tima trebao bi biti tester, a drugi član razvojnog tima može biti analitičar.



Faze u procesu testiranja prihvatljivosti

- Definisanje kriterijuma prihvatljivosti
- Plan testiranja prihvatljivosti
- Izvođenje testova prihvatljivosti
- Pokretanje testova prihvatljivosti
- Analiza rezultata testova
- Odbijanje/prihvatanje sistema



PLAN TESTIRANJA

- Plan testiranja prihvatanja koji podrazumeva plan testiranja prihvatljivosti programa. Ukoliko plan testiranja ne predstavlja poseban dokument, on je povezan sa planom sistemskog testiranja.
- Plan sistemskog testiranja koji podrazumeva plan integrisanja i testiranja sistema. Uključen je u plan testiranja prihvatljivosti ukoliko ne predstavlja poseban dokument.



Evaluacija rezultata Testiranja softvera

- Na kraju neke faze testiranja radi se evaluacija rezultata testiranja.
- U ovoj fazi treba definisati mere (metrike) za evaluaciju kvaliteta testiranja i realizovati Izveštaj o proceni kvaliteta testiranja.
- U ovoj fazi se može uraditi i analiza pronađenih defekata sa ciljem da se ukaže na česte i zajedničke greške razvojnog dela tima, da se preporuče naknadne aktivnosti (kursevi, doobuka korisnika, promena tehnologije,).
-

SISTEMSKO TESTIRANJE SOFTVERA

- end to end test

Sistemsko testiranje bazira se na testiranju kompletног softvera, da bi se proverilo da li sve komponente rade zajedno kao celina (end to end test), to znači da se softver testira u realnom sistemu korišćenja.

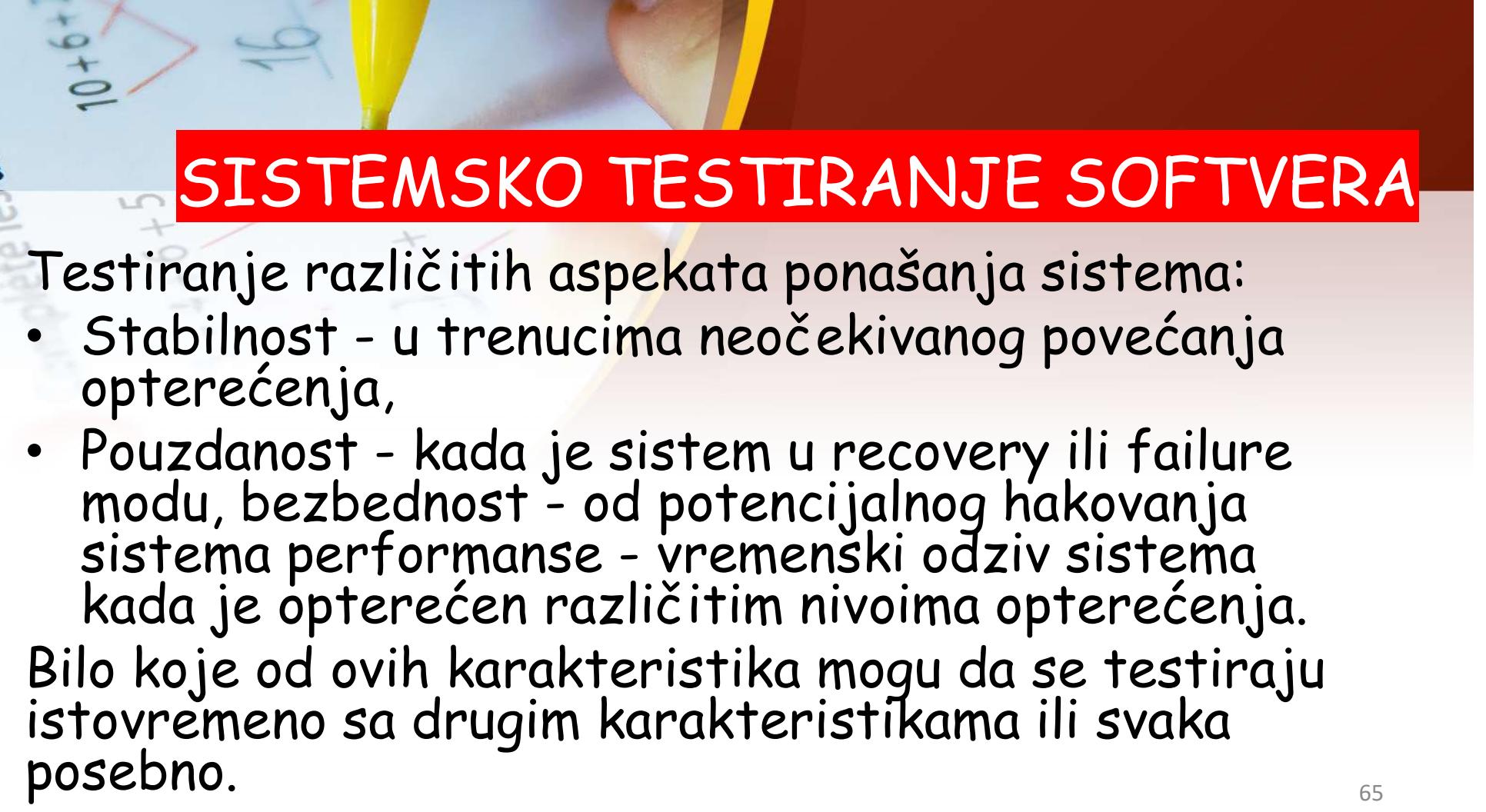
Sistemski testovi testiraju potpuno integrисани sistem da bi se potvrdило da sistem ispunjava sve zahteve korisnika.

SISTEMSKO TESTIRANJE SOFTVERA

- Cilj je testirati funkcionalne i nefunkcionalne zahteve.
- Treba utvrditi u kojim delovima sistem netačno, nepotpuno ili nekonzistentno zadovoljava zahteve

Takođe, treba utvrditi i zahteve koji su:

- Neevidentirani
- zaboravljeni u samoj specifikaciji zahteva. (u praksi se može desiti da takvih zahteva ima jako puno)

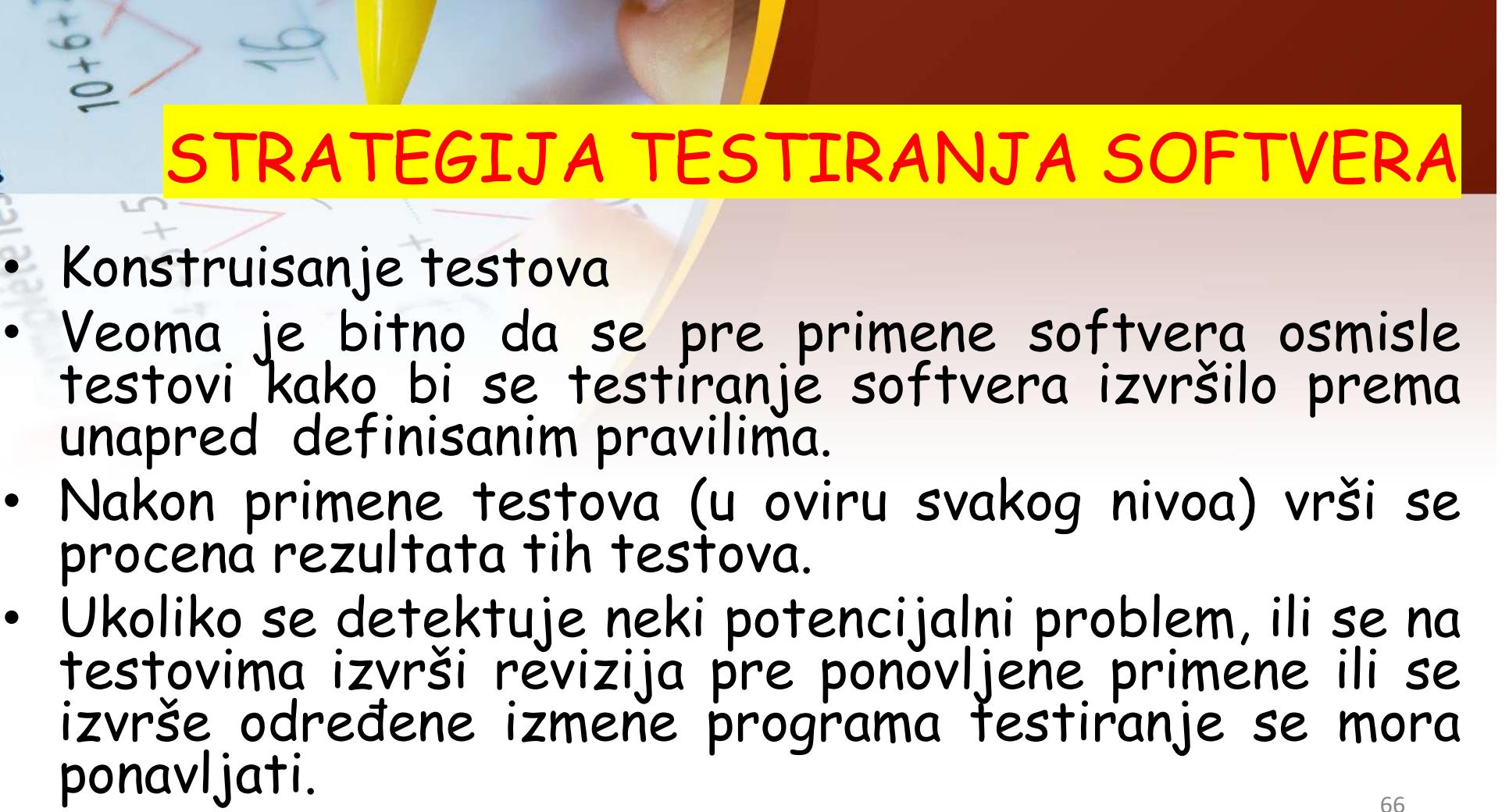


SISTEMSKO TESTIRANJE SOFTVERA

Testiranje različitih aspekata ponašanja sistema:

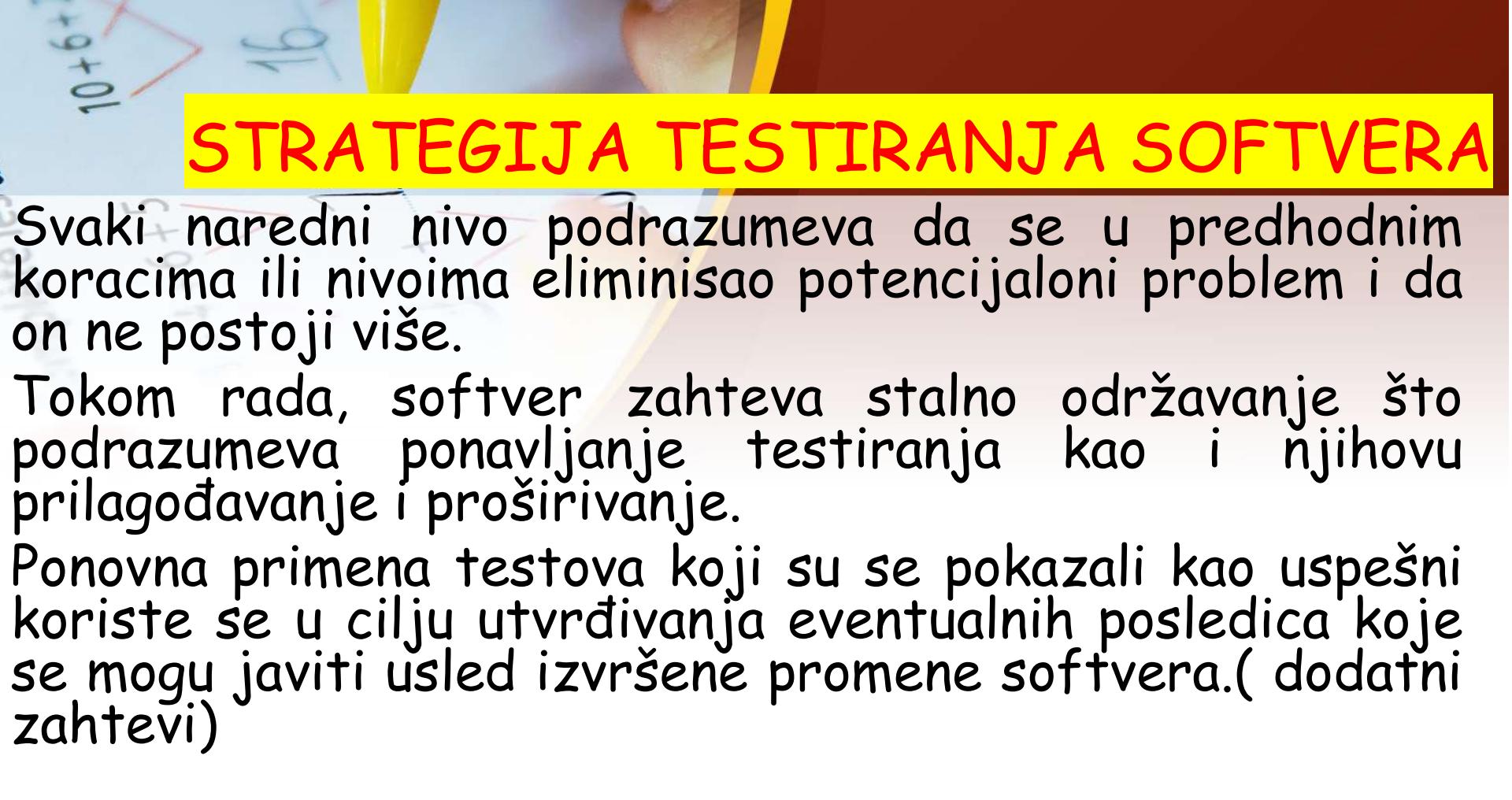
- Stabilnost - u trenucima neočekivanog povećanja opterećenja,
- Pouzdanost - kada je sistem u recovery ili failure modu, bezbednost - od potencijalnog hakovanja sistema performanse - vremenski odziv sistema kada je opterećen različitim nivoima opterećenja.

Bilo koje od ovih karakteristika mogu da se testiraju istovremeno sa drugim karakteristikama ili svaka posebno.



STRATEGIJA TESTIRANJA SOFTVERA

- Konstruisanje testova
- Veoma je bitno da se pre primene softvera osmisle testovi kako bi se testiranje softvera izvršilo prema unapred definisanim pravilima.
- Nakon primene testova (u oviru svakog nivoa) vrši se procena rezultata tih testova.
- Ukoliko se detektuje neki potencijalni problem, ili se na testovima izvrši revizija pre ponovljene primene ili se izvrše određene izmene programa testiranje se mora ponavljati.



STRATEGIJA TESTIRANJA SOFTVERA

Svaki naredni nivo podrazumeva da se u predhodnim koracima ili nivoima eliminisao potencijalni problem i da on ne postoji više.

Tokom rada, softver zahteva stalno održavanje što podrazumeva ponavljanje testiranja kao i njihovu prilagođavanje i proširivanje.

Ponovna primena testova koji su se pokazali kao uspešni koriste se u cilju utvrđivanja eventualnih posledica koje se mogu javiti usled izvršene promene softvera.(dodatni zahtevi)



STRATEGIJA TESTIRANJA SOFTVERA

Testovi na najvišem nivou se razrađuju tako što se odredi da li je i koliko softver verodostojan u primeni zahteva iz specifikacije.

Na nižim nivou, testovi se razrađuju u pravcu provere da li testovi zadatog softvera uključuju sve zahteve (detalje) specifikacije i njegove arhitekture.

Podrazumeva se da je potrebno i zvanično i nezvanično proveravanje svake faze razvoja testova.

Testovi se razvijaju u onolikoj meri u kojoj se razvijaju i programi.

STRATEGIJA TESTIRANJA SOFTVERA

Posle specifikacije sledeći korak je dokumentacija za testiranje.

Testovi koji su definisani kao kvalitetni testovi sastoje se iz više faza.

Cilj je da razrade sistem testova, kako bi sve bilo urađeno do detaljnih postupaka testiranja.

Faze su poznate kao:

- Strategija testiranja,
- Planiranje testiranja,
- Izrade test primera i
- Razrada samog postupka testiranja

Svaki test se razrađuje na osnovu programske specifikacije.

STRATEGIJA TESTIRANJA SOFTVERA

Tek sada možemo napraviti plan testiranja programa

Planom testiranja programa obuhvaćeni su segmenti programa koji će se testirati i u kojoj fazi razvoja, redosled testiranja, na koji način će se strategija testiranja primenjivati na svaku jedinicu i konačno tip testiranja programa.

Kao i za ostale faze testiranja, tako i za plan testiranja važi pravilo koje podrazumeva da on bude dat za ceo projekat ili u drugom slučaju da to bude čitava hijerarhija planova za različite nivoje specifikacije i testiranja.



PLAN TESTIRANJA

Plan testiranja materijalizuje strategiju testiranja, opisuje resurse koji će se koristiti, samo okruženje, odnosno okruženje u kojem će se testiranje izvršiti. Treba voditi računa o ograničenjima (ako postoje) koja će se primeniti.

Takođe treba voditi računa o izvođenja testova, kao i o njihovom redosledu.

Dnevnik testiranja

Planovi testiranja su različiti za različite nivoe.



STRATEGIJA TESTIRANJA SOFTVERA

- Plan testiranja integracije programa: u okviru kojeg su integrisane sve programske komponente. U nekim slučajevima on je deo specifikacije arhitekture.
- Plan testiranja jedinica podrazumeva plan za testiranje svake komponente ponaosob.
- Može biti deo detaljnih specifikacija.

STRATEGIJA TESTIRANJA SOFTVERA

- Realizacijom plana testiranja postoji mogućnost potvrde specifikacije proizведенog softvera. U okviru programske specifikacije:
 - testiranje prihvatljivosti i testiranje sistema.
 - Sledeća faza jeste razvoj test stavki



STRATEGIJA TESTIRANJA SOFTVERA

- Razvoj testa nakon napravljenog plana testiranja za određeni nivo zahteva određivanje skupa test stavki ili test pitanja za svaki objekat koji je obuhvaćen testom na datom nivou. Svaki objekat sadrži određeni broj test stavki koji se može odrediti a samim tim i testirati na svakom nivou.
- Test stavke mogu biti u obliku posebnog dokumenta (specifikacija testiranja ili opisane u obliku samog teksta)

STRATEGIJA TESTIRANJA SOFTVERA

Specifikacija testiranja prihvatanja (kao poseban dokument ili u okviru plana testiranja sistema) određuje test stavke za integraciju i testiranje sistema.

Specifikacija testiranja integracije programa u okviru specifikacije arhitekture određuje test stavke za svaku fazu integracije programskih komponenti koje se testiraju.

STRATEGIJA TESTIRANJA SOFTVERA

- Specifikacija testiranja jedinice (kao deo detaljnih specifikacija) određuje test stavke za testiranje komponenti ponaosob.
- Za uspešno sistemsko testiranje i testiranje prihvatljivosti potreban je veliki broj pojedinačnih test stavki. Indeks takozvanih "unakrsnih referenci" za zahteve i test stavke kao deo specifikacije testiranja neophodan je radi utvrđivanja koji zahtevi su testirani sa kojim test stavkama.

STRATEGIJA TESTIRANJA SOFTVERA

Positive and negative testing. Jako bitno testiranje softvera.

- Testiranje pozitivnih i negativnih vrednosti svih ulaza.
Razlike između pozitivnog i negativnog testiranja se sastoje u tome što se kod pozitivnog testiranja proverava da li program radi ono što treba, a kod negativnog testiranja da li program radi nešto što ne bi trebalo da radi.
- U oba slučaja je bitno odrediti test stavke.

STRATEGIJA TESTIRANJA SOFTVERA

Primena skupa test stavki, kao poslednja faza razvoja testova predstavlja u određenoj meri relativan postupak, i u tom smislu se poredi sa razvojem programskih jedinica sa višeg nivoa funkcionalnog opisa.

STRATEGIJA TESTIRANJA SOFTVERA

- Postupci koji se vrše za svaku test stavku moraju biti unapred određeni.
- Bez obzira koji se postupak testira postupak testiranja je odlučujući pri odabiru postupka uvođenja test stavki.
- Prilikom sprovođenja postupka testiranja, svaki korak se mora ispoštovati, pri čemu se ne sme oslanjati na pretpostavke.

STRATEGIJA TESTIRANJA SOFTVERA

- Poslednja faza jeste dokumentacija rezultata testova.
- Sprovodenjem testova do kraja , za svako izvršenje testa.
- Sve izlazne vrednosti se dokumentuje u određenom fajlu u kome su smešteni rezultati testa.
- U cilju procene uspešnosti testiranja, tj. da bi se utvrdio ishod testiranja, rezultati iz datog fajla se uporede sa kriterijumima zadatim iz same specifikacije.

STRATEGIJA TESTIRANJA SOFTVERA

- Dnevnik testiranja je neophodan kako bi se dokumentovali načini izvršenja testa.
- Takva vrsta dnevnika sadrži osnovne podatke vezane za to kad je koji test izvržen, za ishod izvršenja kao i osnovna zapažanja tokom testiranja.
- Vođenje dnevnika testiranja prilikom testiranja jedinice i integracije programa nije uvek uobičajen, ali je sugestija da se mora imati.

STRATEGIJA TESTIRANJA SOFTVER

- Pregled rezultata testiranja i dokumentacija analiza sadržani su u izveštaju o testiranju.
- Izveštaj se daje u bilo kojoj fazi testiranja.
- Izveštaj o testiranju prihvatljivosti je često u formi ugovora i njime je utvrđena saglasnost o prihvatanju programa.

STRATEGIJA TESTIRANJA SOFTVERA

Kod testiranja softvera najbitnije je odrediti niz test stavki za ono što se testira.

Pre toga moramo odrediti koje informacije se moraju nalaziti u test stavkama.

STRATEGIJA TESTIRANJA SOFTVERA

- Najočitija informacija su ulazi kojih ima dve vrste: preduslovi (okolnosti koje postoje pre izvršenja test stavke) i stvarni ulazi, koje identifikujemo nekim metodom testiranja.

Mnogo je efikasnije dobro razmisliti o vrstama grešaka koje su najverovatnije (ili najštetnije) i tada odabratи metode testiranja koji će verovatno moći da otkriju takve greške.

Nema smisla testirati greške koje verovatno ne postoje.





Mnogi autori u knjigama i naučnim časopisima tvrde sledeće: "U svetu se danas (godišnje) u softverske projekte uloži jako puno novčanih sredstava. (+ VREME)" Iako se tehnologije svakim danom sve više i više razvijaju, odnosno alati za razvoj softvera se razvijaju svaki dan, i dalje je značajan broj grešaka i otkaza koji se javljaju kod gotovih softverskih proizvoda.

CILJ:

Otkriti što više grešaka u što kraćem roku.

Problem vremensko ograničenje i vremenski rokovi

POZITIVNI I NEGATIVNI TEST

POZITIVNI TEST

NEGATIVNI TEST

Pozitivni test

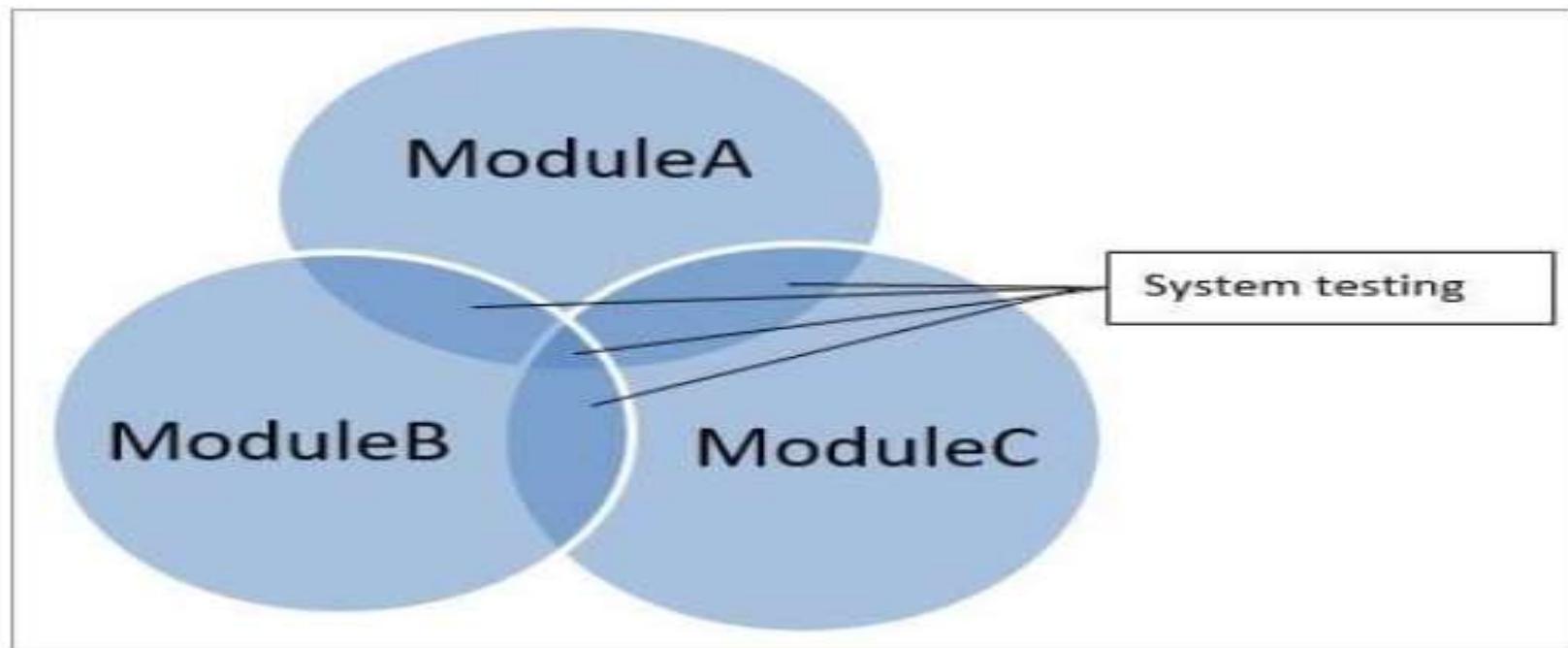
Kada korisnik pažljivo i tačno unosi podatke odnosno ispoštuje sva pravila i unese sve podatke na predviđeni način ovaj test proverava da li je rezultat u skladu sa očekivanim.

Jedan scenario može da ima desetine poslovnih pravila koja moraju da budu zadovoljena. Dobra praksa je testiranje što većeg broja test slučajeva, kako bi se na vreme identifikovali svi bagovi.

Negativni test

predstavlja alternativna scenarija u kojima je osnovna pretpostavka da neće sve da prođe kao što je očekivano.

SISTEMSKO TESTIRANJE SOFTVERA





TESTIRANJE SOFTVERA

Jedan od ideja kvaliteta proizvoda je težnja da taj proizvod bude izrađen u skladu sa specifikacijom (Crosby još 1979).

Međutim postoji nekoliko specifičnosti koje kvalitet softvera odvajaju od kvaliteta ostalih "klasičnih" proizvoda

- Specifikacija svakog proizvoda ide ka ispunjavanju želja i potreba korisnika.
- Međutim, kod razvoja softvera često postoje zahtevi koji nisu izričito specificirani (na primer zahtev za stabilnošću softvera).
- Nepostoji pouzdan način za kvantifikaciju pojedinih karakteristika softvera (na primer: stabilnost, pouzdanost).
- Softverska specifikacija je često nekompletna.



Kvalitet softvera je više dimenzionalni koncept koji se ne može jednostavno definisati.

Potrebno je pratiti različite parametre i obezbediti kvalitet softvera, kreirati planove vezane za kvalitet i ostvariti potrebnu kontrolu.

