UNIVERZITET U SARAJEVU
EKONOMSKI FAKULTET

ZAVRŠNI RAD

PREDNOSTI AGILNOG PRISTUPA U RAZVOJU SOFTVERA – PRIMJENA SCRUM-A

Sarajevo, april 2024.godine MERISA KASUMOVIĆ

U skladu sa članom 54. Pravila studiranja za I, II ciklus studija, integrisani, stručni i specijalistički studij na Univerzitetu u Sarajevu, daje se

IZJAVA O AUTENTIČNOSTI RADA

Ja, **Merisa Kaumović**, student/studentica drugog (II) ciklusa studija, broj index-a **5234** na programu **Master studij u saradnji sa Microsoft-om MA** + 1, smjer **Menadžment i Informacioni sistemi**, izjavljujem da sam završni rad na temu:

PREDNOSTI AGILNOG PRISTUPA U RAZVOJU SOFTVERA – PRIMJENA SCRUM-A

pod mentorstvom Prof. Dr. Nijaza Bajgorića izradio/izradila samostalno i da se zasniva na rezultatima mog vlastitog istraživanja. Rad ne sadrži prethodno objavljene ili neobjavljene materijale drugih autora, osim onih koji su priznati navođenjem literature i drugih izvora informacija uključujući i alate umjetne inteligencije.

Ovom izjavom potvrđujem da sam za potrebe arhiviranja predao/predala elektronsku verziju rada koja je istovjetna štampanoj verziji završnog rada.

Dozvoljavam objavu ličnih podataka vezanih za završetak studija (ime, prezime, datum i mjesto rođenja, datum odbrane rada, naslov rada) na web stranici i u publikacijama Univerziteta u Sarajevu i Ekonomskog fakulteta.

U skladu sa članom 34. 45. i 46. Zakona o autorskom i srodnim pravima (Službeni glasnik BiH, 63/10) dozvoljavam da gore navedeni završni rad bude trajno pohranjen u Institucionalnom repozitoriju Univerziteta u Sarajevu i Ekonomskog fakulteta i da javno bude dostupan svima.

Sarajevo, april 2024. godine Potpis studenta/studentice:

SAŽETAK

Istraživanje se bavilo analizom primjene SCRUM-a kao agilne metoda razvoja softvera u na primjeru projektnih timova u Mistralu. Teorijski dio obuhvata analizu osnovni pojmova vezanih za temu - agilne i tradicionalne metoda razvoja softvera, SCRUM, historija SCRUM-a, SCRUM procesi itd.

Empirijsko istraživanje je izvršeno na uzorku od 100 ispitanika, članova projektnog tima u kompaniji Mistral BiH. Istraživanje je urađeno u periodu od jul-augusta 2023, te je njime potvrđeno svih sedam hipoteza istraživanja. Ovo pokazuje generalno pozitivan uticaj korištenja SCRUM-a u projektnim timovima kompanija.

Ključne riječi: SCRUM, Agilne metodologije, Tradicionalne metodologije, Razvoj softvera, Mistral

ABSTRACT

The research dealt with the analysis of the application of SCRUM as an agile method of software development in the example of project teams at Mistral. The theoretical part includes the analysis of basic terms related to the topic - agile and traditional methods of software development, SCRUM, history of SCRUM, SCRUM processes, etc.

Empirical research was conducted on a sample of 100 respondents, members of the project team at the company Mistral BiH. The research was conducted in the period from July-August 2023, and it confirmed all seven research hypotheses. This shows the generally positive impact of using SCRUM in company project teams.

Keywords: SCRUM, Agile methodologies, Traditional methodologies, Software development, Mistral

POPIS SLIKA

POPIS TABELA

1. UVOD

1.1. Predmet i obrazloženje teme

Upravljanje projektima je trenutno tema o kojoj se dosta piše. U različitim sektorima, građevinski, tehnološki, promijenio se način izrade projekata dok se metodologija upravljanja projektima nije značajno promijenila od 60thih godina prošlog vijeka. Navedeno je rezultiralo jazom između menadžerske perspektive i načina na koji se projekti izvode i upravo je to razlog za traženje novih pristupa u upravljanju projektima (Pareliya, 2018).

Upravljanje projektima je ključ za pronalaženje načina upravljanja, kontrole i koordinacije projekta bilo koje veličine odabirom metoda rada, definisanjem projektnih uloga, pojednostavljenjem projektnog izvještavanja te stalnim praćenjem planiranja tokom trajanja cijelog projekta. Danas, upravljanje projektima utiče na cijelu organizaciju, bez obzira da le se radi o malom preduzeću ili pak velikom javnom (Owen, *et.al,* 2006).

Prošlo je već više od pola vijeka od upravljanja projktima koristeći tradicionalne metode. Međutim, stvarni način upravljanja projetima sada je drugačiji. Danas postoji jaz između tradicionalnog upravljanja projektima i novih metoda. U fazi planiranja izgradnje projekta kada je svaka faza isplanirana mogućnost uticaja na dizajn projekta i proces planiranja je veoma niska. Štaviše, u ovoj fazi konstanto se povećava količina potrošenog novca. Nakon što krene sama izgradnja projekta neprepoznavanje greški može postati veoma skupo u smislu izgubljenog vremena i utrošenog novca. Zbog toga je veoma važno posmatrati te greške i poduzeti sve da se iste ne ponavljaju (Cervone, 2011).

Agilno upravljanje projektima ima svoje korijene u industriji razvoja softvera i razvilo se kroz empirijski napredak. Međutim, upotreba ove metodologije nije ograničena na tu industriju. Ova metodologija definiše vrijednosti i principe koje mogu usvojiti i druge industrije (Owen, *et.al,* 2006).

Za razliku od agilnog tradicionalno upravljanje projektima koje je još poztanto pod nazivom *Waterfall* sastoji se od jednog unaprijed definisanog puta kojeg slijedi cijela organizacija i koji se sastoji od pet faza i to faze inicijacije, faze planiranja, faze izvođenja, faze praćenja i kontrole i posljednje faze zatvaranja (Pareliya, 2018).

Ova metodologija ima skup tehnika koje pomažu projektu da pravovremeno postigne uspjeh unutar budžeta i planiranih specifikacija. Koristi se uglavnom kod onih projekata kod kojih se rijetke promjene tokom izgradnje projekta. Tradicionalno upravljanje projektima se uglavnom mjeri na iskustvu i alatima (Hicks i Foster, 2010).

