Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike
Smjer: Matematika i računarstvo

Slaven Viljevac

Verifikacija i validacija softvera

Seminarski rad

Osijek, 2018.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike
Smjer: Matematika i računarstvo

Slaven Viljevac

Verifikacija i validacija softvera

Seminarski rad
Mentor: doc. dr. sc. Alfonzo Baumgartner

Osijek, 2018.

Sadržaj

1 Uvod

U ovom radu bavit ćemo se verifikacijom i validacijom softvera. Pogledat ćemo koji su razlike između verifikacije i validacije, te pogledati neke od metoda koje se koriste za verifikaciju i validaciju. Osvrnut ćemo se na najčešći oblik validacije i verifikacije što je *statička verifikacija* i pogledati neke tehnike koje spadaju u statičku verifikaciju. Nakon toga reći ćemo nešto o testiranju softvera, koje su vrste i faze testiranja. Na posljetku samo ćemo spomenuti što su to CASE alati i navesti neke od alata.

2 Općenito o verifikaciji i validaciji

Verifikacija i validacija je skup aktivnisti kojima želimo utvrditi odgovara li softver specifikaciji i potrebama korisnika. Koristeći verifikaciju i validaciju želimo se uvjeriti da naš softver odrađuje posao za koji je namijenjen. Ukoliko je naš softver validan i verificiran, to ne znači da on ne može imati grešku, već da je pouzdan i dobar za ono za što ćemo ga koristiti. Budući da softver testiramo za ograničenim skupom podataka, nitko ne može garantirati da ne možemo pronaći podatke na kojima će program "pucati". Stupanj tolerancije na greške ovisi o očekivanjima korisnika, uvjetima na trzištu i samoj funkciji softvera.

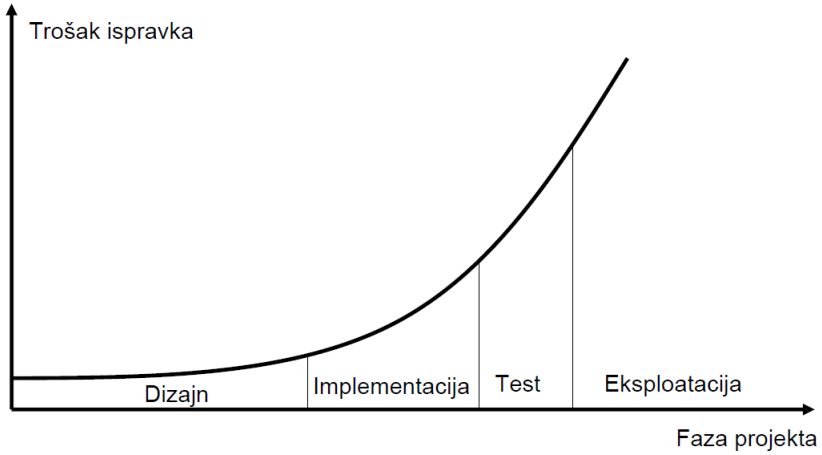
2.1 Razlika između validacije i verifikacije

Ova dva slična pojma se zapravo dosta razlikuju. Verifikacija daje odgovor na pitanje ("Are we building the product right?"). Sam proces verifikacije je proces procjene programskog proizvoda te da li taj proizvod ispunjava zahtjeve i uvijete koji su zadani u dokumentaciji. S druge strane imamo pojam validacija koja daje odgovor na pitanje ("Are we building the right product?") i taj proces provjerava i ocjenjuje programski proizvod provjerava odgovara li on stvarnim potrebama korisnika.
Uz pojam verifikacije i validacije često se veže i pojam testiranje. Testiranje je proces izvođenja programa pod određenim uvijetima pri čemu promatramo rezultate i procjenjujemo neki aspekt sustava. Testiranje je dakle aktivnost ocjenjivanja i poboljšanja proizvoda kroz pronalaženje neispravnosti i pogrešaka. Nikada nećemo planirati test potpunog programskog proizvoda. S financijeske strane gledajući, ispravljanje grešaka koje se pronađu ukoliko se testiranje provodi u kasnim fazama razvoja znatno povećvaju troškovi što je vidljivo iz slike 2.1. Testiranjem se dokazuje prisutstvo neispravnosti a ne njihovo nepostojanje.

