Održavanje softvera

Lakoseljac, Filip

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Filip Lakoseljac

Održavanje softvera

Završni rad

Mentor: izv. prof. dr. sc. Sanja Čandrlić

Rijeka, rujan 2019.

SADRŽAJ

ZADATAK ZA ZAVRŠNI RAD

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 19.06.2019.

Zadatak za završni rad

Pristupnik: Filip Lakošeljac

Naziv završnog rada: Održavanje softvera

Naziv završnog rada na eng. jeziku: Software maintenance

Sadržaj zadatka: Održavanje softvera nosi veliki dio ukupnih troškova životnog ciklusa softvera. Ono uključuje sve izmjene koje se vrše u softveru nakon što je on isporučen korisniku kako bi se ispravila pogreška, poboljšale performanse ili softver unaprijedio prema novim zahtjevima. Student će analizirati aktivnosti koje se odvijaju tijekom faze održavanja softvera te će izučiti probleme koji mogu značajno utjecati na troškove i kvalitetu održavanja.

Mentor

Izv. prof. dr. sc. Sanja Čandrlić

Voditelj za završne radove

Zadatak preuzet: 26.06.2019.

(potpis pristupnika)

SAŽETAK

U ovom završnom radu obrađena je tema održavanje softvera.

Za početak je pobliže objašnjeno što je samo održavanje softvera te što pokreće potrebu za odvijanjem toga procesa. Objašnjeni su različiti oblici održavanja softvera te način na koji se provodi njegovo cjelokupno održavanje. Istraženo je na koje se izazove nailazi prilikom održavanja softvera te na koji se način ispravljaju u njemu uočene greške i problemi. Nadalje, pobliže su pojašnjeni neki modeli održavanja a zatim i faze koje se provode u toku toga procesa. Kako bi se shvatila sveobuhvatnost održavanja softvera pojašnjena je veza kvalitete te testiranja u odnosu ne održavanje softvera. Da bi se dokazalo koliko je održavanje u praksi bitno spomenut će se i troškovi samog održavanja softvera. Također, bit će riječi i o odnosu evolucije i održavanja softvera te za kraj refaktoriranju kao bitnijoj značajki održavanja te ulozi reusa u tom procesu.

Ključne riječi: održavanje softvera, softver, održavanje, model, testiranje, troškovi, refaktoriranje, evolucija

1. UVOD

„Zašto se softver nakon nekog vremena prestaje koristiti?, što je potrebno učiniti kako bi se osiguralo njegovo dugotrajno korištenje?“ S obzirom da se razvitkom tehnologije mijenjaju i hardveri a time i operacijski sustavi, to iziskuje da softver prati zadani tempo razvoja kako bi uvijek bio jednako funkcionalan. Kako bi softver opstao u njega je potrebno dugotrajno ulagati i održavati ga, o čemu će biti riječ u ovome radu. Potrebe za kvalitetnim softverom su svakoga dana sve veće pa stoga opstaju samo oni koji najbolje zadovolje potrebe i zahtjeve današnjeg užurbanog društva, a da pritom održavaju softver široko dostupnim. Zbog takvih se potreba javljaju troškovi održavanja koji su usko povezani s njegovom korisnosti. Ukoliko je softver korisniji, ulaganja u održavanje moraju biti veća, ali je pritom i njegovo korištenje znatno produženo. Kako bi bolje razumjeli kompleksnost procesa razvoja softvera te funkciju i položaj njegova održavanja u njemu valja ga pobliže smjestiti u procesu razvoja.

Osnovni cilj svakog softvera je da zadovolji korisničke zahtjeve potrebe. Podizanjem ljudske svijesti o potrebi za efikasnim i kvalitetnim razvojem softvera počinju se koristiti tehnike bazirane na teorijskim osnovama postavljenim u programskom inženjerstvu. Prema Waterfall modelu razvoja[1], koji je predstavio Royce 1970. godine, životni ciklus softvera se sastoji od 5 faza (Royce, 1970). Sve započinje analizom i definiranjem korisničkih zahtjeva. Na temelju korisničkih zahtjeva dizajnira se model sustava te se može započeti s programiranjem. U toj fazi razvoja bitno je izraditi čitljiv kod i dobro ga dokumentirati kako bi bio jednostavan za nadogradnju i održavanje. U suprotnom dobivamo nekvalitetan softver koji je vrlo teško održavati i nadograđivati. Po završetku programiranja te nakon što je razvojni tim proveo inicijalna testiranja, softver se predaje korisniku na testiranje i sada korisnik dobiva uvid u razvijeni proizvod. Nakon testiranja dobivamo pozitivnu ili negativnu povratnu informaciju, a kada je korisnik zadovoljan i smatra da su svi zahtjevi zadovoljeni softver započinje s produkcijskim radom. Od tog trenutka do kraja životnog ciklusa softvera programski inženjeri zaduženi su za održavanje. Brinu se o otklanjanju novonastalih ili naknadno primjećenih pogrešaka te o implementaciji izmjena.

[1] Vodopadni model (engl. *waterfall model*) predstavlja model u kojem se temeljne aktivnosti cjelokupnog procesa izrade za vrijeme programske potpore gleda kao jedna nezavisna faza razvoja (Jovic, Horvat, Danko, & Frid, 2015).

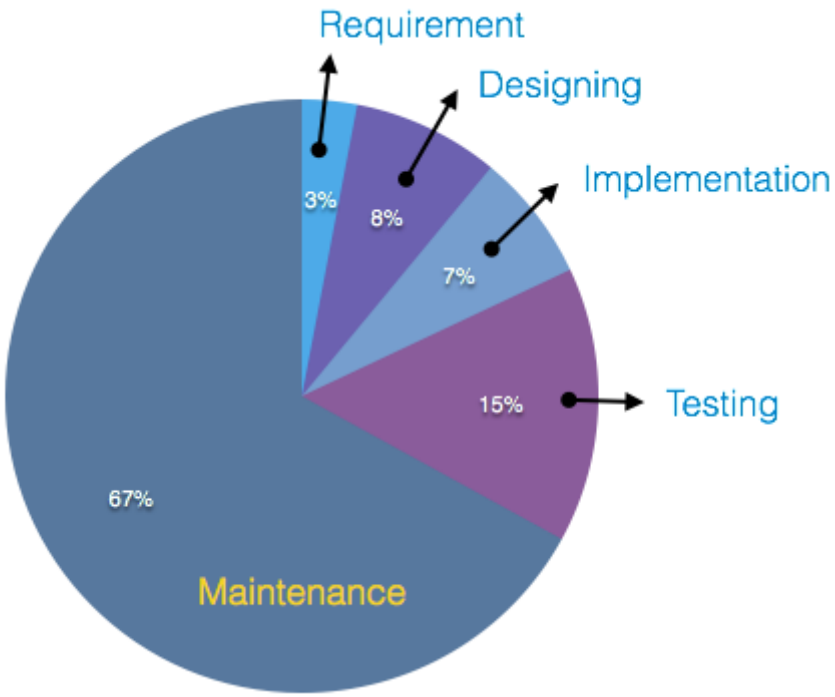
2. OPĆENITO O ODRŽAVANJU SOFTVERA

Održavanje možemo definirati kao napor koji je potreban za prolaženje, ispravak ili izmjenu pogrešaka uočenih u softveru. Ono ispituje učinke kvara softvera te načine na koje se ti učinci mogu smanjiti. Kako bi se osigurala održivost sustava, a time i spriječilo njegovo propadanje, važno je predvidjeti i detaljno isplanirati njegovo održavanje. Vrlo često kvaliteta i održivost softvera njegovim starenjem opadaju. To je rezultat mnogih faktora koji se, uzimajući svaki zasebno ne moraju činiti značajnima, ali se zbrajaju te rezultiraju vrlo teško održivim sustavom. Kvalitetne mogućnosti programiranja i tehnike su lako dostupne. Međutim, dok se ne postavi princip po kojem će se održavanje softvera provoditi, mnogi će sustavi biti izloženi greškama do točke u kojoj ih više neće biti moguće održavati. Velika je vjerojatnost da će se s poboljšanjem održivosti softvera povećati i mogućnost upravljanja te realiziranja dodatnih promjena. Održavanje je u osnovi reaktivna aktivnost više kaotična nego razvoj. Održavanje proizlazi iz potrebe prilagodbe softverskih sustava te okruženju koje se neprestano razvija. U većini slučajeva nije moguće izbjeći održavanje, a niti ga previše odgoditi. Čak i najsitnija promjena zakona i slično može potaknuti aktivnost održavanja. Organizacije se moraju držati u korak s tim promjenama, a to često znači modificiranje softvera koji podržava njihove poslovne aktivnosti. Posljedično, u tim situacijama održavanje softvera se događa na relativno neorganiziran način, jer cjelokupno održavanje u tom trenutku nije planirano. Svrsishodno tome, Anquetil i suradnici govore o razvijanju sustava za upravljanje znanjem za održavanje softvera. Baziraju se na prikupljanju i identificiranju znanja koje je potrebno tijekom održavanja softvera (Anquetil, Oliveira, Dias Greyck, Ramal, & Meneses, 2015).

