Testiranje softvera

UVODNA PREZENTACIJA

autori: Dragan Bojić i Dražen Drašković

Informacije i komunikacija

dr Dragan Bojić
Redovni profesor

bojic@etf.bg.ac.rs

Kancelarija: P23

Grupa: P1

Veb strana predmeta:

http://si3ts.etf.bg.ac.rs

Mejling lista:

13s113ts@lists.etf.rs
(aktivna od kraja 2. sedmice)

dr Dražen Drašković
Docent

drazen.draskovic@etf.bg.ac.rs

Kancelarija: 37a

Grupa: V1

Pravila polaganja

Preduslov

Za izlazak na završni ispit neophodno je imati urađene i odbranjene domaće zadatke (najmanje 20 ostvarenih od maksimalnih 40 poena).

U svakom ispitnom roku student može polagati ispit koji pokriva celo gradivo i boduje se sa 60 poena. Ukoliko je student polagao kolokvijume uzima se najbolja varijanta pismenih provera znanja:

1. oba kolokvijuma se računaju: UK = K1 + K2 + ispit * 1/3
2. računa se jedan od kolokvijuma: UK = Kx + ispit * 2/3
3. računa se samo završni ispit: UK = ispit * 1

U sve 3 varijante, uslov za pozitivnu ocenu je da UK koji se osvoji kroz kolokvijume i završni ispit >= 30 i da sa domaćim zadacima bude UK+DZ1+DZ2 >= 51. Granice formiranja ocena su na 51, 61, 71, itd.

Literatura

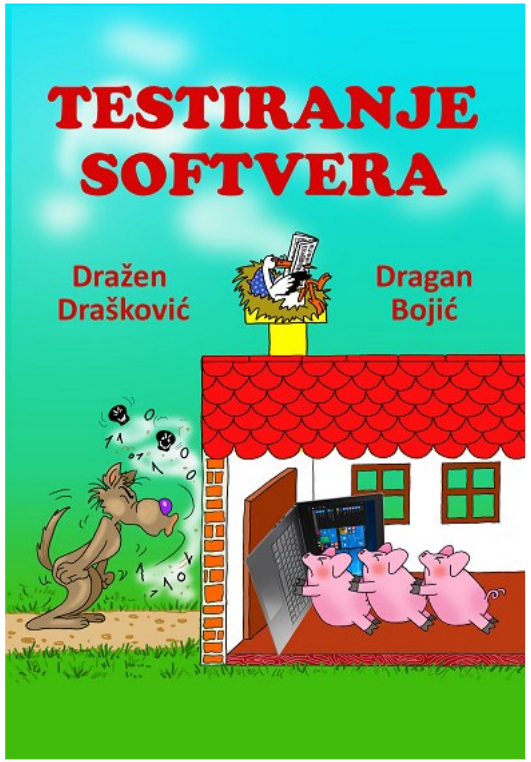
01 Glavni materijal

Materijali sa sajta predmeta (predavanja i vežbe)

02 Udžbenik

D. Drašković, D. Bojić, Testiranje softvera,

izdavač: ETF/Akademska misao, oktobar 2019. godine



03 Dodatna literatura

- Jorgensen, *Software Testing: A Craftman's Approach*, 4. izdanje, 2014, ISBN: 1466560681
- Myers, Sandler, Badgett, *The Art of Software Testing*, 3. izdanje, 2012, ISBN: 1118031962
- ISO/IEC/IEEE 29119-4:2015 Software and systems engineering - Software testing - Part 4: Test techniques
- Mauro Pezze, Michal Young, *Software Testing and Analysis: Process, Principles and Techniques*, 2008, ISBN:9780471455936
- Richard H. Carver, *Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win*, John Wiley & Sons, 2005.