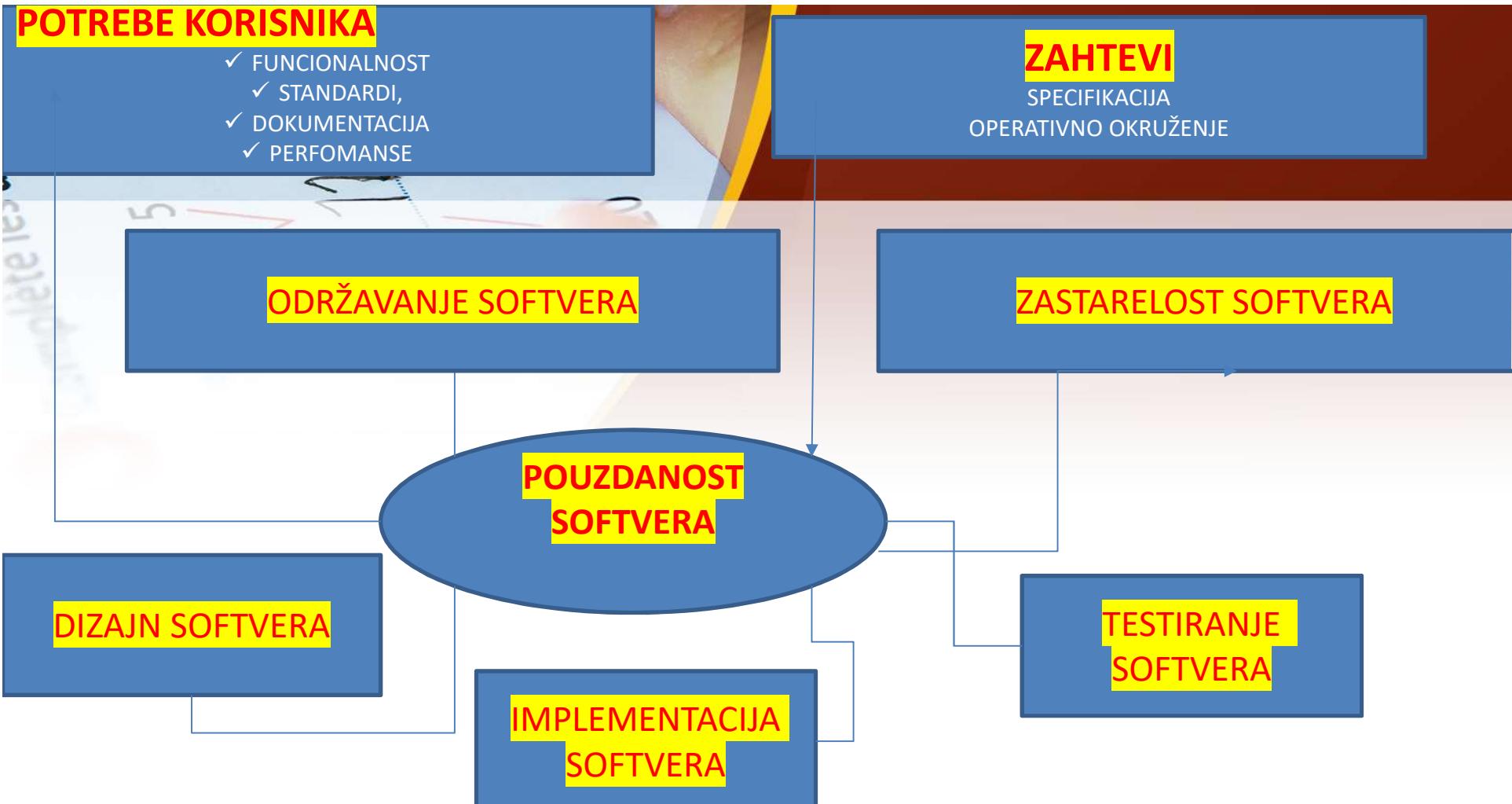
Potrebno je razviti i implementirati standarde sistema i dokumentaciju sistema kvaliteta kvaliteta koja će biti primenjena na softverske proizvode.



U principu jedna od ideja kvalitetnog proizvoda je težnja da taj proizvod bude izrađen u skladu sa specifikacijom (postavljen od Crosby 1979).

Pod kvalitetom u širem smislu podrazumevamo skup svih osobina jednog proizvoda koje zadovoljavaju propisane uslove u pogledu:

- namene proizvoda
- njegove funkcije i pouzdanosti
- jednostavnosti i ekonomičnosti u upotrebi
- održavanja





POUZDANOST

- ✓ Mere konačnog proizvoda softvera govori o kompleksnosti softvera.
- ✓ Mere upravljanja projektom
- ✓ Mere procesa izrade softvera
- ✓ Mere potencijalnih grešaka i otkaza.
- ✓ Mere zahteva za pouzdanost



Među važne osobine kvaliteta softvera su:

- funkcionalnost - u kojoj meri softver zadovoljava specifikacije i potrebe korisnika
- pouzdanost - frekvencija i težina otkaza, sposobnost oporavka, predvidivost, tačnost, srednje vreme između otkaza,...
- upotrebljivost - meri se naporom koji zahteva učenje programa, njegovo korišćenje, priprema ulaza i interpretacija izlaznih rezultata
- efikasnost - brzina odziva, raspoloživost, brzina oporavka, iskorišćenje resursa
- stepen podrške - mogućnost testiranja, proširivanja, prilagođavanja, konfigurisanja, instaliranja.



Provera kvaliteta softvera pruža brojna rešenja ključnim uzrocima problema razvoja softvera:

- Procena statusa razvojnog projekta se izrađuje objektivno, ne subjektivno (vrednuju rezultati testa)
- Objektivno procenjivanje otkriva nekonzistentnosti u zahtevima, dizajnu i implementaciji
- Testiranje i verifikacija su usmereni na oblasti najvišeg rizika
- Greške se otkrivaju rano, (smanjuju se troškovi njihovog ispravljanja)



ROBUSNOST SOFTVERA

Pojam robusnost (u računarstvu) je sposobnost računarskog sistema da se nosi sa greškama tokom izvršavanja i da se nosi sa pogrešnim ulazima.

Formalne tehnike, kao što je faz-testiranje, su od suštinskog značaja za pokazivanje robusnosti jer ova vrsta testiranja uključuje nevažeće ili neočekivane ulaze.

Razni komercijalni proizvodi vrše testiranje softverskih sistema koristeći robusnost i obično je sastavni deo analize procene neuspeha.



ROBUSNOST SOFTVERA

U principu, izgradnja robustnih sistema koji obuhvataju svaku tačku mogućih grešaka je teška zbog velike količine mogućih ulaznih i izlaznih kombinacija.

Pošto bi sve ulazne i izlazne kombinacije zahtevale previše vremena za testiranje, programeri ne mogu da prolaze kroz sve slučajeve detaljno.

Umesto toga, programer će pokušati da uopšti takve slučajeve.



Ocena kvaliteta realizacije softvera

Softver se podvrgava rigoroznim testovima pre izlaska na tržište. Treba da se vodi računa i :

- Da se istakne razlike između prvobitnog koncepta i konačnog izlaza
- Da se verifikujte da softver funkcioniše onako kako su projektanti planirali, problem specifikacije
- Da se validira krajnji proizvod - proizvod mora zadovoljiti zahteve korisnika
- Da se procene karakteristike i kvalitet samog softvera

Ocena kvaliteta realizacije softvera

Svi nedostaci se ispravljaju, a softver ide kroz testiranje regresije - sistem za proveru da li program i dalje funkcioniše nakon modifikacija.

Izveštaj i dobra dokumentacija su jako bitni!

Kvalitet je nepostojanje nedostataka. (Juran 1988)



Kvalitet softvera

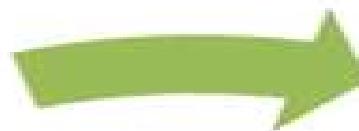
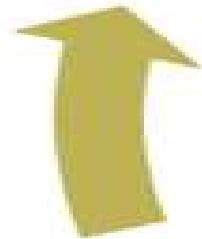
Cilj - kvalitetan softver.

Definicija preporučena od strane IEEE 610.12 (IEEE Standard Glossary of Software Engineering Terminology) je:

1. Stepen u kojem sistem, komponente, ili procesi zadovoljavaju specifične zahteve.
2. Stepen u kojem sistem, komponente ili procesi zadovoljavaju klijentove ili korisnikove potrebe ili očekivanja.

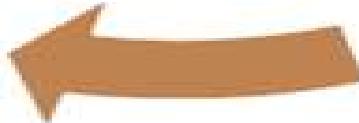
**potrebnih
softverskih
podataka**

kvaliteta softvera



dokumentacije i

procedura,





- Software Testing



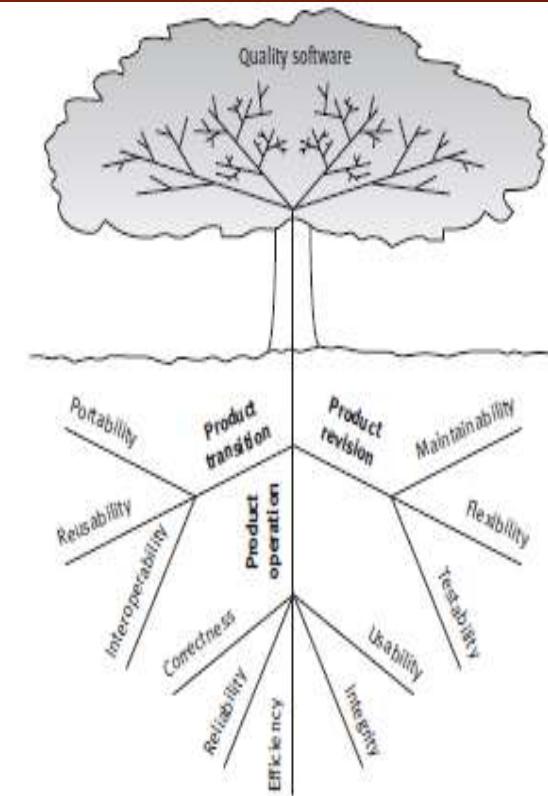
McConnell u knjizi CodeComplete

McConnell u knjizi CodeComplete, istakao 11 faktora za kvalitetan softver o onih je grupisao u 3 kategorije:

-**Faktori proizvoda (softvera)**: korektnost (correctness), pouzdanost (reliability), efikasnost (efficiency), celovitost (Integrity), iskoristivost (usability).

-**Faktori revizije proizvoda (softvera)**: održavanje (maintainability), fleksibilnost (flexibility), pogodan za testiranje (testability).

-**Faktori tranzicije proizvoda (softvera)**: portabilnost (portability), ponovna iskorišnjenost (reusability), interoperabilnost (interoperability).

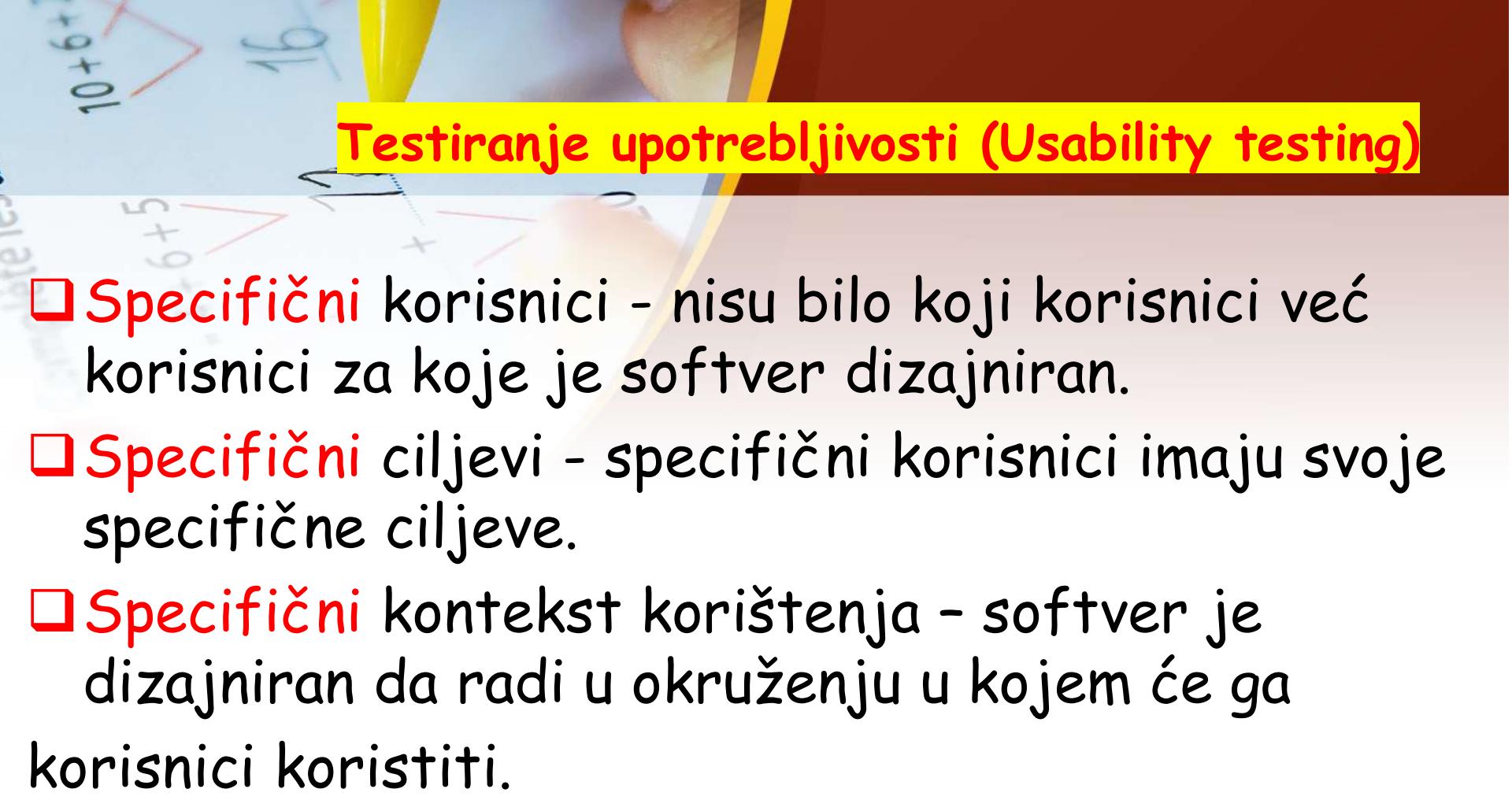




Principi testiranja softvera

Neki važniji principi testiranja softvera su:

- ✓ Svi testovi se moraju pratiti do projektnih zahteva
- ✓ Testovi se moraju planirati znatno pre samog testiranja
- ✓ Važi Pareto princip (80:20 - 80% efekata potiče od 20% uzroka)
80% svih grešaka koje se ne otkriju prilikom testiranja verovatno se odnosi na svega 20% programskih komponenti
- ✓ Testiranje počinje od malih delova i napreduje ka celini
- ✓ Iscrpno testiranje nije moguće (složenost)
- ✓ Testiranje je efikasno ako ga vrši neko drugi (a ne testerski tim i programeri koji su radili na projektu)
- ✓ Resursi za testiranje su ograničeni (npr. vreme)
- ✓ U toku testiranja softver ne treba menjati



Testiranje upotrebljivosti (Usability testing)

- Specifični** korisnici - nisu bilo koji korisnici već korisnici za koje je softver dizajniran.
- Specifični** ciljevi - specifični korisnici imaju svoje specifične ciljeve.
- Specifični** kontekst korištenja - softver je dizajniran da radi u okruženju u kojem će ga korisnici koristiti.



GLAVNE PREDNOSTI TESTIRANJA UPOTREBLJIVOSTI

- Povećanje zadovoljstva korisnika
- Povećanje broja korisnika
- Poboljšanje profitabilnosti - upotrebljiviji proizvodi stvaraju zadovoljne korisnike
- Poboljšanje dizajna interfejsa



Testiranje softvera može se posmatrati kao jedan od metoda koji se koriste za kontrolu kvaliteta softvera.

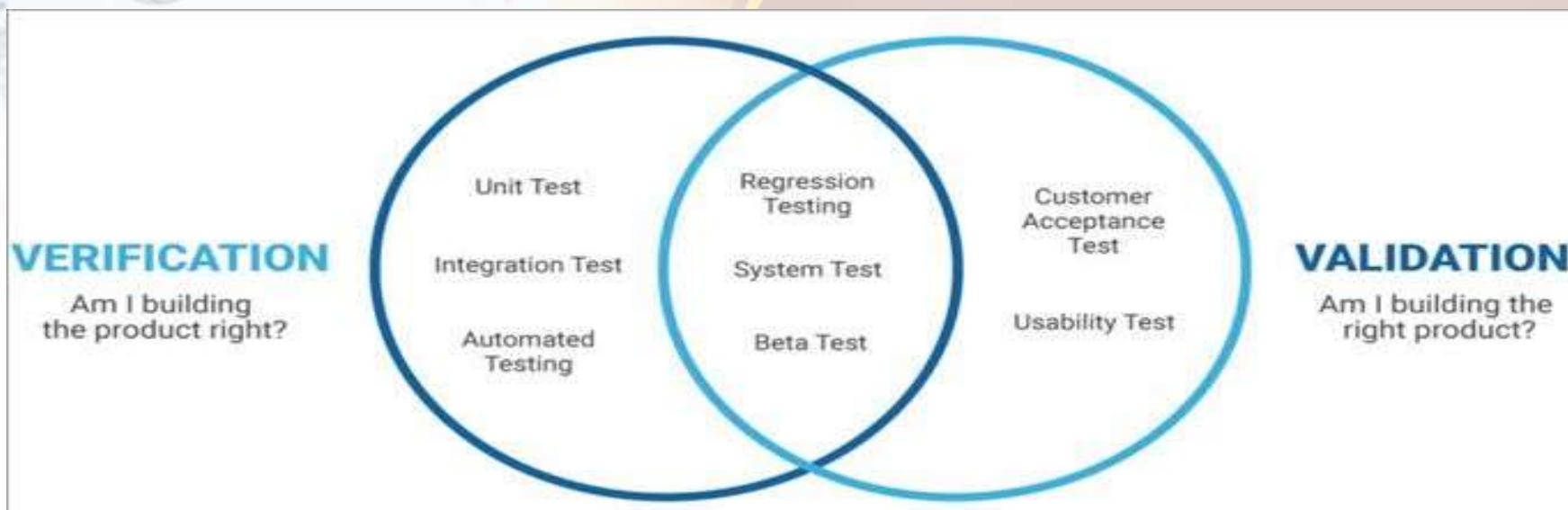
- **BITAN JE : KVALITET SOFTVERA**

Testiranje omogućava da se tokom razvoja i pisanja testova veoma rano identifikuju problemi u arhitekturi softverskog rešenja, funkcionalnoj specifikaciji,....

Mnogi autori u knjigama i naučnim časopisima tvrde sledeće:

"Svrha testiranja softvera može biti verifikacija ili validacija".

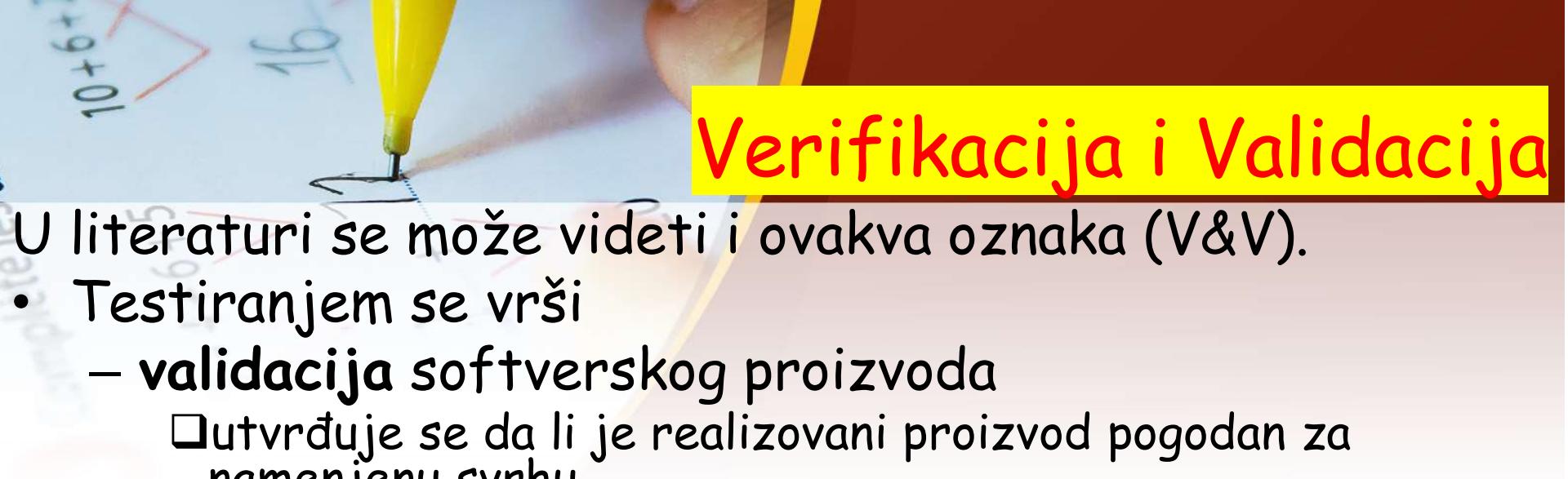
Verifikacija i Validacija





- Verifikacija softvera – Da li dobro razvijamo softver?
- Validacija softvera – Da li razvijamo dobar softver?
- Verifikacija daje odgovor na pitanje ("Are we building the product right?").

– Barry Boehm, 1981



Verifikacija i Validacija

U literaturi se može videti i ovakva oznaka (V&V).

- Testiranjem se vrši
 - **validacija** softverskog proizvoda
 - utvrđuje se da li je realizovani proizvod pogodan za namenjenu svrhu
 - vodi računa o korisnikovom pogledu na aplikaciju
 - **verifikacija** softverskog proizvoda
 - ✓ na nivou pojedinačne faze utvrđuje da li je rezultat realizacije određene faze u skladu sa specifikacijom za tu fazu
 - ✓ više je fokusirana na tehničku realizaciju

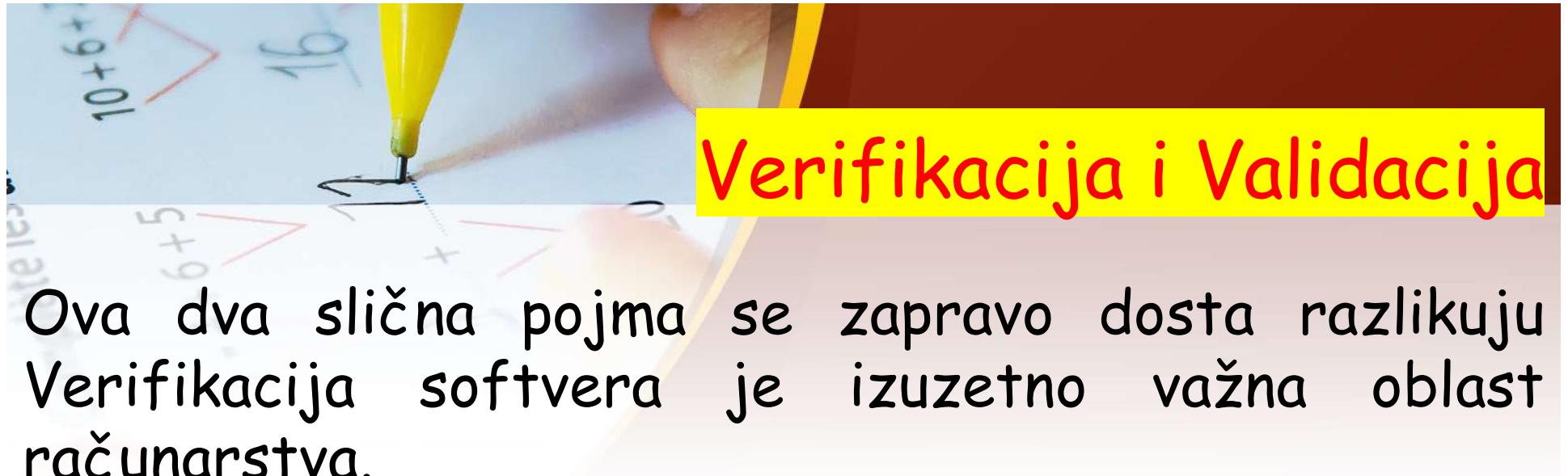


U radovima Džemjsa Witakera se može videte zašto je testiranje programa teško.

Validacija i verifikacija su izrazi koji se najčešće povezuju sa testiranjem programa.

Ove tehnike otelotvoruju principe deduktivnog zaključivanja, iste one koje koriste i programeri prilikom samog konstruisanja programa.

Zašto ne bi koristili iste principe u sistemu za automatsku sintezu, koji može da konstruiše program umesto da samo dokazuje njegovu ispravnost?



Ova dva slična pojma se zapravo dosta razlikuju. Verifikacija softvera je izuzetno važna oblast računarstva.

Neispravan softver može imati katastrofalne posledice, koje uključuju i izgubljene ljudske živote.

- Kako bismo izbegli ovakve situacije, neophodno je precizno utvrditi ispravnost razvijenog softvera



Verifikacija i Validacija

- **Validacija:** proces evaluacije sistema ili komponente za vreme ili na kraju procesa razvoja da bi se utvrdilo da li zadovoljava zahteve definisane od korisnika

Da li kreiramo pravi proizvod?

- **Verifikacija:** proces evaluacije sistema ili komponente kako bi se utvrdilo da li proizvod korektno implementira određenu funkciju
- **Da li kreiramo proizvod na pravi način?**



Šta se podrazumeva pod ispravnim softverom?

Razlikuju se pojmovi potpuno ispravnog i delimično ispravnog softvera.

Ne postoji algoritamski način da se proveri da li se neki program zaustavlja (halting problem), pa se često ni ne ispituje potpuna, već samo delimična ispravnost softvera



- Ukoliko je softver validan i verifikovan, to ne znači da on ne može imati grešku, već da je pouzdan i dobar za ono za šta je namenjen.
- Neretko softver testiramo za ograničenim skupom podataka
- Niko ne može garantovati da ne možemo pronaći test podatke na kojima se može desiti da naš program ne radi.
- **SUGESTIJA:**
Sprovedi testiranje sistema koristeći samo reprezentativni skup ulaznih podataka.



Verifikacija i Validacija

Verifikacija i validacija se provode kroz celi životni ciklus softvera, bez obzira na njegovu kompleksnost, veličini i sl.

Proces verifikacije i validacije se sastoji od tehnika kontrole softvera i testiranja softvera.

Kontrola i testiranje softvera su komplementarne tehnike.



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

Najčešći vid verifikacije softvera je
TESTIRANJE.

Testiranje se često koristi kao sinonim za verifikaciju softvera



Verifikacija i validacija se izvode u svakoj od faza životnog ciklusa razvoja.

Treba da se koriste u procesu testiranja softvera.

Verifikacija obuhvata sistematične procedure pregleda, analize i testiranja ugrađene u životni ciklus, počevši od faze softverskih zahteva, do faze kodiranja.

Verifikacija obezbeduje kvalitet i održavanje softvera.

Koncept verifikacije uključuje dva kriterijuma: softver mora adekvatno i korektno izvršavati sve funkcije i ne sme izvršavati one funkcije koje sam po sebi ili u kombinaciji sa ostalim funkcijama mogu loše da utiču na performanse čitavog sistema.



Verifikacija i Validacija

Verifikacija	Validacija
Da li je softver napravljen prema navedenim zahtevima i specifikacijama dizajna.	Konačni softver kako bi proverio da li on zadovoljava krajnje korisnike
Postoje specifične zahtevi u određenim fazama razvoja softvera	Implementiran je softver.
Statičko testiranje	Dinamičkog testiranje.
Da li softver pravimo ispravno	Da li pravimo pravi softver

Osnovni cilj Verifikacija i Validacija

je da je sistem u potpunosti ispunjava svoju namenu.

Ova dva termina se takođe nazivaju kontrolom kvaliteta softvera koju koriste testeri softvera u životnom ciklusu razvoja softvera. Iako i izgledaju i zvuče slično, razlikuju se u analizi.

Očekivanja korisnika - korisnici mogu imati očekivanja za određenu vrstu softvera.

I na kraju postoji i:

Plasiranje proizvoda na tržište ranije može biti važnije od pronalaženja defekata u softveru. (Procena raznih faktora)



Verifikacija i Validacija

- Ciljevi testiranja:

Direktni ciljevi

- Otkrivanje i smanjenje broja grešaka
- Povećanje kvaliteta testiranog softvera
- Izvršenje planiranih testova uz efikasno raspolaganje resursima - (vreme, ljudi)

- Indirektni ciljevi

- Statistika o vrstama grešaka u cilju preventivnih aktivnosti za otkrivanje grešaka



Posebno se mora voditi računa o Metodama za verifikaciju i validaciju.

Metode koje se koriste za verifikaciju i validaciju najčešće možemo podeliti u dve grupe:

- Statička verifikacija i validacija**
- Testiranje softvera**



Statička verifikacija i validacija se odnosi na analizu i proveru dokumenata, modela, dizajna i programskog koda, te se odvija bez stvarnog izvodenja softvera, dok se

Testiranje svodi na eksperimentalno izvođenje softvera ili delova softvera, sa test skupom podataka uz detaljno analiziranje rezultata.



Statička verifikacija i validacija

- ✓ Jedna statička provera može otkriti puno grešaka, dok testiranje može da otkrije samo jednu grešku i onda se može se desiti da nakon samo te jedne greške dolazi do pada softvera (aplikacije).
- ✓ Kod statičke metode dolazi do izražaja znanje o programiranju budući da se osobe koje obavljaju proveru oslanjaju se na iskustvo i greške koje se najčešće se pojavljuju i na njih se oni fokusiraju.