Kada je u pitanju važnost održavanja za ukupan razvoj softvera, najbolje ćemo ju pokazati dijagramom udjela troškova u pojedinim fazama razvoja softvera (slika 1). Može se primijetiti kako održavanje softvera zauzima više od 50%, štoviše čak 67% ukupnih troškova u razvoju softvera. Prema tomu se može zaključiti da je održavanje softvera od jako bitnog značaja, u usporedbi s ostalim aktivnostima.

Opseg znanja prisutan u inženjerstvu softvera je raznolik i neprestano raste. Može se ustanoviti kako je održavanje softvera intenzivna aktivnost. Osobe koje ga održavaju moraju imati znanje o domeni aplikacije koja se koristi, imati mnogo prakse, a potrebno je i poznavanje više programskih jezika te razne sveobuhvatne programske vještine. Međutim, kao najveći problem tijekom održavanja softvera ističe se nedostatak dokumentacije. Studije su izvijestile da je 40%

do 60% tijekom održavanja softvera posvećeno razumijevanju sustava (Anquetil, Oliveira, Dias Greyck, Ramal, & Meneses, 2015).



Slika 1. Udio održavanja u ukupnom razvoju softvera (Software Maintance Overview, 2019)

Održavanje se temelji na procesu modifikacije putem kojeg se softver nastoji nadograđivati i prilagođava potrebama korisnika kroz duži vremenski period. S obzirom da je klasifikacija *Swansona i Lientza* (Swanson & Lientz, 1980) jedna od najcitiranijih, prema njoj održavanje softvera možemo podijeliti na 4 kategorije: korektivno, adaptivno, perfektivno i preventivno održavanje (Kontogiannis, 2011). Navedene metode su kasnije postale opće prihvaćene te standardizirane u ISO/IEC 14764:2006 (ISO/IEC, 2006) standardu. Sam proces održavanja softvera sadrži aktivnosti koje uključuju plan održavanja koji sadrži pripremu softvera, prepoznavanje problema te saznanje o upravljanju konfiguracijom proizvoda. Stoga proces analize problema uključuje provjeru valjanosti, ispitivanje i pronalazak rješenja te u konačnici dobivanje sve potrebne podrške za podnošenje zahtjeva za izmjenu. Kada je u pitanju moderno održavanje softvera, ono je skoro u potpunosti automatizirano s obzirom da se vrlo često dešava nezavisno od volje korisnika. Tako se primjerice automatski vrše ažuriranja sistemskog i

aplikativnog softvera na mobilnim uređajima a može se i prilagoditi na način da ovisi o spajanju na mrežu odnosno samo u slučaju kada je dostupan *Wi-Fi* signal (Lemeš & Buzađija, 2018).

Kako bi se najbolje opisalo taj proces potrebno je navesti pojedine procese te aktivnosti potrebne za njegovo kvalitetno održavanje. U početku je potrebno provesti identifikaciju svih zahtjeva. Zatim se provodi crtanje ulaza iz povratnih informacija korisnika te zapisnika sustava. U slučaju da je potrebna bilo kakva izmjena, mora joj prethoditi temeljita analiza troškova i učinka kako bi se provjerilo ima li potencijalnih negativnih implikacija u slučaju provođenja tih izmjena. Iz izmijenjenih modifikacija kreira se skup specifikacija zahtjeva. Ukoliko je prisutna potreba za modifikacijama, zastarjeli moduli moraju biti zamijenjeni novima. Ti su novi moduli dizajnirani te implementirani prema specifikacijama zahtjeva. Nakon kreiranja novih modula, generiraju se testni slučajevi za naredno testiranje jedinice i integraciju iste. Mogući potencijalni problemi za krajnjeg korisnika provjeravaju se testovima prihvatljivosti korisnika. Kako bi se riješio problem kompatibilnosti s različitim inačicama operacijskih sustava, izvršava se upravljanje i ujednačavanje slojeva pomoću alata za kontrolu verzija. Krajnje se testiranje softvera odvija na web mjestu klijenta nakon prethodne instalacije te se time proces završava. (Osborne, 1985).

3. IZAZOVI I PROBLEMI U ODRŽAVANJU SOFTVERA

Iako se održavanje softvera sve više smatra ključnim, postupak nije jednostavan te se tijekom njegovog procesa ulažu veliki napori. Za taj su proces potrebni stručnjaci koji dobro poznaju najnovije trendove softverskog inženjerstva i mogu izvesti odgovarajuće programiranje i testiranje ovisno o potrebama. Nadalje, programeri se mogu suočiti s nekoliko tehničko upravljačkih izazova tijekom izvršavanja softverskog održavanja zbog kojih se procesi mogu uvelike odužiti, a time i mnogo skuplje platiti. Treba nadodati da programeri provedu oko 61% vremena na poslovima održavanja tokom bavljenja programerstvom, a samo 39% na poslove razvoja samog softvera (Krneta, 2017).

Jedan od izazova sa kojim se susreću je trošak samog održavanja. Različita istraživanja sugeriraju da održavanje softvera u cjelini iznosi oko 67% troškova ciklusa razvoja softvera. Ta istraživanja također navode da se ti troškovi uglavnom odnose na poboljšanja, a ne ispravak grešaka u samom softveru (Bourque & Richard, 2004).

Nadalje, otkrivanje koji će učinak na cjelokupan softver imati izmjene koje se uvode jedan je od najvažnijih izazova prilikom njegova održavanja. Analizom učinka nastoji se ispitati posljedice uvedenih promjena u kodu s planom smanjenja potencijalnih grešaka u izmijenjenom kodu. Na taj se način uvodi procjena ispravnosti promjena a time i predviđanje potencijalnih izmjena uz procjenu rizika povezanih s njegovim dovršetkom (Gupta & Sharma, 2015).

Jedan od izazova s kojima se susreću prilikom izvođenja softverskog održavanja je pronalaženje osobe ili programera koji je konstruirao program što može biti teško i dugotrajno. Do potencijalnog izazova dolazi i ukoliko promjene unosi pojedinac koji nije u stanju posve jasno razumjeti sam program. Ako sustave ne održavaju izvorni autori, nego druga osoba, to može rezultirati zbrkom i pogrešnim tumačenjem promjena izvršenih u programu i arhitekturi koda.

Nedostatak informacija u analizi između korisnika i programera također može postati veliki izazov u održavanju softvera. Problem predstavlja slučaj kada softver nije dizajniran za jednostavno uvođenje izmjena u kodu već se iziskuje uporaba kompliciranijih rješenja koja sa sobom nose i veće rizike kada je u pitanju rješavanje samih problema. U tom je slučaju potrebno više vremena i veće razumijevanje samog koda kako bi se struktura koda softvera uspjela najefikasnije ispraviti (Osborne, 1985).

4. POTREBA ZA ODRŽAVANJEM SOFTVERA

Održavanje je potrebno kako bi se omogućilo i osiguralo da softver i dalje zadovoljava zahtjeve korisnika. Ono je primjenjivo na softver koji je razvijen korištenjem bilo kojeg softverskog modela životnog ciklusa (na primjer, spiralnog ili linearnog). Softverski proizvodi se mijenjaju zbog korektivnih i ne-korektivnih softverskih postupaka. Održavanje se temelji na ispravljanju grešaka nastalih tokom njegove izrade, a svrsishodno tome i poboljšanju dizajna kako bi softver bio privlačan, a pritom i koristan nakon implementacije poboljšanja. Također potrebno je softver prilagoditi tako da nudi jednake mogućnosti bez obzira na različit hardver, značajke sustava ili pak telekomunikacijske mogućnosti (Bourque & Richard, 2004).

Kako bi se čim bolje shvatila važnost održavanja softvera valja istaknuti nekoliko bitnih karakteristika zbog kojih dolazi do potrebe za njegovim održavanjem.

Mane tržišta jedna su od karakteristika, a možemo ih pojasniti na primjeru digitalizacije. Kada je u pitanju pristupačnost korisniku nastoji se svima omogućiti jednaku uslugu odnosno dostupnost jednakog softvera. Na taj se način teži otvaranju novih tržišta u inače nepristupačnijim područjima kako bi se svim korisnicima osigurala jednaka dostupnost usluge odnosno konačnog proizvoda. Tako primjerice Facebook mijenja svoju aplikaciju i provodi sve njene nadogradnje na besplatnoj osnovi pri čemu omogućava njenu uporabu široj populaciji, a pritom svojim proizvodom promovira ostale platforme i usluge koje one nude (Software Maintenance, 2018).