Industrijski softver u odnosu na „studentske“ projekte

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

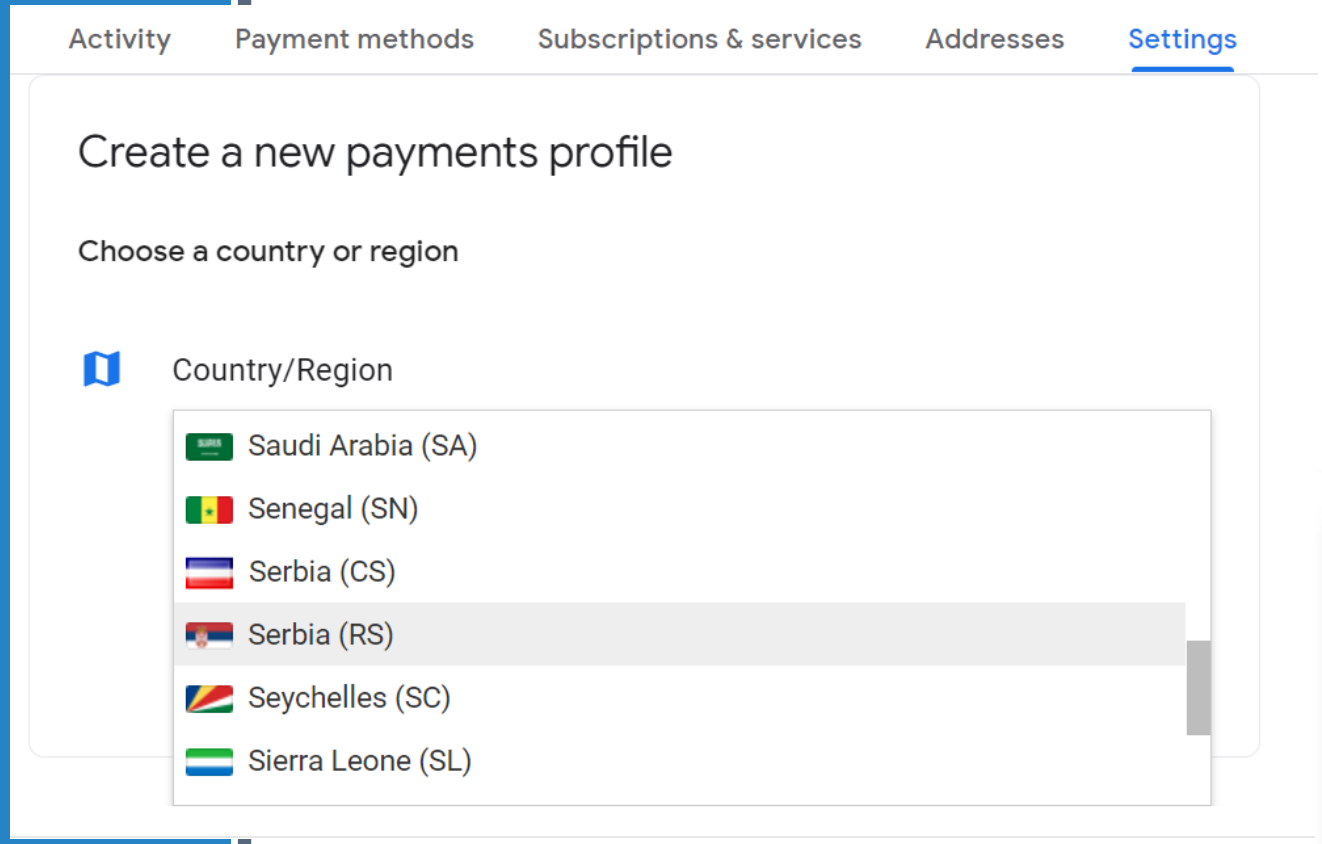
Industrijski softver

- engl. *industrial strength software*

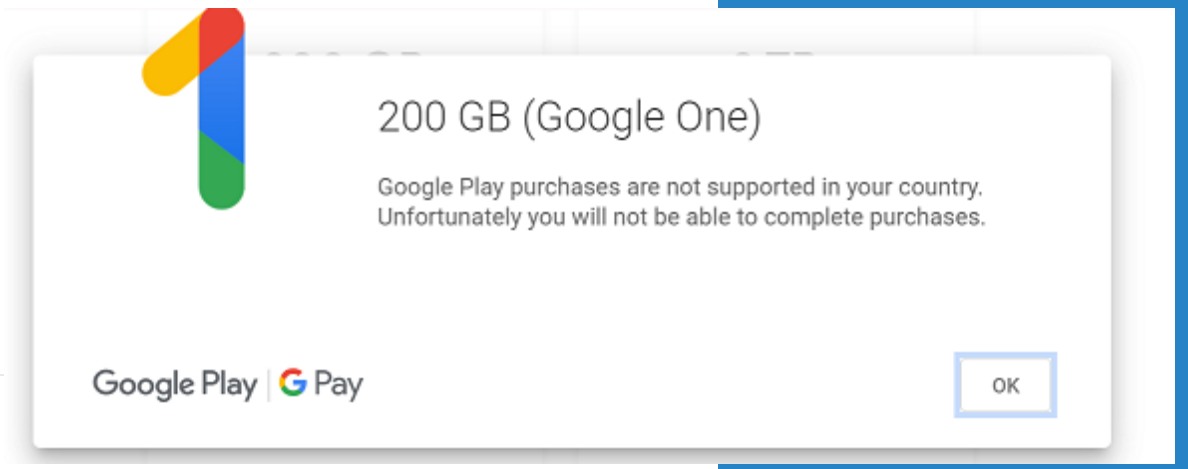
- Napravljen da rešava neki poslovni problem korisnika, tj. važne aktivnosti zavise od korektnog funkcionisanja sistema =>
  loše funkcionisanje izaziva nezadovoljstvo korisnika i finansijske, materijalne gubitke, ili čak ljudske žrtve.

Softverski bagovi su svuda oko nas

- Da li se Vama desilo da nekada ne radi neka aplikacija ili u određenom trenutku „pukne“?



Google Play servis pri prelasku na Google One u jednom scenariju nije radio, kada se odlučite povećati vaš prostor u oblaku!



Pregled skorašnjih interesantnih primera softverskih otkaza

2014

LOS ANĐELES (SAD)

Otkazano stotine letova

Greška u softveru ERAM

2015

Ažuriranje Windows OS.

Otkazivanje semafora košarkaškog

PADERBORN (NEMAČKA)

2017

BEOGRAD (SRBIJA)

Kontrola leta: remont SW

Svi letovi preusmereni van srpskog vazdrušnog prostora.

2018

Softver loše reaguje na rotaciona svetla

11 incidenata sa Tesla automobilima

INDIJANA i FLORIDA (SAD)

2021

SAD

Problem sa Tesla automobilima

Softver daje lažne signala, nakon čega automobil koči bez razloga.

Interesantni primeri softverskih otkaza

- 9. novembra 1979. Računari u sedištu NORAD alarmirali su masovni sovjetski nuklearni napad na SAD. NORAD je preneo informacije Strateškoj vazdušnoj komandi (SAC) i drugim komandnim centrima visokog nivoa.

- Za nekoliko minuta posade američkih interkontinentalnih balističkih raketa (ICBM) bile su pod uzbunom, nuklearni bombarderi pripremljeni za poletanje, a predsednički avion je poleteo (bez predsednika Kartera). Nakon 6 minuta, satelitski podaci nisu potvrdili napad, vodeći zvaničnici su odlučili da nisu neophodne hitne akcije. Kasnije su istrage otkrile da je incident izazvao tehničar koji je greškom ubacio traku za trening koja sadrži scenario za veliki nuklearni napad u operativni računar.

Interesantni primeri softverskih otkaza (1)

- U priručniku za automobil postoji upozorenje: *„Cruise Control* u saobraćaju ne može otkriti sve predmete i ne može kočiti/usporiti za nepomična vozila, posebno u situacijama kada vozite preko 80 km/h, vozilo koje pratite napušta vašu traku, a ispred vas se pojavi nepomično vozilo ili predmet.“
- Autopilot automobila Tesla tokom 2018. godine imao je 11 incidenata sa vozilima sa rotirajućim svetlima (policija, hitna pomoć, vatrogasci), a u 2019. godini na Floridi, u incidentu kada je prošao kroz crveno svetlo, stradao je jedan 22-godišnjak.

Interesantni primeri softverskih otkaza (2)

- 30. aprila 2014. godine: stotine letova u Los Anđelesu su odloženi ili otkazani, jer su se svi računari na aerodromu srušili zbog greške u računarskom sistemu ERAM koji je koštao 2.4 milijarde dolara.
- Kvar sistema je nastao zbog špijunskog aviona U-2 koji je leteo kroz region LA. Sistem je upao u petlju pokušavajući je da popravi grešku, koja je bila posledica nedostatka podataka o visini u planu leta aviona.

- Nakon što je kontrolor vazdušnog saobraćaja uneo procenjenu nadmorsku visinu aviona, sistem je izračunao sve moguće putanje leta da bi se osiguralo da nije na putu sudara sa drugim avionima. Međutim, taj proces je doveo do toga da sistem potroši memoriju i ugasi svaku drugu funkciju obrade podataka o letovima.

Interesantni primeri softverskih otkaza (3)

- 13. marta 2015, Paderborn Baskets, drugoligaški nemački košarkaški tim, prebačen je u niži rang zbog kašnjenja početka utakmice, usled neophodnog 17-minutnog ažuriranja Windows OS na laptopu koji je upravljao semaforom.
- Paderborn je pobedio u igri 69:62, ali je drugi klub Chemnitz protestovao zbog pobede tvrdeći da zbog toga što je utakmica kasnila za 25 minuta, dok nemačka košarkaška pravila dopuštaju samo 15 minuta odlaganja, Paderborn kao domaćina treba kazniti.
- Kao rezultat toga, Paderborn je izgubio bod po kazni i izbačen je iz ProA u ProB diviziju.

