Softversko inženjerstvo

Elektronski fakultet Niš

Testiranje softvera

Sadržaj

- Definicije osnovnih koncepata
- Tehnike i metode testiranja
- Model procesa testiranja

Šta je testiranje softvera?

- Proces evaluacije sistema ili neke njegove komponente radi verifikacije da zadovoljava specificirane zahteve ili da se identifikuju razlike između očekivanih i stvarnih rezultata
  - ANSI/IEEE 729-1983
- Proces koji se koristi da bi se identifikovala korektnost, kompletnost, sigurnost i kvalitet razvijenog softvera
- Izvršenje testova radi provere da sistem ili aplikacija rade ili ne rade prema specifikaciji zahteva

Test podaci i test slučajevi

- **Test podaci (test data)** – podaci odabrani za testiranje sistema ili neke njegove komponente
- **Test slučajevi (test cases)** – specifičan skup test podataka i pridruženih procedura razvijen sa određenim ciljem (npr. da se izvrši određeni put u programu, da se proveri određeni funkcionalni ili nefunkcionalni zahtev)

Test bed

1. Okolina za testiranje koja obuhvata
   - hardver,
   - alate za instrumentiranje,
   - simulatore i
   - ostalu SW podršku neophodnu za testiranje sistema ili komponente sistema
2. Repertoar test slučajeva neophodnih za testiranje sistema ili komponente sistema

Definicija greške (1)
ANSI/IEEE 729-1983

- **Greška (error)**
  - Razlika između izračunate, dobijene i izmerene vrednosti ili uslova i istinite, specificirane ili teoretski tačne vrednosti ili uslova [ANSI]
  - Neispravan korak, proces ili definicija podataka (npr. nekorektna instrukcija u kodu)
  - Netačan rezultat
  - Ljudska aktivnost čiji je rezultat SW koji sadrži neki nedostatak (npr. ispuštanje ili nepravilna interpretacija korisničkih zahteva u specifikaciji softvera, nekorektno prevođenje ili ispuštanje nekog zahteva u specifikaciji projekta)

Definicija greške (2)
ANSI/IEEE 729-1983

- **Pad (failure)**
  - Nemogućnost sistema ili komponente da izvrši zahtevanu radnju sa specificiranim zahtevima performansi
- **Nedostatak (fault, defect)**
  - Manifestacija greške u softveru
- **Propust (mistake)**
  - Ljudska akcija koja proizvodi netačan rezultat

Klase tehnika za detekciju grešaka

- Statička analiza
- Dinamička analiza
- Formalna analiza

Statička analiza

- Analiza zahteva, projekta, koda ili nekog drugog subjekta bez njegovog izvršavanja sa ciljem da se odredi da li su njegove leksičke i sintaksne osobine jednake unapred predviđenim
- Koristi se u svim fazama razvoja softvera
- Statička analiza uključuje: inspekciju, proveru, čitanje koda, analizu algoritama, grafičke tehnike za analizu kontrole toka i podataka, što se često koristi u automatskim alatima za testiranje.
- Tradicionalno, statička analiza se primenjuje na zahteve, projekat i kod, ali može isto biti primenjena i na test-dokumentaciju, delom i na test-slučajeve u cilju verifikovanja da li oni odgovaraju unapred postavljenim zahtevima

Elektronski fakultet u Nišu

Ciljevi procesa testiranja

- Da otkrije nedostatke ili defekte u softveru zbog čega je ponašanje softvera nekorektno, neočekivano ili nije usklađeno sa svojom specifikacijom
  - Ovaj cilj se ostvaruje kroz **defektno testiranje** softvera
  - Projektuju se test slučajevi koji treba da otkriju defekte

Elektronski fakultet u Nišu

Defektno testiranje softvera

- Testiranje programa da bi se otkrilo prisustvo defekata u programu
- Uspešan defekt test je test koji primorava program da se ponaša na abnomalan način
- Defekt testovi pokazuju prisustvo, a ne odsustvo defekata

Elektronski fakultet u Nišu

Metode testiranja

- Metode testiranja se svrstavaju u tri grupe:
  - Metode crne kutije
  - Metode sive kutije
  - Metode bele kutije
- Kriterijumi za ovu kategorizaciju metoda su:
  - da li se pri razvoju test slučajeva pristupa izvornom kodu SW koji se testira i
  - da li se testiranje vrši preko korisničkog interfejsa (UI) ili preko programskog interfejsa (API)