Vodeći računa o očekivanjima kupaca potrebno je voditi računa o njihovim željama i zahtjevima na temelju analize i komunikacije s krajnjim korisnicima. Tvrtka svoj ostanak u vrhu konkurencije ostvaruje nuđenjem najboljeg mogućeg proizvoda u skladu sa potrebama i potražnjom kupaca, tako su tvrtke uporno u potrazi za dodavanjem veće vrijednosti svojim proizvodima u obliku novih značajki, raznih dodataka i nadogradnji svoga proizvoda.

Promjene na različitim platformama dešavaju se svakodnevno. Softveri se svakodnevno unaprjeđuju u skladu sa ljudskim potrebama. Tako su i naši mobiteli svaki mjesec preplavljeni zahtjevima za nadogradnju verzija različitih aplikacija. Kako bismo omogućili te promjene u aplikacijama koje su dovele ove nadogradnje, potrebna je nadogradnja preglednika kako bi aplikacija radila bez ikakvih problema.

Promjena poslovnih ambicija još je jedna od potreba za održavanjem softvera. U vrijeme širenja ili akvizicije poduzeća, uvođenjem promjena poslovanja potrebno je razmišljati o

promjenjivosti te skalabilnosti projekata. Ovo zahtijeva promjene u funkcionalnostima softverskih aplikacija ovisno o planu poduzeća kako bi one mogle raditi jednako dobro kao i prije prelaska na novu razinu.

Kada se govori o većim softverskim sustavima, javlja se potreba za stručnjacima iz različitih područja primjene, od stručnjaka za baze podataka, programera i slično. Međutim oni su često na različitim lokacijama. Samim time javlja se mogućnost uvođenja posebno specijalizirane osobe čiji bi zadatak bio klasificiranje i prosljeđivanje dospjelih zadataka. Na taj način bi se povećala efikasnost održavanja softvera a posljedično i smanjili troškovi. Ti se svi zahtjevi kategoriziraju te se proslijede kasnije s opisom problema upravo onom stručnjaku koji je zadužen za to područje. Cjelokupni proces se odvija na način da korisnici generiraju zahtjev s opisom problema kada primijete određene nepravilnosti. Samim time primjenjuju se određene metode za rješavanje problema, pozivaju se određeni stručnjaci i cijeli proces se odvija bez ljudske intervencije (Blašković, 2016).

5. OBLICI ODRŽAVANJA SOFTVERA

Kada je u pitanju održavanje softvera, na temelju softversko-informativne zbirke znanja SWEBOK (engl. *Software Engineering Body and Knowledge*) održavanje softvera se sastoji od korektivnog, adaptivnog te perfektivnog oblika održavanja (Bourque & Richard, 2004). Prema IEEE 14764 standardu razlikujemo i četvrti oblik održavanja odnosno preventivno održavanje (Ibrahim & Yahaya, 2017). IEEE 14764 klasificira adaptivno i perfektivno održavanje kao poboljšanja softvera u procesu održavanja (ISO/IEC, 2006). Pritom je korektivno i preventivno održavanje kategoriziramo kao kategorija prepravljanja.

Navedeni oblici održavanja izuzetno su korisni jer nam omogućuju da analiziramo i uklonimo sve moguće prijetnje softveru, a na taj način i povećamo stabilnost organizacije (Niessink & Vliet, 2000).

5.1. Korektivno održavanje

Korektivno održavanje na svojstven način izvršava sve promjene u softverskom proizvodu s namjerom ispravljanja problema odnosno pogrešaka u sustavu koje su otkrili korisnici. Takav se postupak održavanja sastoji od aktivnosti koje se odnose na ispravljanje pronađenih pogrešaka kako bi sustav nesmetano radio. Kao glavne uzroke korektivnog održavanja možemo navesti logičke pogreške do kojih dolazi zbog neispravnog logičkog tijeka, pogreške kodiranja koje nastaju kao posljedica grešaka nastalih prilikom kodiranja te pogreške u dizajnu (Al-sharif, 2015).

5.2. Adaptivno održavanje

Drugi oblik održavanja softvera je adaptivno održavanje. Ono se provodi nadogradnjom starijih verzija softvera te prilagođavanjem softvera promjenama u njegovu okruženju nastalih razvitkom tehnologije kao što su primjerice uvođenje novog hardvera ili pak novog izdanja operacijskog sustava. Na taj način održavanje obuhvaća prilagodbu na platforme koje se odnose

i na softverski i hardverski oblik konfiguracije. Prema tome ovaj oblik održavanja ne utječe na funkcionalnost samog softvera već ne njegovu prilagodbu novom okruženju (Niessink & Vliet, 2000).

5.3. Preventivno održavanje

Preventivno održavanje uključuje preventivne mjere u obliku restrukturiranja, dodavanja komentara ili pak ažuriranja dokumentacije vršeći izmjene softvera pritom uklanjajući potencijalne greške u začetku njene izrade kako se ti manji problemi ne bi transformirali u značajnije u budućnosti (Software Maintenance Models, 2018). Na taj se način uvelike sprječava mogućnost pojave većih troškova u budućem procesu održavanja softvera.

5.4. Perfektivno održavanje

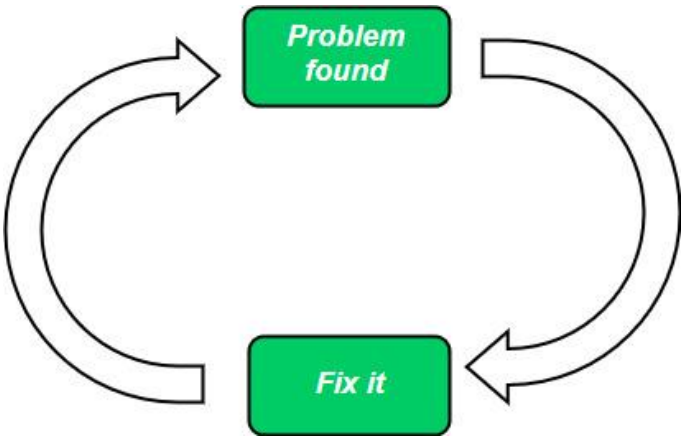
Kada je u pitanju perfektivno održavanje valja istaknuti da je ono prijeko potrebno za poboljšanje pouzdanosti softvera tijekom dugog vremenskog razdoblja dodavanjem poboljšanih funkcija i provođenjem novih korisničkih zahtjeva u skladu sa potrebama korisnika. Ono povećava performanse i održivost te zadovoljava korisničke zahtjeve koji su prvotno zanemareni. Ovaj oblik održavanja može uključivati i restrukturiranje ili reinženjering softvera o kojem će nešto više riječi biti kasnije. Također obuhvaća i prepisivanje, a samim time i prepravljanje dokumentacije, mijenjanje formata i sadržaja izvještaja te učinkovitiju logičku obradu. Potreba za perfektivnim održavanjem pokazatelj je da analitičari nisu otkrili sve potrebe korisnika ili da programeri nisu bili u mogućnosti ispuniti sve njihove zahtjeve prije izdavanja softvera (Osborne, 1985).

6. MODELI ODRŽAVANJA SOFTVERA

Kako bi se nadvladali unutarnji i vanjski problemi softvera predlažu se posebni modeli održavanja softvera koji su se počeli uvoditi još 1970. godine. Ovi modeli koriste različite pristupe i tehnike kako bi pojednostavili proces održavanja, a isto tako ga i učinili isplativim. Pritom se koncentriraju na tri važne aktivnosti: razumijevanje softvera, izmjena i promjena softvera te obnavljanje softvera. Prema tome svaka organizacija ima potrebu koristiti neki model održavanja kako bi poboljšala sam postupak održavanja. Također, treba napomenuti da su prisutne razne aktivnosti pri održavanju softvera. Unatoč tome one nisu jedinstvene te ovise o nekoliko faktora od kojih su najznačajniji razina potrebne izmjene, raspoloživi resursi te analiza stanja postojećeg projekta. Neki od najvažnijih modela održavanja softvera jesu: Quick-fix model, iterative enhancement model, model orijentiran za ponovno korištenje, Boehmov model i Tauteov model (Shivani, 2015).

