***TESTIRANJE SOFTVERA***

***Osnovni pojmovi***

***Otkaz*** – failure, problem, issue, incident

Termin koji označava pojavu da se nešto desilo što se razlikuje od očekivanog ponašanja softvera. To je manifastacija. Odstupanje od zahteva.

***Defekt*** – fault, bug

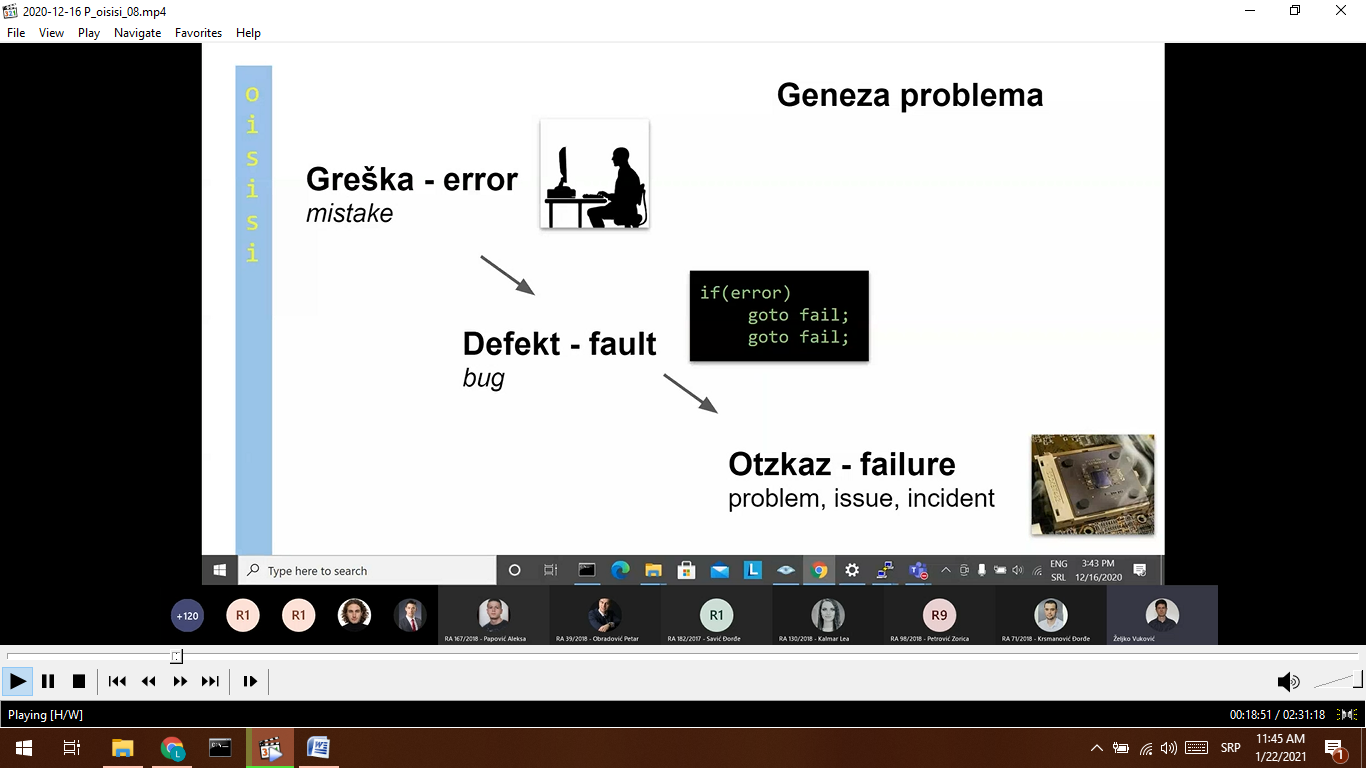
Greška u implementaciji zbog koje je došlo do otkaza ili nepostojanje implementacije.

Ne za manifestaciju nego za uzrok.

***Greška*** – error, mistake

Razlog zašto je došlo do defekta, razlog što smo mi pogrešili, to što je neko pogrešio je prouzrokovalo da postoji defekt, razlozi za to mogu biti raznoliki (neko nije znao kako nešto da uradi ili da nešto treba da uradi, nije umeo, nije pazio, nije razumeo šta tačno treba da radi, nije stigao,..)