Statička verifikacija i validacija

Neke od bitnijih tehnika statičke verifikacije su:

Analiza toka podataka(eng.data flow analysis)

Apstraktna interpretacija (abstract interpretation)

Simbolička analiza(Symbolic analysis)

Proveravanje ograničenih modela primer (Bounded model checking), koja se najviše koristi za verifikaciju logičkih kola.



Automatska statička analiza

Automatizacija procesa generisanja test primera i provera rezultata testiranja posebno je važna jer olakšava i ubrzava proces testiranja.

- ✓ To je metoda koja koristi softverske alate koji prolaze kroz tekst koda i otkrivaju moguće greške i anomalije
- ✓ U okviru statičke automatizovane provere ispravnosti softvera formiraju se uslovi ispravnosti iskazani formulama u terminima matematičkih teorija



Idejom formalne verifikacije bavila su se mnoga velika imena naučnika 70-tih godina 20. veka (Hoare, Dijkstra, Wirth).

Formalna verifikacija - danas je :

- ✓ Predmet intenzivnog istraživanja.
- ✓ Polako postiže sve bolje rezultate.

Postoji varijanta formalne verifikacije koja se zove provera modela (model checking, koja se dosta koristi).



Formalna verifikacija svodi se na matematičko dokazivanje konzistentnosti programa sa svojom specifikacijom.

Ovu metodu moguće je koristiti samo kada je :

- semantika programskog jezika je formalno definisana
- postoji specifikacija za dati program/projekat

Oni koji je koriste tvrde da ona vodi do pouzdanijeg i sigurnijeg softvera.

Dokaz korektnosti programa je velik i složen postupak , pa u njemu mogu da postoje i greške.

Dinamička verifikacija softvera

zasnovana je na tome da se provera ispravnosti softvera vrši tokom njegovog izvršavanja.

- Najčešće se dinamička verifikacija softvera vrši testiranjem i ti pojmovi se poistovećuju, što nije sasvim ispravno.

Testiranje je složen proces koji obuhvata pronalaženje što raznovrsnijeg skupa ulaza, definisanje očekivanih izlaza za svaki od tih ulaza, a zatim izvršavanje programa i provera da li je program za date ulaze vratio odgovarajuće izlaze

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

Metodologije razvoja softvera

Ciljevi testiranja softvera su :

- ❖ Verifikacija i validacija, kojima se proverava da li je softver saglasan sa specifikacijom zahteva
- ❖ Što ne bi značilo uvek da je softver tehnički ispravan, pouzdan i bezbedan.
- ❖ Poboljšanje kvaliteta softvera.
- ❖ Procena pouzdanosti.

Aktivnosti testiranja SOFTVERA

- Planiranje
- ✓ Vremenski raspored procesa testiranja i testnih aktivnosti
- ✓ Objekat testiranja i zahtevi
- ✓ Strategija testiranja i prateći alati
- ✓ Dizajn i specifikacija okruženja
- ✓ Testiranje okruženja
- ✓ primeniti metode za dizajn testiranja na bazi dizajna softvera i zahteva
- ✓ Specifikacija testnih slučajeva sa ciljem postizanja pokrivenosti testiranja.
- ✓ Instalacija testnog okruženja
- ✓ Instalacija alata i testnog okruženja



Faza analize zahteva

Cena greške = vreme potrebno da se utvrde i zapisi 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.



Faze testiranja sofvera

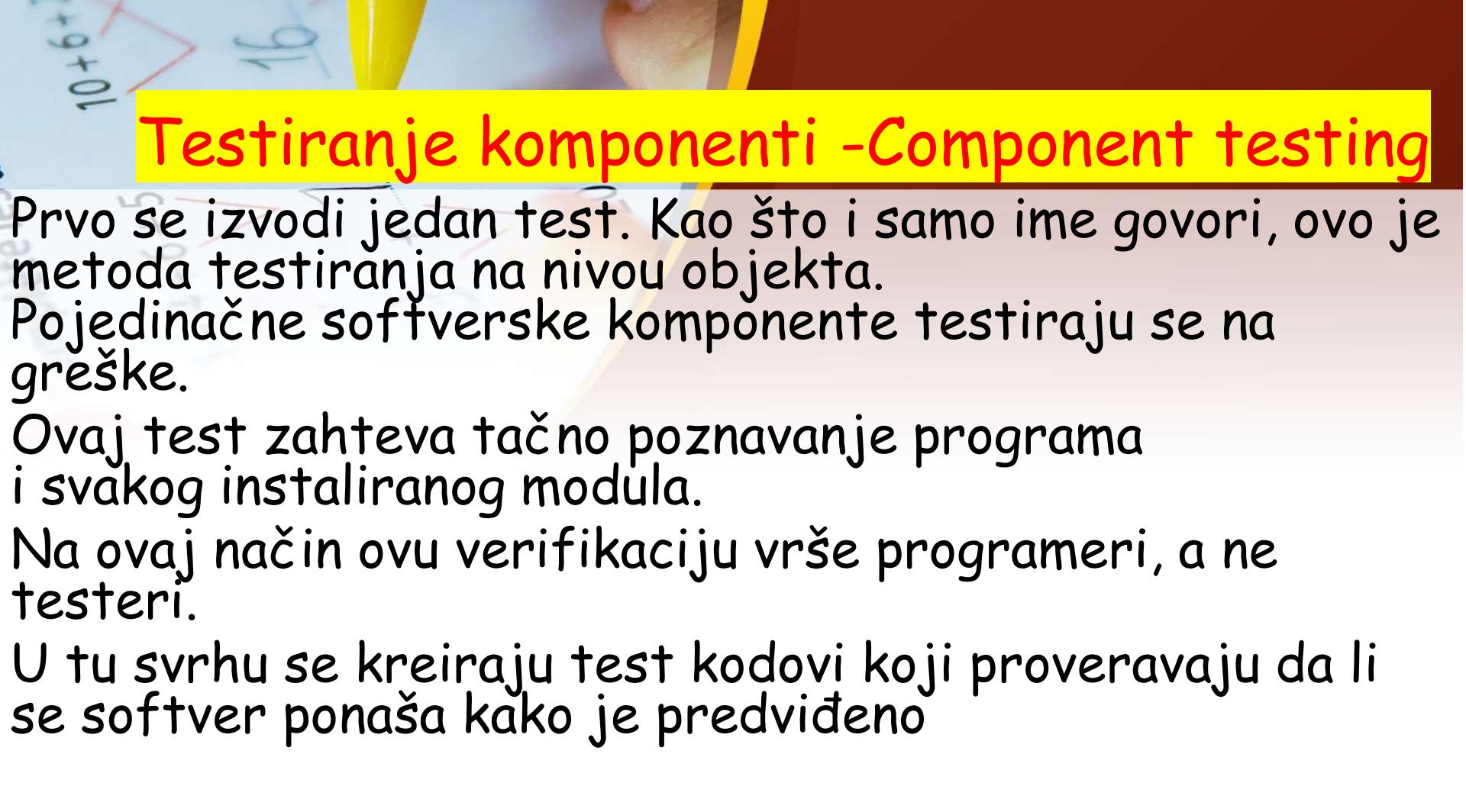
- Razvojno testiranje - sistem se testira u toku razvoja da bi se otkrili bagovi ili drugi defekti.
- Testiranje proizvoda (release testing) - poseban tim testira kompletnu verziju aplikacije pre nego što se ona isporuči korisnicima.
- Korisničko testiranje (user testing) - korisnici ili potencijalni korisnici sistema testiraju sistem u zadatom (sopstvenom) okruženju.



Razvojno testiranje

Obuhvata sve aktivnosti testiranja koje izvršava tim za razvoj softvera.

- ✓ Testiranje jedinica (Unit testing)
- ✓ Testiranje komponenti (Component testing) - nekoliko posebnih jedinica se integrišu kako bi kreirale složene komponente jedinica
Testiranje komponenti se fokusira na testiranje interfejsa komponenti.
- ✓ Testiranje sistema



Testiranje komponenti -Component testing

Prvo se izvodi jedan test. Kao što i samo ime govori, ovo je metoda testiranja na nivou objekta.

Pojedinačne softverske komponente testiraju se na greške.

Ovaj test zahteva tačno poznavanje programa i svakog instaliranog modula.

Na ovaj način ovu verifikaciju vrše programeri, a ne testeri.

U tu svrhu se kreiraju test kodovi koji proveravaju da li se softver ponaša kako je predviđeno

OSNOVNE faze testiranja softvera

Planiranje definiše:

- Vrste testova koje će biti sprovedene
- Opseg testiranja
- Izbor strategije i metode testiranja
- Koji je kriterijum završetka testiranja
- Koji su potrebni resursi i
- Kakav je način komunikacije izmedu članova tima



Testiranje po delovima, gde se identifikuju grupe ulaza koje imaju zajedničke karakteristike i koje treba uraditi na isti način.

- Treba izabrati testove iz svake grupe
- Testiranje po zadatim procedurama i uputstvima
- Izbor testova.
- Uputstva predstavljaju predhodna iskustva o vrstama grešaka koje programeri prave prilikom razvoja komponenti.



Evaluacija testova

Testiranje se obično završava kada je softver isporučen korisniku, mada se može se desiti i u nekim drugim situacijama, na primer, kada je projekat otkazan ili je neki cilj postignut.

TESTIRANJE SOFTVERA SE NIKAD NE ZAVRŠAVA

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.

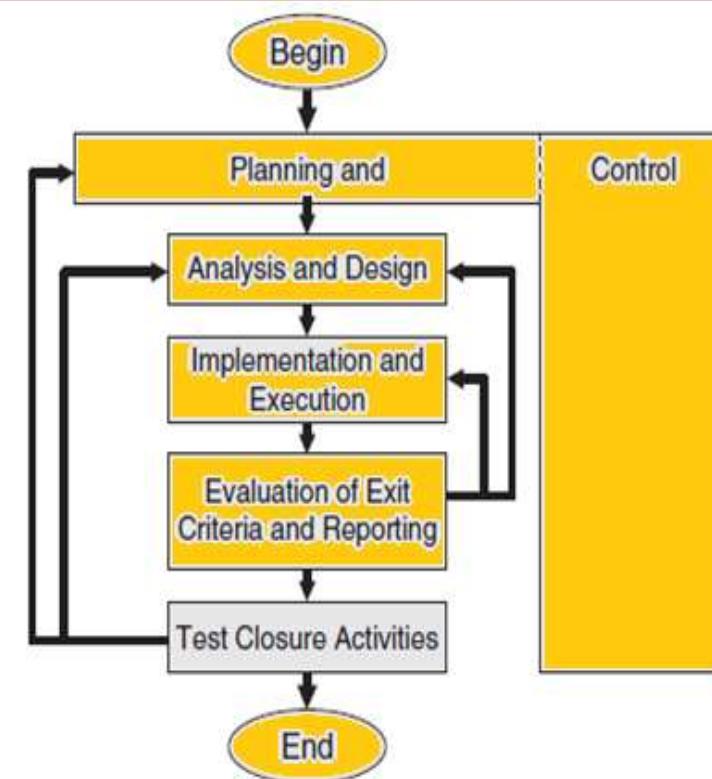
Za svaki nivo testiranja prolazi se kroz kompletan ciklus testiranja softvera.

Nakon svakog ciklusa potrebna je izmena u kodu.
(ako postoji greške)

Nakon izmena potrebno je ponoviti testove.

Upravljanje testiranjem vodi se računa o inicijalizaciji, nadzoru i upravljanju ovim ciklusom

Ciklus testiranja





Kategorije rizika

- ✓ Loša procena vremena
 - Problem sa specifikacijom
 - Nemogućnost pravilne procene funkcionalnosti i vremena potrebnog za razvoj funkcionalnosti.
 - Korišćenje resursa se ne nadgleda pravilno.
 - Neočekivana proširenja obima projekta sa dodatnim zahtevima i izmenom specifikacije uz nemogućnost dobijanja dodatnog vremena

Operativni rizici:

- Rizici uzrokovani nedostatkom pravilne implementacije, sistemskim greškama ili nekim spoljnjim događajima rizika



- Postavljanje novih verzija softvera je bio veliki, složen i rizičan zadatak. Nakon što bi nova verzija softvera bila testirana, razvojni tim bi dobio zadatak da je rasporedi u proizvodnji.
- U zavisnosti od veličine softvera, taj proces je mogao da traje satima, danim ili nedeljama, i zahtevao je detaljan prolazak kroz kontrolne liste, što je činilo mnogo manuelnih koraka, kao i posebnu stručnost.



- Postavljanje novih verzija softvera je
 - ✓ veliki,
 - ✓ složen i
 - ✓ rizičan zadatak.
- Nakon što bi nova verzija softvera bila testirana, razvojni tim bi dobio zadatak da je pusti u prdukciju.
- U zavisnosti od veličine softvera, taj proces je mogao da traje satima, danima ili nedeljama, i zahtevao je detaljan prolazak kroz kontrolne liste, što je činilo mnogo manuelnih koraka, kao i posebnu stručnost.



Tehnički rizici

Tehnički rizici obično dovode do gresaka u funkcionalnosti i performansama sistema.

Uzroci tehničkih rizika su:

- Stalna promena specifikacije
- Problem u vremenu
 - Korišćenje zastarelih tehnologija ili tehnologija u ranoj fazi.
 - Složenost implementacije projekta
 - Kompleksna modularna integracija projekata





Dinamičko testiranja softvera

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

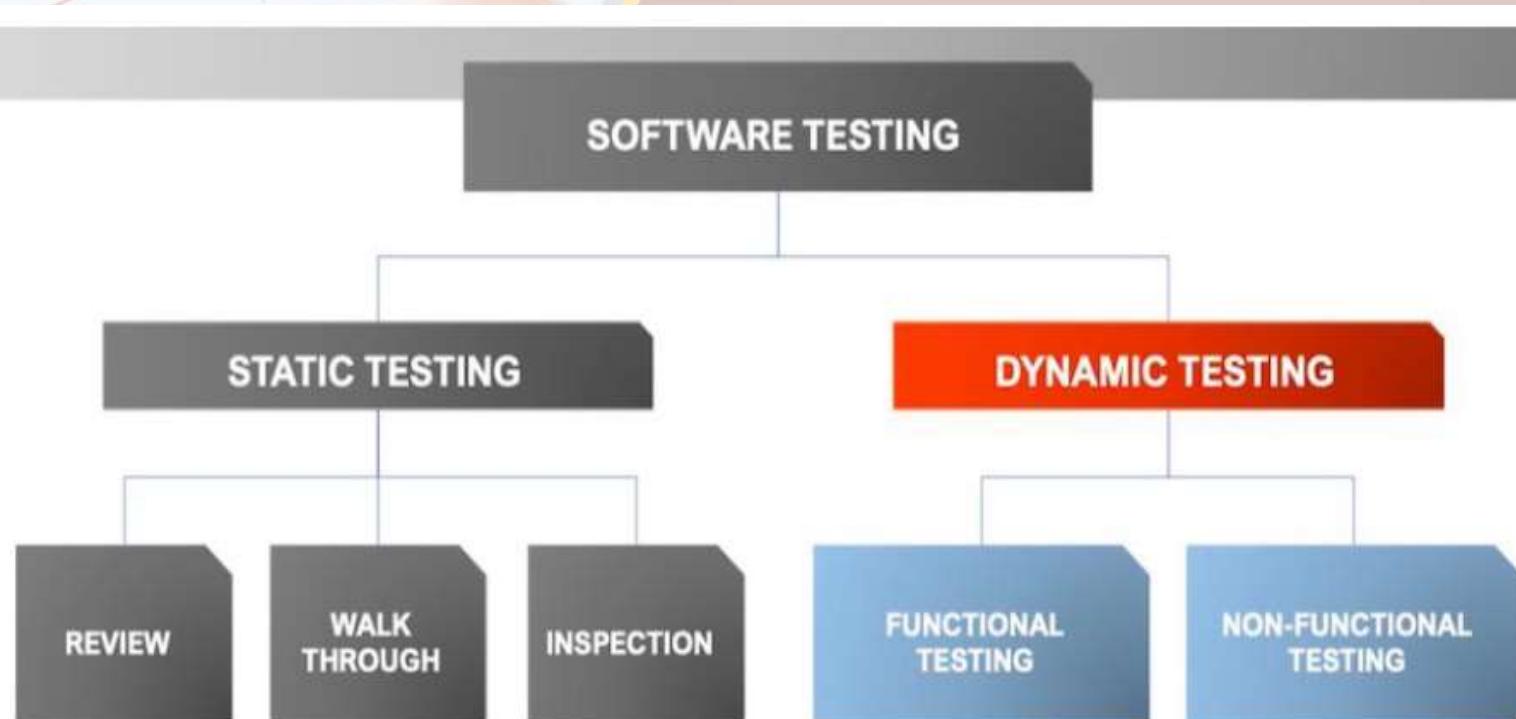
Dinamičkim testiranjem se vrši provera ponašanja sistema, tačnije upoređivanje dobijenog i očekivanog izlaza.

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**.

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

Dinamičko Testiranja softvera





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 vira 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 softvera 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 obraduje neke ulazne podatke i na kraju se dobijaju izlazne vrednosti.

Ulagni 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.



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 strukturu 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 kvalitenim 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





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 inicializacije
- 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 netrivijalne 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 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.

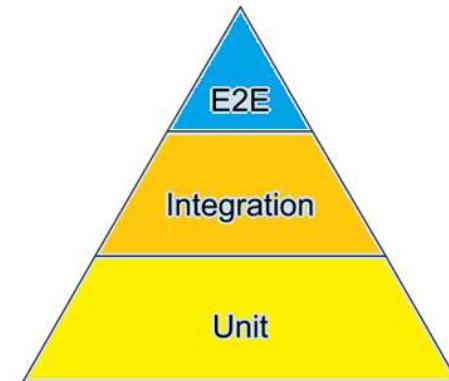
Ova vrsta testiranja zasniva se u potpunosti na softverskim zahtevima i specifikaciji.

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)



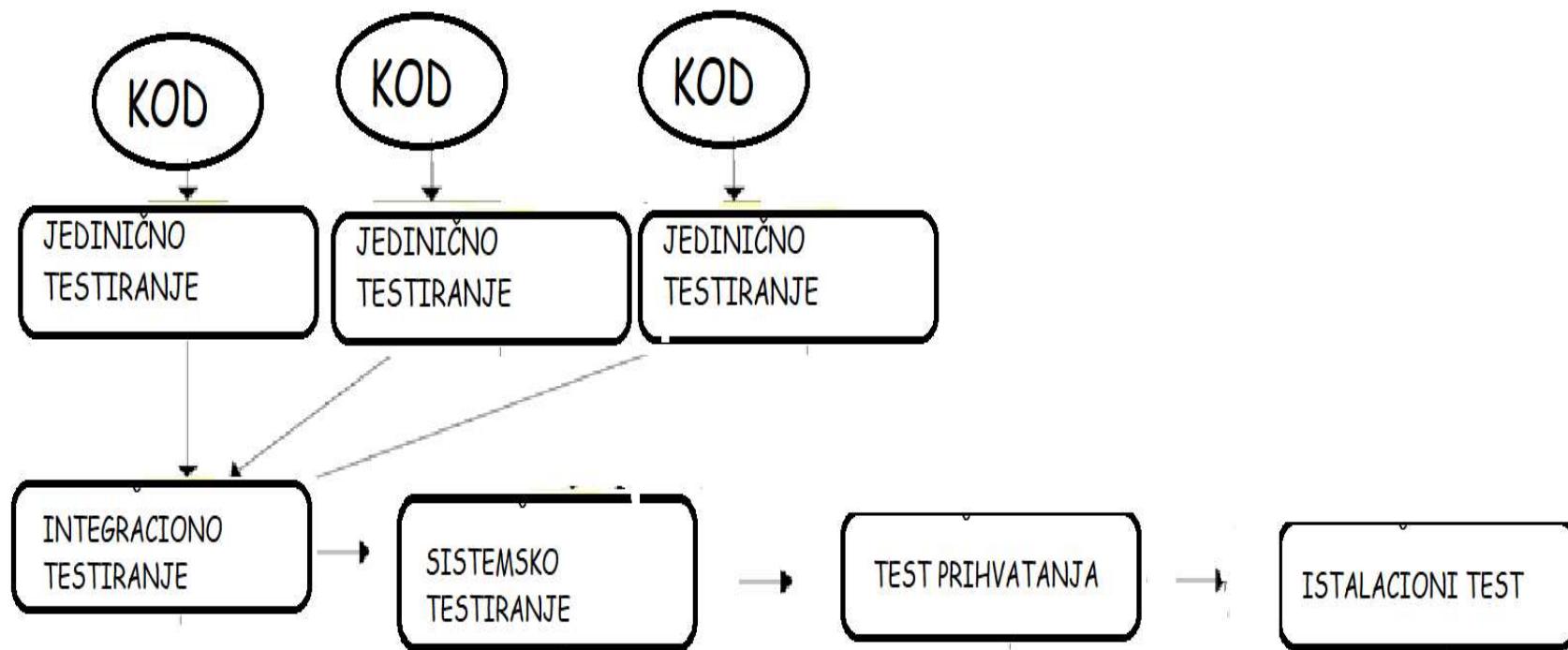


Dokazano je da se koristi za kreiranje jediničnih testova

- 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, softvrsko 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

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



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.

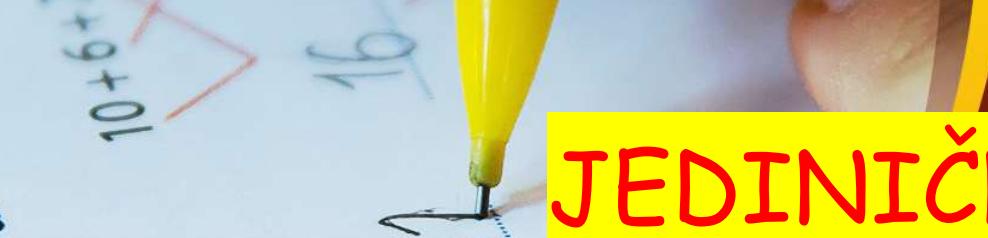
Previše vremena treba za pisanje testova

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

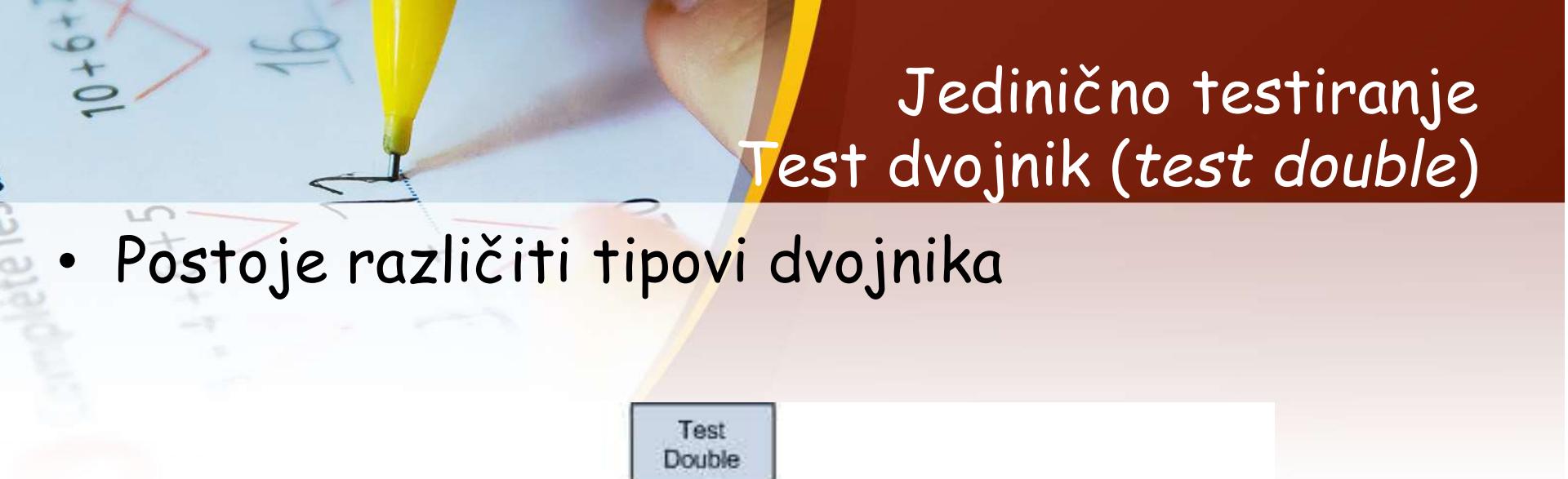


JEDINIČNO TESTIRANJE

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.

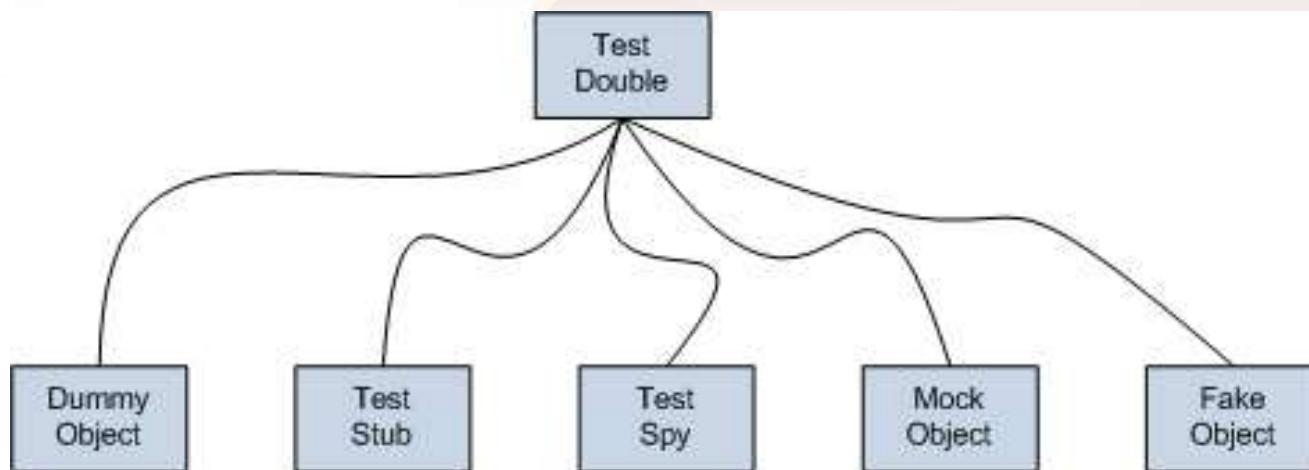
- Prednost ovakvog načina testiranja je što se greške mogu lokalizovati i ispraviti u elementarnim komponentama kako bi se obezbedilo da ispravna komponenta uđe u sistem.
- Kada se vrši jedinično testiranje, sistem se sastoji iz povezanih komponenti i testira se svaka komponenta posebno.
- Cilj jediničnog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



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 debagovanja 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 tehniku 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.



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.



Ispituje se da li su veze izmedu 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 izmedu komponenti

Integracionim testovima proverava se da različite komponente sistema rade ispravno zajedno.

- Testiranje metodama crne kutije



- 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 debagovanja i retestiranja bi bili potpuno nepraktični



Drugo, zadovoljavanje kriterijuma testiranja „bela kutija“ bilo veoma teško, zbog ogromne količine detalja razjedinjavanjem ulaznih datoteka. U stvari, većina integracionih testiranja je tradicionalno limitirano na tehnike funkcionalnog testiranja (Hetzl.) Veliki sistemi mogu zahtevati mnoge integracione faze, počevši od sakupljanja modula u nisko-rangiranim podsistemima, pa onda okupljanje podistema u veće podsisteme i konačno sklapanje podistema 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. 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.



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 struktturnog 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 struktturnog 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 sistemske 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



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



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.

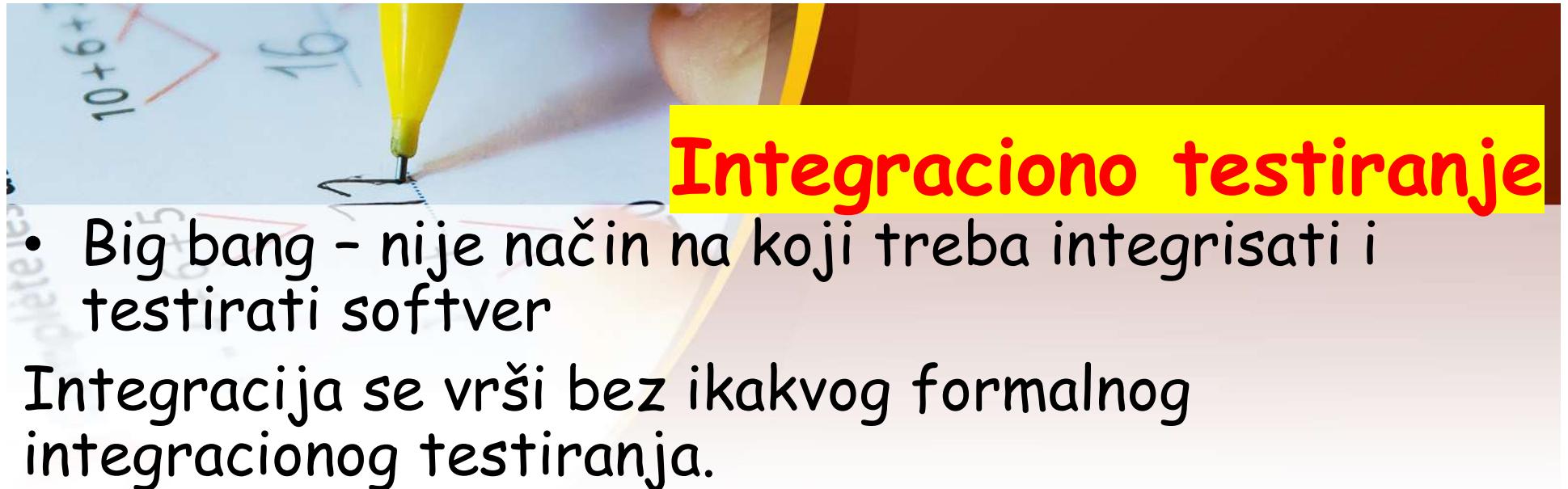
Kod ove strategije, korišćenjem testiranja drajverima, počinje izvođenje strukturnog testiranja modul-nivo na najnižim nivo modulima koristeći test drajvere.