6.1. Quick-fix model

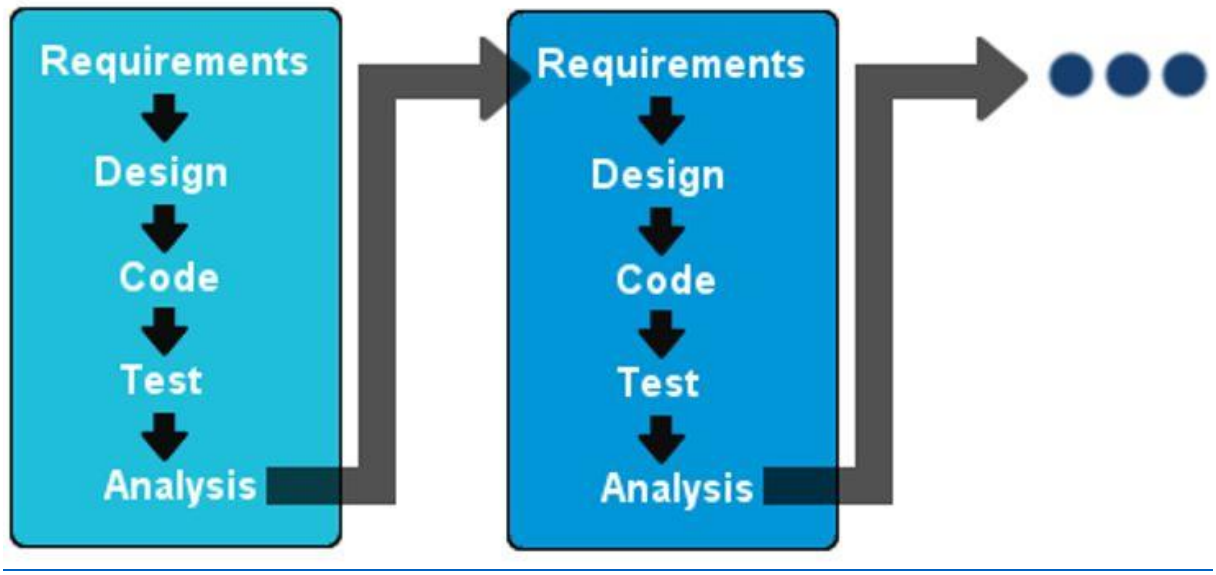
Model za brzo popravljanje (engl. *Quick-fix model*) predstavlja vrlo brz pristup koji se koristi za održavanje softverskog sustava. Cilj ovog modela je s obzirom na vremensko ograničenje identificirati problem a zatim izmijeniti kod da bi se ispravili nedostatci u kodu u što kraćem roku. Naposlijetku je omogućena i izmjena dokumentacije. Kao prednost valja istaknuti da ovaj model svoj posao obavlja brzo pritom iziskujući izrazito male troškove. Na taj način ovaj model pristupa izmjeni softverskog koda sa vrlo malim utjecajem na cjelokupnu strukturu softverskog sustava. Slika 2. prikazuje shemu modela brzog popravljanja.



Slika 2. Prikaz quick-fix modela (Software Engineering - Quick-fix Model, 2013)

6.2. Iterative enhancement model

Model iterativnog unaprjeđenja smatra da su promjene u sustavu iterativne naravi. Na taj način ovaj model uključuje promjene u softveru temeljene na analizi postojećeg sustava u slučaju kada zahtjevi nisu potpuno razumljivi. Dokumentacija se u početku modificira, a zatim se vrše izmjene na razini koda. Kao prednosti možemo istaknuti nekoliko karakteristika: razumijevanje koje se povećava uzastopnim usavršavanjem koje pritom podržava i ponovnu upotrebu (engl. *reuse*). Kako bi se unaprijedio softver redovito se provodi analiza troškova, a nakon svake iteracije rješenje postaje efikasnije. Može se reći da iterativni model ne uključuje visoku stopu složenosti stoga se ona pokušava kontrolirati, a pritom i održati dobar dizajn. Model iterativnog poboljšanja je prema tome podijeljen u tri stupnja: analiza softverskog sustava, klasifikacija zatraženih izmjena te implementacija zatraženih izmjena. Slika 3. prikazuje prikaz iterativnog modela unaprjeđenja (Incremental Model or iterative enhancement model in software engineering, n.d.).

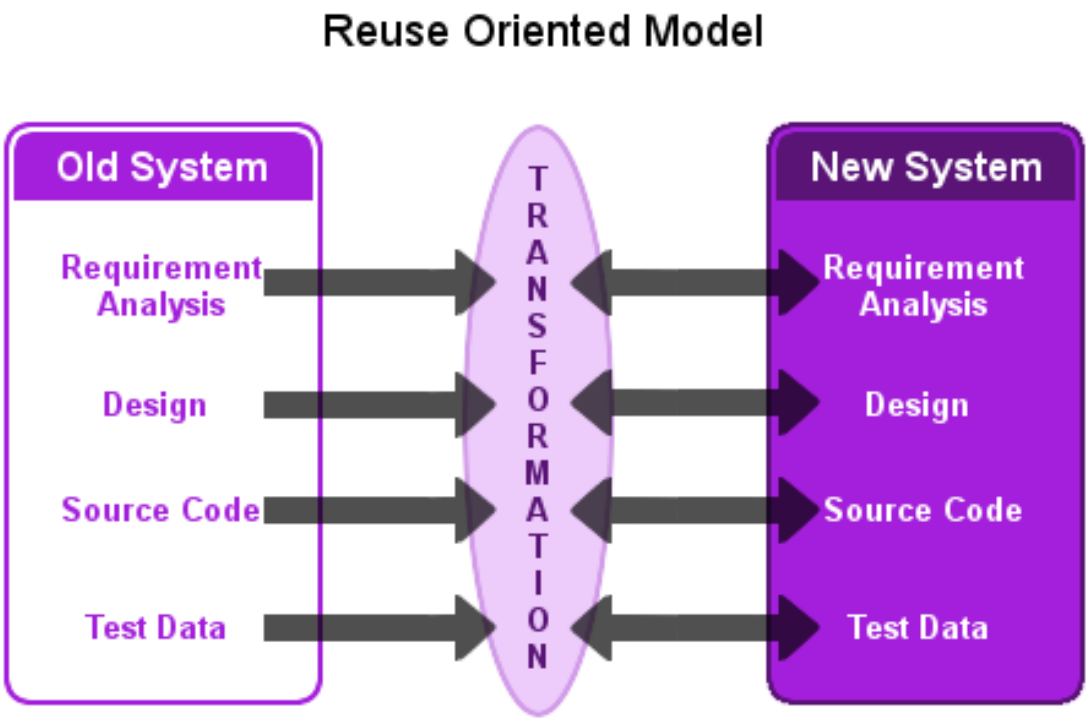


Slika 3. Prikaz iterativnog unaprjeđenja (Software Maintenance Models, 2018)

6.3. Reuse oriented model

Model orijentiran na ponovno korištenje (*engl. Reuse oriented model*) omogućuje da se dijelovi starog odnosno postojećeg sustava koji su prikladni za ponovnu upotrebu identificiraju i razumiju u modelu orijentiranom na ponovnu uporabu. Ti dijelovi tada prolaze kroz izmjene i

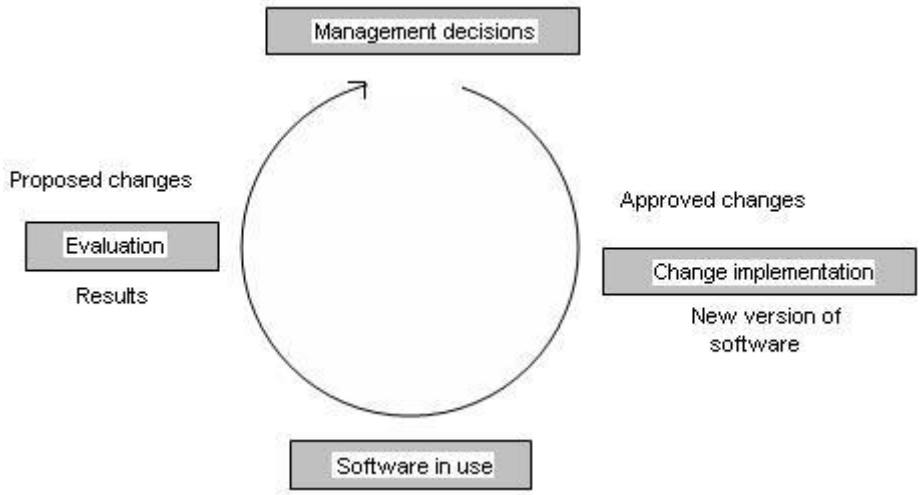
dopune koji se izvode na temelju novih zahtjeva. Kao posljednji korak ovog modela može se istaknuti integracija modificiranih dijelova u novi sustav. Slika 4. daje prikaz modela orijentiranog za ponovno korištenje (Software Maintenance Models, 2018).



Slika 4. Model orijentiran za ponovno korištenje (Software Maintenance Models, 2018)

6.4. Boehm-ov model

Boehmov model provodi postupak održavanja koji se temelji na ekonomskim i načelima korisnosti. Predstavlja postupak održavanja u ciklusu zatvorenog kruga, pri čemu se primarno sugeriraju i odobravaju promjene, a zatim i izvršavaju te na taj način provode. Boehm je predložio model koji nastoji uključiti opće karakteristike kvalitete u kriterij kvalitete kroz tri kuta promatranja s obzirom na vrstu i položaj korisnika. Podijelio ih je na krajnje korisnike, potencijalne korisnike to jest one koji se nalaze na drugim mjestima i potencijalne korisnike koji će to tek postati (Stefanović, Mitrović, & Erić, 2006). Na slici 5 prikazan je Boehmov model.