Metode crne kutije

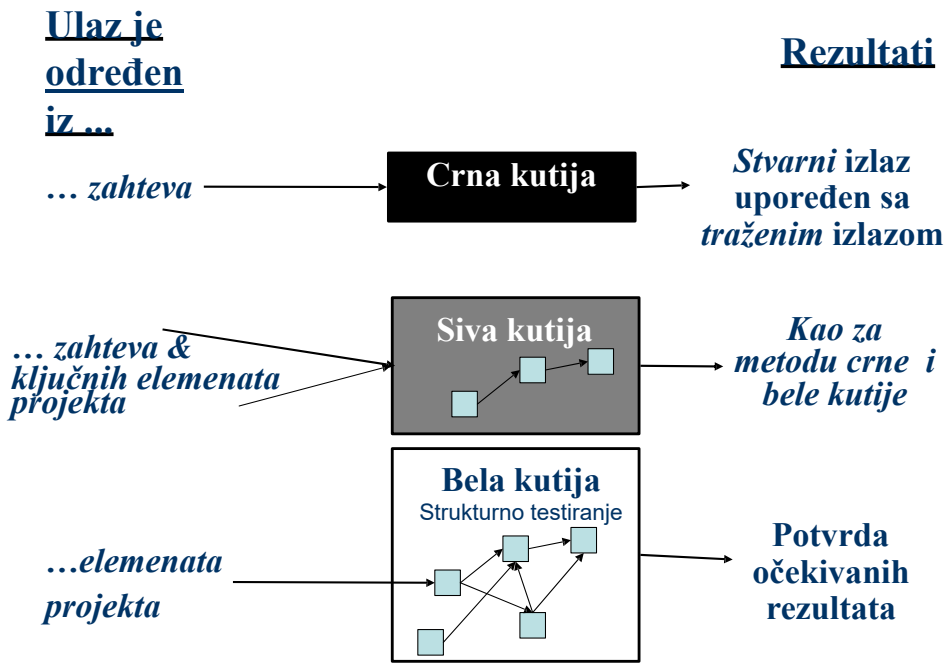
- Test inženjer pristupa SW koji testira preko interfejsa koji je namenjen krajnjim korisnicima (UI)

Metode bele kutije

- Test inženjer pristupa izvornom kodu i može pisati kod koji linkuje sa bibliotekama koje su linkovane sa softverom koji se testira
- Obično se koristi kod komponentnog testiranja

Metode sive kutije

- Test inženjer može postaviti ili manipulisati nekom okolinom za testiranje i može videti stanje SW posle svake akcije
- Koriste ih gotovo isključivo klijent-server test inženjeri ili inženjeri koji koriste bazu podataka kao repozitorijum informacija
- Takođe ih mogu koristiti test inženjeri koji manipulišu XML fajlovima ili konfiguracionim datotekama ili
- Test inženjeri koji poznaju internu strukturu ili algoritam SW koji se testira i mogu pisati testove specifično za određeni rezultat



Testiranje metodom crne kutije

- Prilaz testiranju gde se program posmatra kao „crna kutija“
- Test slučajevi programa se baziraju na specifikaciji sistema ili komponente
- Pristup:
  - Sistemu ili komponenti se dostavljaju ulazi i ispituju se dobijeni izlazi
  - Ako se dobije izlaz koji se ne očekuje, tada se zaključuje da sistem ili komponenta ima neki problem, tj. tada je test detektovao neki defekt

Metode crne kutije

- Podela u particije ekvivalencije (equivalence partitioning)
- Analiza graničnih vrednosti (boundary value analysis)

Podela u particije ekvivalencije (1)

- Ulazni podaci i izlazni rezultati programa se mogu razvrstati u veći broj klasa koje imaju neke zajedničke karakteristike
  - Npr. pozitivni brojevi, negativni brojevi, izbor menija
- Svaka od ovih klasa je jedna **particija (ili domen) ekvivalencije** gde se program ponaša na ekvivalentan način za sve članove klase

Podela u particije ekvivalencije (2)

- Jedan sistematski prilaz za projektovanje test-slučajeva je:
  - Identifikovati sve particije ekvivalencije za sistem ili komponentu koja se testira
  - Projektovati test-slučajeve
    - Treba izabrati bar po jedan test-slučaj iz svake particije ekvivalencije
    - **Preporuka:** Treba izabrati po jedan test-slučaj na granicama particija i bar jedan iz sredine particije

Podela u particije ekvivalencije (3)