1. A German pro basketball team was relegated to a lower division due to a Windows update

Interesantni primeri softverskih otkaza (4)

- Na stotine aviona preusmereno je 10. jula 2017. uveče iz vazdušnog prostora Srbije u okolne zemlje jer Kontrola letenja Srbije i Crne Gore (*SMATSA*) nije radila 50 minuta zbog greške prilikom redovnog servisiranja softvera.

- Greška je nastala usled toga što je spoljni saradnik, unajmljen iz kompanije za avio-saobraćaj *Comsoft,* sa kojom *SMATSA* sarađuje da instalira novu verziju softvera, novi softver ubacio u aktivnu verziju programa, umesto da to uradi u pasivnom režimu, što je dovelo do pada sistema.
  (izvor: dnevni listovi u Srbiji)

Interesantni primeri softverskih otkaza (5)

Tesla povlači 12000 vozila zbog greške u softveru

6. novembar 2021. PC PRESS

Prema najnovijem saznanju Rojtersa, Tesla iz prodaje povlači oko 12 000 vozila, prodatih od 2017. godine, **zbog softverske greške koja može prouzrokovati lažne signale za opasnost od sudara ili neočekivanu aktivaciju sistema za kočenje.**

Industrijski softver

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

Veličina industrijskog softvera

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

Napomena: KLOC – 1000 linija koda, MLOC – 1 milion linija koda

- Veličina poznatih operativnih sistema:
  - Windows XP (2001): ~45 MLOC
  - Windows 10: ~50 MLOC
  - Linux kernel 2.6.32: >12 MLOC
  - Linux kernel 4.2: ~20.2 MLOC
- Američki nacionalni institut za standarde (*NIST*) procenjuje da softverski bagovi izazivaju 60 milijardi $ gubitaka u američkoj ekonomiji godišnje.

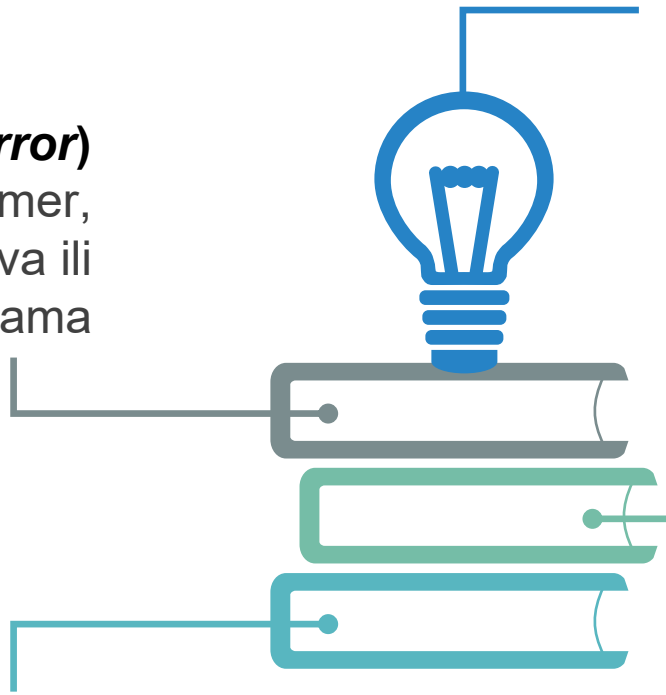
Razvoj softvera u odnosu na druge tehničke discipline

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

Terminologija

IZVOR:
Glossary of Software Engineering Terminology. ANSI/IEEE Std. 729–1983

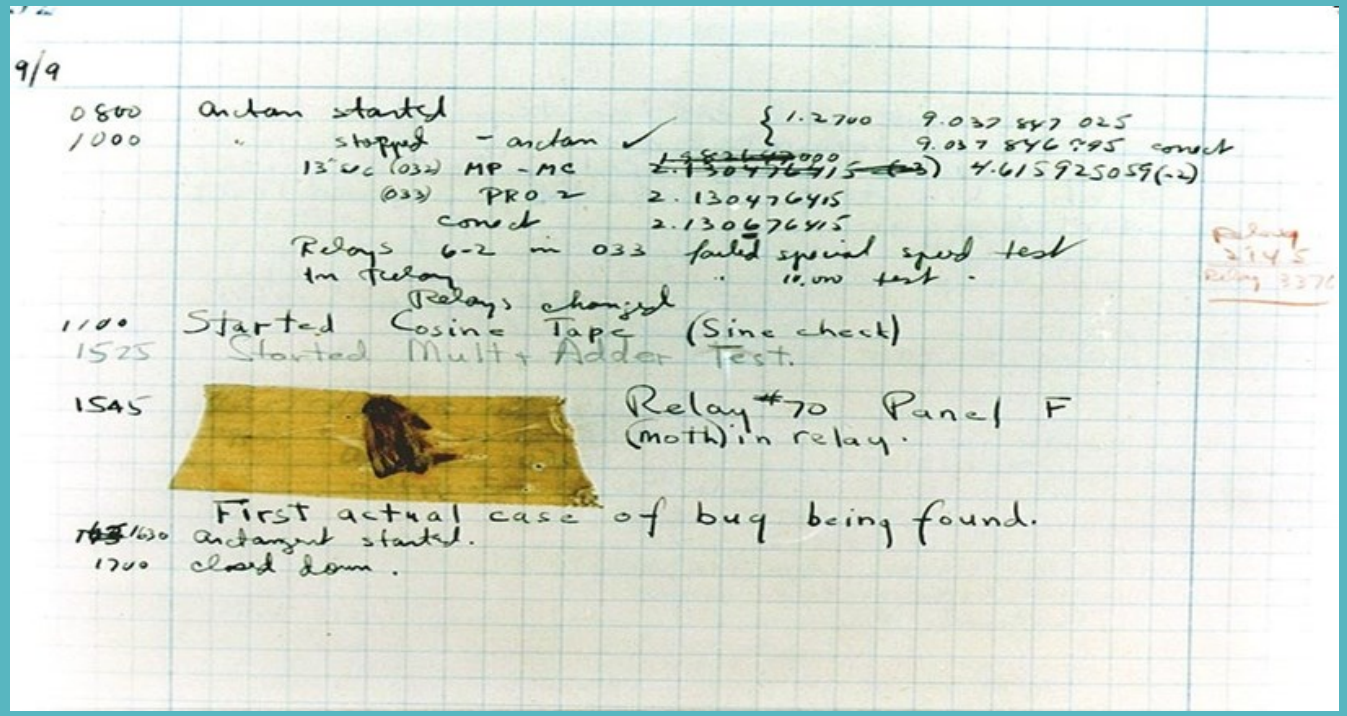
Greška (*Error*)
Napravi je čovek, na primer, prilikom specifikacije zahteva ili kodiranja programa



Mana, defekat (*Fault*)
Posledica greške (na primer, programu nešto nedostaje, ili ima funkciju ali neispravno – „*bug*”)

Otkaz (*Failure*)
Nemogućnost sistema da obavi zahtevanu funkciju najčešće se javlja aktiviranjem (izvršavanjem) defektnog koda.

