UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

DIPLOMSKI RAD

Standardi u testiranju softvera

Profesor: **Student:**

prof dr Dragana Bečejski-Vujaklija Nadica Raičević 128/02

April, 2008

Sadržaj:

1. Uvod

Zar nisu pravila i mere ono što čini život sigurnim i lakšim, jednom rečju organizovanim?! Do koje mere je organizacija neophodna i u kojim oblastima i da li uopšte postoje sfere života u kojima ne postoje standardizovane vrednosti?!

Ovakva pitanja nastala su jako davno i bila nametnuta razvojem industrije i nauke, kao i tendencijom da se nađe „zajednički jezik" razvoja industrije, tehnologije i mnogih drugih grana, kako u internom(nacionalnom), tako i u eksternom(internacionalnom smislu). Otuda standardi.

Zar nije ono što razdvaja dobro i loše, zapravo kvalitet, bilo da se radi o čoveku ili proizvodu?! No, zanima nas proizvod...
Ako proizvodimo, kako da dostignemo željeni nivo kvaliteta, kako da taj nivo održimo i kako da budemo sigurni da će to biti ono što kupac traži?

Otuda standardi kvaliteta.

Postoji li način da razvijamo softver koji će biti kvalitetan, lak za održavanje i zadovoljiti zahteve tržišta?! Postoji li način da taj „način" bude jedinstven, ponovljiv i dokumentovan i da nam umanji troškove prekasno spoznatih nedostataka proizvoda?!

Otuda primena standarda u informacionim tehnologijama, a posebno onih koji se odnose na kvalitet softverskih proizvoda.

Negde između elemenata uspeha i svuda u pozadini procesa razvija se nova i sve popularnija disciplina. Ono što se proizvodi za druge, mora se probati i okusiti na sopstvenom primeru.

Otuda testiranje softvera.

Danas, kada je tržište nemilosrdno, ostaje samo jedan siguran način da se postignu željeni ciljevi i da se obezbedi bar približno kontinuitet uspeha u softverskoj industriji. Softverski proizvod se, kao najnežnija biljka, planira, razvija, prati i kontroliše, a na kraju posebno oblikuje, kako bi se isporučio kupcu i zadovoljio zahteve. Kao i obično, sve se koncentriše oko poslednje karike u lancu - kupca. Čini se da je pored svih napora, dokumentacije, pravila, procesa, pored sve organizacije i planova, najvažnija reč onih koji proizvod zaista koriste. Recept za uspeh je jako važan, a sadržaj recepta čine: standard, kvalitet, testiranje i razvoj softvera i kupac. Nekad i izmešanim redosledom.

2. Standardi

Standard *(eng. standard)*
svaka zakonom utvrđena mera, merilo; zakonska novčana stopa i sl.; nešto što važi kao uzor, obrazac, što je priznato kao klasično; standard života (eng. standard of life) prosečna mera zadovoljavanja ekonomskih potreba kod pojedinca, pojedinih društvenih klasa (apstraktno - naroda); zlato normalno, novčano zlato, zlato od 22 karata; standard obrazac trg. obrazac

Sandardni *(eng. standard)*
koji se odnosi na standard, koji odgovara standardu; istog tipa, tipičan

2.1. Kratka istorija standarda

Standardizacija je proces koji ima korene u drevnim civilizacijama Vavilona i ranog Egipta. Najraniji standardi su bili fizički standardi za težinu i mere.
*Neki su nastali u kraljevskim porodicama. Kralj Henri I je standardizovao merenja, tako što je uveo meru **ell(lakat)**, koji je bio ekvivalent za dužinu njegove ruke.*

Praistorija - *Jedan od najranijih primera standardizacije je kalendar. Drevne civilizacije, oslanjajući se na položaj sunca, meseca i zvezda, određivale su pravo vreme da sade i beru useve, da slave praznike i prate važne događaje.*
Pre više od 20000 godina, preci ledenog doba u Evropi pratili su dane crtajući linije u pećinama štapićima i kostima. Kasnije, kako se razvijala agrokultura i obrađivala zemlja, bili su potebni mnogo precizniji načini da se predvide sezonske promene. Sumeri su, u dolini Tigra i Eufrata, imali kalendar jako sličan ovom koji danas koristimo. Pre 5000 godina , sumerski zemljoradnik je koristio kalendar koji je godinu delio na 30 meseci. Svaki dan je bio podeljen na 12 sati, a sat na 30 minuta. Egipćani su prvi razvili 365- dnevni kalendar. Bazirali su merenje godine na izlasku zvezde Sirus svakih 365 dana i povezivali ga sa početkom uzgajanja u dolini Nila.

Neki standardi su bili ishod čovekove želje da harmonizuje aktivnosti uvođenjem važnih izmena u okruženju. Drugi su opet kreirani kao posledica odgovora na potrebe sve kompleksnijeg društva.

Kako se trgovina i razmena razvijala, pisani dokumenti postali su način da se izraze standardi vezani za proizvode i usluge (u oblasti agrokulture, brodogradnje, građevine i

oružja). U početku su predstavljali samo vid dogovora između snabdevača i korisnika. Kasnije su se koristili u transakcijama formirajući osnovu za modernu standardizaciju.

Industrijska revolucija - *Sa razvojem industrijske revolucije u XIX veku, povećana potreba za transportom dobara dovela je do pojave naprednih oblika transporta. Izum železnice je predstavljao brz, ekonomičan i efektivan način slanja proizvoda širom zemlje. Ovu prednost omogućila je standardizacija šina.*

Nakon industrijalizacije u XIX veku, odsustvo standarda izazivalo je haos i ugrožavanje javne sigurnosti. Brojni su primeri raznih neslaganja u električnim instalacijama koja su dovodila do spojeva, eksplozija i oštećenja.Ovakvi događaji su rezultirali standardizacijom, na opštu dobrobit svih.

XX vek - *Gradovi su doživeli veliku ekspanziju u XX veku. Kako su se gradovi širili i infrastruktura postajala složenija postalo je očito da će biti neophodan jedinstven skup nacionalnih standarda, kako bi se obezbedila opšta sigurnost stanovništva.*
Godine 1904, izbio je požar u skladištu John E. Hurst & Company Building u Baltimoru(SAD). Nakon što je zahvatio celokupnu strukturu, prelazio je sa zgrade na zgradu, dok se nije proširio na 80 blokova grada. Kako bi pomogla u gašenju požara, pristigla su pojačanja iz Njujorka, Filadelfije i Vašingtona, ali to nije uspelo. Creva za vodu nisu se mogla prevezati na hidrante u Baltimoru. Primorani su da bespomoćno gledaju kako se vatra širi i uništava oko 2500 zgrada i gori više od 30 sati.
Bilo je jasno da se mora razviti novi nacionalni standard, kako bi se sprečile slične situacije u budućnosti. [15]

Razvoj tehnologije pratio je razvoj standardizacije. Danas je ona na zavidnom nivou i bavi se i najsitnijim detaljima obezbeđujući red i preciznost. U proteklih 100 godina standardizacija se proširila sa proizvodnje na usluge.[16]

2.2. Pojam standarda

Šta su stanardi?

Objavljeni dokment koji je donešen konsenzusom i odobravanjem uvaženog tela, koji uspostavlja specifikacije i procedure kako bi osigurao da materijal, proizvod, metod ili usluga služi sopstvenoj svrsi I da je konzistentna sa namenom.

ISO/IEC voidič 2: 1996 definiše standard kao dokument, uspostavljen konsenzusom i slaganjem priznatog tela koji omogućava stalnu upotrebu pravila, vodiča ili karakteristika, za aktivnosti ili rezultate aktivnosti, sa ciljem da se dostigne optimalni stepen reda u datom kontekstu.
Prosto rečeno, standard je objavljeni dokument koji uspostavlja specifikacije i procedure, dizajniran da obezbedi da meterijal, proizvod, metod ili usluga služe svrsi i da su konzistentni u upotrebi.
Standardi rešavaju pitanja kompatibilnosti proizvoda, sigurnosti klijenata i zdravstvena pitanja. Standardi takođe pojednostavljuju razvoj proizvoda, raduciraju troškove i omogućavaju korisniku da poredi konkurentske proizvode. Standardi su osnovni gradivni elementi međunarodne razmene. Samo kroz korišćenje standarda se mogu postići zahtevi interkonektivnosti i interoperabilnosti, potvrđen kredibilitet novih proizvoda i novih tržišta i omogućena brza implementacija tehnologije.

Zašto su standardi važni?

Godišnji izveštaj američkog društva za testiranje i resurse (ASTM) iz 1991. godine daje pogodan odgovor na ovo pitanje: "*Standardi predstavljaju točkove komunikacije za proizvođače i korisnike. Oni služe kao zajednički jezik, definišu kvalitet i uspostavljaju sigurnosni kriterijum. Troškovi su manji kada proizvođači koriste standarde. Trening je pojednostavljen. Kupci prihvataju proizvode mnogo lakše kada mogu da ih procene na osnovu pravih vrednosti.*"

Kako nastaju standardi?

Standardi - objavljene specifikacije - su neprocenjivi svetski resurs, a proces koji vodi ka standardizaciji je jako zahtevan. Postoji više od pola miliona objavljenih standarda, koji su proizvod više od 1000 priznatih organizacija za razvoj standarda širom sveta. U suštini, svaka zemlja ima telo koje je zaduženo za nacionalne standarde. To su privatne ili javne organizacije ili kombinacije ova dva oblika. Objavljeni standardi ne obuhvataju brojne interne standarde, koji podupiru svaki uspešan poslovni proces.

Kriterijum za razvoj javno dostupnih standarda je da predstavljaju konsenzus zajednice tehničkih eksperata, naročito odabranih da donose odluke i doprinose procesu standardizacije.Velika kolekcija akumuliranog znanja i ekspertize je skupa. Postoji prateći ekonomski trošak njihovog doprinosa koji danas poprima velike razmere.[17]

U Australiji postoji oko 9000 članova tehničkog komiteta. Procenjeno je da njihov doprinos vredi 30 miliona dolara godišnje, kada se posmatra kao oportunitetni trošak. U Nemačkoj postoji oko 50 000 članova tehničkog komiteta, a u Egleskoj oko 31000. U Sjedinjenim Američkim Državama postoji preko 400 organizacija za razvoj standarda iz oblasti industrije, gde svaka od njih ima mnogo tehničkih komiteta, od kojih su neki po članstvu veći od nekih komiteta u svetu.

Procena broja ljudi koji učestvuju u procesu standardizacije širom sveta ukazuje na cifru od preko pola miliona ljudi. Koristeći australijski oportunitetni trošak kao parametar, realno je zaključiti da se u proseku 1.5 milion dolara investira na globalnom nivou svake godine za kreiranje i menadžment standarda.

Rezime

U početku su standardi bili jedinstveni dokumenti i deo dogovora između dobavljača i korisnika. Kasnije se koncept standarda proširio, kako bi se isti mogli koristiti u mnogim transakcijama. Ova portabilnost proizvela je skup kriterijuma i osnova je moderne standardizacije.
Nakon ubrzane industrijalizacije ranog XIX veka, sveopšte odsustvo nacionalne standardizacije izazvalo je veliku neefikasnost. Nedostatak pravila bio je veliki trošak. Vrednost standardizacije u obliku specifikacija, materijala, testiranja je prepoznata kao nacionalni prioritet tek krajem XIX veka. Do kraja stoleća standardizacija je doživela procvat i nastavljala je sa razvojem u savremenom društvu. Proširila se daleko izvan prvobitnih okvira i uključila sigurnost kupaca, zdravlje u cilju poboljšanja kvaliteta i komfora svakodnevnog života.
Standardi predstavljaju alat za organizaciju tehničkog sveta i merila koja se koriste za uspostavljanje norme za menadžment procedure. Oni podržavaju očekivanja kupaca da će proizvodi koje koriste biti sigurni, pouzdani i da će odgovarati svrsi. Dakle, standardi su postali sastavna komponenta ekonomskog, socijalnog i pravnog sistema u toj meri da se često podrazumevaju i predstavljaju nezaobilazni deo modernog društva.[18]

2.3. Standardizacija

Standardizacija *(eng. standardization)*
racionalizacija proizvodnje putem smanjivanja većeg oblika proizvodnje na manji broj tipičnih obrazaca (standarda istog kvaliteta, oblika, veličine, težine itd.).

Standardizacija je proces koji uključuje planiranje, razvoj i primenu dokumentacije vezane za standarde. To je proces spajanja naučnog istraživanja sa iskustvom primene radi utvrđivanja preciznih, optimalnih tehničkih zahteva jednog aspekta tehnologije. Ishod ovog spajanja je autoritativni dokument koji se naziva "standard". Standard je dokument koji uspostavlja uniformni inženjering ili tehničke specifikacije, kriterijume, metode, procese ili prakse. Neki standardi su obavezni, dok su drugi voluntarni. Neki standardi su de facto, označavaju norme ili zahteve koji imaju neformalni, ali dominantni status. Neki standardi su de jure, označavaju formalno-pravne zahteve. Tela za standardizaciju npr. Internacionalna organizacija za standardizaciju(ISO) ili Američki nacionalni institut za standardizaciju(ANSI) su nezavisne od proizvođača dobara za koje objavljuju standarde. Cilj standardizacije može biti i podsticanje nezavisnosti dobavljača, kompatibilnosti, interoperabilnosti, sigurnosti, pouzdanosti ili kvaliteta. U društvenim naukama, uključujući i ekonomiju, ideja standardizacije bliska je pojmu "rešenja koordinacionog problema", situaciji u kojoj sve strane mogu da realizuju dobit, ali samo ako zajednički donose odluke. Standardizacija je proces za odabir boljih opcija i potvrdu istih u obliku standarda. Ovo stanovište uključuje slučaj "spontane standardizacije procesa" kako bi se kreirali de facto standardi.

Dakle, standardi mogu biti:

- ***De facto*** standardi koji su praćeni neformalnom konvencijom ili dominantnim korišćenjem.
- ***De jure*** standardi koji su deo pravno kreiranih ugovora, zakona ili regulativa.
- ***Voluntarni*** standardi koji su objavljeni i raspoloživi za upotrebu.

Kada bi koristili vojničku terminologiju, pod pojmom standardizacija podrazumeva se: Razvoj i implementacija koncepata, doktrina, procedura i dizajna za postizanje i održavanje zahtevanih nivoa kompatibilnosti, razmene ili jedinstva u operativnom, proceduralnom, materijalnom, tehničkom i administrativnom smislu, kako bi se dostigla interoperabilnost.

Standardizacija je poznata kao neophodna disciplina za sve tržišne učesnike koji teže da budu kompetitivni. Kompanije u današnje vreme integrušu standardizaciju i poslovno planiranje u glavni tehnički i komercijalni element.[17]

3. Standardi u informacionim tehnologijama

3.1. Primena standarda u informacionim tehnologijama

Standardizacija u informacionim tehnologijama doprinosi efikasnijem uspostavljanju informacionih funkcija, njihovoj većoj stabinosti i lakšoj tranziciji. Primena međunarodnih, nacionalnih i internih standarda u procesu razvoja softverskih proizvoda stvara uslove za razvoj efikasnog, ekonomičnog, pouzdanog i sigurnog softverskog proizvoda. Standardizacijom procesa razvoja softvera, njegovim planiranjem, kvantifikovanjem i praćenjem, dokumentovanjem i neprekidnim poboljšanjem i unapređenjem stvaraju se preduslovi za realizaciju softverskih proizvoda definisanog kvaliteta. Dobro dokumentovani sistem, u skladu sa standardima, je lako zamenjiv, prenosiv sa jedne softverske i hardverske platforme na drugu i štiti investiciju.

Da bi se razvio kvalitetan informacioni sistem neophodno je da se njegov razvoj zasniva na usvojenim standardima(međunarodnim, nacionalnim, internim) i da se vrše mnogobrojna vrednovanja tokom njegovog životnog ciklusa. Vrednovanja uključuju: vrednovanje korišćenih softverskih proizvoda, međuproizvoda u vakoj fazi životnog ciklusa i, krajnjih proizvoda tj. instaliranog softvera i dokumentacije.
Vrednovanje i standardizacija alata koji se koriste u procesu razvoja informacionih sistema stvaraju mogućnost da kvalitet samog procesa, kao i krajnjih proizvoda, bude na željenom i očekivanom nivou. Posmatrano sa stanovišta složenih informacionih sistema, u čijem razvoju i implementaciji učestvuje više organizacija, primena standarda, ne samo da obezbeđuje odgovarajući kvalitet krajnjeg proizvoda i procesa razvoja, nego stvara mogućnosti za razmenu projekata između pojedinih organizacija, olakšava obuku korisnika i stvara uslove za zajednički rad na projektima predstavnika različitih organizacija.
Ekspanziju razvoja softvera, u različitim oblastima, pratio je i razvoj standarda, procedura, metoda i alata za razvoj i upravljanje softverom. Raznovrsnost je stvorila moguće poteškoće u upravljanju softverom, posebno kada se radi o softveru koji je vezan za proizvode ili usluge. Ova pojava uslovila je potrebu da se za softversku disciplinu definiše zajednički okvir koji bi poslužio svima koji se bave softverom da „govore istim jezikom“ u razvoju, projektovanju i upravljanju softverom u njihovim okruženjima. Standard je tako projektovan da se može prilagoditi potrebi organizacije, projekta ili specifičnoj primeni. Može se primeniti u slučajevima kada je softver samostalan entitet ili sastavni deo složenog sistema.[9]

Danas se standardi primenjuju u mnogim granama.
Informacione tehnologije predstavljaju ključ za reformu zdravstva. Elektronski zdravstveni izveštaji štede vreme i novac, umanjuju verovatnoću medicinske greške, a kada ih je moguće deliti elektronskim putem, tada utiču na svaki korak u procesu lečenja. Deljenje informacija zahteva interoperabilnost. Postizanje interoperabilnosti znači utvrđivanje standarda, kako bi jedan sistem mogao da govori sa drugim, razmenjujući podatke precizno, efikasno i sigurno.
Zdravstveni IT standardi omogućavaju da službe zdravstva imaju brz, siguran pristup tačnim izveštajima svakog pacijenta.[20]

Herbalna medicina je grana u kojoj se standardizacija odnosi na obezbeđenje procesa proizvodnje materijala koji sadrži određenu koncentraciju specifičnog indikatora koji se u njemu sadrži.
U statistici, standardizacija se odnosi na konverziju u standardne rezultate. U testnoj teoriji, standardizacija označava merenja sprovedena pod preciznim, specificiranim i ponovljivim uslovima.
Sa stanovišta nove institucionalne ekonomije, standardizacioni proces počinje sa socijalnim problemom poznatim kao već pomenuta „koordinaciona dilema“ i realizacija sveopštih ciljeva.

Naravno, kako se informacione tehnologije ubrzano razvijaju u svakom pogledu i ispunjavaju sve sfere života, tako i standardi postaju prateći element koji uvodi red i omogućava kontinuitet napretka.

3.2. Tvorci informacionih standarda

ISO (Međunarodna organizacija za standardizaciju) i IEC (Međunarodna elektrotehnička komisija) zajedno formiraju sistem za opštepriznatu standardizaciju kao celinu. Nacionalne institucije, koje su članice ISO i IEC, učestvuju u razvoju međunarodnih standarda kroz rad u tehničkim komitetima koje je ustanovila odgovarajuća organizacija radi obrade posebnih oblasti tehničkih aktivnosti. Tehnički komiteti ISO i IEC sarađuju na poljima od obostranog interesa. Međunarodne organizacije, vladine i nevladine, koje su u vezi sa ISO i IEC, takođe učestvuju u tom radu.
Međunarodni standardi se objavljuju prema pravilima koja su data u ISO/IEC uputstvima.

U području informacione tehnologije ISO i IEC su ustanovili združeni tehnički komitet, ISO/IEC JTC 1. Nacrti međunarodnih standarda koje je usvojio združeni tehnički komitet šalju se svim nacionalnim telima na saglasnost pre njihovog usvajanja kao međunarodnih standarda. Usvajaju se prema postupku po kome standard mora da usvoji najmanje 75 % članica.[13]

Postoje dva osnovna tipa standarda:

1. Standardi proizvoda(određuju karakteristike i funkcionalne zahteve proizvoda)
2. Standardi procesa(određuju način na koji proizvodi treba da budu razvijeni)

Institucije u Srbiji

Zavod za standardizaciju je nacionalna organizacija za standardizaciju. Članica je međunarodnih organizacija za donošenje standarda ISO i IEC. Na osnovu članstva u tim organizacijama, Zavod za standardizaciju, dobija standarde i ostala dokumenta tehničkih komiteta.

Komisije Zavoda za standardizaciju prate rad međunarodnih tela za donošenje standarda u odgovarajućim oblastima (npr. komisija KSI 1/07 prati rad tehničkog komiteta ISO/IEC JTC1 SC 07, Information technology - Software and system engineering).
U skladu sa mogućnostima i potrebama, komisije Zavoda za standardizaciju učestvuju u radu međunarodnih tehničkih komiteta, prisustvuju njihovim zasedanjima i po potrebi (u zavisnosti koji status imaju: posmatrač ili aktivni član) mogu glasati ili stavljaju primedbe na ponuđene tekstove nacrta međunarodnih standarda.[23]

Institut za standardizaciju Srbije je vladina organizacija. Pored razvoja standarda,organizacija se bavi aktivnostima sertifikacije i nudi tehničku podršku na zahtev korisnika.

