Model životnog cirkusa softvera - sekvencijske i agilne metode

Podreka, Patrick

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

PATRICK PODREKA

Model životnog ciklusa softvera – sekvencijske i agilne metode

Završni rad

Pula, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

PATRICK PODREKA

Model životnog ciklusa softvera – sekvencijske i agilne metode

Završni rad

JMBAG: 0303063382, redoviti student
Studijski smjer: Informatika

Predmet: Programsko inženjerstvo
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Tihomir Orehovački

Pula, travanj 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za magista ekonomije/poslovne ekonomije, smjera Informatika ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom "Model životnog ciklusa softvera, sekvencijske i agilne metode" koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

SADRŽAJ

1.UVOD

Tema ovoga rada tiče se modela životnog ciklusa softvera. Konkretnije, ona obrađuje osnovne značajke i razlikovna obilježja sekvencijskih i agilnih metoda, kao jednog užeg područja u okviru ove kompleksne i multidimenzionalne problematike. S obzirom na značaj softverskog inženjerstva u suvremeno doba, kao i specifičnosti predmetne problematike, smatra se kako je riječ o jednoj aktualnoj problematici, čija razrada biva od iznimnog značaja kako za poslovni svijet, tako i za svakodnevni život i nesmetano funkcioniranje globalnog društva.

Softversko inženjerstvo suštinski je vrlo kompleksno znanstveno područje u okviru informatike, a o njemu danas postoje brojne inozemne i domaće rasprave, kao i svojevrsna istraživanja te slično. Najjednostavnije je reći kako je riječ o faznom procesu izrade učinkovitog i efikasnog softvera, a koji se zasniva na implementaciji nekoliko znanosti i znanstvenih područja, uglavnom inženjerstva i matematike.

Model životnog ciklusa ukazuje na inženjerski model izrade softvera, kao temeljnog elementa informatike i informatičkog djelovanja. Osim kao proces, može se pojmiti kao zasebna disciplina, a to se potvrđuje činjenicom da ovaj proces započinje identificiranjem neke potrebe ili konkretnog problema. Nakon toga, djeluje se kroz nekoliko faza, s ciljem stvaranja takvog softvera koji će otkloniti problem ili zadovoljiti potrebu.

Uzme li se u obzir činjenica da je softver pokretač ili temelj informatike i bilo kakvog procesa ove prirode, jasno je kako je model njegova životnog ciklusa, od razvoja do sazrijevanja pa redom dalje, jedan od temeljnih modela i procesa u ovome području. Stručnjacima i znanstvenicima ovoga područja na raspolaganju su različite metode, a pri tome se misli svakako na sekvencijske i agilne koje se razlikuju po mnogočemu, a nude niz mogućnosti i izbora.

Cilj ovoga rada je istražiti značenje predmetne problematike na cjelovit i kvalitetan način. Pri tome, cilj je razmotriti problematiku softverskog inženjeringa, a u okviru istoga značaj, obilježja i specifičnosti modela životnog ciklusa softvera.
Svrha rada je ukazati na mogućnosti koje nude sekvencijske i agilne metode, za predmetne potrebe. Pored toga, posebna pažnja posvećuje se identificiranju osnovnih razlika među njima.

Strukturno je rad podijeljen u tri poglavlja, uvod i zaključak. Prvo poglavlje posvećeno je softverskom inženjeringu, a u okviru njega obrađuju se odnosne definicije, životni ciklus softvera te metodologija, normizacija i standardi. Sljedeće poglavlje obrađuje sekvencijske metode na način da ih definira, specificira i pobliže određuje. Na jednaki način posljednje poglavlje obrađuje agilne metode, čime se ova problematika zaokružuje u smislenu cjelinu.

Za potrebe istraživanja korištene su metoda analize i sinteze, induktivna metoda, metoda komparacije i metoda apstrakcije. Predočeni tekst oblikovan je metodom deskripcije.

2.SOFTVERSKI INŽENJERING

Za potrebe istraživanja predmetne problematike rada značajno je dati jedan širi uvod u ovo područje. Pri tome, nezaobilazno je razmotriti značenje, interese i obilježja softverskog inženjeringa, koji se odnosi prvenstveno na izradu, razvoj i primjenu softverskih rješenja u praksi. Upravo se o tome raspravlja u okviru ovoga poglavlja. Točnije, daju se osnovne značajke u vezi softverskog inženjeringa, te se pristupa razradi životnog ciklusa softvera, te problematici metodologije, normizacije i standarda.

2.1. O SOFTVERSKOM INŽENJERINGU

Danas o softverskom inženjeringu, koji se naziva i programskim, postoje brojna djela, znanstvene i stručne rasprave, istraživanja i redom dalje. Sukladno tome, moguće je istaknuti nekoliko osnovnih definicija ovoga pojma ili vodećeg termina u području informatike. Najjednostavnije rečeno, softversko ili programsko inženjerstvo predstavlja znanstvenu i stručnu disciplinu koja se bavi svim aspektima ili segmentima proizvodnje softvera (Manger i Mauher, 2010: 8). Ovom se definicijom potvrđuje izravna poveznica između ovih termina, a ukazuje se da softverski inženjering suštinski podrazumijeva područje interesa i aktivnosti koje je usmjereno ka modelima, metodama i alatima koji su potrebni za proizvodnju kvalitetnih softverskih produkta ili outputa uz što niže troškove (Magner i Mauher, 2010: 8).

Riječ je načelno o relativno novom pojmu, ali i znanstvenom području, koje se započinje razvijati sredinom prošloga stoljeća, točnije oko 60-ih godina istoga. Od tada do danas ono bilježi intenzivan razvoj, a posebice dolazi do značaja od pojave suvremenoga doba, to jest od kraja prošloga stoljeća pa sve do danas. Pojavom računala treće generacije dolazi do potrebe za kompleksnijim softverima i rješenjima ove prirode, kako bi dala izravnu podršku i osigurala funkcionalnost novih računala. Jedan od takvih primjera je i multi-tasking operativni sustav.

Softversko inženjerstvo iz navedenoga razdoblja prepoznalo je kako je skup dotadašnjih neformalnih tehnika i metoda individualnog programiranja neprimjeren za realne potrebe, te dolazi do proširenja istih. Točnije, pokreću se ozbiljni razvojni projekti u koje se uključuje veliki broj uspješnih i respektabilnih programera (Maciaszek, 2007).

Od pojave softverskog inženjeringa do danas, ovo se područje intenzivno i dinamično razvijalo, a u konačnici se etabliralo kao presudan element tehnike i računarstva pa se kao takvo i razmatra te istražuje u današnjim suvremenim znanstvenim krugovima. Ono što je važno istaknuti kako je riječ o esencijalnom području u poslovnom i svakodnevnom životu, a od iznimnog značaja biva za sve sektore i djelatnosti, kao i za globalno društvo generalno.

Softversko inženjerstvo neopipljivi je input svakoga procesa u današnjici. To se potvrđuje činjenicom da je današnji način života i poslovanja osobito informatiziran i digitaliziran, a za nesmetano i napredno funkcioniranje zaslužno je upravo ovo područje. Zanimljivo je istaknuti kako je riječ o vrlo dinamičnom procesu koji nije moguće okončati. Točnije, kako se mijenjaju prakse, trendovi, izazovi i potrebe na međunarodnoj razini, tako se razvijaju i nova softverska rješenja u kontekstu ovoga područja. Prema tome, ispravno je tvrditi kako on prati razvoj međunarodne ekonomije i globalnog društva.

2.2. OSNOVNO O SOFTVERU

Softver se danas sve češće naziva softverskim produktom, a taj naziv stvara prostor za ukazivanje na značaj i ulogu softverskog inženjerstva. Riječ je o stručnom i specijaliziranom nazivu, koji se pobliže određuje kao skup računalnih programa i pripadajuće dokumentacije, koji je osmišljen i razvijen u svrhu prodaje ili komercijalizacije. Sukladno tome, on može biti razvijen, od ideje do komercijalizacije, za nekog konkretnog korisnika (engl. customized product) ili za tržište generalno (engl. generic product). Jasno je kako se ova dva proizvoda razlikuju prema konkrentim karakteristikama, ali i funkcijama. Proizvod namijenjen korisniku u većoj je mjeri personaliziran i prilagođen potrebama te željama korisnika, dok je onaj drugi općeg karaktera i šire primjene (Sommerville, 2016: 44).

Softver ili softverski sustav u današnjici prije svega mora biti kvalitetan. Kvaliteta opravdava njegovu cijenu na tržištu, ali jednako tako potvrđuje funkcionalnost i namjenu. Od njega se generalno očekuje da zadovoljava sljedeće elemente ili svojevrsne dimenzije (Magner i Mauher, 2010: 8):

- Mogućnost održavanja – misli se na mogućnost promjene i nadogradnje sukladno s potrebama korisnika;
- Pouzdanost i sigurnost – podrazumijeva se predvidljivost sustava, zaštita podataka i slično;
- Učinkovitost – odnosi se na važnost zadovoljenja performansa, kao i na štedljivost upravljanja strojnim resursima;
- Upotrebljivost – izvršavanje onoga što korisnici očekuju, uz zadovoljavajuće sučelje i dokumentaciju.

Jasno je kako postojanje opipljive tehnologije, odnosno strojnih resursa nema nikakav značaj i ulogu bez adekvatne podrške softvera ili programa. To je ujedno i najjednostavniji način definiranja uloge i značaja ovoga termina. U praksi je riječ o vrlo kompleksnom pojmu i području, koje ima niz funkcija, javlja se u raznim oblicima i koristi među različitim skupinama sudionika. Jedan od najjednostavnijih pregleda softvera je sljedeći (Slika 1.).