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



Prva kompjuterska „bubica“

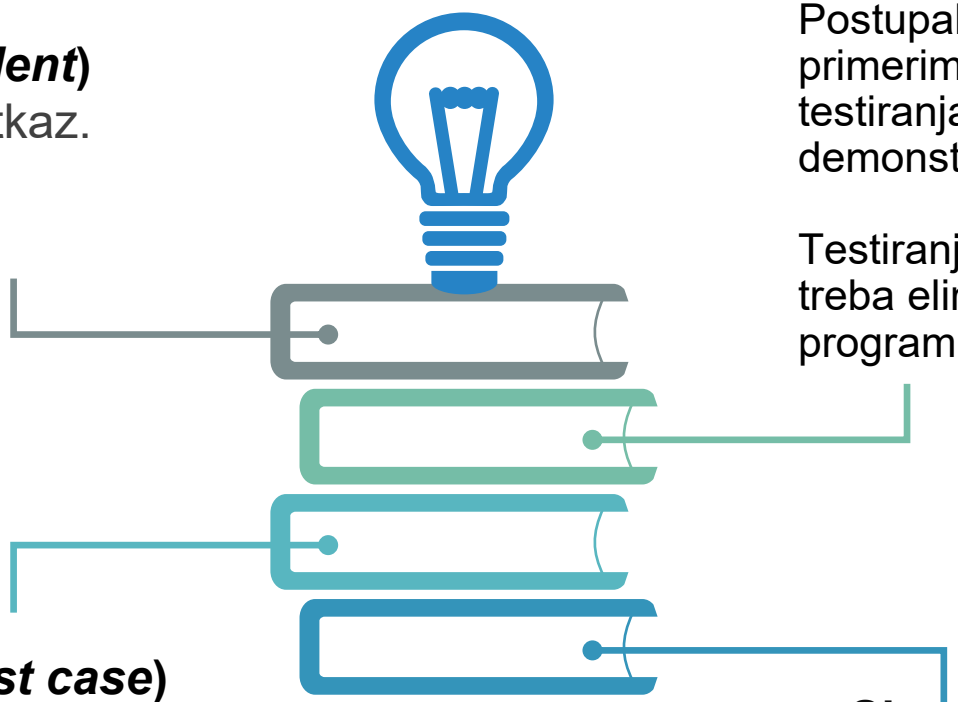
Terminologija (nastavak)

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.

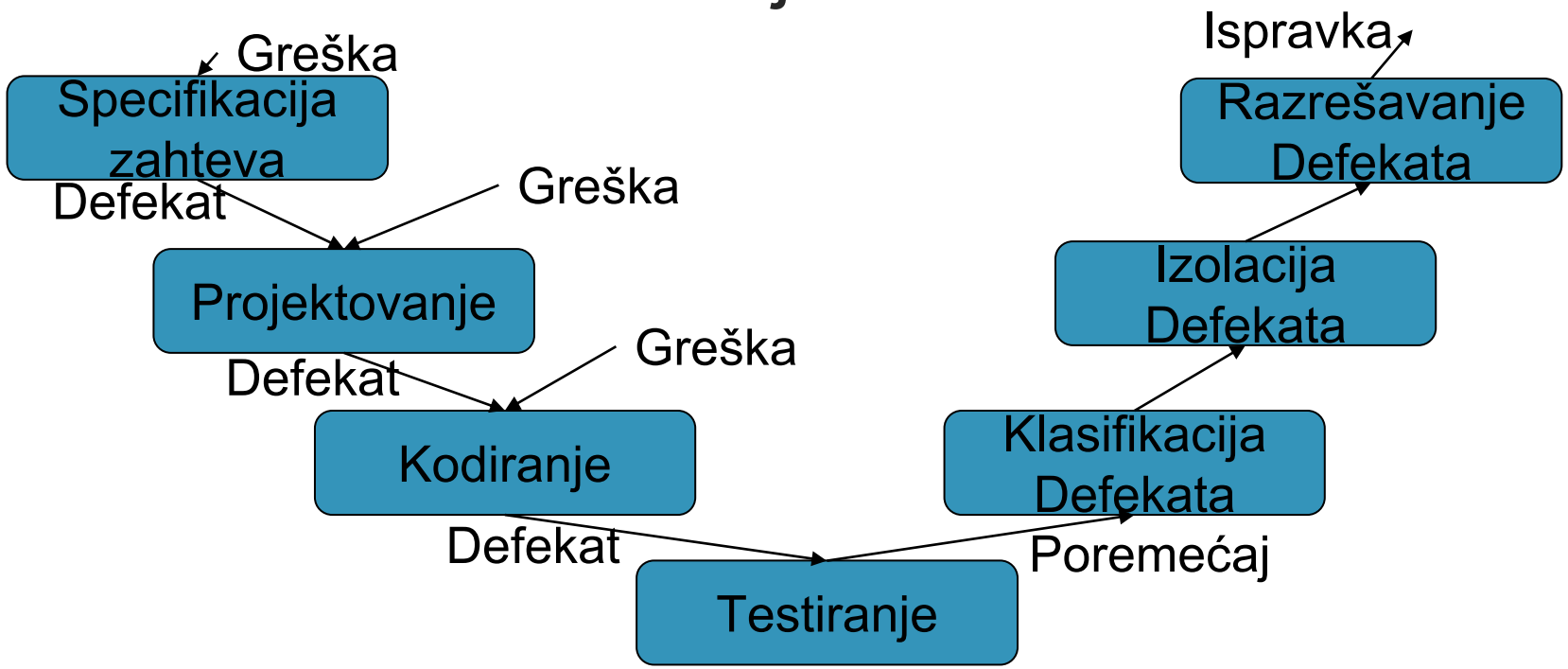
Poremećaj (*Incident*)
Simptom koji korisniku prikazuje otkaz.



Test primer (*Test case*)
Poseduje identitet, i ispituje određeno ponašanje programa.
TP poseduje skup ulaznih veličina i skup očekivanih rezultata.

Skup testova (*Test suite*)
Kolekcija test primera namenjen za testiranje određenog softvera kako bi se pokazao određeni skup ponašanja.

Životni ciklus testiranja



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

Definicija testiranja softvera

G.Myers

Proces izvršavana programa
sa ciljem nalaženja greške.

U širem smislu

Ispitivanje softverskog proizvoda ili servisa sa ciljem objektivnog utvrđivanja kvaliteta.