- Podeliti ulaze i izlaze sistema u klase ekvivalencije
  - Ako je ulaz petocifreni ceo broj iz opsega 10000 i 99999,
  - klase ekvivalencije su <10000, 10000-99999 i >=100000
- Izabrati test slučajeve na granicama ovih particija
  - 0, 9999, 10000, 99999, 100000, 100001

Strukturno testiranje (testiranje metodom bele kutije)

- Jedan od prilaza za projektovanje test slučajeva gde se test slučajevi izvode iz strukture i implementacije programa
- Za identifikovanje dodatnih test slučajeva koristi se znanje o programu

Kriterijumi strukturnog testiranja

- **Pokrivanje naredbi** – svaka naredba programa treba da se izvrši bar jedanput
- **Pokrivanje odluka** – svaka odluka u programu treba da se pokrije sa TRUE i FALSE bar jedanput i da se svaki ulaz pozove bar jedanput; ovaj uslov pokriva prethodni
- **Pokrivanje uslova** – svaki uslov u programu treba da se pokrije sa TRUE i FALSE bar jedanput i da se svaki ulaz pozove bar jedanput
- **Pokrivanje odluka/uslova** – svaka odluka treba da se testira na sve izlaze i da se svaki uslov pozove bar jedanput
  - Nedostatak je što se uslovi mogu međusobno pokrivati
- **Pokrivanje višestrukih uslova** – sve kombinacije uslova u odluci treba da se pokriju sa TRUE i FALSE bar jedanput i da se svaki ulaz pozove bar jedanput; ovaj uslov pokriva prethodne
- **Pokrivanje puteva** – proći svakim putem u programu. Ovaj kriterijum se koristi kao osnovna mera testiranja

Testiranje puteva

- Cilj je obezbediti da skup test slučajeva bude takav da se svaki **nezavistan put programa** izvrši bar jedanput
- Ako se svaki nezavistan put u programu izvrši, tada sve naredbe u programu moraju biti izvršene bar jedanput
- Sve uslovne naredbe u programu treba da se izvrše bar jedanput
- Početna tačka testiranja puteva je graf toka programa (čvorovi predstavljaju odluke, grane tokove upravljanja)
- Nezavistan put programa je onaj put koji prolazi bar jednom novom granom u grafu toka programa

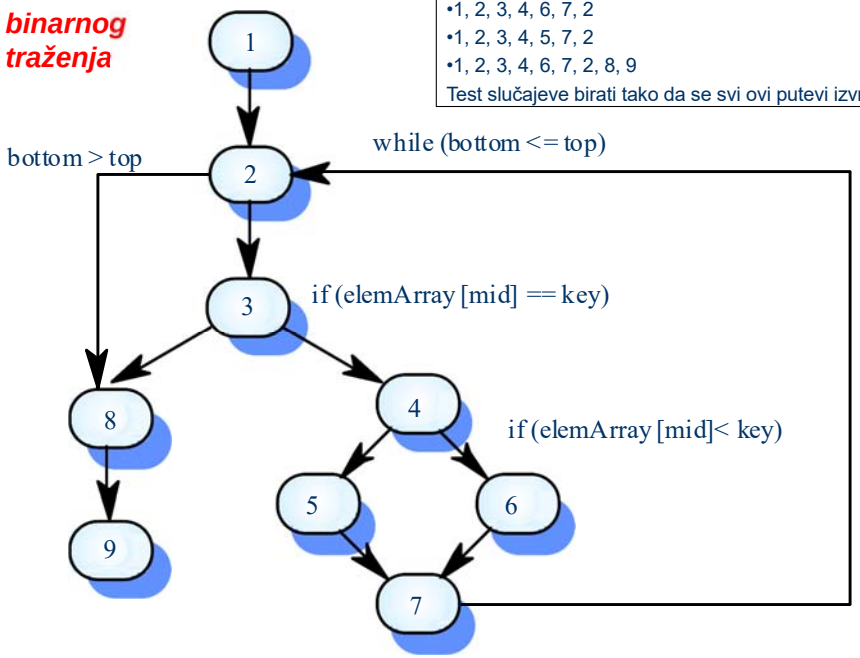
Ciklomatska složenost

- Broj testova za testiranje svih upravljačkih naredbi jednak je ciklomatičnoj složenosti
- Kod strukturnih programa ciklomatična složenost je jednaka broju uslova u programu +1
- Izvršenje svih puteva ne znači izvršenje svih kombinacija puteva