Slika 1. Vrste softvera

Izvor: Znanje (2018)

Ovi se softveri primijenjuju u svakodnevnom životu i poslovanju ljudi diljem svijeta. njihovo korištenje generalno ne zahtijeva posebne napore i dovoljno je imati tek neka osnovna informatička znanja i iskustvo. Oni su dostupni svima, a suvremeno poslovanje bez njih gotovo je nezamislivo.

Kako bi neki program ili softver uopće nastao, važno je provoditi softverski proces, koji podrazumijeva korištenje i niza metoda te alata. Načelno je riječ o vrlo kompleksnom znanstvenom području, koje podrazumijeva primjenu skupa ili niza aktivnosti i rezultata, čiji je cilj razvoj odnosno formiranje softvera ili programa. Osnovne aktivnosti ovoga procesa su (Slavek et al., 2012):

- Specifikacija;
- Oblikovanje;
- Implementacija;
- Verifikacija;
- Validacija;

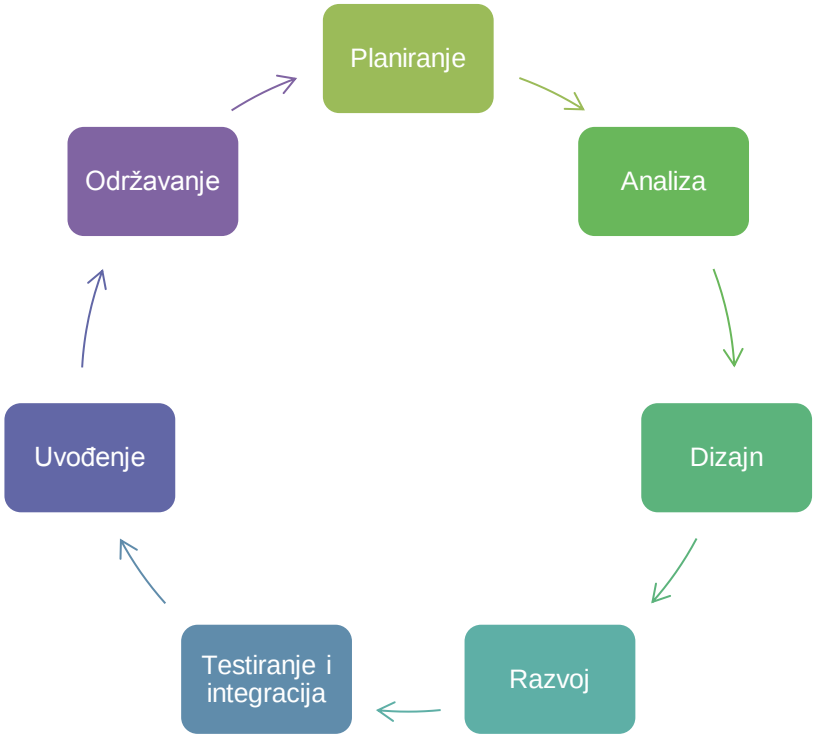
- Održavanje i evolucija.

Daje se zaključiti kako je riječ o faznom procesu. Često se ističe kako se razvoj sustava dijeli u nekoliko osnovnih faza, točnije postupke planiranja, analize, dizajna, razvoja, testiranja, uvođenja i održavanja, a u svrhu razvoja takvog programa koji će optimalno zadovoljiti potrebe krajnjih korisnika. Iz toga se daje proizvesti slikovni prikaz životnog ciklusa razvoja softvera, u svrhu razmijevanja predmetne problematike. Detaljnije o istome, kao i temeljnim fazama ovoga procesa slijedi u nastavku poglavlja.

2.3. ŽIVOTNI CIKLUS RAZVOJA SOFTVERA

O životnom ciklusu softvera dijelom je već prethodno bilo riječi. Nakon osnovnih činjenica u svezi istoga, važno je dati prikaz njegova cikličkog kretanja, te se detaljnije osvrnuti na svaku fazu zasebno (Slika 2.).

Slika 2. Životni ciklus razvoja softvera



Izvor: Prilagođeno prema: Matić et al. (2016)

Početna faza ovoga procesa odnosi se na planiranje. Ona podrazumijeva analizu domene sustava s ciljem razumijevanja onoga što se od budućeg programa očekuje.

Točnije, razvija se percepcija o funkcionalnosti i namjeni programa u budućnosti. Vrlo često se u praksi u tu svrhu koristi analiza postojećeg sustava slične namjene.

Sljedeći korak je analiza zahtjeva, a odnosi se na specificiranje obilježja sustava koja će ispuniti očekivanja, želje i potrebe korisnika. Točnije, riječ je o identificiranju elementa koji će približiti sustav korisniku.

U okviru faze dizajniranja, detaljno se opisuje budući sustav. Zapravo je vidljivo kako su ove tri faze toliko povezane da ih je moguće načelno proučavati u domeni jedne kompleksnije faze. Dizajn programa podrazumijeva generalnu arhitekturu istoga, kao i specifikaciju svake sastavne komponente.

Razvoj sustava je svakako kompleksnija faza u kojoj programeri razvijaju budući softverski sustav ili softversko rješenje. Nakon toga slijedi testiranje oformljenog sustava i njegovo integriranje, nakon čega je sustav spreman za uvođenje, kao fazu koja ujedno podrazumijeva i pripremu dokumentacije, obučavanje korisnika i razvoj znanja o sustavu. Ova faza važna je i za buduće održavanje sustava, o čemu je već bilo riječi, a što je posljednja faza ovoga procesa.

Na danom slikovnom prikazu jasno je kako je riječ o cikličnom i kontinuiranom procesu. To u praksi znači da se neki sustav može redizajnirati, dalje razvijati ili modificirati. Točnije, sustav kao takav ne mora u potpunosti doživjeti kraj, već njegov kraj može podrazumijevati revitalizaciju njegova životnog ciklusa.

Osim ovoga pristupa, razrada životnog ciklusa softvera, moguće je istaknuti i nešto prošireniji model. Konkretno, riječ je o ciklusu aktivnosti u razvoju, korištenju i održavanju softvera, koji podrazumijeva prolazak softvera kroz veći broj faza i to redom (Čubranić et al., 2013: 241):

- „Inicijalizacija sustava je aktivnost u kojoj se navodi podrijetlo softvera;
- Analiza i specificiranje zahtjeva je aktivnost u kojoj se identificiraju problemi koje je potrebno riješiti novim softverom;
- Specifikacija funkcija je aktivnost u kojoj se identificiraju i formaliziraju podaktivnosti definiranja predmeta obrade, identificiranje atributa i veza objekata i operacija;

- Strukturiranje i izbor dijelova su aktivnosti kojima se na osnovi identificiranih zahtjeva i specifikacije funkcija strukturira softver na takve dijelove kojima se može upravljati, a koji predstavljaju logičke cjeline;
- Specifikacija strukture je aktivnost u kojoj se definiraju međusobne veze između dijelova strukture i sučelje između modula sustava;
- Specifikacija detaljnih komponenti dizajna je aktivnost u kojoj se definiraju procedure putem kojih se izvori podataka svakog pojedinog modula transformiraju iz potrebnih ulaza u zahtijevane izlaze;
- Implementacija komponenti i otklanjanje nedostataka je aktivnost u kojoj se kodiraju dizajnirane procedure i procesi i pretvaraju u izvorni kod;
- Integracija i testiranje softvera je aktivnost koja potvrđuje i održava cjelokupnu integralnost komponenti softvera putem verifikacije konzistentnosti i kompletnosti uvedenih modula;
- Provjera dokumentacije i uvođenje softvera je aktivnost koja obuhvaća izradu sistemske dokumentacije i uputa za korisnika;
- Obuka i upotreba je aktivnost koja osigurava korisnicima softvera instrukcije i upute za razumijevanje mogućnosti i ograničenja u cilju uspješne upotrebe sustava;
- Održavanje softvera je aktivnost koja podržava operacije sustava u ciljnom okruženju na način da osigura potrebna unapređenja, proširenja, popravke, zamjene i drugo;
- Gašenje softvera (povlačenje iz primjene) je posljednja aktivnost u životnom ciklusu.

Vidljivo je kako je sam proces sačinjen od nekoliko ključnih faza, koje se ovisno o potrebama mogu dijeliti na nekoliko dodatnih. U tom smislu riječ je o mogućnosti specifikacije ovoga procesa, no načelno je njegov tijek u praksi uvijek isti.

Nakon razrade tijeka i specifičnosti ovoga procesa pristupa se analizi metoda koje se primijenjuju u te svrhe. O metodologiji, normizaciji i standardima detaljnije slijedi u nastavku predmetnog rada, u zasebnom poglavlju. Na taj način se predmetna poglavlja međusobno povezuju, a u svrhu boljeg razumijevanja predmetne problematike, te se daje osnova za pristupanje analizi središnje teme ovoga rada.