Nečija greška u radu rezultira time da u kodu postoje problemi, ako dođe do izvršavanja tog koda, to će ćesto rezultirati otkazom i mi ćemo to primetiti. Ali, ne uvek, ***neće svi defekti rezultirati otkazom***. Neki kod se možda nikada neće izvršiti, možda će se izvršiti ali ćemo dobiti odstupanje, tj pogrešne rezultate a da pritom nemamo signalizaciju da je došlo do problema. Izvšavanje se nastavilo, ali nije u skladu sa očekivanim ponašanjem. I takve stvari su zapravo i najopasnije. Jer te rezultate posle neko može da koristi i mogu dalje da se gomilaju problemi.

Jedan od takvih ***primera*** je i deo koda iz apple-ove komponente koja implementira određenu funkcionalnost za bezbednost. Ono što se desilo je da je neko slučajno dva puta napisao liniju goto fail. Bez obzira da li je uslov ispunjen ili nije ovaj drugi se sigurno izvršava. I spletom okolnosti, posledica nije bila da niko nije pušten, što bi se odma primetilo, već je svako pušten, tj određeno stanje koje je problematično bude ignorisano. I to je godinama tako bilo i ta ranjivost je sigurno više puta bila iskorištena.

Najbolji problemi za nas su oni koji uvek rezultuju otkazom i za koje znamo tačan niz koraka, tačno stanje tačne uslove kada će problem da rezultuje otkazom, tj ***problemi koji su ponovljivi***. Izmedju ovih koje nikada ne vidimo i onih koji se uvek pojavljuju, imamo i probleme koji se javljaju ponekad, ponekada dobijemo grešku ponekada otkaz ali ne možemo da otkrijemo razlog jer se ne dešava uvek nego ponekad.

***VERIFIKACIJA I VALIDACIJA***

***Verifikacija*** - Provera da li se sistem ponaša u skladu sa zahtevima, da li na pravi način razvijamo proizvod

***Validacija*** - Provera da li sistem ispunjava potrebe korisnika, da li razvijamo pravi proizvod

To što nema grešaka i daje dobre rezultete ne znači da su ti rezultati nekom potrebni, ponekada možemo doći i u situaciju da je postojanje nekih funkcionalnosti neželjeno. Da je to što naš sistem daje neke informacije je nešto što korisnici ne žele da bude očigledno.

Testiranjem zelimo da utvrdimo dal i su zahtevi ispunjeni.

Da bi se testiranje moglo sprovoditi, uslovi za njega moraju biti jasno definisani, i uglavnom je to neka specifikacija sa kojom poredimo ponasanje sistema. Ako je specifikacija loša ili nepostoji biće nam teško da tesitiramo nekada i nemoguće. U takvim slučajevima koristi se exploratory teksting gde onaj ko testira počne da koristi softver, da puni sistem informacijama, pokreće neke funkcionalnosti. U tavim situacijama jedino što može da se nadje jeste neki sled okolnosti zbog kog dolazi do otkaza pri čemu taj otkaz mora da bude vidljiv da bi znali da je do otkaza došlo.

***TEST PROCES***

1. ***Priprema test podataka***

Da bi imali na čemu da isprobamo test objekt trebaju nam test podaci. Ti podaci i njihovo kreiranje može da bude ili ***nasumična*** ili ciljano da bude takvo da ***proveravamo*** neke ***krajnje slučajeve***, a može biti i ***kombinacija***. Ali za neka druga testiranja kao testiranje nefunkcionalnih zahteva, performansi, treba nam količina podataka i tada nam možda i nije bitno kakvi su ti podaci.

1. ***Planiranje izvršavanja***

Moramo ostaviti vremena za testiranje, jer ako ne uračunom, testiranja neće biti ili će doći kada više nemamo koristi od njega, nakon što je proizvod isporučen.

1. ***Dizajn i implementacija testova***

Ključna je faza dizajna, osmišljavanja testova, kako trebaju da izgledaju i kako najbolje da porverimo poklapanje implementiranog sistem sa zahtevima. Ne mozemo tesitirati sve, čak i za najjednostavnije, pa osmisljavamo testove koji pokrivaju što više mogućih slučajeva koji će biti takvi da je veća mogućnost da dovedu do problema ukoliko on postoji.

1. ***Izvršavanje testova***

Ono moze biti automatsko i ručno.

1. ***Analiza rezultata***
2. ***Dokumentovanje***

Dobro je da se dokumentuje svako izvrsavanje testa pa i ono gde nema problema. To može pred nekim biti način da pokažemo da smo preduzeli neke korake u proveri kvaliteta softvera, a nama može biti koristan podatak da utvrdimo u kom trenutku je nešto prestalo da radi.

