VAŽNOST I NAČIN PRIMJENE SDLC OKVIRA U RAZVOJU SOFTVERSKIH RJEŠENJA

IMPORTANCE AND USAGE OF SDLC FRAMEWORK IN SOFTWARE DEVELOPMENT

Marija Šuško

Renato Barišić

Visoko učilište Algebra, Zagreb, Hrvatska/Croatia

E-mail: renato.barisic@racunarstvo.hr

UDK/UDC: 004.02+004.4'22

JEL klasifikacija/JEL classification: M15

DOI: 10.30657/hdmk.2020.39

Stručni članak/Professional paper

Jezik/Language: Hrvatski/Croatian

SAŽETAK

Životni ciklus razvoja softvera može se definirati kao vodič koji daje detaljan opis kako ispuniti korisnikove zahtjeve i ciljeve, planirati, izgraditi i održavati softversko rješenje te na taj način osigurati visoku kvalitetu i učinkovitost cijelog projekta. Konvencionalne aktivnosti SDLC okvira koje pomažu u ispunjavanju zahtjeva klijenta distribuiraju se kroz šest glavnih SDLC faza. U prvoj fazi prikupljanja i analize korisničkih zahtjeva, koju obično provode stariji članovi tima, moraju se prikupiti i razumjeti svi korisnikovi zahtjevi kako bi se analizom mogao kreirati dokument za daljnji dizajn i razvoj željenog softverskog rješenja te time pružiti veća mogućnost uspješnosti projekta. U fazi dizajna se, prema dokumentu o specifikacijama korisničkih zahtjeva, detaljno dizajniraju sistemski i softverski dokument kako bi se što lakše i uspješnije mogla izvršiti iduća i najduža faza u kojoj se konačno, načinom ovisnim o odabranoj platformi i programskog jeziku, razvija i gradi softversko rješenje. Iako tijekom razvoja proizvoda programeri provjeravaju i testiraju softver kako bi provjerili rad aplikacije, potrebni su timovi testera koji će, u SDLC fazi testiranja, kreirati scenarije za provjeru radi li cijela aplikacija bez pogrešaka u skladu sa zahtjevima kupca. Konačno, nakon uspješno provedene faze testiranja razvijenog softverskog rješenja koje je ispunilo sve korisničke zahtjeve, isto se pušta u rad, postaje dostupno publici i započinje faza održavanja softverskog rješenja, u kojoj se popravljaju prijavljeni bugovi, ažurira i nadograđuje softversko rješenje, a faza traje sve dok ono postoji.

Ključne riječi: SDLC, softverska rješenja, okvir, metode razvoja softvera.

1. UVOD

U današnje vrijeme, zahvaljujući digitalnom svijetu u kojem živimo, web stranice više nisu mogućnost, nego nužnost za marketing i uspješno poslovanje. Web ima puno veći doseg nego bilo koji drugi oblik oglašavanja. „55% ljudi će prije kupovine pretraživati online recenzije i preporuke“.¹ Stoga, ne korištenjem web stranica, gubi se veliki broj mogućnosti u poslovanju, što posebice vrijedi za mala poduzeća: „19% vlasnika malih poduzeća, koji su bez

¹ <https://wpforms.com/the-ultimate-list-of-online-business-statistics/>, (pristupano 25.10.2019.).

web stranice, vjeruju da bi njihov posao porastao za 25% u tri ili manje godina kada bi imali prednost web stranica.² Upravo su zato traženi web developeri kako bi, putem interneta, na profesionalan način predstavili kompanije i njihove proizvode potencijalnim korisnicima. Cilj takvih web stranica, koje će privući i zadržati korisnike, je omogućiti kompanijama konkuriranje na tržištu. No biti dobar web dizajner nije jedini preduvjet za stvaranje kvalitetnih web stranica. Naime, važno je imati i sposobnost razumijevanja korisnikovog viđenja konačnog proizvoda i prikupljanja ključnih informacija za uspješnu implementaciju onoga što im je potrebno. Često korisnici nisu u poziciji davati zahtjeve jer nisu sigurni što točno žele ili pak ne razumiju u potpunosti način izrade traženih web stranica. U tom slučaju, potencijalne procjene i korisni prijedlozi developera su neiskoristivi, ako su utemeljeni na neusavršenim korisnikovim zahtjevima. Ignoriranje situacije, gdje su specifikacije web rješenja utemeljene na korisnikovim nerazjašnjenim ciljevima te kretanje u implementaciju takvog rješenja je upravo ono što će osigurati neuspjeh projekta. Također, napori web developera na redizajnu ili preuređivanju stranica su uzaludni ako ih korisnik nije zahtijevao, a opet treba biti spreman na promjene tijekom implementacije koje su gotovo neizbježne.