S druge strane, SCRUM se u potpunosti temelji na praksi agilnog razvoja softvera i posebno se primjenjuje kod upravljanja softverskim proizvodima. Ovdje je bitno istaknuti

da se alati agilnog upravljanja značajno razlikuju od uobičajenih alata za razvoj i upravljanje softverom. SCRUM projekat uključuje zajednički napor da se stvori novi proizvod, usluga ili neki drugi rezultat kako je to definisano u Izjavi o viziji projekta. Na projekte utiču vremenska ograničenja, troškovi, obim, kvaliteta, resursi, organizacijske mogućnosti i druga ograničenja koja otežavaju planiranje, izvršavanje, upravljanje i u konačnici uspjeh projekta. Međutim uspješna implementacija rezultata završenog projekta daje značajne poslovne koristi organizaciji. Zbog toga je veoma važno da organizacije odaberu i implementiraju odgovarajući pristup upravljanju projektima (Sapathy, 2013).

SCRUM je jedna od najpopularnijih metoda agilnog upravljanja projektima. Prilagodljiva je, iterativna, fleksibilna, brza i predstavlja efektivan okvir za isporuku značajne vrijednosti brzo i tokom cijelog projekta. Ova metodologija osigurava transparentnost u komunikaciji i stvara okruženje kolektivne odgovornosti i kontinuirani napredak i zadovoljstvo zaposlenika (Sapathy, 2013), što je jedan od glavnih razloga zbog kojih sam izabrala ovu temu.

Predmet ovog istraživanja je agilni pristup upravljnju projekom razvoja softvrea uz korištenje SCRUM u IT sektoru u BiH odnosno kako korištenje SCRUM-a utiče na zadovoljstvo zaposlenih.

1.2. Ciljevi, svrha i problem istraživanja

Ovo istraživanje za cilj ima identifikovati strukturisati temeljne izazove s kojima se suočavaja projekt menadžeri i organizacijski lideri u pravljanju velikim projektima razvoja softvera. Istraživački dio radi se u okviru industrije razvoja softvera koja pruža usluge razvoja softvera poput savjetovanja, internog razvoja, eksternalizacije resursa/outsourcinga ili na temelju proizvoda/rješenja za razvoj softvera. Budući da je fokus istraživanja upravljanje projektima velikih razmjera najvjerovatnije će uključiti sve vrste razvoja bez obzira na tehnologiju, platformu, kompetencije itd. Primarni fokus istraživanje će biti istražiti kako primjena SCRUMA utiče na zadovoljstvo zapoelsnih i produktivnost u IT sektoru. Rezultati istraživanja temeljit će se na kolektivnom iskustvu sagovornika i pregledu literature.

Agilno upravljanje projektima je širok pojam i uključuje nekoliko zanimljivih područja istraživanja gdje komunikacija i povjerenje spadaju u najzanimljivija područja. Dakle, problem istraživanja je ograničen fond o agilnom pristupu upravljanja projetima - primjena SCRUM u razvoja softvera u Mistralu u BIH s fokusom na komunikaciju i povjerenje.

1.3. Polazne hipoteze

H1: Primjena SCRUM metode ima pozitivan uticaj na zadovoljstvo zaposlenika u kompaniji.

H2: SCRUM metoda poboljšava saradnju među članovima tima i pozitivno utiče na stvaranje prijateljskog radnog okruženja u kompaniji.

H3: SCRUM metoda olakšava dobijanje konstruktivnijih povratnih infirmacija od kolega u timu.

H4: SCRUM metoda pomaže prepoznavanje truda i zalaganja kod zaposlenika, pomaže da se čuju njihova mišljenja i prepozna njihov trud.

H5: SCRUM omugućava zaposlenicima više slobode kod procesa donošenja odluka u kompaniji Mistral BiH. Naime, na taj način zaposlenici su direktno uključeni u proces donošenja odluka, odnosno svoja mišljenja ili kritike mogu dijeliti s drugim članovima tima.

H6: SCRUM metoda pomaže ravnomjernoj raspodjeli zadataka među članovima tima u kompaniji.

H7: SCRUM metoda odnosno njegova primjena ima pozitiva efekat na usavršavanje zaposlenika i njhovu produktivnost i lični razvoj.

1.4. Istraživačka pitanja

1. Koje su prednosti korištenja agilne metodologije?
2. Kako se može upravljati komunikacijom i povjerenjem u razvoju softvera velikih razmjera?
3. Koje su to prepreke i izazovi kada se govori o komunikaciji i povjerenju kod upravljanja projktima velikih razmjera?
4. Koji su efekti nedostatka komunikacije i povjerenja?
5. Koja su moguća rješenja za prevazilaženje navedenih izazova?
6. Kako primjena SCRUM-a utiče na zadovoljstvo i produktivnost zaposlenih?
7. Da li primjena SCRUM-a pomaže zaposlenicima da kreiraju kvalitetniji proizvod (u smislu da smanjuje nivo stresa) i doprinosi boljem balansu između radnih onaveza iprivatnog života?

8. Da li primjena SCRUM-a ima pozitivan efekat općenito na radnu kulturu i transparenetnost te kao takva predstavlja prikladan okvir za upravljanje IT projektima?

1.5. Metodologija

U cilju ispitivanja istinitosti polaznih hipoteza kao i dogovora na postavljena istraživačka pitanja bit će korištene kvantitativne i kvalitativne metode istraživanja. Prikupljanje podataka bit ce pomocu online anketnog upitnika koji ce biti poslani uposlenicima kompanije Mistral u Sarajevu.

Ispitni uzorak je 100 ispitanika. Ispitanici će biti zaposlenici, članovi projektnog tima u kompaniji Mistral BiH.

Anketni uptinik će biti napravljen uz pomoć alata Google forms. Za ispitivanje istinitsti tvrdnji iz anketnog upitnika koristit će se Likertova skala s pet mogućnosti gdje je 1=u potpunosti se slažem, 2=slažem se, 3=niti se slažem niti se ne slažem, 4=ne slažem se, 5=u potpunsti se ne slažem. Upitnik se se sastoji od dva dijela. Prvi dio uptinika s eodnosi na demografske karakteristike ispitanika. Drugi dio upitnika se odnosi na izjave/stavove ispitanika o efektima SCRUM-a na različite segmente rada i života zaposlenika. U samom upitniku su navedene i pozitivne i negativne izjave kako bi se osiguralo da ispitanici pažljivo pročitaju svaku tvrdnju prije nego što na nju odgovore. Za nalizu rezultata koristit će se IBM-ov statistički softver SPSS i do Cronbach alpha, deskriptivne statistika i po potrebi i drugi statistički testovi.