Graf toka binarnog traženja

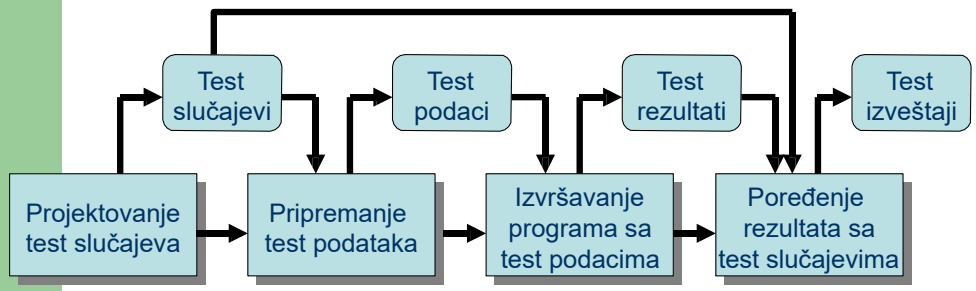
Nezavisni putevi:
- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9

Test slučajeve birati tako da se svi ovi putevi izvrše



Elektronski fakultet u Nišu

Model procesa testiranja softvera



Elektronski fakultet u Nišu

Test slučaj

- U SWE test slučaj je skup uslova na osnovu kojih tester može utvrditi da li SW delimično ili potpuno ispunjava neki zahtev
- Za testiranje SW potreban je veliki broj test slučajeva
- RUP preporučuje kreiranje bar po 2 test slučaja za svaki zahtev
- Test slučaj treba da sadrži opis funkcionalnosti koja se testira i kako treba pripremiti okruženje da bi bili sigurni šta testiramo

Struktura test slučaja (1)

- Uvod
- Aktivnost test slučaja
- Očekivani rezultati

Struktura test slučaja (2)

- **Uvod** – sadrži opšte informacije o test slučaju
  - **Identifikator** – jedinstvenu oznaku test slučaja
  - **Vlasnik ili kreator** test slučaja
  - **Verzija definicije** test slučaja
  - **Naziv** test slučaja
  - **Identifikator zahteva** koji je pokriven ovim test slučajem
  - **Namena** – koja se funkcionalnost testira
  - **Zavisnosti**

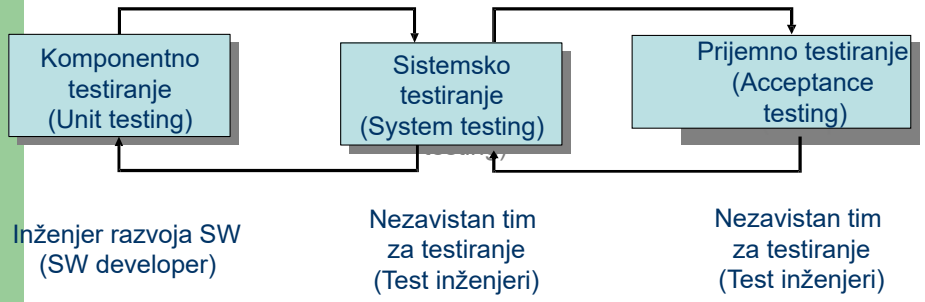
Struktura test slučaja (3)

- **Aktivnost test slučaja**
  - **Okruženje/konfiguracija** test slučaja – potreban HW i SW koji mora biti obezbeđen da bi se izvršio test slučaj
  - **Inicijalizacija** – šta treba da se obezbedi pre izvršenja test slučaja (npr. da se otvori datoteka)
  - **Finalizacija** – opis akcije koja treba da se uradi nakon izvršenja test slučaja (npr. ako test slučaj obori bazu treba je oporaviti pre nego što se test slučaj ponovo izvrši)
  - **Akcije** – šta treba uraditi korak po korak da bi se test kompletirao
  - **Ulazni podaci**
- **Očekivani rezultati** – opisuje šta će tester videti posle izvođenja svih koraka

Test skripte

- Test skripta je kratak program koji se piše na nekom programskom jeziku za testiranje neke funkcionalnosti SW sistema
- Svaki test napisan kao kratak program se posmatra kao automatski test
- Test skripte se pišu korišćenjem specijalnih test alata ili programskih jezika opšte namene

Proces testiranja (1)



Elektronski fakultet u Nišu

Proces testiranja (2)

