Univerzitet „Džemal Bijedić“ Mostar

Fakultet informacijskih tehnologija

Dinamička verifikacija softvera

- Formalne metode -

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

**IZJAVA O AUTORSTVU**

Ja, **Edin (Kemal) Pinjić,** student Fakulteta informacijskih tehnologija, Univerziteta „Džemal Bijedić“ u Mostaru, pod punom moralnom, materijalnom i krivičnom odgovornošću,

**izjavljujem**

da je rad pod naslovom:

**„Dinamička verifikacija softvera“**

u potpunosti rezultat sopstvenog istraživanja, gdje su korišteni sadržaji (tekst, ilustracije, tabele itd.) drugih autora jasno označeni pozivanjem na izvor i ne narušavaju bilo čija vlasnička ili autorska prava.

Mostar, datum Edin Pinjić

Sadržaj

[1. Uvod 5](#_Toc41557419)

[2. Formalne metode 6](#_Toc41557420)

[2.1. Klasifikacija formalnih metoda 7](#_Toc41557421)

[2.2. Prednosti formalnih metoda 8](#_Toc41557422)

[2.3. Nedostaci formalnih metoda 8](#_Toc41557423)

[3. Dinamička verifikacija 9](#_Toc41557424)

[3.1. Jedinično testiranje (Unit testing) 9](#_Toc41557425)

[3.1.1. Praktični prikaz jediničnog testiranja 9](#_Toc41557426)

[3.2. Integracijsko testiranje (Integration testing) 10](#_Toc41557427)

[3.2.1. Praktični prikaz integracijskog testiranja 10](#_Toc41557428)

[4. Zaključak 11](#_Toc41557429)

[5. Literatura i reference 12](#_Toc41557430)

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 specifikacija. 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. Ovaj rad će imati fokus na dinamičkoj verifikaciji korištenjem integracijskog testiranja nad već postojećim sistemom pod nazivom „Studentski servis“ u razvojnom okruženju Visual Studio 2019. Međutim, prije samog integracijskog testa, potrebno je uraditi i jedinično testiranje, što će se u daljem radu i prikazati.

Cilj ovog rada je da se prikažu osnovni koncepti dinamičke verifikacije i da se kroz primjer softvera „Studentski servis“ prikažu formalne metode, te na koji način se mogu koristiti kako bi se otkrile nepravilnosti u kôdu. „Studentski servis“ je aplikacija koja je rađena kao seminarski rad za predmeta Razvoj softvera II.

# Formalne metode

Formalne metode su tehnike dizajniranja sistema koje koriste strogo određene matematičke tehnike i alate za specifikaciju, dizajn i verifikaciju softverskih i hardverskih sistema. Ovo znači da se specifikacija sastoji od dobro formiranih izjava koristeći matematičku logiku i da se formalna verifikacija sastoji od edukcija izvedenih iz te logike. Snaga formalnih metoda jeste da omogućavaju verifikaciju kompletnog prostora sistema i da dokazane osobine, koje stoje u sistemu, će stojati i za sve moguće ulaze. Kada se formalne metode ne mogu koristiti kroz čitavu fazu razvoja sistema, zbog kompleksnosti sistema, nedostatka alata ili nekih drugih razloga, one se i dalje mogu koristiti na dijelovima sistema, kao na primjer u dijelovima sistema koji su zaduženi za sigurnost kritičnih komponenti. Veliki broj različitih formalnih metoda je rezultat različitih metoda modeliranja i dokaznih pristupa potrebnih različitim domenama primjena. Također različite faze razvoja sistema mogu zahtijevati različite metode, tehnike ili alate. Iako je veliki broj razvijenih formalnih metoda rezultat istraživačkih napora na univerzitetima, sve je više formalnih metoda dostupnih izvan akademskih zajednica. Kada se novi sistem implementira obično je prvi korak pisanje specifikacije zahtjeva. Ta specifikacija treba da tačno opiše željeno ponašanje sistema te da bude kompletna i nedvoznačna, što može biti teško za izvesti. Nakon završetka specifikacije, programer, koji također razumije specifikaciju, prevodi istu u kôd te ispravlja sve nejasne i dvoznačne dijelove. Programer koji piše kôd može napraviti grešku u kôdu, zatim sama veličina sistema, koja može biti tolika da je teško pratiti sve dijelove sistema koji moraju raditi prema uputama specifikacije. Nadalje, često se koristi više različitih timova u razvoju sistema, što također može biti izvor grešaka, jer će svaki tim imati svoj način rada, svoju interpretaciju specifikacije zahtjeva i informacija dijeljenih tokom faze razvoja sistema. Tokom i nakon faze pisanja koda, obično se vrše testiranja sistema kako bi se utvrdilo da ispunjava postavljane uslove te da nema prisutnih grešaka (bug-ova). Testiranje velikih i složenih sistema može koštati mnogo vremena, te zbog same veličine sistema i količine kôda, iscrpno testiranje nije ostvarljivo. Uprkos tome, kada je u pitanju sistem, koji je sigurnosno kritičan, njegova ispravnost mora biti garantovana, što zahtjeva iscrpno testiranje ili način dokazivanja da je kod ispravno implementiran prema zadanoj specifikaciji. Koncept formalnih metoda nam predstavlja alate za matematičko opisivanje sistema ili dijelova sistema u specifikaciji i za dokazivanje da rezultirajući program zapravo ispunjava uslove zadane u specifikaciji. Formalna verifikacija je precizna i nema mogućnosti za pogrešnom interpretacijom. Naravno, u praksi se ne može u potpunosti garantovati da implementacija nema grešaka, jer same formalne metode mogu imati svoje mane ili čak može biti i greška u samom dokazu. Međutim, povećano korištenje formalnih metoda i alata će rezultirati boljim implementacijama i pouzdanijim metodama i alatima. Ograničenje formalnih metoda je to što mogu samo biti korištene za dokazivanje ispravnosti sistema prema specifikaciji. Stoga, samo zato što je implementacija matematički dokazana da prati pravila specifikacije, to ne znači da je specifikacija sama po sebi ispravna i bez grešaka. Bez obzira, osobine i dalje mogu biti dokazane kako bi se pojačalo uvjerenje da specifikacija tačno predstavlja željene funkcionalnosti.