***TEST SLUČAJ***

Jedan test case sadrži detaljan opis testa:

* Opis testa i njegovog cilja - šta pokušava da utvrdi i proveri, najčešće tekstualno
* Preduslove za izvršavanje - šta je neophodno da se pripremi u smislu okruženja da bi se taj test mogao izvršiti
* Neophodne test podatke koji ce biti korišćeni u toku izvršavanja
* Uključuje očikvane rezultate tj očekivano ponašanje (test oracle - mehanizam kojim se odredjuje da li je test pao ili prosao)

***TEST SCENARIO***

Ponekad nije dovoljan jedan test da bi se određena funkcionalnost proverila, jer je sam softver kompleksniji i podrazumeva niz koraka koje treba obaviti. Jedan način da to testiramo je jedan test slučaj koji je veoma kompleksan, dugačak i bio bi nezgodan i za dizaj, implementaciju i izvršavanje.

Umesto toga možemo svaki od tih koraka da testiramo posebnim test slučajevim pa da definišemo test scenario koji predstavlja neki skup testova koji zajedno proveravaju neku kompleksniju funkcionalnost. Ti testovi koji su deo jednog scenarija su jedni drugima preduslov.

***TEST SWEET***

Kolekcija testova koji međusobno i ne moraju da budu usko povezani ali koji su svi vezani za jedan softverski sistem ili jedan podsistem pa smo ih tako grupisali.

***SAVRSEN TEST***

Kao što ne možemo napisati savršen softver ne možemo sprovesti apsolutno sveobuhvatno testiranje i zato je neophodno da planiramo testiranje ne samo u smislu da ostavimo vremena i resursa za njega, već i da upravljamo detaljnosti i sveobuhvatnosti testiranja u skladu sa rizicima.

Posledica grešaka u softveru mogu biti različite u zavisnosti od oblasti primene. Zato je te rizike potrebno izmeriti i u skladu sa time i odrediti koliko će paznje tj resursa biti posvećeno testiranju.

Ukoliko softver specificiramo i implementiramo na striktno formalan način koristeći striktne matematičke formalizme onda možemo da koristimo metematiku da dokažemo da li je nešto konzistentno ili korektno, što se može automatizovati.

Problem je što se na ovaj način pokriva smao verifikacija, ali ne i validacija. Dokaz da će rezultati koje softver daje biti tačni ne znači da će oni biti nekom korisni. Takođe, pitanje je i koliko zaista kompleksan softver može da se razvija na ovakav način, šta je to za šta se može na formalan način specificirati izvršavanje softvera. I na kraju kada imamo tako zapisane zahteve koliko ce oni biti razumljivi onom za koga je softver namenjen. Na taj način smo napravili formalnu specifikaciju i implementaciju ali to ne rešava problem komunikacije sa korisnicima, klijentima u ranim fazama dok nemamo implementaciju, šta više mozemo reći da taj problem još više otežava.

***TROSKOVI TESTIRANJE***

* Ljudi - oni koji razvijaju ili neki drugi kojima je testiranje posao sam po sebi
* Vreme
* Novac
* Oprema za neke specifične primene koja ce se koristiti tokom testiranje, čak i u nekim najobičnijim softverima treba opet oprema neki računari,..

Neotkrivene greške takođe koštaju:

* Izgubljenog profita jer nekom nismo uspeli da isporučimo proizvod koji smo razvijali
* Neko može da nas tuži jer smo mu prozveli neku štetu
* Izgubljen ugled
* Izgubljeni korisnici
* Zdravlje i život ljudi – sve češće danas softver imamo u sklopu fizičkih sistema, softver upravlja vozilima, avionima, vozovima,..

Ali ukoliko se sve odradi kako treba, rizik za ljude može biti i manji nego kada svim tim upravljaju ljudi ali nezgodna stvar je što taj rizik i dalje postoji i prebačen je na programera. U takvim primenama testiranje mora ozbiljno da se shvati i testiranje je tada preovlađujuća delatnost.

***ANALIZA I DIZAJN TESTA***

***Proveriti osnove koje koristimo za testiranje.*** Da li je dovoljno jasna specifikacija da bi na osnovu nje mogli da kreiramo test.

***Dokumentovati probleme sa zahtevima.*** Isto kao da smo pronašli problem u implementaciji i ove probleme treba dokumentovati. To nam je i nešto što ima veću vrednost jer smo u nekoj ranijoj fazi otkrili da ćemo potencijalno imati probleme.