Zatim, izvođenje modul-nivo struktorno 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.

-



- 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



- Big bang - nije način na koji treba integrisati 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



- 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



- 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



- 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 postoji kod testiranja graničnih vrednosti.

Kada se radi metoda klase 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 provera 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 identificuje da pripada posebnoj klasi ekvivalencije

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



Metoda je dopuna metode klase ekvivalencije.

Bavi se vrednostima koje su na granicama klase 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



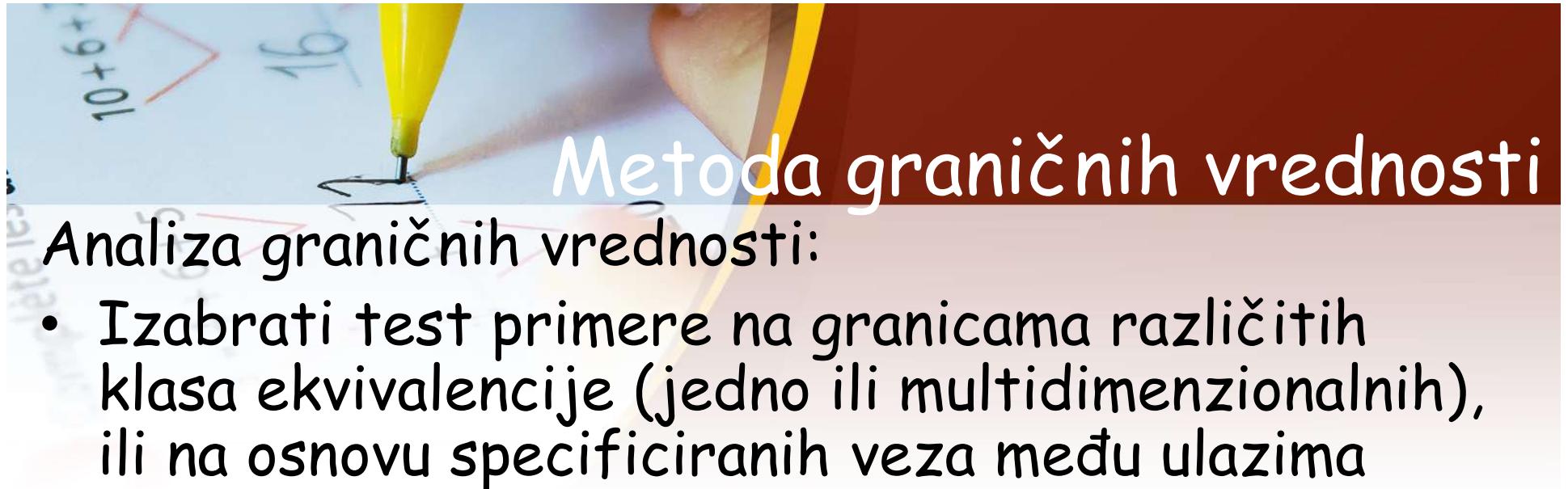
Metoda graničnih vrednosti

Pripada testiranju crne kutije.

Skoro uvek se koristi zajedno sa klasama ekvivalencije.

Kada se radi o metodi testiranje 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:

- Izabratи test primere na granicama različitih klasа еkvivalencije (jedno ili multidimenzionalnih), ili na osnovу specificirаних veza među ulazima

Po jedan test primer za svaku graničnu vrednost

- Testove pravimo tako što za svaku prethodno odredenu 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



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 zavšetak ovog vida testiranja softvera:

KRITERIJUM

Postignuti nivo pokrivenosti svih graničnih vrednosti

Izračunava se kao odnos testiranih i ukupnih graničnih vrednosti

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

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.

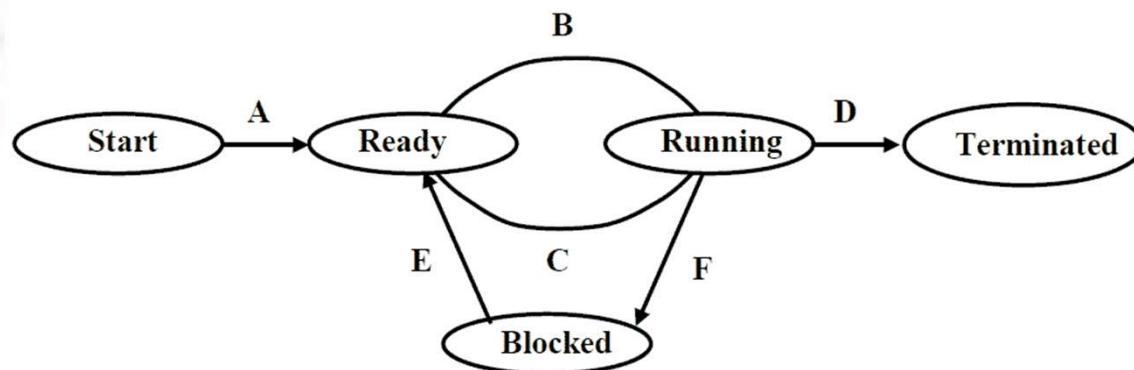
Testiranje bazirano na tabeli odluke

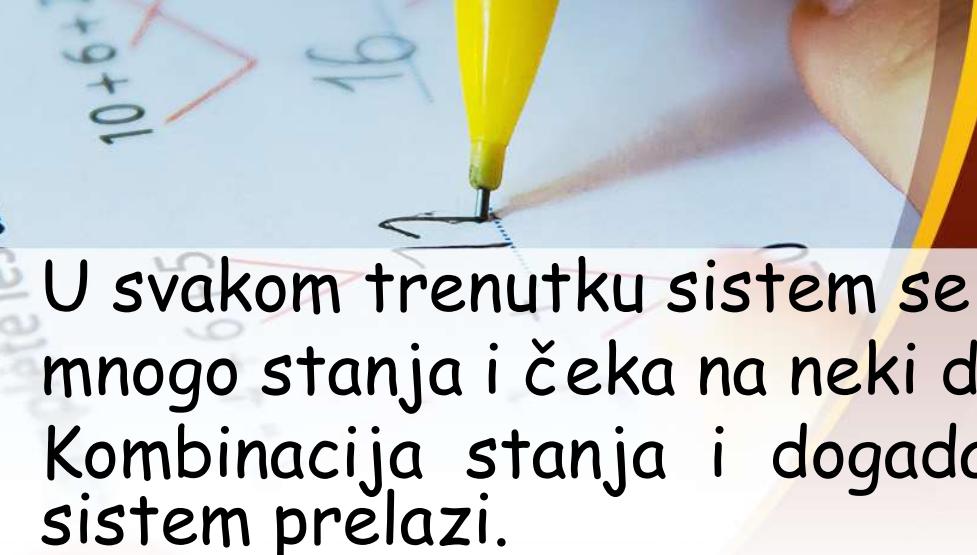
C1: a,b,c form a triangle	F	T	T	T	T	T	T	T	T
C2: a=b ?	-	T	T	T	T	F	F	F	F
C3: a=c ?	-	T	T	F	F	T	T	F	F
C4: b=c ?	-	T	F	T	F	T	F	T	F
A1: nije trougao	X								
A2: raznokraki									X
A3: jednakokraki			X		X	X			
A4: jednakostručni	X								
A5: nemoguć		X	X		X				

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 vrednos T, što znači da a, b i c formiraju troug da je a=b, b=c i a=c, slede da je trouga jednakostručni i tako dalje.

Testiranje prelaza stanja





Dijagrami stanja

U svakom trenutku sistem se nalazi u nekom od konačno mnogo stanja i čeka na neki dogadaj.

Kombinacija stanja i dogadaja odreduje 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



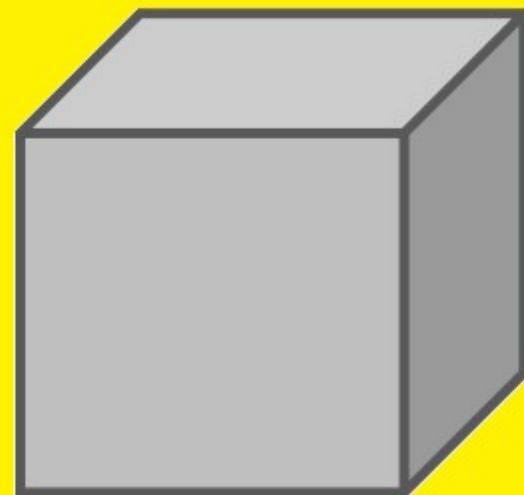
Logički zasnovane tehnike

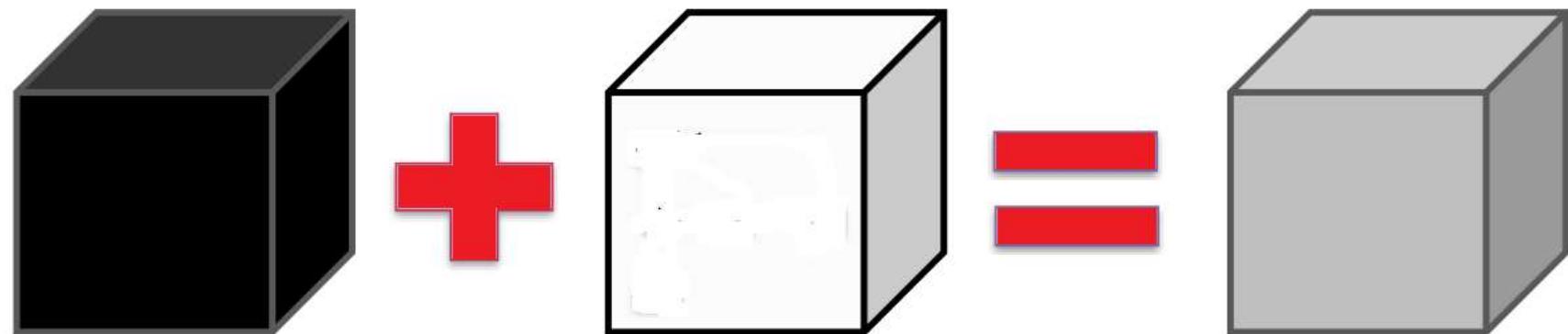
- ✓ Posmatraju i međusobne zavisnosti između ulaznih parametara.
- ✓ Za razliku od drugih tehnika koje pri kreiranju test slučajeva svaki parametar nezavisno posmatraju

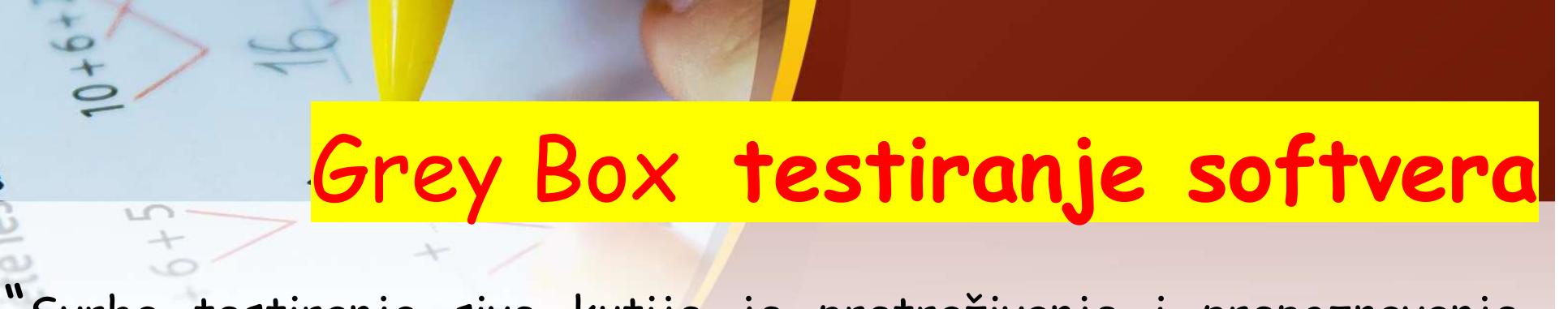
Postoje različite logički zasnovane tehnike:

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

Grey Box testiranje softvera







Grey Box testiranje softvera

"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 testeri 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 ponavanjem internog rada aplikacije.

Tester je delimično upoznat sa internom strukturom.

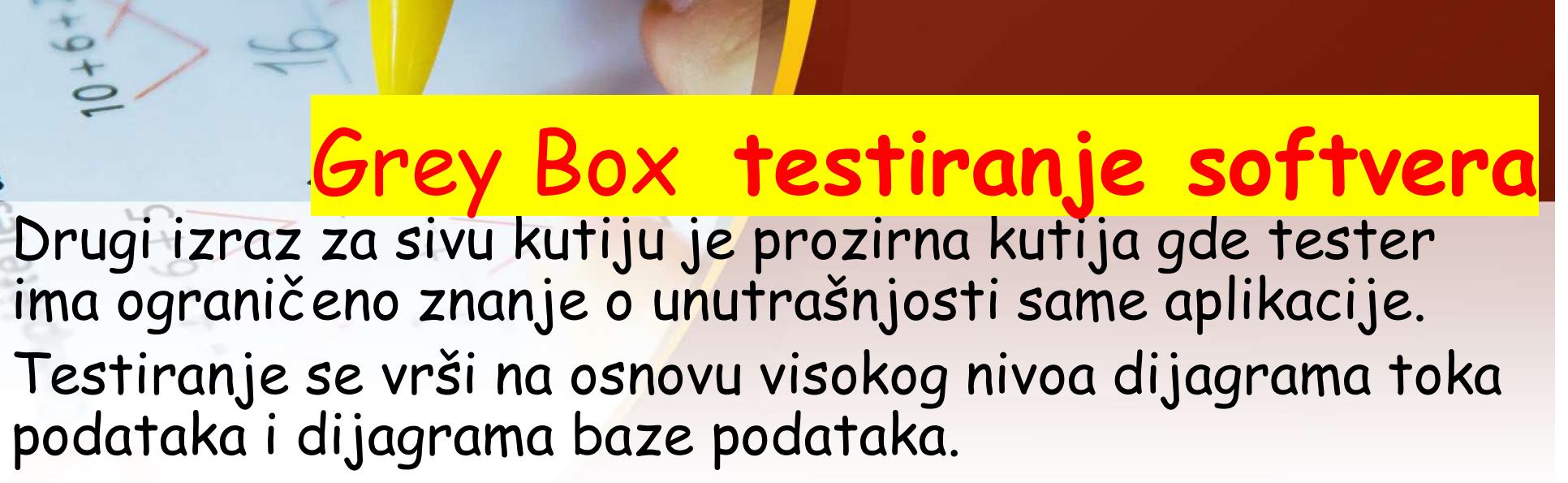
Grey Box testiranje softvera

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.
- ✓ Testeri 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.



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.





Bitan korak u razvoju softvera je - Testiranje softvera.

Proizvođači softvera bi želeli da predvide broj grešaka u softverskim sistemima pre nego što ih primene, kako bi mogli da procene kvalitet kupljenog proizvoda i teškoće koje se javljaju u samom procesu održavanja.

Cilj testiranja softvera je da se ustanovi da li se softver ponaša na način koji je predviđen zadatom specifikacijom.

Prema tome, PRIMARNI CILJ Testiranja softvera je otkrivanje grešaka u softveru.



TESTIRANJE SOFTVERA

Jedno od velikih pitanja koje se javlja kod testiranja softvera je **reprodukција greške** (testeri otkrivaju greške a programeri otklanjaju bagove).

Očigledno je da mora postojati koordinacija između testera i programera.

Reprodukciјa greške je slučaj kada bi bilo najbolje da se problematični test izvede ponovo i da znamo kad, i na kom mestu u programu i pod kojim uslovima se tačno greška desila.

Dakle, idealnog testa nema, kao što nema ni idealnog proizvoda.



- ✓ Dizajniranje i pregled test strategija, test planova i test slučaja
- ✓ Upravljanje konfiguracijom i promenama
- ✓ Prilagođavanje procesa testiranja potrebama projekta
- ✓ Metrike
- ✓ Kriterijumi za završetak testiranja



TESTIRANJE SOFTVERA

Vrste nefunkcionalnih testova:

- ✓ Installation testing - testiranje instalacije
- ✓ Performance testing - testiranje karakteristika i performansi
- Stress testing testiranje opterećenja
- Endurance testing - testiranje izdržljivosti
- Load testing - testiranje opterećenja
- Spike testing (kako sistem funkcioniše u takozvanim ekstremnim situacijama)
- ✓ Documentation testing - testiranje dokumentacije
- ✓ Reliability testing - testiranje pouzdanosti
- ✓ Security testing - bezbednosni testovi



Spada u tehnike testiranja zasnovane na defektima.

Osnovni koncept testiranja na bazi defekata je da se izaberu slučajevi testiranja koji prave razliku između programa koji testiramo i alternativnih programa koji sadrže neke greške.

Ovo se obično postiže izmenom programa koji se testira da se proizvedu "pogrešni programi".

Ovakvo testiranje se može koristiti za:

- ✓ procenu temeljitosti (adekvatnosti) serije testova,
- ✓ za izbor novih test slučajeva da se dodaju seriji testova,
- ✓ da se proceni broj defekata u programu



Sanity testiranje

U nekim literaturama se nalazi pod nazivom

- test zdravog razuma (Sanity test). On je veoma sličan smoke testu.

Ovi termini se često mešaju u testiranju softvera.

Iako razlike između ova dva tipa nisu velike, **one ipak postoje**.

- Smoke test se koristi za testiranje builda softvera da bi se verifikovao ispravan rad osnovnih funkcionalnosti i izvršava se pre bilo kojeg detaljnog testiranja, kako testeri ne bi gubili vreme u slučaju nekih kritičnih problema.



Sanity testiranje

Sanity test se izvršava na relativno stabilnim softverom. Ovaj tip testiranja vrše testeri nakon primanja builda u kome postoje izmene u kodu ili je dodana nova funkcionalnost.

Cilj sanity testiranja je da se utvrdi da li dodana funkcionalnost radi kako treba i da osnovne funkcionalnosti softvera nisu ugrožene.

Ako sanity test ne prođe, build se odbija i ne počinje se dalje testiranje, čime se štedi vreme i smanjuju troškovi.



Sanity testiranje

Smoke i Sanity testiranje služe da bi se utvrdilo da li je softver dovoljno stabilan za detaljno testiranje, odnosno da se uštedi vreme u slučaju da postoji neka kritična greška.

Oba tipa testiranja se obavljaju nad istim buildom, prvo se izvršava smoke a nakon njega sanity.

Ovi tipovi testiranja se mogu izvršiti i manuelno ili nekim alatom za automatizaciju.

U slučaju da se koristi neki alat za automatizaciju, najčešće se pokretanje tih testova inicijalizira sa istim procesom koji pravi sam build.



Smoke testiranje

Smoke testiranje obuhvata niz jednostavnih testova koji za cilj imaju potvrdu rada najvažnijih funkcija sistema.

Ovaj tip testiranja se radi kako bi se ispitalo da li je sistem dovoljno stabilan da podrži dalje testiranje tako da se ne ulazi u "dublji" test ako ključne funkcije ne rade.

Ovaj oblik testiranja softvera nije detaljan, niti ulazi u dubinu funkcionalnosti.



Smoke testiranje

Prema (Rasmussen, 2016.,) se navode dobre osobine testiranje :

- ✓ Proverava je li aplikacija ispravno isporučena
- ✓ Proverava je li okruženje ispravno postavljenog,
- ✓ Da li su svi delovi arhitekture ispravno povezani i postavljeni pravilno.



Smoke testiranje

U nekim radovima se ovo testiranje navodi kao Build Verification Testing.

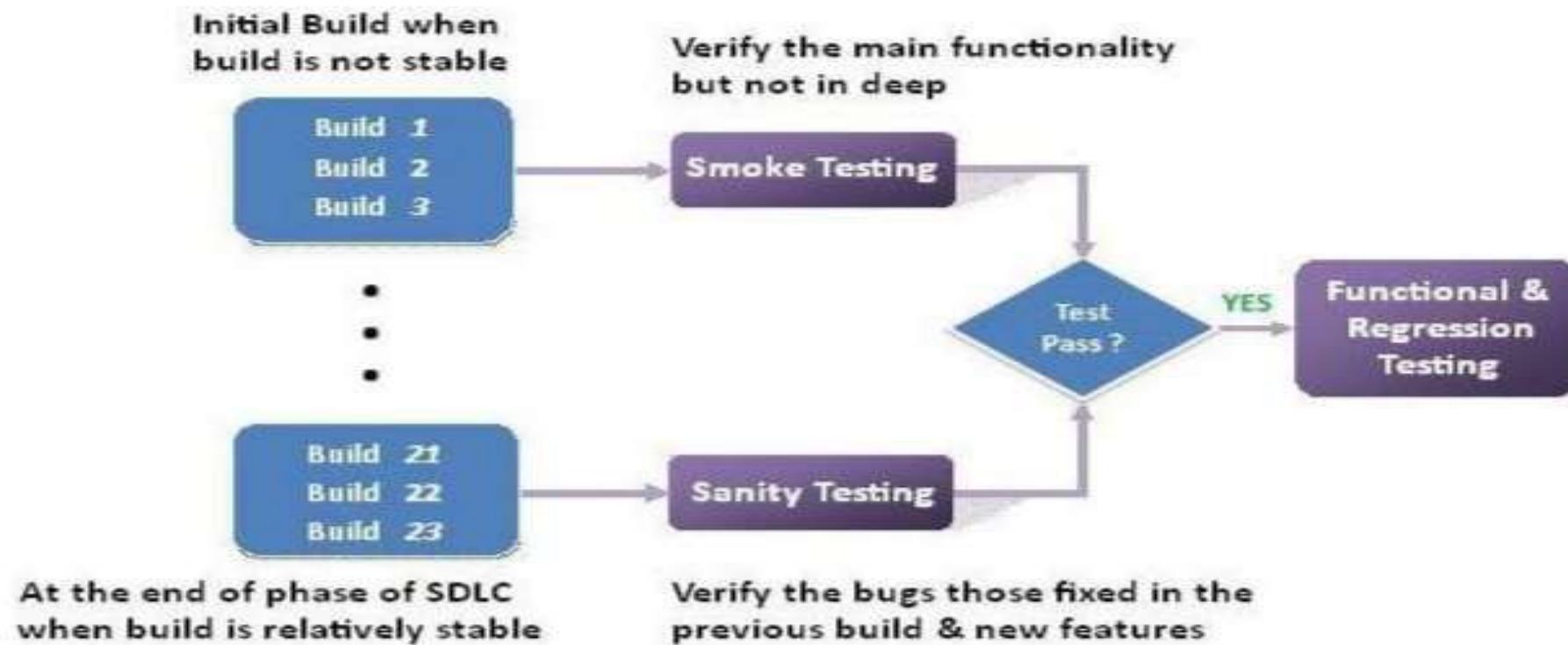
Smoke testiranje predstavlja test ključnih funkcija. Ukoliko su ključne funkcije ispravne, moguće je dalje testirati softver.

Prilikom svake izmene koda, stanje sistema se menja.

Zbog toga je, pre bilo kakve detaljnije provere softvera, potrebno testirati osnovne funkcionalnosti (Smoke Testing).

Ukoliko nema kritičnih softverskih grešaka, može se nastaviti sa testiranjem, a ukoliko neka od osnovnih funkcionalnosti nije ispunjena, ona obično blokira testiranje i mora se popraviti u najkraćem roku.

Sanity i Smoke testiranje





TESTIRANJE SOFTVERA

Prihvaćeni način za dokazivanje stvari u programima je korišćenje pravila zaključivanja kao što ih je izneo Hoare.

Ova pravila se obično formulišu tako da se mogu primeniti na poslednju programsku liniju, čime se proverava jedan ili više kratkih programa i možda neke logičke formule.

Iterativnom primenom pravila zaključivanja, zadatak dokazivanja ispravnosti programa svodi se na dokazivanje programskih linija u računu verovatnoće



TESTIRANJE SOFTVERA

PODSEĆANJE:

Tipovi defekata pronađeni statičkim testiranjem:

- Nejasna i nepotpuna specifikacija zahteva
- Greške u arhitekturi
- Greške u dizajnu
- Problemi u specifikaciji interfejsa između komponenti
- Neusklađenost sa zahtevanim i postavljenim standardima
- Prekomplikovan kod



Pozitivni i negativni test TESTIRANJA:

Pozitivni test koji predstavlja osnovni scenario korišćenja softvera. Kada korisnik ispoštuje sve procedure unosa (unosi podatke na predviđeni i korektni način). Tada je rezultat u skladu sa očekivanim.

Dobro namerni korisnik

Dobra praksa je testiranje što većeg broja slučajeva, ako ne i svih poslovnih pravila, kako bi se na vreme identifikovali propusti. Uneti više istih set podataka nekoliko puta.

Negativni test predstavlja scenarijo u kojima je osnovna pretpostavka da neće sve da prođe kao što je očekivano.

Ne dobro namerni korisnik (strah od novog, strah od nepoznatog)



TESTIRANJE SOFTVERA

Na osnovu toga da li je i u kojoj meri program adekvatan za obavljanje datog posla, testiranje se može definisati kao aktivnost:

- ✓ Provere usklađenosti realizacije programa sa njegovom specifikacijom (neraskidivo povezano sa osobinom tzv. korektnosti programa, koja predstavlja meru u kojoj program zadovoljava specifikaciju),
- ✓ Provere njegovog ponašanja u okruženju (usko povezano sa pouzданošću programa, koji predstavlja otpornost na ulazne podatke koji su neregularni).
-



Testiranje slabih struktura je, kao što se pojavljuje u literaturi je slaba varijanta strukturnog testiranja.

Slabo strukturno testiranje je lakši oblik izvođenja manualnog strukturnog testiranja.

Prema tome, cilj slabo strukturnog testiranja je da se izvrši testiranje programa po značajno manjoj ceni pre automatske podrške kada je strukturno testiranje dostupno.



Pod pojmom uspešnog testa mogu se podrazumevati dva, i to suprotna koncepta, što je i od najveće važnosti za opšti pristup testiranju.

Posmatrano sa stanovišta izrade korektnog programa, test se smatra uspešnim ukoliko **nema grešaka**.

No, međutim, ukoliko se uzme u obzir utrošeno vreme kao i uloženi budžet na testiranje sasvim je opravdano diskutovati o uspešnom testu u smislu dokaza da grešaka ima.

Obzirom na to da i jedan i drugi prilaz imaju smisla, u prvom slučaju govorimo o konstruktivnom pristupu testiranju, a u drugom o destruktivnom pristupu testiranju i isto tako o konstruktivno uspešnom i destruktivno uspešnom testu.



TESTIRANJE SOFTVERA

Destruktivno testiranje može se definisati kao aktivnost analize izvršavanja programa sa ciljem otkrivanja grešaka.

Primarna osobina ovog načina testiranja je mogućnost dokaza prisustva grešaka ne podrazumevajući pri tome dokaze da je program korektan.

Glavni cilj destruktivnog testiranja je otkriti što više grešaka uz najmanje troškove, tj. u najkraćem mogućem vremenskom okviru.



Destruktivni pristup testiranju podrazumeva veliki broj metoda koje se mogu podvesti pod tri strategije:

- ✓ Analiza programa (bez primene računara),
- ✓ Strategija bele kutije (White Box Strategy) ili strukturno testiranje
- ✓ Strategija crne kutije (Black Box Strategy) ili funkcionalno testiranje



TESTIRANJE SOFTVERA

Neki od problema u razvoju softvera na koje treba da se obrati pažnja:

- ✓ Precizno planiranje(resursa, budžet samog projekta, trajanja, obuke potencijalnog kadra koji bi radio na datom softveru i drugo)
- ✓ Identifikaciju, procenu i kontrolu rizika na softverskom projektu
- ✓ Utvrđivanje merenja kvaliteta softverskog proizvoda
- ✓ Kvantitativno upravljanje procesom testiranja tj. aktivnostima osiguranja kvaliteta softvera u cilju povećanja efikasnosti i efikasnosti otkrivanja i otkrivanja grešaka u toku razvoja softvera





White-box

ili

Metodologija struktturnog
testiranja

ili

Providna ili staklena kutija

TESTIRANJE SOFTVERA- White-box

Program se shvata kao otvorena (bela) kutija čija je unutrašnjost poznata. Opšti cilj testera koji koristi ovu metodu je da proveri svaku putanju u programu.