3. METODOLOGIJA, NORMIZACIJA I STANDARDI

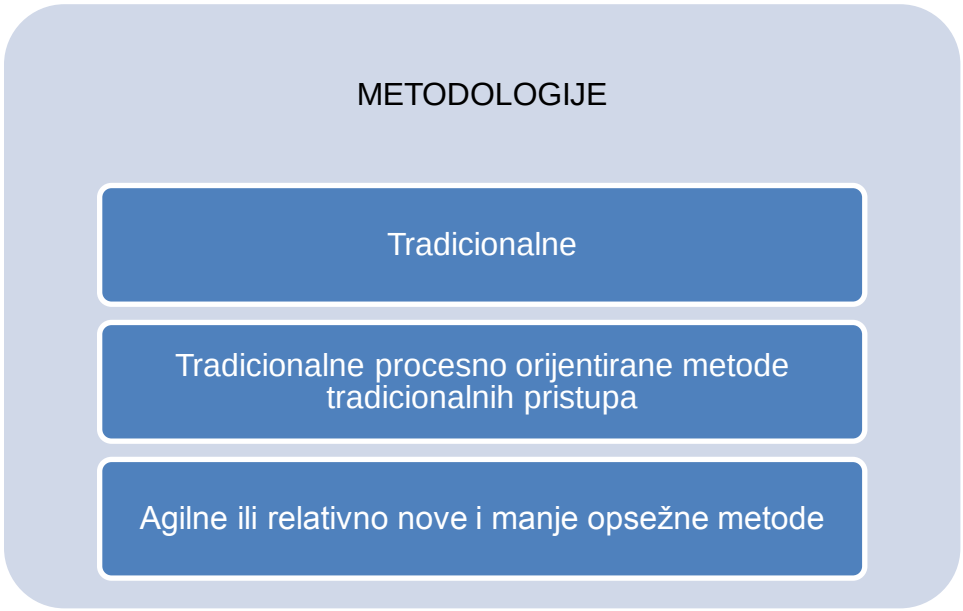
Životni ciklus softvera, kao što je i prethodno spomenuto, predstavlja proces njegova razvoja. Jednom riječju, riječ je o osmišljavanju programskog rješenja za konkretne potrebe i probleme. U te svrhe koriste se razne metode, norme i standardi ili jednostavnije rečeno konceptualni modeli razvoja softvera. U ovome poglavlju raspravlja se upravo o njima.

3.1. METODOLOGIJE

Metodologija razvoja softvera zapravo podrazumijeva pristup ili način, tijek, specifičnosti i ostalo u svezi ovoga procesa, načina osmišljavanja i komercijalizacije softvera. U današnjici su razvijene različite metode, odnosno pristupi razvoja softvera, a pri njihovu razmatranju riječ je o analizi metodologija izrade softvera.

Može se tvrditi kako su u današnjici najprimjenjivanije i najraširenije one metodologije koje se ističu kvalitetom i prilagodljivošću, a optimalno zadovoljavaju potrebe sudionika (Čubranić et al., 2013). S obzirom na njihovu brojnost, u znanosti i javnosti navode se neke od skupina ili vrsta ovih metodologija, a pri njihovoj razradi riječ je o klasifikaciji metodologija razvoja softvera (Slika 3.).

Slika 3. Metodologije razvoja softvera



Izvor: Čubranić et al. (2013) Str. 243.

Ove skupine metoda imaju niz razlika, a kao takve se u praksi i primijenjuju. Ono što je posebno važno istaknuti jest činjenica da sekvencijske metode bivaju tradicionalno orijentirane, dok agilne metode poseban fokus usmjeravaju na male jedinice posla i s naglaskom na vrijednostima i principima umjesto na procesu. To ih čini specifičnima u odnosu na prethodnu skupinu, a često se navode i kao preciznije, kvalitetnije i personaliziranije metode razvoja softvera.

Osim navedene klasifikacije, metodologije razvoja softvera mogu se općenito podijeliti u dvije glavne skupine i to (Krneta, 2018):

- Teške i opsežne metodologije razvoja softvera (engl. *heavyweight methodologies*) – sadrže mnogo pravila, načina postupanja i dokumentacije, a traže vrijeme i disciplinu za ispravno slijeđenje. Često se nazivaju se i „debelim“ metodama (engl. *thick methods*);
- Lakše i manje opsežne metodologije razvoja softvera (engl. *lightweight methodologies*) – obično sadrže tek nekoliko pravila i načina postupanja koji su lagani za slijeđenje. Nazivaju se i „tankim“metodama (engl. *thin methods*).

Neovisno o vrsti metodologije koja se primijenjuje, a o čemu odlučuje menadžment, cilj svake od njih je razvoj kvalitetnog softvera uz minimalne troškove, kao što je već i

istaknuto. Njihov značaj je izniman jer uspješno odabrana i primijenjena metodologija implicira kvalitetu projekta, odnosno softvera.

U praksi, metodologija odabire metode, prilagođava ih konačnom cilju, a također propisuje redoslijed upotrebe metoda te proces modeliranja od početka do kraja životnog ciklusa softvera. Danas postoji čitavi niz metodologija i metodika, što je rezultat intenzivnog i dinamičnog razvoja softverskog inženjeringa, o čemu se raspravljalo u prethodnom poglavlju rada. Pojedine vrste istih nisu predmet istraživanja ovoga rada pa se izostavljaju iz analize. Detaljnija rasprava posvećuje se tek sekvencijskim i agilnim metodama ili metodologijama, koje se detaljnije i opsežnije istražuju u narednom poglavlju rada, čime se on zaokružuje u smislenu i funkcionlanu cjelinu.

3.2. CASE TEHNOLOGIJE

U okviru ove problematike važno je svakako spomenuti i CASE tehnologije. Riječ je o pojedinačnim alatima koji služe automatizaciji zadataka i procesa u području razvoja softvera. Njihova svrha je formiranje istih kao funkcionalne cjeline i one načelno ne podrazumijevaju zamjenu metoda ili tehnika razvoja, već iste nadopunjuju (Crunchbase, 2018).

U praksi se one koriste na interaktivan način, što potvrđuje kako su izravno prilagođene korisnicima i to je jedno od vodećih obilježja ovih tehnologija. Korištenje grafike sljedeće je značajno obilježje. Ciljevi primjene ovih tehnologija mogu se razmatrati kroz (Milošević, 2013):

- Povećanje produktivnosti projektanata;
- Uštedu vremena izrade softvera;
- Unapređenje kvalitete i performansi softvera.

Riječ je načelno o programskim alatima za razvoj softvera, što ukazuje na njihovu iznimnu ulogu u ovome području istraživanja. Danas postoji nekoliko vrsta ovih tehnologija koje su vrlo poznate u svijetu informatike i softverskog inženjerstva, a među primjenjivanijima su redom (Milošević, 2013):

- Cor-Vision, Cortex Corporation;
- Promod PLUS, Promod INC;
- Oracle CASE, Oracle Corporation;
- Westmount I-CASE, Westmount Technology;
- Excelerator, Intersolv INC;
- CASE for Informix, Informix Softvare INC;
- AD/Cycle, IBM;
- BPWin;
- ERWIN;
- Rational rose;
- MS Wisio;
- Matlab CASE;
- Oracle designer.

U praksi se one najčešće klasificiraju s obzirom na fazu životnog ciklusa koju pokrivaju, kao i s obzirom na njihove funkcije. Pregled ovih vrsta to jest klasifikacija daje se u nastavku (Tablica 1.).

Tablica 1. CASE tehnologije

Klasifikacija s obzirom na pokrivenost faze životnog ciklusa razvoja softvera	Klasifikacija s obzirom na funkcije
• *Upper CASE – planiranje i upravljanje projektima;* • *Middle CASE – analiza i projektiranje;* • *Lower CASE – programiranje, testiranje i uvođenje;* • *CASE tool – namijenjeni pojedinim aktivnostima;* • *CASE toolkit – namijenjeni pojedinim fazama ili aktivnostima u više faza;* • *CASE workbench – integrirana kolekcija CASE paketa kojom se pokrivaju sve faze.*	• *Alati za planiranje poslovnih sistema – prate informacijske tokove između OJ;* • *Alati za upravljanje projektima – prate glavne upravljačke aktivnosti, npr. planiranje, procena vrednosti, resurse, rizik, troškove, kvalitetu, standarde;* • *Alati podrške – dokumentiranje, podrška sistemskom softveru,upravljanje bazama podataka;* • *Alati za analizu i dizajn – najvažniji alati, omogućavaju kreiranje sistema;* • *Alati za programiranje – podržavaju kreiranje programskog koda;* • *Alati integracije i testiranja – prikupljanje testnih podataka, analiza izvornog koda i pomoć u aktivnostima testiranja;* • *Alati prototipskog razvoja – služe za izradu prototipa;* • *Alati za podršku održavanju – koriste se za reverzibilni inženjering, rekonstrukciju koda i reinženjering*

Izvor: Prilagođeno prema: Milošević (2013)

Evidentno je kako je riječ o vrlo značajnom segmentu u području softverskog inženjeringa i procesu razvoja softvera. S obzirom na klasifikacije, u praksi je pojednostavljeno njihovo korištenje.

3.3. NORMIZACIJA I STANDARDI

U okviru ovoga poglavlja, važno je još osvrnuti se na normizaciju i standarde. Dijelom je već bilo riječi o njima, no u nastavku teksta oni se pobliže analiziraju. Istaknuto je kako, prema osnovnim načelima softverskog inženjeringa, korištenje metoda, tehnika i tehnologija treba biti prilagođeno i usklađeno u najvećoj mogućoj mjeri. Tome upravo služe norme i standradi.