***Analiza test objekta.*** Treba utvrditi da li je on implementiran tako da ga je moguće testirati uopšte i da li je moguće da ga testiramo onim resursima kojima raspolažemo možda nam je nešto još dodatno potrebno za proveru i ako nam to nedostaje opet prijavljaujemo kao i sve ostale probleme.

***Osmisljavamo strategiju i sagledavamo rizike.*** Kako će se test sprovoditi, izgledati, kako da test utvrdi poklapanje ponašanja sa očekivanim,posmatramo koliko je sam deo sistema koji testiramo kritičan, koji su to delovi sistema koji za korisnika nosi veći rizik, veću vrednost koji su to zahtevi koji su korisniku bitni i onda te delove sistema i te zahteve testiramo detaljnije, njima dajemo veci prioritet. Dobro nam je da odredimo prioritet i komponenti i testova jer se možda možemo naći u sistuaciji da ne možemo sve testove da izvršimo da nemamo budžet za sve testove i onda izvršimo samo prirotetnije da bi utvrdili probleme u kritičnim delovima.

***Obezbediti sledljivost.*** Trebamo voditi računa i o tome koji test, šta testira, koji zahtev proverava, jer ako se zahtev naknadno promeni možda treba menjati i sve testove koji to testitraju.Dobo je da imamo povezane i delove implementacija sa testovima koji to proveravaju. Tada je laše pronađeni problem povezati sa testovima koji to proveravaju.

***Definisati test podatke.*** Dobro je da oni obuhvataju ne samo očekivane, već i neočekivane slučajeve ili akcije korisnika.

***IMPLEMENTACIJA I IZVRSVANJE TESTOVA***

***Proveriti kompletnost***. Da li imamo instalirane sve neophodne alate delove sistema, da li su stvari podešene kako treba (preduslovi testa).

***Smoke test.*** Prvo se testiraju osnovne funkcionalnosti jer ako ni one ne rade onda nema smisla testirati dalje.

***Dokumentovanje izvršavanja.*** Kada izvrsimo test neophodno je da iza toga ostane trag, da beležimo da smo ga pokrenuli i njegov rezultat. Veoma zgodno može biti neko logovanje koje za vreme izvršavanja testa ostavi sve potrebne informacije dokumentovane.

***Ponovljivi testovi.*** Sami testovi moraju biti takvi da se mogu izvrsiti vise puta i da ih mogu izvršiti razni ljudi. Pa tako i specifikacija testa, ako se radi o ručnom testiranju, ukoliko nije dovoljno tačno napisana, može doći do razilaženju u njenom razumevanju.

***Dokumentovanje otkaza.*** Ukoliko testom otkrijemo otkaz neophodno je da ga dokumentujemo i za to postoje specijalni alati, bug tracker-i.

***REGRESIONO TESTIRANJE***

Nakon otklanjanja problema treba ponovo izvršiti sve testove. Time što je popravljen ili dodat neki deo koda može da poremeti prethodno implementirane funkcionalnosti, zato je neophodno ponovo testirati. U nekim slučajevima moguće je napraviti i automatske testove, koji će se pokretati prilikom svakog build-ovanja projekta, gde se ova operacija neće završirti ukoliko neki od testova propada.

***KAKO ZNAMO DA TESTOVI RADE KAKO TREBA?***

Testovi testiraju sami sebe. Moguće je da će se problem u dizajnu ili implementaciji testa otkriti u toku samog testiranja što nam je dovoljno. Zgodno je da imamo na umu da problem može biti u testu. Ako imamo prijavljen problem i ako smo uradili analizu implementacije i ne nađemo uzrok problema možda je u pitanju test.

***PRINCIPI I PREPOURKE***

Testovi ne mogu da garantuju da greške ne postoje, uspešan test može samo da nađe probleme koji postoje.

Dobro je da se o testiranju misli i da se započne sprovođenje u ranijim fazama razvoja softvera. To je aktivnost koja tera na dodatno razmišljanje i o funkcionalnim i nefunkcionalnim aspektima softvera, i o samim zahtevima, stoga možemo doći i do bitnih zaključka, da su možda zahtevi problematični, i tada su posledica toga mnogo manje nego da smo ih uočili kasnije.

Savremene metodologije razvoja softvera se trude da omoguće da se što pre dođe do implementacije, do nekih rezultata. A manjkavost zahteva i promena zahteva da zahvaljujući metodologiji koja se primenjuje ne budu pogubne i da nas manje koštaju.