Sve su to mogući problemi i poteškoće kod planiranja i razvoja web rješenja za čije je razrješavanje potreban životni ciklus razvoja sustava (eng. Software Development Life Cycle - SDLC). Aktivnosti SDLC-a su usmjerene k određenom cilju te kao takve ne slijede metodologiju jednake veličine za sve nego se prilagođavaju potrebama korisnika. Prije kretanja u razvoj novog projekta potrebno je utvrditi koji su sve zahtjevi korisnika i identificirati kako će SDLC pokriti sve zahtjeve da se postigne najbolji rezultat. Zatim je potrebno odabrati najbolji SDLC model, ili kombinaciju modela, kako bi pristupili najboljem načinu izvršavanja SDLC-a. Cilj SDLC procesa je istaknuti najbolji način razvoja web rješenja, što će biti i prikazano kroz konkretan primjer izrade i implementacije web rješenja.

2. SDLC

„SDLC je proces koji se sastoji od niza planiranih aktivnosti za razvoj ili promjenu softverskih proizvoda.“³ Tijekom godina, različitim definicijama se pokušavalo što preciznije opisati SDLC okvir, ali temelj svakoj je bio isti, a to je skup aktivnosti, politika i procedura, odnosno kontroliranih faza SDLC-a, usmjerenih k zadovoljenju kupca traženim rješenjima. Prva faza SDLC okvira, koju čine prikupljanje i analiza korisnikovih zahtjeva, a služe postavljanju obujma projekta te planiranju unaprijed i predviđanju budućeg upravljanja projektom, početni su ključ uspjehu čitavog projekta.

„If you fail to plan, you are planning to fail (Benjamin Franklin)“. Većina organizacija koje, od cjelokupnog vremena koliko traje neki projekt, ne potroše dovoljno vremena na prvu fazu imaju stopu neuspjeha projekta od 70%.⁴ Razlog tome je veliki obujam projekta, a nerijetko i veliki broj timova koji, ako nemaju mehanizam za upravljanje i predodređene faze za praćenje razvoja projekta, neće znati koliko su popraćeni korisnikovim zahtjevima i kako napreduju k izvršenju projekta. Stoga je, prije nego što se uopće krene u razvoj novog projekta, potrebno odrediti što sve obuhvaća SDLC i odrediti pristup web rješenju na temelju nekih od SDLC modela, najprikladnijih za praćenje i upravljanje projektom te za razvoj i održavanje željenog web rješenja.

² <https://wpforms.com/the-ultimate-list-of-online-business-statistics/#online-businesses>, (pristupano 25.10.2019.).

³ <https://www.tutorialspoint.com/sdlc/index.htm>, (pristupano 3.11.2019.).

⁴ <https://4pm.com/2019/05/26/project-failure/>, (pristupano 3.11.2019.).

U rijetkim slučajevima se puštanje prvog web rješenja u rad može smatrati završenim. Razvoj web rješenja neprekidnim SDLC ciklusom, kao što je predloženo na Slici 1, omogućuje pronalaženje i ispravljanje pogrešaka te implementiranje dodatnih funkcionalnosti.

Slika 1: SDLC ciklus

SDLC proces vizualno predložava projekt i određuje okvir unutar kojeg će se izvršavati te omogućuje praćenje i kontrolu rada kako bi osigurali isporuku web rješenja, one kvalitete koja je u skladu s korisnikovim očekivanjima. Za razvoj takvog rješenja web developeri koriste strukturirane faze SDLC-a. Način izvršavanja faza SDLC-a ovisi o SDLC modelu odabranom za projekt, ali neovisno o modelu, za uspješnost svake SDLC faze neophodna je uspješnost prethodne joj faze, što će na koncu omogućiti uspješnost cijelog projekta.

3. PRIKUPLJANJE I ANALIZA KORISNIKOVIH ZAHTJEVA

Prvu i temeljnu fazu SDLC-a čine 2 dijela: analiza i prikupljanje korisničkih zahtjeva. Ova faza čini početak ciklusa, a započinje raspravama s korisnikom koje zakažu poslovni analitičar i projektni menadžer. Na njima sudjeluju stariji članovi tima, stručnjaci za domenu u industriji i svi dionici na projektu kako bi prikupili i zatim analizirali sve relevantne informacije koje će dati jasniju sliku opsega projekta te predviđenih pitanja, prilika i smjernica koje su uopće potaknule njegovo začecje. Većina organizacija koje, od cjelokupnog vremena koliko traje neki projekt, ne potroše dovoljno vremena na ove dvije faze i ne razjasne s korisnikom sve detalje oko projektnih zahtjeva, imaju stopu neuspjeha od 70%.⁵ Stoga, kretanje u razvoj softverskog rješenja, za koji nije postignuta potpuna suglasnost s korisnikom o njegovim zahtjevima i ciljevima, neće imati učinka i u većini slučajeva će rezultirati neuspjehom.