Deo šire oblasti osiguranja kvaliteta (SQA), koja se bavi kontrolom procesa izrade da se obezbedi kvalitetan finalni proizvod u predvidivom vremenskom roku.

Test primeri i serije testova

Test primer

Test primer je trojka [I, S, O] gde

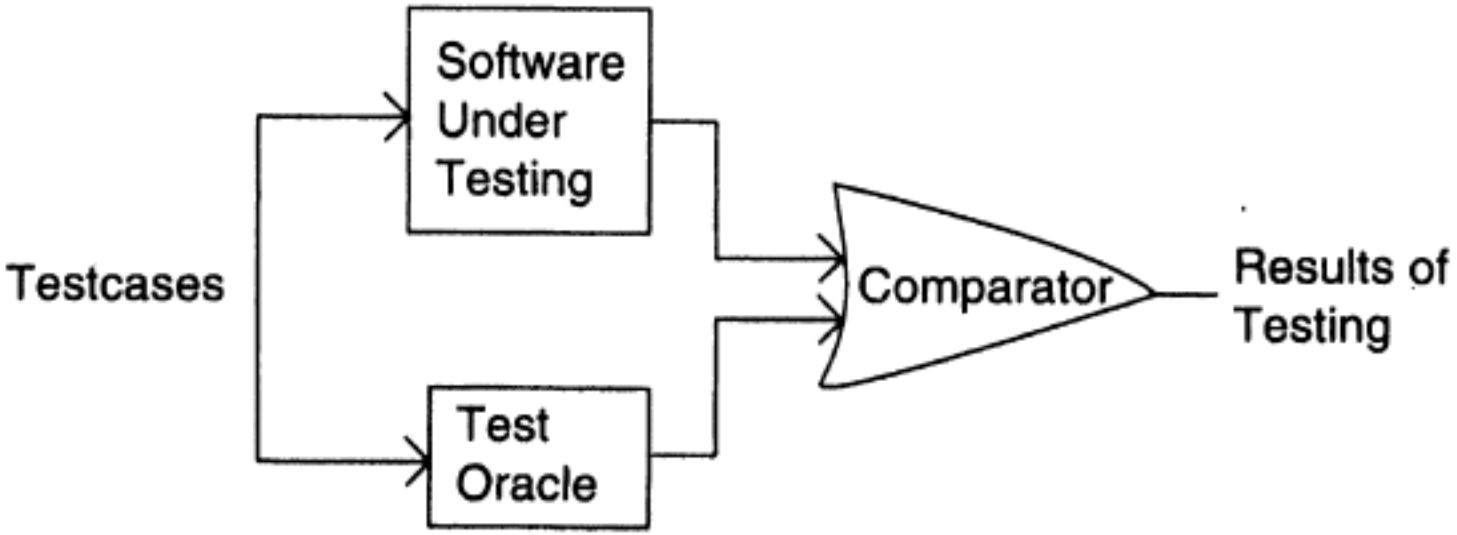
- I predstavlja ulazne podatke
- S je stanje sistema u kome će podaci biti uneti
- O je **očekivani** izlaz

Serija testova

Serija testova je skup svih test primera.

Test primeri se ne biraju na slučajan način.
Umesto toga treba ih **projektovati**.

Očekivani rezultati testa



Predviđanje, predikcija rezultata testa (eng. *test oracle*)

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

Prediktor testa

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

Projektovanje test primera

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

- Broj test primera ne određuje koliko je testiranje delotvorno.

- Da bismo uočili grešku u sledećem kodu:
  if(x>y) max = x; else max = x;
  {(x=3, y=2); (x=2, y=3)} zadovoljava
  {(x=3, y=2); (x=4, y=3); (x=5, y=1)} ne zadovoljava

- Svaki test primer trebalo bi da razotkrije različitu grešku!

Kriterijumi selekcije testova

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

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

Problemi sa kriterijumima selekcije

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

7 principa testiranja softvera

1. Testiranje pokazuje prisustvo defekata, ne može garantovati njihovo odsustvo.

2. Iscrpno testiranje je nemoguće – nije moguće ili isplativo testirati softver za sve moguće kombinacije ulaza.

   Primer: `sort [-cmu] [-ooutput] [-Tdirectory] [-y [ kmem]] [-zrecsz] [-dfiMnr] [-b] [ tchar] [-kkeydef] [+pos1[-pos2]] [file...]`

   Može se identifikovati 20 raznih parametara za sort komandu na Linux OS.
   Uzimajući u obzir njihove moguće vrednosti postoji oko 1.9×10^9 kombinacija.

3. Rano testiranje - testiranje treba da počne što ranije u razvoju softvera. Oko 50% defekata nastaje u fazi određivanja zahteva i projektovanju. Otklanjanje grešaka u kasnijim fazama košta višestruko više.

4. Klasterizacija defekta – mali broj softverskih komponenata odgovorno je za najveći procenat defekata (*Pareto princip*) => potrebno je izvršiti procenu rizičnih modula i fokusirati se na njih.

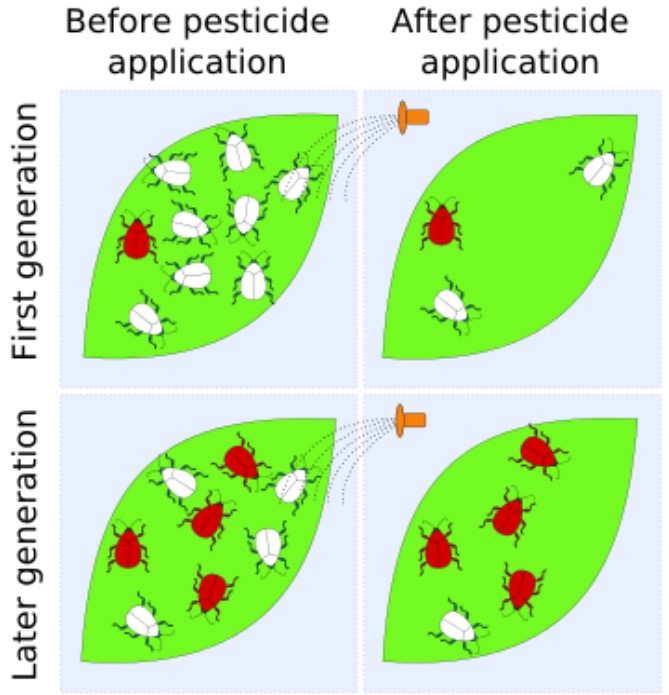
7 principa testiranja softvera

5

Paradoks pesticida: Ponovljena upotreba iste mešavine pesticida za iskorenjivanje insekata tokom poljoprivredne proizvodnje vremenom će dovesti do selekcije insekata koji razvijaju otpornost na pesticid. Isto važi za testiranje softvera.