Analogno pojmu model crne kutije uveden je pojam model bele kutije, kod koga su poznati zakoni ponašanja i procesi u njemu kao dinamičkom sistemu.

Tester bira načine testiranja koda te on određuje scenarije testnog procesa.

DA LI JE OVO MOGUĆE ?

Ako imamo jednostavni program postoji jako puno različitih putanja.

Dakle, generišu se testovi koji izvršavaju sve naredbe u programu, pozivaju sve funkcije i prolaze kroz sve tokove kontrole unutar komponente

Mnogo rešenja...

TESTIRANJE SOFTVERA- White-box

- ✓ Podrazumeva dostupnost izvornog koda, testira se i analizira programski kod, to jest, zasniva se na manuelnoj, statickoj analizi izvornog koda programa.
- ✓ plan testiranja se formira na osnovu strukture programa
- ✓ Dizajn konkretnog softverskog proizvoda

Opšti cilj testera je da proveri svaku putanju u programu.

Kada se vrši Testiranje nekih kompleksnih softverskih rešenja postoji mogućnost da se koriste obe (white i black box) metode.

TESTIRANJE SOFTVERA- White-box

Neke od dobrih osobina su:

- ✓ Testiranje može početi rano u projektu
- ✓ Optimizacija koda i mogućnost pronađenja skrivenih pogrešaka (grešaka).
- ✓ Testiraju se svi delovi koda
- ✓ Metodom analize se može otkriti više (suptilnih) grešaka nego upotrebom alata za automatizovano testiranje
- ✓ Pokrivenost testovima - obezbediti da svaka linija koda bude proverena nekim testom, odnosno što testovi koriste ulaze kako bi pokrili sva grananja u kodu
- ✓ Prednost - pokrivenost testovima obezbeđuje određenu sigurnost

TESTIRANJE SOFTVERA- White-boxA

Neki od nedostaka su:

- Vremenski zahtevno
- Kod korisničkog unosa prilikom izvršavanja programa, je teško utvrditi da li postoji neki bagovi (grešaka)ili ne
- Mana - veliki broj testova je neupotrebljiv čim se kod promeni
- Testovi jako kompleksni i potrebno je da ih piše i testira veoma stručan tester koji dobro poznaje razvoj softvera
- Testiranje bele kutije oduzima puno vremena, potrebno je vreme za potpuno testiranje većih aplikacija.
- Nemogućnost testiranja softverskih performansi (vreme odgovora, pouzdanost, load)

TESTIRANJE SOFTVERA- White-box

Analogno pojmu model crne kutije uveden je pojam model bele kutije, kod koga su poznati zakoni ponašanja i procesi u njemu. Metoda podrazumeva detaljnu analizu, što podrazumeva puno utrošenog vremena.

Testeri moraju da analiziraju kod koji je neophodan kako bi se pronašao uzrok. (kada nešto ne radi)

Ispitivanje na osnovu strukture i staklene kutije su druga imena za ovaj metod.

Osnova za testiranje je izvorni kod test objekta, Pa se zato u literaturu se koristi termin testiranje zasnovano na strukturi ili kodu.

POTPUNO
POZNAVANJE
'SOURCE CODE'

*Testiranje kao
Developer*

TESTIRANJE SOFTVERA- White-box

Strategija bele kutije je testiranje koje je veoma vremenski zahtevno i obično se primenjuje na manje delove sistema.

Korisno je za nalaženje grešaka u dizajnu, za nalaženje logičkih grešaka Primarni izvor za projektovanje testova je izvorni kod sa "akcentom" na tok kontrole i tok podataka.



TESTIRANJE SOFTVERA- White-box

U literaturu se može videti i sledeći termin "glass box" testiranje .

U slučaju tehnike **bele kutije**, unutrašnja struktura sistema za testiranje je poznata.

Tester ima pristup kodu softvera koji se testira i razume način njegove implementacije.

- Akcenat je na strukturnim elementima kao što su naredbe i petlje.

Testiranje zasnovano na principu bele kutije se uobičajeno koristi za verifikaciju softvera.

Kriterijum uobičajene „bele kutije“ je da izvrši svaki izvršni iskaz tokom testiranja i da svaki ishod tokom testiranja bude zapisan u dnevnik testiranja.



TESTIRANJE SOFTVERA- White-box

Kod ovog testiranja se celokupan izvorni kod uzima u obzir tokom testiranja, što olakšava uočavanje greške, čak i kada su softverske pojedinosti nejasne ili nekompletne.

U slučaju testiranja metodom crne kutije, nije moguće upoznati se sa korišćenom logikom, pa se za izradu testnih slučajeva moraju koristiti specifikacije ili intuicija.

Posmatranjem koda, tokom testiranja metodom bele kutije, koristi se struktura koda, na osnovu koje se definišu vrste potrebnih testova. U mnogim slučajevima, preporučeni testovi ispituju putanje u kodu i prolazak kroz određene strukture koda

TESTIRANJE SOFTVERA- White-box

Često se tehnike testiranja metodom bele kutije povezuju sa metrikama pokrivanja, kojima se meri procenat izabranih putanja za testiranje, koje se preoveravaju u pojedinačnim testnim slučajevima.

Strukturno testiranje, poznato kao metoda bele ili staklene kutije, je tehnika testiranja gde je testerima poznata implementacija softvera.

Za razliku od modela crne kutije koji je fokusiran na to što softver radi, model bele kutije je fokusiran na to kako softver radi

TESTIRANJE SOFTVERA- White-box

Kod white box testiranja postoje više alternativnih pristupa planiranja testnih slučajeva za testiranje podataka i korektnosti programa.

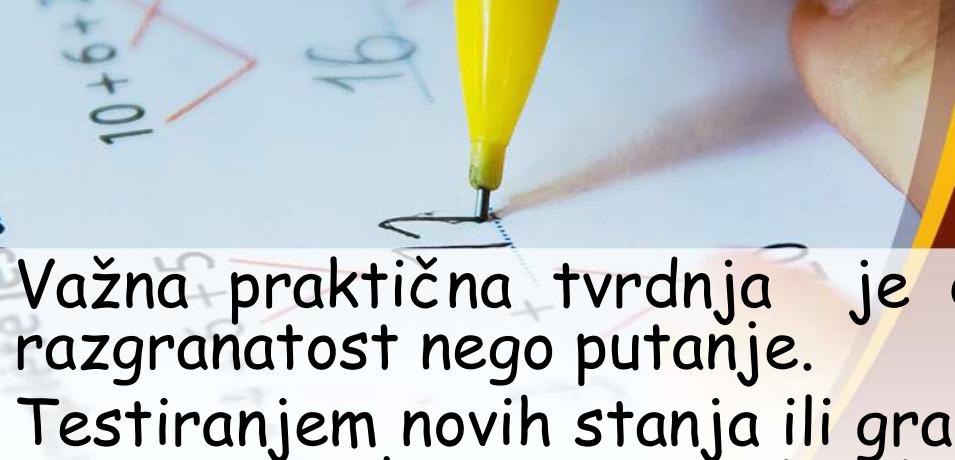
- Testiranje naredbi
- Testiranje grana
- Testiranje uslova
 - jednostavno testiranje uslova
 - višestruko testiranje uslova
 - minimalno višestruko testiranje uslova
- Testiranje putanja



Testiranje naredbi

Fokus je dat samo na pokrivenost svih naredbi. Ovde postoji problem jer ne pokriva sve ili dovoljno moguće tokove programa.

- Ono ne pokriva izvršavanje grana koje nemaju naredbe
- Čak i za ovakve testove je veoma teško obezbediti veliku (100) pokrivenost
 - Pojedine izuzetke u kodu je teško izazvati



Testiranje grana

Važna praktična tvrdnja je da je lakše testirati stanje i razgranatost nego putanje.

Testiranjem novih stanja ili grana uvek se može poboljšati sam proces struktturnog testiranja. Prema tome, prvo se treba koncentrisati na netestirane grane izlaza, a zatim se premešta na netestirane putanje.

Konačno, možda i najznačajnije, automatski alat obezbeđuje tačnu verifikaciju koji bi trebao da zadovolji zadate uslove testiranja programa.

Manualno poreklo testiranja podataka je proces sklon greškama, u kojem nameravane putanje nisu često izvođene za set datih podataka



Testiranje grana

Pronalazi i dizajnira minimalni broj testnih slučajeva tako da se izvrše svi mogući pravci kretanja (grane).

Tehnika je naprednija od testiranja naredbi

- Grane koje postoje u grafu toka programa se analiziraju
- Prati se i analiziraju se koje grane toka programa se izvršavaju (od uslovnih izraza)
- Pokrivenost se određuje u odnosu na procenat grana izvršenih u okviru testa

$$(\text{broj izvršenih grana} / \text{ukupan broj grana}) * 100\%$$

Potrebno je pokriti i grane bez naredbi

- npr. if naredbu bez else dela

Ovaj koncept nije praktičan u mnogim slučajevima jer je potrebno puno resursa i testnih slučajeva za njegovo izvršavanje.



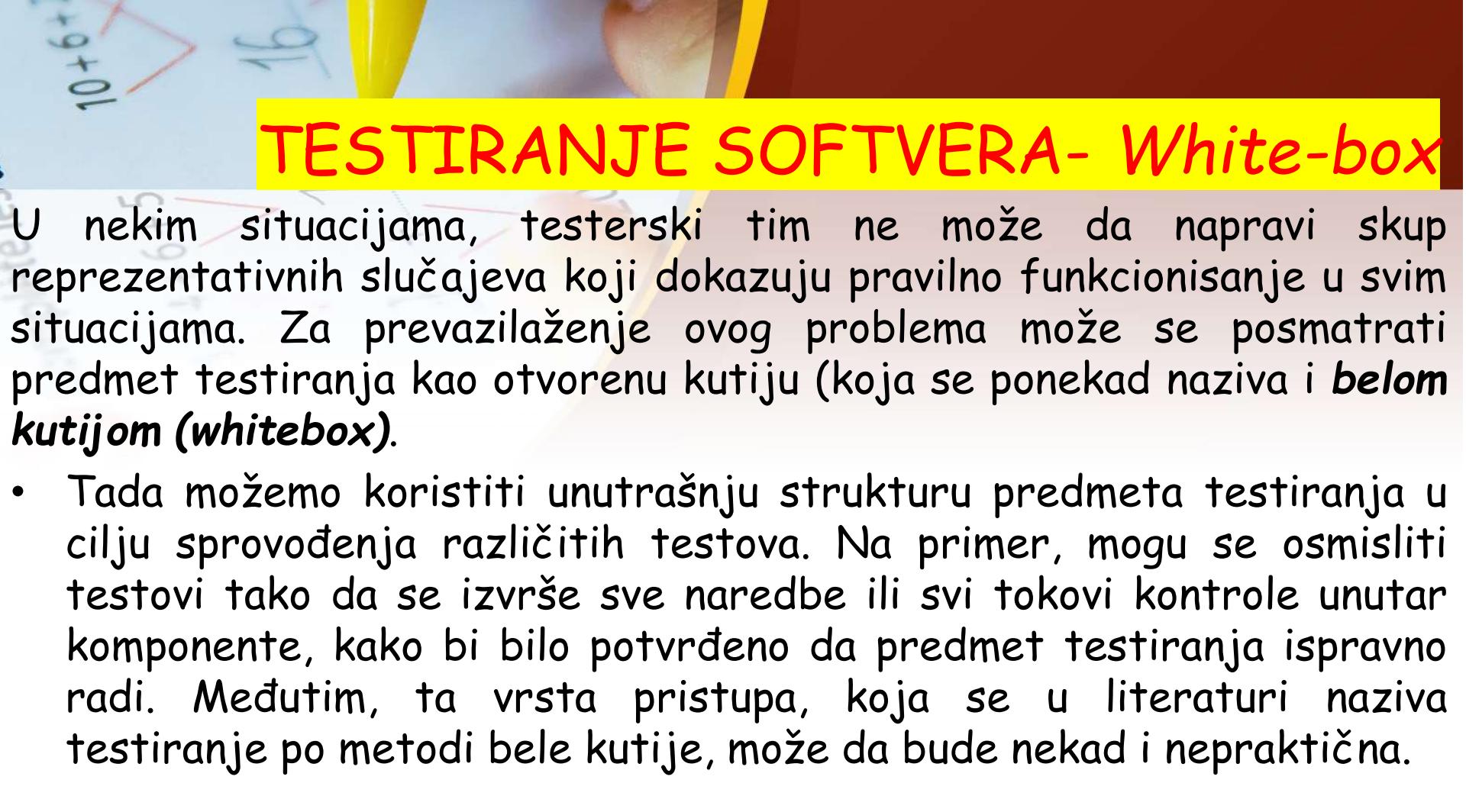
Testiranje uslova

Analiziraju se konkretni uslovni izrazi koji uzrokuju prelazak toka programa na određenu granu

Za razliku od testiranja grana koje se fokusira na proveru grana, testiranje uslova se bavi time šta dovodi do toga da program pređe u određenu granu

Uslov može biti složen

- ✓ više prostih uslova povezanih logičkim operatorima
- ✓ prost uslov je onaj koji
 - ✓ daje logičku vrednost kao rezultat i
 - ✓ ne sadrži logičke operatore
 - ✓ sadrži relacione operatore koji daju logički rezultat



TESTIRANJE SOFTVERA- White-box

U nekim situacijama, testerski tim ne može da napravi skup reprezentativnih slučajeva koji dokazuju pravilno funkcionisanje u svim situacijama. Za prevazilaženje ovog problema može se posmatrati predmet testiranja kao otvorenu kutiju (koja se ponekad naziva i **belom kutijom (whitebox)**).

- Tada možemo koristiti unutrašnju strukturu predmeta testiranja u cilju sprovodenja različitih testova. Na primer, mogu se osmislit testovi tako da se izvrše sve naredbe ili svi tokovi kontrole unutar komponente, kako bi bilo potvrđeno da predmet testiranja ispravno radi. Međutim, ta vrsta pristupa, koja se u literaturi naziva testiranje po metodi bele kutije, može da bude nekad i nepraktična.

TESTIRANJE SOFTVERA- White-box

Plan testiranja se određuje na osnovu elemenata implementacije softvera, kao što su programski jezik, logika i stilovi.

Testovi se izvode na osnovu strukture programa.

Specifičnim testovima može se proveravati postojanje beskonačnih petlji ili koda koji se nikada ne izvršava.

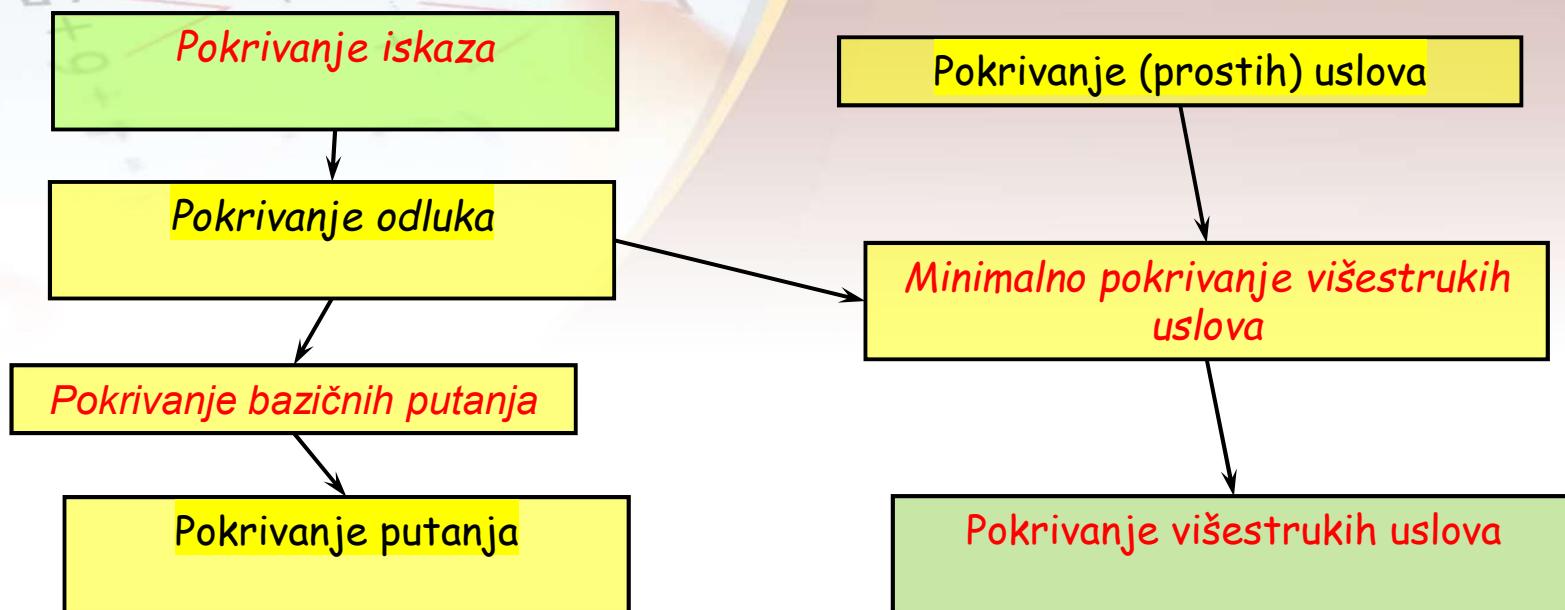
Pri dizajnu test slučajeva uzima se u obzir koji delovi koda će se izvršiti pri pokretanju test slučajeva



TESTIRANJE SOFTVERA- White-box

- Tehnike zasnovane na toku kontrole
- Pokrivanje iskaza
- Pokrivanje odluka
- Pokrivanje putanja
- Pokrivanje bazičnih putanja
- Minimalno pokrivanje višestrukih uslova
- Tehnike zasnovane na toku podataka

Poređenje metoda



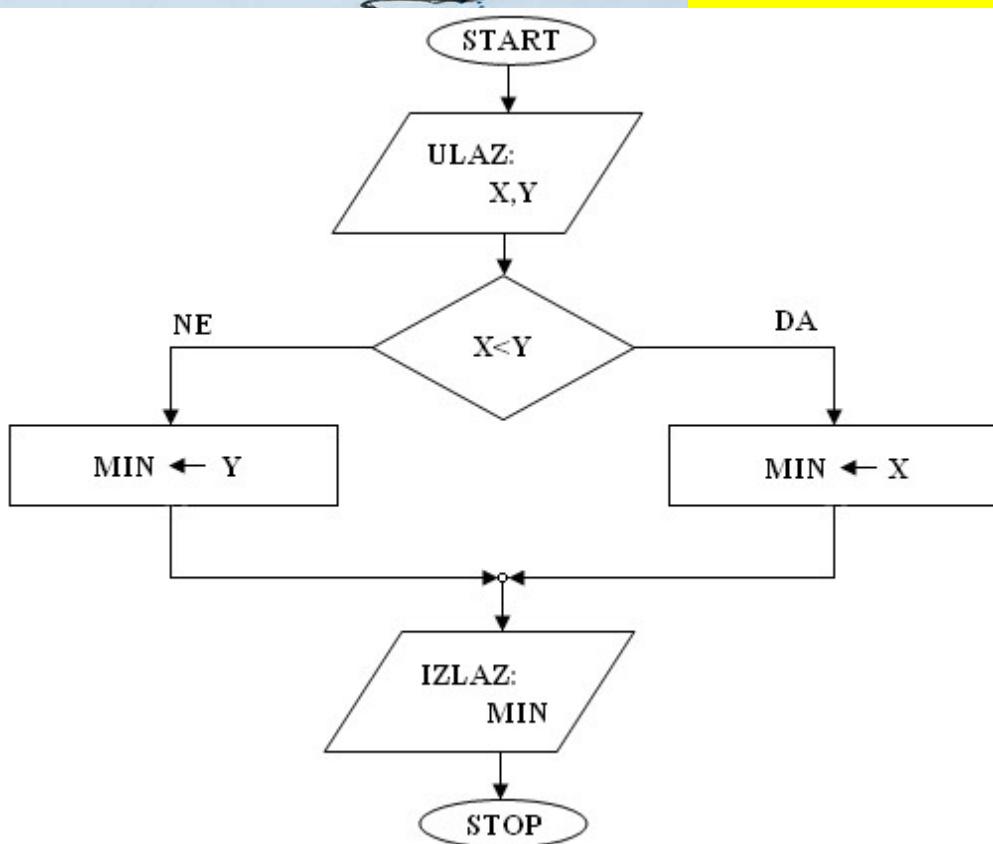


TESTIRANJE SOFTVERA

Koliko testnih slučajeva je potrebno da se postigne 100 % pokrivenost odluka (*decision coverage*).

Cilj pokrivanja uslova je proveriti pojedinačne rezultate za svaki logički uslov.

TESTIRANJE SOFTVERA

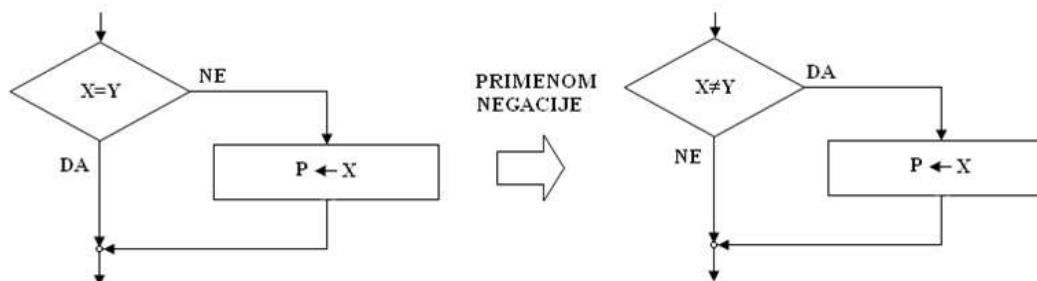
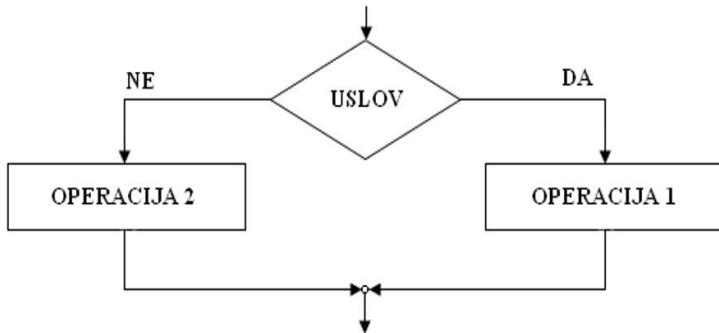


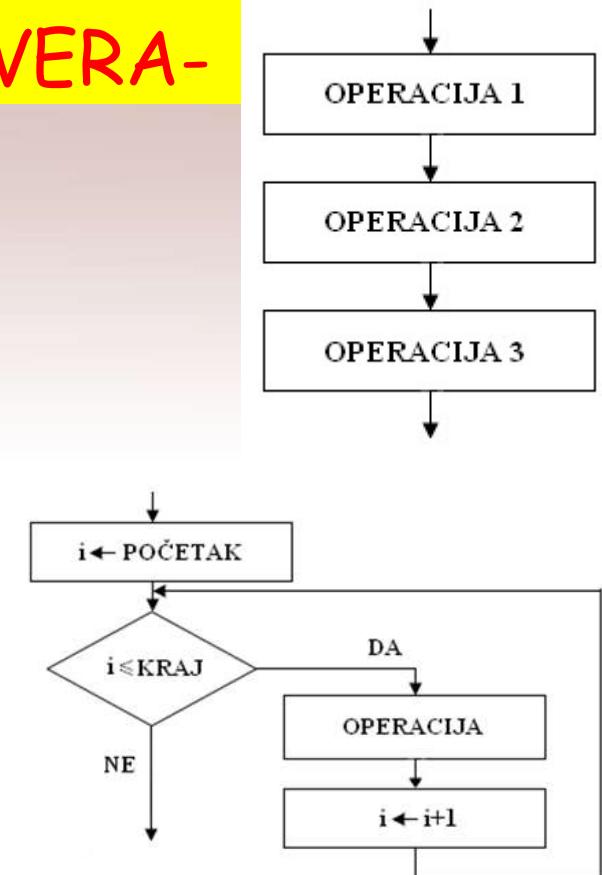
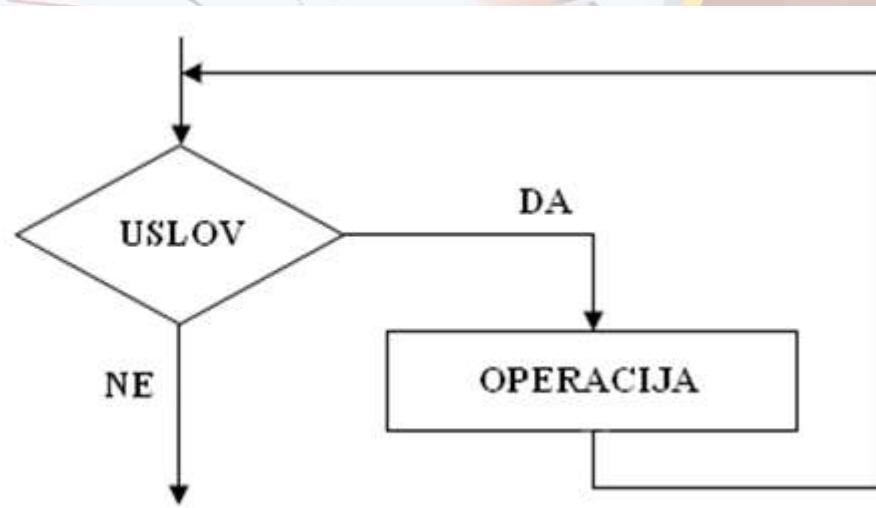
- Da li su svi uslovi ispunjeni ?
Na primer / šta se dešava kada je

$X=Y$

Smisliti dobre - odgovarajuće poruke da budu [to jasniji korisnicima]

TESTIRANJE SOFTVERA- White-box





TESTIRANJE SOFTVERA- White-box

Tehnike zasnovane na kontroli toka

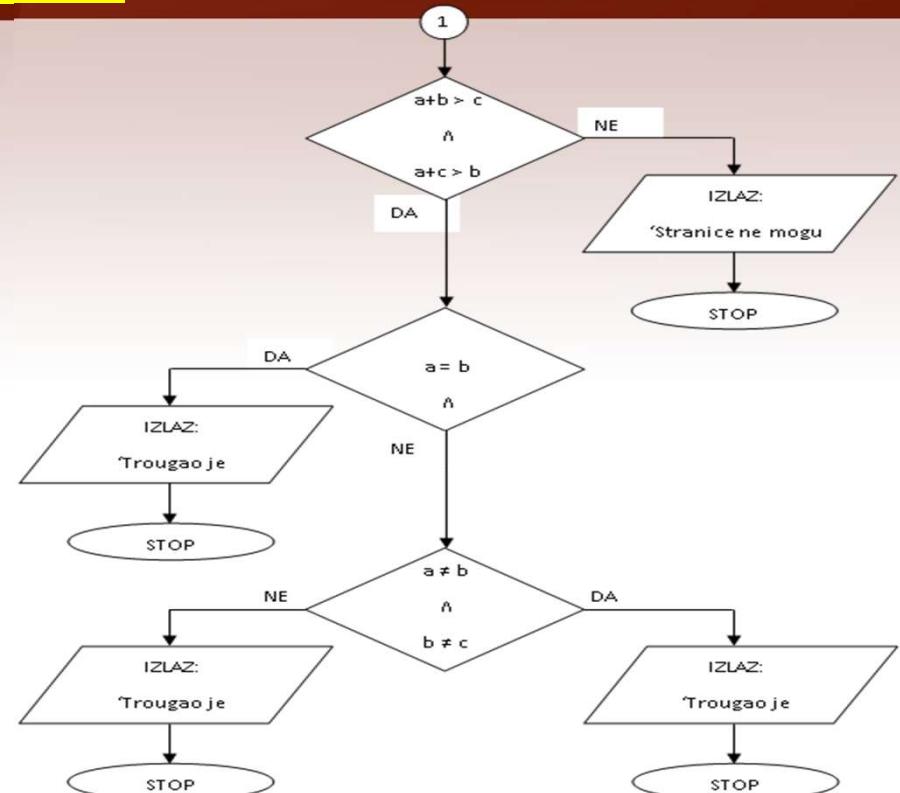
- Pokrivenost višestrukih uslova (Multiple Condition Coverage)
- Modifikovana pokrivenost odluka i uslova (MC/DC)
- Pokrivenost uslova (Condition Coverage)
- Pokrivenost odluka i uslova (Decision/Condition Coverage)
- Pokrivenost odluka ili pokrivenost grana (Decision or Branch Coverage)
- Minimalna pokrivenost višestrukih uslova (Minimal Multiple Condition Coverage)
- Pokrivenost iskaza (Statement Coverage) negde se zove i pokrivenost koda

TESTIRANJE SOFTVERA- White-box

Komponenta sa mnogo grananja i petlji sadrži veliki broj putanja koje treba proveriti.