U cilju dubljeg razumjevanja problematike radit će se i intervju među odabranim najstručnijim članovima SCRUM tima.

Intervju će biti obavljejeni preko online aplikacija (Google meet, Zoom etc). U ovom istraživanju koristit će se i osnovne istraživačke metode, kao što su metoda analize, metoda indukcije, metoda dedukcije, komparacije.

1.6. Doprinos istraživanja

Rezultati rada će biti od koristi menadžerima/voditeljima organizacija različitih veličina koji su zainteresirani za razumijevanje izazova na organizacijskom ili timskom nivou s kojima se suočavaju menadžeri globalno. Predstavljene strategije će također pomoći i studentima u prevazilaženju izazova i pronalasku rješenja kod upavljna projektima velkih razmjera u razvoju softvera. Dalje, rad će dati doprinos razumijevanju kritičnih faktora uspješnosti primjene metodologija projektnog menadžemnta te kako bi se mogla poboljšati efikasnost istih. Istraživanje će predstaviti i preporuke koje će pomoći projekt menadžerima da lakše prvazilaze brojne izazvose s kojima se svakodnevno susreću u svom radu.

2. UVOD U SOFTVERSKI INŽENJERING

Tradicionalne metodologije softverskog inženjeringa vođene su planom u kome rad počinje otkrivanjem i dokumentacijom, dok kompletan set zahtijeva praćenje i inspekciju arhitektonske i projektantske izrade. Ova metodologija postala je poznata kao teška zbog navedenih aspekata. Neki praktičari smatrali su ovaj proces koji je fokusiran na razvoj softvera frustrirajućim te su tvrdili da predstavlja poteškoće kada su stope promjena još uvijek relativno niske. Nekoliko konsultanata je kao rezultat toga razvilo metodologije i prakse prihvatanja i odgovaranja na neizbježne promjene. Navedene metodologije i prakse zasnovane su na iterativnim poboljšanjima, što je tehnika uvedena 1975. godine te koja je kasnije postala poznata kao agilne metodologije (Alian i Javanmard, 2015).

Naziv "agilni" je nastao 2001. godine, kada je sedamnaest metodologa ovog procesa održalo sastanak na kome su razgovarali o budućim trendovima u razvoju softvera. Ovi metodolozi su primijetili da njihove metode imaju mnoge zajedničke karakteristike (da su lagani i dovoljni), što je dovelo do imena agilni procesi. Kao posljedica ovog sastanka pojavio se "Agile Alliance" i njen manifest za agilni razvoj softvera (Awad, 2005).

Agilne metodologije razotkrivaju organizacionu disfunkciju. Za razliku od tradicionalnih, agilne metodologije prihvaćaju iteracije, a ne faze. Agilne metodologije koriste kratke iterativne cikluse, mala/kratka izdanja/ jednostavan dizajn, refaktorisanje kontinuirane integracije te oslanjanje na prešutno znanje unutar tima za razliku od dokumentacije. Neke od popularnih agilnih metoda su Extreme Programming, SCRUM, Lean, Kanban, Dynamic System Metoda razvoja, razvoj vođen funkcijama i adaptivni razvoj softvera (Alian i Javanmard, 2015).

Ključna razlika između teških i agilnih metodologija jeste faktor prilagodljivosti. Kada je u agilnim metodologijama potrebna velika promjena, tim ne zamrzava svoj radni proces, već određuje način na koji je bolje postupati sa promjenama koje se dešavaju tokom cijelog projekta. Proces verifikacije u agilnim metodama javlja se mnogo ranije u procesu razvoja. S druge strane, teške metode zamrzavaju zahtjeve za proizvodnju i zabranjuju promjene. One implementiraju prediktivni proces te se oslanjaju na definisanje i dokumentovanje stabilnog skupa zahtjeva na početku projekta (Alian i Javanmard, 2015).

2.1. Metodologije razvoja softvera

Razvoj softvera predstavlja veoma složenu aktivnost. Njega karakteriziraju promjenjivi zahtjevi, potreba za specijalizovanim i raznovrsnim vještinama, promjenljiva i sofisticirana tehnologija koja se koristi za razvoj i implementaciju softvera, te poteškoće u upravljanju ljudima koji se svakodnevno bore sa ovakvom složenošću. Nije neuobičajeno naići na organizacije koje su preplavljene inherentnom složenošću pri implementaciji projekata razvoja sistem. Zbog toga je procesa razvoja sistema moguće opisati kao složene,

nepredvidive i loše definisane. Drugim riječima, ovi procesi ne posjeduju dobro definisane ulaze i izlaze te se stoga smatraju neponovljivim (Devi, 2013).

U posljednju deceniju ili dvije razvoj softvera je prošao kroz značajne promjene. Dolazak ekstremnog programiranja (XP) u kasnom 20. vijeku predstavljao je nastanak (ali ne i rađanje) novog pristupa koji je postao Agilni razvoj softvera (eng. ASD - Agile Software Development). Historija razvoja softvera postojala je već i prethodno tomu, čak sve do perioda prije prvih elektronskih digitalnih računara koji su nastali 1940-ih godina. Nakon početnog perioda kada se smatrao zanatom, na razvoj softvera obično se gledalo kao na neku vrstu inženjeringa. Činilo se da agilni razvoj softvera pruža novu alternativu, koja je pomalo zanatski nastrojena ali razvijenija. Ipak, i dalje postoje oni koji na ASD gledaju kao na oblik softverskog inženjeringa (Devi, 2013).

Prema Devi (2013), navedeno je donijelo određene probleme. To je zbog toga što je nastala značajna promjena u pristupu, dok ASD nije nužno ono što se smatra softverskim inženjerstvom - tačnije, softverski inženjering danas razlikuje se od onoga što se prije smatralo softverskim inženjeringom.

2.1.1. Tradicionalni razvoj softvera

Skoro od svog početka, racionalan pristup zasnovan na inženjeringu, kao što je metoda vodopada, koristi se za razvijanje projekata. Čini se da je ovaj sistem bio utemeljen na "tvrdosistemskom razmišljanju", koje pretpostavlja da problemi mogu biti dobro definisani, da se procesi mogu optimizovati, a rezultati da se mogu dobro predvidjeti. Također je urađeno opsežno planiranje unaprijed, sa ciljem kontrolisanja varijacija u životnom ciklusu razvoja (Alian i Javanmard, 2015).