⁵ <https://4pm.com/2019/05/26/project-failure/>, (pristupano 7.12.2019.).

3.1. Prikupljanje korisničkih zahtjeva

Tri su osnovna pitanja koja se trebaju razjasniti sa svakim korisnikom u fazi prikupljanja zahtjeva:

- Što kupac točno želi?
- Tko će biti krajnji korisnik?
- Koja je svrha proizvoda?

Iznimno je važno razumjeti kakvo rješenje korisnik želi i steći znanja o proizvodu prije kretanja u razvoj jer se te informacije koriste u analizi korisničkih zahtjeva za planiranje osnovnog projektnog pristupa, osiguranja kvalitete proizvoda i prepoznavanja potencijalnih rizika. Naime, potencijalne procjene i korisni prijedlozi developera bili bi neiskoristivi, ako su utemeljeni na neusavršenim korisnikovim zahtjevima. Iz tog razloga je prijeko potrebno biti upoznat s vrstama klijenata i spreman na načine postupanja sa svakom od njih, kako bi se u ovoj fazi riješile sve nejasnoće. Postoje klijenti koji će zaboraviti na projekt netom nakon prvog sastanka, a postoje i oni koji će neprestano dolaziti s novim idejama i informacijama te time otežavati kontinuirani rad i razvoj. Klijent za kojeg se razvijalo višejezično web rješenje, spada u tip koji je došao na prvi sastanak s nekoliko konkretnih informacija, ali se oslanjao na to da će mu programer moći predložiti puno bolje rješenje od nečega što bi on sam smislio i time nije bio voljan odgovarati na mnoštvo pitanja. U slučaju susreta s još jednim tipom korisnika, koji ne zna što točno želi jer ne razumije način izrade željenog softverskog rješenja, potrebno je pružiti relevantne informacije i setom konkretnih pitanja voditi ga kroz bitne točke svakog projekta, kako bi on naposljetku uspio opisati precizne i detaljne zahtjeve.

3.2. Analiza korisničkih zahtjeva

Za kvalitetnu i uspješnu analizu korisničkih zahtjeva potrebna je suradnja i komunikacija poslovnog dijela firme i IT timova, uključujući i developere i testere. Prikupljenim informacijama od korisnika provodi se studija izvodljivosti proizvoda na različitim područjima (eng. feasibility study):

- Ekonomskim;
- Pravnim;
- Operativnim;
- Tehničkim;
- Rasporednim.

Rezultat studije tehničke izvodljivosti je mehanizam za upravljanje koji će olakšati razvojnom timu praćenje napretka projekta i izvršavanja korisničkih zahtjeva. Kako bi se prilikom određivanja razvojnog pristupa mogle uspostaviti razumne prekretnice (eng. milestones), poželjno je svim korisničkim zahtjevima pridružiti prioritete koji će odrediti redoslijed nadogradnje pojedinih značajki softverskog rješenja. Korisnik ima glavnu ulogu u određivanju prioriteta zahtjevima, ali je moguća preraspodjela istih od strane developera s obzirom na kompatibilnost značajki u razvoju.

Ovu fazu se može nazvati i „Fazom planiranja“ jer svaki uspješan projekt započinje planiranjem razvoja softverskog rješenja, koristeći se zahtjevima prikupljenim od korisnika, kako bi se postavio jasan opseg projekta i predložilo korisniku što ulazi u njegove granice, a čime se ista prekoračuje. Za to je ključan i nezaobilazan korak, koji čini osnovu cijelog projekta, izrada SRS (eng. Software Requirements Specification) dokumenta. Ključne informacije koje SRS treba sadržavati su:

- Koja je svrha softverskog rješenja;
- Opći opis softverskog rješenja;
- Specifični zahtjevi i značajke koje softversko rješenje treba ispuniti;

- Ako ih ima, pitanja nerazjašnjena tijekom prikupljanja korisničkih zahtjeva.

Dokument treba biti spreman za pregled na 2. sastanku s korisnikom kako bi se mogle raspraviti i razjasniti sve moguće nejasnoće te odgovoriti na pitanja od strane developera i/ili korisnika. Ovim načinom se treba osigurati da svaki sudionik projekta, uključujući neizbježno i kupca, razumije sve zahtjeve i zadatke projekta, način na koji će isti biti razvijan i održavan tijekom cijelog životnog ciklusa te služiti kao referenca koja će spriječiti moguće buduće sukobe.

