Univerzitet „Džemal Bijedić“ Mostar

Fakultet informacijskih tehnologija

Dinamička verifikacija softvera

- Formalne metode -

Profesor: van. prof. dr Nina Bijedić Student: Edin Pinjić

OVDJE IDE SADRŽAJ

OVDJE IDE INDEKS SLIKA

# Uvod

Od samog početka kreiranja informacijskih sistema, pa sve do njegovog završetka nastaju greške koje su u tim fazama zasigurno neizbježne. Najčešći način provjere tih grešaka, u informacijskim sistemima, je upotreba testiranja i simulacije. Ipak ove tehnike ne mogu garantirati nepostojanje nedostataka u sistemu. Metod koji je mnogo više pouzdaniji po pitanju uočavanja i otklanjanja grešaka u sistemu jeste korištenje formalnih metoda tokom procesa razvoja informacijskih sistema, na primjer, pisanjem formalne specifikacije sistema na kojem se mogu dokazati različita svojstva i matematički dokazati da se implementacija sistema pridržava specifikacije. Cilj ovog završnog rada jeste prikazati dinamičku verifikaciju softvera upotrebom formalnih metoda, te prikazati šta su formalne metode i kako mogu smanjiti broj grešaka u sistemu. U pomenutim formalnim metodama postoje dvije vrste verifikacija, odnosno testiranja softvera, a to su dinamička i statička verifikacija. Statička verifikacija softvera je vrsta testiranja koje se izvršava kako bi se provjerili nedostaci u softveru bez izvršavanja samog softvera. Međutim, kod dinamičke verifikacije je obratno, tj. kôd softvera se pokreće i na taj način se otkrivaju anomalije istog. Treba napomenuti da je statičko ispitivanje, za razliku od dinamičkog mnogo jeftinije, a i greške ispitivanja su lakše uočljive i pogodnije za ispravljanje, jer vrši se u ranoj fazi razvoja samog softvera. Kada se govori o dinamičkoj verifikaciji, zavisno o pogledu testiranja, mogu se kategorizirati tri familije. Prva je „Unit testing“, tj. testiranje jedinki, gdje se vrši testiranje na malo, odnosno testira se samo jedna funkcija ili klasa. Moglo bi se reći da je ovaj način testiranja ujedno i najjednostavniji. Druga familija testiranja jeste testiranje na veliko u koje spadaju modul testiranje i integracijsko testiranje. U ovoj familiji se vrši testiranje grupa klasa, dakle ili jedan modul ili više modula. I treća familija testiranja jeste test prihvatljivosti koji je ustvari formalni test koji je definiran da provjerava prihvatljivosti kriterija samog softvera. U pomenutoj familiji pripadaju dva testiranja, a to su test funkcionalnosti i test nefunkcionalnosti. U ovom završnom radu će biti prikazana dinamička verifikacija korištenjem integracijskog testiranja.

Postoje uglavnom dvije vrste tehnika koje se koriste u statičkom ispitivanju,a to su: pregled i statička analiza. U ovom seminarskom radu bit će prikazana statička verifikacija korištenjem statičke analize. Statička analiza uključuje procjenu kvalitete koda koju pišu programeri. Za analizu koda i usporedbu istog sa standardom koriste se različiti alati. Također pomaže u identificiranju sljedećih nedostataka: mrtvi kod, beskonačne petlje, varijable sa nedefiniranom vrijednošću i pogrešna sintaksa.

Statička analiza u ovom seminarskom radu će biti prikazana preko tri parametra, a to su: ciklomatična složenost (cyclomatic complexity), indeks održivosti (maintaibility index) i broj linija koda. Ciklomatična složenost je mjerenje složenosti izvornog koda koje je povezano sa brojnim pogreškama kodiranja. Izračunava se razvijanjem grafikona kontrolnog toka koda koji mjeri broj linearno nezavisnih staza kroz programski modul.[1] Indeks održivosti koda jedan je od važnijih mjernih podataka koji pokazuju koliko je lahko održavati kod. Kada se počne pisati kod za određenu metodu, indeks održivosti ima vrijednost 100, te sa nekvalitetnim promjenama na kodu i drugim faktorima metrike, indeks se počinje smanjivati. Smanjivanje indeksa ovisi o broju spojenih klasa u jednu cijelinu, kao i o dubini naslijeđivanja među klasama. Linije koda označavaju broj izvršivih redaka koda u metodi. Ovaj je broj približan broj, a temelji se na intermedijarnom jeziku koda (IL). Uključuje samo izvršne retke koda, tako da su isključeni komentari, zagrade i bijeli razmak. Kao što se može pretpostaviti, što je više koda u aplikaciji, to je više koda za održavanje. Stoga, za linije koda, niska vrijednost je dobra, a visoka vrijednost loša. Za potrebe ovog seminarskog rada koristit će se alati NDepend i alat Code Analysis Tool koji je dio razvojnog okruženja Visual Studio. NDepend je alat za analizu statičkog koda u kojem je primarni cilj omogućiti analiziranje aplikacije i sticanje jasne slike o tehničkom dugu, prekršenim pravililma, ciklomatičnoj složenosti unakrsnim ovisnostima objekata, te stepenu povezanosti među njima. S tim, NDepend je alat dizajniran za dobivanje više razine statičke analize koji nam pomaže dizajnirati bolji sistem, a ne pisati ispravniji kod. Code Analysis Tool je alat koji nam daje informacije o indeksu održivosti sistema, ciklomatičnoj složenosti i broju linija koda,a na taj način razvojni programeri mogu u bilo kojem trenutku provjeriti na kojoj je razini kvaliteta koda softvera koji se razvija. S tim je moguće da se prijevremeno reagira na neke propuste u kodu,tako što se u ranim fazama razvoja rješava nepotrebna kompleksnost u određenim dijelovima koda. Tokom rada na ovom seminarskom radu za potrebe testiranja koristit će se jedan .NET demo projekat.