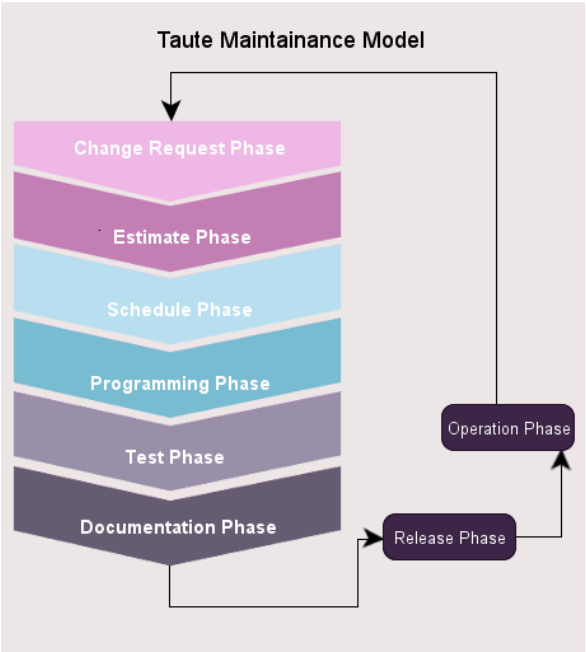


Slika 5. Boehemov model (Boehms Software model, 2019)

6.5. Taute-ov model

Taute-ov model održavanja stvorio je B. J. Taute 1983 godine. Tauteov model tipičan je model održavanja koji se sastoji od osam faza u ciklusu. Proces održavanja započinje zahtjevom promjene i završava njegovim radom. Faze Tauteova modela održavanja su: faza zahtjeva za promjenu, faza procjene, faza rasporeda, faza programiranja, faza ispitivanja, faza dokumentacije, faza otpuštanja i faza rada (Software Maintenance Models, 2018).

izvor: (Shivani, 2015)



Slika 6. Tauete-ov model (Software Maintenance Models, 2018)

7. FAZE ODRŽAVANJA SOFTVERA

Održavanje softvera nastoji se sprovesti u fazama kako bi se njegovim procesom obuhvatila cjelokupna mogućnost potrebe za izmjenama i ispravcima dijelova softvera. Faze se mogu proširiti tako da mogu biti uključeni razni prilagođeni predmeti i procesi. Te se aktivnosti nadovezuju jedna na drugu te zajedno sprovedene u proces održavanja čine veću cjelinu.

Kod aktivnosti prepoznavanja i praćenja može se spoznati da ono uključuje aktivnosti koje se odnose na utvrđivanje zahtjeva za izmjenom ili održavanjem bez prisutnih preskakanja dijelova u toku utvrđivanja i identifikacije problema. Na taj ih način generira korisnik ili sustav koji ih određuje skupom koraka prema specifikaciji zahtjeva. Zaključujemo da problem prilikom održavanja softvera i faze identifikacije i uočavanja problema čine izvještaji i poruke korisnika. Kada je u pitanju analiza, do promjena se dolazi temeljitom analizom planiranih promjena i njihov utjecaj na rad softvera. Ukoliko je potencijalni utjecaj ozbiljan i nesiguran, traži se alternativno rješenje. Skup svih potrebnih izmjena moguće je materijalizirati u specifikacije zahtjeva. U ovoj se fazi trošak modifikacije odnosno održavanja analizira i zaključuje na temelju stručne procjene. Dizajn je sljedeća faza koja se kao aktivnost provodi na način da novi moduli koje je potrebno zamijeniti ili modificirati budu dizajnirani prema specifikacijama zahtjeva postavljenim u prethodnoj fazi. Uz sve to, izrađeni su i testni slučajevi za provjeru valjanosti novog dizajna kako bi se osigurala njegova učinkovitost. Da bi se provela faza implementacije, novi moduli se kodiraju uz pomoć strukturiranog dizajna stvorenog u fazi dizajna, a pritom se očekuje da će svaki programer paralelno obaviti testiranje jedinica. Implementacijom se nastoji osigurati veću točnost izvršavanja softvera pri uklapanju u novo okruženje. Po završetku implementacije dolazi do testiranja sustava pri čemu se integracijsko testiranje provodi među novostvorenim komponentama. Integracijsko testiranje provodi se i na razini povezanosti pojedine komponente i čitavog sustava. Također, sustav je testiran kao cjelina kako bi sve nadogradnje funkcionirale bez prisutnosti i pojavljivanja mogućih obavijesti o greškama koje se izvode u realnom vremenu i izvornom okruženju. Ispitivanje prihvatljivosti predstavlja fazu kojom se nakon internog testiranja sustava testira prihvaćanje svih zahtjeva koje je postavio korisnik. Ako u toj fazi korisnik uoči neke probleme može ih prijaviti te će se oni pokušati ispraviti u sljedećoj iteraciji. Nakon prihvaćanja uvedenih promjena od strane korisnika dolazi do isporuke na način da se uvedene promjene uvode bilo malim paketom ažuriranja ili potpuno novom instalacijom sustava (Khanna, Shah, Jain, & Ramanathan, 2017). Po završetku se odvija završno testiranje koje se provodi od strane klijenta nakon isporuke

softvera. Po potrebi se osigurava prostor za obuku uz prilaganje tiskanog korisničkog priručnika. Kada je u pitanju upravljanje održavanjem ono nam omogućuje rješavanje problema korisnika prilikom instalacije ili samog rada softvera. Istovremeno se sve prijavljene greške u softveru bilježe kako bi se ispravile prilikom uvođenja sljedećih izmjena i ažuriranja. U tome potpomažu alati za upravljanje verzijama softvera (Software Maintance Overview, 2019).

8.VEZA KVALITETE I ODRŽAVANJA SOFTVERA

Kvaliteta softvera može se okarakterizirati atributima kao što su razumijevanje, mogućnost analize, promjenjivost i mogućnost testiranja, što čini softver održivijim.

Prvi zadatak procesa održavanja softvera razumijevanje je postojećeg softvera. Jedna od glavnih briga bilo koje organizacije za održavanje je razumijevanje i procjena troškova održavanja izdanih softverskih sustava. Razumijevanje sustava glavna je točka u fazi održavanja koja se odnosi na sposobnost programera da razumije funkcije i njihove odnose unutar sustava. To im omogućuje da prepoznaju pogreške ili dijelove sustava na kojima je potrebno izvršiti izmjenu odnosno modifikaciju.

„Razumljivost“ softvera vrlo je važan faktor kvalitete softvera jer omogućava programerima da s lakoćom razumiju strukturu sustava, što ujedno pojednostavljuje i procese održavanja. Dakle, razumijevanje sustava ovisi o kognitivnim sposobnostima i sklonostima, poznavanju domene aplikacije i skupa pomoćnih uređaja koje pruža okruženje softverskog inženjerstva. Samim time razumijevanje utječe na ukupan napor i troškove potrebne u procesu održavanja. Stoga ono za cilj ima predstaviti dovoljno informacija o sustavu. Ako dokumentacija i izvorni kod nisu povezani, postupak održavanja bit će vrlo težak i neprecizan. Drugim riječima, razumljivost će biti velika ako i samo ako su izvorni kod i dokumentacija usko povezani.

Kada je u pitanju analiza razvijenog softvera ona je ključna za karakterizaciju ponašanja softvera. Da bi se prepoznale pogreške ili dijelovi koji se trebaju izmijeniti, potrebno je izvršiti detaljnu analizu sustava. Analizom se određuje sposobnost sustava da otkrije nedostatke, uzroke kvarova u softveru ili pak identifikaciju dijelova koji zahtijevaju izmjenu koja bi mogla biti od primarne važnosti.

Jedan od najvećih izazova s kojima se suočavaju softverski inženjeri je upravljanje kontrolom promjena. Procjenjuje se da troškovi kontrole promjena mogu iznositi između 40% i 70% troškova čitavog životnog ciklusa softvera (Hunt, Turner, & McRitchie, 2008). Promjenjivost je sposobnost mijenjanja softverskog proizvoda. Na taj način programerima je omogućeno da mijenjaju isporučeni sustav pritom koristeći različite kriterije pomoću kojih ostvaruju promjenu usprkos ograničenoj količini dostupnih informacija. U cilju pojednostavljivanja procesa mijenjanja softvera i minimiziranja nuspojava modifikacije na ostalim dijelovima sustava potrebno je zadovoljiti neke kriterije atributa promjenjivosti: modularnost, općenitost, proširivost, stabilnost i uređenost.