4. DIZAJN RJEŠENJA

Zahtjevi prikupljeni u prethodnoj fazi i analizirani SRS dokumentom su input informacije za formuliranje najbolje arhitekture sustava koja će omogućiti prijelaz u sljedeću fazu gdje će se razvijati i implementirati softversko rješenje. Dokument o specifikaciji softverskog razvoja služi kao referenca za koncipiranje DDS (eng. Design Document Specification), dokumenta specifikacije dizajna softvera. Za kompletiranje DDS dokumenta razmatra se jedan ili više dizajnerskih pristupa izradi arhitekture softverskog rješenja. Posredstvom svih dionika i evidencijom utjecajnih parametara izabire se najbolji pristup. Parametri koji se uzimaju u obzir:

- kompleksnost i veličina proizvoda;
- dizajn kompatibilan s korisničkim zahtjevima i uobičajenim standardima;
- potencijalni rješivi i nerješivi rizici;
- vremenska i financijska ograničenja.

DDS dokument jasno definira detaljan opis dizajna softverskog rješenja i svih modula predložene arhitekture. Takav dizajn se stvara postupkom odozgo prema dolje, tj. ispunjavanje zahtjeva i realiziranje značajki se kreće izvršavati na najvišoj razini te se zatim postepeno ulazi u detalje. Međutim ovakav dokument može se raščlaniti na dvije vrste projektnih dokumenata:

- HLD (eng. High Level Design) – visoka razina dizajna;
- LLD (eng. Low Level Design) – niska razina dizajna.

HLD sadrži popis svih modula s nazivima, kratkim opisom i funkcionalnostima te opisuje njihovu međusobnu ovisnost i odnos sa sučeljem, tablice i dizajn baze podataka kategorizirane ključnim elementima softverskog rješenja te arhitektonski dijagrami protoka i strukture podataka s pojedinostima o tehnologiji.

LLD definira funkcionalnosti sustava te logiku rada, inpute i outpute za svaki od njegovih modula. Sadrži sve detalje sučelja te dijagrame klasa sa svim metodama i međuodnosima istih, tablicu baza podataka s njihovom vrstom i veličinom te pristupa svim pogreškama (eng. errors) i problemima (eng. dependency issues) ovisnosti koji se mogu pojaviti. Fazom dizajna, kada govorimo o izradi web rješenja, može se nazvati i fazom izrade prototipova u kojoj, nakon što se shvate svi zahtjevi, okupljaju se programeri i softverski arhitekti kako bi mogli početi dizajnirati softversko rješenje na visokoj razini. Prije kretanja u proces dizajna web rješenja na više razina te odabira najboljeg pristupa i stvaranja arhitektonskog dizajna potrebno je razmotriti:

- Koje tehnologije je najpogodnije koristiti s obzirom na zahtjeve, značajke i komponente traženog web rješenja;
- Kakva je sposobnost tima i njihov tijek rada;
- Koji su potencijalni rizici i tehnički problemi koji se mogu pojaviti te načini njihova sprječavanja ili rješavanja;
- Uzeti u obzir komunikaciju s trećim stranama (eng. third party);
- Kakav je protok korisnika (eng. user flow);
- Rad s bazom podataka;
- Kakva su vremenska i financijska ograničenja proizvoda.

Nakon razmatranja svih faktora koji će utjecati na željeno web rješenje, isti se zapisuju konvencionalnim načinom u već spomenuti DDS dokument, a veliku ulogu u dokumentaciji ima poslovni analitičar. Prije kretanja u dizajn također je potrebno provesti analizu rizika arhitekture kojom se provodi revizija sigurnih praksa dizajniranja i sigurnih platformi koje neće biti podložne promjenama tijekom cijelog trajanja projekta. Wordpress ispunja i tu stavku jer ne spada u nove tehnologije za koje je upitno hoće li opstati za vrijeme izrade cijelog projekta, a s druge strane je popularna platforma čije web stranice prednjače na internetu.

Kako bi se zatim krenulo u proces dizajna provodi se evaluacija šablona za razvoj arhitekture aplikacija, razvoja softvera i rješavanje algoritamskih problema na dosljedan način. Ova faza uključuje izradu prototipa za usporedbu rješenja kako bi se pronašlo ono najbolje koje ispunjava tražene korisničke zahtjeve i uobičajene standarde. Iz ove faze proizlaze:

- Dizajnerski dokumenti s popisom šablona i komponenata određenih za projekt te prototip koje će se koristiti za daljnji frontend i backend razvoj;
- Počinje se s izradom strategije kojom će se testirati proizvod.