Suštinska priroda tradicionalnog životnog ciklusa razvoja softvera je sljedeća (Alian i Javanmard, 2015):

1. Cilja se ka temeljnom razumijevanju potreba korisnika, izradi solidnog dizajna, besprijekornom razvoju softvera, te implementaciji funkcionalnog sistema koji bi zadovoljio potrebe korisnika;

2. Veliki naglasak stavlja se na temeljno planiranje za suočavanje s rizicima;

3. Ovaj ciklus je zasnovan na principima tvrdih sistema - dakle na identifikovanju alternativnih načina da se od početnog stanja (S0) dođe do željenog stanja (S1) te da se izabere najbolji način za postizanje istog (S0 - S1);

4. Ovakav pristup pretpostavlja dobro definisanje problema te optimalno rješenje do koga se dolazi opsežnim, unaprijednim planiranjem;

5. Ovaj pristup također pretpostavlja da su procesi predvidljivi te da se mogu optimizovati i učiniti ponovljivim;

6. Ovaj se pristup također zasniva na pretpostavci da se procesi mogu adekvatno izmjeriti te da se može identifikovati i kontrolisati izvore varijacije tijekom životnog ciklusa razvoja;

7. Jednostavno rečeno, tradicionalni životni ciklus razvoja softvera visoko je usmjeren na proces.

Na osnovu prethodnog razumijevanja razvoja sistema, organizacije usvajaju stil upravljanja koji je, prema Devi (2013) (a) Zasnovan na komandi i kontroli, sa postavljenom hijerarhijom. To je razlog zbog koga su pretežno mehaničke organizacije usmjerene na visoke performanse u stabilnom okruženju; (b) Odlikuje se visokom formalizacijom i standardizacijom. Ljudima sa različitim specijalizacijama dodijeljene su uloge kako bi se proizveli definisani ishodi. Pored toga, proizvode i značajnu količinu dokumentacije koja objašnjava softver i njegove tehničke i dizajnerske specifikacije; (c) Značajan je i po tome što, unatoč tome što kupci igraju važnu ulogu, njihovo učešće je maksimalno samo tijekom faze specifikacije i implementacije.

Tradicionalni pristupi iz perspektive kupaca imaju prednosti i nedostatke. Definitivna prednost tradicionalnih pristupa je njihova skalabilnost. Veoma veliki projekti i projekti koji su od kritične važnosti za misiju zahtijevaju snažan plan te zatvaranje nadzora i potrebna im je stabilnost. Nedostaci ovih pristupa zasnovani su na pretpostavci da se zahtjevi kupaca dobro razumiju na početku te da se ne mijenjaju mnogo, što je rijedak slučaj. Drugi fundamentalni problem je što ovi procesi predugo traju, što dovodi do toga da se mnoge stvari (uključujući zahtijeve korisnika), drastično promijene do momenta kada oni vide krajnji sistem (Awad, 2005).

Ukratko, postoje dvije široke terme koje se pojavljuju pri razvoju tradicionalnih sistema, oko kojih su centrirani problemi (Alian i Javanmard, 2015):

1. Životni ciklus razvoja i njegovi procesi - predstavljaju način na koji su procesi shvaćeni i dizajnirani te kojima se upravlja uz visok fokus na proces. Ovdje se koristi tvrdosistemsko razmišljanje, koje je zasnovano na inženjeringu u svom pristupu. Fokus je na postizanju stabilnosti;

2. Stil upravljanja - predstavlja način na koji su ljudi organizovani i kontrolisani te na koji se njima upravlja, što se postiže kroz "visoko komandovanje i kontrolu", te formalizovan i standardizovan pristup sa ograničenom interakcijom sa klijentima (Devi, 2013).

2.1.1.1. WaterFall

Ova metoda uključuje uzastopno dovršavanje svake faze u potpunosti nakon koga slijedi prelazak na sljedeću fazu. Waterfall nije bio namijenjen da bude praktično koristan pristup razvoju softvera. Često se nema razumijevanja za činjenicu da zahtjeve nije moguće u potpunosti ispuniti - programeri softvera moraju riješiti barem dio zadatka, napose problem domene. Praktično je nemoguće da analiza, dizajn i implementacija budu u potpunosti završene5. Dio razloga je i to što se tokom razvoja dešava učenje, što kasnije rezultira potrebom da se raniji rezultati modificiraju. U stvarnosti, određeni projekti za cilj imaju približavanje Waterfall SDLC, sa što manje dorada ili ispravki koliko je moguće (Alian i Javanmard, 2015).

Waterfall pristup naglašava strukturiranu progresiju između definiranih faza. Svaka faza sastoji se od određenog skupa aktivnosti i rezultata koje je potrebno ostvariti prije početka sljedeće faze. Faze se uvijek nazivaju različitim imenima, ali osnovna ideja je da prva faza pokušava uhvatiti šta radi sistem, njegove sistemske i softverske zahtjeve, dok druga faza određuje kako će isti biti dizajniran. Treća faza predstavlja mjesto gdje programeri počinju testiranje sistema, dok je završna faza fokusirana na zadatke implementacije kao što su trening i teška dokumentacija. Međutim u inženjerskoj praksi, termin vodopad se koristi kao generički naziv za sve metodologije sekvencijalnog softverskog inženjeringa (Awad, 2005).

Waterfall model definirao je Winston W. Royce 1970. godine. Ovaj model poznat je i kao model linearnog sekvencijalnog životnog ciklusa. On je jednostavan za razumijevanje i upotrebu. Svaka faza mora biti završena prije nego sljedeći model može početi, a na kraju svake faze projekat se revidira sa ciljem osiguravanja usklađenosti sa zahtjevima (Stoica *et.al,* 2013).

Neke od prednosti ovoga modela su: (a) projektiranje dokumentacije i strukture predstavljaju prednost kod novih članova koji se pridružuju timu; (b) lako ga je razumjeti i koristiti; (c) lako ga je koordinirati zahvaljujući modelu krutosti - svaka faza ima očekivane rezultate i proces evaluacije; (d) faze se provode jedna po jedna, u sekvencama; (e) ovaj model se preporučuje za male projekte, sa jasno shvaćenim zahtjevima (Stoica *et.al,* 2013).