Standardizacija je prvi put bila formalno, institucionalno organizovana 1946. godine. Vlada je usvojila Dekret o standardizaciji, na osnovu koga je organizovana Komisija za standardizaciju čija odgovornost je bila da razvija standarde. Razvoj je kasnije uneo transformacije u različite institucionalne forme i omogućila oficijelni status vladinim organizacijama i Institutu za standardizaciju. Aktivnosti Instituta imaju za cilj da promovišu standardizaciju i kvalitet i da doprinesu poboljšanju proizvoda i usluga. Odgovornost Instituta se reguliše od strane Zakona o standardizaciji koji je stupio na snagu 2006. godine. Od 1996. godine standardi se razvijaju isključivo kao voluntarni standardi.
Institut predstavlja Srbiju među internacionalnim i evropskim organizacijama za standardizaciju(ISO, IEC, CEN i CENELEC) i u sistemima/šemama za sertifikaciju (IECEE, IECEx i IECQ).[22]

3.3. Primena standarda na kvalitet softvera

Stanje u softverskoj tehnologiji još ne obezbeđuje dovoljno dobru i široko prihvaćenu šemu za ocenjivanje kvaliteta softverskih proizvoda. Od 1976.godine urađeno je mnogo od strane pojedinaca na definisanju osnove za kvalitet softvera. Usvojeni su i prošireni mnogi modeli (McCall-a, Boehm-a, US Air Force i drugi).
Duže vreme pouzdanost je bila jedini način za merenje kvaliteta. Vremenom su kroz razne studije predloženi i drugi modeli. Iako su studije bile korisne, stvorile su zabunu, zbog toga što su ponuđeni mnogi aspekti kvaliteta. Dakle, postojala je potreba za jednim modelom standarda.

Iz tog razloga je ISO/IEC JTC1 počeo da radi na usaglašavanju i ohrabruje opšte prihvaćenu standardizaciju.
Prva razmatranja potiču iz 1978. godine, a 1985.godine počeo je razvoj standarda ISO/IEC 9126. Predloženi modeli prvenstveno uvode svojstva softvera koja zavise od aspekata primene ili implementacije, radi opisivanja kvaliteta softvera.

Prvi korak ISO tehničkog komiteta u sistematskom uređivanju svojstava je propao zbog nedostatka definicija. Eksperti su različito interpretirali termine. Sve pominjane strukture nisu imale zajednički osnovu. Na kraju je odlučeno da je najbolji način za postavljanje međunarodnog standarda postavljanje skupa karakteristika koje se zasnivaju na definiciji kvaliteta.[12]
Konačno, nastali su i prvi standardi koji se odnose na kvalitet proizvoda i kvalitet softvera: ISO 9000, ISO/IEC 9126 i mnogi drugi prateći standardi.

4. Kvalitet softvera

Kvalitet *(lat. qualitas)*
Kakvoća, svojstvo, osobina; vrlina, vrednost, dobrota, dobra osobina

4.1. Pojam kvaliteta softvera

Kvalitet predstavlja sposobnost da se proizvede softver koji zadovoljava ili nadmašuje postavljene zahteve(prema definisanim merljivim kriterijumima) i koji je proizveden definisanim procesom.Ovaj proces ne svodi se samo na zadovoljenje definisanih zahteva, već se u okviru njega moraju definisati mere i kriterijumi koji usmeravaju proces postizanja kvaliteta. Potrebno je usvojiti pravila za jedan ponovljiv i upravljiv proces čiji proizvodi će dostizati određeni nivo kaliteta.
Jedna od definicija koja se pominje u softverskoj literaturi je i definicija sa stanovišta potrošača. Potrošač definiše kvalitet kao meru zadovoljavanja potreba kupca od strane proizvoda ili usluge. Drugom rečju, upotrebnim kvalitetom.

Najćešće zablude o kvalitetu:

1. ***Kvalitet može biti naknadno dodat i" utestiran" – kvalitet mora biti opisan i ugrađen u proces stvaranja proizvoda.***
2. ***Kvalitet dolazi sam od sebe – kalitet ne nastaje tek tako. Proces razvoja mora se definisati, sprovoditi i kontrolisati kako bi se dostigao određeni nivo kvaliteta.***
3. ***Kvalitet je jednodimenzionalna karakteristika i svakom znači isto- kvalitet ima više dimenzija od kojih su najvažnije: funkcionalnost, pouzdanost, upotrebljivost, efikasnost, stepen podrške. Svaku od dimenzija prati odgovarajući tip testiranja softvera.***

4.2. Standardi kvaliteta

Dat je pregled nekih od vodećih standarda kvaliteta softvera.

4.2.1. ISO9000

Ovaj standard sadrži pet dokumenata razvijenih 1987.godine od strane Međunarodne organizacije za standarde (International Standards Organization). Standardi ISO 9000 su

generalno povezani sa neinformacionim procesom proizvodnje, ali organizacije koje se bave razvojem softvera mogu imati koristi od njih. Svi standardi ISO 9000 su vodiči i interpretacije, jer im nedostaju jasno definisana i stroga pravila. ISO sertifikacija postaje sve važnija za proizvođače hardvera u Evropi i SAD-u, a sertifikati će predstavljati obavezu za dobavljače softvera.

Standardi ISO9000 predstavljaju konačan skup standarda kvaliteta, ali takođe i standarde kvaliteta kao deo programa upravljanja kvalitetom (TQM – Total quality management). Ovaj standard je jako razumljiv i definiše sve elemente kvaliteta u cilju pružanja podrške dobavljaču prilikom dizajna proizvoda i dostizanja određenog nivoa kvaliteta. ISO9002 pokriva kontrolu dizajna dobavljača i razvojne aktivnosti. ISO9003 predstavlja dobavljačevu sposobnost da otkrije i kontroliše nekonformitet proizvoda prilikom ispitivanja i testiranja. ISO9004 opisuje standarde kvaliteta povezane sa ISO9001, ISO9002 i ISO9003 i obezbeđuje listu kvaliteta proizvoda.

Međunarodno	SAD	Evropa	U.K.
ISO9000	ANSI/ASQA	EN29000	BS5750 (deo 0.1)
ISO9001	ANSI/ASQC	EN29001	BS5750 (deo 1)
ISO9002	ANSI/ASQC	EN29002	BS5750 (deo 2)
ISO9003	ANSI/ASQC	EN29003	BS5750 (deo 3)
ISO9004	ANSI/ASQC	EN29004	BS5750 (deo 4)

Tabela 1. Odgovarajući ISO standardi kvaliteta

Model zrelosti sposobnosti - CMM (Capability Maturity Model)

Institut za softverski inženjering(SEI) krirao je model za procenu zrelosti softverskih procesa organizacije i identifikaciju ključnih praksi koje su neophodne za povećanje zrelosti procesa. Organizacije unapređuju karakteristike sopstvenih softverskih procesa i prolaze kroz različite faze zrelosti. Model opisuje principe i prakse čija je baza zrelost softverskog procesa i ima za cilj da pomogne u organizaciji softvera. Sadrži pet nivoa zrelosti:

1. ***Početni***: Softverski proces se karakteriše kao ad hoc, ponekad i kao haotičan proces. Samo neki procesi su definisani, a uspeh zavisi od individualnih napora i snage.
2. ***Iterativni***: Osnovno upravljanje procesima projekta obuhvata tri stavke: troškove, raspored i funkcionalnosti. Nezaobilazno je ponavljanje ranijih uspešnih praksi na sličnim projektima.
3. ***Definisanost***: Softverski proces je dokumentovan, standardizovan i integrisan u standardni softverski proces organizacije, u pogledu menadžmenta kao i u pogledu inženjerskih aktivnosti. Svi projekti su odobrene, dizajnirane verzije standardnog softverskog procesa organizacije za razvoj i održavanje softvera.

4. ***Upravljački***: Detalji merenja softverskog procesa i kvaliteta proizvoda se skupljaju. Softverski proces i proizvodi se kontrolišu.
5. ***Optimizirajući***: Kontinuirani proces poboljšavanja je omogućen kvantitivnom povratnom vezom iz procesa i lansiranjem inovativnih ideja i tehnologija.

Model zrelosti sposobnosti: ljudski resursi - PCMM(The People Capability Maturity Model)

Ovo je model koji pomaže organizaciji da uspešno prepozna kritične ljudske resurse. Zasnovan je na bazi najboljih tekućih praksi, iz oblasti kao što su ljudski resursi, upravljanje znanjem i razvoj organizacije. PCMM vodi organizaciju ka unapređenju procesa za upravljanje razvojem radnih snaga. Model pomaže organizaciji da definiše zrelost praksi radne snage, sprovede program kontinualnog razvoja radne snage, definiše skup prioriteta za poboljšanje akcija, integriše razvoj radne snage sa unapređenjem procesa i uspostavi kulturu uspeha. Od nastanka modela 1995. godine ova praksa se koristi u malim i velikim organizacijama širom sveta.
PCMM model se sastoji od pet nivoa zrelosti koji predstavljaju sukcesivne podloge za kontinuirano unapređenje individualnih kompetencija, razvoj timova, motivaciju za poboljšanje performansi i oblikovanje radne snage u organizaciji tako da odgovori zahtevima budućih poslovnih planova.

CMMI

Paket proizvoda CMMI omogućava najsvežije i najbolje prakse za razvoj i održavanje proizvoda i usluga. Modeli ovog tipa sadrže najbolje prakse CMM modela za softver, Systems Engeneering Capability Model-a(SECM) i Integrated Product Development Capability Maturity Model-a (IPD - CMM).[10]

4.2.2. ISO 9126

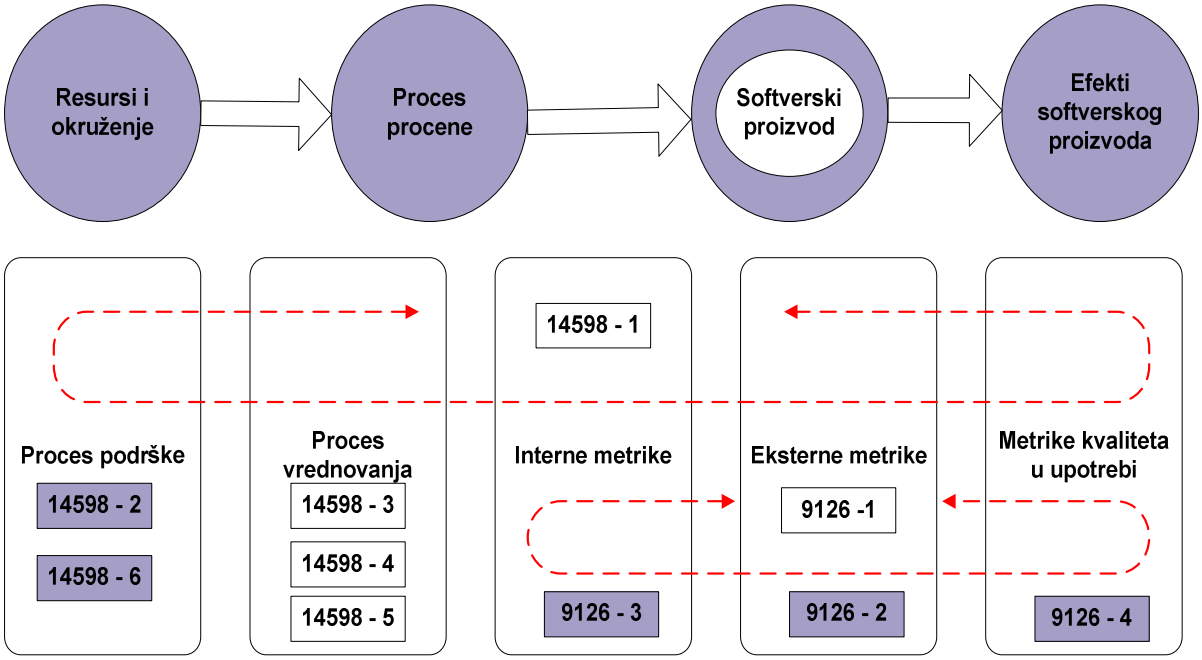
Šta je ISO 9126?

ISO 9126 je međunarodni standard koji se odnosi na kvalitet softverskih proizvoda. Standard nosi naziv: Softverski inženjering – Kvalitet proizvoda i sastoji se iz četri dela: modela kvaliteta, eksterne i interne metrike i metrike kvaliteta u upotrebi. Standard određuje odgovarajuće karakteristike kvaliteta uzimajću u obzir namenu softverskih proizvoda. Određuje šest karakteristika kvaliteta i opisuje proces vrednovanja modela softverskog proizvoda. Ovaj standard, prvobitno ISO 9126:1991, podeljen je na dva standarda ISO/IEC 9126 (Kvalitet softverskog proizvoda) i ISO/IEC 14598 (Vrednovanje softverskog proizvoda).

ISO/IEC 9126 (Kvalitet softverskog proizvoda)

Prvi deo standarda ISO 9126 odnosi se na Model kvaliteta. Standard opisuje dva tipa kvaliteta:

1. ***Interni i eksterni kvalitet*** (šest karakteristika eksternog i internog kvaliteta sa podkarakteristikama)
2. ***Kvalitet u upotrebi*** (četri karakteristike kvaliteta u upotrebi - kombinacija efekata šest karakteristika kvaliteta softverskog proizvoda)



Slika 1.Veza između standarda ISO/IEC 9126 i ISO/IEC 14598

Neophodno je definisati neke od osnovnih pojmova ovog standarda.

Potrebe korisnika za kvalitetom: *zahtevi za kvalitetom koji se definišu pomoću metrika kvaliteta u upotrebi, eksternih i internih metrika. Zahtevi se koriste kao kriterijumi vrednovanja proizvoda.*

Zahtevi za interni kvalitet*: određuju nivo kvaliteta sa internog aspekta proizvoda. Koriste se za specificiranje karakteristika međuproizvoda. Uključuju statičke, dinamičke modele, druge dokumente ili izvorni kod. Koriste se za validaciju u različitim fazama projektovanja proizvoda.*

Zahtevi za eksterni kvalitet*: zahtevani nivo kvaliteta sa eksternog aspekta. Zahtevi proističu iz zahteva korisnika za kvalitetom i zahteva za upotrebnim kvalitetom.*

Interni kvalitet*: zbir karakteristika softverskog proizvoda sa internog aspekta. Interni kvalitet softverskog proizvoda se meri i vrednuje na osnovu zahteva internog kvaliteta. Detalji kvaliteta mogu se poboljšati tokom implementacije koda, preispitivanja i ispitivanja, ali osnovna postavka kvaliteta softverskog proizvoda predstavljena internim kvalitetom ostaje nepromenjena, osim u slučaju ponovnog projektovanja.*

Procenjeni (ili prognozirani) eksterni kvalitet*: kvalitet koji se procenjuje ili prognozira za kraj svake faze projektovanja softvera za svaku karakteristiku kvaliteta na osnovu poznavanja internog kvaliteta.*

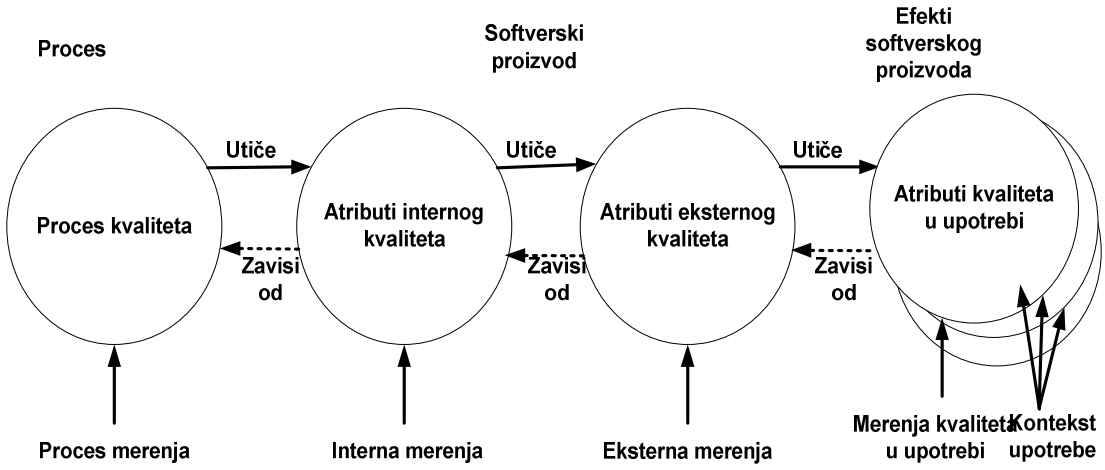
Eksterni kvalitet*: zbir karakteristika softverskog proizvoda sa eksternog aspekta. Predstavlja kvalitet izvršavanja softvera, koji se obično meri i vrednuje tokom testiranja u simuliranom okruženju sa simuliranim podacima korišćenjem eksternih metrika. Tokom testiranja, treba da se otkrije i eliminiše većina grešaka. Ipak, neke greške mogu postojati nakon testiranja. Pošto je teško ispraviti arhitekturu softvera ili osnovne postavke razvoja softvera, osnove razvoja se ne menjaju tokom testiranja.*

Procenjeni (ili prognozirani) kvalitet u upotrebi*: kvalitet koji se procenjuje ili prognozira za kraj svake faze projektovanja softverskog proizvoda za svaku karakteristiku kvaliteta u upotrebi na osnovu poznavanja internog i eksternog kvaliteta.*

Kvalitet u upotrebi: *kvalitet softverskog proizvoda sa korisničkog aspekta kada se koristi u specifičnom okruženju i u kontekstu specifične upotrebe. Meri stepen do kog korisnici mogu postići svoje ciljeve u određenom okruženju, a ne meri osobine samog softvera.*

Kako se meri kvalitet?

Kada se govori o kvalitetu i zadovoljenju zahteva za kvalitetom, tada se ističe pozicija kupca ili naručioca tj. korisnika kome se proizvod distribuira. Korisnički zahtevi za kvalitetom odnose se na kvalitet proizvoda u upotrebi. U toku životnog ciklusa proizvoda vrši se i vrednovanje softverskog proizvoda, kako bi se zadovoljenje zahteva za kvalitetom ugradilo u proces razvoja softvera. Kvalitet se procenjuje merenjem intenih(obično statičko merenje međuproizvoda) i eksternih(merenje ponašanja koda kada se izvršava) atributa ili merenjem atributa kvaliteta u upotrebi. Osnovu za merenje kvaliteta pruža ovaj model kvaliteta.



Slika 2. Kvalitet u životnom ciklusu

Povezanost procesa kvaliteta i kvaliteta u upotrebi je jasna. Proces kvaliteta doprinosi kvalitetu proizvoda, a kvalitet proizvoda upotrebnom kvalitetu. Moguća je i povratna sprega od proizvoda u upotrebi, ka kvalitetu proizvoda tj. procesu kvaliteta. Interni atributi utiču na eksterne atribute, a eksterni vrše uticaj na kvalitet u upotrebi. Može se zaključiti sledeće:

Zahtevi koji se odnose na kvalitet softverskog proizvoda uključuju kriterijume ocenjivanja za interni kvalitet, eksterni kvalitet i kvalitet u upotrebi radi zadovoljenja potreba projektanata, onih koji održavaju proizvod, naručilaca i korisnika. (ISO/IEC 14598-1:1999, tačka 8.)

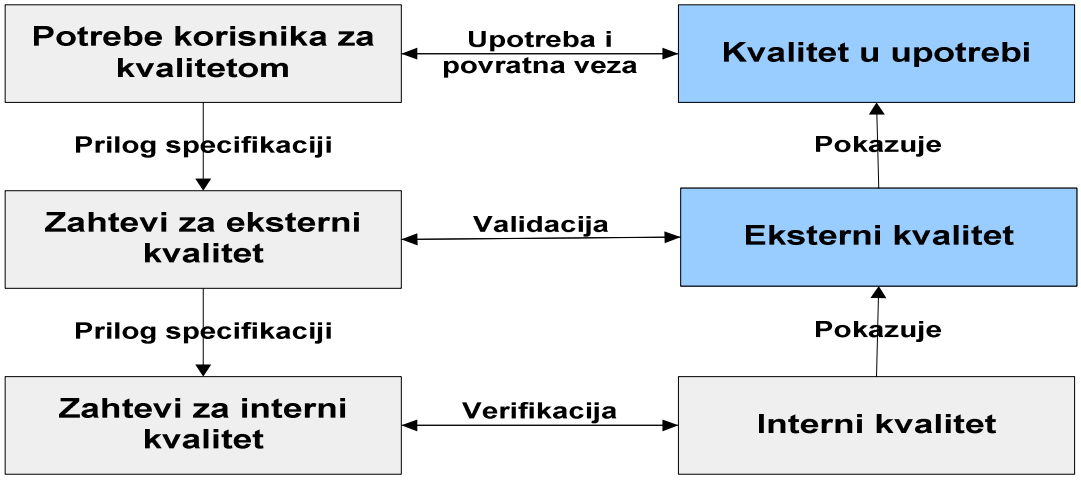
Životni ciklus proizvoda ima određenu dinamiku, pa se tako menjaju i interne i eksterne karakteristike i ovo treba imati u vidu kada se meri kvalitet proizvoda. Jasno je da se kvalitet konačnog proizvoda meri preko eksternih atributa i upotebne vrednosti proizvoda, a da se npr. kvalitet međuproizvoda ogleda u internim karakteristikama. Standard ISO 8402 definiše kvalitet kao mogućnost zadovoljenja iskazanih i podrazumevanih potreba. Iskazane potrebe korisnika nisu isto što i podrazumevane, najčešće iz nekoliko razloga:

- Nesvesnosti korisnika o stvarnim potrebama.
- Promenjivosti potreba nakon iskazivanja istih.
- Različitosti okruženja različitih korisnika.
- Nemogućnosti da se konsultuju svi tipovi korisnika.

Drugi deo procesa merenja kvaliteta odnosi se i na upotrebu proizvoda i korisnost u upotrebi. Zahtevi kvaliteta koje proizvod treba da ispuni podležu podeli na:

- nezadovoljavajući i zadovoljavajući ili
- nadmašeni, ostvareni, minimalno prihvatljivi i neprihvatljivi zahtevi.