Postoji i metodologija *test driven development*, koja nalaže da se testovi pišu pre samo implementacije. Pišu se tako da padnu, ne prođu i onda implementiramo dok testovi ne pozelene. Tako smo obezbedili da imamo testove na kraju ali sa druge strane, smo koristili taj dizajn i implementaciju testova kao nešto čime ćemo dodatno analizirati i razmatrati zahteve i dok to radimo verovatno ćemo dobiti neke ideje o implementaciji. Jedan od izazova je da ne krenemo u implementaciju dok nemamo testove.

Empirijski je utvrđeno da su defekti često grupisani, njihova raspoređenost po kodu nije ravnomerna. Ako nađemo grešku na nekom delu sistema, isplati se detaljnije testirati taj deo, jer vrlo verovatno okolo ima još grešaka. Razlozi za to su razni, neke stvari su možda rađene pri kraju projekta pa se zbrzalo,..

Paradoks insekticida. To što smo jednom napisali jedan skup testova i što su oni prošli ne znači da možemo u budućem razvoju tog istog sistema da zaboravimo na testiranje, ako nastavimo razvoj i nastavimo da dodajemo stvari u implementaciju, nove greške će se javljati u novim delovima koda koji nisu pokriveni postojećim testovima, čak i ako nam to nije u prvi mah vidljivo.

***NIVOI TESTOVA***

***Unit testovi***

Proveravaju ispravnost jednog modula softvera, u objektno orjentisanim jedan modul se svodi na jednu klasu pa jedan unit test testira jednu klasu. Ove testove obično pišu ljudi koji pišu softver, uporedo sa impelemntacijom. Minimum koji unit test proverava su sve javne metode klase, a dobar unit test će pokrivati i sve krajnje slučajeve za moguće opsege ulaznih vrednosti i kombinacije parametara tih metoda (ako imamo metodu klase niz koj a treba da dobavi član sa nekog indeksa, onda testiramo i za slučaj dobavljanja prvog, poslednjeg člana niza i negde iz sredine i kada dobavljamo nepostojeći član, prosledimo negativnu vrednost, prosleđen nevalidni tip podataka.

***Mock, mockup***

Dobro osmišljeni objektno orjentisani programi su dekomponovani na niz klasa od kojih svaka ima neki svoj uzak problem kojim se bavi pa će ta jedna klasa biti povezana sa mnoštvom drugih klasa. Zato klasi koju testiramo trebaju neke druge komponente za rad. Umesto pravih komponenti možemo koristiti mock-ove ili mockup-ove. Oni su zamena za pravu komponentu koja neće biti njena puna implementacija, nego ćemo tu zamensku komponentu implementirati, tako da ima samo dovoljno da se poveže sa tom klasom koju testiramo i da u tim testovim vraća očekivane rezultate. Mocko-vi nam trebaju jer te klase sa kojima testirana klasa komunicira možda nisu još razvijene, možda tim drugim klasama trebaju neke stvari, resursi da bi radili, ali i da sve imamo i sve radi. Razlog tome je što unit testiranje testira jedan modul jedan unit, želimo da testiramo njega, ne želimo da test zavisi od toga da li su druge komponente validne i dobre. Želimo da imamo kontrolisane uslove za izvršavanje testa koji testira pojedinačni unit. To nekada u slučaju da imamo pravu implementaciju, zahteva dodatan napor da se naprave mock-ovi , sa druge strane dizaj samog softvera treba da bude takav da možemo da zamenimo neki deo nekim mockup-om. Time se povećava testability softvera, olakšavamo testiranje i činimo ga pouzdanijim tj. rezultiti postaju korisniji. Ako u unit koristimo prave komponente i imamo problem koji je lokalizovan u jednoj od tih komponenti propadace testvi za sve komponente koje sa njime saradjuju pa nećem oznati gde je problem.

***Integreation test***

Testovi koji proveravaju komunikaciju između komponenti, proveravaju jedan modul, ali čitav sistem je izgradjen iz mnoštvo takvih modula koji međusobno sarađuju. Oni proveravaju međusobnu saradnju određenih modula.

***Sistemski testovi***

Proveravaju kako čitav sistem funkcioniše nakon što se integriše. U ovoj vrsti testova obično nas ne interesuje kako je sistem organizovan nego proveravamo interakciju sistema sa okolinom, proveravaju sistem kao celinu.