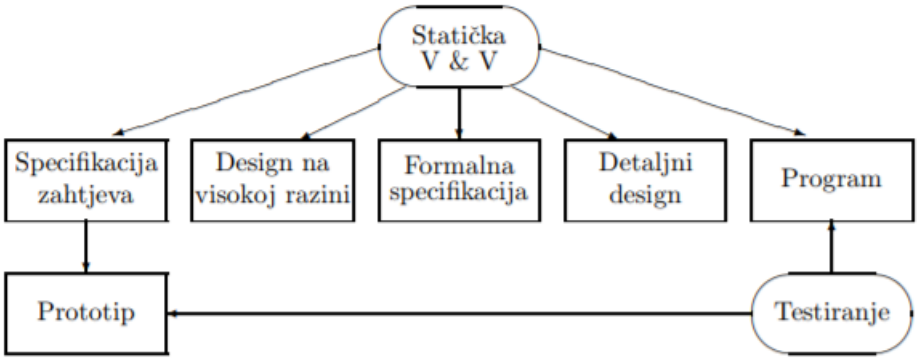
2.2 Metode za verifikaciju i validaciju

Metode koje se koriste za verifikaciju i validaciju najčešće možemo svrstati u dvije grupe, a to su *statička verifikacija i validacija* i *testiranje*. Objema metodama ćemo se kasnije zasebno posvetiti, no u ovom trenutku dovoljno je reći da se statička verifikacija i validacija svodi na analizu i provjeru dokumenata, modela sustava, dizajna i porgramskog koda, te se odvija bez stvarnog izvođenja softvera, dok se testiranje svodi na pokusno izvođenje softvera ili dijelova softvera, na "umjetnim" podatcima uz detaljno analiziranje rezultata. Uobičajeno je koristiti metodu testiranja iako statička verifikacija i validacija ima sljedeće prednosti.

- jedna statička provjera može otkriti puno grešaka, dok testiranje otkriva po jednu grešku budući da nakon greške dolazi do pada aplikacije.
- Kod statičke metode dolazi do izražaja znanje o programiranju budući se osobe koje obavljalju provjeru oslanjaju na iskustvo i greške koje se najčešće pojavljuju i na njih se fokusiraju.



Slika 2.1: Svaka kasno otkrivena greška je skuplja za ispravak.



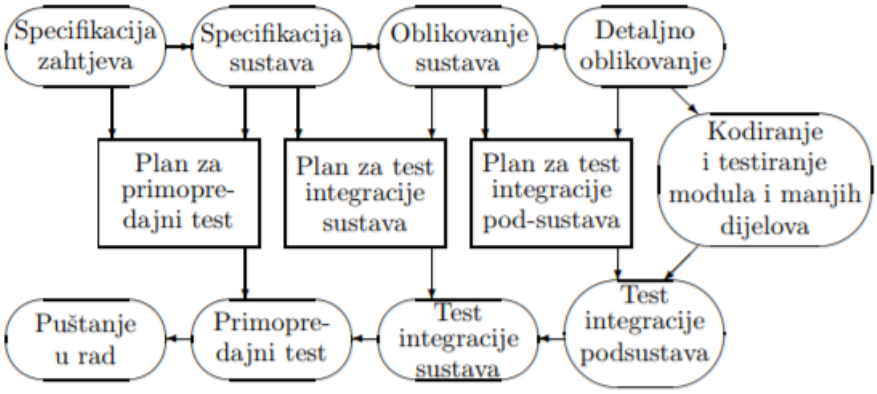
Slika 2.2: Primjena verifikacije i validacije u fazama razvoja.