Naravno, postoje različiti aspekti kvaliteta i različite metrike vezane za faze životnog ciklusa proizvoda.



Slika 3. Kvalitet u životnom ciklusu softvera

Kako se kvalitet vrednuje?

Vrednovanje elemenata može se vršiti direktnim merenjem ili merenjem posledica. Definisani model kvaliteta se koristi za vrednovanje kvaliteta softverskog proizvoda. Ovaj model se koristi u fazi postavljanja ciljeva kvaliteta za proizvod i međuproizvode. Vrši se hijerarhijska dekompozicija softverskog proizvoda koja se oblikuje u listu parametara koje su u vezi sa kvalitetom. Naravno, nije moguće merenje svih internih i eksternih karakteristika, kao ni merenje kvaliteta u upotrebi u svim mogućim slučajevima.

Model kvaliteta (Eksterni i interni kvalitet)

Model kvaliteta deli atribute kvaliteta prema šest karakteristika:

- ***funkcionalnost*** (sposobnost softverskog proizvoda da obezbedi funkcije koje ispunjavaju iskazane i podrazumevane potrebe kada se softver koristi pod uslovima)
- ***pouzdanost*** (sposobnost softverskog proizvoda da održava specificirani nivo performansi kada se koristi pod specificiranim uslovima)
- ***upotrebljivost*** (sposobnost softverskog proizvoda da bude razumljiv, korišćen i atraktivan za korisnika, kada se koristi pod specificiranim uslovima)
- ***efikasnost*** (sposobnost softverskog proizvoda da obezbedi odgovarajuće performanse u odnosu na korišćene resurse pod postavljenim uslovima)
- ***pogodnost za održavanje*** (sposobnost softverskog proizvoda da bude modifikovan. Modifikacija može da uključuje ispravke, poboljšanje ili adaptaciju

softvera u odnosu na promene okruženja, kao i u zahtevima i funkcionalnim specifikacijama)
- ***prenosivost*** (sposobnost softverskog proizvoda da bude prenosiv iz jednog okruženja u drugo)

Ove karakteristike se dalje dele na podkarakteristike koje se mogu meriti internim ili eksternim metrikama.

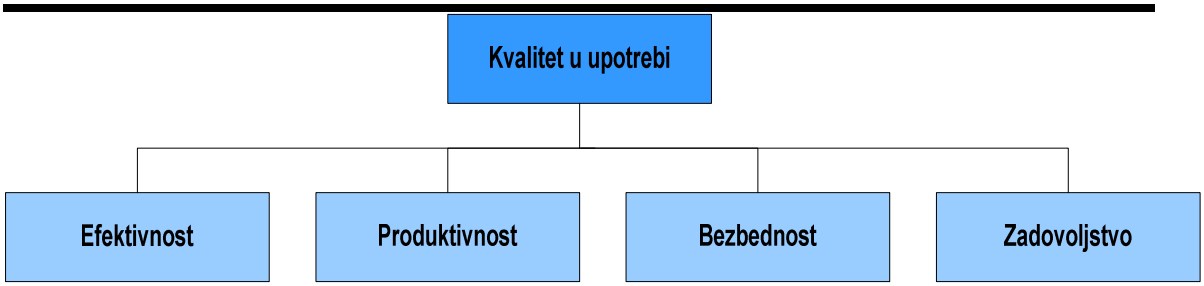
Slika 4. Model kvaliteta za eksterni i interni kvalitet

Za svaku karakteristiku i podkarakteristiku, sposobnost softvera je određena skupom internih atributa koji mogu da se izmere. Karakteristike i podkarakteristike mogu da se mere eksterno stepenom sposobnosti koji je obezbeđen sistemom koji sadrži softver.

Model kvaliteta u upotrebi

Svojstva kvaliteta u upotrebi su podeljena na četiri podkarakteristike:

- ***efektivnost*** (sposobnost softverskog proizvoda da omogući korisnicima postizanje ciljeva sa tačnošću i kompletnošću u kontekstu specifične upotrebe)
- ***produktivnost*** (sposobnost softverskog proizvoda da omogući korisnicima upotrebu odgovarajuće količine resursa u vezi sa efektivnošću postignutom u specificiranom kontekstu upotrebe)
- ***bezbednost*** (sposobnost softverskog proizvoda za postizanje prihvatljivih nivoa rizika štete ljudima, poslu, softveru, imovini ili okruženju u kontekstu specifične upotrebe)
- ***zadovoljstvo*** (sposobnost softverskog proizvoda da zadovolji korisnike u kontekstu specifične upotrebe)



Slika 5. Model za kvalitet u upotrebi

Kvalitet u upotrebi je kvalitet sa stanovišta korisnika. Merenje se obično zahteva na sva tri nivoa, jer ispunjavanje kriterijuma za interni kvalitet je obično nedovoljno za obezbeđenje ispunjenja kriterijuma za eksterni kvalitet i ispunjavanje kriterijuma za eksterno merenje podkarakteristika je obično nedovoljno za obezbeđenje ispunjenja kriterijuma za kvalitet u upotrebi.

...o metrikama kvaliteta softvera

Za merenje kvaliteta softvera prema modelu kvaliteta koriste se interne i eksterne metrike.

Interne metrike mogu se primeniti na softverske proizvode koji se ne izvršavaju (kao što su specifikacije ili izvorni kod) za vreme projektovanja i kodiranja. U toku projektovanja softverskog proizvoda međuproizvodi mogu da se vrednuju korišćenjem internih metrika koje mere unutrašnja svojstva. Osnovna namena internih metrika je obezbeđenje postizanja zahtevanog eksternog kvaliteta i kvaliteta u upotrebi. Interne metrike obezbeđuju pomoć korisnicima, osobama koje vrše vrednovanje, osobama koje vrše testiranje i projektantima, omogućavaju vrednovanje kvaliteta softverskog proizvoda i ukazuju na elemente kvaliteta mnogo ranije nego što softverski proizvod postane izvršiv. Interne metrike mere interne atribute ili pokazuju eksterne atribute analizom statičkih svojstava međuproizvoda ili softverskog proizvoda koji se isporučuje. Merenja internih metrika koristi učestalost kompozicije softverskih elemenata koji se, na primer, pojavljuju u naredbama izvornog koda, dijagrama upravljanja, toka podataka i prikazima promene stanja.

Eksterne metrike koriste merenja softverskog proizvoda koje su izvedene iz merenja ponašanja sistema čiji je on deo, testiranjem, izvršavanjem i posmatranjem softvera koji se izvršava ili sistema. Pre naručivanja ili korišćenja softverskog proizvoda treba ga vrednovati korišćenjem metrika koje su zasnovane na poslovnim objektima povezanim sa upotrebom, eksploatacijom i upravljanjem proizvodom u specificiranom organizacionom i tehničkom okruženju. Eksterne metrike obezbeđuju pomoć korisnicima, osobama koje vrše vrednovanje, osobama koje vrše testiranje i projektantima i omogućavaju im vrednovanje kvaliteta softverskog proizvoda u toku testiranja ili izvršavanja.

... o primeni internih i eksternih metrika

Nakon definisanja zahteva, definišu se i karakteristike i podkarakteristike koje se odnose na zahteve kvaliteta. U ovom trenutku se definišu i odgovarajuće eksterne metrike i kriterijumi prihvatljivosti kojima se potvrđuje da softver zadovoljava potrebe korisnika. Nakon toga se definišu interni atributi kvaliteta softvera i specificira plan radi konačnog postizanja zahtevanog eksternog kvaliteta i kvaliteta u upotrebi i ugrađuju u proizvod tokom razvoja. Interne metrike i kritertijumi prihvatljivosti se definišu u cilju određivanja atributa internog kvaliteta koji se mogu koristiti za verifikaciju da međuproizvod zadovoljava specifikacije internog kvaliteta tokom razvoja.

Preporučuje se da interne metrike koje se koriste imaju jaku vezu sa ciljnim eksternim metrikama, tako da se mogu koristiti za predviđanje vrednosti eksternih metrika. Međutim, prilično je teško razviti teorijski model koji obezbeđuje jaku vezu između internih i eksternih metrika.

...metrike kvaliteta u upotrebi

Ove metrike odnose se na stepen do kog proizvod ispunjava potrebe korisnika radi postizanja specificiranih ciljeva sa efektivnošću, produktivnošću, bezbednošću i zadovoljstvom u kontekstu upotrebe. Vrednovanje kvaliteta u upotrebi vrši validaciju kvaliteta softverskog proizvoda u scenarijima korisnik – zadatak.
Veza između kvaliteta u upotrebi i drugih karakteristika kvaliteta softverskog proizvoda zavisi od vrste korisnika:

- krajnji korisnik, za kog je kvalitet u upotrebi uglavnom rezultat funkcionalnosti, pouzdanosti, upotrebljivosti i efikasnosti;
- osoba koja je odgovorna za implementaciju softvera, za koju je kvalitet u upotrebi uglavnom rezultat pogodnosti za održavanje;
- osoba koja prenosi softver, za koju je kvalitet u upotrebi uglavnom rezultat prenosivosti;[12]

ISO/IEC 14598 (Vrednovanje softverskog proizvoda)

Drugi deo standarda ISO/IEC 9126 je ISO/IEC 14598 (Vrednovanje softverskog proizvoda). ISO/IEC 14598-1 sastoji se od nekoliko delova koji se pojavljuju pod opštim nazivom Informaciona tehnologija – Vrednovanje softverskih proizvoda.

Vrednovanje softverskog kvaliteta odnosi se na model kvaliteta, metod vrednovanja, merenje softvera, i alate za podršku. Ovaj standard namenjen je projektantima, onim koji vrše nabavku i nezavisnim ocenjivačima. Primenjene vrednosti standarda koriste rukovodiocima i projektantima, analitičarima koji se bave vezom između internih i eksternih metrika.

Proces vrednovanja

ISO/IEC 14598 serija međunarodnih standarda obezbeđuje uputstvo i zahteve za proces vrednovanja u tri različite situacije:

- Razvoj (unapređenje) (ISO/IEC 14598-3) – novi proizvodi ili poboljšanje postojećih
- Nabavku (ISO/IEC 14598-4) – nabavka proizvoda ili ponovno korišćenje postojećeg proizvoda
- Nezavisno vrednovanje (uključujući vrednovanje od treće strane) (ISO/IEC 14598-5): po zahtevu projektanata, nabavljača ili treće strane

Za vrednovanje softverskog kvaliteta, prvo se utvrđuju zahtevi vrednovanja, tada se specificiraju projektovanje i izvršenje vrednovanja.
Određivanje svrhe vrednovanja, osim opštog, može se posmatrati sa više stanovišta:

- nabavka – nabavljač utvrđuje zahteve eksternog kvaliteta, zahteve za isporučioca i procenjuje potencijalne prodavce,
- isporuka – isporučilac utvrđuje da li proizvod odgovara zahtevima kvaliteta prema nabavljaču ili drugim proizvodima,
- razvoj – korisnici i upotrebna vrednost,
- funkcionisanje – validacija da li su zahtevi kvaliteta zadovoljeni pod različitim uslovima rada,
- održavanje - validacija da li su zahtevi kvaliteta udovoljeni, i da li su zahtevi za održljivost i prenosivost dostignuti.

Tipovi proizvoda i međuproizvoda koji se vrednuju razlikuju se u zavisnosti od faze životnog ciklusa.[14]

Nadalje se objašnjava proces koji u savremenom softverskom razvoju zauzima sve važnije i ozbiljnije mesto. Testiranje softvera predstavlja najefikasniji i najbolji način dostizanja i održavanja kvaliteta softverskih proizvoda. Utemeljeno je na pravilima i principima kvaliteta u okviru standarda koji se odnose na kvalitet i, na neki način, sprovodi u delo sve unapred definisane metode i tehnike za stvaranje i održavanje kvalitetnog softvera.

5. *Testiranje softvera*

Testiranje – *(lat.testari svedočiti, posvedočiti) ispitivanje fizičkih i psihičkih osobina i radnih sposobnosti pomoću testova.*

Test – *ogled, proba.*

Testirati – *proveriti, ispitati, utvrditi testom.*

Tester *– onaj koji testira.*

Test scenario *– sekvenca test koraka koji opisuju interakciju korisnik – sistem.*

5.1. Pojam testiranja softvera

Testiranje softvera je proces za merenje kvaliteta softvera. Kvalitet se obično vezuje za pojmove tačnosti, potpunosti, sigurnosti, ali takođe uključuje i neke zahteve propisane ISO standardom, kao i tehničke zahteve ISO 9126: karakteristike, pouzdanost, efikasnost, portabilnost, istrajnost, kompatibilnost i korisnost.

Testiranje je proces tehničke istrage, čiji je cilj da otkrije informacije vezane za kvalitet proizvoda, naravno, sa poštovanjem konteksta funkcionisanja proizvoda. Ovaj proces uključuje izvršavanje programa ili aplikacije sa namerom pronalaženja grešaka. Kvalitet nije apsolutna, već vrednost za neku osobu. Kada se ovo uzme u obzir testiranje se ne može uzeti kao parametar za uspostavljanje kontrole nad softverom; testiranje obezbeđuje kritiku i komparaciju za stanje i ponašanje proizvoda u odnosu na specifikaciju.
Postoje mnogi pristupi testiranju softvera, ali je efektivno testiranje kompleksnih proizvoda u osnovi proces istrage, a ne samo pitanje kreiranja i praćenja rutinske procedure.

Jedna od definicija testiranja je: „Proces ispitivanja softvera u cilju procene istog“, gde se ispitivanje odnosi na operacije koje tester pokušava da izvrši nad proizvodom, a proizvod odgovara na to ponašanjem izazvanim probama testera. Testiranje se vezuje i za dinamičko analiziranje proizvoda kroz faze razvoja. Za statičko testiranje smatra se istraživanje, a za dinamičko pokretanje programa uz predefinisani skup test scenarija na određenom nivou razvoja .[17]

5.2. Istorija testiranja softvera

U dokumentu Gelperina i Hetzela iz 1988. godine pominje se podela istorije testiranja na sledeće faze: [19]

Perod od	Period do	Faza	Opis
-	1956	***Debagovanje***	Programe je pisao, a zatim i testirao, programer, do trenutka uklanjanja svih bug-ova. Testovi su kreirani ad hoc, na osnovu iskustva programera i fokusirani na operativnost sistema. Ne postoji jasna granica između testiranja i razvoja.
1957	1978	***Demonstracija***	Debagovanje se i dalje izvodilo kako bi sistem proradio, ali je dodat još jedan nivo testiranja - da se demonstrira da softver ne samo radi, već da radi ono što treba.
1979	1983	***Destrukcija***	Fokus testiranja se pomera sa obezbeđenja da softver radi ono što treba, na otkrivanje grešaka u sistemu.
1983	1987	***Evaluacija***	Testiranje postaje aktivnost na kraju svakog nivoa u razvoju životnog ciklusa, jer se prihvata stanovište da što ranije otkrivanje grešaka u sistemu izaziva manje troškove.
1988	-	***Prevencija***	Ideja je da se spreče greške u bilo kom stadijumu životnog ciklusa proizvoda testiranjem svakog nivoa. Testiranje se fokusira na ispravljanje grešaka.

Tabela 2. Istorijski razvoj testiranja

Odvajanje debagovanja od testiranja je uveo Glenford J. Mayers 1979. godine. Gelperin i Hetzel klasifikovali su 1988. godine faze i ciljeve softverskog testiranja koji su dati u tabeli. Dr Gelperin je osnovao IEEE 829-1989 (Standard testne dokumentacije) sa dr Hetzelom napisao knjigu „Kompletan vodič za softversko testiranje“. Ova dva rada bila su osnova savremene kulture testiranja i ostala konzistentan izvor i referenca. Dr Gelperin i Jerry E. Durant krenuli su sa razvojem Istraživačke tehnologije jakog uticaja(High Impact Inspection Technology) koja je sagrađena na tradicionalnim temeljima istraživanja, ali sa dodatkom testnih sastojaka.[17]

Kroz istoriju razvoja softvera pojavljivale su se razne definicije testiranja:

1. ***Debagovanje:*** *p*edesetih godina XX veka testiranje softvera se definisalo kao „ono što programeri rade u cilju pronalaženja grešaka u sopstvenim programima". Testiranja su bila iscrpljujuća zbog svih mogućih kombinacija koda koje su se trebale proveriti i nije bilo moguće do kraja istestirati aplikaciju, jer je: (1) domen ulaza bio prevelik, (2) ima previše mogućih ulaznih kombinacija i (3) dizajn i specifikacija se teško testiraju. Sa razvojem softvera u periodu od 60-tih do 70-tih aktivnost razvoja softvera prerasla je u „kompjutersku nauku".

2. ***Demonstracija***: ranih 70-tih testiranje softvera definiše se kao „ono što se radi da bi se demonstrirale dobre strane programa" ili kao „ proces uspostavljanja poverenja da program ili sistem rade ono što bi trebalo". Iako je ovaj pristup teoretski obećavao puno, u praksi je zahtevao previše vremena i bio je nepotpun. Kada se radilo o prostim testovima bilo je lako pokazati da softver radi i dokazati da će teorijski raditi. Međutim, u većini slučajeva softver nije trebalo testirati samo na ovaj način i veliki broj grešaka ostajao je neotkriven do same implementacije. Ubrzo je izveden zaključak da „dokaz ispravnosti" nije dovoljan metod u testiranju softvera.

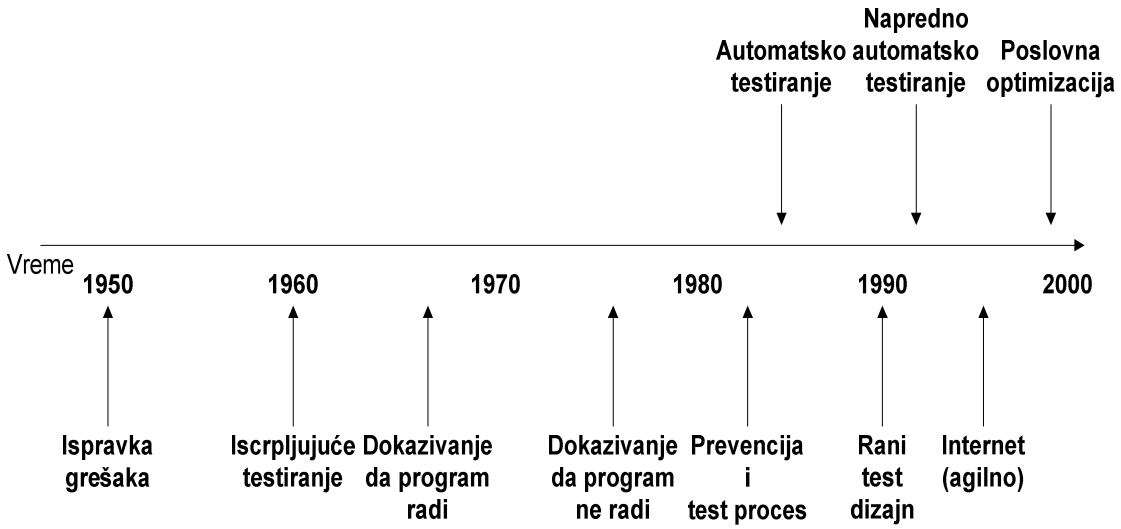
3. ***Destrukcija***: kasnih 70-tih ustanovljeno je da je testiranje proces izvršenja programa sa namerom pronalaženja greške, a ne omogućavanja da softver radi. Naglašeno je da je dobar test onaj kod kog postoji velika verovatnoća za pronalaženje neotkrivene greške.

4. ***Evaluacija i prevencija*****:** u periodu 80-tih definicija testiranja se proširila i uključila prevenciju grešaka. Dizajniranje testova predstavlja jednu od najefektivnijih tehnika za prevenciju. Predlaže se da metodologija testiranja bude obavezna, da uključuje kontrolu kompletnog životnog ciklusa i da bude organizovan proces. Ovaj pravac ističe značaj, ne samo testiranja programa, već i zahteva, dizajna, koda, samih testova i na kraju programa. „Testiranje" se tradicionalno (do 80-tih) odnosi na ono što je urađeno sa sistemom do isporuke koda koji radi, a danas je testiranje „više testiranje" u kom tester treba da učestvuje u skoro svakom aspektu razvoja u životnom ciklusu softverskog proizvoda. Pored testiranja i provere koda isporučenog na testiranje, u trenutku kada nešto nije dobro moraju se ispitati i prethodne faze razvoja. Ako je greška izazvana nedostatkom u dizajnu ili programskim propustom, neophodno je identifikovati izvor problema. Istraživanja pokazuju da oko 50% grešaka nastaje kada se definiše ono što se želi od softvera (faza zahteva) ili u fazi dizajna, i da ove greške utiču na ostale defekte za vreme kodiranja.

5. ***Automatsko i novi oblici testiranja:*** sredinom 80-tih pojavili su se alati za automatsko testiranje i pokrenuta je automatizacija manuelnih testova kako bi se povećala efikasnost i kvalitet aplikacije. Poznato je da računar može da uradi više

testova nego čovek ručno, i da ih, naravno, izvodi pouzdanije. Ovi alati su u početku bili jako prosti i nisu posedovali napredan jezik za skriptovanje. Ranih 90-tih uočene su prednosti ranog dizajniranja testova. Testiranje se definiše kao planiranje, dizajn, izgradnja, održavanje i izvršavanje testova i testnih okruženja. U ovom periodu pojavljuju se napredniji alati za testiranje koji nude veći obim jezika za skriptovanje i oblika izveštavanja. Razvili su se i alati za merenje performansi sistema. Ovi alati testirali su stres i preopterećenje sistema kako bi odredili slabe tačke. Mada je koncept testa kao procesa opstao kroz celokupan razvoj životnog ciklusa, sredinom 90-tih, softver se razvijao i bez specifičnih modela testiranja, što je značajno unazadilo ovaj proces. Ovaj problem se naziva „agilno testiranje".