Naime, kako bi se postigla maksimalna povezanost između korištenih metoda, tehnika i tehnologija u procesu razvoja softvera ili novog proizvoda, potrebno je provesti standardizaciju u kanalima komunikacije između ovih resursa ili elemenata. Prema tome, ispravno je tvrditi kako norme i standardi predstavljaju alat ili instrument za definiranje javnih protokola komunikacije između metoda, tehnika i tehnologija, kao vodećih elemenata i resursa ovoga procesa

Danas postoje brojne institucije koje se bave propisivanjem i/ili održavanjem standarda u razvoju softvera, a među njima svakako treba spomenuti i izdvojiti sljedeće (Čubranić et al., 2013):

- SEI (engl. *Software Engineering Institute*);
- DoD (engl. *US Department of Defense*);
- ANSI (engl. *American National Standards Institute*);
- IEEE (engl. *Institute of Electrical & Electronic Engineers*);
- ESA (engl. *European Space Agency*);
- ISO (engl. *International Organization for Standardization*);
- IEC (engl. *Commission Electrotechnique Enternationale - International Electrotechnical Commission*).

Kao što je i istaknuto, riječ je visoko specijaliziranim agencijama, odnosno institucijama koje imaju vodeću ulogu u ovome području, ali i šire. Osim što se bave predmetnim

aktivnostima, pridaju im se i mnoge druge funkcije, a svakako treba istaknuti praćenje međunarodnih trendova i praksi, informiranje, educiranje i unapređenje suradnje u ovome području.

Osnovni izvor normi u razvoju softvera su u početku bile ANSI i IEEE norme. Nakon izvjesnog vremena njih preuzima ISO u suradnji s IEC-om pa se ističe kako se ove norme razmatraju u današnjici kao rezultat suradnje i integriranog rada združene tehničke komisije - JTC (engl. *Joint Technical Committee*).

Važno je istaknuti kako je izgradnja ili projektiranje softvera zapravo završna faza u proizvodnom ciklusu, o tome je bilo riječi u prethodnom poglavlju pri analizi značenja ovoga procesa. S obzirom da se u praksi ovaj proces kontinuirano mijenja i razvija, važno je uspostaviti automatizaciju njegova provođenja. To zapravo znači da se svi postupci ove prirode i namjene povežu u jednostavnu, jedinstvenu i sustavno prilagodljivu proceduru. Aktivnosti koje su obuhvaćene tim pojmom su (Čubranić et al., 2013):

- Preuzimanje koda iz repozitorija;
- Prevođenje;
- Izvođenje testnih skripti;
- Prilagodba izvršnih elemenata za različite okoline i platforme;
- Implementacija u radnu-produkcijsku okolinu.

Vidljivo je kako je riječ o kompleksnoj proceduri koja je sastavljena on niza aktivnosti koje je važno povezati i ukomponirati serijski ili paralelno. Automatizacija postupaka vrlo je značajna jer se na taj način unapređuje i povećava brzina izgradnje, kao i sama kvaliteta softvera. Točnije, na ovaj način podržavaju se osnovna načela, vrijednosti i funkcije softverskog inženjeringa, a o kojima je već bilo riječi.

4.SEKVENCIJSKE I AGILNE METODE

O sekvencijskim i agilnim metodama već se raspravljalo u prethodnom dijelu poglavlja, točnije u području analize metodologija predmetnog procesa. Jasno je kako je riječ o jednom od esencijalnih elemenata predmetnog procesa. Ove se dvije skupine metoda razlikuju po mnogočemu, a osnovno je istaknuti kako se sekvencijske metode oslanjaju na tradicionalne pristupe, dok one agilne mogu biti razmatrane kao suvremene i specijalizirane metode. O čemu je konkretno riječ biti će jasnije u ovome poglavlju rada.

4.1. O SEKVENCIJSKIM METODAMA

Sekvencijske metode, kao što je i istaknuto oslanjaju se na tradicionalne metode razvoja softvera. One se javljaju 70-ih godina prošloga stoljeća, a osnovno obilježje je ono koje ističe da se one sastoje od faza koje opisuju pojedine faze životnog ciklusa softvera, odnosno njegova razvoja. Prema tome, sve faze su jasno određene, a svaka se provodi i završava prema jasno utvrđenim aktivnostima i postupcima.

Ove metode imaju za cilj i za obvezu jasno dokumentirati svaku fazu ovoga procesa. U praksi se nazivaju i klasičnim metodama, a sadrže nekoliko modela upravljanja životnim ciklusom softvera. Neke od njih su (Bileta, 2016):

- Model vodopada;
- V-model;
- Prototipiranje;
- Inkrementalni razvoj;
- RAD model
- Evolucijski model.

Nakon razrade vrsta ovih modela, važno je generalno se osvrnuti na prednosti i nedostatke istih. To ujedno biva i osnova za razlikovanje ove metodologije od agilne. Prikaz navedenoga slijedi u nastavku (Tablica 2.).

Tablica 2. Prednosti i nedostatci sekvencijskih metoda

PREDNOSTI	NEDOSTATCI
• Korisnici sustava su uglavnom aktivno uključeni u razvoj; • Konačni sustav je jednostavniji za korištenje i održavanje; • Korisnički zahtjevi su bolje provedeni; • Konačni sustav ima manje nepotrebnih mogućnosti; • Rana identifikacija problema; • Veća kvaliteta dizajna.	• Performanse konačnog sustava; • Konačni sustav ima manje mogućnosti; • Konačni sustav je kompleksan za održavanje; • Manja kvaliteta dizajna; • Zahtjeva iskusne stručnjake; • Slaba fleksibilnost; • Ograničene mogućnosti.

Izvor: Prilagođeno prema: Galinac Grbac (2018)

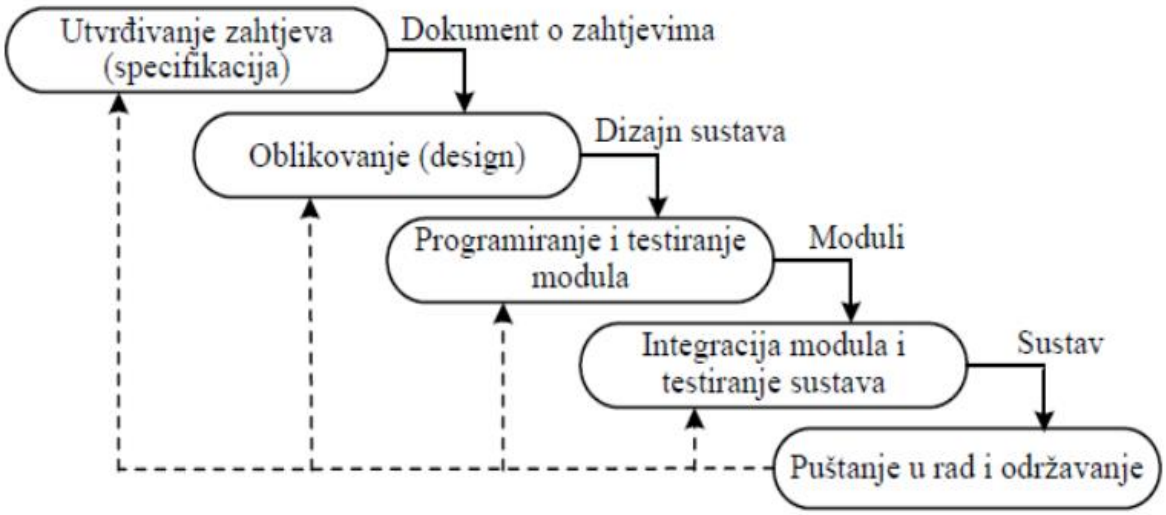
Detaljnije o odabranim modelima, ove skupine metodologija slijedi u nastavku poglavlja. Konkretnije, obrađuju se model vodopada i V-model. Pri tome se daje njihov prikaz, uz objašnjenje i specificiranje obilježja, namjene i specifičnosti.

4.2. VRSTE SEKVENCIJSKIH METODA

4.2.1. Model vodopada

Prvi na danom popisu je model vodopada. Riječ je o jednom od najpopularnijih modela ili metoda ove prirode, a na njegovoj osnovi razvijaju se mnogi drugi, poput V-modela. Prikaz istoga slijedi u nastavku (Slika 4.).

Slika 4. Model vodopada



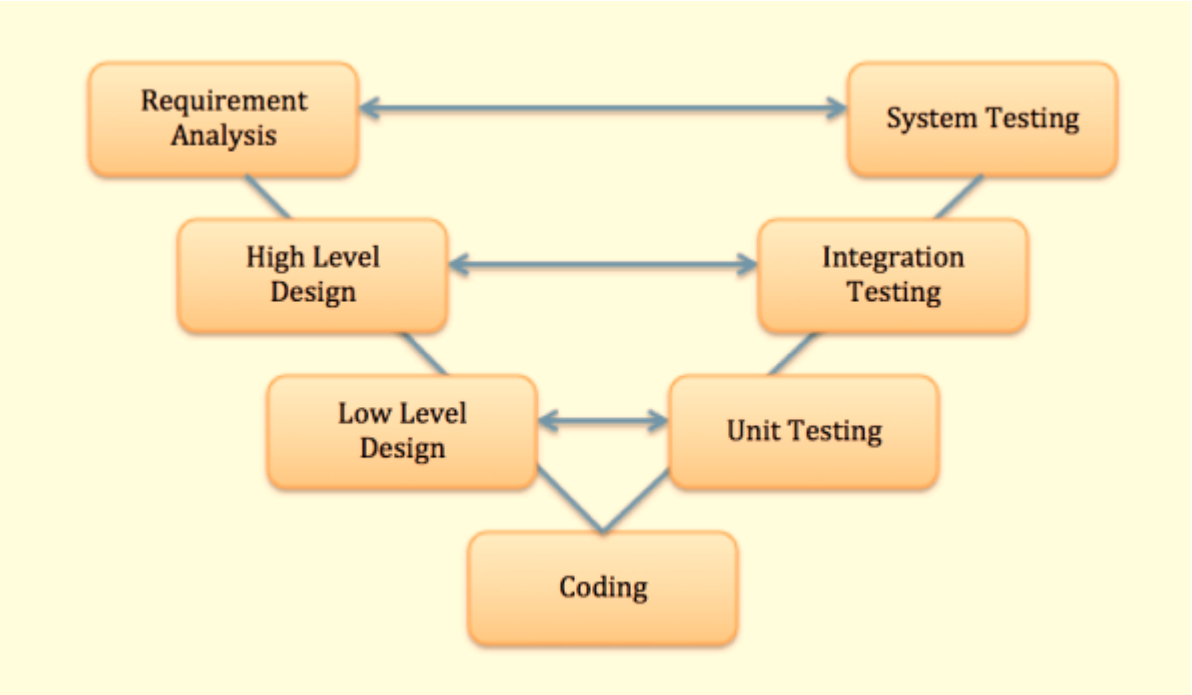
Izvor: Bileta (2016) Str. 4.