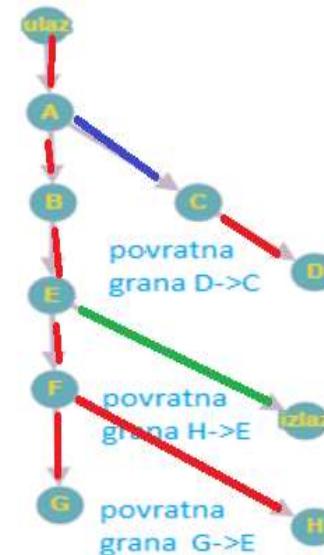
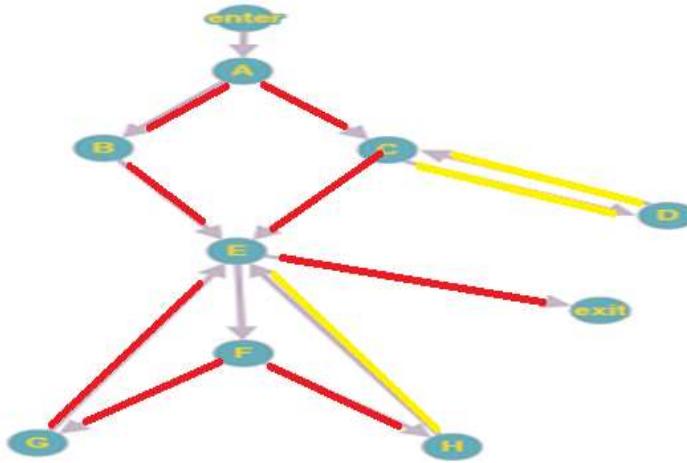
Teško je temeljno testirati komponentu sa velim brojem putanja.

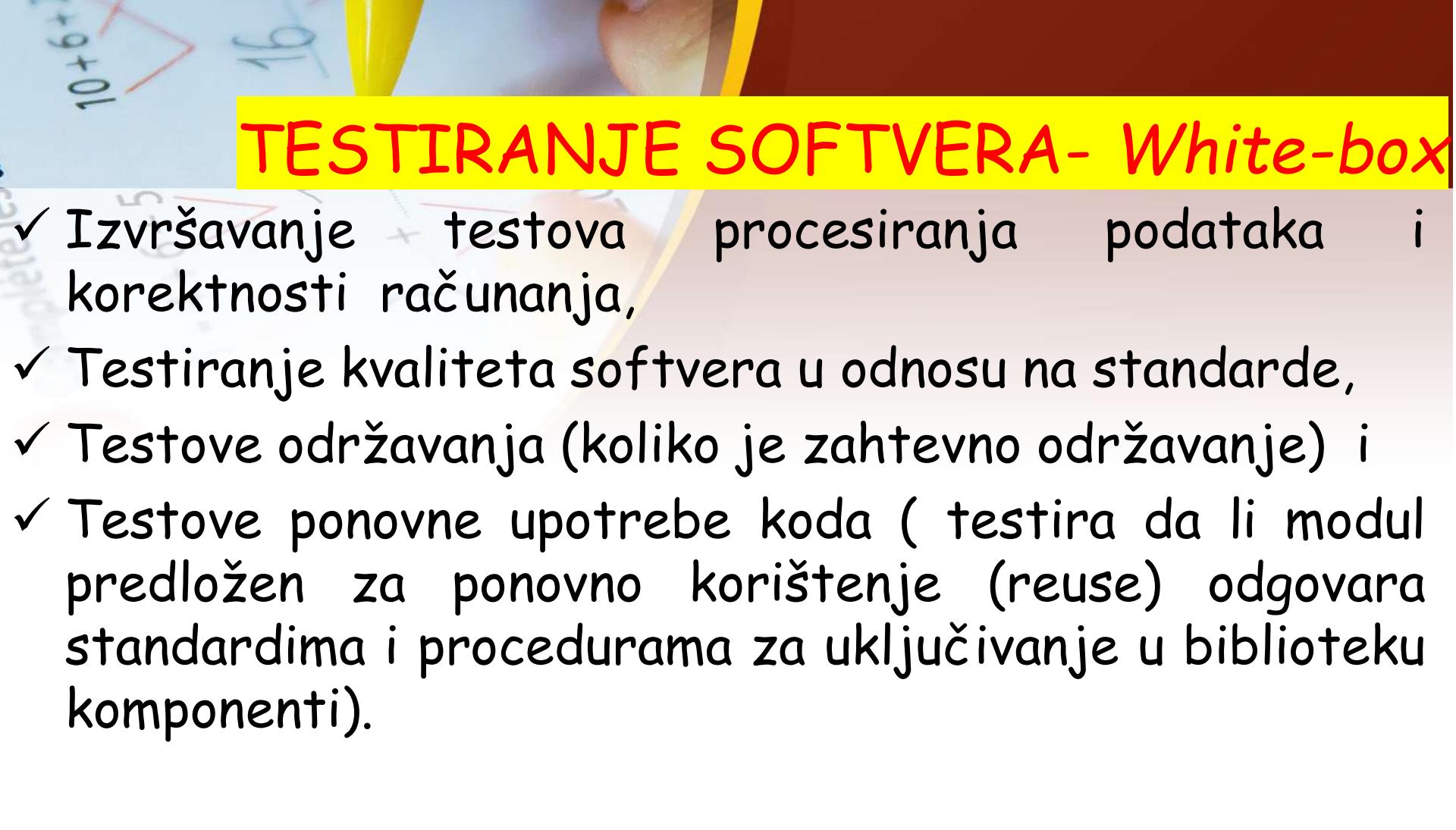
Preporučuje se za testiranje algoritama.





Dijagram toka (flow chart) i graf programskog toka (flow graph) se koriste za određivanja broja puteva i broja linija





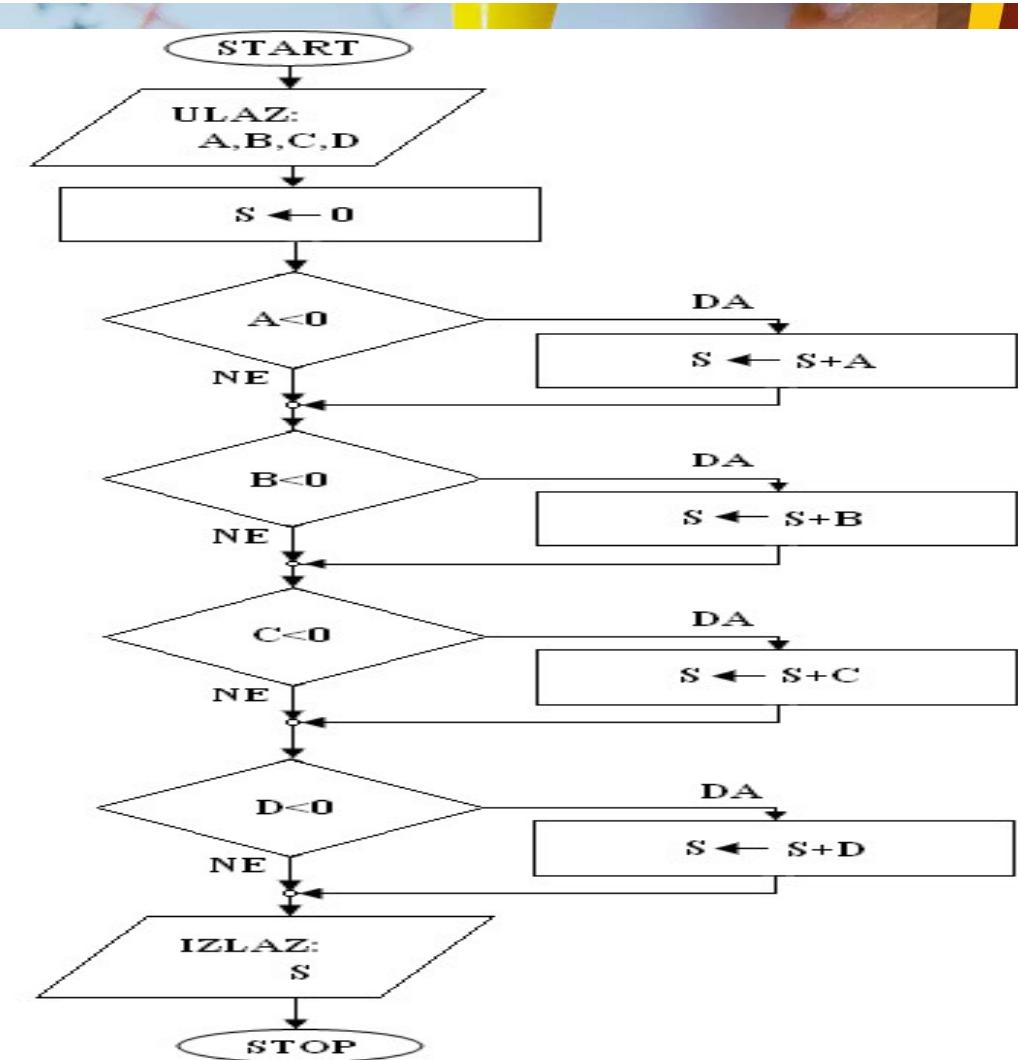
TESTIRANJE SOFTVERA- White-box

- ✓ Izvršavanje testova procesiranja podataka i korektnosti računanja,
- ✓ Testiranje kvaliteta softvera u odnosu na standarde,
- ✓ Testove održavanja (koliko je zahtevno održavanje) i
- ✓ Testove ponovne upotrebe koda (testira da li modul predložen za ponovno korištenje (reuse) odgovara standardima i procedurama za uključivanje u biblioteku komponenti).



Testovi procesiranje podataka i korektnosti računanja

- ✓ Zadatak testova korektnosti ("white box correctness test") je da se ispita svaka operaciju.
- ✓ Ispitivanje se vrši sa pripremljenim testnim slučajevima.
- ✓ Testni slučaj je skup akcija koji se izvršava da bi se izvršila verifikacija pojedinačne karakteristike ili funkcionalnosti aplikacije.
- ✓ Testni slučaj sadrži ulazne podatke i na osnovu tih podataka se računa izlaz za koji se proverava da li je on očekivan ili nije.
- **Odobrene test procedure** - Procesi testiranja se izvršavaju u skladu sa testnim planom i procedurama testiranja koje su odobrene



TESTIRANJE SOFTVERA

- Da li su svi uslovi ispunjeni ?
Unos slova ?
Na primer / šta se dešava kada je
A, B,C,D promeni znak (>) Odgovarajuće poruke



Pokrivanje iskaza - Statement coverage

- ✓ Test primeri se tako projektuju da se svaki iskaz programa izvrši bar jednom.
- ✓ Ne možemo znati da li u nekom iskazu postoji greška ukoliko ga ne izvršimo

Ograničenja metoda pokrivanja iskaza je :

- ✓ Ukoliko se utvrdi da se iskaz ispravno izvršava za jednu ulaznu vrednost to znači da :
- ✓ Nema garancije da će se ispravno izvršavati za svaku drugu ulaznu vrednost

TESTIRANJE SOFTVERA- White-box

Poređenje različitih tehnika

- Pokrivanje uslova
- Ne garantuje pokrivanje svih odluka
- Samim tim nije garantovano ni pokrivanje svih iskaza



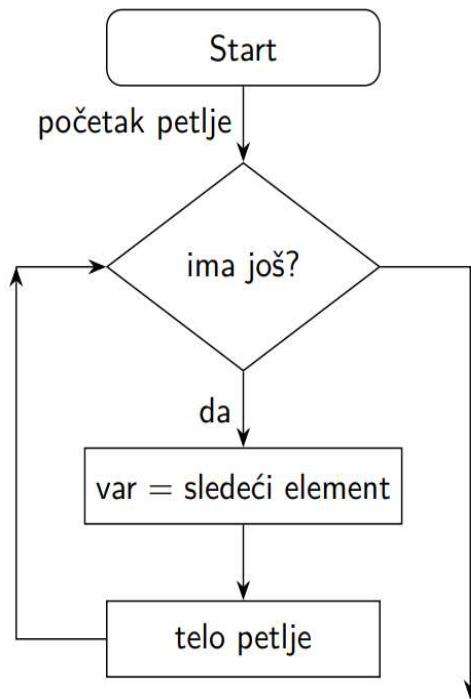
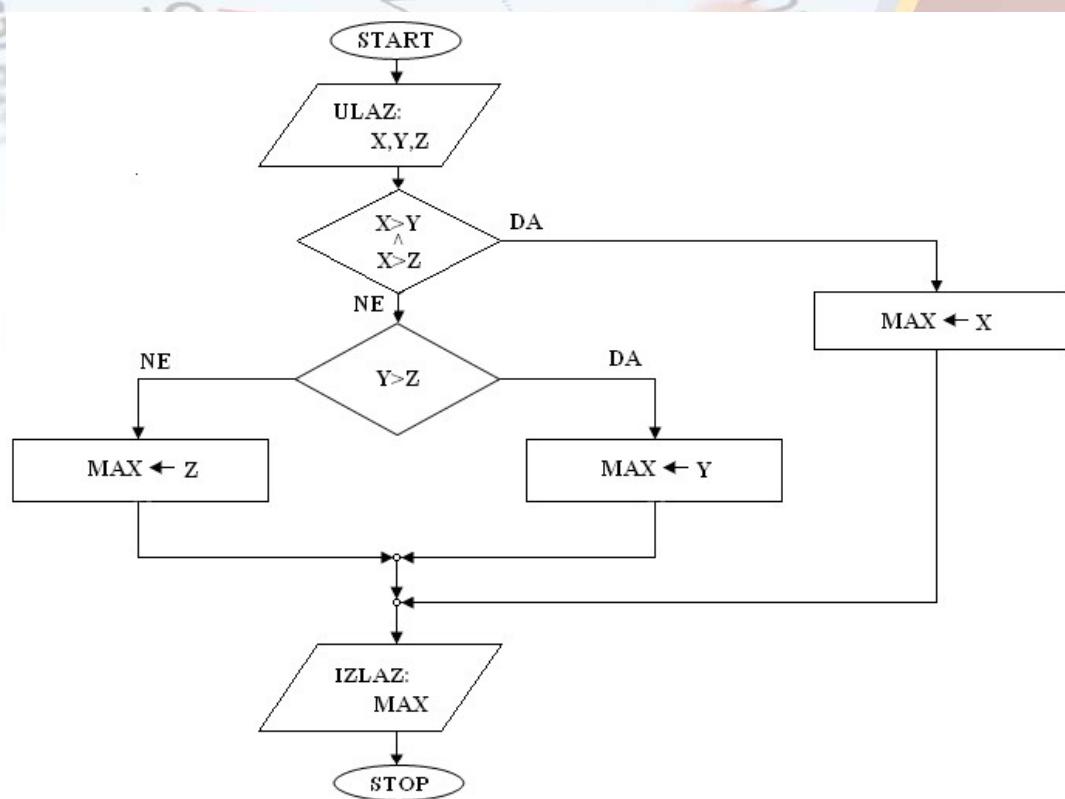
TESTIRANJE SOFTVERA

Pokrivenost testiranja (coverage) je mera do koje je neki softver ili određena softverska komponenta istestirana nekim skupom testova, u smislu procenta obuhvaćenih stavki.

Da li su sve test stavke obuhvaćene testom ?

U slučaju da pokrivenost nije 100%, potrebno je proširiti skup testova s novim testovima. Pokrivenost se povećava tako da se sistemski pišu novi testovi da bi se pokrili svi delovi softvera (projekta), odnosno da se svi delovi softvera izvrše bar jednom

Pokrivanje iskaza - Statement coverage





Decision/branch coverage

Prednosti pokrivenosti testiranja:

- Pomaže u kreiranju dodatnih testova kako bi se povećala pokrivenost;
- Pomaže u pronalaženju delova programa koji uopšte nisu pokriveni testovima;
- Određuje kvantitativni meru pokrivenosti koda, koja indirektno predstavlja meru kvaliteta softvera.

Pokrivanje odluka

Mane pokrivenosti testiranja:

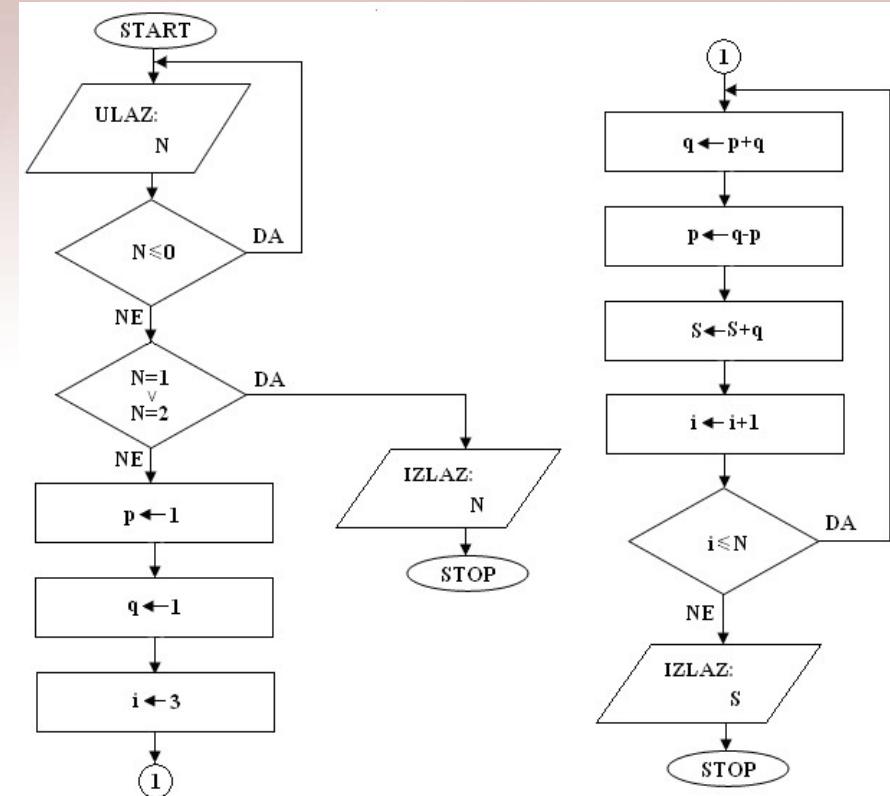
- Može da izmeri pokrivenost koda koji je napisan, ali se ne zna ništa o delu koji još uvek nije napisan;
- Nije efikasna u slučaju da specificirana funkcionalnost nije implementirana ili ukoliko je izostavljena iz specifikacije, pošto se posmatra samo kod.

TESTIRANJE SOFTVERA

```
• switch(k) {  
•     •     •  
•         case 6: cprintf("PETAK");  
•             break;  
•         case 0: cprintf("SUBOTA");  
•             break;  
•         case 1: cprintf("NEDELJA");  
•             break;  
•         case 2: cprintf("PONEDELJAK");  
•             break;  
•         case 3: cprintf("UTORAK");  
•             break;  
•         case 4: cprintf("SREDA");  
•             break;  
•         case 5: cprintf("CETVRTAK");  
•     }  
• }
```



Zbir prvih N članova
Fibonačijevog niza.
Fibonačijev niz je:
 $1, 1, 2, 3, 5, 8, 13, 21, \dots$



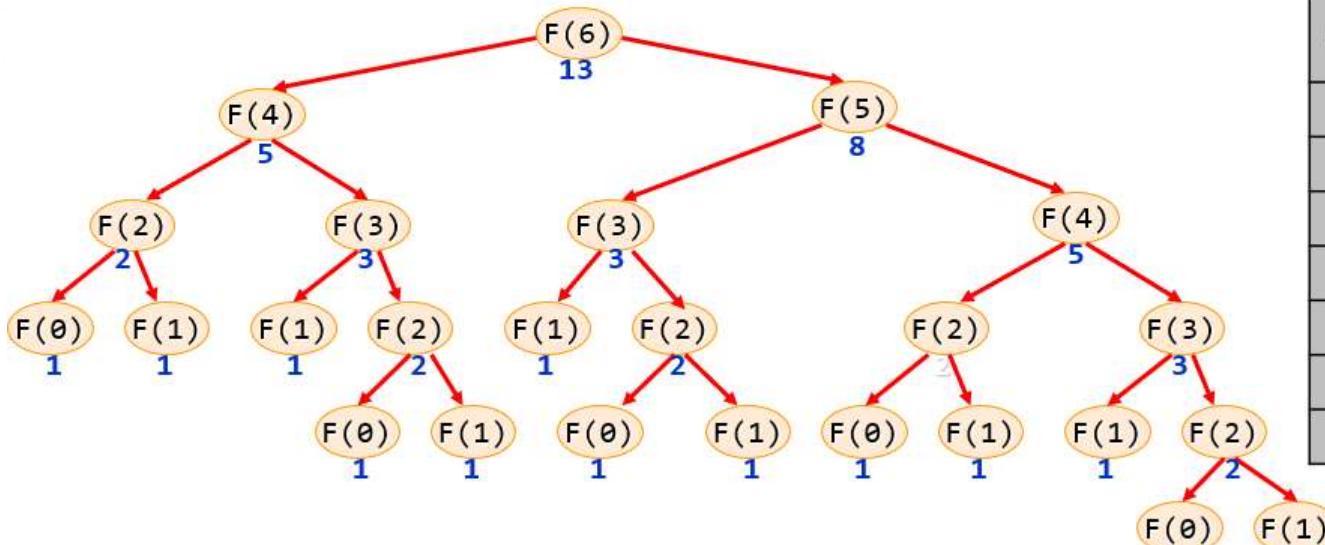
FIBONACIJEV BROJ

1, 1, 2, 3, 5, 8, 13, 21, 34,...

$$F_0 = F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2}; \quad i > 1$$

Fibonačijevi brojevi predstavljaju niz 1,1,2,3,5,8,... sekvenca počinje sa 1,1 sledeći broj se računa kao zbir prethodna dva



Poziv	Broj izvršavanja
F(6)	1
F(5)	1
F(4)	2
F(3)	3
F(2)	5
F(1)	8
F(0)	5

FIBONAČIJEV BROJ

File Edit Format Run Options Window Help

```
# KOMENTARI
# 1 1 2 3 5 8 13 21

print(" FIBONACIJEV NIZ / FIBONACI")
# DEFINISEMO FUNKCIJU
def fibonacijevbroj(n):
    if n == 1:
        return 1

    elif n == 2: #
        return 1

    else: # Rekurzivni pozivi
        return fibonacijevbroj(n-1) + fibonacijevbroj(n-2)
n = int (input ("Unesite poziciju broja za Fibonačijev broj: "))
# Izlaz prikaz Fibonačijevog broja
print ("Fibonačijev broj za indeks", n, "je", fibonacijevbroj(n))
```

```
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
=====
RESTART: C:/Users/markoni/Desktop/PYTHON
FIBONACIJEV NIZ / FIBONACI
Unesite poziciju broja za Fibonačijev broj: 7
Fibonačijev broj za indeks 7 je 13
```

TESTER:

**Da li postoji DOBRONAMERNI
KORISNIK DA ILI NE ?**

Mane:

U kodu nema dovoljan broj komentara

Unos negativnog broja

Unos slova

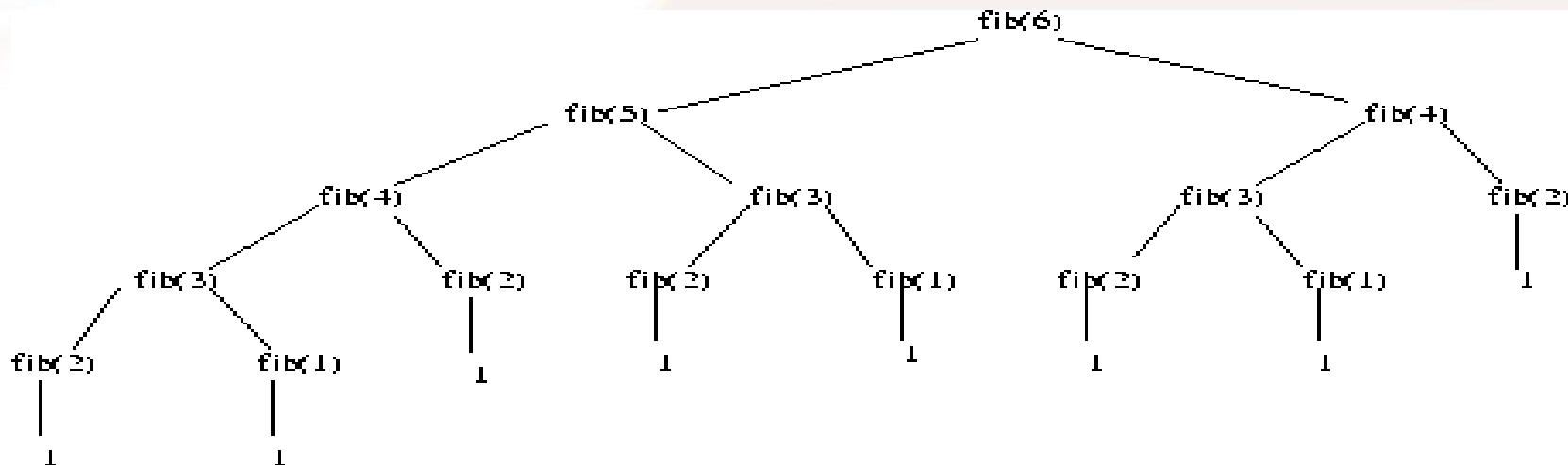
Fibonacci rekurzivno

:

- ✓ rekurzija se zasniva na manjim vrednostima
- ✓ postoji osnovni slučaj koji se ne oslanja na rekurziju

Da li će rekurzivna varijanta raditi efikasno?...

Rekurzivno rešenje je neefikasno jer obavlja puno ponovljenih izračunavanja!





- I na prstom primeru ili na najedntavni primer može da se desi da i on bude težak za testere

PRIMER> Napisati deo C programa koji utvrđuje najveću vrednost između tri uneta broja x , y i z i rezultat upisuje u promenljivu max .

1 Način

```
•     if(x>y)
•     {
•         if(x>z)           max=x;
•         else max = z;
•     }else{
•         if(y>z) max = y;
•         else max = z;
•     }
```

Pokrivanje uslova

2 Način

```
•     if(x>y&&x>z) max = x;
•     else {
•         if(y>z) max = y;
•         else max = z;
•     }
• }
```

3 Način

```
•     max = x;
•     if(y>max) max = y;
•     if(z>max) max = z;
```

• TESTERI:

- Koji je najefikasniji način
- Da li testiramo sve načine ?
- RAD PO MODULIMA
- Problem kad svako u timu piše delove koda koje nisu koordinisane sa različito sa PROJEKT MENADZER ILI PRODUKT MENADZER
- OPTIMIZACIJA KODA



Data flow coverage

- Za dizajn testnih slučajeva biraju se putevi u skladu sa lokacijama definicije i korištenja promenljivih.
- Ova tehnika nije praktična za intezivno korištenje.
- Pogodna je za module sa ugnježđenim if izrazima i petljama.



Matematički bazirano struktorno testiranje ima mnogo prednosti. Neke od prednosti su te što svi osnovni setovi putanja pokrivaju sve grane grafa toka programa i one zadovoljavaju kriterijume strukturnog testiranja, a samim time automatski zadovoljavaju slabije grane i osnovne kriterijume.

Sledeće kod struktturnog testiranja je to, da je testiranje proporcionalno kompleksnosti, odnosno minimalni broj testiranja potreban da zadovolji kriterijum struktturnog testiranja je zapravo ciklična kompleksnost.

Pored toga, pošto je minimum potrebnih brojeva testova poznat unapred, struktorno testiranje podžava planiranje testiranja i traži konstantan nadzor procesa testiranja



TESTIRANJE SOFTVERA

Druga prednost struktturnog testiranja je ta, da precizne matematičke interpretacije „nezavisnog“ kao „linearno nezavisnog“ struktturnog testiranja garantuju nezavisno testiranje izlaza. Na osnovu ovoga se može videti da, za razliku od drugih uobičajenih strategija testiranja, struktorno testiranje ne dozvoljava unošenje proizvoljnih ulaznih podataka za testiranje.

Prikazan je jedan C program:

```
Prazna funkcija()
{
    if (uslov1)
        a = a + 1;
    if (uslov2)
        a = a - 1;
}
```

DA LI JE OVO PRIMER DELA DOBROG PROGRAMA ?