6. ***Novo doba:*** početkom XXI veka Mercury Interactive je uveo još širu definiciju testiranja uvođenjem koncepta optimizacije poslovne tehnologije (BTO). BTO povezuje IT strategiju i izvršenje sa poslovnim ciljevima. Koncept pomaže upravljanju prioritetima, ljudima i procesima IT-a. Osnovni pristup podrazumeva merenje i maksimiziranje vrednosti u okviru životnog ciklusa, kako bi se obezbedio kvalitet, performanse i ciljevi dostupnosti.[10]



Slika 6. Istorija testiranja softvera

5.3. Razvoj softvera - testiranje softvera: paralele

Prvi računari razvijeni su 50-tih godina XX veka. Koristio se FORTRAN kao programski jezik. Krajem 60-tih pojavio se koncept strukturnog programiranja koji je podrazumevao da se ma koji program može napisati korišćenjem tri prosta elementa: proste sekvence, if/then/else i do/while iskaza.

Sedamdesetih godina razvoj naglašava dizajn tehnike. U krugovima koji su se bavili razvojem postaje jasno da strukturno programiranje ne garantuje kvalitet – program se mora dizajnirati pre kodiranja. Prihvaćene su i razvijene tehnike strukturnog dizajna i kompozitne dizajn tehnike (Yordon, Mayers, Constantin). Orijentacija je i dalje bila procesna.
Filozofija strukturnog dizajna podrazumevala je particionisanje i organizovanje delova sistema. Pod pojmom „particionisanje“ podrazumeva se deljenje problema na manje podprobleme, tako da svaki podproblem ne neki način odgovara delu sistema. Usko povezani delovi problema treba da pripadaju istom delu sistema. Nepovezani delovi problema treba da se nalaze u nepovezanim delovima sistema.

Osamdesetih je utvrđeno da strukturno programiranje i tehnike softverskog dizajna nisu dovoljne: moraju se prvo definisati zahtevi, kako bi se pravi sistem isporučio korisniku. U centru priče je kvalitet koji nastaje kada korisnik primi tačno ono što je želeo.
Pojavile su se mnoge tehnike za definisanje zahteva, kao što su dijagrami toka podataka. Važan deo dijagrama toka podataka je skladište, kao reprezentacija dela sistema gde treba da se skladište podaci.
Upravo tada kreirao se koncept modela podataka: pojednostavljen opis realnog sistema u pogledu podataka, npr. logičkog izgleda podataka. Komponente ovog pristupa uključivale su entitete, veze, kardinalnosti, referencijalni integritet i normalizaciju. Do ove logičke reprezentacije podataka, u centru pažnje bili su procesi koji su komunicirali sa bazom. Zagovornici logičkog pogleda na podatke inicirali su da u središtu analize prvo budu podaci, pa potom proces. Kasnije je prihvaćeno da se prilikom definisanja zahteva moraju razmatrati i proces i podaci.

Sredinom osamedesetih godina pojavio se koncept informacionog inženjeringa. Bila je to nova disciplina koja je vodila svet u novu infomacionu eru. Pristup se bavi temama kao što su: razumevanje kako informacije mogu da se skladište i reprezentuju i kako se informacije procesuiraju za različite servise i aplikacije. Na probleme dizajna primenjene su analitičke tehnike za rešavanje problema, uz podršku matematičkih i drugih teorija. Informacioni inženjering naglasio je značaj preduzetničkog pogleda na razvoj aplikacija. Prilikom modelovanja kompletne organizacije u pogledu procesa, podataka, rizika, kritičnih faktora uspeha i drugih dimenzija, zaključeno je da i menadžement može da profitira iz ovog pristupa u smislu unapređenja upravljanja organizacijom.
Za vreme ovog perioda, tehnike razvoja softvera su jako napredovale i proces razvoja je postao brži i lakši. Nažalost, brzi zaokret na polju aplikacija, doveo je do nazadovanja na polju osnovnih rezvojnih tehnika.

Ekstremno programiranje(XP) je primer ovog nazadovanja. XP je specifičan pristup razvoju softvera i smatra se da nema aspekte dizajna. Metodologija ekstremnog programiranja nalaže radikalno napuštanje opšteprihvaćenih procesa razvoja softvera. Postoje dva pravila XP-a:
- Malo dizajna i bez zahteva.

- Fokus na korisničku stranu priče i ništa se ne dizajnira dok ne dođe vreme programiranja.

Iako mnogi ljudi u razvoju softvera smatraju zahteve i dokumentaciju vitalnom, XP preporučuje kreiranje što manje dokumentacije. Developer u ovom pristupu ima test scenarije pre nego li uradi bilo šta drugo. Osnovna premisa testa pre dizajna je da se test klasa piše pre prave klase; na ovaj način svrha prave klase nije samo da zadovolji zahtev, nego i da prođe sve testove koje sadrži testna klasa. Problem ovog pristupa je to što je potrebno dodatno nezavisno testiranje kako bi se dopunilo ono o čemu developer nije razmišljao ili nije bio u stanju da otkrije sopstvenim testiranjem.

5.4. Komparacija pravaca - ekonomski aspekti

U literaturi se navode prednosti nekih od pravaca ili metoda u testiranju posmatrano sa aspekta troškova pre svega:

1. ***Prevencija ili detekcija*:** Kvalitet se ne može postići na već gotovom proizvodu. Cilj je da se spreče defekti ili nedostaci i da se proizvod učini merljivim pomoću parametara kvaliteta. Neke od mera kvaliteta su: struktuiranje razvojnog procesa uz pomoć standarda za razvoj softvera i podršku razvojnog procesa metodama, tehnikama i alatima. Nedetektovane greške koje su izazvale milione poslovnih gubitaka uslovile su rast nezavisnog testiranja. Troškovi proizvodnje se smanjuju sa ranim otkrivanjem i ispravkom grešaka. Uz alate za automatsko testiranje, dugoročno gledano, rezultat su proizvodi visokog kvaliteta i smanjeni troškovi održavanja. Ukupni trošak efektivnog upravljanja kvalitetom predstavlja sumu četri komponente troškova: prevencije, inspekcije, internih i eksternih otkaza. Preventivni troškovi sastoje se od akcija koje se preduzimaju kako bi se sprečili defekti. Inspekcioni troškovi sastoje se od merenja, procene i provere proizvoda ili usluga i njihovog slaganja sa standardima i specifikacijama. Troškovi internih otkaza su oni koji podrazumevaju ispravku proizvoda sa greškama pre njihove isporuke. Troškovi eksternih otkaza sastoje se od grešaka otkrivenih nakon lansiranja proizvoda. Prevencija se najviše isplati jer povećavanje preventivnih troškova smanjuje broj defekata koji idu kupcu neotkriveni i tako se unapređuje kvalitet proizvoda i smanjuju troškovi proizvodnje i održavanja.

2. ***Verifikacija ili validacija:*** Verifikacija omogućava da proizvod zadovolji zahteve koji si specificirani tokom prethodnih aktivnosti i oblikovani kroz životni ciklus proizvoda, a validacija zadovoljenje zahteva kupaca na kraju životnog ciklusa proizvoda. Kreiranje test proizvoda je bliže validaciji nego verifikaciji. Testiranje softvera se posmatra kao validacioni proces. Nakon završetka programiranja sistem se validira ili testira da bi se odredile funkcionalne ili operativne performanse. Naravno, savremeni procesi testiranja se ne odvijaju samo nakon završetka programiranja, već su utkane u sam proces razvoja softvera. Dobitna kombinacija je korišćenje verifikacije i validacije u procesu testiranja.

Verifikacija obuhvata sistematične procedure pregleda, analize i testiranja ugrađene u životni ciklus, počevši od faze softverskih zahteva, do faze kodiranja. Verifikacija obezbeđuje kvalitet i održavanje softvera. Koncept verifikacije uključuje dva kriterijuma: softver mora adekvatno i korektno izvršavati sve funkcije i ne sme izvršavati one funkcije koje sam po sebi ili u kombinaciji sa ostalim funkcijama mogu degradirati performanse čitavog sistema. Verifikacija takođe omogućava interoperabilnost među različitim sekcijama u softverskoj dokumentaciji i povezanim delovima zahteva. Jedina kritika verifikacije je da povećava troškove razvoja softvera. Kada se troškovi softvera posmatraju kroz čitav životni vek proizvoda, verifikacija zapravo umanjuje troškove softvera. Uz efektivan proces verifikacije redukuju se defekti instaliranih sistema. Ukupna ušteda prevazilazi početni trošak, jer ispravka grešaka može koštati 20 do 100 puta više za vreme operacija i održavanja sistema, nego za vreme dizajniranja.[10]

5.5. Proces testiranja

Testiranje se sastoji od sledećih aktivnosti:

1. ***Planiranje testiranja*** (određuje se predmet, cilj i razlog testiranja(na osnovu zahteva, modela i drugih ulaza testiranja), gde se testiranje vrši, kada se vrši i ko izvršava testove)
2. ***Dizajn testova*** (određuje kako se sprovodi testiranje na osnovu artefakta tj. test primera)
3. ***Implementacija testova*** (pravljenje višestruko upotrebljivih test scriptova koji realizuju test primere)
4. ***Izvršavanje testova*** (izvršavanje implementacije testa radi provere funkcionalnosti sistema)
5. ***Evaluacija testova*** (procena testova tj. validnosti izvršavanja testa, analiza izlaza, pregled zbirnih rezultata, uticaj promene zahteva i ulaza na plan testiranja)

Svaka od ovih aktivnosti ima ulaze i izlaze. U procesu testiranja koriste se i različiti alati koj pospešuju proces i daju bolji uvid u rezultate(npr.Rational, Test Complete i sl.).

Organizacija procesa testiranja

Proces testiranja mora se odvijati tokom svih faza životnog ciklusa. Za ovaj proces moraju se realizovati standardizovani propisi i procedure koje definišu način rada i aktivnosti test odeljenja, kao i svih učesnika u razvoju. Sve aktivnosti članova tima koji učestvuju u realizaciji procesa testiranja moraju biti dokumentovane i moraju pokriti sledeće:

- Odgovornosti i zaduženja za planiranje testa, izvršenje i evaluaciju.
- Ciljeve testa, tipove koji će se primeniti, raspored i zaduženja kod testiranja kroz plan testiranja.

- Odgovarajući materijal (dokumenta, test primeri i dr.).
- Test materijal je podložan periodičnim aktivnostima kontrole kvaliteta.[11]

Tipovi testiranja

Inspekcije, recenzije i pregledi:

Inspekcije, recenzije i pregledi su specifične tehnike fokusirane na ocenjivanje artefakata(pogodno za dokumentaciju, programski kod) i efikasne su metode za poboljšanje kvaliteta i razvojnog procesa proizvoda. Sprovode se na sastancima, gde jedan od učesnika ima ulogu vodećeg, a ostali uloge zapisičara(beleži pitanja, predloge, probleme i sl.). Ove tehnike opisuju se u okviru IEEE standarda na sledeći način:

Recenzija (review) – formalni stastanak na kome se artefakt ili skup artefakata predstavlja korisniku ili drugim stranama koje učestvuju u procesu odobravanja ili komentarisanja.

Inspekcija (inspection) – fomalna tehnika procene u kojoj se artefakti detaljno pregledaju. Pregled vrše osobe koje nisu autori, da bi se lakše uočile greške i drugi problemi.

Pregledi (walkthrough) – Tehnika pregleda u kojoj autor upoznaje ostale članove tima sa elementima artefakta koji je izgradio. Ostali članovi učestvuju aktivno u raspravi.

Tehnike testiranja

1. ***Jedinično testiranje (unit testing)*** – primenjuje se na pojedine klase, module ili komponente programskog koda. Ova tehnika deli se na tehnike bele i crne kutije.

2. ***Integraciono testiranje*** – primenjuje se na softverski sitem kao celinu.

3. ***U testiranja višeg reda spadaju:***

- ***testiranje sigurnosti*** (security testing): da li su funkcije dostupne onim i samo onim korisnicima kojima su i namenjene.
- ***testiranje količine podataka*** – verifikovanje da li softver može obraditi veliku količinu podataka.
- ***testiranje upotrebljivosti(usability testing)*** – estetski aspekti, konzistentnost korisničkog interfejsa, korisnička dokumentacija.
- ***testiranje integriteta(integrity testing)*** – robusnost(otpornost na otkaze), konzistentna upotreba resursa i sl.

- ***test u stresnim uslovima(stress testing)*** – vrsta testa pouzdanosti sistema pod nenormalnim uslovima (velika opterećenja sistema, nedovoljno memorije ili drugih resursa, neraspoloživi servisi i sl.).
- ***etalonski test*** - upoređenje performansi novog sistema sa nekim, referentnim, poznatim sistemom.
- ***test zagušenja*** – provera da li sistem može da zadovolji višestruke zahteve različitih aktera za istim resursom.
- ***test opterećenja*** – vrsta testa performansi kojim se procenjuju operativni limiti nepromenjivog sistema pod različitim opterećenjima ili različitih konfiguracija sistema pri istom opterećenju. Najčešće se mere protok i vreme odziva (srednja i granična vrednost).
- ***test konfiguracije (configuration testing)*** – testira ponašanje softvera u različitim hardversko/softverskim okruženjima.
- ***test instalacije(installation testing)*** – testira instaliranje softvera na različitim sistemima i u različitim situacijama (npr. prekid napajanja ili nedovoljno prostora na disku).

4. ***Regresiono testiranje*** – na osnovu jednom razvijenog testa više puta se sprovodi testiranje softvera (tipično nakon neke izmene u softveru da bi se utvrdilo da nisu pokvarene funkcionalnosti softvera).

...pregled tehnika testiranja

Testiranje se posmatra kao posebna faza životnog ciklusa proizvoda koja prati kodiranje. Moderni model softverskog životnog ciklusa izbegava ovakvo gledanje na stvari i u prvi plan stavlja iterativno testiranje kroz razvoj životnog ciklusa.
Testiranje omogućava ranu detekciju grešaka i ispravku istih i tehnički uvid u pravu prirodu performansi sistema. Obično se koristi nekoliko testnih metodologija da bi se obradili različiti aspekti softverskog proizvoda. [24]
Kako su do sada objašnjeni samo neki od tipova i tehnika testiranja u tabeli je dat pregled savremenih tehnika testiranja sa opisom. Tehnike se kombinuju i variraju.

Tehnika	Kratak opis
Acceptance testing – Testiranje prihvatljivosti	Finalno testiranje bazirano na specifikacijma krajnjeg korisnika/potrošača, ili bazirano na korišćenju krajnjih korisnika/potrošača u određenom vremenskom periodu.
Ad hoc testing – Ad hoc testiranje	Slično istraživačkom testiranju, ali se često odnosi na to da testeri dobro razumeju softver pre samog testiranja.

Alpha testing – Alfa testiranje	Testiranje kada je razvoj pri kraju; manje promene u dizajnu moguće su kao rezultat ovog testiranja. Obično ga izvršavaju krajnji korisnici ili drugi, a ne programeri i testeri.
Basis path testing – Testiranje osnovnih tokova	Identifikovanje testova na bazi tokova i putanja programa ili sistema.
Beta testing – Beta testiranje	Kada su razvoj i testiranje u suštini završeni i potrebno je pronaći konačne greške i probleme pre samog lansiranja proizvoda. Obavljaju ga krajnji korisnici ili drugi, ne programeri ili testeri.
Black-box testing – Black-box testiranje	Testovi se generišu na osnovu funkcionalnosti sistema.
Bottom – up testing – Testiranje od dna ka vrhu	Integrisanje modula ili programa počevši od dna.
Boundary value testing – Testiranje graničnih vrednosti	Testovi se generišu iz graničnih vrednosti odgovarajućih klasa.
Branch coverage testing – Branch(gransko) testiranje	Verifikacija da svaka grana ima true ili false izlaze.
Branch/condition coverage – Grana/uslov testiranje	Verifikacija da svaki uslov obuhvata sve moguće izlaze .
Cause/effect graphing – Uzrok/posledica graf	Označavanje višestrukih simultanih ulaza koji mogu uticati na uslove testa.
Comparison testing – Uporedno testiranje	Upoređivanje slabosti i dobrih strana softvera sa konkurentskim proizvodima.
Compatibility testing – Testiranje kompatibilnosti	Testiranje koliko dobro softver radi u određenom hardversko/softverskom/OS/mrežnom okruženju.
Condition coverage testing – Testiranje pokrivenosti uslova	Potvrda da svaki uslov obuhvata sve moguće izlaze.
CRUD testing – CRUD testiranje	Pravljenje CRUD matrice i testiranje svih

	kreacija objekata, čitanja, ažuriranja i brisanja.
Database testing – Testiranje baze podataka	Provera integriteta vrednosti u poljima baze podataka.
Decision tables – Tabele odluka	Tabele koje pokazuju kriterijume odluke i respektivne akcije.
Desk checking – Provera	Developer pregleda kod.
End-to-end testing – Sa kraja na kraj testiranje	Slično sistemskom testiranju, uključuje test kompletnog okruženja aplikacije u situaciji kada se oponaša stvarni korisnički svet, npr. interakcija sa bazom podataka, korišćenje mrežne komunikacije, ili interakcija sa drugim hardverom, aplikacijama ili sistemima.
Equivalence partitioning – Ekvivalentno particionisanje	Svaki ulazni uslov se deli na dve ili više grupa. Testovi se generišu iz reprezentativnih validnih ili nevalidnih klasa.
Exception testing – Testiranje izuzetaka	Identifikovanje poruka o grešci i obrade izuzetaka i uslova koji ih okidaju.
Exploratory testing – Istraživačko testiranje	Često označava kreativno, neformalno testiranje koje se ne bazira na formalnim test planovima ili test case-ovima; testeri uče o softveru za vreme testiranja.
Free form testing – Slobodna forma testiranja	Korišćenje ad hoc ili brainstorming intuicije za definisanje test case-ova.
Gray-box testing – Gray-box testiranje	Kombinacija Black-box i white-box testiranja, kako bi se izvukle najbolje strane oba ova tipa.
Histograms – Histogrami	Grafička reprezentacija izmerenih vrednosti organizovanih prema frekvenciji događanja koja se koristi da naglasi „vruće“ tačke.
Incremental integration testing –	Kontinualno testiranje aplikacije kada se

Inkrementalno integraciono testiranje	doda nova funkcionalnost, zahteva da različiti aspekti funkcionalnosti aplikacije budu dovoljno nezavisni da rade zasebno, pre kompletiranja svih delova programa. Izvršavaju ga testeri ili programeri.
Inspections – Inspekcije	Formalni pregled koji koristi čekliste, ulazne i izlazne kriterijume.
Integration testing – Integraciono testiranje	Testiranje kombinovanih delova sistema radi utvrđivanja da li zajedno dobro funkcionišu. Delovi mogu biti kodni moduli, individualne aplikacije ili klijent/server aplikacije na mreži. Ovaj tip testiranja je jako važan za klijent/server i distribuirane sisteme.
JADs	Tehnika koja spaja korisnike i developere pri dizajnu sistema.
Load testing – Load testiranje	Testiranje aplikacije pod opterećenjem.
Mutation testing – Mutaciono testiranje	Metod određivanja da li je skup testnih podataka ili testova koristan, namernim uvođenjem raznih promena u kodu(„bugs“) i retestiranje sa originalnim test podacima/slučajevima kako bi se odredilo da li su otkrivene greške. Zahteva velike resurse.
Orthogonal array testing – Ortogonalno testiranje niza	Matematička tehnika određivanja koje varijacije parametara treba da se testiraju.
Pareto analysis – Pareto analiza	Analiza paterna defekata radi identifikacije uzroka i posledica.
Performance testing – Testiranje performansi	Koristi se često sa stres i load testovima. Definisano je u zahtevima ili test planovima.
Positive and negative testing – Pozitivno i negativno testiranje	Testiranje pozitivnih i negativnih vrednosti svih ulaza.
Prior defect history testing – Testiranje prethodnih defekata	Testovi se kreiraju ili ponovo pokreću za svaki defekt koji se pronađe u prethodnim

	testovima sistema.
Prototyping – Prototipovanje	Pristup prikupljanja podataka od korisnika izgradnjom i demonstracijom nekog dela potencijalne aplikacije.
Random testing – Nasumično testiranje	Nasumična selekcija iz skupa ulaznih vrednosti, gde je svaka vrednost jednako važna.
Range testing – Testiranje opsega	Za svaki ulaz definiše se opseg za koji bi ponašanje sistema trebalo da bude isto.
Recovery testing – Testiranje oporavka	Testiranje koliko dobro se sistem oporavlja od padova, hardverskih otkaza ili drugih velikih problema ovog tipa.
Regression testing – Regresiono testiranje	Testiranje sistema sa manjim izmenama za vreme spiralnog razvoja, debagovanja, održavanja ili razvoja u novom release-u.
Risk-based testing – Testiranje zasnovano na riziku	Meri stepen poslovnog rizika u sistemu kako bi se poboljšalo testiranje.
Run charts – Grafici izvršavanja	Grafički prikaz kako karakteristike kvaliteta variraju u vremenu.
Sandwitch testing – Sendvič testiranje	Intergacija modula ili programa sa vrha i dna simultano.
Sanity testing – Zdravorazumsko testiranje	Inicijalni test napor kako bi se odredilo da li se novi softver dobro ponaša, kako bi se prešlo na veći stepen testiranja. Npr. Ako novi softver obara sistem svakih 5 minuta, neće biti potrebno testiranje u takvom stanju.
Security testing – Testiranje sigurnosti	Testiranje koliko dobro sistem reaguje na neautorizovane napade.
State transition testing – Testiranje prelaza stanja	Tehnika u kojoj se prvo identifikuju stanja sistema, a potom se pišu test case-ovi kako bi se testirali okidači koji prouzrokuju prelaz iz jednog stanja u drugo.