Tijek ovoga modela, koji podsjeća na vodopad, osnova je za definiranje njegova naziva. Vidljivo je kako se on sastoji od 5 osnovnih faza koje su međusobno povezane i uvjetovane. Pri tome, svaka faza započinje tek po okončanju one prethodne. Ovaj model iznimno je koristan jer omogućuje precizno planiranje i praćenje samoga procesa, odnosno njegova tijeka u praksi.

4.2.2. V-model

Sljedeći na popisu je V-model, koji se zasniva na prethodno analiziranom. Riječ je o nešto proširenijoj verziji modela vodopada, odnosno njegovoj ekstenziji. Osnovna razlika je ta što se na prethodnom modelu tijek aktivnosti konstantno provodi prema dolje, dok na ovome primjeru nakon implementacije slijedi reverzija prema gore. Na taj način omogućeno je vraćanje iz procesa koji dolaze kasnije u procese koji su prethodno izvršeni (Slika 5.).

Slika 5. V-model



Izvor: Guru (2018)

Treba istaknuti kako se ova metoda provodi na jednostavniji način, što je već i potvrđeno. Ovaj model nudi više mogućnosti, što je vidljivo iz danog prikaza. On pokazuje kako se sve aktivnosti provode s vrha prema dolje, a vrlo je jednostavan za korištenje jer svaka faza ima specifične rezultate. Planovi se u ovome modelu mogu unaprijed razvijati, to jest omogućena je verifikacija i validacija proizvoda u ranijim fazama.

4.2.3. Prototipiranje

Prototip je brzo i jednostavno razvijeni program, koji obnaša budući sustav. Pri tome, on se lako mijenja i nadograđuje, a omogućuje isprobavanje raznih ideja. Često se ističe kako prototipiranje nije istraživačko programiranje. To se potvrđuje činjenicom da je osnovni cilj ove metode zapravo otkrivanje i validacija zahtjeva, nakon čega se sustav oblikuje standardnim ili konvencionalnim načinima.

Ova metoda u naravi je metoda koja odgovara potrebi za bržim, jednostavnijim i jeftinijim pristupima razvoja sustava. Danas postoji nekoliko prototipskih metoda, koje

odgovaraju ovim zahtjevima, a sukladne su rapidnom razvoju tehnologije i poslovanja generalno. U današnjici, novije metode razvoja prototipa aktivno uključuju sve sudionike, a cilj je pri tome razviti efikasan, stabilan i isplativ informacijski sustav, koji će biti prikladan poslovnim izazovima koje korisnik ima pred sobom.

Modeli s prototipskim pristupom postepeno su zamijenili stariji, tradicionalni vodopadni model. Tehnike ovih metoda su razvoj u dinamičkim jezicima visoke razine, programiranje za baze podataka i pozivanje gotovih dijelova ili aplikacija (Štimac, 2015).

4.2.4. Inkrementalni razvoj

Osnovni korijeni na kojima nastaje ovaj model odnose se na težnju da se korisniku isporučuje sustav u manjim cjelinama koje mogu samostalno raditi. Važno je pri tome reći kako se opći proces odvija u iteracijama, a na način da svaka iteracija procesa isporuči skup samostalnih cjelina krajnjem korisniku (Galinac, 2018).

Treba napomenuti kako se sustav inkrementalno izgrađuje, a svaki inkrement pri tome predstavlja samostalnu radnu cjelinu sustava. Kod inkrementalnog razvoja prisutna je zapravo kombinacija evolucijskog prototipiranja s kontrolom, a koju zahtijeva veliki projekt.

Sam razvoj informacijskog sustava provodi se inkrementalno, odnosno po dijelovima, kako je već i istaknuto. Primjenom svakog od tih dijelova nastoji se zapravo izbjeći konstantna promjena, koja obilježava evolucijski prototip. Arhitektura ili okvir sustava utvrđuju se vrlo rano, u okviru istraživanja i razvoja, a služi kao osnova daljnjeg razvoja ili procesa (Galinac, 2018).

Kod ove metode, inkrementi ili dijelovi se, po razvoju, dostavljaju korisniku i potvrđuju. Postoji mogućnost promjene njihova okvira razvoja, no one nastupaju tek kada se otkriju pogreške.

Vrlo su važne povratne informacije klijenata ili korisnika o isporučenim komponentama, a na osnovu njih utječe se na budući dizajn i razvoj istih. Na temelju navedenoga može

se zaključiti kako je inkrementalni razvoj lakši za upravljanje u odnosu na evolucijski, a njegov plan i dokumentacija imaju za obvezu specificirati svaki korak jer se time dobiva mogućnost ranog dobivanja povratne informacije u svezi grešaka i problema (Galinac, 2018).

Osnovni problemi na ovome primjeru metode je sama arhitektura sustava, koja mora biti uspostavljena prije nego su zahtjevi sustava potpuni. To utječe na ograničenost naknadnih promjena.

4.2.5. Evolucijski model

Ovaj model temelji se na ideji razvoja ili evolucije početne izvedbe sustava, koji se naposlijetku isporučuje korisniku, a on ga ocjenjuje i komentira. Od tuda i sam naziv modela. Mišljenja korisnika služe unapređenju sustava, točnije njegovu usavršavanju sve do razine kada dobije pozitivne ocjene korisnika. Tek tada riječ je o krajnjem proizvodu (Galinac, 2018).

Treba istaknuti kako ovaj pristup zapravo dinamički reagira na dinamične promjene, kao i nepredvidive izazove. Zbog toga se može istaknuti kako je riječ o realnom pristupu razvoja sustava, kod kojeg je teško identificirati detaljne specifikacije. Uspjeh ove metode očituje se u korištenju onih tehnika koje osiguravaju brze iteracije sustava.

U slučaju predloženih izmjena od strane korisnika, konkretne promjene moraju se brzo inkomponirati i demonstrirati, a to se postiže korištenjem programskih jezika visoke razine koji se koriste za razvoj sustava i aplikacija, posebnim razvojnim okruženjima te integriranim softverskim alatima (AES, 2010).

One organizacije ili korisnici koji planiraju koristiti ovu metodu za razvoj sustava moraju imati na umu neke konkretne činjenice u svezi njega. Pored navedenoga, misli se na životni vijek sustava koji će biti vrlo kratak, kao i strukturu sustava koja će u budućnosti postati preuska i vrlo teška za nadogradnju. Ova obilježja uzimaju se ujedno i kao vodeći nedostatci predmetne metode.

4.3. O AGILNIM METODAMA

O sekvencijskim metodologijama moguće je raspravljati na opsežan način, a kada je riječ o agilnim metodologijama, tada je rasprava još kompleksnija. Agilne metodologije načelno predstavljaju kontrast ili suprotnost klasičnim metodologijama. Već je istaknuto kako je riječ o suvremenim metodologijama specificiranog i prilagođenog karaktera. To zapravo znači da se one javljaju kao odgovor na klasične metodologije, a njihova osnovna svrha je otkloniti nedostatke prethodnih i ponuditi kvalitetnija rješenja i postupke ovoga procesa.

Iako se javljaju nakon klasičnih metodologija, agilne metodologije imaju svoju povijest razvoja. Njihovi začetnici su Ken Beck i Alistair Cockburn, koji su pristupili, uz ostale sudionike, promociji ovih metodologija na prijelomu stoljeća. Oni su pobliže odredili principe ovih metodologija, koji ujedno dočaravaju njihove koristi ili prednosti. Konkretno, riječ je o (Bileta, 2016):

- Zadovoljstvu klijenata;
- Dobrodošlim zahtjevima za promjenama u svako vrijeme;
- Naglasku na iteracijama;
- Suradnji pojektanata koji su inicirali projekt;
- Motivaciji svih sudionika projekta;
- Naglasku na izravnoj komunikaciji;
- Orijentaciji prema stvaranju kvalitetnog programa koji učinkovito i efikasno radi;
- Kontinuiranom razvoju programa;
- Jednostavnosti;
- Timovima koji se kontinuirano razvijaju;
- Periodičkom ispitivanju postupaka.

Ovi principi osnova su agilnih metoda. Iako danas, kao i na prethodnoj skupini, postoji nekoliko vrsta istih, sve one poštivaju navedene principe i bivaju determinirane upravo njima. U nastavku slijedi pregled odabranih agilnih metoda, kao i na prethodnom primjeru.

Iz provedenog istraživanja daje se ukazati na niz prednosti agilnih metoda, u odnosu na klasične ili tradicionalne. Međutim, važno je prepoznati i neke nedostatke. Njihov pregled slijedi u nastavku (Tablica 3.).