Ako se sprovede isti skup ponovljenih testova,

metoda će biti beskorisna za otkrivanje novih defekata.



7 principa testiranja softvera

6

Testiranje je zavisno od konteksta - Koriste se drugačiji pristup, metodologije, tehnike i vrste testiranja u zavisnosti od tipa aplikacije. Na primer testiranje veb prodavnice će biti drugačije od testiranja bankomata.

POTERA ZA TROJICOM PLJAČKAŠA
Razneli bankomat i ukrali 15.000 evra

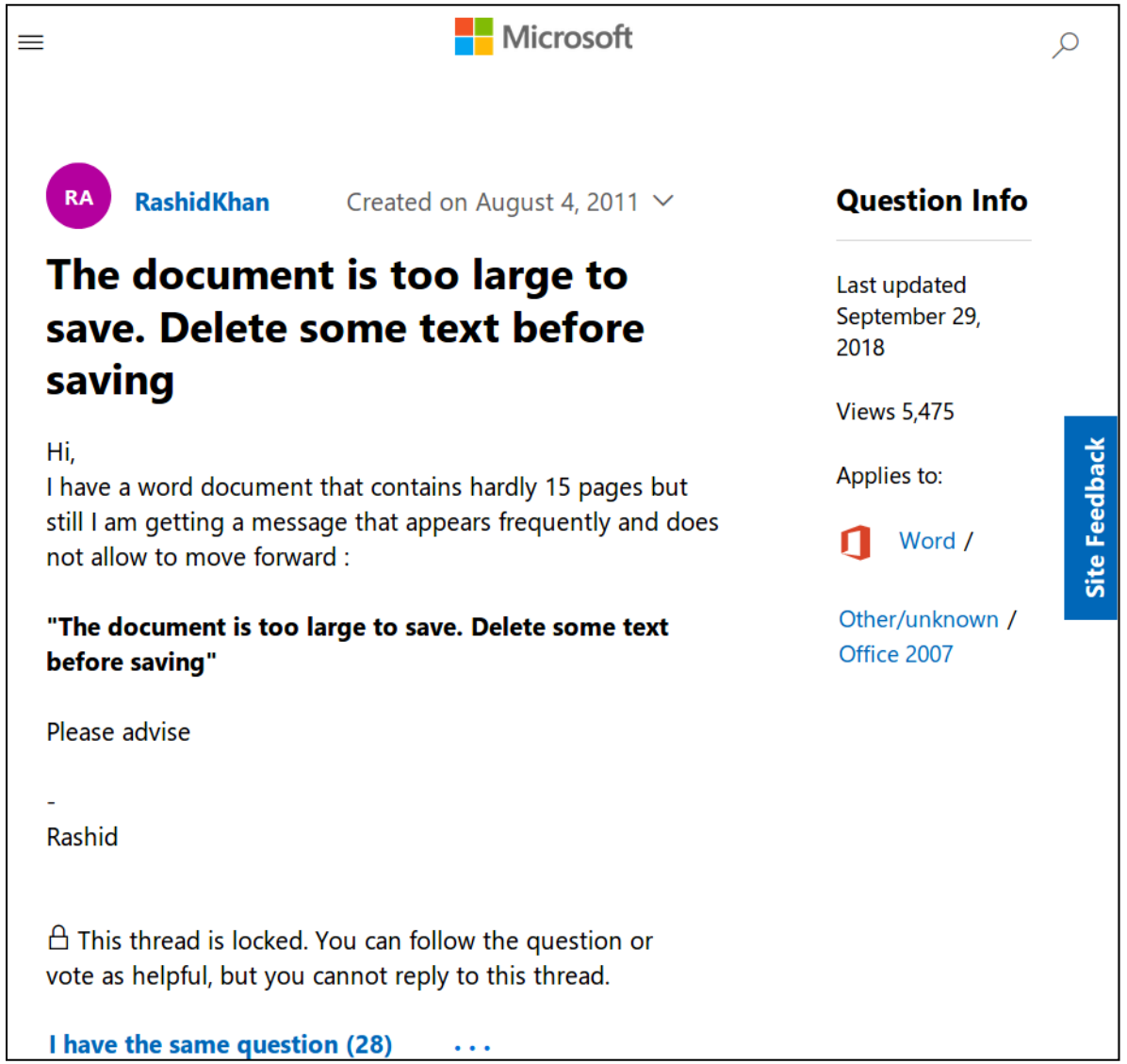
7 principa testiranja softvera

7 Odsustvo grešaka – zabluda:

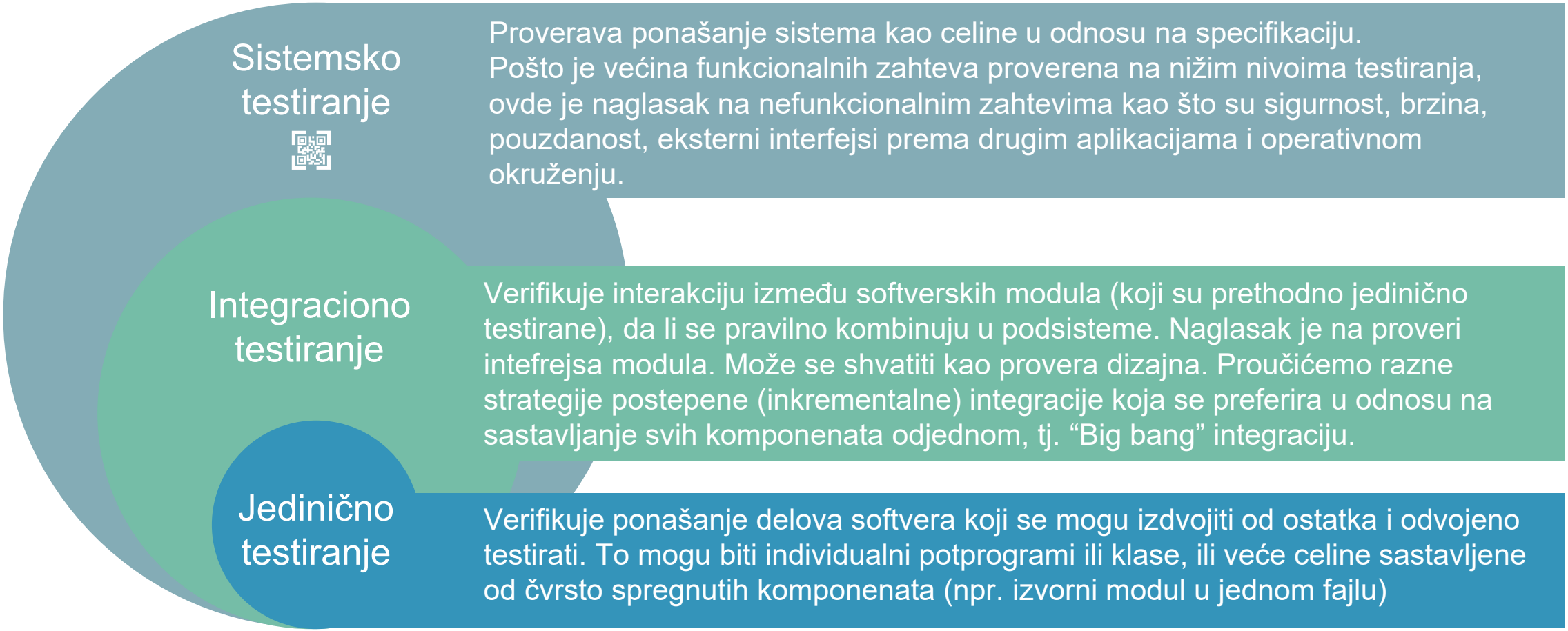
Moguće je da je softver koji je 99% bez greške i dalje neupotrebljiv.