Ukratko, korištenjem formalnih metoda mogu se pronaći greške u ranim fazama razvoja sistema, te se neke klase grešaka mogu skoro u potpunosti ukloniti.

## Klasifikacija formalnih metoda

Postoji nekoliko različitih klasifikacija formalnih metoda, ali generalno gledajući formalne metode se koriste u dva aspekta:

• Da bi se primijenilo željeno ponašanje sistema prema specifikaciji. Specifikacija je model sistema koji opisuje kako se sistema mora ponašati, a formalne metode se koriste za validaciju metode;

• Da potvrdi da implementacija ima isto ponašanje kao što je navedeno u specifikaciji ili da dobavi implementaciju koja ima isto ponašanje kao i specifikacija. Ovdje se govori o formalnim vezama implementacije i specifikacije;

Još jedan način klasifikacije je na osnovu nivoa korištenja formalnih metoda:

• Nivo 0: Kada se formalne metode koriste samo za opisivanje željene funkcije i za usmjeravanje razvoja;

• Nivo 1: Kada se formalne metode koriste za provjeru funkcija;

• Nivo 2: Kada se formalne metode koriste za provjeravanje cijelog sistema kroz sve funkcije;

Još jedna generalna klasifikacija formalnih metoda može biti prema tome kako je model sistema opisan:

• Kao tranzicija sistema sa stanjima, tranzicija i stanje transformacije;

• Kao neka programska logika sa pre (prije) i post (poslije) uslovima kao i sa aksiomima i pravilima zaključka;

Treći način klasifikacije formalnih metoda je:

• Formalne metode koje se koriste za specificiranje i analizu specifikacije;

• Formalne metode koje se koriste za specificiranje i dokazivanje osobina specifikacije (formalna verifikacija);

• Formalne metode koje se koriste za specificiranje i izvođenje implementacije iz specifikacije;

• Formalne metode koje se koriste za specificiranje i transformaciju specifikacije, transformacije koje ili sakrivaju detalje specifikacije ili obogaćuju specifikaciju sa više detalje;

## Prednosti formalnih metoda

Formalne metode su dobar način otkrivanja pogrešaka u specifikaciji, utvrđivanju tačnosti sistema i predstavljanju sistem na nedvosmislen način. Upotreba ovakve metode može smanjiti troškove ukoliko se koristi na nekim manjim, ali kritičnim dijelovima sistema za koje je bitno da su precizni, bez grešaka. Neke prednosti korištenja formalnih metoda su sljedeće:

* bolji uvid u zahtjeve, otklanjanje nesporazuma, smanjenje mogućnosti grešaka (što sve pridonosi pouzdanosti softvera);
* analiziranje matematičkim metodama (potpunost, konzistentnost);
* može služiti kao podloga za formalnu verifikaciju implementiranog sistema;
* mogućnost „animacije“ specifikacije u svrhu prototipiranja;

## Nedostaci formalnih metoda

Kao i sve drugo, ni pomenuta metoda nije bez mana. Unatoč svim gore navedenim prednostima, formalne metode ograničene su u praktičnom razvoju softvera, kako za veće sustave tako i za kritične dijelove sustava. Stoga je i malo stručnjaka koji imaju iskustva u razvoju ovakvom metodom. Razlozi za nekorištenje formalne metode su sljedeći:

* Nerazumljiva je korisnicima i menadžmentu (korisnicima je teže provjeriti ispunjava li specifikacija njihove zahtjeve);
* Zahtijeva posebno osposobljene softverske inžinjere;
* Nije pogodna za interaktivne sisteme i sučelja;
* Nije skalabilna, za veće sustave količina posla postane prevelika;
* Nije kompatibilna s agilnom metodom razvoja softvera;