2.3 Verifikacija i validacija unutar softverskog procesa

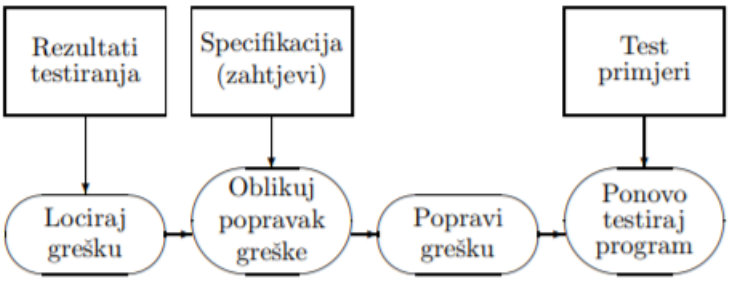
Ako pogledamo shematski prikaz dan slikom 2.2 možemo uočiti da se statička verifikacija i validacija može obavljati u svim fazama softversko procesa dok se testiranje obavlja samo na prototipu ili na programu. Dakle program se prije testiranja podvrgava provjeri statičkim metodama. Poželjno je da planovi za testiranje budu definirani što je prije moguće, dakle već u fazi specifikacije i oblikovanja sustava što možemo vidjeti iz slike 2.3. Plan testiranja je dokument koji sadrži sljedeće: popis zahtjeva koji će se testirati, opis procesa testiranja, opis djelova softvera koji će se testirati, kalendar testiranja, opis na koji će se način bilježiti rezultati testiranja, popis hardvera i softvera koji je potreban za testiranje, te ograničenja koja utječu na sam proces testiranja.

2.4 Odnos verifikacije, validacije i debugiranja

Kako smo već ranije spomenuli, tijekom verifikacije i validacije dolazi do otkrivanja grešaka. Dolazi do promjena na softveru da bi se popravile greške. Iako je proces debuggiranja često integriran s validacijom i integracijom, to su ipak različiti procesi. Verifikacija i validacija je kako smo već spomenuli proces koji utvrđuje postojanje grešaka, a debuggirenje je proces

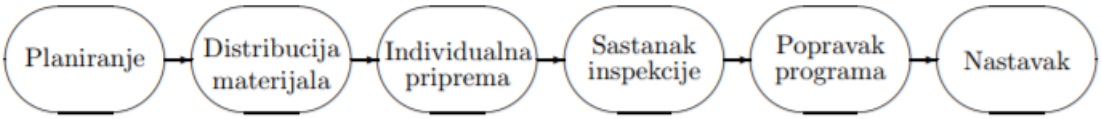


Slika 2.3: Nastanak planova za testiranje.



Slika 2.4: Proces debuggiranja.

koji pronalazi i popravlja grešku. Kako izgleda sam proces debuggiranja možemo vidjeti na slici 2.4. Teško je za opisati što je zapravo proces debuggiranja. Može se reći da je debuggiranje "umjetnost" jer se greške gotovo uvijek pojavljuju simptomatski. Službena definicija debuggiranja bi bila da je debuggiranje proces pronalaženja i rješavanja grešaka ili problema u računalnom programu koji onemogućuje korektno izvršavanje operacija računalnog sustava ili softvera. Programeri koji se bave debugiranjem najčešće se oslanjaju na iskustvo i poznavanje čestih grešaka, no uz iskustvo, za pomoć mogu se koristiti i debuggeri. Debbugeri su računalni programi koji se koriste za uklanjanje grešaka te za testiranje i provjeru pravilnosti rada programa.



Slika 3.1: Proces inspekcije programa.