Neki od nedostataka modela su: (a) neki zahtjevi mogu nastati nakon što je početno prikupljanje zahtjeva završeno, što negativno utiče na razvoj proizvoda; (b) svi problemi koji su otkriveni tokom jedne faze nisu potpuno riješeni tijekom iste faze; (c) nema fleksibilnosti u podjeli projekta u faze; (d) novi zahtjevi koje dodaju klijenti mogu dovesti do dodatnih troškova, budući da se oni ne mogu riješiti u trenutnom izdanju proizvoda; (e) teško je procijeniti vrijeme i budžet za svaku fazu; (f) ne može se izraditi prototip do završetka životnog ciklusa; (g) ako testiranje otkrije određene probleme, veoma je teško vratiti se u fazu projektovanja; (h) postoje visok rizik i neizvjesnost; (g) ne preporučuje se

za složene i objektno orijentisane projekte. Na kraju, Waterfall model preporučuje se u sljedećim slučajevima: (a) kada su zahtjevi dobro shvaćeni, jasni i konačni; (b) kada je definicija proizvoda stabilna; (c) kada se tehnologija razumije; (d) kada ne postoje dvosmisleni zahtjevi; (e) kada su resursi koji uključuju stručnost slobodno dostupni; (f) kada je projekat kratak (Stoica *et.al,* 2013).

2.1.1.2. Spiral Model

Spiralni model također predstavlja jednoga od kandidata za poboljšanje situacije oko odabira modela softverskog procesa. Glavna karakteristika spiralnog modela jeste to da on stvara pristup softverskom procesu vođenom rizikom, a ne primarno procesu vođenom dokumentom ili kodom. Ovaj model uključuje mnoge prednosti drugih modela, rješavajući istovremeno neke od njihovih poteškoća (Boehm, 2005).

možda maznuti sliku.... Spiralni model softverskog procesa razvijao se tokom nekoliko godina, na osnovu iskustva sa raznim usavršavanjima modela vodopada primijenjenog na velike vladine softverske projekte. Kao što će biti diskutovano, spiralni model može prihvatiti većinu prethodnih modela kao posebne slučajeve te i dalje pružati smjernice o tome koja kombinacije prethodnih modela najbolje odgovara datoj softverskoj situaciji. Razvoj TRW softverskog sistema produktivnosti (TRW-SPS) predstavlja najpotpuniju primjenu ovog modela do sada (Boehm, 2005).

Svaki ciklus spirale počinje identifikacijom: (a) Ciljeva dijela proizvoda koji se razrađuje (performanse, funkcionalnost, sposobnost prilagođavanja promjenama, itd); (b) Alternativnim načinima implementacije ovog dijela proizvoda (dizajn A, dizajn B, ponovna upotreba, kupovina itd.); (c) ograničenjima koje nameće primjena alternative (cijena, raspored, interfejs itd) (SNSCT, 2010).

Sljedeći korak jeste evaluacija alternativa u odnosu na ciljeve i ograničenja. Ovaj proces će često identificirati područja neizvjesnosti koja predstavljaju značajne izvore projektnog rizika. U tom slučaju, potrebno je da sljedeći korak uključuje formulisanje isplative strategije za rješavanje izvora rizika. Navedeno može uključivati izradu prototipa, simulacija, benchmarking, provjeru referenci, administriranje upitnika korisnika, analitičko modeliranje, ili kombinacije ovih i drugih tehnika rješavanja rizika (TutorialsPoint, 2009).

Kada su rizici procijenjeni, sljedeći korak određen je relativnim preostalim rizicima. Ako rizici performansi ili korisničkog sučelja snažno dominiraju razvojem programa ili unutrašnjim rizicima kontrole interfejsa, sljedeći korak može biti evolutivni razvoj - minimalan napor da se specificira cjelokupna priroda proizvoda, plan za sljedeći nivo izrade prototipa te razvoj detaljnijeg prototipa za nastavak rješavanja velikih problema rizika (SNSCT, 2010).

U slučajevima kada je ovaj prototip operativno koristan i dovoljno robustan da služi kao niskorizična osnova za buduću evoluciju proizvoda, naredni koraci vođeni rizikom bi bili

razvoj serija evolucijskih prototipova (slika 2 možda maznut). Opcija pisanja specifikacija u ovom slučaju bi se adresirala ali ne i koristila. Dakle, rizik razmatranja može dovesti do toga da projekat implementira samo podskup svih potencijalnih koraka u modelu (SNSCT, 2010).

S druge strane, u slučajevima kada su prethodni pokušaji izrade prototipa već riješili sve rizike performansi ili korisničkog sučelja, te dominiraju rizici razvoja programa ili kontrole interfejsa, sljedeći korak slijedi osnovni pristup Waterfalla (koncept rada, softverski zahtjevi, idejni projekat itd.), modifikovan prema potrebi uključivanja inkrementalnog razvoja. Svaki nivo specifikacije softvera slijedi korak validacija i priprema planova za naredni ciklus. U ovom slučaju, opcije za prototip, simulaciju, modeliranje i tako dalje su adresirane, ali ne i vježba, što dovodi do upotrebe drugačijeg podskupa koraka (TutorialsPoint, 2009).

Ova podskupina koraka spiralnog modela vođena rizikom omogućava modelu smještanje bilo koje odgovarajuće mješavine orijentisanja na specifikacije, orijentisanja na prototip, orijentisanja na simulaciju, orijentisanja na automatsku transformaciju, te orijentisanja na drugi pristup razvoju softvera. U takvim slučajevima, bira se odgovarajuća mješovita strategija uzimajući u obzir relativnu veličinu rizika programa i relativnu efikasnost različitih tehnika u rješavanju rizika. Na sličan način razmatranja upravljanja rizikom mogu odrediti količinu vremena i truda koji treba posvetiti takvom projektu, kao što su planiranje, upravljanje konfiguracijom, osiguranje kvaliteta, formalna verifikacija i testiranje. Konkretno, specifikacije vođene rizikom mogu imati različite stepene potpunosti, formalnosti i granularnosti, u zavisnosti od relativnih rizika od premalo ili previše specifikacije (SNSCT, 2010).

Važna karakteristika modela kičme, kao i kod većine drugih modela, je da se svaki ciklus završava pregledom koji uključuje dotične primarne ljude ili organizacije sa proizvodom. Ova recenzija pokriva sve proizvode razvijene tokom prethodnog ciklusa, uključujući planove za naredni ciklus i resurse potrebne za njihovo sprovođenje. Glavni cilj revizije jeste osiguravanje da su sve zainteresirane strane međusobno posvećene pristupu za sljedeću fazu (SNSCT, 2010).

Planovi za naredne faze mogu uključivati i podjelu proizvoda na inkremente za uzastopni razvoj ili komponente koje će razviti pojedine organizacije ili osobe. U potonjem slučaju, vizualizira se niz paralelnih ciklusa kičme, po jedan za svaku komponentu, uz dodavanje treće dimenzije koncepta. Na primjer, odvojene spirale mogu se razvijati za odvojene softverske komponente ili inkremente. Stoga, korak pregleda i posvećenosti može se kretati od individualnog prolaska kroz dizajn komponenti jednog programera do glavnih zahtjeva pregleda koji uključuje programere, kupce, korisnike i organizacije za održavanje (Boehm, 2005).