Tablica 3. Prednosti i nedostatci agilnih metoda

PREDNOSTI	NEDOSTATCI
• Promjena je prihvatljiva – s kraćim ciklusom planiranja, lako je prilagoditi i prihvatiti promjene u bilo kojem trenutku izvedbe projekta; • Krajnji cilj može biti nepoznat: Agilnost je veoma korisna za projekte gdje krajnji cilj nije jasno definiran; • Brze i kvalitetne isporuke; • Snažna timska interakcija i suradnja; • Klijenti su saslušani – klijenti imaju poseban značaj i utjecaj na sami proces; • Kontinuirana poboljšanja.	• Planiranje može biti manje konkretno; • Tim mora biti upućen; • Vrijeme posvećenosti programera – proporcionalni odnos; • Dokumentacija se može zanemariti; • Finalni proizvod može biti veoma različit.

Izvor: Milošević (2016)

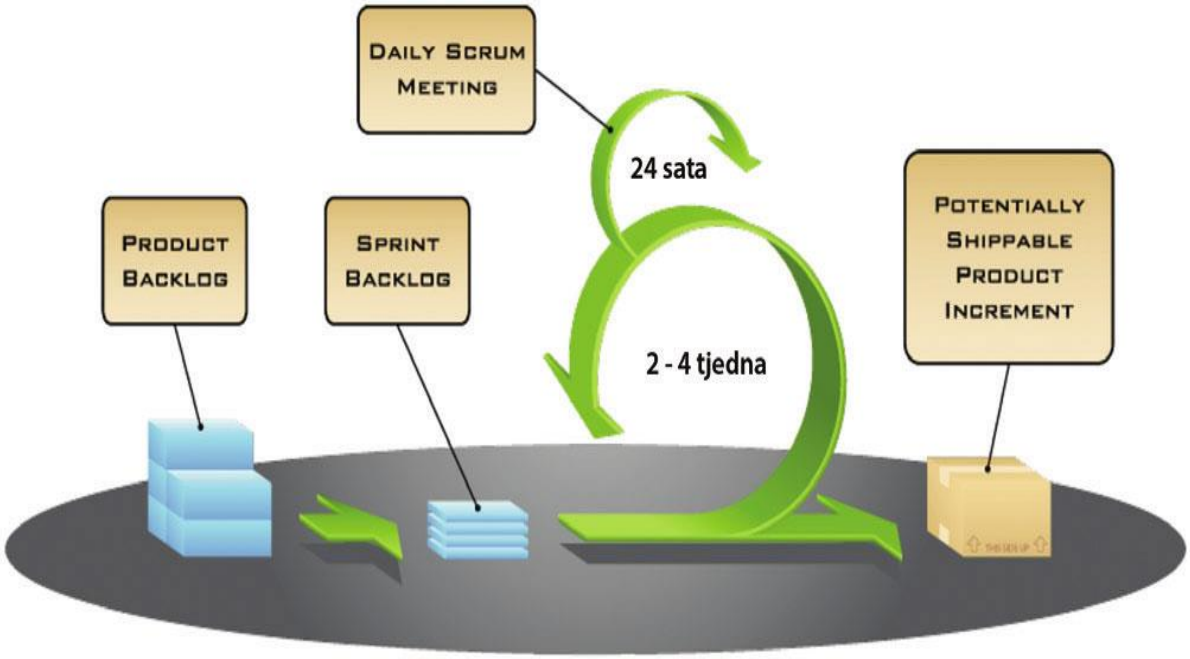
Sasvim je jasno kako sve metode, tehnike i ostali elementi u informatici i šire imaju niz prednosti i nedostataka. Iste je važno poznavati i procijeniti prije samog odabira metode koja se namjerava koristiti. Odluka pri tome treba biti utemeljena na objektivnoj procjeni.

4.4. VRSTE AGILNIH METODA

4.4.1. Scrum

Među brojnim metodama ove prirode, u praksi je najzastupljenija Scrum. Prikaz ovoga modela slijedi u nastavku (Slika 6.).

Slika 6. Scrum model



Izvor: Keba (2013)

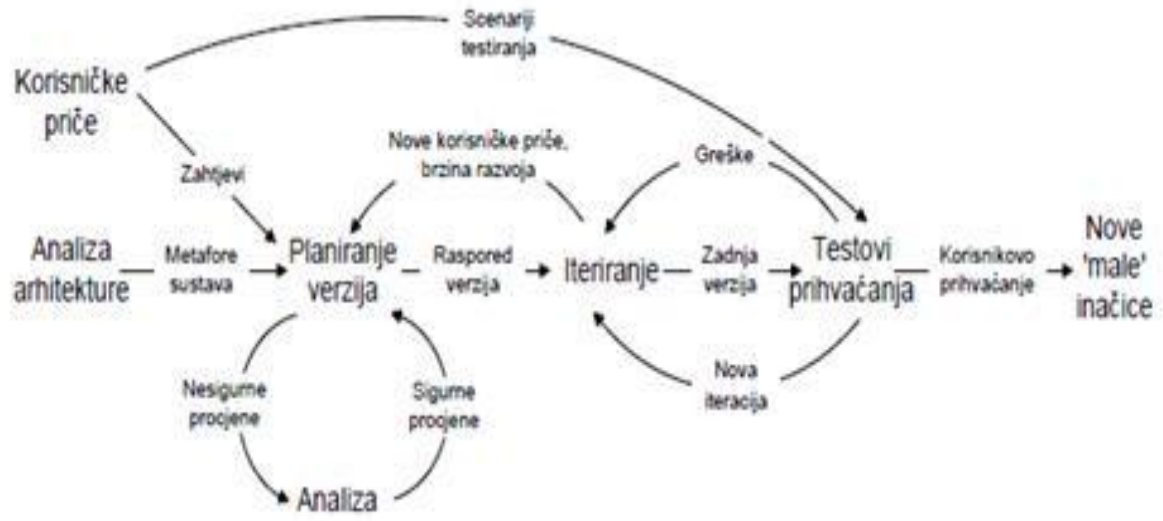
Dani prikaz može se s jednostavnošću tumačiti. Naime, prema navedenome, manji razvojni tim, koji obilježava postojanje opsežnih znanja i vještina za analizu, kodiranje i test proizvoda, u vrlo kratkim iteracijama (sprint do 4 tjedna) razvijati će dio po dio. Svrha jest da se svaki sljedeći inkrementalno nadograđuje na onaj iz prethodne iteracije (Keba, 2013). Količinu posla koju je moguće obaviti određuju upravo članovi tog time,koji su odgovorni za čitavi proces. Kratki dnevni sastanci (Daily Standup) podrazumijevaju praćenje ostvarenog plana za neki sprint. Nakon okončanja svake iteracije, moguće je pratiti konkretan tijek i uspjeh ovoga procesa.

Sukladno navedenome, napredak ili progres mjerljiv je unaprijed definiranim kodom, a konačna odluka o spremnosti zasniva se na zadovoljenju testa i postojanju dokumentacije. Važno je istaknuti kako je ovaj sustav vrlo fleksibilan na promjene zahtjeva kupaca jer u svakoj iteraciji tim radi na zahtjevima najvišeg prioriteta u danom trenutku. Osim toga, posebno je važno istaknuti ulogu kupca, budućeg korisnika. On stalno i transparentno prati razvoj te daje timu povratnu informaciju na temelju demonstriranog dijela. Riječ je o uspješnoj suradnji i integraciji ovih dionika.

4.4.2. Ekstremno programiranje

Osim ove metode, vrlo je poznato i ekstremno programiranje (XP). Riječ je o metodi ili modelu koji je predstavljen 1999. godine od strane Kenta Becka, Warda Cunninghama i Rona Jeffriesa. Kao i prethodni primjer, orijentiran je posebno na male timove, a opisuje se kao mehanizam za socijalne promjene (Tadić, 2005). Riječ je o svojevrsnom stilu razvoja, odnosno pokušaju da se kvalitetno ukomponiraju humanost, produktivnost i disciplina u ovaj proces. Njegov prikaz slijedi (Slika 7.).

Slika 7. Ekstremno programiranje



Izvor: Novitas (2018)

Ovaj model omogućuje proizvodnju dugotrajnog softvera i sposobnost reagiranja na iznenadne trendove, promjene i zahtjeve. Vrlo je fleksibilan, a u tome se očituje

njegova vodeća prednost. On se načelno zasniva na vrijednostima poput jednostavnosti, komunikacije, hrabrosti, poštovanja i povratnih informacija.

Način funkcioniranja može se opisati na vrlo sličan način kao i kod prethodnog primjera. Naime, tim je povezan s obzirom na namjeru da se stvori takav program koji će biti prilagodljiv svakoj situaciji. Ovi projekti stvaraju dnevno realne ekonomske vrijenosti klijentima, a s obzirom na to moguće je njima efikasno upravljati, prema zahtjevima i željama korisnika.

Ovaj model orijentiran je na nekoliko ključnih segmenata, a misli se na one aspekte koji programerima osiguravaju slobodu u prilagodbi formalne validacije, konfiguraciji menadžmenta, vremenskim dimenzijama i standardima kvalitete (Tadić, 2005). Generalno se timovi sastoje od 2 do 12 članova, supervizora i jednog ili više klijenata. On se zasniva na bezuvjetnoj suradnji ovih subjekata, kao što je primjer i kod ostalih metoda ove prirode ili skupine. Ponovno se naglašava aktivna uloga klijenta ili korisnika, na jednaki način kao što je već i opisano. U tome se ujedno i dočarava specifičnost ovih metoda, kao i personalizirani pristup razvoja proizvoda.