# Dinamička verifikacija

Dinamička verifikacija softvera nije ništa drugo nego testiranje samog softvera kada je u stanju da se uspješno izvrši njegovo pokretanje. Jer kao što je ranije pomenuto, ova vrsta formalne metode se vrši tako što se kôd softvera mora prvo pokrenuti, pa tek nakon toga vršiti poznate metode testiranja. Kada se govori o dinamičkoj verifikaciji, odnosno dinamičkom testiranju, zavisno o pogledu testiranja, mogu se kategorizirati tri familije:

* Prva familija je „Unit testing“, tj. testiranje jedinki ili jedinično testiranje, 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 je 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;
* 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;

## 3.1. Jedinično testiranje (Unit testing)

Jediničnim testiranjem provjeravamo ispravnost pojedini jedinice bez ikakve veze sa drugim jedinicama i to je prva vrsta testiranja koja se izvodi nad našim sistemom. Ovim načinom testiranja možemo biti sigurni da se sve pronađene greške odnose samo na testiranu jedinicu (unit), te ih je zbog toga jednostavnije ispraviti. Ukoliko se provjeri da se svaki dio programskog kôda ponaša na željeni način, manja je vjerovatnoća da će se pojaviti greške u završnom proizvodu. Prednost jedinično testiranja je pronalazak grešaka u veoma ranoj fazi implementacije, zbog čega je jednostavnije i brže ispraviti pronađene anomalije. Osim toga, jedinično testiranje pomaže pri održavanju programskog kôda, jer se osigurava da nastale promjene ne utiču na ostatak programskog kôda. Ukratko rečeno, ovakvo testiranje je najjednostavnije i jako je prosto, ali svejedno ima velik uticaj u pronalaženju i ispravljanju grešaka u programskom kôdu. U današnjem svijetu razvoja softvera, skoro da je nezamislivo upotpuniti razvoj sistema bez ijednog jedinično testa.

### Praktični prikaz jediničnog testiranja

U nastavku će biti prikazana primjena jediničnog (unit) testiranja nad već pomenutim sistemom „Studenski servis“ u razvojnom okruženju Visual Studio 2019.

## 3.2. Integracijsko testiranje (Integration testing)

Nakon jediničnog testiranja slijedi vršenje integracijskog testiranja našeg sistema. Integracijom nazivamo podsistem ili veću jedinicu koja nastaje povezivanjem programskih jedinica. Iako je svaka programska jedinica testirana, nakon povezivanja nekoliko jedinica potrebno je provjeriti da li surađuju ispravno. Integracijskim testiranjem provjeravamo zajednički rad nekoliko programskih jedinica i njihovu međusobnu interakciju. Međutim, jedan od najvećih problema pri integracijskom testiranju je određivanje najboljeg načina povezivanja programskih jedinica. Cilj je povezati jedinice u smislene cjeline kako bi se sustav lakše testirao.

Za grupiranje programskih jedinica može se koristiti neki od sljedećih pristupa:

* Inkrementalna integracija – Integracija koja se realizira sa postupnim dodavanjem

programskih jedinica;

* Integracija odozgo prema dole (Top - Down) – Integracija se vrši od najvišeg modula prema nižim modulima;
* Integracija odozdo prema gore (Bottom - Up) – Integracija se vrši od najnižih modula prema višim modulima;
* Sendvič integracija – Predstavlja kombinaciju pristupa odozgo prema dole i odozdo prema gore;
* Veliki prasak – Testiranje svih pojedinačnih modula, te se u jednom koraku, nakon testiranja, integriraju u cjelinu;

### 3.2.1. Praktični prikaz integracijskog testiranja

U nastavku će biti prikazana primjena integracijskog testiranja nad već pomenutim sistemom „Studentski servis“ i već postojećim programskim jedinicama u razvojnom okruženju Visual Studio 2019.

# Zaključak

# Literatura i reference

[1] Dr. Sc. Denis Mušić, PDF i video materijali iz predmeta „Razvoj softvera I“, FIT, Mostar 2018/2019;

[2] Dr. Sc. Denis Mušić, PDF i video materijali iz predmeta „Razvoj softvera II“, FIT, Mostar 2018/2019;

[3] Ivana Jukić (2018.), Diplomski rad, Automatsko testiranje programa, Sveučilište u Zagrebu Fakultet Organizacije Informatike, Varaždin;

[4] Dragana Ostopanj, Softversko inženjerstvo – Odjel za matematiku, Formalna specifikacija

[5] Andreas Spillner, Tilo Linz, Hans Schaefer, Software Testing Foundations, A Study Guide for the Certified Tester Exam, Rocky Nook 4th Edition;

[6] Web stranice:

-<https://www.tutorialspoint.com/software_testing_dictionary/dynamic_testing.htm>

-<https://www.guru99.com/dynamic-testing.html>

-<http://docplayer.rs/180177341-Verifikacija-softvera.html>

-<https://hr.natapa.org/difference-between-static-and-dynamic-testing-1812>

-<https://www.semanticscholar.org/paper/Between-Testing-and-Verification%3A-Dynamic-Software-Godefroid/af40c62d370b8842aab758bae646154732221404>

-<http://www.verifikacijasoftvera.matf.bg.ac.rs/>