3 Statička verifikacija

U prethodnom poglavlju mogli smo vidjeti kako se statička verifikacija i validacija može odvijati u raznim fazama softverskog procesa. Sada ćemo se posvetiti najčešćem obliku verifikacije i validacije, a to je statička verifikacija. Statička verifikacija radi analizu i pregled koda, bez da se sam program stvarno izvodi. Kako bi se testiranje ubrzalo statička verifikacija se izvodi prije samog testiranja programa.
Pogledajmo sada neke tehnike koje spadaju u statičku verifikaciju.

Inspekcija Programa
Ovu tehniku korisitle su velike korporacije kao što su HP i IBM. Inspekcija programa je reguliran proces u kojem sudjeluje grupa ljudi sa strogo zadanim ulogama:

- *autor ili vlasnik*: programer koji je odgovoran za programski kod i za popravak grešaka,
- *inspektor*: pronalazi greške, propuste i nekonzistentnosti u programu,
- *čitač*: glasno čita programski kod na sastanku,
- *zapisničar*: bilježi rezultate sastanka,
- *moderator*: planira i organizira sastanke, upravlja procesom.

Na slijedećoj slici, Slika 3.1, možemo vidjeti kako teče proces inspekcije, a najvažnija faza je dobro pripremljen sastanak na kojem će inspekcijska grupa pregledati program i pri tome otkrivati greške. Uspjeh sastanka ovisi o sljedećem:

- postoji precizna specifikacija programa,
- postoji ažurna i dovršena verzija koda,
- postoji "check-lista" čestih programerskih grešaka,
- članovi su upoznati s organizacijskim standardima i voljni su surađivati.

Moderator je pri tome osoba koja odlučuje treba li ponoviti cijeli postupak.

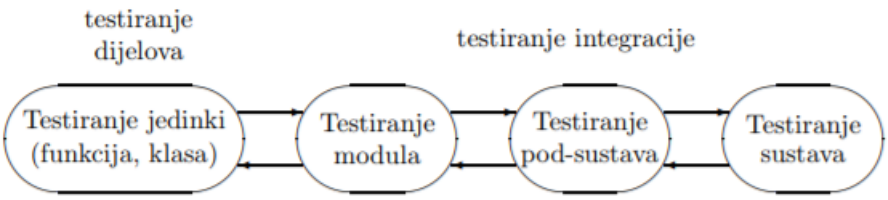
Automatska statička analiza
Automatska statička analiza je metoda koja koristi softverske alate koji prolaze kroz tekst koda i otkrivaju meguće greške i anomalije u tom tekstu. Prilikom korištenja ove metode, skreće se pozornost na mjesta na kojima bismo očekivali da se pojavi greška.

Automatska statička analizaje dobra za jezike poput *C*-a, dok su npr. u Javi izbjegnute neke jezične konstrukcije koje bi lako dovodile do grešaka, primjerice nema goto naredbi (teže je za stvoriti nedostupan kod), upravljanje memorijom je automatski (nema pointera.) Neki od programa koji se koriste za statičku analizu su: Checkmarx SAST koji se koristi u jezicima C, C#, Python itd., Codacy (Python, JavaScript...), Veracode (C, C++, Java ...), IBM Security AppScan (C, C++, Java ...).

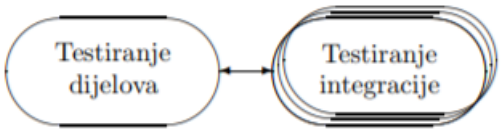
Formalna verifiakcija
Formalna verifikacija svodi se na matematičko dokazivanje konzistentnosti programa sa svojom specifikacijom. Ovu metodu moguće je koristiti samo ako vrijedi slijedeće:

- semantika programskog jezika je formalno definirana
- postoji normalna specifikacija za program

Dokazivanje se vrši tako da polazni uvijeti iz specifikacije i naredbe iz programa koriste kao činjenice, i iz njih se izvode uvijeti koji trebaju biti zadovoljeni nakon izvršenja programa.



Slika 4.1: Proces testiranja grešaka.



Slika 4.2: Sažeti prikaz procesa testiranja grešaka.