4.4.3. Dinamična metoda razvoja sustava

Dinamična metoda razvoja sustava (engl. *dynamic system development method*) razvijena je u Velikoj Britaniji polovicom posljednjeg desetljeća prošloga stoljeća. U znanosti i praksi ona predstavlja spoj i nastavak razvoja aplikacija te iterativnih metoda (Šimunović, 2017).

Može se reći da se ona odlikuje s mnogo infrastrukture koja je karakteristična za zrele tradicionalne metode, a slijedi principe agilnih metoda. Poseban fokus postavlja na fiksiranje vremena i resursa.

Faze ove metode obrađuju se na sljedeći način (Šimunović, 2017):

- Studija izvedivosti – faza koja dovodi do odluke da li će se koristiti ova metoda. Ista se provodi na osnovu ispitivanja tipa projekta te organizacijskih i ljudskih problema;

- Poslovna izvedivost – predlažu se organizacijske radionice s ciljem olakšavanja razumijevanja poslovne domene projekta, a rezultata iste je izrada definicija arhitekture sustava i plan nacrta prototipa;
- Iteracija funkcionalnog modela – analiza, pisanje koda i razvoj prototipa;
- Iteracija razvoja i dizajna – izgradnja većinskog dijela sustava te ispitivanje dizajna i funkcionalnosti od strane kupaca;
- Implementacija – konačna faza isporuke sustava krajnjim korisnicima.

Treba istaknuti kako načelno projekt ne mora biti razvijen i završen u okviru jednoga ciklusa, već može biti vraćen u bilo koju od navedenih faza. Velika pažnja se kod ove metode poklanja visokoj razini kvalitete i prilagodljivosti različitim zahtjevima, a iteracije kao ciklusi traju od 2 do 6 tjedana (Šimunović, 2017). Koristi se za razvoj softvera i rješenja koja nisu striktno IT. Bavi se zajedničkim neuspjesima ovih projekata, a neki od primjera su prekoračenje budžeta, probijanje rokova, nedostatak integracije sudionika i slično. Osam načela čine njegov temelj, a misli se na fokus na poslovne potrebe, dostavljanje na vrijeme, suradnju, kvalitetu bez kompromisa, gradnju od čvrstih temelja, razvijanje korak po korak, komuniciranje neprekidno i jasno i demonstriranje kontrole nad razvojem.

4.4.4. Razvoj baziran na karakteristikama

Razvoj baziran na karakteristikama (engl. *Feature-driven development* – FDD) predstavlja ponavljajući proces razvoja softvera koji se uspješno uklapa u industriju najboljih praksi u jednom pristupu. Zasniva se na pet osnovnih aktivnosti, a to su razvoj sveobuhvatnog modela, izgradnja liste karakteristika ili tehničkih zahtjeva, plan, dizajn po karakteristikama i izgradnja softvera po karakteristikama (Milošević, 2016).

Ova metoda javlja se također 90-ih godina prošloga stoljeća, a primarno je korištena za velike projekte u bankarskom sektoru. Faze koje podrazumijevaju korištenje ove metode u praksi su redom (Milošević, 2016):

- Razvoj općenitog modela – provodi je stručnjak za domenu, a prezentira opseg i kontekstu sustava svim članovima tima te glavnom arhitektu;

- Razvoj liste funkcionalnosti – odnosi se na razvoj kategorizirane liste funkcionalnosti na temelju zahtjeva;
- Planiranje prema funkcionalnosti – razvojni tim podliježe organiziranju setova funkcionalnosti sukladno prioritetu i ovisnostima, a kasnije ih dodjeljuje glavnim programerima;
- Dizajn prema funkcionalnosti i razvoj prema funkcionalnosti – odabir funkcionalnosti iz čitavog seta, te odabir timova na temelju vlasnika klasa, nakon čega slijedi iteraktivni proces razvoja dijagrama niza za odabrane funkcionalnosti.

Treba istaknuti da se odabrani dijagrami naposlijetku dostavljaju programerima, koji implementiraju potrebit kod za svaku funkcionalnost. U praksi može postojati nekoliko timova za iste, a oni istovremeno dizajniraju i razvijaju setove koji su im dodijeljeni. Nakon toga slijedi testiranje i pregled.

4.4.5. Adaptivni razvoj sistema

Adaptivni razvoj sistema (engl. *Adaptive system development* – ASD) zasniva se na ideji da projekti uvijek trebaju biti u stanju kontinuiranog prilagođavanja promjenama. Zasniva se na ciklusu od tri serije koje se ponavljaju, a to su (Milošević, 2016):

- Špekuliranje;
- Suradnja;
- Učenje.

Kod ove metode karakteristično je da istraživanje i razmišljanje zamjenjuju planiranje i čine idejni plan. Poseban značaj pridaje se suradnji, a ona podrazumijeva ujedinjenje upravljanja i izvođenja projekta. Također, učenje kao element biva temeljna sastavnica ove metode. Ono se odnosi u praksi na ispitivanje pretpostavki, analizu procesa, zahtjeva i proizvoda.

Svojstva ove metode su (AES, 2010):

- Orijentiranost zadatku – svaka iteracija mora biti sukladna sa zadatkom;

- Razvoj komponenti - u svakom ciklusu razvija se barem jedna komponenta koja radi;
- Iterativnost;
- Vremensko ograničenje – svaka iteracija vremenski je ograničena;
- Tolerancija na promjene – nužnost prilagodbe promjenama;
- Rizik kao vodstvo – rizičniji djelovi rade se na početku projekta.

Tijek izvedbe ove metode može se opisati kroz nekoliko faza. Pri tome se misli na pokretanje projekta, adaptivno i cikličko planiranje, konkurentni razvoj komponenti, pregled kvalitete i završni pregled te isporuku. Pri tome, prve dvije faze obilježene su istraživanjem i promišljanjem, dok konkurentni razvoj biva zasnovan na suradnji. Posljednje dvije faze zasnivaju se na učenju.

4.4.6. Lean razvoj softvera

Lean razvoj softvera (engl. *Lean Software Development* – LSD) je metoda kod koje je moguće identificirati sedam načela ili principa, a to su eliminacija svega što je nepotrebno, pojačano učenje, odlučiti što je kasnije moguće, dostaviti što je brže moguće, osnažiti tim, graditi integritet i sagledati cjelinu (Šimunović, 2017).

Ova metoda svoje korijene vuče iz sustava proizvodnje kompanije Toyota, a koji je doprinijeo povećanju kvalitete i učinkovitosti. Osnova iste je unapređenje i samog proizvoda. (Kovaček,2016)

Specifičnost ove metode očituje se u njezinim načelima, a misli se na (Kovaček, 2016):

- Unapređenje dizajna;
- Male verzije;
- Pridržavanje standarda kodiranja;
- Kolektivno vlasništvo koda;
- Jednostavan dizajn;
- Održivi korak.

Treba istaknuti kako sva ova načela u praksi čine cjelinu te ih je kao takve potrebno spoznati i primjenjivati. Osim toga, treba spomenuti i neke aktivnosti koje čine osnovna ograničenja ove metode, a to su kašnjenje u izradi softvera, nejasni zahtjevi, nedovoljno testiranje, preopterećenje birokracijom, spora interna komunikacija. Kako bi se one izbjegle, važno je na vrijeme identificirati predmetne rizike te implementirati alat za njihovo reduciranje, izbjegavanje i otklanjanje. Greške se sprječavaju testiranjem. Testiranje je najbolje provoditi odmah nakon pisanja koda.

Na primjeru ove metode, poizvod se kupcima pokazuje i izdaje tek kada je zasnovan na čvrstim temeljima, a ne na predviđanjima. Kod onih složenijih softvera ugrađuje se više mogućnosti koje služe promjenama, a osnovna težnja usmjerena je prema efikanosti i brzini isporuke klijentima.

4.4.7. Cristal clear

Cristal Clear je dio porodice metodologija koja se koristi s cjelinom od šest do osam programera i koncentrira se prvenstveno na ljude, a ne procese i objekte. Ona zahtijeva čestu isporuku korisnih kodova, reflektiranje poboljšanja i osmotsku komunikaciju.

Ove metode razvio je Alistar Cockburn 1991. godine, a pod uvjerenjem kako su to metode koje se mogu implementirati u svim situacijama. Među njima je najpoznatija i u praksi najprimjenjivanija upravo Cristal clear metoda, koja se temelji na sljedećim načelima (Kovaček, 2016: 6):

- „Učestala isporuka–kroz određeni period, korisnike se upoznaje s početnim verzijama produkta, dobiva se povratna informacija korisnika;
- Kontinuirane povratne informacije–projektni tim raspravlja o aktivnostima, provodi se validacija dosadašnjeg projekta te se raspravlja o mogućim greškama i problemima
- Stalna komunikacija–veliki timovi su lokacijski povezani, mali timovi su svi smješteni u jednu sobu;
- Sigurnost–svi članovi tima komuniciraju, rade bez represije, svi projekti nisu kritično isti;

- Fokus–članovi tima se moraju upoznati s prioritetnim ciljevima, treba im dati mogućnost da ih samostalno rješavaju;
- Raspoloživost korisnika–članovima tima se omogućuje konstantna komunikacija s klijentima;
- Automatski testovi i integracija–konstanto verifikacija proizvoda radi manje mogućnosti grešaka u daljnjem radu.

Treba istaknuti kako je za uspješnu provedbu projekta presudno uspješno komuniciranje, a posebna pažnja pridaje se i eliminaciji birokracije, testiranju i brzini izvedbe.