5. IZRADA RJEŠENJA

Nakon finaliziranja faze dizajna kreće se u iduću i najdužu fazu u kojoj započinje izravni razvoj softverskog rješenja. Nakon što developer dobije DDS dokument, s obzirom na odabranu tehnologiju za razvoj softverskog rješenja, priprema se potrebna platforma za razvoj, zadaci se razvrstavaju po jedinicama (eng. units) ili modulima te dodjeljuju pojedinim developerima, dizajn softvera se prevodi i generira u izvorni kod (eng. source code), i to vrlo jednostavno ukoliko je dizajn realiziran detaljno i temeljito, a developeri slijede smjernice za kodiranje definirane njihovim organizacijama i programskim alatima. Developeri za izgradnju cijelog sustava odabiru, s obzirom na vrstu softvera koji se razvija, jedan od programskih jezika visoke razine, koji mogu uključivati:

- C;
- C++;
- Java;
- JavaScript;
- HTML;
- CSS;
- PHP.

Nakon odabira prikladnog programskog jezika, developeri započinju s pisanjem koda, razvojem grafičkog sučelja i logikom interakcije sa serverom, a sistem administratori započinju postavljati softversko okruženje.

Ovisno o odabiru SDLC modela napredak ove faze, ispunjavanje korisničkih zahtjeva i značajki te usavršavanje softverskog rješenja se može odvijati postepeno u određenim vremenskim rokovima, tj. iteracijama kao što je praksa u Inkrementalnom i Agilnom modelu ili se cijelo softversko rješenje može implementirati odjednom, u ne ponavljajućim fazama razvoja, kao što je slučaj u Vodopad modelu, ali bez obzira na odabir SDLC metode, timovi developera trebaju implementirati sve komponente i proizvesti funkcionalan softver u što kraćem roku.

U ovoj fazi je neizbježno biti u kontaktu s korisnikom i svim dionicima uključenim u projekt kako bili u toku s razvojem željenog softverskog rješenja te pružali povratne informacije o tome razvija li se ono u skladu s njihovim očekivanjima.

Izlaz (eng. output) ove faze treba biti potpuno funkcionalan proizvod, spreman za fazu testiranja, neovisno o tome jesu li ispunjene sve značajke i zahtjevi ili će se isti dodavati u sljedećim iteracijama. Kako bi se omogućila provjera rada proizvoda testerima zaduženima za

pisanje testnih slučajeva, za pregled funkcionalnosti svih komponenata razvijenog softverskog rješenja u samoj fazi testiranja, iste sastavljaju te izvršavaju također i programeri kako bi provjerili rad i ispravnost novog koda te uopće mogli pustiti (eng. deploy) proizvedeni softver u testno okruženje.

6. TESTIRANJE RJEŠENJA

Fazu testiranja može se smatrati podskupom drugih faza SDLC modela danas jer se testiranje i konvencionalne testne aktivnosti provode u svim SDLC fazama. Nakon što programeri završe testiranje jedinica (eng. unit testing) u fazi razvoja i implementacije te proizvedu operativno te djelomično ili potpuno dovršeno softversko rješenje, s obzirom na odabrani SDLC model, ono se zatim može, zajedno sa svim svojim modulima, pustiti u testno okruženje gdje tim za testiranje započinje s provjeravanjem svih funkcionalnosti sustava i uspoređuje iste s propisanim projektnim dokumentima. Testeri se referiraju na SRS i DDS dokumente kako bi mogli provesti testiranje i pronaći nedostatke u softveru kojim će se uvjeriti da je softversko rješenje proizvedeno i radi u skladu s dokumentiranim standardima, značajkama i zahtjevima kupca. Iako je faza razvoja i implementacije softverskog rješenja najdulja, fazom testiranja se provjerava kvaliteta implementiranog proizvoda, je li isti ispunio korisnikova očekivanja, ima li nedostataka i bugova, „softverskih problema koji uzrokuju neispravan rad rješenja, a posljedica su pogrešne logike te je li korisničko sučelje prijateljski nastrojeno (eng. user-friendly) te je stoga ova faza ključna kako bi se dobilo osiguranje da je softversko rješenje valjano i spremno za puštanje u rad.⁶

Timovi testera su u neprestanoj komunikaciji s developerima tijekom provođenja temeljitog testiranja softvera i kada pronađu bug ili nedostatak softverskom rješenju, kreiraju izvješće te pronađeni problem prijavljuju i dodjeljuju nekome od programera. Programeri provjeravaju je li prijavljeni problem važeći te mu pristupaju i rješavaju ga na optimalan način, a zatim implementiraju novu verziju softvera koju vraćaju timu testera zajedno s ažuriranom dokumentacijom na ponovno testiranje. Svaka nova verzija softvera iziskuje i nova testiranja kojima se treba utvrditi kako su ispravljeni svi pronađeni bugovi, a da ispravljanje pri tome nije utjecalo na druge značajke i funkcionalnosti.