Testiranje softvera definirano je kao stupanj do kojeg sustav ili komponenta omogućuju uspostavu kriterija testiranja odnosno njegova izvođenja kako bi se utvrdilo jesu li ti kriteriji ispunjeni. Stoga još možemo reći da je to sposobnost provjere ispunjava li softver svoje zahtjeve. Koristi se za provjeru oštećenja softvera prije puštanja softverskog proizvoda u prodaju. Prema (Singh & Saha, 2010), mogućnost testiranja softvera definirana je na temelju kontroliranja i promatranja. Kontroliranje predstavlja sposobnost upravljanja ulazom, dok promatranje predstavlja sposobnost mjerenja izlaznog sustava. U postupku održavanja, testiranje pokazuje je li promjena softvera postigla svoje ciljeve i postoje li bilo kakve nuspojave na ostale funkcije uvođenjem te promjene. Testiranje smanjuje nedostatke nastale kao posljedica lošeg dizajna softvera. Proces testiranja postiže se pružanjem određene vrste procjene vrijednosti od strane pojedinca ili skupine kako bi se provjerila funkcionalnost komponenti sustava te osigurala mogućnost obavljanja potrebne funkcije.

Dakle, kvaliteta softvera i održivost su neraskidivo povezani. Što je veći broj kvalitetnih atributa ugrađenih u softver tijekom razvoja, veći je i stupanj povjerenja u njegovu održivost. Iako je znatno lakše atribute kvalitetno ugraditi u softver on se također može poboljšati i za vrijeme održavanja. Trošak dodavanja kvalitete postojećem softveru je međutim znatno skuplji.

9. VEZA TESTIRANJA I ODRŽAVANJA SOFTVERA

Testiranje se primarno vrši radi pronalaženja grešaka i propusta pri izradi softvera. Također ono se izvodi kako bi se osigurala prikladna logika i struktura softvera. Temeljito testiranje jedinice, integracija i testiranje sustava značajno povećava vjerojatnost da će softver biti pouzdan i ispravan. Prema tome, testiranje pruža sigurnost da su aktivnosti za pravilno održavanje softvera sprovedene.

Testiranje i održavanje dvije su odvojene, ali podjednako važne faze životnog ciklusa razvoja softvera (SDLC[2]).

Nakon što su zahtjevi fiksirani to jest dogovoreni, razvojni tim započinje s izradom koda za stvaranje potrebnoga proizvoda. Pri završetku razvoja, testerima se dodjeljuje zadatak izgradnje instanci koda kako bi testirali i potvrdili da je proizvod kreiran prema zahtjevima. Nakon završetka ispitivanja proizvod se pušta u prodaju te na taj način stavlja na raspolaganje kupcima odnosno široj javnosti.

Nakon puštanja u prodaju proizvod prelazi u fazu održavanja. Ovdje proizvod prolazi kroz stalno testiranje, nova poboljšanja, izmjene, uklanjanje i dodavanje značajki i ispravljanje neželjenih novonastalih pogrešaka. Pri svakoj promjeni u kodu proizvoda zbog dodavanja ili uklanjanja bilo koje karakteristike ili funkcionalnosti proizvoda potrebno je ponovno provesti testiranje.

Testiranje je stalan dio SDLC-a, jer posao testera započinje dovršetkom zahtjeva. Prema tome ispitivač započinje s kreiranjem test slučajeva na temelju tih zahtjeva i kriterija prihvatljivosti. Po završetku razvoja za utvrđene zahtjeve, započinje testiranje. Dok se god uvode nove promjene, nastavlja se i postupak ispitivanja tijekom faze održavanja.

Stoga testiranje potvrđuje i pokazuje da je proizvod odnosno sam softver izgrađen prema zahtjevima i da nijedna značajka niti funkcionalnost nije prekršena u izradi softvera. S druge strane održavanje podrazumijeva da je proizvod već pušten u javnost, to jest kupcu na upotrebu, a sada se radi na poboljšanju i unaprjeđenju softvera u skladu s novim potrebama ili zahtjevima tržišta. Možemo reći da, u tehničkom pogledu održavanje poglavito nije dio testiranja već da je testiranje zapravo dio održavanja softvera (Osborne, 1985) .

[2] SDLC (engl. *Software Development Life Cycle*) predstavlja okvir koji definira zadatke koji se obavljaju u svakom koraku u procesu razvoja softvera. SDLC predstavlja strukturu koja prati razvojni tim unutar softverske organizacije (Software Development Life Cycle (SDLC), n.d.).

10. REINŽENJERING SOFTVERA

Potreba za ažuriranjem softvera kako bismo ga zadržali na trenutnom tržištu, a da ne utječemo na njegovu funkcionalnost, naziva se reinženjeringom softvera. To je temeljit proces u kojem se dizajn softvera postupno mijenja a programi se prepisuju.

Naslijeđeni softver ne može održavati usklađenost s najnovijom tehnologijom dostupnom na tržištu. Kako hardver zastarijeva, ažuriranje softvera postaje sve teže. Čak i ako softver s vremenom ostari, njegova funkcionalnost ostaje ista.

Primjerice, u početku je Unix razvijen u asemblerskom jeziku. Kad je nastao jezik C, Unix je redizajniran na C-u jer je rad na asemblerskom jeziku bio izrazito težak.

Osim navedenog, povremeno programeri uočavaju da je za pojedine dijelove softvera potrebno više i učestalije održavanje od ostalih te im je na taj način potreban učestaliji re-inženjering.

11. TROŠKOVI I STANDARDI ZA ODRŽAVANJE SOFTVERA

Kada je održavanje softvera u pitanju ono predstavlja oko 67% ukupnog troška razvoja softvera. Troškove održavanja možemo podijeliti na perfektivno održavanje koje se odnosi na promjene, poboljšanja te proširenja softvera čineći oko 50% troškova održavanja softvera. S druge strane adaptivno i korektivno održavanje čine između 20% i 25% troškova dok preventivno čini nešto manje od 5% svih aktivnosti održavanja. Pod troškove održavanja možemo uključiti uložen rad za njegovo provođenje, novac uložen u taj proces te ono najbitnije, utrošeno vrijeme.

Iako su ukupni troškovi ostalih faza razvoja softvera brzo porasli, omjer kretanja troškova održavanja u odnosu na troškove faza razvoja je ostao relativno isti. To je rezultat porasta ovisnosti mnogih organizacija o softveru, za što je zaslužan produljeni život softvera i sve veća složenost i veličina pojedinih aplikacija. Nažalost procjenjuje se da je prosječna starost softvera između 10 i 15 godina (Software Maintance Overview, 2019). Iz toga se razloga može reći da se stariji softveri teže održavaju od novijih zbog rasta sustava tijekom vremena pri čemu dolazi i do loše organiziranosti te manje razumljivosti samog softvera. Kontinuirani napredak u suvremenoj tehnologiji programiranja (npr. moderni alati i metode za softversko inženjerstvo, jezici četvrte generacije, generatori aplikacija) omogućuje da se sofisticiranije sustave izrađuje u manjem trošku. Procjena ove je da što je sustav korisniji, duže će biti i u uporabi pa će time i izazvati veći trošak u toku svoga vijeka trajanja s obzirom na potrebu za još većom prilagodbom modernoj tehnologiji. Iz toga će razloga trošak potreban za održavanje nekog softvera kroz njegov životni vijek zbog razvitka tehnologije biti veći. Na taj su način i neke vrste aplikacija teške za održavanje te se najčešće s vremenom mijenjaju novijima (Osborne, 1985).

Kontrola ukupnih softverskih troškova se temelji na sveobuhvatnom i učinkovitom upravljanju cijelim procesom. Što je veća disciplina uključena u održavanje softvera, ono će rezultirati većom kvalitetom softvera. Ponovna uporaba postojećih softvera koja uključuje uporabu vanjskih paketa također može pomoći u spuštanju troškova. Od ostalih rješenja valja istaknuti upotrebu ne-proceduralnih jezika, veću upotrebu moderne tehnologije, alata i metoda softverskog inženjerstva te dobru strukturiranost sustava. Učinkovitije kontrole upravljanja također nude mogućnosti smanjenja i kontrolu troškova pri održavanju softvera (Mader & Egyed, 2012) .

12. ODRŽAVANJE I EVOLUCIJA

Održavanje softvera obično ne uključuje velike promjene u arhitekturi sustava. Te se promjene provode izmjenom postojećih komponenti i dodavanjem novih komponenti u sustav. Evolucija softvera je pak širi pojam koji uključuje i održavanje softvera i veće promjene u različitim fazama životnog ciklusa softvera. Možemo reći da je evolucija svojstvena samoj prirodi razvoja softvera. S druge strane pod održavanjem podrazumijevamo planirani rutinski dio procesa koji se sastoji od manjih izmjena dok evolucija može dovesti do mnogo bitnijih i većih promjena na samom softveru.