***Acceptance testovi***

Testovi prihvatanja su krajnja provera da li sistem adekvatno zadovoljava potrebe i zahteve korisnika. U ovoj fazi testiranja imamo i validaciju i verifikaciju. Ovakvom vrstom testova se određuje da li će sistem biti prihvaćen na korišćenje, da li je adekvatan za potrebe, da li će obezbediti željena unapređenja u odnosu na ono što se trenutno koristi, da li će sistem ugroziti svoju okolini,..

***User acceptence test (UAT)***

Ove testove sprovode korisnici.

***Factory acceptance test (FAT)***

Poslednja provera kod nas, pre nego što se sistem isporuči. Pokušamo kod nas da dočaramo okolinu u kojoj će sistem živeti tamo gde će stvarno biti korišćen da ima što je moguće realnije uslove.

***Site acceptance test (SAT)***

Provera kod njih. Radi se nakon isporučivanja sistema, konfiguracije, kod stvarnih korisnika u stvarnom okruženju sa stvarnim uređajima. Poslednja provera pre nego što se stvarno počne korsititi.

***AKREDITACIJA***

Za određene primene se može raditi i akreditacija softvera. To je proces koji definitivno sprovodi neka treća strana, neko drugi proverava naš sistem, da li je on u skaldu sa nekim zahtevima, standardima, regulativom i ako prođe ta provera onda imamo zvaničnu potvrdu.

***PRISTUPI TESTIRANJU***

* ***Black box*** – Testiramo tako da onaj ko testira nema uvid u impelmentaciju i ona ga ne interesuje, vidi sistem kao celinu kao crnu kutiju, zna šta su ulazi i izlazi. Njegov zadatak je da za neke ulaze proverava da li dobija dobre izlaze.
* ***White box*** – Imamo na raspolaganju detalje implementacije, izvorni kod i možemo da se vodimo i time kada osmišljamo i sprovodimo testiranje.

***NAČINI TESTIRANJA***

* ***Statička analiza*** – Prolazimo kroz kod, da li čitanjem pa tumačimo ili nad izvornim kodom pustimo alate koji će da provere da li je kod u skladu sa preporukama, da li imamo u kodu mesta do kojih se nikada neće doći ni u jednoj grani izvršavnaja (što može dovesti do bezbednosnih problema). Podrazumeva analizu tih izvornih resursa bez izvršavanja.
* ***Dinamička analiza*** – Podrazumeva da se pokreće sistem i da se on izvršava, a tokom izvršavanja se prati ponašanje i utvrđuje da li je u skladu sa zahtevima.

***TEST PODACI I KORISNICI***

Često treba napuniti aplikaciju podacima da bi mogli da porverimo da li nesto radi.

* Nasumično generisani: Gdsagja Gadhago, od koristi nama kada testiramo, ali kada demonstriramo korisnicama oni loše reaguju na nasumično generisane podatke.
* Generički generisani: Petar Petrovic, Marko Markovic
* Anonimizirani smisleni: Dimitrije Popadić

***TESTIRANJE OD STRANE KORISNIKA***

Testiranje u realnoj primeni.

Dobro je obezbediti način da korisnici prijave probleme i treba obezbediti proces koji će time biti iniciran da bi ga rešili. Nakon rešavanja treba ponovo regresivno testirati da nismo nešto drugo time pokvarili.

Isto tako treba omogućiti da korisnici prijave i neke nove zahteve ideje,..

***TESTOVI U FUNKCIJI RAZVOJA***

Razlog zašto su testovi dobri, korisni i zašto ih ne treba gledati kao nešto što je nepotreban dodatni napor, je da nam mogu pomoći i u toku samog razvoja. Često način na koji se radi, je da se prvo implementira na najjednostavniji mogući način koji je sigurno razumljiv, razumljiva implementacija, u nju smo sigurni, i da su drugi ljudi u stanju da je razumeju. Zatim, imaćemo testove koji će pokriti tu implementaciju i ukoliko je implementacija dobra oni će prolaziti. Samo u slučaju da imamo potrebu, neke stvari možemo refaktorisati, reorganizovati arhitekturu, tražiti neki optimalan algoritam, i sl. Tada testove će nam testovi veoma lako reći da li smo napravili funkcionalno adekvatnu zamenu i moćićemo da koristimo te iste testove da proverimo i neke nefunkcionalne zahteve. Kada imamo testove lagodnije je raditi refaktorisanje jer ćemo brzo znati ako to ne radi.