Kada se uzme u obzir ovakva prezentacija spiralnog modela, važno je dati odgovore i na pitanja o načinu na koji spirala počinje, načinu na koji se izlazi iz spirale kada se program

želi prekinuti ranije, razlozima iza naglog završavanja spirale te o poboljšanju ili održavanju softvera (Boehm, 2005).

Odgovori na ova pitanja uključuju zapažanje da se model kičme jednako dobro primjenjuje na napore razvoja ili poboljšanja. U oba slučaja, spirala počinje hipotezom da bi određena operativna misija (ili skup misija) mogla biti poboljšana softverskim naporom. Spiralni proces u tom slučaju omogućuje testiranje ove hipoteze: u bilo koje vrijeme, ako hipoteza na prođe test (npr. ako kašnjenja u softveru dovedu do toga da proizvod propusti svoj tržišni prozor ili ako superioran komercijalni proizvod postane dostupan), spirala je prekinuta. Ona se u suprotnom završava instalacijom novog na modificiranom softveru, a hipoteza se testira promatranjem efekta na operativnu misiju. Iskustvo sa operativnom misijom obično vodi do dalje hipoteze o poboljšanju softvera, te se pokretanjem nove spirale održavanja testira hipoteza. Pokretanje, završetak i ponavljanje zadataka i proizvoda u prethodnim ciklusima stoga su implicitno definisani u spiralnom modelu (Boehm, 2005).

Primarna prednost spiralnog modela jeste njegov raspon opcija što mu omogućuje prilagođavanje dobrim karakteristikama postojećih modela softverskih procesa, dok njegov pristup vođen rizikom izbjegava mnoge od njihovih poteškoća. U odgovarajućim situacijama, model spirale postaje ekvivalentan jednom od postojećih modela procesa. U drugim situacijama pruža smjernice o najboljoj kombinaciji postojećih pristupa datom projektu - na primjer, njegova primjena na TRW-SPS pružila je mješavinu specificiranja, izrade prototipa, vođenja rizika i evolucijskog razvoja (Boehm, 2005).

Primarni uslovi pod kojima spiralni model postaje ekvivalentan ostalim glavnim procesnim modelima sažeti su kako slijedi (SNSCT, 2010):

1) Ako projekat ima nizak rizik u oblastima kao što je dobijanje pogrešnog korisničkog interfejsa ili ne ispunjava stroge zahtjeve za performanse te ako postoji visok rizik u predvidljivosti i kontroli budžeta i rasporeda;

2) Ako su zahtjevi softverskog proizvoda vrlo stabilni (što podrazumijeva nizak rizik od skupog dizajna i lomljenja koda zbog promjena zahtjeva tokom razvoja), te ako prisustvo grešaka u softverskom proizvodu predstavlja visok rizik za misiju kojoj služi, ova razmatranja rizika pokreću spiralni model tako da on liči na dvokraki model preciznih specifikacija te na razvoj formalnog deduktivnog programa (SNSCT, 2010);

3) Ako projekat ima nizak rizik u oblastima kao što su gubitak budžeta i raspored predvidljivosti i kontrole, te nailazi na probleme integracije velikih sistema, ili suočavanja sa infromacijskom sklerozom, te ako postoji visok rizik u područjima kao što su dobijanje pogrešnog korisničkog interfejsa ili zahtjeva za podršku odlučivanju korisnika, onda ova razmatranja rizika dovode model kralježnice do ekvivalentnosti evolucijskom modelu razvoja (SNSCT, 2010);

4) Ako su dostupne mogućnosti automatizovanog generisanja softvera, onda se pokreće spiralni model ili predstavlja opciju za brzo prototipiranje primjene modela transformacije, u zavisnosti od razmatranja uključenih rizika;

5) U slučaju ako visokorizični elementi projekta uključuju mješavinu gore navedenih rizičnih stavki, tada spinalni pristup održava odgovarajuću kombinaciju modela iznad, pri čemu će funkcija izbjegavanja rizika generalno izbjeći teškoće drugih modela (TutorialsPoint, 2009).

Spiralni model također nudi i niz dodatnih prednosti:

1) Ranu pažnju usmjerava na opcije koje uključuju ponovnu upotrebu postojećeg softvera. Koraci koji uključuju identifikaciju i evaluaciju alternativa podstiču ove opcije. Ovo uključuje pripremu za evoluciju životnog ciklusa, rast i promjene softverskog proizvoda. Glavni izvori promjene proizvoda uključeni su u ciljeve proizvoda, a pristupi skrivanja informacija predstavljaju atraktivan arhitektonski dizajn alternativa po tome što umanjuju rizik od nemogućnosti prilagođavanja ciljeva naplati proizvoda (SNSCT, 2010).

2) Pruža mehanizam za uključivanje ciljeva kvaliteta softvera u razvoj softverskih proizvoda. Ovaj mehanizam proizlazi iz naglaska na identifikaciji svih vrsta ciljeva i ograničenja tokom svakog kruga spirale (Boehm, 2005);

3) Fokusira se na rano eliminisanje grešaka i neprivlačnih alternativa. Analiza rizika, validacija i koraci posvećenosti pokrivaju ova razmatranja. Za svaki od izvora projektne aktivnosti i utroška resursa odgovara ključno pitanje "Koliko je dovoljno?" Koristeći pristup vođen rizikom, može se vidjeti da odgovor nije isti za sve projekte i da je odgovarajući nivo napora određen nivoom rizika koji nastaje kada se ne čini dovoljno (Boehm, 2005).

4) Ne ukljzučuje odvojene pristupe za razvoj i poboljšanje softvera (ili održavanje). Ovaj aspekt pomaže u izbjegavanju statusa "građanina drugog reda"", koji je često povezan sa održavanjem softvera. Također pomaže u izbjegavanju mnogih od problema koji trenutno nastaju kada se pristupi visokorizičnim naporima za unapređenja na isti način kao i rutinsko održavanje (TutorialsPoint, 2009);

5) Pruža održiv okvir za integrisani hardversko-softverski razvoj sistema. Fokus na upravljanju rizicima i na eliminisanju neatraktivnih alternativa rano i jeftino podjednako je primjenjiv na hardver i softver (TutorialsPoint, 2009).