TESTIRANJE SOFTVERA

U ovom C primeru prepostavlja se da je vrednost promenljive „a“ neizmenjena pod bilo kojim uslovima (samim tim ta prepostavka je očigledno netačna).

Obzirom da u ovom C programu postoje dva uslova, pri čemu se moraju posmatrati oba ishoda, proizilazi da postoji četiri moguća rešenja.

Kriterijum testiranja može biti zadovoljen sa dva testa koji nisu uspeli da otkriju grešku.

Prvo neka obe odluke izlaza budu NETAČAN, u kojem slučaju vrednost promenljive „a“ nije promenjena.

Zatim se postavljaju obe odluke izlaza da budu TAČAN, u kojem slučaju je vrednost promenljive „a“ prvo povećana, a zatim smanjena na početnu vrednost.

Sa druge strane, strukturno testiranje garantovano uočava grešku. Potrebne su tri nezavisne test putanje, tako da makar jedan (od dva ishoda) bude TAČAN, a drugi NETAČAN, ostavljajući vrednost promenljive rastućom ili opadajućom, prema čemu se detektuje greška.



TESTIRANJE SOFTVERA

Mnogi autori pokazuju korelaciju između kompleksnosti i grešaka, jednako kao i odnos između kompleksnosti i teškoće da se razume kako je nastala potencijalna greška.

Pouzdanost je kombinacija testiranja i razumevanja.

" Softver je toliko dobar da je test koji je izvršen adekvatan da predviđa željeni nivo poverenja. "McMillan

Pošto kompleksnost čini softver težim za testiranje i težim za razumevanje, kompleksnost je blisko vezana sa pouzdanošću.

Sa jedne tačke gledišta, kompleksnost meri napore potrebne za postizanje datoq nivoa pouzdanosti.

```

File Edit Format Run Options Window Help
def calc():
    while True:
        print ("PROGRAM KALKULATOR!")
        print ("Izaberite stavku menija:")
        print (" ")
        print ("1) SABIRANJE")
        print ("2) ODUZIMANJE")
        print ("3) MNOŽENJE")
        print ("4) DELJENJE")
        print ("5) IZLAZ")
        print (" ")
        izbor = int (input("Unesite jednu od  opciju:"))
        if izbor == 1:
            n1= int (input ("Unesi prvi broj A:"))
            n2= int (input ("Unesi drugi broj B:"))
            rez = n1+n2
            print (" Rezultat /SABIRANJE JE A +B:\n ", rez )
        elif izbor == 2:
            n1= int (input ("Unesi prvi broj A:"))
            n2= int (input ("Unesi drugi broj B:"))
            rez = n1-n2
            print (" Rezultat /Oduzimanje A - B \n", rez )
        elif izbor == 3:
            n1= int (input ("Unesi prvi broj A:"))
            n2= int (input ("Unesi drugi broj B:"))
            rez = n1 * n2
            print (" Rezultat /MNOZENJE A * B \n", rez )
        elif izbor == 4:
            n1= int (input ("Unesi prvi broj A:"))
            n2= int (input ("Unesi drugi broj B:"))
            if n2 == 0:
                print("Drugi broj ne moze biti 0")
            else :
                rez = n1 / n2
                print (" Rezultat / DELJENJE A / B \n", rez )
        elif izbor ==5:
            print (" KORISTIMO KALKULATOR " )
            break
        ch= input("IZABERI K da izadjes iz programa")
        if ch in "Kk":
            print (" KORISTIMO KALKULATOR " )
            break

```

• Poziv

```

File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug
AMD64) ] on win32
Type "help", "copyright", "credits" or "
>>>
===== RESTART: C:/Users/markoni/Desktop/
PROGRAM KALKULATOR!
Izaberite stavku menija:
1) SABIRANJE
2) ODUZIMANJE
3) MNOŽENJE
4) DELJENJE
5) IZLAZ

Unesite jednu od  opciju:4
Unesi prvi broj A:5
Unesi drugi broj B:0
Drugi broj ne moze biti 0
IZABERI K da izadjes iz programa

```

Graf toka kontrole

Predstavlja reprezentaciju programa u obliku grafa i implicitno pokazuje putanje izvršavanja.

Čvorovi grafa su instrukcije.

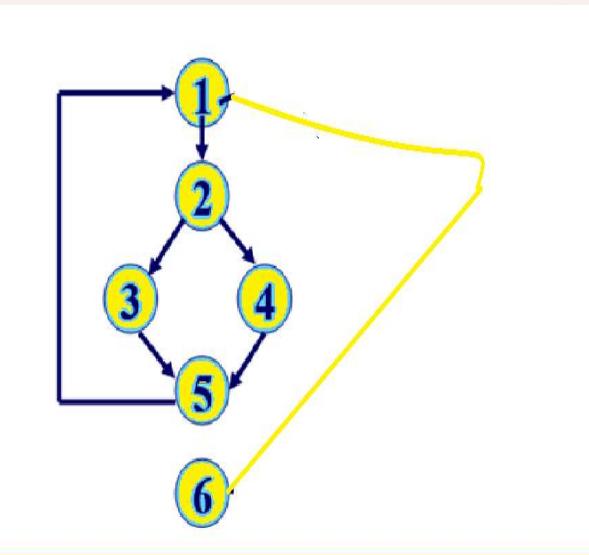
Graf toka kontrole programa opisuje redosled u kome se izvršavaju instrukcije programa

Graf kontrole toka je grafovska reprezentacija svih puteva kojima se može proći kroz program tokom njegovog izvršavanja.

Ovaj graf se pravi odvojeno za svaku funkciju.

•

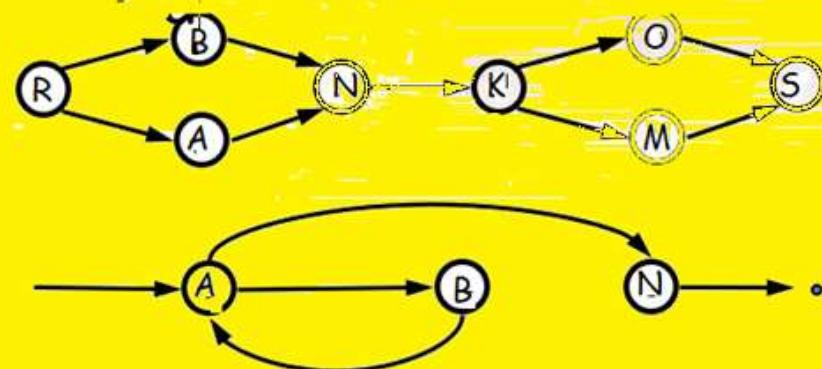
- Svaki čvor grafa kontrole toka predstavlja osnovni blok naredbi koje ne sadrži nikakvo grananje, pa se stoga izvršavaju sekvenčano.



Graf toka kontrole

Treba zadati neki kriterijum po kome se vrši testiranje:

Sve putanje u **Grafu toka kontrole** treba dobro rešiti pre svega serijom testova





METRIKA I KVALITET SOFTVERA

Još početkom '80-ih počeo se prvi put pojavljivati ideje da se na neki način ustanove ili procene karakteristika softvera koje bi se mogle na neki način izmeriti.

Na taj način mogla bi se izvršiti komparacija za određeni softver, analizirati njegovi atributi, ali i proces njegovog razvoja i rada - Tu je Životni ciklus softvera,

METRIKA I KVALITET SOFTVERA

Prilikom izgradnje softvera, veoma je važno znati da se greške u softveru mogu naći u bilo kojoj fazi njegovog razvoja.

Zato je bitno da se greške pronađu što je pre moguće.
Da se MORA pronaći način procene broja potencijalnih grešaka u sistemu.

Ne možete kontrolisati ono što ne možete meriti" - Tome DeMarco (još davne 1982 godine)

METRIKA I KVALITET SOFTVERA

Metrike za pouzdanost dizajna i koda se uglavnom odnose na veličinu softvera izraženu u broju linija koda (LOC-Lines Of Code), kompleksnost koda izraženu preko ciklomatskog broja (McCabe) i modularnost softvera.

Što je modul kompleksniji veće su poteškoće u negovom razumevanju i veća je verovatnoća defekata nego u manje kompleksnim modulima .

Iako su veličina i kompleksnost korisne metrike često se koristi i njihova korelacija da bi se dobile dodatne informacije.

Za kvalitet objektno orijentisanih struktura koriste se OO metrike.

METRIKA I KVALITET SOFTVERA

Broj linija koda može biti izbrojan samo po završetku programa, što se dešava u veoma kasnoj fazi projekta. /Myers - još davne 1982 godine

Alternativna metrika - metoda funkcionalnih tačaka meri veličinu projekta na osnovu funkcionalnosti, datih u klijentovoj ili tenderskoj specifikaciji zahteva.

- Metoda je prezentovana davne 1979 godine.

METRIKA I KVALITET SOFTVERA

Metrički sistem:

- skup kriterijuma, parametara, mernih uređaja, podataka i jedinica za generisanje i prikazivanje rezultata merenja
- podrazumeva procese evaluacije i monitoringa performansi

Rezultati merenja:

- objektivni i sirovi podaci koji se mogu automatski generisati

METRIKA I KVALITET SOFTVERA

Softversko inženjerstvo se bavi razvojem efikasnog, pouzdanog i bezbednog softvera kao proizvoda namenjenog tržištu. ANSI/IEEE729/1983).

Testiranje softvera obuhvata različite vrste testiranja kako bi se osiguralo da softverski proizvod neće imati funkcionalne i nefunkcionalne nedostatke.

Metrike i testovi su najbolji način sagladevanja arhitekture tj. ispunjenosti nefunkcionalnih zahteva.

Metrika za verifikaciju i validaciju softvera



Metrika softvera

Sa rastom kompleksnosti realizovanih funkcija i primena posebno je narastao zahtev za kvalitetom softvera u pogledu pouzdanosti - posebno kod kritičnih softvera, pogodnosti za testiranje i održavanje, ponovne upotrebljivosti, otpornosti na greške i drugih faktora kvaliteta softvera.

Ima mnogo metoda za testiranje softvera, ali treba izabrati pravu metodu.

Za ostvarivanje potrebnog kvaliteta softvera od velikog je značaja određivanje metrike softvera i sprovodenje adekvatnog procesa testiranja.



Upotrebljivost softvera je u velikoj meri određena načinom na koji se meri.

Metrika je „sveobuhvatni termin koji opisuje metod koji se koristi za merenje nečega, rezultirajuće vrednosti dobijene merenjem, kao i izračunati ili kombinovani skup mera“ (Labate, 2016).

Metrika se definiše i kao „sistem ili standard merenja predstavljen u jedinicama koje se mogu koristiti za opisivanje više od jednog atributa“ (Mifsud, 2015)



Metrika softvera

- ✓ Razvoj softvera je proces koji je veoma specifičan i složen.
- ✓ Ideja svakog poslovanja je zadovoljan korisnik.
- ✓ Kada se radi neka od vrsta testiranja softvera, moraju se kreirati neki izveštaji o potencijalnim propustima (Defect reports) i naravno na kraju kao važan factor svega jesu i neke metrike.
- ✓ Ti podaci su veoma korisni koji mogu da budu sastavni deo baze znanja Project i Product Menadžerima za neke nove projekte.



Metrika softvera

Metrika softvera obično se deli na:

- metrike procesa (process metrics)
- metrike proizvoda (product metrics)
- metrike projekta (project metrics)

Potrebno je eliminisati zavisnost metrike od metodologije razvoja softvera.

Postoje:

- Tehničke metrike
- Human metrike



Metrika softvera

Proces metrike su vezane za aktivnosti povezane sa produkcijom softvera.

Koriste za poboljšanje razvoja i održavanja softvera.

Primeri ovih metrika su

- efektivnost uklanjanja defekata za vreme razvoja,
- vreme odziva za fiksiranje problema i slično.

Ove metrike vode dugoročnjem poboljšanju procesa.

Metrike projekta uglavnom se odnose na resurse projekta kao što su ljudi, znanje, hardver.

Primeri ovih metrika su broj developera, budžet, vremenski raspored i produktivnost



Metrika softvera

Metrika softvera obično se deli na:

- Metrika za rokove
 - broj taskova koji su uspešno završeni na vreme
 - broj taskova koji nisu završeni na vreme
 - broj taskova čiji su rokovi promenjeni
 - broj taskova koji su odloženi za kasnije
- Metrika za definisanje korisničkih zahteva
 - broj taskova ili zahteva koji su vezani za samu specifikaciju
 - broj novih zahteva koji su naknadno pristigli
 - RFC dijagram (Requests For Change)



Metrika softvera

Metrika softvera obično se deli na:

- Metrika za testiranje
 - praćenje najčešće procenta SLOC (Source Lines of Code) koji je pokriven testovima;
 - Posebno se prati taj procenat jer povećanjem tog procenta poboljšava se kvalitet i smanjuje broj grešaka koje će otkriti korisnici
- Metrika za ukupni rizik projekta
 - Metrika za kvalitet (za greške u softveru)
 - gustina grešaka
 - broj otkrivenih i otklonjenih grešaka



- ✓ Metrika za testiranje
 - praćenje procenta produktivnosti koji je pokriven testovima; povećanjem tog procenta poboljšava se kvalitet i smanjuje broj grešaka koje će otkriti korisnici
- ✓ Metrika za kvalitet (za greške u softveru)
 - gustina grešaka (broj grešaka produktivnosti) je dobar pokazatelj kvaliteta softvera
 - broj otkrivenih i otklonjenih grešaka (fault arrival and closing rates); proizvod je spreman za isporuku kada ove mere padnu na ≈ 0
- ✓ Metrika za ukupni rizik projekta
 - stepen spremnosti proizvoda za instaliranje i rad (zahteva veće angažovanje IT i product menadžera)



“Upotrebljivost softvera je u velikoj meri određena načinom na koji se meri.” - Myers

Metrika je :

„Sveobuhvatni termin koji opisuje metod koji se koristi za merenje nečega, rezultirajuće vrednosti dobijene merenjem, kao i izračunati ili kombinovani skup mera” (Labate, 2016).

Metrika se definiše i kao :

„Sistem ili standard merenja predstavljen u jedinicama koje se mogu koristiti za opisivanje više od jednog atributa” (Mifsud, 2015)



Metrika softvera

Važnost metrike u testiranju softvera može se navesti kroz nekoliko koraka:

- One pomažu u donošenju odluka za sledeću fazu aktivnosti
- One pomažu u razumevanju potrebe za poboljšanjem softvera
- Olakšava postupak odlučivanja ili promene tehnologije.

To je skup aktivnosti kojima se na osnovu određenih metrika i indikatora performansi utvrđuje kojim tempom se testira zadati sistem.

METRIKA SOFTVERA je:

- kontinualni proces
- podrazumeva podešavanje mera i dobijanje informacija i preduzimanje nekih mera iz iskustva ako je potrebno

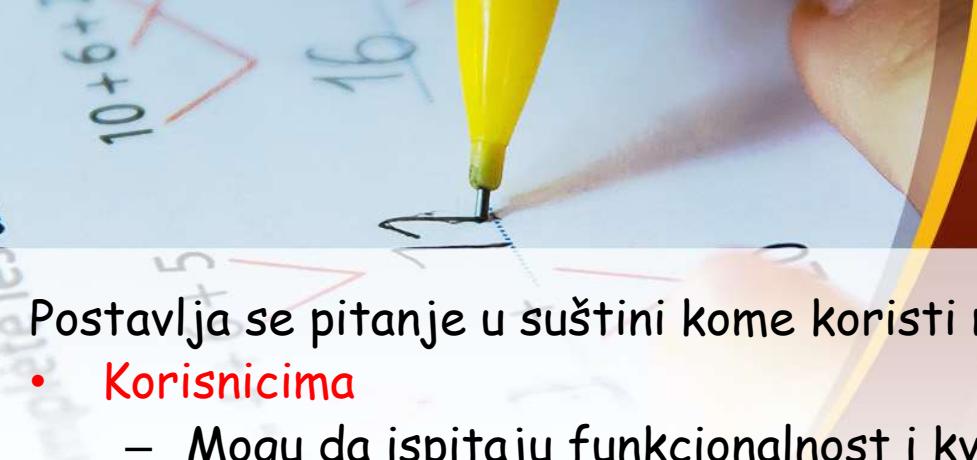


Nakon definisanja šta su softverske metrike potrebno je navesti i kategorije podataka koje se mere - to su takozvani merni podaci.

Merni podaci omogućuju procenu uspeha neke aktivnosti, a uspeh se svakako može definisati kao postizanje određenog cilja.

- Cilj metrika koje se definisu kroz testiranje softvera su:
 - ✓ Postizanje određenog kvalitete softvera,
 - ✓ Zadovoljstvo korisnika,
 - ✓ Isporuka pouzdanog softvera,
 - ✓ Mali troškovi razvoja.
 - ✓ Mali troškovi održavanja softvera

Jedan od mernih podataka je pokrivenost testom. Pokrivenost testom može se razjasniti koji su to delovi softvera testirani.



Metrika softvera

Postavlja se pitanje u suštini kome koristi metrika softvera ?

- Korisnicima
 - Mogu da ispitaju funkcionalnost i kvalitet samog softvera
- Product i Project menadžerima
 - Daje potrebne, tačne informaciju o tome kako softver napreduje
 - Daje potrebne, tačne informaciju o tome kako je softver završen
 - Daje potrebne, tačne informaciju o tome da li je softver uspešan ili nije
 - Daje potrebne, tačne informaciju o održavanju softvera (postoji tim)
- Softverski tim koji razvija softver
 - Daje potrebne, tačne informaciju o aktivnosti koje vode ka uspešnom završetku softvera
- Softverski tim koji održava softver



METRIKA I KVALITET SOFTVERA

Softversko inženjerstvo zahteva upotrebu kako analitičkih, tako i deskriptivnih alatki koje su razvijene u okviru računarskih nauka, zajedno sa rigoroznim pristupom koji donose inženjerske discipline u cilju postizanja odgovarajuće pouzdanosti i bezbednosti, a sve to kroz timski rad softverskih inženjera koji funkcionišu u jednom kohezionom okruženju.

Ako se pravilno upotrebi, test metrika može značajno da doprinese inicijativi unapređenja postojećeg procesa razvoja softvera



Metrika softvera

Slaganje sa eksplisitnim izraženim funkcionalnim zahtevima i zahtevima za performanse, eksplisitno dokumentovanim razvojnim standardima, i implicitnim karakteristikama koje se očekuju od svakog stručno razvijenog softvera" (Pressman, davne 2001 godine)

Bilo kakve aktivnosti merenja moraju imati jasan cilj.

Prilikom testiranja softvera, kreiraju se izveštaji o propustima (Defect reports) i druge metrike.

Metrike kvaliteta softverskog procesa zasnovane na metrikama kompleksnosti programa



Metrika softvera

Kako se prikupljaju neki podaci koji su neophodni za metriku softvera:

- Analiza prethodno završenih projekata
- Analizom iz sličnih projekata koji je već neko radio
- Pravljenje baze podataka za buduće projekte (broj SLOC (Source Lines of Code) iz prethodnih projekata, produktivnost programera, rokovi, greške,...)



Kako izabrati metriku za konkretan projekat?

- ✓ Povezati metriku sa poslovnim ciljevima
- ✓ Odabratи metriku koju razumeju i menadžeri, i programeri
- ✓ Odabratи merenja koja se mogu sprovesti
- ✓ Odabratи metriku koja je stvarno bitna za aktivnosti datog projekta
- ✓ Odabratи konzistentnu metriku za više projekata odabratи metriku koja može da ukaže na načine za povećanje produktivnosti i/ili kvaliteta
- ✓ Odabratи metriku za koju se mogu definisati odgovarajuće akcije
- ✓ Najbolje metrike su one koje prirodno proističu iz radnog procesa



METRIKA I KVALITET SOFTVERA

Pouzdanost je jedan od ključnih faktora kvaliteta i zato pouzdanost isporučenog koda zavisi od kvaliteta svih procesa

Podaci o pouzdanosti softvera pomažu u povećanju produktivnosti, redukciji isporučenih defekata, redukciji frekvecija prekida i redukciji padova sistema.

Da bi se dobili podaci u tim fazama treba upotrebiti softverske metrike za pomoć u vrednovanju pouzdanosti.

Na taj način se povećava kvalitet a sa samim tim i pouzdanost.

METRIKA I KVALITET SOFTVERA

Kod testnih metrika postoje dva pristupa za vrednovanje pouzdanosti:

- Prvi se odnosi na vrednovanje test plana da se obezbedi da sistem ima funkcionalnost predviđenu specifikacijom zahteva.
- Drugi pristup vezan za pouzdanost se sastoji u vrednovanju broja grešaka u kodu i i otklanjanju istih.

Podaci o greškama se dobijaju za vreme testiranja i stvarne upotrebe softverskog proizvoda

Ovi podaci se koriste za izgradnju modela pouzdanosti na osnovu koga se može izvršiti predikcija vremena sledeće greške na osnovu istorijata prethodnih grešaka.

STATISTIKA! TESTIRANJA SOFTVERA

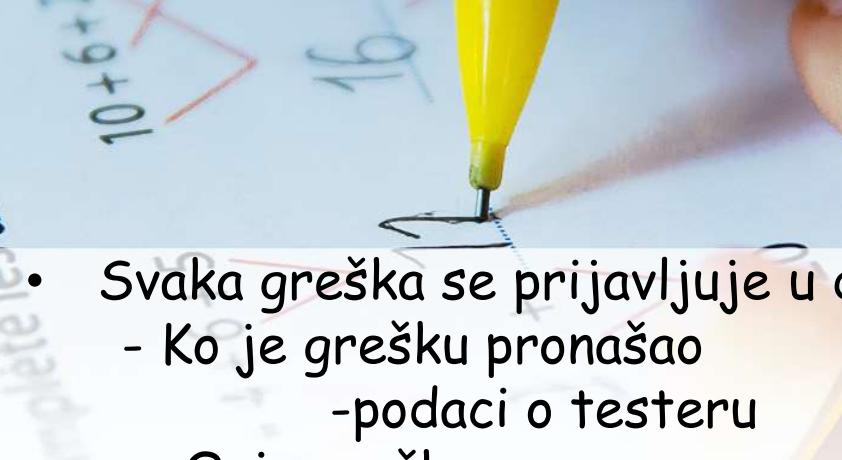


Metrika softvera

Standard ISO/IES 9126 se bavi aspektima koji udvrđuju kvalitet softverske aplikacije:

- Model kvaliteta,
- Eksterna metrika
- Interna metrika
- Pokazatelj kvaliteta u upotrebi.

Pokrivenost koda je metrika koja se koristi za izražavanje koje delove softvera testovi ne koriste, koliki deo izvornog koda je testiran.



Prijavljivanje grešaka

- Svaka greška se prijavljuje u dogovorenom formatu:
 - Ko je grešku pronašao
 - podaci o testeru
 - Opis greške
 - sadrži i opis kako greška može biti reprodukovana (to je posebno izazovno jer ponekad ne možemo tako lako da reproduujemo grešku)
 - Test okruženje u kojem se greška desila
 - Prioritet
 - Tip greške
 - status
 - Komentar

Po velikom broju literature postoji tvrdnja da je :

- Korisnike je teže obučiti da prijavljaju potencijalne greške po unapred dogovorenoj proceduri

	Attribute	Meaning
Identification	Id / Number	Unique identifier/number for each report
	Test object	Identifier or name of the test object
	Version	Identification of the exact version of the test object
	Platform	Identification of the HW/SW platform or the test environment where the problem occurs
	Reporting person	Identification of the reporting tester (possibly with test level)
	Responsible developer	Name of the developer or the team responsible for the test object
	Reporting date	Date and possibly time when the problem was observed
Classification	Status	The current state (and complete history) of processing for the report (section 6.6.4)
	Severity	Classification of the severity of the problem (section 6.6.3)
	Priority	Classification of the priority of correction (section 6.6.3)
	Requirement	Pointer to the (customer-) requirements which are not fulfilled due to the problem
	Problem source	The project phase, where the defect was introduced (analysis, design, programming); useful for planning process improvement measures
Problem description	Test case	Description of the test case (name, number) or the steps necessary to reproduce the problem
	Problem description	Description of the problem or failure that occurred; expected vs. actual observed results or behavior
	Comments	List of comments on the report from developers and other staff involved
	Defect correction	Description of the changes made to correct the defect
	References	Reference to other related reports



Softverski timovi se organiziraju u skladu sa nekom od metodologija, a metodologije slede jedan ili više modela razvoja softvera.

Potrebno je pratiti različite parametre i obezbediti kvalitet softvera, kreirati planove vezane za kvalitet i ostvariti potrebnu kontrolu.

Ono što je karakteristično za današnji razvoj softvera je činjenica da se vreme od ideje do razvoja softvera, sve više skraćuje.

MANJAK VREMENA ZA TESTIRANJE SOFTVERA



Statički atributi koda mogu omogućiti da se nauči puno o strukturi sistema, kao i o područjima u programu koji mogu biti potencijalni uzročnici bagova (grešaka).

Što je kod kompleksniji, program je teže testirati, održavati a to prouzrokuje i mnoge greške.

POJMOVI:

- ✓ **Veličina** (size): Tipična merenja linija koda (LOC) , Metrike linije koda (Lines of Code - LOC)
- ✓ **Cyclomatic Complexity**: Mere kontrole toka unutar modula.
- ✓ **Halstead kompleksnost**: Mere broja operatara i operanada u kodu.
- ✓ **Informacijski tok** (Information Flow): Merenja toka podataka u i iz modula.
- ✓ **Kompleksnost sistema** (System Complexity): Merenja kompleksnosti celokupnog sistema u terminima održavanja i/ili dizajna.
- ✓ **Objektno-orientirane strukturalne metrike**: Merenje različitih struktura objektno orijentiranih programa (u odnosu na funkcionalno dizajnjane programe).



METRIKA I KVALITET SOFTVERA

Veličina je jedan od najčešćih atributa softvera.

Neka od ključnih pitanja o kojima korinici žele da znaju uključuje veličinu projekta.

- Koliko je velik softver ?
- Koliko je vremena trebalo da se uradi dati projekat?
- Koliko je velik neki softver u odnosu na druge projekte?
- Koliko napora je uloženo za implementaciju softvera?
- Kako se kvalitet (softvera koji sada radi) poredi sa drugim projektima?
- Kako se produktivnost koja je uložena u dati projekat poredi sa drugim projektima?

Veličina (size) je osnova svih ovih metrika.

- STANDARDIZACIJA



Metrika softvera

- Alternativna metrika - metoda funkcionalnih tačaka meri veličinu projekta na osnovu funkcionalnosti, datih u klijentovoj ili tenderskoj specifikaciji zahteva.

Metrike kvaliteta softverskog procesa koje se računaju zasnovano na metrikama veličine su:

- Metrike gustine grešaka
- Metrike ozbiljnosti grešaka
- Metrike efektivnosti uklanjanja grešaka

METRIKA I KVALITET SOFTVERA

Softverske metrike mere su uspešnosti softverskog procesa.

Sama metrika se može podeliti na metriku kontrole i metriku predviđanja.

Potrebno je eliminisati zavisnost metrike od metodologije razvoja softvera.

Ako prijavljeni problem uspemo da reprodukujemo, onda ćemo ga sigurno i rešiti. (Reproduction= solution)



Metrika softvera

- ✓ Probabilistički model
- ✓ Modeli rasta pouzdanosti
- ✓ Modeli stope neuspeha
- ✓ Mek-Kejbova ciklomatična složenost

$$C(G) = E - V + 2P,$$

gde je:

E = broj grana grafa

V = broj čvorova grafa

P = broj povezanih komponenti grafa



Izgradnja visoko pouzdanog softvera zavisi od učešća atributa kvaliteta u svakoj fazi životnog ciklusa razvoja sa naglaskom na prevenciju grešaka, specijalno u ranim fazama.

Potrebe korisnika: određeni nivo kvaliteta, a ne samo funkcionalnost

Da bi se ovi atributi merili sa ciljem poboljšanja kvaliteta i pouzdanosti potrebno je definisati softverske metrike za svaku razvojnu fazu (dokumentaciju, kod, planove testiranja i samo testiranje na samo kraju, kao i održavanje.

METRIKA I KVALITET SOFTVERA

U razvojnom ciklusu softvera sve je značajniji zadatak Procesa Testiranja Softvera ili Verifikacije i Validacije koji treba da obezbedi zahtevani nivo poverenja u ispravnost (korektnost) softvera kao i obezbeđenja ostalih zahtevanih karakteristika softvera.