- Komponentno testiranje
  - Testiranje komponenti programa
  - Testovi se formiraju na osnovu iskustva inženjera razvoja SW
  - Metode bele kutije
- Sistemsko testiranje
  - Testiranje grupe komponenti integrisanih u sistem ili podsistem
  - Testovi se formiraju na osnovu specifikacije SW
  - Testiraju se funkcionalni i nefunkcionalni zahtevi
  - Metode crne, bele ili sive kutije
- Prijemno testiranje
  - Testiranje programa u radnom okruženju pomoću podataka korisnika
  - Otkrivaju se propusti u definiciji zahteva
  - Metode crne kutije

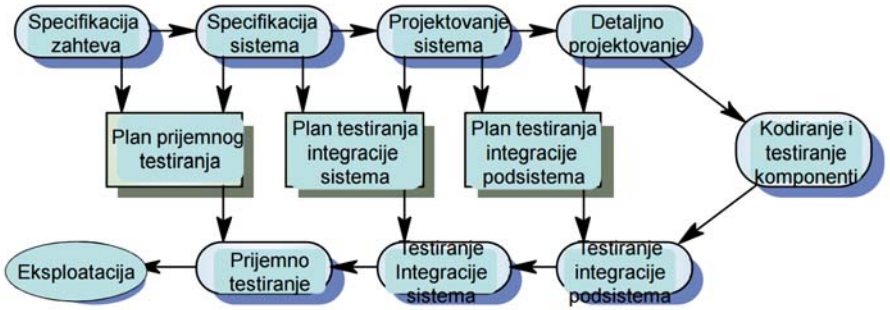
Elektronski fakultet u Nišu

Faze testiranja

- Komponentno testiranje (Unit testing)
  - Testiraju se pojedinačne komponente
- Testiranje modula (Module testing)
  - Testiraju se povezane kolekcije zavisnih komponenti
- Testiranje podsistema (Sub-system testing)
  - Moduli se integrišu u podsisteme i testiraju
  - Fokus je na testiranju interfejsa
- Testiranje sistema (System testing)
  - Testiranje sistema u celini
  - Testiranje bitnih svojstava
- Prijemno testiranje (Acceptance testing)
  - Testiranje sa podacima korisnika da bi se proverilo da li je sistem prihvatljiv za korisnika

Elektronski fakultet u Nišu

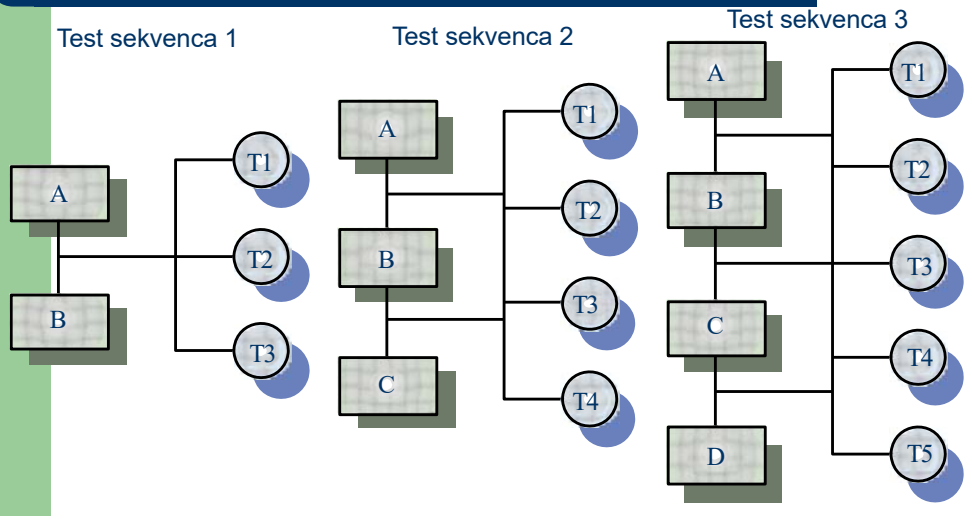
V-model razvoja softvera (Faze testiranja u SW procesu)



Integraciono testiranje

- Testiraju se kompletni sistemi ili podsistemi sastavljeni od integrisanih komponenti
- Integraciono testiranje je testiranje po metodi crne kutije sa testovima koji su izvedeni iz specifikacije
- Prilazi
  - Monolitno
  - Inkrementalno
- Glavni problem je lokalizacija grešaka
- Inkrementalno integraciono testiranje redukuje ovaj problem

Inkrementalno integraciono testiranje



Strategije integracionog testiranja

- Top-down testiranje
  - Startuje se sa najvišim nivoom sistema i sistem se integriše s vrha ka dnu, pri čemu se netestirane komponente zamenjuju stabovima (stub)
- Bottom-up testiranje
  - Integrišu se komponente po nivoima s dna ka vrhu sve dok se ne integriše sistem
  - Za integraciju se koriste drajverski moduli
- U praksi se koristi kombinacija ovih strategija