Nakon što se proizvod pusti u rad, poznato je kako je taj softverski sustav potrebno konstantno mijenjati i prilagođavati ne bi li se softver održao u koraku primarno sa korisničkim zahtjevima. Ovdje u obzir treba uzeti i promjene poslovnog okruženja kao i napredak hardvera. Upravo taj proces u kojem se softver mijenja naziva se održavanje, ali je isto tako poznat i kao evolucija. Naime, ove se riječi vrlo često koriste kao sinonimi. Međutim potrebno je istaknuti njihove razlike. Dakle, kada se govori o održavanju softvera ono čini planirani rutinski dio cjelokupnog procesa koji se dakako sastoji od ponešto manjih izmjena. Kada se govori o evoluciji ona može doprinijeti neke bitnije, ali ujedno i veće promjene iako postoje i neke općenite definicije koje je potrebno navesti. Za održavanje softvera se može reći da je to modifikacija softverskog proizvoda nakon isporuke radi ispravljanja pojedinih kvarova, odnosno grešaka, poboljšavanje performansi ili drugih atributa te prilagođavanje proizvoda modificiranom okruženju. Ovakva definicija je definirana u IEE standardu. Sličnu definiciju daje ISO/IEC (ISO/IEC, 2006) , on navodi kako se softverski proizvod podvrgava modifikaciji koda i pridruživanju dokumentacije zbog problema ili potrebe za poboljšanjem. Cilj je izmijeniti postojeći softverski proizvod uz očuvanje njegove cjelovitosti. Pojam evolucija softvera neki istraživači i praktičari koriste kao preferiranu zamjenu za održavanje, kako je prethodno i navedeno (Bennett & Rajlich, 2000). Naravno, održavanje i razvoj softvera odlikuju ogromnim troškovima i malom brzinom implementacije. Ipak, to su neizbježne aktivnosti, pa samim time gotovo svaki koristan i uspješan softver prati korisničke zahtjeve koji moraju težiti ka promjenama i poboljšanjima. Potrebno je da se prate trendovi te da se softver mijenja i unaprjeđuje u skladu s potrebama budućnosti (Bennett & Rajlich, 2000).

Promjena softvera predstavlja osnovni postupak evolucije softvera i servisa softvera.

Prilikom održavanja softvera itekako je potrebna evolucija uključujući procese, organizaciju i ljudske aspekte (Bennett & Rajlich, 2000).

Današnje društvo informacijske tehnologije sve se više oslanja na softver na svim razinama. Unatoč tome, kvaliteta softvera općenito i dalje ne zadovoljava očekivanja, a softverski sustavi i dalje pate od simptoma starenja jer su prilagođeni zahtjevima i okruženju koji su promjenjivi. Jedini način da se prevladaju ili izbjegnu negativni učinci starenja softvera je postavljanje promjena i evolucije u središte procesa razvoja (Mens, i dr., 2005).

13. REFAKTORIRANJE

Kada se govori o refaktoriranju, možemo reći da je to proces promjene softverskog sustava na takav način da taj softver ne mijenja vanjsko ponašanje koda, a ipak poboljšava njegovu unutarnju strukturu. To je discipliniran način čišćenja koda koji umanjuje šanse za unošenje grešaka. U suštini kada se provodi „refaktor“ zapravo se poboljšava dizajn koda nakon što je kod već napisan. Pojednostavljeno rečeno, to je promjena u unutarnjoj strukturi softvera kako bi se došlo do lakšeg razumijevanja te jeftinijeg i modificiranijeg koda.

„Je li refaktoriranje samo čišćenje koda?“ Na neki je način odgovor „da“, ali refaktoriranje ide dalje jer pruža učinkovitiju tehnologiju čišćenja koda ali ne kontroliran način. Prije svega svrha refaktoriranja je da se napravi kod koji će se lakše razumjeti i mijenjati.

U trenutnom razumijevanju razvoja softvera vjerujemo da najprije dizajniramo, a zatim šifriramo (kodiramo). Dobar dizajn je na prvom mjestu, a kodiranje na drugom. S vremenom će se kod mijenjati kao i integritet sustava, a njegova će struktura prema tom dizajnu postupno izgubiti početni smisao. Postupkom refaktoriranja moguće je preurediti loš dizajn pa tako i cjelokupan kod. Svaki je korak jednostavan, čak i pojednostavljen. Stoga, kumulativni efekt čak i malih promjena može radikalno poboljšati dizajn. To je upravo obrnuta normalna predodžba propadanja softvera. Refaktoriranjem pronalazimo radnu ravnotežu koja se mijenja. Nastala interakcija dovodi do programa s dizajnom koji ostaje dobar i razvoj se nastavlja.

Za refaktoriranje je od ključne važnosti provjeriti da li postoje čvrsti testovi. Ti testovi daju sigurnost tijekom mijenjanja programa kasnije. Između ostaloga ti testovi mogu poslužiti i kao samoprovjera (Fowler, Beck, Brant, & Opdyke, 2002).

13.1. Zašto bi se trebalo koristiti refaktoriranje i koje su mu prednosti

Refaktoriranje nam pomaže u rješavanju problema nastalih na softveru. Istovremeno je to i vrijedan alat koji uvelike može pomoći pri boljem razumijevanju koda. To je alat koji se može i treba koristiti u više svrha. Refaktoriranjem se poboljšava dizajn softvera odnosno bez refaktoriranja dizajn programa propada. Refaktoriranje je nalik na uređivanje koda. Radi se na uklanjanju bitova koji stvarno nisu na pravom mjestu. Gubitak strukture koda ima kumulativan učinak. U tom je slučaju teže vidjeti dizajn u kodu, teže ga je sačuvati i jako brzo propada. Ponekad može biti jako teško vidjeti, a zatim i sačuvati dizajn u kodu, što kao posljedicu

uzrokuje njegovo propadanje. Upravo redovito refaktoriranje pomaže kodu zadržati oblik. Naglasak se stavlja i na duplikate u kodu koje je potrebno ukloniti. Eliminacijom duplikata osigurava se da kod sve kaže jednom i samo jednom što je suština dobrog dizajna.

Refaktoring može pomoći u povećanju čitljivosti koda. Kreće se od koda koji radi, ali nije idealno strukturiran. Malo vremena potrošenog na refaktoriranje može povećati svrhu cjelokupnog koda. Uz pomoć refaktoringa programeri će lakše i u puno kraćem vremenu shvatiti tuđi kod. Refaktoriranje zapravo ljudima pomaže da budu puno učinkovitiji u pisanju robusnog koda. U konačnici refaktoriranje pomaže da se razvije puno više koda na puno brži način. Ujedno se poboljšava kvaliteta koda, poboljšava se dizajn, čitljivost te se smanjuju greške (Fowler, Beck, Brant, & Opdyke, 2002).

13.2. Problemi sa refaktoriranjem

Kada se usvoji tehnika koja uvelike poboljšava produktivnost, teško je uvidjeti kada se ta produktivnost umanji. Problem se javlja kada se refaktoriranje primjeni samo na jednom projektu. Na taj način je teže uvidjeti što zapravo može uzrokovati da tehnika bude manje učinkovita ili pak štetna. Međutim kod refaktoriranja vrijede neka ograničenja.

Kao problematično područje za refaktoriranje možemo istaknuti baze podataka. Većina poslovnih aplikacija usko je povezana za shemu baze podataka koja ih podržava. To je jedan od razloga zašto je bazu podataka teško promijeniti. Drugi razlog je migracija podataka. Čak i ako se je sustav dizajniran da minimizira ovisnost između sheme baze podataka i objektnog modela, mijenjajući shemu baze podataka prisiljava se na migraciju podataka, što može biti dug i veliki zadatak. S bazama podataka koje ne uključuju objekte način za rješavanje ovog problema je postavljanje zasebnog sloja softvera između određenog objektnog modela i postojećeg modela baze podataka. Na taj način je moguće izolirati promjene dva različita modela. Dok se ažurira jedan model, ne ažurira se drugi. Čak i ako se ne radi o refaktoriranju treba obratiti pažnju kada se radi sa bazama podataka ili nekim složenim modelom nad kojim nemamo kontrolu. Neke objektno orijentirane baze podataka omogućuju automatsku migraciju s jedne verzije objekta na drugi. To smanjuje napor, ali ipak je potrebno da prođe neko vrijeme, jer migracija podosta traje. Veći problem se dešava kada spomenuta migracija nije automatska, već je potrebno to ručno učiniti, što iziskuje puno veći napor (Kovaček, 2016).

14. ULOGA REUSA U ODRŽAVANJU

Ponovna upotreba softvera predstavlja proces stvaranja softverskih sustava iz unaprijed definiranih komponenti softvera. Uz pomoć reusa omogućeno je sustavni razvoj komponenta koristiti za višekratnu upotrebu. Upravo sustavna upotreba ovih komponenti može poslužiti za stvaranje novih sustava.

Komponenta za višekratnu upotrebu može biti kod, ali veće prednosti ponovne uporabe dolaze iz šireg i višeg pogleda na ono što se može ponovno upotrijebiti. Kandidati za ponovnu upotrebu mogu biti: specifikacije softvera, dizajn, testni primjeri, podaci, prototipovi, planovi, dokumentacija, okviri te predlošci. Ponovna upotreba softvera može smanjiti vrijeme i troškove razvoja softvera.