Testiranje i regresijsko testiranje softverskog rješenja se može smatrati manjim internim životnim ciklusom testiranja softverskog rješenja jer se ono ponavlja sve dok:

- tim testera ne pronađe što je moguć veći broj bugova i nedostataka;
- razvojni tim ne riješi sve prijavljene probleme;
- ne postigne se stabilno softversko rješenje;
- ne ispune se svi korisnikovi zahtjevi i značajke te proizvod bude u skladu s dokumentiranim potrebama softverskog rješenja.

Konačan output ove faze je u potpunosti funkcionalan proizvod spreman za puštanje i uporabu u proizvodnom okruženju.

Niz testova se primjenjuje na implementiranom proizvodu kako bi se prijavili i popravili bugovi, a zatim isti testovi i ponovili, čime se stvara ciklus koji se provodi sve dok se ne postigne konačan proizvod koji je bez nedostataka, stabilan i radi kako je definirano projektnom dokumentacijom.

⁶ <https://www.techopedia.com/definition/24864/software-bug->, (pristupano 16.01.2020.)

7. PUŠTANJE RJEŠENJA U RAD

Nakon što se uklone i korigiraju svi bugovi te uspješno provede i završi faza testiranja, tada započinje završni proces izdavanja i puštanja softverskog rješenja u rad (eng. deployment), kojem je konačan output stavljanje rješenja u proizvodno okruženje čime ono postaje dostupno publici.

Testiran proizvod, koji je bez bugova i spreman za uporabu se pušta u odgovarajuće okruženje, a cijeli proces puštanja softverskog rješenja u rad se, ovisno o poslovnoj strategiji pojedinog poduzeća i očekivanju kupca, može odvijati u fazama. Primjerice, u poduzećima visoke razine zrelosti ova se faza niti ne osjeti nego se softversko rješenje pušta u rad onog trena kada se potvrdi kao dovršen proizvod i bez pogrešaka, a u poduzećima srednje ili niže zrelosti, te u nekim visoko reguliranim industrijama, proces uključuje prolaz kroz nekoliko stadija i dobivanje određenih odobrenja. Ukoliko je dogovor organizacije da se proizvod izda u ograničenom segmentu i testira u stvarnom poslovnom okruženju, provodi se testiranje prihvatljivosti korisnika UAT i stvara se replika proizvodnog okruženja u kojemu kupac zajedno s programerima provodi dodatno testiranje i provjerava ima li problema s puštanjem softverskog rješenja u rad. „Ispitivanje prihvatljivosti korisnika, poznato i kao beta ili testiranje krajnjeg korisnika, definira se kao testiranje softvera od strane korisnika ili klijenta kako bi se utvrdilo može li ga prihvatiti ili ne. Ovo je završno ispitivanje koje se provodi nakon što su funkcionalna, sistemska i regresijska ispitivanja završena.“⁷. Na temelju povratnih informacija, ako se ne prijave nikakvi problemi i kupac smatra da je aplikacija onakva kakvom je i zamišljena, proizvod se pušta u proizvodno okruženje takav kakav jest ili s predloženim poboljšanjima, ukoliko ih ima.

Neovisno o strategiji organizacije, proces i aktivnosti pripreme softverskog rješenja za pokretanje i puštanje istog u rad, svode se na instalaciju, konfiguraciju, testiranje i po potrebi unošenje promjena radi optimiziranja performansi softverskog rješenja. Navedene aktivnosti puštanja softverskog rješenja u rad omogućuju, prije svega, uštedu vremena i resursa te povećavaju sigurnost dobrim praksama. Olakšano je upravljanje softverskim rješenjem jer se mogu nadzirati aktivnosti korisnika te prepoznati, a samim time i riješiti problemi koji se pojave u radu softverskog rješenja. Istodobno, implementacija softverskog rješenja pomaže u ažuriranju istog automatiziranim pretragama potrebnih verzija. Za što uspješniju implementaciju softverskih rješenja su se, tijekom godina, razvile sljedeće dobre prakse:

- Implementacija popisa za puštanje softverskog rješenja u rad kojim se stvara proces rada i izvršavanja zadataka kako bi se osiguralo da se sprovedu svi koraci neophodni za uspješnu implementaciju softvera;
- Odabir ispravne metode za implementaciju kojom se neće prekoračivati predviđeni budžet, a ipak će se olakšati integraciju softverskog rješenja s postojećim lokalnim aplikacijama i alatima;
- Automatizacija postupka implementacije omogućuje izbjegavanje ručnog uvođenja i ažuriranja novih verzija softvera koje, osim što uzrokuje gubitak vremena, može izazvati pojavu brojnih grešaka i bugova;
- Usvajanje kontinuirane isporuke koje se postiže prvotno puštanjem aplikacije u prototipno okruženje gdje se provjerava funkcioniranje aplikacije i njeno ispunjenje korisničkih zahtjeva što konačno osigurava isporuku koda za potrebnu implementaciju;
- Korištenje poslužitelja kontinuirane integracije omogućava izbjegavanje nerijetkih potencijalnih problema integracije i stvara veću sigurnost rada razvijenog softverskog rješenja na stroju programera te time čini ključ uspješne agilne implementacije.

⁷ <https://www.softwaretestinghelp.com/what-is-user-acceptance-testing-uat/>, (pristupano 17.01.2020.).

Navedene aktivnosti i cijeli postupak implementacije i puštanja softverskog rješenja u rad olakšavaju prikladni alati, prilagođeni za rad na različitim platformama i vrstama infrastrukture, provođenjem automatskih ili predodređenih zadataka. Alati za puštanje softverskih rješenja u rad pogodni su posebice za same programere koji neprestano rade na poboljšanju i ažuriranju softvera za korisnike, a alati za implementaciju im olakšavaju upravo taj proces te kontinuiranu integraciju. SolarWinds Patch Manager, Octopus Deploy, Jenkins i Bamboo navode se kao 4 visoko specijalizirana alata za puštanje softverskih rješenja u rad u 2020. godini.⁸

Implementacija softverskog rješenja pruža mnoge prednosti od kojih, prije svega, treba navesti unaprjeđenje poslovanja organizacije jer ono olakšava i ubrzava poslovne procese, pomaže s upravljanjem velikim podacima, korisnici mogu pristupiti softverskom rješenju s bilo kojeg uređaja u bilo kojem vremenu te pomaže u integraciji s Internetom stvari (eng. Internet of things) jer osigurava povezanost korisničkih i drugih aparata koji olakšavaju život korisniku. Zahvaljujući provođenju implementacije konvencionalnim praksama, ista se automatizirala te omogućuje automatsko ažuriranje novih verzija softvera te uvođenje softvera novim korisnicima i uređajima. Kako bi se smanjilo traženje pomoći od strane korisničke podrške, korisnicima se olakšava samostalna instalacija softverskog rješenja.

Međutim, iako se provode brojna testiranja te ispravci pogrešaka i bugova nije moguće niti jednim od načina testiranja pronaći sve uzroke problema, što fazu puštanja softverskog rješenja u rad ne čini posljednjom fazom SDLC ciklusa. Održavanje softverskog rješenja čini posljednju fazu SDLC-a jer će se u njoj prijaviti i ispraviti krajnji problemi softverskog rješenja, a po potrebi uvoditi te ugrađivati nove funkcionalnosti i značajke.

8. ODRŽAVANJE

Nakon što su ispunjeni svi korisnički zahtjevi i značajke softverskog rješenja te je provedeno testiranje i ispravljene su sve pogreške, proizvod se pušta u rad, postaje dostupan publici na korištenje i započinje faza održavanja softverskog rješenja. Ovom fazom ne završava SDLC ciklus jer ona traje sve dok postoji softversko rješenje i dostupno je korisnicima te ju se stoga naziva i „krajem početka“.

Softversko rješenje u radu potrebno je neprestano nadzirati kako bi se osigurao pravilan rad njegovim korisnicima. Tri su neizbježna događaja tijekom ove faze:

- Pojavljivanje, prijava te ispravljanje bugova i nedostataka u scenarijima koji se nisu mogli predvidjeti i riješiti u fazi testiranja;
- Nadogradnja (eng. upgrade) novih verzija softverskog rješenja;
- Poboljšanje (eng. update) postojećeg softverskog rješenja dodavanjem novih značajki.

Prilikom ispravljanja bilo kakvih pogrešaka i bugova kreiraju se izvješća koja se predaju programerima i premda se ne može ponovno prolaziti kroz cijeli SDLC ciklus, potrebna je barem skraćena verzija postupka kako bi se spriječila regresija i pojavljivanje novih poteškoća. Kupci tijekom korištenja proizvoda prijavljuju pronađene pogreške te se ovisno o razini poteškoće i hitnosti programeri odlučuju o vremenu njihovog popravka te nadogradnje i poboljšanja softverskog rješenja. Iako ova faza SDLC-a može biti u nekim slučajevima pasivna za razvojne timove i timove testera, neizbježno je otkriti sva pitanja i probleme koji se pojavljuju kod korisnika kako bi se osiguralo nesmetano funkcioniranje i korištenje softverskog rješenja.