Optimizacija koda i brzina

METRIKA I KVALITET SOFTVERA

Određivanje mera koje reflektuju bitne karakteristike svakog softverskog projekta

- obim projekta i kvalitet izlaznog softvera
- isporučivanje proizvoda i vremenski rokovi koji su potrebni za izradu softvera
- napor potreban da se projekat završi
- Testiranje softvera
- Pravljenje kvalitetne dokumentacije

METRIKA I KVALITET SOFTVERA

- **Održavanje softvera je**

- Proces modifikacije softverskog sistema ili komponente **nakon isporuke** radi ispravljanja grešaka, poboljšanja performansi ili drugih atrubuta ili prilagodjenja novoj sredini [IEEE Std 610.12-1999]
- Softverski proizvod podleže modifikaciji koda i odgovarajuće dokumentacije usled nekog problema ili potrebe za poboljšanjem.
- Cilj toga je da se modifikuje postojeći softverski proizvod a da se u isto vreme sačuva njegov integritet. [ISO Std 12217]

METRIKA I KVALITET SOFTVERA - Evolucija softvera

Evolucija softvera je

- Niz aktivnosti, tehničkih koje obezbeđuju da softver nastavlja da ispunjava organizacione i poslovne ciljeve pritom iskorišćavajući poverena sredstva na najbolji način (Institut za istraživanja na polju evolucije softvera)[Research Inst. On Sw. Evolution]
- Svaka programerska delatnost koja je namenjena stvaranju nove verzije softvera od neke ranije verzije (Lehman i Ramil 2000 godine)
- Primena mera (aktivnosti) i procesa održavanja softvera koji stvaraju novu radnu verziju softvera sa promjenjenom funkcionalnošću (prema iskustvima korisnika) ili sa svojstvima prethodne radne verzije, zajedno sa odgovarajućim merama i procesima koje osiguravaju kvalitet, i rukovodenjem tim merama i procesima (Ned Chappin davne 1999 godine)

METRIKA I KVALITET SOFTVERA

Koji su razlozi zašto neko menja neki softver koji mu i dalje radi?

Neki od razloga su!

- ✓ Poboljšanja (Enhancements)
- ✓ Popravak (Repairs)
- ✓ Performanse (Performance improvements)
- ✓ Menjanje, dodavanje, sa ciljem da softver brže bolje i sigurnije radi
- ✓ Zaštita sistema
- ✓ Konfiguracija hardvera (Configuration changes)
- ✓ (Environment upgrades)

METRIKA I KVALITET SOFTVERA - Tipovi održavanja softvera

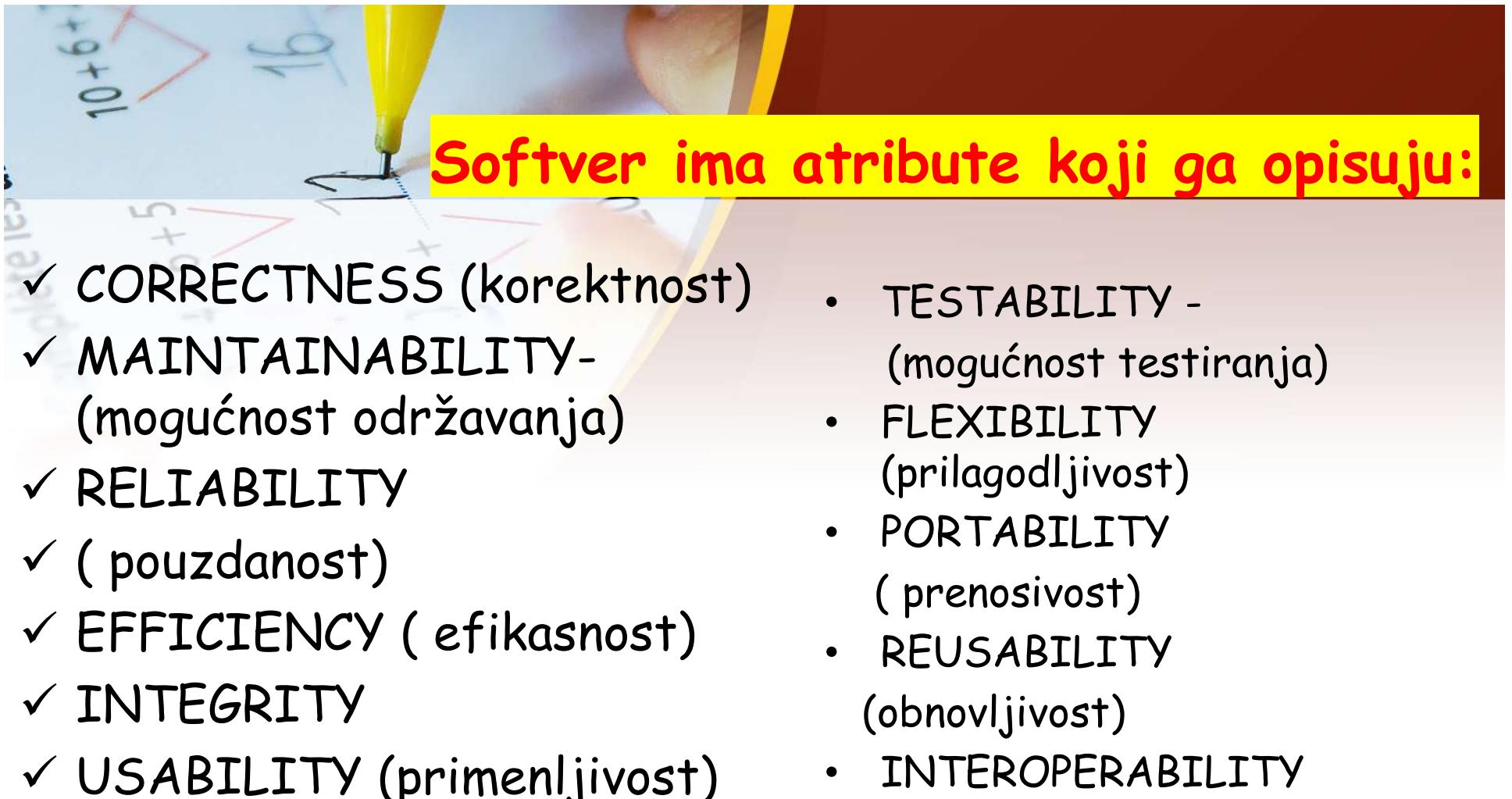
Prema standardu ISO/IEC 14674 za softverski inženjerинг - održavanje softvera

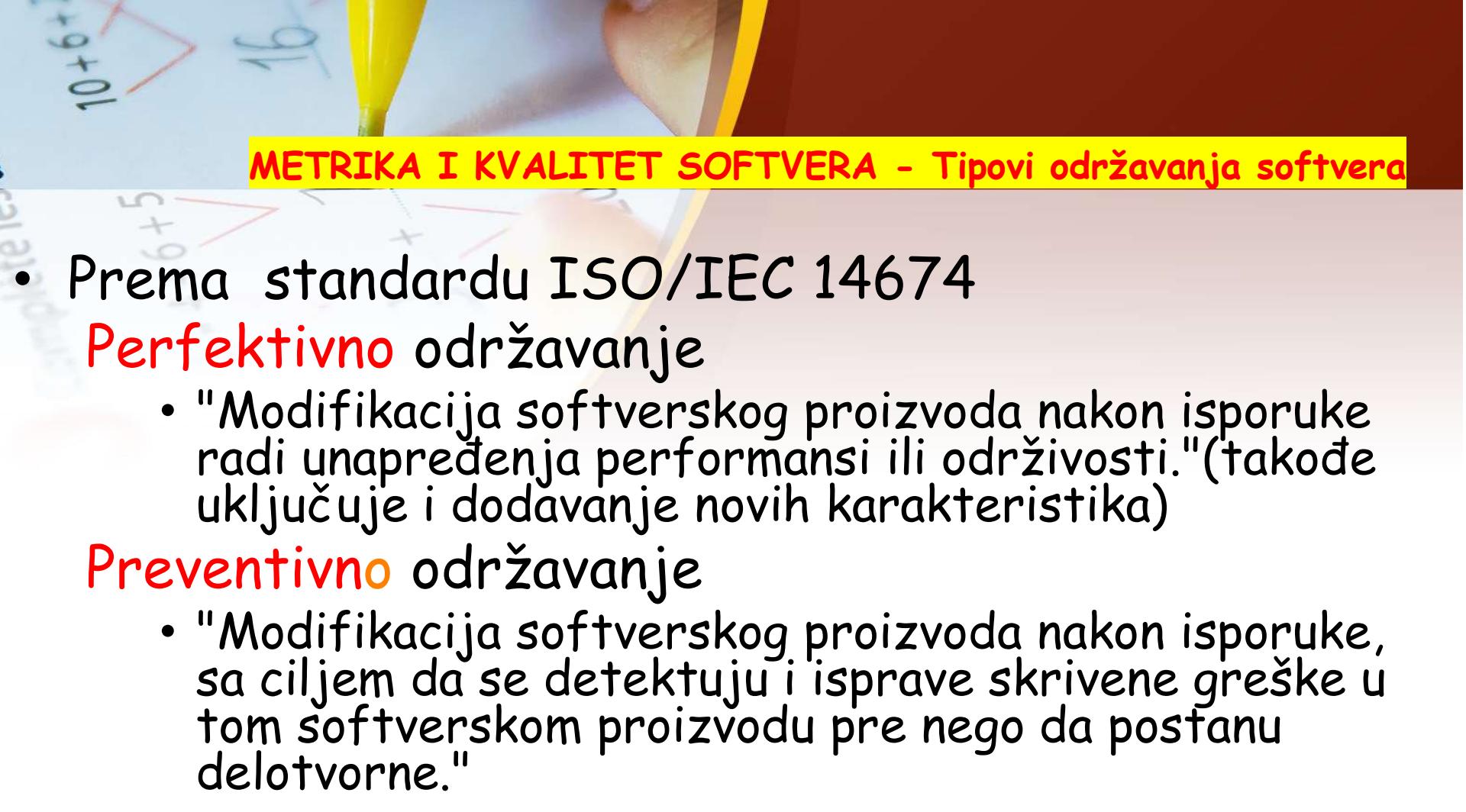
Adaptivno održavanje

- "Modifikacija softverskog proizvoda koja se izvodi nakon isporuke, a sa ciljem da se tom softverskom proizvodu sačuva upotrebna vrednost u promenjenoj sredini ili sredini koja se upravo menja."

Korektivno održavanje

- "Reaktivna modifikacija softverskog proizvoda koja se vrši nakon isporuke, radi popravke otkrivenih grešaka."

- 
- Softver ima atribute koji ga opisuju:**
- ✓ CORRECTNESS (korektnost)
 - ✓ MAINTAINABILITY-
(mogućnost održavanja)
 - ✓ RELIABILITY
 - ✓ (pouzdanost)
 - ✓ EFFICIENCY (efikasnost)
 - ✓ INTEGRITY
 - ✓ USABILITY (primenljivost)
 - TESTABILITY -
(mogućnost testiranja)
 - FLEXIBILITY
(prilagodljivost)
 - PORTABILITY
(prenosivost)
 - REUSABILITY
(obnovljivost)
 - INTEROPERABILITY



METRIKA I KVALITET SOFTVERA - Tipovi održavanja softvera

- Prema standardu ISO/IEC 14674

Perfektivno održavanje

- "Modifikacija softverskog proizvoda nakon isporuke radi unapređenja performansi ili održivosti."(takođe uključuje i dodavanje novih karakteristika)

Preventivno održavanje

- "Modifikacija softverskog proizvoda nakon isporuke, sa ciljem da se detektuju i isprave skrivenе greške u tom softverskom proizvodu pre nego da postanu delotvorne."

METRIKA I KVALITET SOTVERA Tipovi održavanja softvera





METRIKA I KVALITET SOFTVERA Tipovi održavanja softvera

Klasifikacija zasnovna na objektivnom dokazu
[Chapin i drugi 2001]

- Klasificuje evoluciju softvera i mere i procese održavanja unutar softvera (uključujući dokumentaciju)
 - koda
 - funkcionalnosti za korisnika
- poredeći softver pre i posle evolucije

METRIKA I KVALITET SOFTVERA Tipovi održavanja softvera

PROBLEMI ODRŽAVANJA - Lienzi Swanson1977 -NosekPalvia1990

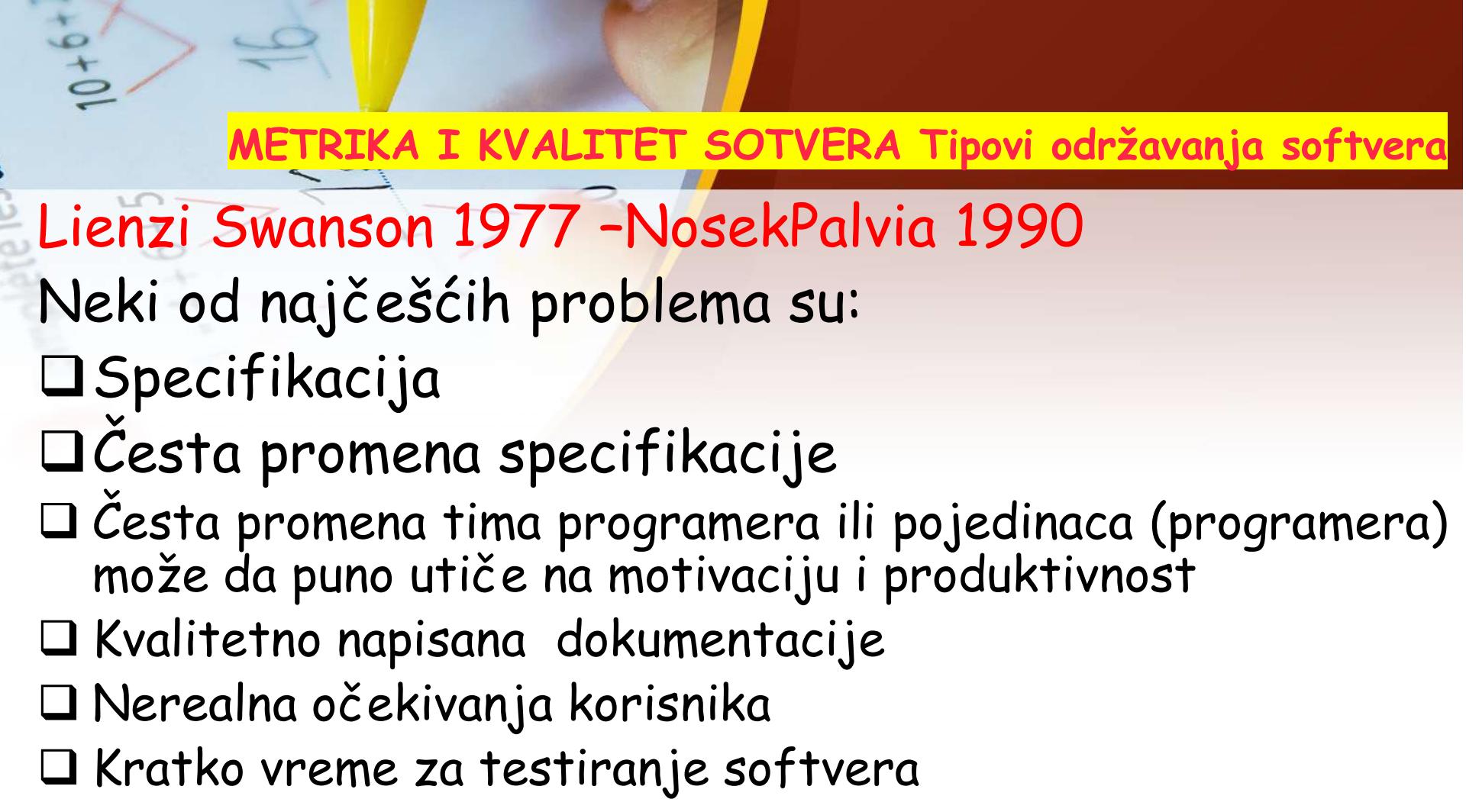
- Velika količina zahteva za održavanje različitih softvera
- Neefikasno korištenje programerskih resursa u timovima za održavanje
- Povećan angažman nakon implementacije
- Određivanje prioriteta uvođenja novih funkcionalnosti
- Povećan broj grešaka u novim isporukama softvera,
- Česta promena programera u timovima za održavanje,
- Dokumentacija ako postoji je često nepotpuna i nedosledne,
- Ne-metodološki pristup razvoju softvera koji otežava održavanje
- Manja produktivnost programerskog tima zbog specifičnosti poslova u održavanju



METRIKA I KVALITET SOTVERA Tipovi održavanja softvera

PROBLEMI ODRŽAVANJA

- ✓ Različiti kvalitete programera u timovima za održavanje
- ✓ Sporo uključivanje novih programera u proces održavanja,
- ✓ Manje iskusni programeri koji se često zapošljavaju u timovima za održavanje
- ✓ Nerealna očekivanja korisnika vezano uz rokove za isporuku novih zahteva
- ✓ Osiguravanje i povećanje kvaliteta proizvoda
- ✓ Osiguravanje i povećanje kvaliteta usluga
- ✓ Osiguravanja kontinuiteta održavanja
- ✓ Povećanje efikasnosti procesa održavanja**



METRIKA I KVALITET SOTVERA Tipovi održavanja softvera

Lienzi Swanson 1977 - NosekPalvia 1990

Neki od najčešćih problema su:

- Specifikacija
- Česta promena specifikacije
- Česta promena tima programera ili pojedinaca (programera) može da puno utiče na motivaciju i produktivnost
- Kvalitetno napisana dokumentacija
- Nerealna očekivanja korisnika
- Kratko vreme za testiranje softvera



REDIZAJN SOFTVERA

Mada je održavanje jedan neprekidan proces, ozbiljno se mora posmatrati situacija kod redizajniranja softverskog sistema.

Glavno pitanje je da li se sistem može održavati ili ne može.

Neke od problema su:

Nove verzije operativnih sistema, -
problem sa drajverima.

Drugim rečima treba da postoji sofverski tim koji mogu da analiziraju opšte stanje softvera i kada bude neophodno odlučuju da je softver bolje redizajnirati nego ga nastaviti korektivno održavati.

SUGESTIJA: ANALIZIRATI SLEDEĆE:

Troškovi budžeta i održavanja softvera kada postane sklon bagovima, neefikasan i skup, moraju biti dobro iznalazirani sa onima koji se zalažu za redizajniranje sistema čak i kada redizajniranje nije najbolje rešenje.



Ne postoji (bilo koje) pravilo koje određuje kada je bolje redizajnirati sistem nego održavati postojeći.

Postoje neki faktori koji su veoma bitni, a koje treba uzeti u obzir:

- Česti sistemski otkazi
- Kodovi stari više od pet godina (Predviđeni životni ciklus velikog dela aplikacija je 3- 5 godina. Pogoršanje karakteristika softvera sa godinama je rezultat brojnih u zargonu reči poput "opravki" i "krpljenja".
- Previše kompleksne programske strukture
- Programirano i rađen softver za predhodnu generaciju harvera

METRIKA I KVALITET SOFTVERA Tipovi održavanja softvera

- ✓ Obuka
- ✓ Konsultacija
- ✓ Vrednovanje
- ✓ Redizajn
- ✓ Ažuriranje

METRIKA I KVALITET SOFTVERA Tipovi održavanja softvera

- ✓ Testiranje
- ✓ Dokumentacija
- ✓ Performanse
- ✓ Adaptacija
- ✓ Ograničenja u hardveru i operativnom sistemu
- ✓ Izmena na postojećem kodu i ažuriranje
- ✓ Nove verzije

METRIKA I KVALITET SOFTVERA

- Promene u zahtevima korisnika
 - Modifikacije i proširjanja na osnovu zahteva od strane korisnika
- Otklanjanje grešaka
 - Redovne popravke
 - Vanredne popravke - koje više koštaju usled velikog pritiska (vreme)
- Promene u formatima podataka
 - Novi standardi: UML, XML, wsdl, json,...
- Promene hardvera
- Unapređenja efikasnosti



METRIKA I KVALITET SOFTVERA - Problemi održavanja

Održavanje je težak i složen proces.

Sistem je već u eksploataciji

- ✓ svako modifikovanje mora da se uskladi sa potrebama korisnika
- ✓ neki sistemi dopuštaju, a neki ne da izvesno vreme ne budu aktivni
- ✓ mora se pronaći neko alternativno rešenje koje ne bi ugrozilo korisnika (rezervni sistem)



Održavanje je težak i složen proces.

Mogu se javiti personalni i organizacioni problemi

- ✓ nedovoljno razumevanje funkcionisanja sistema i nedostatak potrebnih veština od strane korisnika (rešenje kvalitetna obuka i dokumentacija)
- ✓ nesklad između prioriteta Menadžmenta i korisnika
- ✓ moral tima za održavanje (primer iz prakse-jedno od rešenje je rotacija programera iz razvojnog tima u tim za održavanje)

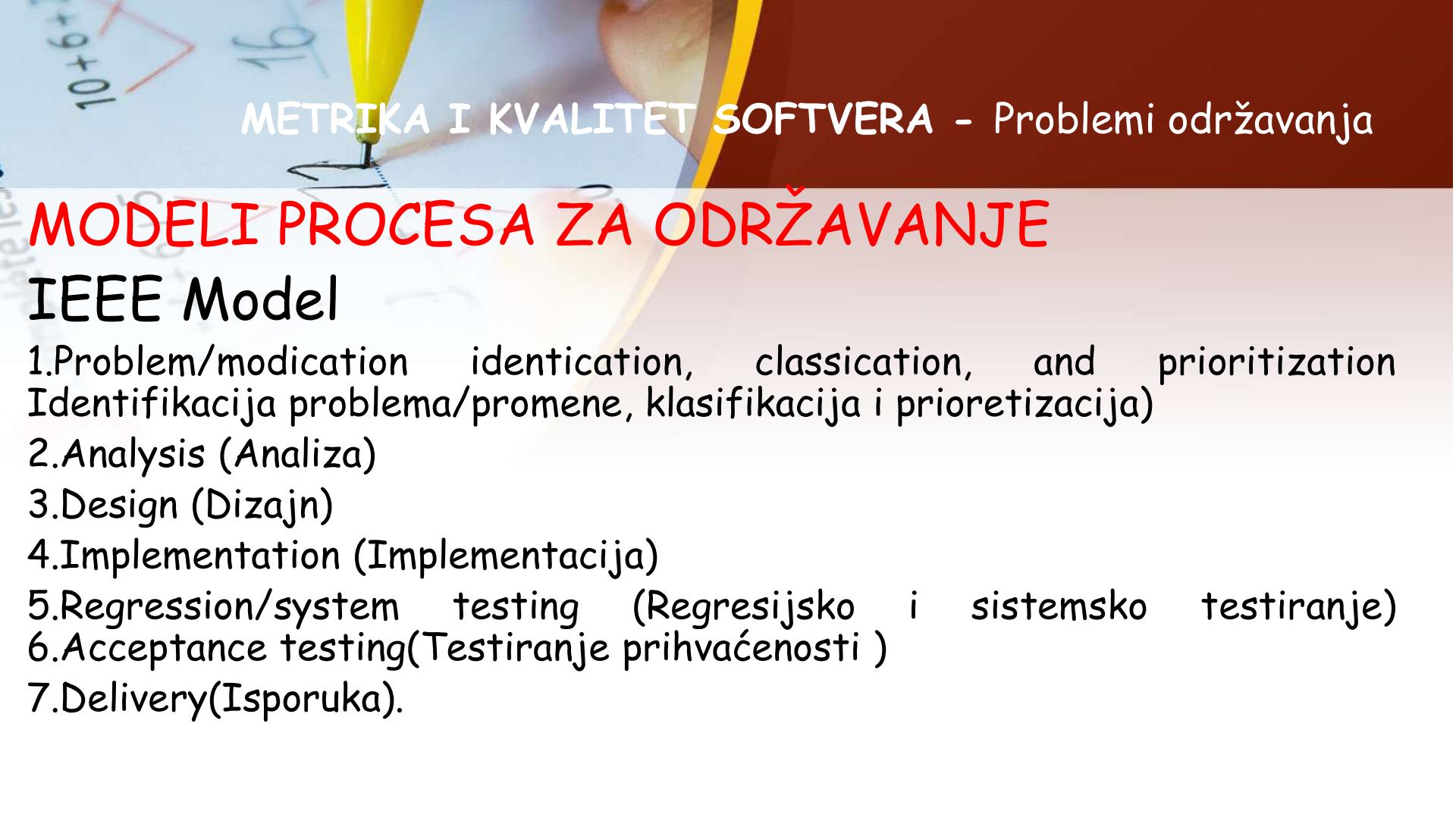


Održavanje je težak i složen proces.

Mogu se javiti tehnički problemi

- ✓ zbog načina projektovanja i implementacije sistema
- ✓ zbog izabranih hardverskih i softverskih platformi korišćenih u realizaciji (nepouzdane i podložne otkazima)

Projektni budžet održavanja celog sistema-projekta



METRIKA I KVALITET SOFTVERA - Problemi održavanja

MODELI PROCESA ZA ODRŽAVANJE

IEEE Model

1. Problem/modication identification, classification, and prioritization
(Identifikacija problema/promene, klasifikacija i prioretizacija)
2. Analysis (Analiza)
3. Design (Dizajn)
4. Implementation (Implementacija)
5. Regression/system testing (Regresijsko i sistemsко testiranje)
6. Acceptance testing (Testiranje prihvaćenosti)
7. Delivery (Isporuka).



METRIKA I KVALITET SOFTVERA - Problemi održavanja

Održavanje je težak i složen proces.

Uputstvo za instalaciju

Uputstvo za instalaciju je tehnički dokument u kome je detaljno, korak po korak, opisan postupak instalacije softvera.

- navode se uslovi neophodni za regularan rad softvera
- hardverska i softverska platforma
- minimalna i maksimalna konfiguracija
- Koji operativni sistem se koristi, (verzije)



METRIKA I KVALITET SOFTVERA - Problemi održavanja

Održavanje je težak i složen proces.

Vrste dokumentacije

- ✓ Uputstvo za korisnika
- ✓ Uputstvo za operatera
- ✓ Uputstvo za instalaciju
- ✓ Uputstvo za programere



METRIKA I KVALITET SOFTVERA

HALSTEAD metrike kompleksnosti

Halstead metrika kompleksnosti je razvijena od strane Maurice Halsteada sa namenom određivanja kvantitativnih mera kompleksnosti na osnovu operatora i operanada u modulima izvornog (source) koda.

- Pošto se primjenjuje na kod, često se koristi i kao metrika održavanja.
- Halstead merenja su uvedena još 1977 godine i predstavljaju najstarija merenja programske kompleksnosti.



HALSTEAD metrike kompleksnosti

- ✓ Halstead volumen (V) opisuje veličinu implementacije algoritma.
- ✓ Računanje V je bazirano na broju operacija koji se izvršavaju i broju operanada koji se pojavljuju u algoritmu.
- ✓ Volumen funkcije trebao bi biti najmanje 20 i najviše 1000.
- ✓ Volumen jedno linjske funkcije bez parametara, koja ima telo funkcije je oko 20.
- ✓ Volumen fajla bi trebao biti između 100 i 8000.

METRIKA I KVALITET SOFTVERA

Problemi sa koji se javljaju u sistemima:

- Često rade na zastarelom hardveru
- Teško ih je održavati, poboljšati i proširiti
- Opšte odsustvo razumevanja sistema:
 - Nema nikoga da objasni kako sistem radi, nedostatak programera, sistema analitičara,
 - Nedostatak dokumentacija i uputstva za upotrebu
 - Sa novim verzijama softvera, problem sa drajverima

METRIKA I KVALITET SOFTVERA

Troškovi kontrole:

Obuhvataju troškove koji se troše za sprečavanje i otkrivanje grešaka softvera u cilju da ih svede na prihvatljiv nivo.

Neuspeli troškovi kontrole:

Obuhvataju troškove koji nisu uspeli da otkriju ili spreče greške koje su se dogodile u softveru.

Ovaj model troškove dalje razlaže na podklase.

Troškovi kontrole se mogu podeliti na interne i eksterne troškove.



METRIKA I KVALITET SOFTVERA

Interni troškovi nedostataka uključuju troškove greške koje su otkrivene u dizajnu, testiranju softvera i testiranju prihvatljivosti, a završen pre nego što je instaliran na opremi kod klijenta, ili pre nego što je klijent preuzeo aplikaciju sa Interneta. (clouda..)

Eksterni troškovi nedostataka uključuju sve troškove vezane za ispravljanje propusta otkrivenih od strane kupaca ili tima za održavanje nakon što je softverski sistem instalirao dati softver



Metriku za konkretan projekat, šta je najbolje?

U literaturi se kao očigledan primer za potvrdu nemogućnosti kompletног testiranja softvera navodi jednostavan misaoni eksperiment koji je zamislio Beizer, a koji se sastoji u tome da ako imamo program koji učitava niz od 10 znakova i izvršava neku od potrebnih operacija nad njim. I ovako jednostavna situacija ima jako veliki broj slučajeva ulaznih vrednosti koje treba testirati i samo testirati.



METRIKA I KVALITET SOFTVERA

- Postoji i metrika koje se koriste za merenje programske kompleksnosti i koje su izuzetno vredne.
- Neke od njih su:
 - Branching complexity (Sneed Metric),*
 - Data access complexity (Card Metric),*
 - Data flow complexity (Elshof Metric),*