Statement coverage testing – Testiranje izraza	Svaki izraz programa se izvršava bar jednom.
Statistical profile testing – Statističko profil testiranje	Statističke tehnike se koriste da razviju korisnički profil sistema koji pomaže da se definišu prelazne putanje, uslovi, funkcije i tabele podataka.
Stress testing – Stres testiranje	Testiranje funkcionalnosti sistema pod opterećenjem, ponavljanjem određenih akcija ili ulaza, veliki ulazni brojevi ili kompleksni upiti nad bazom.
Structured walkthroughs – Strukturni pregledi	Tehnika za upravljanje sastankom na kom učesnici projekta ispituju rad proizvoda.
Syntax testing – Sintaksno testiranje	Tehnika vođena podacima i test kombinacijama ulazne sintakse.
System testing – Sistemsko testiranje	Black-box tip testiranja koji se bazira na zahtevima; pokriva sve kombinovane delove sistema.
Table testing – Tabelarno testiranje	Testiranje pristupa, sigurnosti i integriteta podataka tabela ulaza.
Thread testing – Testiranje niti	Kombinovanje individualnih jedinica u okviru niti funkcionalnosti koje zajedno postižu neku funkciju ili skup funkcija.
Top-down testing – Testiraje od vrha ka dnu	Integracija modula ili programa počevši od vrha.
Unit testing – Jedinično testiranje	Mikro testiranje određenih funkcija ili modula koda. Obično ih obavlja programer, a ne tester, jer zahteva detaljno poznavanje internog dizajna i koda programa.
Usability testing – Testiranje korisnosti	Test izgleda aplikacije za korisnika. Subjektivno je i zavisi od ciljne grupe korisnika. Koriste se intervjui, video zapisi i druge tehnike. Programeri i testeri ne odgovaraju za ovaj tip testiranja.
White-box testing – White-box testiranje	Testovi se definišu ispitivanjem logičkih

	iskaza sistema.

Tabela 3.Opis tehnika za testiranje

Česta je i podela testiranja na:

1. black-box, white-box i gray-box testiranje,
2. manuelno i automatsko testiranje,
3. statičko i dinamičko testiranje.

1. Black-box, white-box i gray-box testiranje

Black-box ili funkcionalno testiranje počiva na test uslovima koji se razvijaju na nivou programa ili funkcionalnosti sistema. Tester zahteva informacije o ulaznim podacima i izlazima koje prati, ali zapravo ne zna kako sistem radi. Isto kao što nije neophodno u detalje poznavati kako automobil radi da bi mogao da se vozi, nije neophodno poznavati internu strukturu programa da bi se isti izvršavao. Tester se fokusira na testiranje funkcionalnosti programa koje nisu u skladu sa specifikacijom. Na ovaj način, tester posmatra program kao crnu kutiju i ne zanima ga unutrašnja struktura programa ili sistema. Neki od primera ove vrste testiranja su: tabele odluka, testiranje opsega, testiranje graničnih vrednosti, testiranje integriteta baze, testiranje izuzetaka, nasumično testiranje i sl. Najveća prednost ovog testiranja je da testovi obuhvataju ono što sistem ili program treba da radi i potpuno su prirodni i razumljivi za bilo koga. Ova tehnika se verifikuje tehnikama kao što su struktuirani pregledi, inspekcije i zajedničko dizajniranje aplikacija (JADs).

White-box ili strukturno testiranje obuhvata ispitivanje interne strukture programa ili sistema. Testni podaci se dobijaju ispitivanjem logike programa ili sistema, bez brige o zahtevima koje treba da zadovolji. Tester poznaje internu strukutru i logiku programa, isto kao što mehaničar zna unutrašnji mehanizam automobila. Specifični primeri koji spadaju u ovu kategoriju uključuju testiranje izraza, gransko testiranje, testiranje uslova i sl. Prednost ovog pristupa je da je potpun i da se fokusira na proizvedeni kod. Greške i nenamerni propusti, zbog poznavanja unutrašnje strukture ili logike, imaju više šanse da budu otkriveni. Jedna od mana ovog testiranja je da ne proverava da li su specifikacije tačne, već da se fokusira samo na internu logiku. Druga loša karakteristika je da ne postoji način da se otkriju elementi koji nedostaju i greške vezane za osetljive podatke. Npr. Ako izraz u programu treba da se kodira kao „ako $|a - b| < 10$“, ali se kodira kao „ako $(a - b) < 1$“, nemoguće je otkriti grešku bez detalja iz specifikacije.
Poslednji nedostatak je da ovaj tip testiranja ne može da obuhvati kompletnu logiku programa, jer bi to podrazumevalo kreiranje jako velikog broja testova.

Gray-box ili funkcionalno i srukturno testiranje predstavlja kombinaciju black i white-box testiranja. Tester proučava zahteve i komunicira sa developerom kako bi razumeo internu strukturu sistema.

2. Manuelno i automatsko testiranje

Osnove manuelnog testiranja počivaju na tome da su njegovi nosioci ljudi i da nije implementirano na računaru; osnove automatskog, upravo na tome da je implementirano na računaru.

3. Statičko i dinamičko testiranje

Statičko testiranje ne zavisi od vremena npr. provera sintakse, strukture i sl.
Dinamičke tehnike zavise od vremena i podrazumevaju izvršavanje specifičnog niza instrukcija na papiru ili računaru.[10]

5.6. Edvard Deming

Radni okvir kvaliteta razvio je Edvard Deming. Deming je učestvovao u procesu unapređenja kvaliteta japanske proizvodnje. Definisao je 14 principa kvaliteta koji se moraju koristiti konkurentno, kako bi se dostigao određeni nivo kvaliteta. Ovi principi su primenjivani u industriji, školama i bolnicama, ali su mnogi od njih primenjivi i na dostizanje softverskog kvaliteta.

Principi kvaliteta	***Opis***
Obezbediti konstantnost rezultata.	Planiranje budućnosti, inovacije, investicije u istraživanja i obrazovanje i stalno poboljšanje proizvoda i usluga. Ovo se dostiže kroz plan obezbeđenja kvaliteta, zahtevanjem da testeri kreiraju i održavaju test palnove po projektima, ohrabrivanjem analitičara i testera da iznose nove ideja za poboljšanje kvaliteta i stremljenjem ka konstantnom unapređenju procesa kvaliteta.
Usvojiti novu filozofiju.	Kvalitet mora postati nova religija. Prinicip se sprovodi u delo kroz: edukaciju organizacije o potrebama i vrednostima kvaliteta, isticanju činjenice da je sektor za obezbeđenje kvaliteta važan koliko i svaki drugi, da nije samo kontrolisanje i propitivanje, već mnogo više od toga.
Ukinuti zavisnost od završne inspekcije.	Ispitivanje se često ostavlja za krajnji proizvod, kada je teško otkriti defekte i kada oni mnogo koštaju.

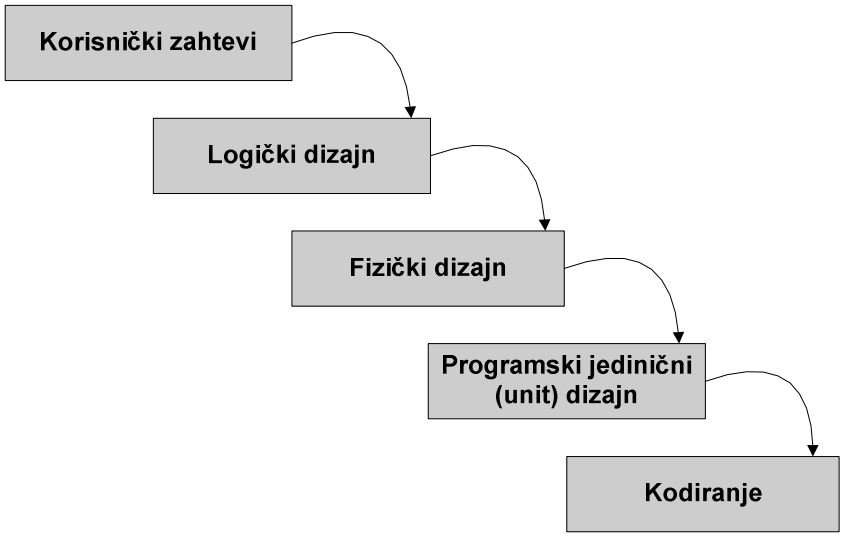
	Princip se obezbeđuje kroz: podržavanje stalnih inspekcija, pregleda i kontrola tokom čitavog životnog ciklusa, usađivanje mišljenja da je kvalitet važan u organizaciji i zahtevanjem statističkih dokaza o kvalitetu.
Prekinuti praksu napredovanja u poslu samo na bazi cene.	Zahtevati kvalitet softvera i forsirati obezbeđenje statističkih dokaza kvaliteta. Princip se dostiže kroz: zahteve kvaliteta i testiranje prodavaca, izbor najboljih prodavaca za svaki alat za obezbeđenje kvaliteta, test alat ili uslugu, i razvoj i održavanje veza u skladu sa planom kvaliteta.
Razvijati konstantno i trajno sistem proizvodnje i usluga.	Napredovanje nije trenutan, već trajan proces. Dostiže se kroz: konstantno poboljšanje kvaliteta i procesa testiranja, korišćenje statističkih tehnika za otkrivanje problema i analizu testova.
Negovati obuku i trening radnika.	Negovanje modernih tehnika i praksi. Dostiže se kroz: institucionalizaciju modernih praksi; ohrabrivanje osoblja da konstantno unaređuju znanja o kvalitetu i tehnikama testiranja, kroz stalne kurseve i seminare; nagrađivanjem zaposlenih za nove ideje i korišćenje statističkih tehnika za određivanje u kom trenutku je potreban trening i edukacija.
Negovati rukovođenje. .	Jasno definisati poslove. Dostiže se kroz: obuku developera za unit testiranje kada je broj grešaka koji je SQA sektor pronašao veliki, poboljšanje nadzora, omogućavanje da vođa projekta provede što više vremena pomažući timu na poslu, korišćenje statističkih metoda za određivanje grešaka.
Isterati strah.	Strah ubija kreativnost. Princip se dostiže kroz: promovisanje ideje da je kvalitet pozitivan i da ga treba

	nagrađivati i uveriti se da pre testiranja svako razume osnove kvaliteta.
Ukloniti barijere među ljudima u organizaciji.	Timski rad je osnova uspeha. Dostiže se kroz: isticanje potrebe da sektor za obezbeđenje sarađuje sa ostalim sektorima, isticanje činjenice da se greške otkrivene ranije, i u organizaciji, neće otkriti na tržištu, od strane kupca.
Eliminisati slogane i opomene za radnu snagu.	Ne pomažu osoblju. Prinicip se dostiže kroz: ohrabrenje menadžmenta da ne koristi slogane, umesto toga razvijati dokumentovane standarde kvaliteta, procedure i procese koje ostatku organizacije mogu poslužiti kao sredstvo za maksimiziranje kvaliteta.
Eliminisati numeričke ciljeve.	Brojke koče kvalitet. Dostiže se kroz: posmaranje standarda kvaliteta , a ne samo brojeva; izbegavanje objavljivanja stepena grešaka na individualnom ili nivou organizacije; saradnju u cilju stvaranja standarda kvaliteta; neformalni razgovor sa menadžerom, kada su posebni problemi u pitanju.
Ukloniti prepreke tipa „ponos“ radnika.	Ljudi se često posmatraju kao sredstvo za postizanje rezultata, koji se samo vrate na tržište kada ne ispune svoju svrhu. Princip se dostiže kroz: kreiranje slike kvaliteta kao važnog proizvoda i sredstva, delegiranje odgovornosti u pogledu dostizanja kvaliteta.
Negovati snažan program edukacije i treninga.	Učenje novih tehnika i veština. Dostiže se kroz: ohrabrenje osoblja na učenje novih tenhika, nagrađivanje osoblja za nove ideje, pojedinačni trening.
Preduzeti nešto da bi se postigla transformacija.	Sprovoditi ove principe.

Tabela 4. Principi kvaliteta

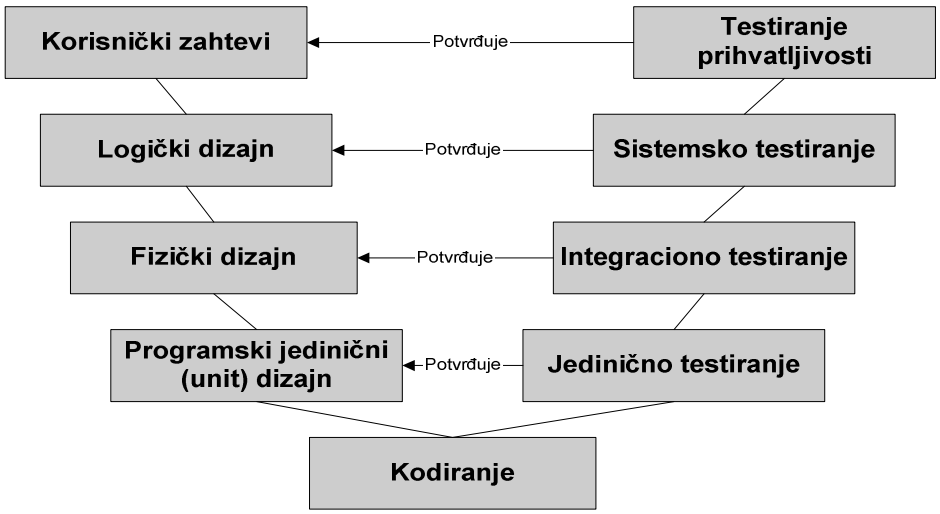
Demingovi principi se primenjuju na softversko testiranje u okviru tradicionalnog „vodopad“ (waterfall) modela i u okruženjima aplikacija „spiralnog“ razvoja (Konstantnog unapređenja procesa).

Model vodopada podrazumeva pristup sa predefinisanim sekvencijalnim koracima razvoja, koji su praćeni jasno definisanim zahtevima.



Slika 7. Model vodopada

Testiranje životnog ciklusa softvera znači da se testiranje obavlja paralelno sa razvojnim ciklusom i da je kontinualan proces. Testiranje softvera treba da započne u ranoj fazi životnog ciklusa aplikacije, ne samo u tradicionalnoj validacionoj fazi testiranja, nakon što je kodiranje završeno. Testiranje mora biti integrisano u razvoj aplikacije. Kako bi se to ostvarilo, potrebno je da deo organizacije koji se bavi razvojem blisko komunicira sa funkcijom za obezbeđenje kvaliteta.



Slika 8. Faze razvoja i tipovi testiranja

Faza zahteva podrazumeva definisanje kriterijuma koje aplikacija treba da zadovolji. Zahtevi se mogu odnositi i na deo programa, funkcionalnost ili problem koji treba rešiti. Opšti dokument zahteva sadrži podatke o sistemu, dokumentaciji i problemu koji se posmatra. Poseban deo zahteva odnosi se na akcije koje treba preduzeti kao bi se zahtev zadovoljio tj. na rešenje problema. U kreiranju ovog dokumenta učestvuje profesionalni kadar koji je u bliskom kontaktu sa korisnicima.

Slika 9. Forma zahteva

Za vreme faze zahteva razvija se test plan. Plan obuhvata organizaciju test aktivnosti. Za vreme logičog, fizičkog i jediničnog programskog dizajna, razvija se test plan sa više detalja. U ovom periodu kreiraju se i test case-ovi.

Test case se definiše kao skup test podataka i test skriptova. ***Test skriptovi*** vode testera kroz test i obezbeđuju odvojena izvršavanja testa. Test uključuje očekivane rezultate koji potvrđuju da je test ispunio cilj. Test skriptovi se izvršavaju i rezultati se analiziraju za vreme aplikacionog testiranja.[10] Opšta forma test case-a tj. slučaja predstavlja dokument koji definiše ulaze, izlaze i scenario jednog prolaska kroz niz akcija u programu koje vode ka ostvarenju rezultata tj. specifičnog cilja.

Datum: _____ Tester: _____
Sistem: _____ Okruženje: _____
Cilj: _____ TestID: _____
Funkcija: _____ Broj zahteva: _____
Verzija: _____ Tip testiranja: _____

Opis test case-a:

_____

_____

Preduslovi:

_____

Scenario:

_____

_____

_____

_____

_____

_____

Očekivani rezultati:

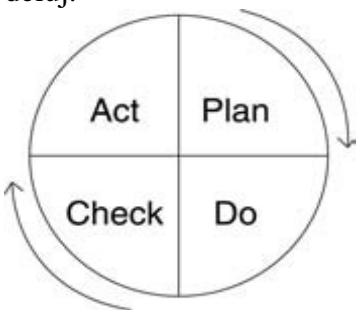
_____

Pass/fail kriterijum:

_____

Slika 10. Forma test case-a

U ***spiralnom pristupu***, sekvencijalni koraci mogu, sa određenim nivoom varijabiliteta, nedostajati ili biti različiti. Spiralni pristup podrazumeva krug: planiraj, radi, proveri i deluj.



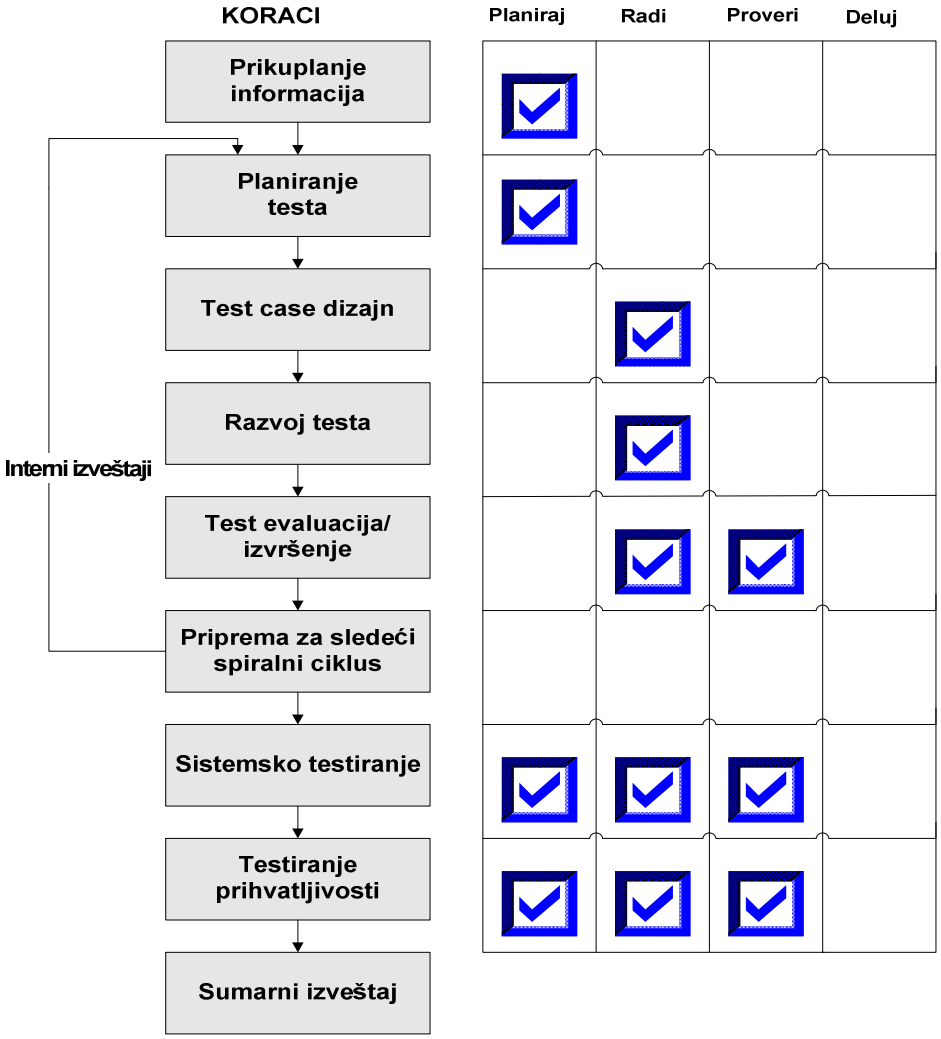
Slika 11. Spiralni model razvoja

1. ***Planiraj***: Korak razvoja test plana kao osnove za uspešno testiranje. Dobar test plan sadrži: uvod, celokupni prikaz plana, zahteve za testiranje, test procedure i detalje test plana. Ove stavke moguće je razbiti na poslovne funkcije, test scenarije i skriptove, funkcija/test matrice, očekivane rezultate, čeklistu test case-ova, izveštaje neslaganja, zahtevani softver, hardver, podatke, osoblje, test raspored, ulazne test kriterijume, izlazne kriterijume i sumarne izveštaje. Test plan je promenjiv dokument.

2. ***Radi***: Ovaj korak sastoji se iz dizajna test case-ova, razvoja testova i izvršavanje testova. Ovaj korak opisuje kako da se dizajniraju i izvrše testovi koji se nalaze u test planu. Dizajn uključuje funkcionalne testove, GUI testove, podelu sitema i testove prihvatljivosti. Kada se završi faza dizajna, počinje razvoj testova koji uključuje izradu test skriptova i procedura koje obezbeđuju detalje za svaki test case. Ovaj korak uključuje i postavku testa, regresiono testiranje novih i starih testova i beleženje otkrivenih defekata.

