Formalne metode, kao metode koje putem matematičke logike daju tačnu formalnu specifikaciju sistema, tj. da li se implementirani sistem pridržava specifikacija, nisu ništa novo. Ovakve metode su korištene još u ranim fazama naše civilizacije. Npr. Platon nam je u Sofistu 400-te godine prije Hrista dao struktuiranu analizu. Dakle, mnogo prije nego što je bilo poznato za šta se konkretno upotrebljava.

U pomenutim formalnim metodama postoje dvije vrste verifikacija (testiranja), a to su dinamička i statička verifikacija.

Statička verifikacija je vrsta testiranja koje se izvršava kako bi se provjerili nedostaci u softveru bez njegovog izvršavanja. Dakle, još u ranim fazama razvoje se ovom metodom mogu pronaći greške, koristeći određene metode koje se uvrštavaju u statičku verifikaciju.

Međutim, kod dinamičke verifikacije je potpune obratno, tj. kood softvera mora biti pokrenut da bi se pronašle greške u softveru.

Iz svega toga se zaključuje da je statičko ispitivanje, za razliku od dinamičkog mnogo jeftinije, a i greške ispitivanja su lakše oučljivije i pogodnije za ispravljanje.

Ali ne kraju svega, za kvalitetno testiranje bi trebalo koristiti statiču i dinamičku verifikaciju u kombinaciji.

--------------------------------------------------------------------------------------------------------------------------------------

Već sam ranije rekao da se formalne metode baziraju na matematičkoj logici, te na taj način dobijamo potvrdu da se implementira sistem prema specifikacijama. Treba napomenuti da se fomralne metode ne mogu koristiti baš kroz čitavu fazu razvoja, a razlog tome je kompleksnost sistema, nedostatak alata ili iz nekih drugih razloga. Ali skoro uvijek se mogu koristiti u dijelu sistema koji su zaduženi za sigurnost kritičnih komponenti.

Jedna od prednosti formalnih metoda jeste bolji uvid u zahtjeve. Dakle tu ubrajamo otklanjanje nesporazuma, te shodno tome smanjuje se mogućnost grešaka, što naravno pridonosi pouzdanosti samog softvera.

Glavna mana formanih metoda jeste ta da je ne može baš svako i koristiti. Potrebno je određeno znanje i sposobnost da se izvrši sa što manjim stepenom greške pri izvođenju. Ona je skoro pa u potpunosti nerazumljiva korisnicima (mislim na klijente) i samom menadžmentu.

Dakle potrebni su posebno osposobljeni inžinjeri kako bi se formalne metode izvšile prema standardu i samim tim da se zagarantuje kvalitet softvera koji se razvija.

Ima više klasifikacija formalnih metoda, međutim dvije su glavne i to:

Da se primjeni željeno ponašanje softvera prema specifikaciji

Da potvrdi ponašanje implementacije prema specifikacijama (Veza između implementacije i specifikacije)

--------------------------------------------------------------------------------------------------------------------------------------

U kratkim crtama, dinamička verifikacija softvera nije ništa drugo nego primjenjivanje različitih testiranja nad sami softver.

Ta testiranja koja se primjenjuju u dinamičkoj verifikaciji su

1. jedinično testiranje

2. Integracijsko testiranje

3. Test prihvatljivosti u kojem se nalaze test funkcionalnosti (provjera ponašanja sistema prema specifikacijama kupca) i test nefunkcionalnosti (procjena da li je ponašanje testiranja softvera u skladu sa svojim specifikacijama)

--------------------------------------------------------------------------------------------------------------------------------------

Jedinično testiranje, poznatije kao unit testing, je od prethodno nabrojanih testiranja najjednostavnije testiranje. Sa njim provjeravamo ispravnost pojedinih jedinica bez ikakve veze sa drugim jedinicama. Zasniva se na tome da uzima jedan mali dio koda, konkretno jedna funkcija u klasi. Ako nam vraća onakav rezultat kako smo definisali sa poslanim odgovarajućim parametrima, test je u redu i možemo reći da funkcija ispravna. Ako je svaka pojedinačno testirana jedinaka ispravni, teško da će se naći greška u koodu prilikom kompleksnijih testiranja. Iako je vrlo jednostavno i prosto, ima veliki uticaj u pronalaženju grešaka u ranim fazama razvoja.

Dobra osobina je ta što nam kokretno govori gdje se greška u koodu nalazi, ali ima svoju i cijenu, a to je oduzimanja dosta vremena, jer ako postoji velik broj funkcija u klasi koje je potrebno testirati, za svaku funkcionalnost je potrebno napraviti zaseban test.

U nastavku će se prikazati primjenjeno jedinično testiranje na već postojećoj aplikaciji razvijena tokom posljenjeg semestra iz predmeta razvoj softvera I.

\*Gledati kood i objašnjavati\*

Dalje će biti prikazana greška koju je jedinični test prepoznao

\*Gledati kood i objašnjavati\*

--------------------------------------------------------------------------------------------------------------------------------------

Integracijsko testiranje je sljedeća vrsta testiranja u dinamičkoj verifikaciji i u jednu ruku se nadovezuje na jedinično testiranja, jer koristi sve jedinke našeg kooda i spaja ih u jednu cjelinu, te na taj način vrši testiranje kooda.

Spajanje, odnoso grupisanje programskih jedinki se može vršiti na više načina i to

inkrementalnom integracijom (postupno dodavanje programskih jedinica)

integracija top - down (od najvišeg modula prema nižim)

integracija bottom - up (od najnižih prema višim modulima)

sendvič integracija (kombinacija top - down i bottom - up integracija)

veliki prasak (testiranje svih pojedinačnih modula, te u jednom trenutku u jednom trenutku se sve jedinke integriraju u jednu cjelinu)

Prednost ovog testiranja jeste ta što razločiti moduli našeg kooda rade u jedinstvu, dakle ne oduzima toliko vremena za pripremu kooda testa kao jedinično testiranje, međutim jedna od glavnih mana ove vrste testiranja jeste ta što se veoma teško otkriva problematični modul, jer nije uvezano sa razvojnim okruženje, odnosno sa kompajlerom. Mi kada napišemo "neispravan" kood, kompajler nas neće upozoriti konkretno na tu krešku, jer sintaksički je ispravno, međutim osim sintaksičkih grešaka imamo i logičke greške, a to rješavamo upravo ovim testom.

U nastavku će se prikazati primjenjeno integracijsko testiranje na isti projekat kao kod jediničnog testiranja.

\*Gledati kood i objašnjavati\*

Dalje će biti prikazana greška koju je integracijski test prepoznao

\*Gledati kood i objašnjavati\*

--------------------------------------------------------------------------------------------------------------------------------------

Preporučuje se implementacija dinamičke verifikacije, odnosno testiranja razvijenog softvera, ako ništa barem izvršiti jedinično testiranje gdje nakon testiranja programskih jedinki možemo koliko toliko biti sigurni da kao cjelina naš softver neće zakazati.

Iz priloženoga se moglo vidjeti i zaključiti da su nam ovakve metode jako korisne, a nisu ni pretjerano komplikovane. Barem ne oni najjednostavniji dijelovi načini testiranja. Ovakve metode nam znatno pojednostavljuju pisanje i ispravljanje kooda projekta, te smatram da bi svaki programer trebao znati rukovati sa testovim.

Naravno, sama dinamička verifikacija nije dovoljna za potpuno verifikovanje softvera od grešaka. Zbog toga je je najbolje upotrijebiti u kombinaciji sa statičkom verifikacijom softvera, te na taj način svesti greške na što veći minimum.