4 Testiranje softvera

Testiranje je proces izvođenja programa/komponenti pod određenim uvjetima promatrajući rezultate i procjenjujući neki aspekt sustava ili komponente. Proces analize dijelova programskog proizvoda u svrhu utvrđivanja razlika s zahtjevanim uvjetima i procjena svojstva tih dijelova. Na poslijetku, testiranje je aktivnost ocjenjivanja i poboljšanja kvalitete proizvoda pronalaskom neispravnosti i pogrešaka. Svrha testiranja može biti validacija ili verifikacija, a mi ćemo se posvetiti dijelu testiranja u svrhu verifikacije.

Vrste i faze testiranja

Ovisno o načinu zadavanja testnih podataka i o vrsti zahtjeva razlikujemo dvije vrste testiranja. Prvo je **Testiranje grešaka ili Black box testing** koje služi za provjeru funkcionalnih zahtjeva. Testni slučajevi su izvedeni iz specifikacije zahtjeva, podatci nisu konstruirani tako da simuliraju pravilan rad aplikacije već da ukažu na prisutstvo greške. Druga vrsta je **Statičko testiranje ili White/Glass box testing** i ona se koristi za provjeru nefunkcionalnih zahtjeva. Ovim testiranje analizira se interna logika i struktura.Testni podatci su takvi da sliče stvarnim podatcima.

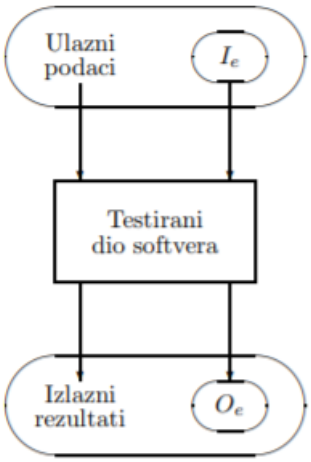
Osvrnimo se sada na testiranje grešaka. Kako bi te takvo testiranje smatralo uspješnim, potrebno je pokazati da softver ne radi dobro. Nakon testiranja grešaka radi se debuggiranje. Bitno je za primijetiti da se može pokazati da u softveru postoje greške, no ne može se dokazati da grešaka nema.

Kako se često radi o velikim sustavima, testiranje se odvija u nekoliko faza koje možemo vidjeti na Slici 4.1 ili sažeti prikaz Slika 4.2.

Testiranje vršimo tako da testiramo dijelove softvera, testiramo jesu li ti dijelovi ispravno integrirani u veće cjeline, veće cjeline u još veće itd. U fazi testiranja integracije fokusiramo se na interakciju među pojedinim dijelovima i na fukncionalnost cjeline. U toj fazi često dolazi do otkrivanja i grešaka u dijelovima pa se zbog toga faze mogu ponavljati.

Testiranje dijelova softvera

Kako bi testiranje pojedinih dijelova softvera bilo uspješno važno je testiranje provoditi s odgovarajućim podatcima. Pogledat ćemo sada neke od pristupa koji se razlikuju po odabiru testnih podataka.



Slika 4.3: Black-box testing.

Prvo ćemo nešto reći o **Funkcionalnom testiranju (black-box testing)**. U ovoj tehnici testiranja testira se funkcionalnost softvera bez da pregledavanja unutarnje strukture koda i implementacijskih detalja. Testiranje se bazira isključivo na specifikacijama i zahtjevima softvera. Black-box testing se fokusira na inpute i outpute softverskog sustava bez da se zamara s internim poznavanjem softverskog programa. Na slici 4.3 možemo vidjeti shematski prikaz black-box testinga. Neka I_e označava proizvoljni (nepoznati) skup podataka koji izazivaju neispravno ponašanje, koje ćemo prepoznati po izlazima iz O_e. Pri tome se prilikom testiranja nastoji izabrati podatke (ulaze) koji pripadaju skupu I_e sa što većom vjerojatnosti.