4.4.8. Ostale agilne metode

U praksi postoje i mnoge druge metode ili modeli koji se koriste u ove svrhe. Moguće je, iz analiziranog, zaključiti kako se sve one uglavnom zasnivaju na jednakim načelima i principima, uz minimalne oscilacije. U kontekstu agilne metodologije, važno je spomenuti kako postoje i ostale metodologije za implementiranje agilnosti. Pri tome je moguće govoriti o (Milošević, 2016):

- Agilnom Modeliranju (AM) koje se koristi za modeliranje i dokumentaciju softverskih sustava, a predstavlja dodatak ili ekstenziju drugim agilnim metodologijama kao što su Scrum, Extreme Programming (XP) i Rational Unified Process (RUP). On može poboljšati modele sa kodom, ali ne uključuje programske aktivnosti;
- *Rational Unified Process*predstavlja prilagodljiv framework za razvijanje softvera. RUP-je poput online mentora koji pruža smjernice i daje preporuke za razvoj softvera;
- Lean razvoj koji se fokusira na otklanjanje aktivnosti koje ne daju nikakvu vrijednost i korist. Principi koji se koriste vrlo su slični onima na primjeru agilnih metoda;
- *Test-Driven Development* (TDD) se oslanja na kratke cikluse razvoja koji se ponavljaju. Prvo, programer piše automatizirani test slučaj za novu funkciju i brzo dodaje test s minimalnom količinom koda kako bi prošao taj test. Onda razvija novi kod do prihvatljivih standarda.

Daje se zaključiti kako agilne metode pomažu u fokusiranju na korisnike ili klijente, te njihove stvarne želje i potrebe. Rezultat toga je stvaranje kvalitetnijeg proizvoda boljih performansi.

5.ZAKLJUČAK

Softversko inženjerstvo kompleksno je znanstveno područje, unatoč relativno kratkom povijesnom razvoju istoga. ono što je doprinijelo njegovoj kompleksnosti uglavnom se očituje kroz značaj i ulogu softverskog inženjeringa u poslovnom i svakodnevnom životu, posebice od pojave suvremenoga doba. Smatra se kako se ono intenzivno i dinamično razvije tijekom posljednjih 30-ak godina, a nastavak ovoga trenda izvjestan je i u budućnosti.

Jedan od temeljnih elemenata komplesnog znanstvenog područja softverskog inženjeringa, a koje se često razmatra i kao osnova njegova postojanja i razvoja, jest proces formiranja softvera, programa ili softverskog proizvoda. Ovaj proces najjednostavnije je pojmiti kao svrhu ili temeljni cilj softverskog ili programskog inženjeringa. Međutim, važno je istaknuti kako nije dovoljno samo razviti proizvod, već je važno razviti isti koji će raspolagati što boljim karakteristikama, ostvarivati optimalne rezultate i performanse, a na cjelovit i kvalitetan način zadovoljiti zahtjeve i želje korisnika. Pored toga, važno je razmotriti i troškovni aspekt, to jest razviti spomenuti proizvod uz što niže troškove.

Proces razvoja programa često se razmatra kao životni ciklus istoga, koji je sačinjen od nekoliko osnovnih faza. Iako se one različito klasificiraju s obzirom na autore, predmet istraživanja i slično, načelno je riječ o jednakom tijeku procesa u praksi. U svrhu njegove provedbe na raspolaganju je niz metoda, tehnika i instrumenata. Pri tome, svi ovi elementi, koji čine resurse samoga procesa moraju biti adekvatno normatizirani i standardizirani te kvalitetno integrirani.

Metode razvoja programa mogu se generalno razmatrati kroz klasične ili tradicionalne, odnosno sekvencijske i agilne, odnosno moderne ili suvremene metode. Već se iz toga daju istaknuti osnovne razlike među njima, a generalno je moguće zaključiti kako su agilne metode daleko kompleksnije, učinkovitije, kvaliteti orijentiranije, fleksibilnije i prilagođene klijentima te kupcima. Upravo zbog tih obilježja vjeruje se kako daju bolje rezultate.

U obje skupine moguće je identificirati nekoliko modela ili metoda razvoja programa, a njihov broj u budućnosti će nastaviti rasti. Međutim, očekuje se kako će nešto veći naglasak biti postavljen na razvoj i unapređenje agilne metodologije, uslijed karakteristika i funkcija ovih metoda koje optimalno zadovoljavaju trendove, zahtjeve i prakse suvremenoga doba i globalnog društva.

LITERATURA

Knjige:

1. Maciaszek, L. A. (2007) Requirements Analysis and System Design. Harlow: Pearson Education Limited
2. Manger, R., Mauher, M. (2010) Programsko inženjerstvo – priručnik. Zagreb: Algebra d.o.o.
3. Matić, M. et al. (2016) Razvoj i primjena informacijskih sustava. Zagreb: Tehničko veleučilište u Zagrebu.
4. Sommerville, I. (2016) Softwre Wngineering. Harlow: Pearson Education Limited

Članci:

Slavek, N. et al. (2012) Model mjerenja softverskog procesa. Tehnički vijesnik. Vol. 19. No. 1. Str. 11.-17.

Internet izvori:

1. AES (2010) Agilne metode. Dostupno na: https://www.aes.hr/_download/repository/AgilneMetode2010.pdf (06.09.2018.)
2. Bileta, T. (2016) Kristalna obitelj metodologija razvoja programskih proizvoda. Dostupno na: https://bib.irb.hr/datoteka/834547.bileta_toni_unipu_2016_zavrs_sveuc.pdf (18.07.2018)
3. Crunchbase (2018) CASE technologijes. Dostupno na: https://www.crunchbase.com/organization/cace-technologies (18.07.2018)
4. Galinac Grbac, T. (2018) Modeli procesa razvoja programskog proizvoda. Dostupno na: http://www.riteh.uniri.hr/zav_katd_sluz/zr/nastava/proginz/materijali/Modeli_procesa.pdf (18.07.2018).

5. Guru (2018) What is V-model?. Dostupno na: https://www.guru99.com/software-testing-lifecycle.html (18.07.2018).
6. Kovaček, M. (2016) Ekstremno programiranje. Dostupno na: https://bib.irb.hr/datoteka/835422.kovacek_mateo_unipu_2016_zavrs_sveuc.pdf (06.09.2018)
7. Milošević, B. (2013) CASE alati. Dostupno na: http://vtsnis.edu.rs/starisajt/specijalisticke_studije/4_srt/predmeti/programski_alati_za_razvoj_aplikacija/CASE%20alati%20PRIKAZ.pdf (18.07.2018).
8. Milošević, D. (2016) Agilna metodologija. Dostupno na: https://dusanmilosevic.com/agilna-metodologija/ (18.07.2018).
9. Novitas (2018) Ekstremno programiranje. Dostupno na: https://www.info-novitas.hr/o-nama/metodologije-rada/ekstremno-programiranje-xp/ (18.07.2018).
10. Šimunović, D. (2017) Analiza primjena agilnih metoda u razvoju softvera kod IT tvrtki s obzirom na globalne trendove. Dostupno na: https://repozitorij.efst.unist.hr/islandora/object/efst:1090/preview (06.09.2018)
11. Tadić, B. (2005) Ekstremno programiranje i primjena na balkanu. Dostupno na: http://www.quality.unze.ba/zbornici/QUALITY%202005/034-Q05-028.pdf (18.07.2018)
12. Znanje (2018) Osnove računara. Dostupno na: http://www.znanje.org/abc/tutorials/computer_basic/01/060_software.htm (17.07.2018).

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Unatoč relativno kratkoj povijesti razvoja, softversko inženjerstvo do danas se razvija sve dinamičnije i intenzivnije. Sukladno tome, ističe se istinskom kompleksnošću, ali i značajem u suvremeno doba. Smatra se kako ono daje izravnu podršku suvremenom društvu i međunarodnoj ekonomiji. Generalno se može pojmiti kao znanstveno područje u svezi razvoja i upravljanja softverima to jest programskim rješenjima.

Kada je riječ o razvoju softvera, zapravo se misli na fazni proces njegova životnog ciklusa. Svaka faza pri tome je jasno određena, a podrška provedbi ovoga procesa dolazi od integriranog, normatiziranog i standardiziranog kombiniranja metoda, tehnika i tehnologija.

Agilne metode javljaju se kao odgovor na pritiske suvremenoga doba i nedostataka klasičnih metoda programiranja. One otklanjaju vodeće nedostatke sekvencijksih, tradicionalno orijentiranih metoda. Pri tome, ističu se većom fleksibilnošću, kvalitetom i orijentaciji prema korisnicima. U praksi donose veću učinkovitost i efikasnost.

Ključne riječi: softversko inženjerstvo, softver, životni ciklus softvera, agilne metode, sekvencijske metode.

SUMMARY

Despite a relatively short history of development, software engineering today is becoming increasingly dynamic and intensive. Accordingly, it is emphasized by the true complexity, but also by its significance in the contemporary time. It is considered to be a direct support to the modern society and the international economy. It can be generally considered as a scientific area for software development and management, that is, software solutions.

Software development actually consideres thephases of its life cycle process. Each phase is clearly defined and has to be supportedwith the implementation of integrated, normatized and standraded combination of methods, techniques and technologies.

Agile methods appear as a response to contemporary pressures and the disadvantages of the classical programming methods. They eliminate the main disadvantages of sequential, traditionally-oriented methods. In this regard, greater flexibility, quality and customer orientation are emphasized. In practice they bring greater efficiency and quality for customers.

Key words: software engineering, software, software life cycle, agile methods, sequencing methods.