3. ***Proveri***: Korak uključuje metričke mere i analize. Važno je objaviti međuizveštaje testiranja. Uključeno je i beleženje rezultata i povezivanje sa test planom i ciljevima testa.

4. ***Deluj***: Ovaj korak uključuje pripremu za sledeću spiralnu iteraciju. Zahteva obradu funkcionalnih/GUI testova, test paketa, test case-ova, test skriptova, testova delova sistema i testova prihvatljivosti i izmene sistema za praćenje defekata i sistema verzije i kontrole, ako je to potrebno. Korak uključuje i određivanje mera za odgovarajuće radnje koje su povezane sa poslom koji nije izvršen po planu ili rezultate koji nisu očekivani. Primeri uključuju reevaluaciju test tima, test procedura i tehnoloških dimenzija testiranja.



Slika 12. Spiralna metodologija testiranja (Konstantno unapređenje procesa)

Nakon nekoliko spiralnih testiranja i potvrde aplikacije kao funkcionalno stabilne, počinje kompletno i testiranje prihvatljivosti sistema. Ovaj korak je često opcioni. Poslednja aktivnost u kontinualnom procesu unapređenja je sumiranje i izveštaji o rezultatima spiralnog testa.

5.7. QA

Veliki napredak informacionih tehnologija novog doba počeo je u poslednjoj dekadi dvadesetog veka. Na početku novog milenijuma mnoge IT kompanije su zatvorene ili su merdžovane i počelo je doba konsolidacije u industriji. Najveći stepen restruktuiranja

događao se u multinacionalnim kompanijama širom sveta. Poslednje tri godine su predstavljale period konsolidacije i jedna od dobrih stvari koja se iz tog perioda iznedrila odnosi se na značaj kvaliteta softvera.
Američki institut za kvalitet softvera(ASQ) i Institut za obezbeđenje kvaliteta(QAI) definisali su različite prakse osiguranja kvaliteta koje su usvojene od strane raznih organizacija. Mnoge organizacije formirale su grupe za obezbeđenje kvaliteta da bi unapredile i procenile softverske proizvode. Ove grupe u različitim organizacijama mogu igrati različite uloge i mogu realizovati planove kroz različite procedure. Softversko testiranje je u nekim organizacijama odgovornost ovih grupa. U drugim, testiranje je odgovornost razvojne grupe ili nezavisne organizacije.
Mnoge grupe za softverski kvalitet razvijaju planove kvaliteta, koji su slični test planovima.

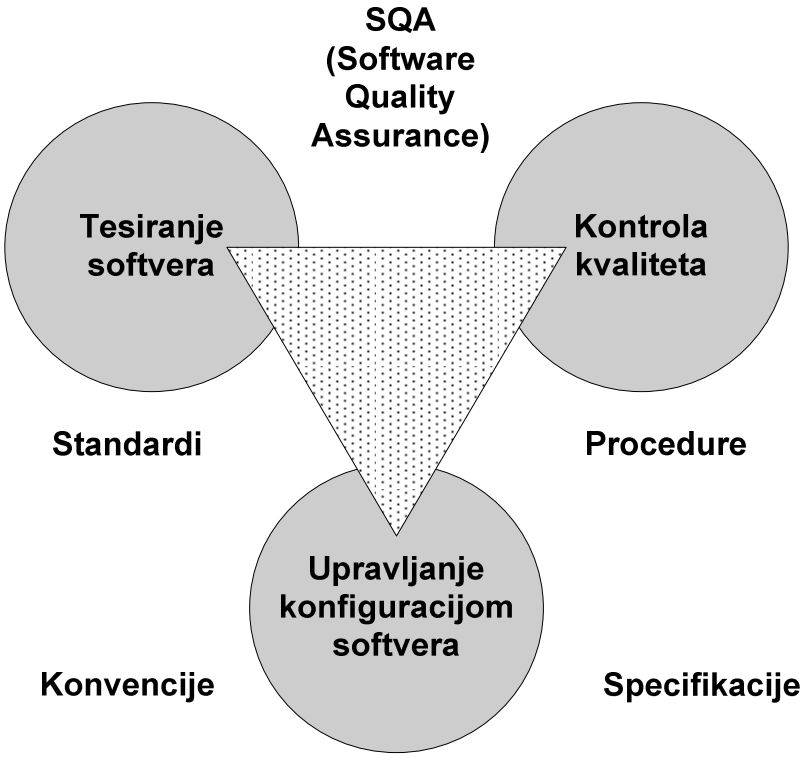
Osiguranje kvaliteta softvera (Software Quality Assurance)

Pojam SQA odnosi se na sistematizovane aktivnosti koje obezbeđuju dokaz podobnosti krajnjeg softverskog proizvoda za upotrebu. Postiže se korišćenjem prihvaćenih pravila za kontrolu kvaliteta za proveru integriteta i produžavanje života softvera. Veza između osiguranja kvaliteta, kontrole kvaliteta i funkcije provere softvera, kao i softverskog testiranja se često mešaju.
Obezbeđenje kvaliteta obuhvata skup podržanih aktivnosti potrebnih da bi se obezbedilo adekvatno poverenje u sprovedeni proces i stalno unapređenje u cilju stvaranja proizvoda koji zadovoljavaju specifikacije i odgovaraju svrsi. Kontrola kvaliteta je proces u kom se kvalitet proizvoda poredi sa primenjenim standardima i akcijom koja se preduzima kada se otkrije neodgovarajuća karakteristika proizvoda. Provera je ispitivanje koje verifikuje slaganje sa planovima, politikom i procedurama.

Obezbeđenje kvaliteta softvera je planiran napor da softver ispuni određene kriterijume i da ima dodatne specifične atribute npr. portabilnost, efikasnost, fleksibilnost. Kolekcija je aktivnosti i funkcija koje se koriste za praćenje i kontrolu projekta tako da specifični ciljevi budu postignuti sa željenim stepenom sigurnosti. Ovo nije odgovornost samo SQA funkcije, već i menadžera projekta, lidera projekta, osoblja i korisnika.

QA je funkcija odgovorna za vođenje kvaliteta. Reč ***obezbeđenje*** znači da menadžment može slobodno očekivati kvalitet proizvoda ako se prate procedure. SQA je strategija za upravljanje rizikom, jer je usko vezana za troškove.

Komponente SQA mogu se podeliti na testiranje softvera(verifikacija i validacija), upravljanje konfiguracijom softvera i kontrolu kvaliteta. Uspeh ovog procesa uveliko zavisi od kolekcije standarda, praksi, konvencija i specifikacija.



Slika 13. Komponente funkcije obezbeđenja kvaliteta

1. ***Testiranje softvera*** – određuje da li su zadovoljeni funkcionalni zahtevi. Testovi su jednako dobri kao i test case-ovi, ali se mogu preispitivati radi provere da li su svi testovi izvršeni sa svim mogućim ulazima i u svim mogućim stanjima sistema.

2. ***Kontrola kvaliteta*** – procesi i metode za nadgledanje posla i zadovoljenja zahteva.

3. ***Upravljanje konfiguracijom softvera*** – pronalaženje, praćenje i kontrolisanje promena softverskih elemenata sistema. Obuhvata identifikaciju komponenti sistema, kontrolu verzije, razvoj konfiguracije i kontrolu izmena. [10]

6. Praksa i primeri

Bug – *greška, otkaz, nedostatak u kompjuterskom programu koji dovodi do neželjenog ponašanja programa*

6.1. Kvalitet softvera u praksi

U najužem smislu kvalitet proizvoda se ogleda u što manjem broju „bug"-ova. Takođe se odnosi i na osnovni deo odgovaranja zahtevima kupaca, jer ako softver sadrži previše funkcionalnih anomalija, osnovni zahtevi neće biti zadovoljeni. Ova definicija se često izražava na dva načina, i to kao: stepen defekta (npr. broj defekata po milionu linija koda, po jedinici funkcije, ili drugoj jedinici) i pouzdanost (npr. broj otkaza po n sati operacije, srednje vreme otkaza ili verovatnoća operacija bez otkaza u određenom intervalu vremena). Zadovoljstvo kupaca se obično meri procentom zadovoljnih ili nezadovoljnih kupaca subjektivno gledano. Ovde se primenjuju tehnike intervjua gde onaj koji intervjuiše ne zna potrošača, a potrošač ne zna o kojoj se kompaniji ispitivača radi. Definicija kvaliteta koja u prvi plan postavlja kupca jako je važna u softverskoj industriji. Greške u zahtevima jedan su od najvećih problema u razvoju softvera. Prema Jones-u (1992) , 15 % i više svih softverskih anomalija su greške u zahtevima. Proces razvoja koji ne posvećuje pažnju kvalitetu zahteva svakako će se susresti sa problemom nekvalitetnog softvera.

Primer: IBM prati zadovoljstvo potrošača primenjujući nivoe CUPRIMDSO atributa (functionality – funkcionalnost, usability – korisnost, performance – performanse, reliability – pouzdanost, installability – instalabilnost, maintainability – održavanje, documentation/information – dokumentcija/informacija, service – servis i ostalo). Hewlett-Packard se fokusira na FURPS model (funkcionalnost, korisnost, pouzdanost, performanse i servis).
Ostale kompanije koriste slične dimenzije zadovoljstva kupaca. Juran ove atribute naziva parametrima kvaliteta ili parametrima pogodnosti u upotrebi.

Totalno upravljanje kvalitetom

TQM (Total Quality Management) je pristup razvijen 1985. godine kao opis japanskog stila rukovođenja za poboljšanje kvaliteta. Pojam je imao više značenja u zavisnosti od

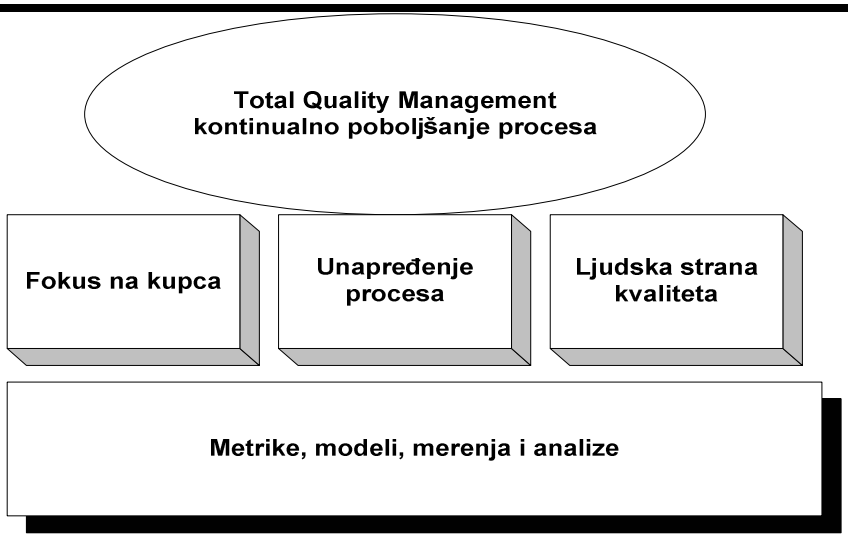
interpretacije i primene. Zapravo, predstavlja stil menadžmenta usmeren na dostizanje dugoročnog uspeha povezivanjem kvaliteta i zadovoljstva kupaca. Zasniva se i na kreiranju kulture u kojoj svi članovi organizacije učestvuju u unapređenju procesa, proizvoda i usluga. Od osamdesetih godina mnoge kompanije usvojile su TQM pristup kvalitetu.

***Primer**: Hewlett Packard uveo je TQC(Total Quality Control) pristup, Motorola strategiju Šest sigma i IBM Kvalitet vođen tržištem. HP TQC strategija se fokusira na ključne oblasti kao što su predanost menadžmentu, rukovođenje, fokus na kupca, ukupno učešće i sistemska analiza. Svaka oblast ima strategije i planove za poboljšanje kvaliteta, efikasnosti i odziva, sa konačnim ciljem dostizanja uspeha kroz zadovoljstvo kupca(Shores, 1989).*
Šest sigma Motorolina strategija fokusira se na dostizanje nivoa kvaliteta u cilju da se dostigne potpuno zadovoljenje kupca. Redukcija vremena ciklusa i participativni menadžment predstavljaju glavne korake ove strategije(Zimmer, 1989).
IBM strategija vođena tržištem ističe da kupac daje konačnu ocenu. Strategija obuhvata četri elementa: eliminaciju defekata, redukciju vremena ciklusa, zadovoljenje kupaca i poslovnih partnera i odanost Baldrige disciplini.

Bez obzira na varijacije u implementaciji, TQM sistem se može predstaviti kroz sledeće principe:

1. ***Fokus na kupca:*** Cilj je da se dostigne potpuno zadovoljstvo potrošača. Ovaj pristup podrazumeva proučavanje kupčevih želja i potreba, prikupljanje zahteva i merenje i upravljanje zadovoljstvom potrošača.
2. ***Proces:*** Cilj je da se smanje varijacije procesa i dostigne kontinuirano poboljšanje. Ovaj element uključuje i poslovni proces i proces razvoja proizvoda. Kroz poboljšanje procesa, dostiže se i kvalitet proizvoda.
3. ***Ljudska strana kvaliteta*****:** Cilj je kreiranje kulture kvaliteta. Ključne oblasti obuhvataju liderstvo, predanost upravljanju, ukupno učešće, unapređenje zaposlenih i druge socijalne, psihološke i ljudske faktore.
4. ***Merenje i analiza:*** Cilj je da se sprovede kontinuirano napredovanje svih parametara kvaliteta uz pomoć ciljno orijentisanog sistema merenja.

Potrebno je da organizacija koja podržava TQM pristup ima izvršno liderstvo, da se fokusira na infrastrukturu, trening i edukaciju, kao i na strateško planiranje kvaliteta.



Slika 14. Ključni elementi TQM pristupa

Razni organizacioni okviri podržavaju TQM filozofiju. Neki od primera su Planiraj-Radi-Proveri-Deluj pristup (Deming), Paradigma poboljšanja kvaliteta (Basili), SEI (Software Engineering Institute) model zrelosti (Humphrey) i Oslanjanje na menadžment preduzeća (Womack).[2]

Modeli procesa razvoja softvera

Do danas su razvijeni mnogi modeli procesa razvoja softvera. Modeli se koriste pojedinačno ili u kombinaciji i teže ostvarenju određenog nivoa kvaliteta i ponovljivom i stabilnom procesu proizvodnje softvera. Mnogi od njih su već pomenuti i objašnjeni:

- Model vodopada,
- Prototipski pristup i njegove varijacije,
- Spiralni model,
- Model procesa iterativnog razvoja,
- Objektno orijentisani proces razvoja,
- Cleanroom metodologija,
- Proces prevencije defekata i dr.

Procesi variraju prema stepenu implementacije u različitim organizacijama, pa čak i po projektima. Prema okviru određenog procesa razvojni tim obično definiše specifične elemente kao što su implementacione procedure, metode i alate, metrike i merenja i td. Neki modeli procesa odgovaraju tačno određenim tipovima projekta u određenom okruženju, ali uspeh projekta zavisi pre svega od zrelosti implementacije. Dimenzije povezane sa celokupnim sistemom upravljanja kvalitetom jako utiču na rezultate softverskih projekata. Za utvrđivanje zrelosti procesa organizacije ili projekata

primenjuju se različite metode: SEI (Software Engineering Institute) metode za utvrđivanje zrelosti Istraživanja softverske produktivnosti (SPR), Melcom Baldrige disciplina i procesi, i ISO 9000 proces registracije. SEI i SPR metode se odnose na softverski proces, a ostala dva okvira su procesi kvaliteta i standardi upravljanja kvalitetom.

6.2. Najbolje prakse u testiranju softvera

Testiranje softvera je sve zastupljenije u savremenim organizacijama koje se bave razvojem softvera. U IBM istraživanju Centra za softverski inženjering opisuju se najbolje prakse u softverskom testiranju. Neke od njih podrazumevaju alate za testiranje, ali su uglavnom prakse u pravom smislu te reči. Istraživanje je obuhvatilo više softverskih organizacija koje su primenjivale ove prakse i prihvatile iste kao ključ uspeha.

Kolekcija praksi ima mnoge izvore. Neke potiču iz literature, neke iz praktične primene i onog što se u praksi ceni. Prakse su podeljene na osnovne, utemeljene i inkrementalne.

Osnovne prakse predstavljaju korake na kojima se sve zasniva i koji kada se jednom savladaju postaju dobra osnova za dalji razvoj. U osnovne prakse se ubrajaju:

1. ***Funkcionalne specifikacije*** – predstavljaju ključni deo mnogih procesa razvoja koji se pojavio sa Modelom vodopada. Često opisuju potrebe korisnika sitema, koje funkcije su neophodne, kao i zahtevane karakteristike ulaza i izlaza. Funkcionalne specifikacije predstavljaju dokumentaciju koja opisuje zahtevano ponašanje sistema. Testeri ih koriste za zapisivanje test case-ova na nivou black-box testiranja. Prednost posedovanja funkcionalne specifikacije je da se generisanje testova može odvijati paralelno sa razvojem koda. Na ovaj način se ostvaruje paralelizam izvršenja i uklanjaju uskih grla u razvojnom procesu, jer su testovi gotovi kada se završi pisanje koda. Takođe, na ovaj način se forsira određeni nivo jasnosti iz perspektive dizajnera i arhitekte koji je neophodan za efikasan razvoj. Konačno, funkcionalne specifikacije postaju dokumentacija koju je moguće deliti sa korisnicima, kako bi uvideli još jednu dimenziju onog što se razvija.

2. ***Pregledi i inspekcije*** – Softverske inspekcije je izmislio Mike Fagan sredinom sedamdesetih godina u IBM-u i smatra se za jednu od najefikasnijih metoda debagovanja koda. Danas, 20 godina kasnije, postoji literatura, alati i konsalting organizacije koje se bave praksom softverskih inspekcija.

3. ***Formalni ulaz i izlazni kriterijum***- Ovaj pojam potiče iz evolucije Modela vodopada i modela zvanog ETVX(Entry criteria, Tasks, Verifications and Validations, Exit criteria). Ideja je da svaki korak procesa ima precizan ulaz i precizan izlazni kriterijum. Kriterijumi se definišu u toku procesa razvoja i nadgledaju od strane menadžmenta.

4. ***Funkcionalni test – varijacije*** – Većina funkcionalnih testova napisana je kao black- box test na osnovu funkcionalne specifikacije. Broj generisanih testova obično je varijacija ulaznih parova i izlaznih uslova. Varijacija se odnosi na određenu kombinaciju ulaznih, za dostizanje određenih izlaznih uslova. Pisanje funkcionalnih testova uključuje i pisanje različitih varijacija, kako bi se pokrilo što više stanja programa. Najbolja praksa uključuje razumevanje načina pisanja varijacija i kreiranja adekvatnog pokrića za test funkcije.

5. ***Višeplatformsko testiranje*** – Mnogi proizvodi dizajnirani su da funkcionišu na različitim platformama koje predstavljaju dodatni teret pri lansiranju i testiranju proizvoda. Kada se kod prenosi sa jedne platforme na drugu neophodne su modifikacije kako bi se unapredile ili održale performanse.

6. ***Interne bete – Beta verzija*** je prva verzija programa koja je lansirana van organizacije ili zajednice koja razvija softver u cilju procene ili black/gray-box testiranja. Ideja ***bete*** je da se proizvod isporuči ograničenom broju kupaca i da se dobije povratna veza, kako bi se ispravili problemi širih razmera. Za veće kompanije, mnogi proizvodi se koriste interno, na ovaj način formirajući tzv. beta publiku. Tehnike koje podržavaju interne beta testove povećavaju efikasnost korišćenja unutrašnjih resursa. Reduciraju se i troškovi i izdaci u odnosu na eksternu betu.

7. ***Automatsko izvršavanje testa*** - Cilj automatskog testa je da minimizira količinu manuelnog rada koji se utroši pri izvršavanju testa i rezultira većom pokrivenošću tj. većim brojem test case-ova.

8. ***Beta programi –*** odnose se na interne beta verzije.

9. ***Noćni bildovi –*** obično predstavljaju verzije koje se automatski kreiraju od strane kontrolnog i sistema za buildove, obično preko noći, omogućavajući testerima da odmah testiraju izmene. Koncept noćnih bildova je u modi već duže vreme. Podrazumeva učestale buildove, za razliku od onih koji se rade na dnevnom nivou. Prednost je da ako se dese veće regresije zbog napravljenih grešaka, one se brzo otkrivaju, a noviji release-ovi softvera dostupni su ranije i testerima i developerima.

Utemeljene prakse obuhvataju:

1. ***Korisničke scenarije –*** Integrisanjem više softverskih proizvoda i kreiranjem korisničkih aplikacija zadatak testiranja karakteristika za krajnje korisnike se komplikuje. Jedna od vodećih metoda testiranja je razvoj korisničkih scenarija koji obuhvataju funkcionalnost aplikacija. Opšteprihvaćen naziv je ***korisnički scenariji***. Prednost scenarija je u tome da testiraju proizvod na način koji u najvećoj meri odslikava korišćenje od strane kupaca. Jedna od prednosti je i smanjivanje kompleksnosti pisanja test case-ova fokusiranjem na scenarije koji obuhvataju karakteristike aplikacije. Međutim, metodologija razvoja korisničkih scenarija i korišćenja istih kao pokrića na funkcionalnom nivou, ostaje težak zadatak. Ova praksa trebalo bi da sadrži metode snimanja korisničkih scenarija i razvoja test case-ova koji su na njima bazirani.