Dakle, model pune spirale može se uspješno primijeniti u mnogim situacijama, ali neke poteškoće se moraju riješiti prije nego što se može nazvati zrelim, univerzalno primjenljivim modelom. Tri primarna izazova uključuju usklađivanje sa ugovornim softverom, oslanjanje na ekspertizu procjene rizika, te potrebu za daljom razradom spiralnog modela stepenice. Ove poteškoće u nastavku će biti detaljno objašnjene i sumirane (Boehm, 2005):

1) Usklađivanje sa ugovornim softverom - Potreban je dalji rad kako bi se spiralni model uskladio sa svijetom ugovorne nabavke softvera. Interni razvoj softvera posjeduje veliku dozu fleksibilnosti i slobode uskađivanja obaveza po fazama, odlaganja obaveza na specifične opcije, uspostavljanja minispinale za rješavanje stavki kritičnog puta, prilagođavanje nivoa napora ili prilagođavanje takve prakse kao što su prototipovi, evolucijski razvoj ili dizajn prema cijeni. Ove stepene fleksibilnosti i slobode teže je postići u svijetu ugovorne nabavke softvera bez gubljenja odgovornosti i kontrole (Boehm, 2005).

Nedavno je učinjen veliki napredak u uspostavljanju fleksibilnijih mehanizama ugovaranja, kao što je upotreba konkurentnih front-end ugovora za koncept definicija ili prototip fly-offa, korištenja nivoa truda i ugovora o dodjeli naknada za evolucijski razvoj, te korištenja ugovora od dizajna do troškova. Iako su ovi generalno uspješni, procedure za njihovo korištenje još uvijek moraju biti razrađene do stepena kada se menadžeri akvizicije osjećaju potpuno ugodno prilikom njihovog korištenja (SNSCT, 2010).

Oslanjajući se na stručnost u procjeni rizika, spiralni model umnogome se oslanja na sposobnost programera softvera da identifikuju i upravljaju izvorima rizika projekta. Dobar primjer za navedeno jeste specifikacija spiralnog modela vođena rizikom, koja prenosi elemente visokog rizika do velikog broja detalja i ostavlja elemente niskog rizika da budu razrađeni u kasnijim fazama; do tog vremena, manji je rizik od loma (SNSCT, 2010).

Međutim, tim neiskusnih ili loših programera također može proizvesti specifikaciju sa drugačijim uzorkom varijacije u nivoima detalja: velika razrada detalja za dobro shvaćene elemente niskog rizika i mala razrada loše shvaćenih, visokorizičnih elemenata. Osim ako ne postoji pronicljiv pregled takvih specifikacija od strane iskusnog osoblja za arzvoj ili nabavku, ova vrsta projekta daće iluziju napretka tokom perioda u kome se zapravo dešava katastrofa (Boehm, 2005).

2) Specifikacija vođena rizikom također će ovisiti o ljudima. Na primjer, dizajn koji je radio stručnjak mogu implementirati oni koji nisu stručnjaci. U ovom slučaju vještak, kome nije potrebna velika detaljna dokumentacija, mora izraditi dovoljno dodatne dokumentacije da nestručnjaci ne zalutaju. Recenzenti specifikacije također moraju biti osjetljivi na ove nedoumice (Boehm, 2005).

Uz konvencionalni pristup zasnovan na dokumentima, zahtjev za nošenjem svih aspekata specifikacije do ujednačenog nivoa detalja eliminiše neki potencijal problema i omogućava adekvatan pregled nekih aspekata od strane neiskusnih recenzenata. Ovdje se također stvara veliki odliv vremena oskudnih stručnjaka, koji moraju kopati za kritičnim pitanjima unutar velike mase nekritičnih detalja. Nadalje, ako elementi visokog rizika bacaju sjenu kroz impresivne reference na loše razumljive mogućnosti, postoji još veći rizik da će konvencionalni pristup stvoriti iluziju napretka situacije koja zapravo ide ka katastrofi (SNSCT, 2010).

Pored navedenih poteškoća, postoji i potreba za daljnjom razradom koraka spiralnog modela. Generalno, potrebno je dodatno razraditi korake procesa spiralnog modela kako bi se osiguralo da svi učesnici u razvoju softvera funkcionišu konzistentno. Neki od primjera ovoga su potreba za detaljnijim definicijama prirode specifikacije spiralnog modela i prekretnice, prirode i ciljeva pregleda spiralnog modela, tehnike za procjeni i sinhronizaciju rasporeda i prirode spirale indikatora statusa modela te procedura praćenja troškova u odnosu na napredak. Druga potreba je za smjernice i kontrolne liste za identifikaciju najvjerovatnijih izvora rizika projekta te najefikasnijih tehnika rješavanja rizika za svaki izvor rizika (SNSCT, 2010).

Iz prikazanih podataka možemo izvući četiri zaključka:

(1) Priroda spiralnog modela vođenog rizikom prilagodljivija je cijelom rasponu softverskih projektnih situacija nego pristupi prvenstveno vođeni dokumentima kao što je model Waterfall ili pristupi primarno vođeni kodom kao što je evolucijski razvoj. Posebno je primjenjiv na veoma velike, složene i ambiciozne sisteme (SNSCT, 2010).

(2) Spiralni model bio je prilično uspješan u svojoj najvećoj primjeni do sada: razvoju i unapređenju TRW-SPS. Sve u svemu, ovaj model je postigao visok nivo mogućnosti okruženja softverske podrške u vrlo kratkom vremenu te je pružio fleksibilnost koja je neophodna za prilagođavanje visokog dinamičkog raspona tehničke alternative i ciljeva korisnika (Boehm, 2005).

(3) Spiralni model još nije u potpunosti razrađen kao etablirani model. To je razlog zbog koga ga može primijeniti iskusno osoblje, ali navedeno iziskuje dalju razradu u oblastima kao što su ugovaranje, specifikacije, prekretnice, pregledi, zakazivanje, praćenje statusa i identifikacija rizičnih područja kako bi bili u potpunosti upotrebljivi u svim situacijama. isto

(4) Djelimične implementacija spiralnog modela, kao što je upravljanje planom rizika, kompatibilne su sa najsavremenijim modelima procesa te su od velike pomoći za prevazilaženje glavnih izvora projektnog rizika (Boehm, 2005).

2.1.2. Agilne metodologije razvoja softvera

Postoji nekoliko dostupnih metoda agilnog razvoja softvera. Iako svaki od njih ima jedinstven pristup, oni dijele vrijednosti i vizije koje opisuje agilni manifest. Svi oni uključuju trajnu komunikaciju, planiranje, testiranje i integracije. Unatoč tome što pomažu pri razvoju softvera, ono što definiše agilne metode je poticanje saradnje koje čini zajedničke odluke dobrim i brzim (Stoica, Mircea, i Ghilic-Micu, 2013).