Uz pomoć reusa dolazi do veće produktivnosti softvera, smanjuje se vrijeme njegovog razvoja, na razvoj softvera može se uključiti manji broj ljudi, smanjuju se troškovi razvoja, ali i održavanja (Software Reuse, 2019).

Još jedna dobra stvar koja se javlja kod reusa je pouzdanost, ona se poboljšava s obzirom na prethodni proizvod te se time smanjuje rizik (Software Reuse Considerations, 2019).

15. ZAKLJUČAK

Kada je u pitanju održavanje softvera za vjerovati je da se praćenjem faza i pojedinih modela softvera vrlo uspješno mogu uklanjati greške i problemi na koje nailazimo u radu softvera primjenom redovitih nadogradnji. Procesom održavanja omogućena je prilagodba potražnji na tržištu uz mogućnost vršenja izmjena na softveru kako bi čim više odgovarao okolini i potrebama tržišta. Održavanje kao faza u izradi softvera može pružiti velika poboljšanja na softveru pa čak i spasiti softver od propadanja. Unatoč tome troškove ni na kakav način nije moguće izbjeći. Kroz ovaj je rad razjašnjen značaj održavanja za dugotrajnost upotrebe te kvalitetu softvera na kojem se ono primjenjuje. Da bi se to postiglo potrebno ga je provesti prema planu i proceduri koja je prethodno isplanirana. Prema svemu navedenom zaključujemo kako je održavanje softvera uistinu jedan od bitnijih faza izrade softvera pri čemu mu je primarna svrha omogućiti iskoristivost i prilagodbu softvera kroz duži vremenski period.

16. POPIS SLIKA

17. LITERATURA

Al-sharif, I. (05 2015). Reduction of software perfective and corrective maintenance cost. Preuzeto 07. 09 2019

Anquetil, N., Oliveira, K., Dias Greyck, M., Ramal, M., & Meneses, R. (08. 07 2015). Knowledge for Software Maintenance. Preuzeto 20. 07 2019

Bennett, K. H., & T., R. V. (2000). Software Maintenance and Evolution: a Roadmap. Preuzeto 09. 09 2019 iz http://www.cs.wayne.edu/~severe/publications/Bennett.ACM.2000.Roadmap.pdf

Binkley, D., & Lawrie, D. (2010). Maintenance and Evolution: Information Retrieval Applications. *Computer Science Department Loyola University* . Preuzeto 05. 09 2019 iz https://www.academia.edu/2852076/Maintenance_and_evolution_Information_retrieval_applications

Blašković, K. (2016). Održavanje softvera: kvalitetna implementacija izmjena u softveru. Preuzeto 07. 09 2019

Boehms Software model. (2019). Preuzeto 09. 09 2019 iz tutor: https://www.tutorhelpdesk.com/homeworkhelp/Computer-Science-/Boehms-Software-Model-Assignment-Help.html

Bourque, P., & Richard, F. E. (2004). *Guide to the Software Engineering Body of Knowledge.* IEE computer society. Preuzeto 05. 09 2019 iz http://beamphys.triumf.ca/info/SWEBOKv3.pdf

Fowler, M., Beck, K., Brant, J., & Opdyke, W. (2002). Refactoring: Improving the Design of Existing Code. Preuzeto 09. 09 2019

Gupta, A., & Sharma, S. (01. 01 2015). Software Maintenance:Challenges and Issues. Preuzeto 07. 09 2019 iz http://www.ijcse.net/docs/IJCSE15-04-01-037.pdf

Hunt, B., Turner, B., & McRitchie, K. (2008). Software Maintenance Implications on Cost and Schedule. Preuzeto 05. 09 2019 iz https://885ccec9-a-62cb3a1a-s-sites.googlegroups.com/site/cs602softwareengineering/reading-materials/softwaremaintenanceimplications.pdf?attachauth=ANoY7cojF8gcOtEy4Aem7BpamxLPJKH_uW7I0z8KNzinVUnxtgSx0UkrSrYEfMBQzh9UfITGy42-dTTh_HRkrmFRG5qTE1hIeK99jZcWBDg

Ibrahim, K. S., & Yahaya, J. H. (2017). Towards the Quality Factor of Software Maintenance Process. Preuzeto 07. 09 2019

Incremental Model or iterative enhancement model in software engineering. (n.d.). Preuzeto 08. 09 2019 iz Ecomputer Notes: http://ecomputernotes.com/software-engineering/incremental-model

ISO/IEC. (09 2006). Preuzeto 20. 07 2019 iz ISO: https://www.iso.org/standard/39064.html

Jovic, A., Horvat, M., Danko, I., & Frid, N. (2015). *Procesi programskog inženjerstva.* Preuzeto 09. 09 2019

Khanna, S., Shah, A., Jain, S., & Ramanathan, L. (2017). *International Journal of Advanced Research in Computer Science.* Preuzeto 13. 09 2019 iz ijarcs: http://www.ijarcs.info/index.php/Ijarcs/article/view/3149/3125

Kontogiannis, K. (2011). Techniques for Software Maintenance. Preuzeto 13. 09 2019 iz http://www.softlab.ntua.gr/~kkontog/publications/books/b1-2011.pdf?fbclid=IwAR0wRBmT3SvmB00UFpnetgdX0VJO12-LD3RV5iZw3CO7aIrHptlAsD0Ns8U

Kovaček, M. (09 2016). *Ekstremno programiranje.* Preuzeto 09. 09 2019 iz https://bib.irb.hr/datoteka/835422.kovacek_mateo_unipu_2016_zavrs_sveuc.pdf

Krneta, P. (13. 10 2017). *Održavanje Poslovnog Softvera.* Preuzeto 28. 08 2019 iz https://www.hgk.hr/documents/predavanje-hgk59e0a9a3ba5a6.pdf

Lemeš, S., & Buzađija, N. (2018). Standardni model održavanja softvera. Preuzeto 20. 08 2019 iz http://odrzavanje.unze.ba/zbornici/2018/032-O18-030.pdf

Lientz and Swanson on Software Maintenance. (2011). Preuzeto 20. 08 2019 iz DZone: https://dzone.com/articles/lientz-and-swanson-software

Lientz, B. P., & B., S. E. (1980). Software Maintenance Management. Preuzeto 20. 08 2019

Mader, P., & Egyed, A. (2012). AssessingtheEffectofRequirementsTraceabilityforSoftwareMaintenance. Preuzeto 07. 09 2019 iz http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.361.6997&rep=rep1&type=pdf

Mens, T., Wermelinge, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. (2005). Challenges in Software Evolution. Preuzeto 09. 09 2019

Niessink, F., & Vliet, V. H. (2000). Software Maintenance from a Service Perspective. Preuzeto 07. 09 2019 iz https://www.cs.vu.nl/~hans/publications/y2000/SMRP00-2.pdf

Osborne, W. M. (1985). *Computer Science and Technology*. Preuzeto 20. 07 2019 iz https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nbsspecialpublication500-130.pdf

Royce, W. W. (1970). Managing the development of large software system. Preuzeto 20. 08 2019 iz http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

Shivani. (07 2015). Detailed Study of Software Maintenance Model. Preuzeto 08. 09 2019 iz https://www.airo.co.in/paper/admin/upload/international_volume/6968Shivani%20International_%20vol%205.pdf

Singh, Y., & Saha, A. (01 2010). Improving the testability of object oriented software through software contracts. Preuzeto 20. 08 2019

Software Case Tools. (n.d.). Preuzeto 10. 09 2019 iz Tutorialspoint: https://www.tutorialspoint.com/software_engineering/case_tools_overview.htm

Software Development Life Cycle (SDLC). (n.d.). Preuzeto 09. 09 2019 iz techopedia: https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc

Software Engineering - Quick-fix Model. (2013). Preuzeto 20. 08 2019 iz GeeksforGeeks: https://www.geeksforgeeks.org/software-engineering-quick-fix-model/

Software Maintance Overview. (2019). Preuzeto 09. 09 2019 iz Tutorialspoint: https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm

Software Maintenance. (09 2018). Preuzeto 13. 09 2019 iz ProfessionalQA.com: http://www.professionalqa.com/software-maintenance

Software Maintenance Models. (24. 08 2018). Preuzeto 07. 09 2019 iz Professional.QA.com: http://www.professionalqa.com/software-maintenance-models

Software Reuse. (2019). Preuzeto 09. 09 2019 iz Select Business Solution: http://www.selectbs.com/software-asset-management/software-reuse

Software Reuse Considerations. (2019). Preuzeto 09. 09 2019 iz Accendo Reliability: https://accendoreliability.com/software-reuse-considerations/

Stefanović, M., Mitrović, S., & Erić, M. (05 2006). Model kvaliteta softvera. Preuzeto 08. 09 2019 iz http://www.cqm.rs/fq2006/pdf/A/12%20-%20Stefanovic%20M.,%20Mitrovic%20S.,%20Eric%20M.pdf

Swanson, E. B., & Lientz, B. P. (1980). Software Maintenance Management. Preuzeto 20. 08 2019