2. ***Testiranje korisnosti –*** Za većinu proizvoda važi da je korisnost konačni arbitar kvaliteta. Ovo važi za veliki broj desktop aplikacija koje su uzele učešće na tržištu obezbeđivanjem dobrog korisničkog ugođaja. Testiranje korisnosti(usability testing) ne daje samo informacije o tome koliko je proizvod koristan, nego i obezbeđuje povratnu vezu ka metodama koje poboljšavaju korisničko iskustvo i tako stvaraju pozitivnu sliku kvaliteta. Ova praksa mora sadržati i znanja o napretku u oblasti korisničkog interfejsa.

3. ***Unutarprocesne ODC povratne sprege –*** Ortogonalna klasifikacija defekata(ODC) je metod za merenje koji koristi tok defekta da obezbedi preciznu merljivost u okviru proizvoda ili procesa. Jedna od upotreba ODC-a pružala je mogućnost da se zatvore povratne sprege u razvojnom procesu softevera, što je tradicionalno bio težak zadatak. I dok ODC može da se koristi za različite softverske menadžment metode, zatvaranje povratnih sprega je proteklih godina mnogo zahtevniji proces poboljšanja i kontrole troškova.

4. ***Višestruki release ODC/Leptir profili*** - Ključna karakteristika ODC merenja je mogućnost da se vide višestruki release procesi i razvije profil korišćenja i odredi uticaj garantnih troškova. Ova tehnologija analize omogućava menadžeru proizvodnje da kreira odluke strateškog razvoja, kao što je optimizacija troškova razvoja, tržišta i kvaliteta za prepoznavanje potrošačkih trendova, korisničkih šema i proizvodnih performansi.

5. ***Zahtevi za planiranje testa*** - Jedna od uloga softverskog testiranja je da obezbedi da proizvod zadovolji zahteve klijenata. Na ovaj način, primanje zahteva postaje osnovni deo, ne samo razvoja, već i kreiranja test planova koji mogu da ispituju da li će proizvod zadovoljiti zahteve kupaca. U manjim organizacijama za razvoj softvera često je zadatak upravljanja zahtevima vezan za ono šta treba da se razvija, naspram onog šta je potrebno na tržištu. Upravljanje zahtevima i deo kreiranja test planova predstavlja jako važan korak.

6. ***Generisanje automatskog testa*** - Skoro 30% zadataka u testiranju može da se odnosi na pisanje test case-ova. Ovaj zadatak je potpuno manuelan i prvi je kanditat za uštedu kroz automatizaciju. Tehnologija automatizacije nije se razvijala onoliko brzo koliko je potebno. Alati za automatsko generisanje testova često proizvode prevelik skup testova, stavljajući u senku dobre strane automatizacije. Postoji nekoliko tehnika i alata koje su prepoznatljive kao dobre metode za automatsko generisanje test case-ova. Praksa treba da razume koji je od ovih metoda uspešan i u kojim su okruženjima efikasne.

Inkrementalne prakse obuhvataju:

1. ***Povezivanje testera i developera –*** Poznato je da povezivanje testera i developera unapređuje i test case-ove i kod koji se razvija. Ekstremni slučaj ove prakse je Microsoft, gde svaki developer ima svog testera. Ova praksa obuhvata razgraničavanje koje vrste povezivanja imaju efekta, i u kojim se okruženjima mogu upotrebiti.

2. ***Pokriće koda –*** Koncept se zasniva na strukturnoj dimenziji koda. Pokriće koda odnosi se na numeričko merenje koje određuje elemente koda koji su testirani. Postoje mnoge metrike: izrazi, grane i podaci na koje se odnosi ovaj pojam. Postoji i nekoliko alata koji pomaži merenju i obezbeđuju dodatnu pomoć da se odrede elementi koji nisu obuhvaćeni. Ova oblast je takođe predmet debate već nekoliko decenija. Ova praksa, dakle, treba da sadrži informacije o alatima i metodama koje se sprovode pri testiranju koda i prate rezultate pozitivnih iskustava.

3. ***Automatski generator okruženja(Drake) –*** Jako zahtevan deo u smislu vremenskih resursa je postavljanje testnih okruženja kako bi se sprovodilo testiranje. Ovo vreme može biti još duže ako postoji više operativnih sistema, verzija i koda koji se izvršava na više platformi. Ovo igra veliku ulogu u sistemskom testiranju. Alati koji mogu automatski postaviti okruženja, izvršavati test case-ove, snimati rezultate i potom automatski preći na novo okruženje, imaju jako veliku vrednost. IBM je razvio alat pod nazivom DRAKE koji precizno radi. Ova praksa obuhvata probleme, alate i tehnike koji su povezani sa postavkom okruženja, padovima okruženja i automatskim pokretanjem test case-ova.

4. ***Testiranje u cilju pomaganja isporuci na zahtev –*** Ova praksa je ideja Microsofta u kojoj se posmatra test proces kao nešto što omogućava kasne promene i prilagođavanje zahtevima tržišta. Ona na neki način menja ulogu testiranja u onu koja obezbeđuje dobru regresivnu sposobnost i rad sa kasnim izmenama koje još uvek ne narušavaju proizvod i planirani izlazak na tržište. Ubraja se u filozofska gledišta testiranja, koji postavlja testiranje u novu dimenziju. Praksa mora identifikovati kako da se razradi ovaj koncept u organizacijama i sa proizvodima na specifičnim tržištima. Može se primeniti u elektronskoj trgovini, gde je veća i interakcija kupaca i veći kompetitivni pritisak.

5. ***Dijagram stanja zadataka(Tucson) –*** Obuhvata funkcionalne operacije aplikacije ili modula u obliku dijagrama prelaza stanja. Prednosti ovog oblika su da omogućava automatsko kreiranje test case-ova ili kreiranje metrika koje su bliže funkcionalnoj dekopoziciji aplikacije. Postoje i alati koji podržavaju ovu praksu. Problemi su najčešće dobijanje funkcionalnog izgleda proizvoda koji ne postoji u kompjuterizovanoj ili dokumentovanoj formi i kreiranje dijagrama prelaza stanja. Jedan od alata Test Master zaista koristi ove dijagrame za generisanje funkcionalnih testova.

6. ***Simulacija otkaza memorijskih resursa –*** Praksa se odnosi na određeni bug, gubitak memorije zbog lošeg upravljanja heap-om ili nedostatka garbage collection-a. Ovo je veliki problem za mnoge C programe i Unix aplikacije. Postoji i na drugim platformama i jezicima. Postoje komercijalni alati koji simuliraju pad memorije i proveravaju memorijske nedostatke. Praksa obuhvata generisanje i razvoj metoda i tehnika za upotrebu različitih platformi i jezičkih okruženja.

7. ***Statističko testiranje(Tucson) –*** Koncept statističkog testiranja obuhvata ideju da se, umesto metoda za debagovanje, koristi testiranje kako bi se postigla pouzdanost softvera. Postoje mnogi argumenti koji govore u prilog ovoj metodi. Statističko testiranje odnosi se na razrađivanje softvera u operativnom smislu i potom merenje međuotkaza koji se koriste da procene pouzdanost. Dobar proces razvoja treba da doprinosi povećanju srednjeg vremena među otkazima svaki put kada se ispravi bag. Nakon toga ovaj proces postaje release kriterijum i izvor uslova za zaustavljanje testiranja.

8. ***Poluformalne metode –*** Formalne metode u softverskom inženjeringu traju nekoliko decenija. Ključni koncept formalne metode je posezanje za verifikacijom programa umesto testiranja i debagovanja. Metode verifikacije varijabilne su: neke od njih su dokazi teorema, dok neke određuju koji izrazi mogu biti validirani. Vizija formalnih metoda je oduvek bila takva da, ako je specifikacija softvera dobra, ona može dovesti do automatskog generisanja koda, koji zahteva minimum testiranja. U praksi se raspravljalo o opsegu semiformalnih metoda koji industrija ignoriše. Semiformalna metoda je takva metoda u kojoj specifikacija može biti u obliku dijagrama prelaza stanja ili tabela koji se mogu koristiti i za generisanje testova.

9. ***Testovi provere koda –*** Ova ideja odnosi se na spregu automatskog test programa (obično regresionog testa) sa kontrolnim sistemom promena. Microsoft je koristio ovakve sisteme. Na ovaj način se omogućava pokretanje automatskog testa na tek izmenjenom kodu, tako da su šanse za neuspelost builda zbog koda svedene na minimum. Kod Microsofta se neguje praksa da dok god kod ne prođe test, ne prolazi ni sledeći build.

10. ***Minimiziranje regresije test case-ova –*** U organizacijama koje imaju istoriju razvoja i proizvode koji sazrevaju tokom mnogih release etapa, nije neobično uočiti skup regresionih testova koji je jako velik. Negativne posledice ovolikog skupa testova su predugo trajanje naspram male dodatne vrednosti. Postoji nekoliko metoda koje umanjuju regresione testove. Jedna od njih posmatra pokriće proizvedenog koda i svodi test caseo-ve na minimalan broj. Ovakva metoda, mada atraktivna, meša strukturnu metriku sa funkcionalnim testom. Ipak, ona je način implementacije minimiziranja.

11. ***Implementirane verzije za MTTF[1]-*** Prednosti koju beta programi omogućavaju su da pojedinac dobija veliki uzorak korisnika za test proizvoda. Ako je proizvod napravljen tako da se otkazi snimaju i vraćaju prodavcu, on će obezbediti odličan izvor merenja srednjeg vremena među otkazima softvera. Postoji nekoliko upotreba ove metrike. Prvo, može se koristiti kao mera kvaliteta softvera u pogledu koji je kupcu značajan. Drugo, omogućava merenje srednjeg vremena među otkazima istog proizvoda pod različitim profilima korisnika ili grupe korisnika. Treće, može biti podrška za uočavanje prvog otkaza podataka koji može doprineti dijagnozi i rešavanju problema. Microsoft je potvrdio da sprovodi najmanje prva dva principa u svojim beta verzijama.

12. ***Benchmark trendovi –*** Benchmarking je širok koncept koji se primenjuje na mnoge discipline u različitim oblastima. U svetu softverskog testiranja može se interpretirati kroz tehnike i performanse metoda testiranja koje su iskusili drugi developeri softvera.

13. ***Bug prednosti-*** Ovaj metod se koristio u Microsoftu i IBM-u. Prednosti bagova se odnose na inicijative koje menjaju organizaciju u smislu fokusa na otkrivanje softverskih grešaka. Ponekad uključuju i nagrade. Iskustvo potvrđuje da ovakav napor teži da identifikuje veći od uobičajenog broja bagova. Potrebni su i dodatni resursi za ispravku. Konačan rezultat je veći kvalitet proizvoda.[1]

6.3. Alati za testiranje softvera

Pored raznih tehnika i metoda testiranja softvera razvijeni su i mnogi alati koji olakšavaju ili potpuno automatizuju proces testiranja. Gotovo da je nemoguće učiniti proces testiranja jasno definisanim i ponovljivim bez prednosti koje alati za testiranje nude. Osnovna podela testiranja podrazumeva manuelno i automatsko testiranje. Naravno, manuelno testiranje se ne odigrava u potpunosti ručno. Oba oblika testiranja koriste alate koji omogućavaju efikasnije testiranje.

[1] mean time to failure (srednje vreme otkaza)

Razvoj automatskih alata za testiranje je reakcija na napredovanje web baziranih, klijent/server i alata za razvoj softvera koji su omogućili ubrzani razvoj aplikacija sa znatno boljim funkcionalnostima. Sektori za testiranje dolaze u dodir sa softverom koji je jako unapređen, ali i kompleksan. Novi alati za testiranje se razvjaju kao podrška procesu obezbeđenja kvaliteta.

Postoji nekoliko kategorija alata po ciljevima i karakteristikama:

- ***Alati funkcija/regresija*** – pomažu u testiranju grafičkog korisničkog interfejsa. Neki pomažu i kod drugih tipova interfejsa. Npr. web test alati koji testiraju kroz pretraživač(capture/replay) alati.

- ***Alati za test dizajn/podatke*** – pomažu stvaranju test case-ova i generisanju test podataka.

- ***Load/performance alati*** – alati za test opterećenja i performansi.

- ***Test proces/menadžment alati*** – pomažu organizovanje i izvršavanje paketa testova na nivou komandne linije, API-ja ili protokola. Neki alati imaju grafički korisnički interfejs, ali nemaju nikakvu specijalnu podršku za testiranje proizvoda koji imaju izvorni GUI.

- ***Unit test alati*** – ovi alati, okviri i biblioteke podržavaju unit testiranje, koje se obično izvršava od strane developera, obično korišćenjem interfejsa ispod javnog.

- ***Alati za test implementacije*** – pomažu testiranju u vreme izvršavanja.

- ***Alati procene testa*** – pomažu procenu kvaliteta testova. Uključuju alate za pokriće koda.

- ***Statistički test analajzeri*** – analiziraju programe bez pokretanja istih. Metrički alati spadaju u ovu kategoriju.

- ***Alati za upravljanje defektima*** – alati koji prate softverski proizvod i defeke i upravljaju zahtevima za njihovo poboljšanje. Obuhvataju stanja defekta od otkrića do zatvaranja problema.

- ***Alati za praćenje performansi/usaglašenosti aplikacije*** – mere i maksimiziraju vrednosti u IT životnom ciklusu isporuke, kako bi obezbedili da aplikacije zadovolje nivo kvaliteta, performanse i ciljeve dostupnosti.

- ***Alati runtime analize*** – analiziraju programe dok su pokrenuti.

6.4. Testiranje softvera sa Visual Studio 2005 Team System-om

Kako je Visual Studio 2005 moćan alat za razvoj softvera koji se danas uveliko koristi i ima brojne prednosti, prosto je nemoguće ne pomenuti i njegovu drugu stranu koja je vezana za testiranje softvera. VS 2005 danas predstavlja moćan alat za kreiranje manuelnih i automatskih testova. Predstavlja paket alata koji su neophodni softverskim dizajnerima, developerima, test inženjerima, program menadžerima i svima na kojima se temelji projekat. VSTS ne samo da sadrži neophodne alate, već omogućava njihovu integraciju i neometan rad.

Visual Studio Team System teži da obezbedi alate i tehnologiju koja pokriva celokupan životni ciklus proizvoda – praćenje bugova i posla, kontrolu izvornog koda, automatski build sistem, testiranje, arhitekturu itd. Sastoji se od dela namenjenog razvoju i dela za testiranje softvera. Deo koji se odnosi na testiranje softvera omogućava kreiranje i izvršavanje testova, sa mogućnošću objavljivanja i deljenja rezultata sa članovima tima. Ovu prednost obezbeđuje zahvaljujući funkcionalnosti izveštavanja Team Foundation Servera(TFS), koji omogućava kreiranje izveštaja koji pokrivaju sve aspekte podataka na TFS-u – Source Controle provere (broj dokumenata, broj promenjenih linija, promene uopšte, pregled koda), promene radnih elemenata, pokriće koda i rezultate testova.

VSTEST se fokusira na nekoliko različitih tipova testova. Neki do njih koriste testerima koji se fokusiraju na rad developera(unit i web testiranje), dok drugi koriste kada u centru nije developer(load i manuelno testiranje).

Unit testiranje: VSTEST omogućava kreiranje test koda dodavanjem odgovarajućih atributa metodama i klasama koje se uključuju u test. Podržani su jezici: Visual Basic, C# i C++/CLI. Ostali jezici su takođe podržani, mada ne postoji IDE podrška za pokretanje testova npr. Visual J#, u kome se mogu kreirati testovi, ali se za izvršavanje moraju koristiti alati komandnih linija.

Web testiranje: Tehnologija koja omogućava implementiranje funkcionalnosti za testiranje web-a. Testiranje koje VSTEST obezbeđuje nije web testiranje bazirano na pretraživaču – neće pratiti kliktanje mišem kroz HTML i skript, ali je i pored toga dobro rešenje za testiranje weba, koje obezbeđuje snimanje (snimanje navigacije među stranicama, prilikom popunjavanja formi) i skup kompleksnih dodataka za validacuju podataka, kao i operativnost na nivou baze.

Load testiranje: Ovaj tip testa je kontejner tip. Sadrži mnoge druge testove(bilo kog tipa) i omogućava korisniku da podesi parametre za izvršavanje (broj korisnika, vremenske varijacije). Na ovaj način je omogućena velika fleksibilnost kreiranja testova.

Manuelno testiranje: Ovaj tip testiranja je na neki način ograničen, jer je jedino obezbeđen prost tekstualni dokument koji sadrži korake koji treba da se izvrše, zajedno sa pass/fail rezultatima i komentarima korisnika. Korisno je za male projekte, ali nije baš odgovarajuće za upravljanje velikim brojem test case-ova koji treba da se izvrše od strane različitih korisnika, koji treba da prijavljuju bug-ove, probleme i beleže druge informacije koje su važne za test. Postoje dva tipa manelnih testova u VSTEST-u: tekst bazirani i Microsoft Word bazirani testovi. Prvi su u prostom tekst formatu, dok drugi podrazumevaju kreiraje testova u MSWord-u, koji može da se eksportuje u HTML.

Generičko testiranje: Jedan od problema koji je VSTeam Test razvojni tim iskusio za vreme razvoja bio je zahtev korisnika da imaju mogućnost integrisanja postojećih test elemenata u VSTeam System. Ovo je bio potpuno logičan zahtev, naročito za one korisnike koji su hteli da pređu na ovaj sistem bez gubitka postojeći testova. Rešenje su generički testovi. Ovi testovi prosto pozivaju komande vezane za specifično okruženje i čekaju da se proces završi, i u zavisnosti od vraćenog koda određuju kriterijum prolaznosti testa.

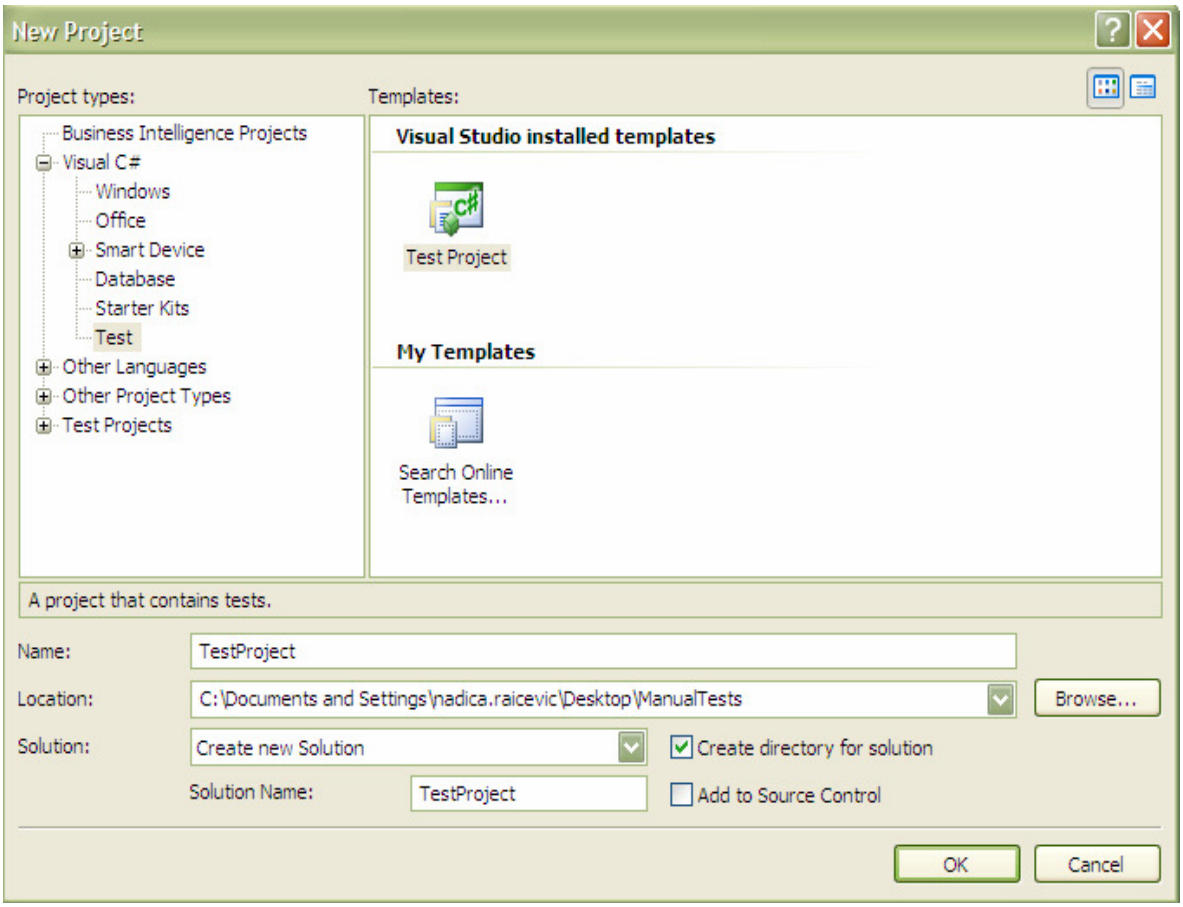
Uređeno testiranje: Opšti pristup koji VSTEST ima kada je reč o redosledu izvršavanja testova je „nedefnisan“(testovi se izvršavaju različitim redosledom u različitim etapama). Međutim, postoje situacije kada je neophodno da se testovi izvršavaju tačno određenim redosledom npr. unit test baze, skladištene procedure ili pristup podacima. Sistem omogućava definisanje sopstvenog redosleda i razne prednosti vezane za pass/fail kriterijume testova.

Pokriće koda: Ova funkcionalnost omogućava pregledanje delova aplikacije koji su obuhvaćeni automatskim ili manuelnim testovima. Takođe obezbeđuje informacije po linijama koda koji su bili izvršavani, i na ovaj način omogućava uvid u informacije koji su testovi dovoljno puta izvršavani, a koji ne.