To može biti slučaj ako se sistem ispituje temeljno prema pogrešnom zahtevu.

Testiranje softvera nije samo pronalaženje defekata, već i provera da li softver rešava potrebe korisnika.



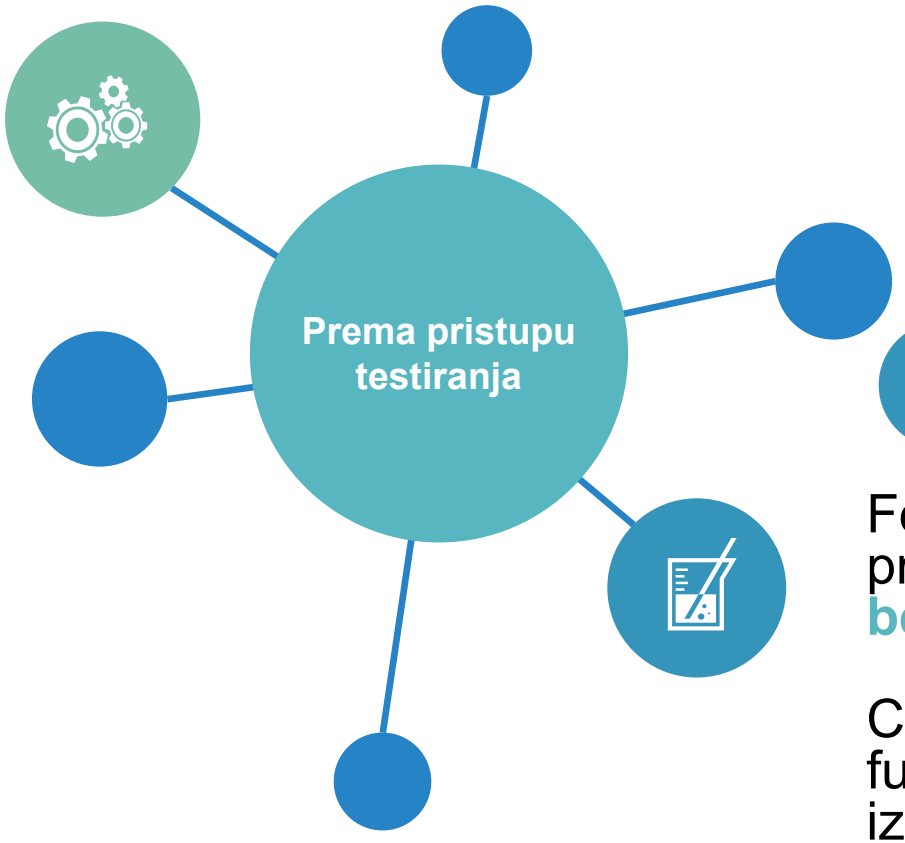
Klasifikacija prema nivou granularnosti



Klasifikacija prema pristupu testiranja

Funkcionalno

Program se posmatra kao funkcija koja mapira vrednosti iz ulaznog domena u izlazni. Implemenacija nije poznata, program predstavlja **crnu kutiju** (eng. *black box*).



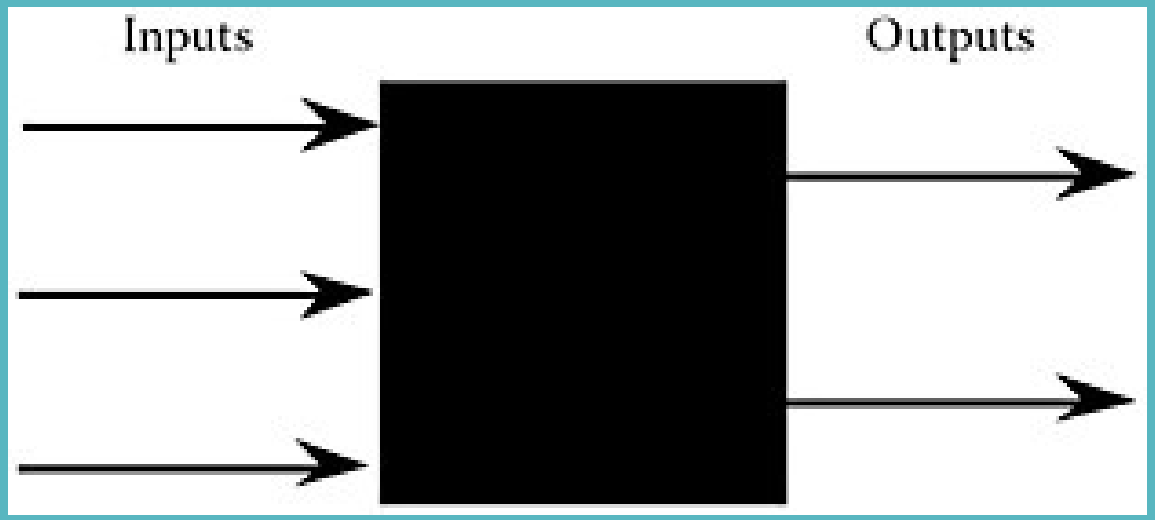
Strukturno

Fokusira se na implementaciju programa. Naziva se još pristup **bele kutije** (eng. *white box*).

Cilj nije da se izvrše sve moguće funkcije programa, već da se izvrše/aktiviraju različite programske i strukture podataka u programu.