Objektno-orijentisano testiranje

- Komponete koje treba testirati su objekti klasa
- Veća granulacija nego pojedine funkcije tako da treba proširiti testiranje bele-kutije
- Nivoi OO testiranja
  - Testiranje operacija (metoda) asociranih sa objektima
  - Testiranje objekata klasa
  - Testiranje klastera objekata koji kooperiraju
  - Testiranje kompletnog OO sistema

Testiranje objekata klase

- Testiranje svih operacija (metoda) asociranih sa objektom
- Postavljanje i ispitivanje svih atributa klase
- Ispitivanje objekata u svim mogućim stanjima

- Nasleđivanje otežava projektovanje testova za objekte klasa zbog toga što nisu lokalizovane informacije koje treba testirati

Elektronski fakultet u Nišu

Integracija objekata

- U OO sistemima nivoi integracije su manje jasni
- Klaster testiranje se odnosi na integrisanje i testiranje klastera kooperativnih objekata
- Identifikovati klastere korišćenjem znanja o operacijama objekata i karakteristikama sistema koje su implementirane ovim klasterima

Elektronski fakultet u Nišu

Primeri klaster testiranja

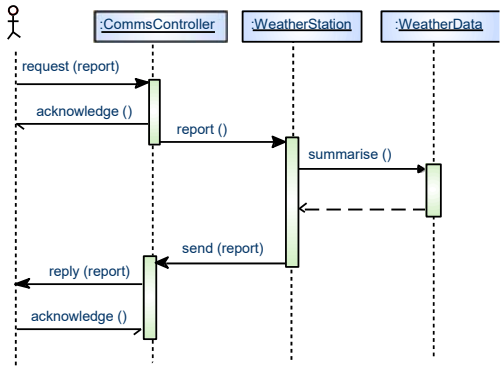
- Testiranje slučajeva korišćenja (use cases) ili scenarija
  - Testiranje se zasniva na interakciji korisnika sa sistemom
  - Prednost je što se testiraju karakteristike sistema koje korisnici prepoznaju
- Testiranje niti (thread testing)
  - Testiraju se odgovori sistema na događaje kao niti obrade kroz sistem
- Testiranje interakcije objekata
  - Testiraju se sekvence interakcija objekta; testiranje se zaustavlja kada operacija objekta ne poziva nijednu operaciju drugog objekta

Elektronski fakultet u Nišu

Testiranje zasnovano na scenariju

- Iz slučajeva korišćenja identifikovati scenarija i dodati podatke iz dijagrama interakcije koji ukazuju na objekte koji učestvuju u scenaruju
- Primer scenarija meteorološke merne stanice - Weather station

- Izvršiti nit metoda
  - *CommsController.request → WeatherStation.report → WeatherData.summarise*
- Ulazi i izlazi
  - Ulaz je zahtev za izveštajem (*report*), a izlazi su potvrda prijema zahteva (*acknowledge*) i traženi izveštaj (*report*)
  - Može se testirati tako što se kreiraju sirovi podaci i proverava da li su obrađeni (*summarise*) korektno
  - Koristiti iste sirove podatke za testiranje objekta *WeatherData*



Alfa, beta i gama testiranje (1)

- Alfa testiranje
  - Vrši se u *alfa fazi* životnog ciklusa softvera, tj. u toku razvoja softvera
  - Koriste se uglavnom metode bele kutije
  - Dodatna inspekcija se može vršiti metodama crne ili sive kutije
    - To rade posebni timovi za testiranje
    - Naziva se drugi stepen alfa testiranja

Elektronski fakultet u Nišu

Alfa, beta i gama testiranje (1)

- Beta testiranje
  - Testiranje koje dolazi posle alfa testiranja, u tzv. *beta fazi* razvoja SW proizvoda
  - Beta verzije SW proizvoda se distribuiraju ograničenom broju krajnjih korisnika koji koriste SW uz obavezu slanja izveštaja razvojnom timu o uočenim nedostacima
  - Alternativa je da se beta verzija SW javno publikuje čime se povećava broj testera i povećava verovatnoća da defekti u programu budu uočeni
  - Koriste se uglavnom metode crne kutije
- Gama testiranje
  - Testiranje koje se nastavlja nakon ulaza SW u eksploataciju