Prednosti VSTEST-a

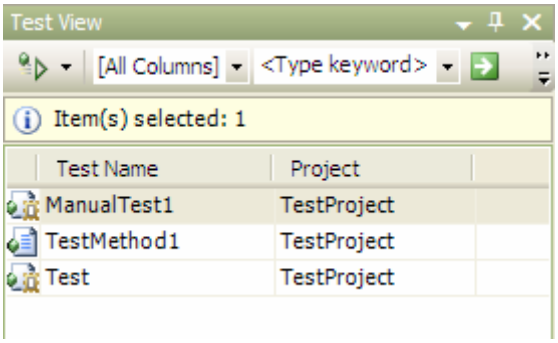
Prednost VSTEST-a u odnosu na druge alate je integracija sa Visual Studio-om kroz test projekte, test poglede, test menadžer i test rezultat prozore.

Test projekat je najvažniji element kada se radi o upravljanju testovima. Predstavlja nešto slično klasi biblioteka projekta, a jedina razlika je što umesto koda sadrži testove.



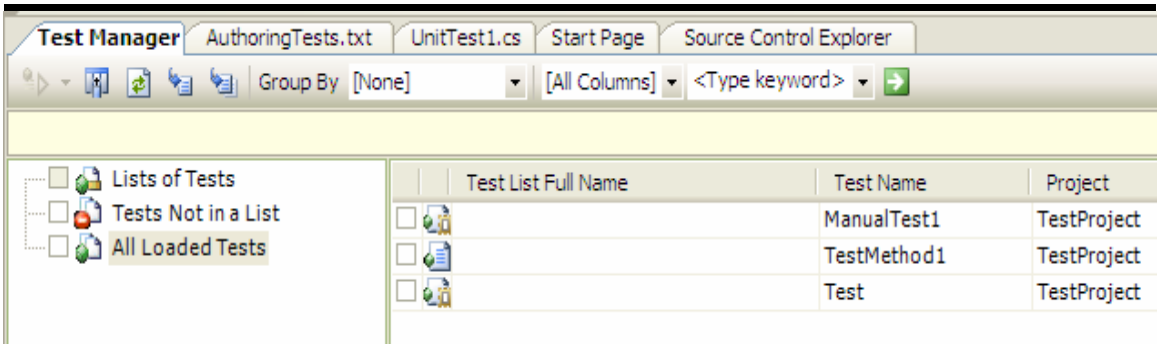
Slika 15. Novi test projekat

Test pogled je prozor za pregled i pokretanje testova. Omogućava listu svih trenutno raspoloživih testova. Duplim klikom otvara se test u editoru. Koristi se za debagovanje, grupisanje, pokretanje i filtriranje liste testova.



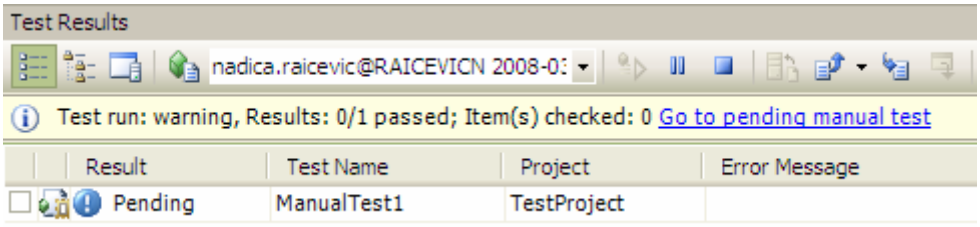
Slika 16. Test pogled

Test menadžer predstavlja deo koji se bavi kategorizacijom testova, korišćenjem liste testova i metapodataka.



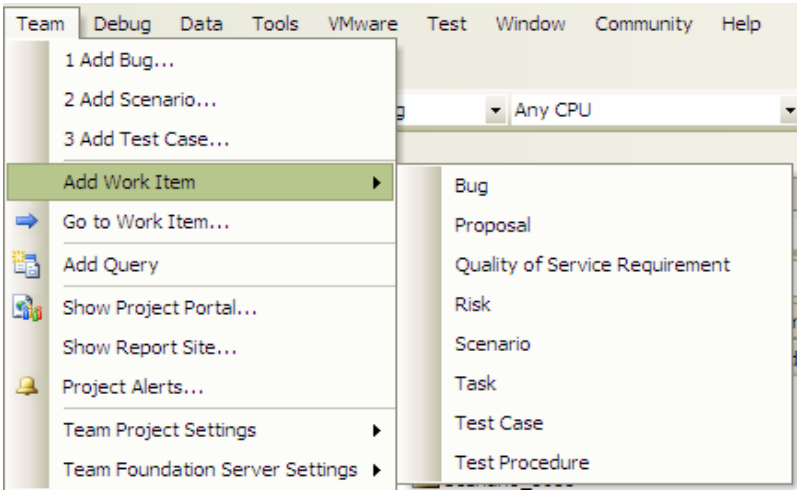
Slika 17. Test menadžer

Test razultati se prikazuju u posebnom prozoru. Omogućava se lista i kategorizacija, sa filter/grupisanje funkcionalnošću iz Test pogleda i Test menadžera. Moguće je ponovo pokretati, pauzirati i stopirati testove, kao i pregledati rezultate koji već postoje.



Slika 18. Test rezultati

Prednost je i integracija sa TFS-om. Na ovaj način se omogućava dostupnost podataka i brza reakcija kada se u izvršavanju uoče greške. Bug se kreira iz test rezultata, desnim klikom na test rezultat i odabiranjem opcije „Kreiraj work item“. Otvara se forma za kreiranje bug-a na TFS-u, sa predefinisanim poljima: naziv, opis, detalji greške i rezultati bug-a. Na ovaj način, bilo ko može pregledati bug-ove i dobiti neophodne informacije, kako bi istraživao problem, bez ponovnog pokretanja testa.

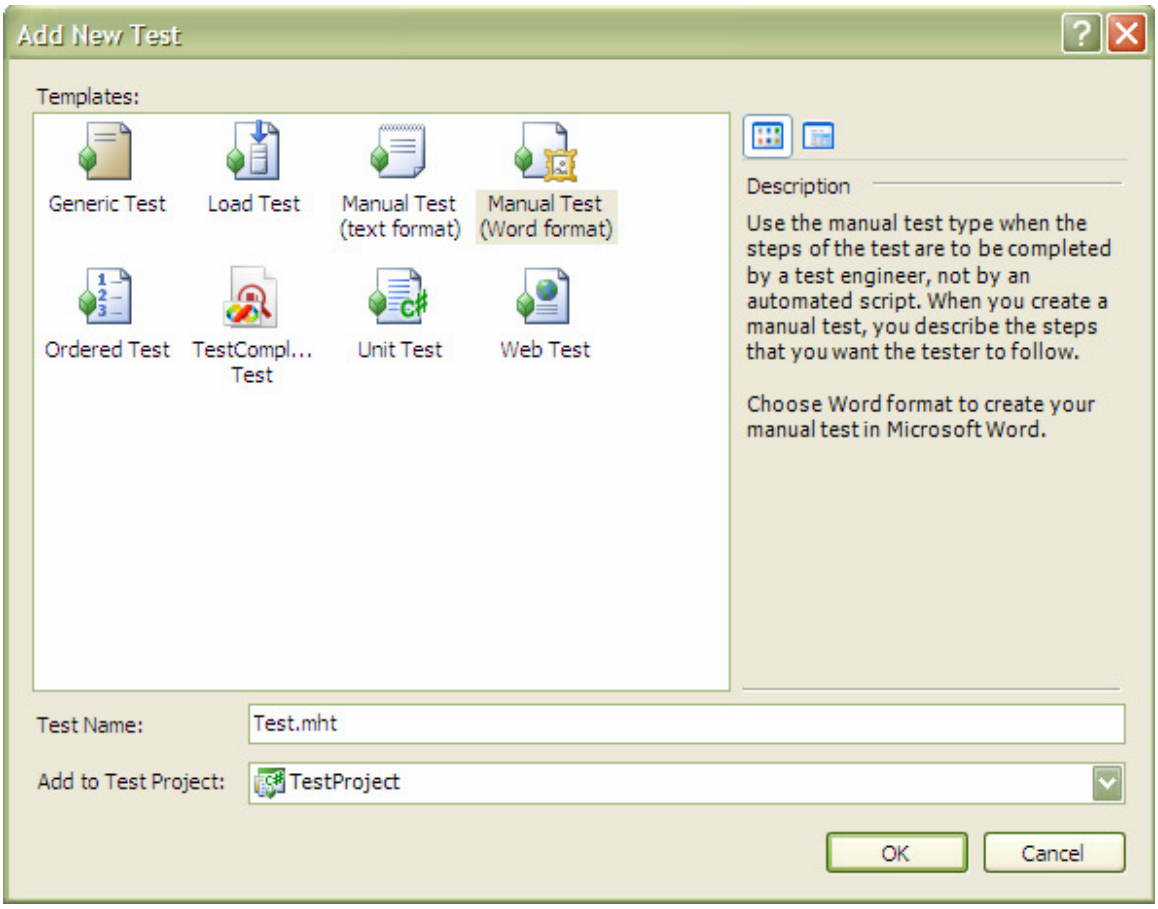


Slika 19. Dodavanje novog work item-a

Moguće je objavljivati rezultate testova na TFS-u. Rezultati se vezuju za bulid. Moguće je vezati neuspele test case-ove za bug-ove i pratiti njihovu povezanost. Na ovaj način vidi se istorija (broja test case-ova, stepena uspešnosti, kretanja i pokrivenosti koda testovima), koja omogućava kreiranje izveštaja koji su neophodni u procesu upravljanja testiranjem. [8]

Primer kreiranja manuelnog testa u VSTEST-u

Kreiranje novog manuelnog testa predstavlja niz akcija u VS i manuelno dodavanje podataka koji nedostaju i koje mora uneti tester. U pozadini se nalaze predefinisane forme za svaki tip testa.



Slika 20. Test tipovi

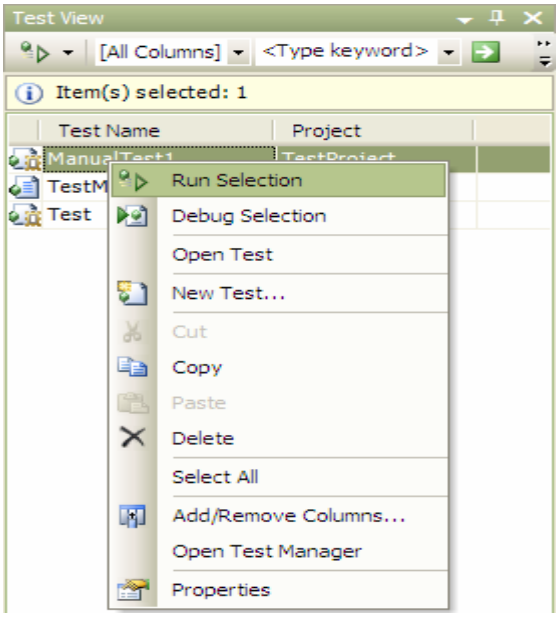
Korisnik odabira tip testa Manual test(Word format) i unosi naziv dokumenta, kao i projekat kome test pripada i potvrđuje unos. Na ovaj način kreira se dokument u koji se unose detalji testa.

Test #
001
Test
Test pregleda stampe izvoda.
Initial conditions and background
1. Korisnik ulogovan na aplikaciju. 2. Korisnik ima izvode za dati racun.
Test Procedure Steps
List the exact steps you take to perform the test, test case by test case. Each step has two parts: a user intention and system response:

Step No.	User Input	System Response
1.	Korisnik odabira racun iz liste.	Otvara se stanje racuna.
2.	Korisnik aktivira kontrolu Izvodi.	Prikazuje se lista izvoda.
3.	Korinik oznacava izvod iz liste.	Kursor se pozicionira na izvod.
4.	Korisnik aktivira kontrolu Stampaj.	Sistem prikazuje stampu izvoda.
5.	Korisnik proverava detalje izvoda.	Sistem u istom stanju.

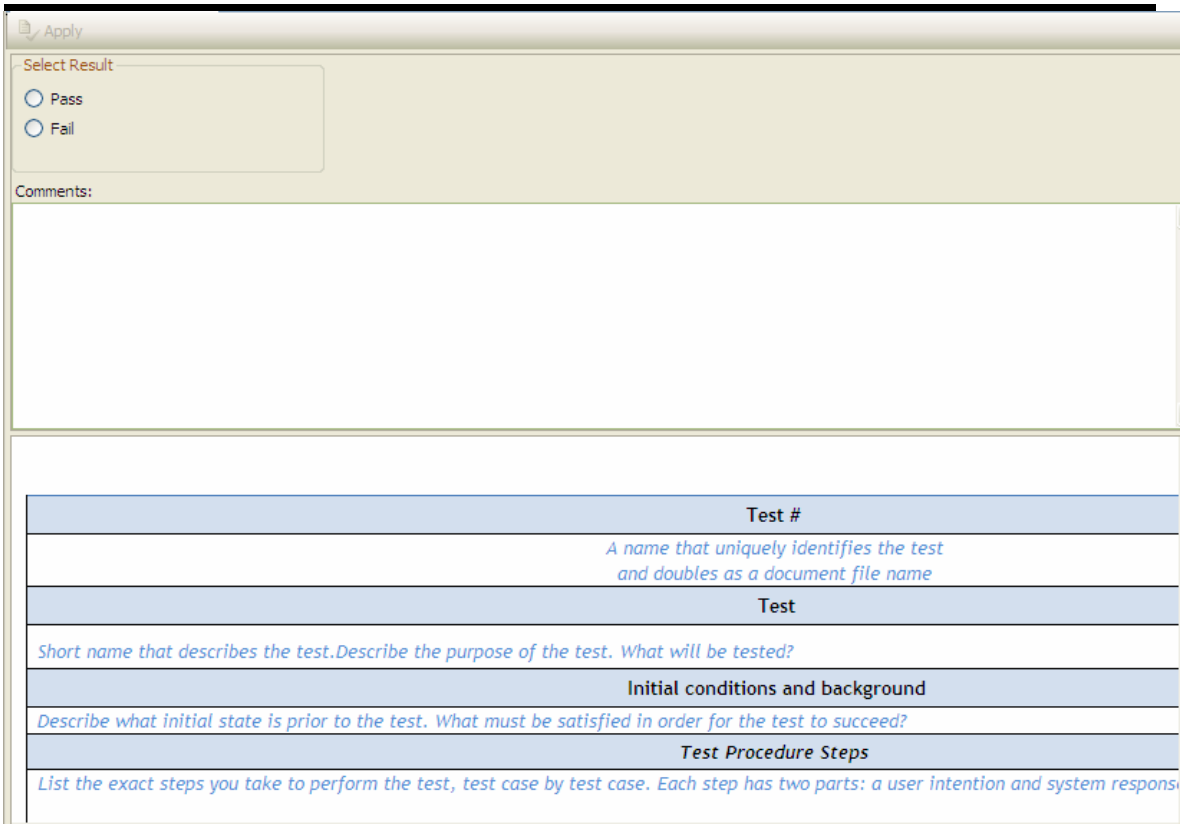
Slika 21. Manuelni test dokument

Dokument se nakon unosa podataka čuva i moguće ga je videti u listi testova u test pogled prozoru. Odavde se i pokreće, desnim klikom na test i pokretanjem akcije Run.



Slika 22. Pokretanje testa

Potom se otvara prozor sa detaljima testa. Tester izvršava neophodne korake i označava da li je test uspešan ili ne i objavljuje rezultate testa, najčeće unoseći i kometare i verziju na kojoj je test izvršen.



Slika 21. Pokrenuti test

Na ovaj način se obezbeđuju javno dostupne informacije o testovima i rezultatima, koji su korisni za sve članove tima, kao i za kreiranje konačnih izveštaja koji obezbeđuju pregled celokupnih rezultata po projektima.

Problem i rešenje:

U praksi će postojati aplikacija koja se testira. Postojaće i zahtevi koje ona teba da zadovolji. Testeri će aktivno učestvovati i testiranjem doprinositi kvalitetu aplikacije. Kreiraće se novi projekat, najčešće sa nazivom aplikacije na koju se odnosi. U okviru projekta testeri će kreirati test case-ove. Test case-ovi će se kreirati na osnovu korisničkih zahteva. U test case se unose neophodni podaci o ulazima, izlazima, kriterijumima prolaska testa i nizu koraka pri testiranju. Često se unose i preduslovi testiranja(neophodni uslovi da bi se test izvršio). Nakon pisanja testa, test je moguće videti u listi i pokretati po potrebi. Prilikom svakog pokretanja, tester će na jednom mestu imati sve podatke i izvršavaće test bez problema i na kraju označavati da li je prošao ili ne. Objavljivanjem rezultata, svi učesnici u testiranju moći će da vide šta je testirano i da li je ishod dobar. Ukoliko tester otkrije greške, prosto će, vezujući se za test case, kreirati bug, koji će potom moći da vide i developeri, kako bi iste ispravljali. Na ovaj način nastaje zatvoren krug pri testiranju, koji je ponovljiv i lak za praćenje. Kako je testiranje manuelno, sposobnost testera određuje kvalitet softvera, jer se isključivo na sposobnosti uočavanja i poznavanja sistema, zasniva konačan dobar rezultat.

7. Zaključak

Jasno je da proces obezbeđenja kvaliteta softvera zahteva mnogo napora, da postoje sredstva i potporni elementi ovog procesa, koji su danas brojni i fleksibilni. S toga, oganizaciji koja se bavi razvojem softvera ostaje da sama pronađe put, da pored svih metoda razvoja softvera odabere odgovarajuću, da je sprovede i prilagodi, i potom, paralelno sa razvojem, počne sa primenom standarda kvaliteta i standarda uopšte.

Formalnu prati uvek i neformalna struktura, koja često biva značajnija i inovativnija. Kreiraju se i interni standardi koji, ako su proizvod dogovora i zajedničke inicijative zaposlenih, znaju da budu ključ dobrog funkcionisanja i uspeha.
Od ovog trenutka mehanizam postaje kompletan i zahuktava se. Celokupna organizacija odgovorna je za proces razvoja i održavanja kvaliteta softvera, iako postoje sektori koji formalno snose odgovornost.

Organizacije u kojima se standardi uvode naprasno, bilo da se radi o internim ili formalnim, uvek pružaju određeni stepen otpora. Potrebno je uveriti ljude u organizaciji da dodir sa stvarnošću ima presudnu ulogu. Developeri često razvijaju sofver gubeći vezu sa stvarnim svetom. Njihova projekcija sistema postaje vremenom nešto što zaboravlja na korisnike i oni prestaju da gube sliku o detaljima, koji u praksi mnogo znače. Često se razvijaju veliki projekti, gde developer nije u mogućnosti da vidi druge delove sistema(zbog nedostatka vremena ili interesovanja) i tada dolazi do krupnih propusta, koji se otkrivaju integracionim testiranjem. Lek za ovakve probleme leži u testiranju.
Tester je zadužen da spozna funkcionisanje sistema u celini, da zna odgovore na pitanja i da, na kraju krajeva, drži revnotežu i podseća da aplikaciju koriste obični ili neobični ljudi, koji vole izveštaje i slike, sortiranje i prečice. To je neizbežno.

Kako bi se izbegli ovi problemi, problemi kasnog sprovođenja kontrole kvaliteta, danas je sve zastupljenija ideja da se testiranje uvodi paralelno sa razvojem, da od starta bude senka razvoja. Na ovaj način, uklanjaju se brojni nedostaci kasne spoznaje grešaka, gubljenje vremena i smanjuju troškovi. Proces testiranja se usavršava i automatizuje i na taj način štedi vreme i povećava kvalitet. Ovo je prava formula uspeha.

8. Literatura

[1] Chillarege, R.: „Software Testing Best Practices“, Center for Software Engineering, IBM Research 1999
[2] Kan S.H.: „Metrics and Models in Software Quality Engineering“, (2nd Edition) Addison Wesley 2002
[3] Copeland L.: „A pratctioner’s Guide to Software Test Design“, Artech House 2004
[4] Myers G.J., Badgett T., Thomas T.M., Sandler C.: „The Art of Software Testing“, Second Edition - John Wiley & Sons, Inc. 2005
[5] Elfriede, Dustin: „Effective Software Testing (50 Ways To Improve Your Testing)“, Addison Wesley 2002
[6] Black R.: „Critical Testing Processes -Plan, Prepare, Perform, Perfect“, Addison Wesley 2003
[7] Craig R.D., Jaskiel S.P.: „Systematic Software Testing“, Artech House 2002
[8] Arnold T., Hopton D., Leonard A., Frost M.: „Professional Software Testing with Visual Studio® 2005 Team System, Tools for SoftwareDevelopers and Test Engineers“, Wiley Publishing, Inc. 2007
[9] Ivaneža D.: „Primena standarda u razvoju informacionih sistema“, Zavod za informatiku i internet 2003
[10] Lewis W.E.: „Software Testing and Continuous Quality Improvement“, Second Edition - Auerbach Publications 2005
[11] Proces testiranja 1 i 2, interni dokument, 2006
[12] ISO/IEC 9126
[13] JUS ISO/IEC 15910
[14] ISO/IEC 14598
[15] www.ansi.org, januar 2008.
[16] www.thinkieeestandards.net/history.html, januar 2008.
[17] http://en.wikipedia.org, mart 2008.
[18] www.australia-standards.com, decembar 2007.
[19] http://www.softwaretestingwiki.com, januar 2008.
[20] www.hhs.gov, decembar 2007.
[21] http://www.softwaretestingwiki.com, mart 2008.
[22] www.worldwidestandards.com, februar 2008.
[23] www.ksi.com, februar 2008.
[24] http://www.levela.com , mart 2008.