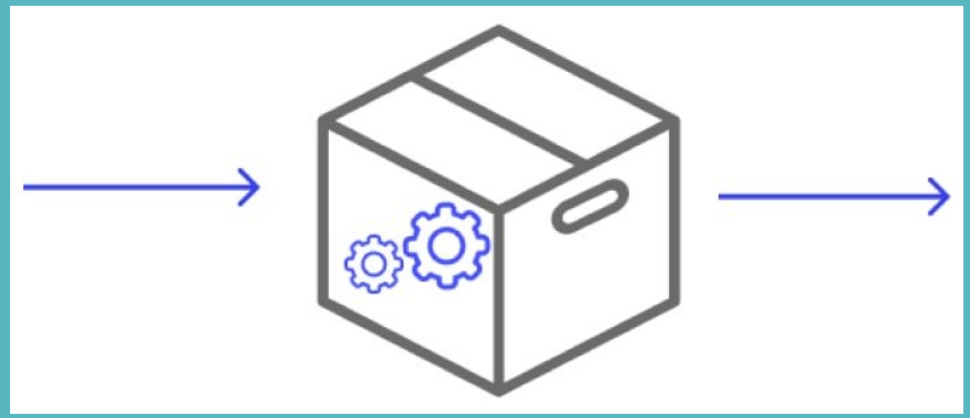
Funkcionalno testiranje – prednosti i nedostaci

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

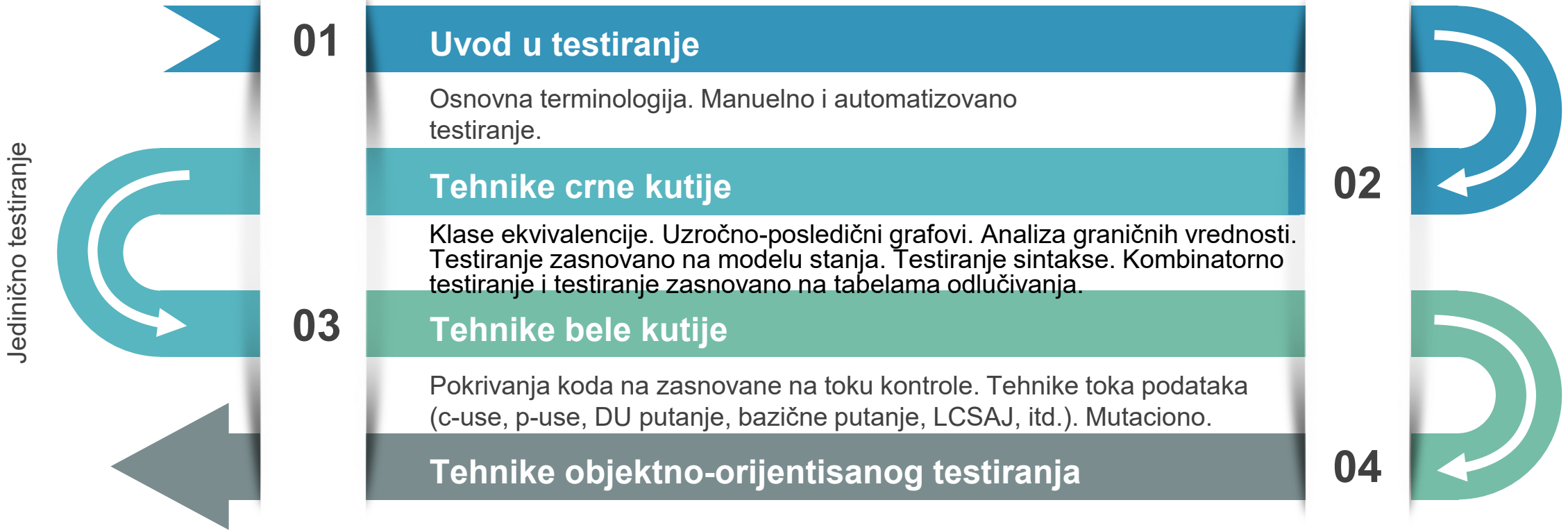


Strukturno testiranje – prednosti i nedostaci

- Različiti kriterijumi strukturnog testiranja znatno preciznije od kriterijuma funkcionalnog testiranja definišu koje programske strukture treba pokriti (npr. iskaze, odluke, putanje).
- Ovim testiranjem nije moguće otkriti da li neki element specifikacije nije implementiran u programu, pošto se specifikacija ne uzima u obzir za selekciju testova.

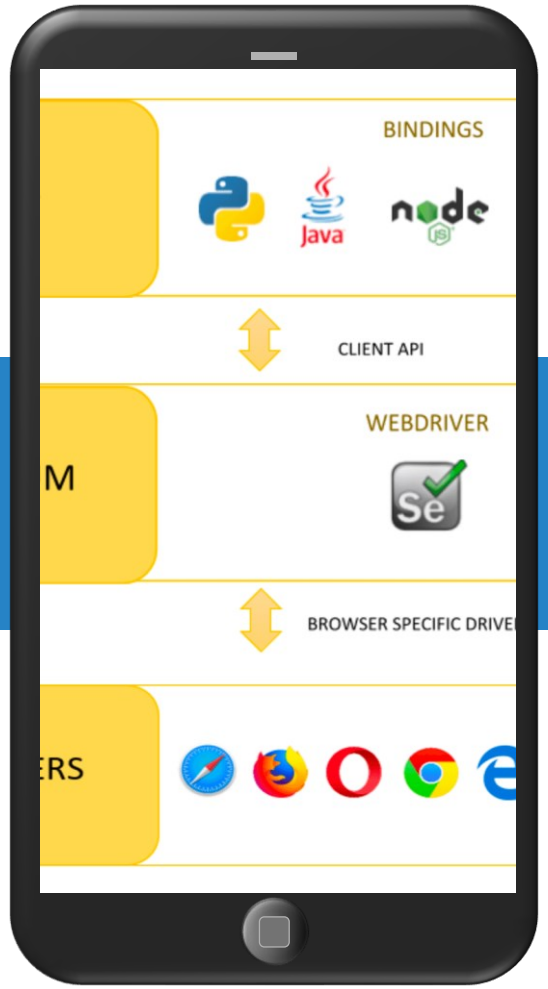


Program predmeta (1)



Program predmeta (2)

Praktična nastava



Laboratorijske vežbe (demonstracija rada alata za testiranje)

Upoznavanje sa test planom i dizajnom test primera. Rad u alatima za testiranje softverva, crnom kutijom (*Selenium IDE*, *Selenium WebDriver*, *TestNG*, *Katalon Studio*), belom kutijom (*JUnit*, *CodeCoverage*, *Mock* alati), automatizovano testiranje (*Cypress*), performansno testiranje (*JMeter*), testiranje sigurnosti, mobilno i GUI testiranje. Upoznavanje sa pojmovima CI/CD/CT.

Domaći zadaci (samostalni rad, kod kuće)

Prvi domaći: testiranje veb aplikacije funkcionalno, kroz manuelne i automatske testove, uz prethodno definisan detaljan test plan i dizajn svih test primera.
Drugi domaći: strukturno testiranje objektno orijentisanih aplikacija pisanih u Java programskom jeziku; analiza pokrivenost koda; upotreba mock alata i testiranje delova koda koji nedostaju.