⁸ <https://www.dnsstuff.com/software-deployment-tools>

Različite načine održavanja softverskih rješenja može se podijeliti u četiri osnovne skupine po načinu kojim održavaju softverska rješenja te prema postotku zastupljenosti tipa i utrošenog vremena:⁹

- Korektivni tip održavanja čiji pristup se koristi za ispravljanje pronađenih problema, pogrešaka i bugova koji otežavaju ili onemogućavaju upotrebu softvera, zastupljen je svega 20%;
- Prilagodljivi tip, zastupljen 25%, radi na održavanju softvera kako bi ga učinilo prilagodljivim radu na novim operacijskim sustavima i novim okruženjima;
- Perfektivni tip, najzastupljeniji od čak 50%, održava softver kako bi bio nesmetano funkcionirao i nakon dodavanja novih značajki softverskom rješenju;
- Preventivni tip održavanja, najmanje je zastupljen sa svega 5%, provodi promjene kojima se pokušava spriječiti pojavljivanje problema i pogrešaka.

9. ZAKLJUČAK

SDLC okvir, ukoliko se smisljeno i valjano iskoristi, pruža veliku mogućnost kontrole upravljanja projektom i nadgledanja razvoja softverskih rješenja. Svaki od SDLC modela pruža konvencionalne aktivnosti i faze koje se, ovisno o modelu drukčije spajaju, dok se sam model za razvoj softverskog rješenja odabire ovisno o karakteristikama i potrebama projekta, ali rezultat korištenja svakog od modela je isti, a to je olakšano upravljanje projektom i veća mogućnost za konačnim uspjehom. Detaljna dokumentacija, s pojašnjenim svim detalja projekta pruža jasne ciljeve i mehanizam rada, kako programerima i testerima koji rade na razvoju softverskog rješenja i testiranju, tako svim direktno i indirektno uključenim dionicima. SDLC ciklus se razvije na šest faza i stvori se plan rada i razvoja softverskog rješenja što olakšava detaljan nadzor, praćenje i izvršavanje svake od faza, a time i uspješno izvršenje cijelog projekta. Svako softversko rješenje je u početku samo ideja nekog korisnika koja, kad se provuče kroz faze SDLC-a, postaje funkcionalnim, iskoristivim i održivim proizvodom.

Abstract:

IMPORTANCE AND USAGE OF SDLC FRAMEWORK IN SOFTWARE DEVELOPMENT

Software development life cycle can be defined as a guide that provides a detailed description of how to meet client's requirements and objectives, plan, build and maintain a software solution and thus ensure high quality and effectiveness of the entire project. The SDLC framework conventional activities, that help in meeting client requirements, are distributed through six main SDLC phases. In the first phase of collecting and analysing user requirements, usually performed by senior team members, all user requirements must be collected and understood to enable the analysis to create a document for the further design and development of the desired software solution and thus provide a greater chance of project success. In the design phase, a system and software document is designed in detail, according to the user requirements specification document, so that the next and the longest phase can be completed as quickly and successfully as possible, in which the software solution is finally developed and built in a manner dependent on the chosen platform and programming language. While developers are validating and testing software during product development to test the application, test teams are required to, in the SDLC testing phase, create scenarios to verify that the entire application is bug-free according to customer requirements. Finally, after successfully completing the testing phase for the developed software solution, which eliminated all user requirements, the software is released, made available to the public and the software solution maintenance phase begins, in which the reported bugs

⁹ <http://ecomputernotes.com/software-engineering/types-of-software-maintenance>

are repaired, the software solution is updated and the phase is completed. while the software solution exists. Said phases appear and act in various ways in numerous SDLC models, some of which will be covered in detail in this bachelor thesis.

Key words: *SDLC, software, framework, software development methodologies*

10. LITERATURA

1. Langer, A. M., *Guide to Software Development - Designing and Managing the Life Cycle*, London, Springer, Verlag, 2012.
2. Barišić, R., *Upravljanje informacijskim sustavima*, Visoko učilište Algebra, Zagreb, 2018.
3. Krakar, Z., *Korporativna informacijska sigurnost, (odabrana poglavlja revizije informacijskih sustava)*, Fakultet organizacije i informatike, Varaždin, 2009.
4. <https://online.husson.edu/software-development-cycle/>, pristupano studeni 2019.
5. <https://www.tutorialspoint.com/sdlc/index.htm>, pristupano studeni 2019.
6. <https://4pm.com/2019/05/26/project-failure/>, pristupano studeni 2019.
7. https://www.tutorialspoint.com/sdlc/sdlc_overview.htm, pristupano studeni 2019.