Pogledajmo nadalje **Strukturalno testiranje (white-box testing)**. To je metoda testiranja softvera u kojoj tester poznaje internu strukturu, dizajn i implementacijski dio softvera. Tester odabire inpute za isprobavanje puteva kroz kod i određuje odgovarajuće outpute. Prednost ove metode je što se testiranje može krenuti u ranijim dijelovima razvoja, nema potrebe čekati da GUI bude dostupan. Spomenimo još i **Testiranje po putovima (path testing)**. Ovdje se radi o posebnoj vrsti strukturalnog testiranja. Ovim pristupom želimo se uvjeriti da je svaki put kroz program izvršen barem jednom. Najčešće se koriste dynamic analyzer tool da bi se provjerilo da je sav kod izvršen.

Testiranje integracije

Kako smo već ranije spomenuli, testiranje integracjie obavlja se nakon što se manji dijelovi integriraju u veće cjeline. Ovim testiranjem želimo doći do grešaka koje nastaju u međukomunikaciji različitih dijelova. Testiranje integracije najčešće se korisit u sustavima koji su objektno orijentirane građe. Provodi se nakon Unit testiranja i prije Sistemskog testiranja. Pogledajmo sada neke od tipova sučelja između udruženih dijelova.

- **Proceduralno sučelje,** gdje jedan dio softvera poziva proceduru drugog dijela softvera,
- **Parametarsko sučelje,** podaci ili reference na njega prenose se kao parametri u pozivu procedure iz jednog dijela sotvera na drugi,

- **Slanje poruka,** djelovi softvera komuniciraju preko poruka tako da jedan dio pošalje poruku drugom (zahjeva uslugu) pri čemu povratna poruka sadrži rezultat,
- **Zajednička memorija,** blok memorije je dostupan raznim djelovima softvera, pri čemu jedan dio sprema a drugi dio čita podatke s bloka.

Kada govorimo o greškama u korištenju sučelja, one se mogu razvrstati tri vrste. **Nerazumjevanje sučelja, Pogrešna uporaba sučelja i greške u timingu.**
Pristupi mogu biti sljedeći:

- **Big Bang** pristup gdje su svi ili većina dijelova spojeni i testira se odjednom. Koristi se kada testni tim dobije kompletan softver u paketu.
- **Top Down** pristup koristi se za testiranje po razinama, prvo se testira najviša razina, zatim niža itd. najniža razina testira se zadnja.
- **Bottom Up** pristupom testiramo prvo donje razine i idemo prema višim.
- **Sandwich/Hybrid** pristup kombinira Bottom Up i Top Down pristupe.

CASE alati za testiranje
Svako testiranje softvera je specifično, zato se nastoje kreirati vlastite kombinacije CASE alata za testiranje. CASE alate možemo podijeliti na *klasične* i *prave*. Klasični alati su npr. debuggeri i kompajleri, dok se pravi alati mogu razvrstati na *Upper, Lower i Integrirane.* Lower CASE alati se koriste u kasnijim fazama razvoja softvera, a Upper CASE alati na početnim fazama. Neki od primjera CASE alata su: Microsoft Visio, Dia, Enterprise architect i mnogi drugi.

5 Zaključak

Verifikacija i validacija softvera treba demonstrirati da produkt zadovoljava sve svoje nužnosti. Korisnici će imati više povjerenja u softverski produkt koji je tijekom programiranja podvrgnut provjeri nego u onaj koji je podvrgnut zanemarivoj procjeni i testiranju prije objavljivanja.

Literatura:

[1] R. Manger, *Softversko Inženjerstvo*, Zagreb, 2005

[2] https://en.wikipedia.org/wiki/Debugging

[3] http://users.monash.edu/ jonmc/CSE2305/Topics/12.24.SWEng3/html/text.html#debugging

[4] http://www.riteh.uniri.hr/zav_katd_sluz/zr/nastava/proginz/
materijali/Testiranje%20programskog%20proizvoda.pdf