Neke od agilnih metoda razvoja softvera su: Adaptivni softver razvoj (eng. ASD - Adaptive Software Development), Razvoj vođen funkcijama (eng. Function Determined Development), Crystal Clear, Dinamična metoda razvoja softvera (eng. DSDM - Dynamic

Software Development Method), Brzi razvoj aplikacija (eng. RDD - Rapid Applications Development), SCRUM, Ekstremno programiranje (XP), i Rational Unify Process (RUP) (Stoica, Mircea, i Ghilic-Micu, 2013).

Sa vremenom su se na bazi nekoliko metoda zasnovanih na nekim od ovih vrijednosti pojavili i principi. Ovi principi bili su označeni kao agilni, a neki od njih su: (1) Ekstremno programiranje (XP), SCRUM, Agilni objedinjeni proces (AUP), Agilno modeliranje, Osnovni objedinjeni proces (EssUP), Otvoreni objedinjeni proces (OpenUP), Metoda dinamičkog razvoja sistema (DSDM), Razvoj vođen karakteristikama (FDD), Crystal itd (Awad, 2005).

Agilna grupa metoda zasniva se na iterativnom i inkrementalnom razvoju, gdje specifikacije i rješenja dolaze iz individualne saradnje među organizovanim timovima, ali sa zajedničkim ciljem. Ove metode zasnivaju se na principu 12, koji je sintetizovan u tzv. "Agilnom manifestu" koji je objavljen 2001. godine: (a) zadovoljstvo klijenta kroz brzu isporuku upotrebljivog softvera; (b) ispunjavanje specifikacija, čak i ako se dešava kasno u razvoju; (c) česta isporuka upotrebljivog softvera (sedmično); (d) upotrebljivi softver je glavna mjera napretka; (e) održivi razvoj, sposoban da održi stabilan ritam; (f) saradnja između programera i klijenata; (g) saradnja licem u lice kao najbolji način komunikacije; (h) projekti koje grade motivisane i kredibilne osobe; (i) jednostavnost; (j) individualno organizovani timovi; (k) prilagođavanje promjenjivim okolnostima; (l) stalna pažnja na odličnoj tehnici i dobrom dizajnu (Awad, 2005).

2.1.2.1. SCRUM

SCRUM predstavlja iterativnu i inkrementalnu metodu čija je svrha da pomogne razvojnim timovima u koncentrisanju na postavljene ciljeve te minimiziranju obavljanja manje važnih zadataka pri radu. SCRUM cilja na zadržavanje jednostavnosti u složenom poslovnom okruženju. Termin dolazi iz ragbija, u kome označava strategiju vraćanja izgubljene lopte u igru timskim radom. SCRUM ne obezbjeđuje nivo implementacije tehnike; fokusira se na način na koji članovi razvojnog tima treba da komuniciraju kako bi stvorili fleksibilan, prilagodljiv i produktivan sistem koji stalno mijenja okruženje. SCRUM je zasnovan na dva elementa: timskoj autonomiji i prilagodljivosti. Timska autonomija znači da vođe projekta utvrđuju zadatke koje tim mora ispuniti, ali tim u svakoj iteraciji slobodno odlučuje kako će raditi, sve sa ciljem povećanja produktivnosti tima. SCRUM ne predlaže poseban softver za razvojne tehnike, ali koristi metodu pijeska instrumenata pri upravljanju raznim fazama, kako bi se izbjegla konfuzija koju stvaraju složenost projekta i nepredvidivost (Stoica *et.al,* 2013).

SCRUM je opisan kao "okvir unutar koga možete koristiti različite procese i tehnike", a ne proces ili tehnika za izgradnju proizvoda. SCRUM okvir prvenstveno je baziran na timu i definira povezane uloge, događaje, artefakte i pravila. Tri primarne uloge u okviru SCRUM-a su: (1) Vlasnik proizvoda koji predstavlja zainteresovane strane; (2) SCRUM

master koji upravlja timom i SCRUM procesom; (3) Tim, oko 7 ljudi, koji razvijaju softver (Cervone, 2011).

Zahtjevi koji čine osnovu projekta objedinjuju se u ono što se naziva "zaostatak projekta", koji se redovno ažurira. Karakteristike koje su povezane sa ovim zahtjevima nazivaju se korisničke priče. Rad je vremenski upakovan u seriju ciklusa od 1 do 4 nedjelje gdje su posao i projekat timske procjene o tome koje su korisničke priče u opadajućem redoslijedu prioriteta ostvarive u svakom ciklusu. Ovaj podskup korisničkih priča iz zaostatka projekta čini osnovu zaostatka iteracije planiranog za isporuku u periodu od dvije sedmice (Cervone, 2011).

U okviru SCRUM-a postoje 3 sastanka sa vremenskim okvirom (ili sa fiksnim trajanjem) koji se održavaju tokom iteracije, plus dnevni stand-up sastanak za tim, SCRUM majstora i (idealno) vlasnika proizvoda. Na početku sprinta, o funkcijama koje će se razviti tokom sprinta odlučuje se na sastanku planiranja sprinta. Na kraju iteracije održavaju se još dva sstanka - pregled iteracije i retrospektiva iteracije, gdje tim pregleda proizvod i demonstrira upotrebu softvera, kao i razmatranje i poboljšanje iteracije samog procesa. isto Nakon što je sprint završen, sljedeći set korisničkih priča bira se iz zaostatka projekta i čitav proces počinje ponovo. Stopa spaljivanja prati se kako bi se odredilo kada će finansiranje biti iscrpljeno (Schwaber i Sutherland, 2020).

SCRUM koncepti, dakle, su sljedeći (Schwaber, 2014):

1. Projekat - diskretan skup zahtjeva krajnjeg korisnika koji su grupisani, prioritetni i finansirani;

2. Zahtjev - izjava krajnjeg korisnika koja ocrtava njegovu potrebu za informacijama;

3. Sprint - predstavlja događaj koji traje od 1 do 4 sedmice te je fokusiran na isporuku podskupa korisnika;

4. Priče - preuzete iz zaostatka projekta;

5. Zaostatak projekta - predstavlja trenutnu listu korisničkih priča za projekat. Korisničke priče mogu biti dodane, izmijenjen ili uklonjene iz zaostatka tokom projekta;

6. Sprint Backlog - podskup korisničkih priča iz projekta zaostaci koji se planiraju isporučiti kao dio sprinta;

7. Korisničke priče - korisnička priča je jedan ili dva reda opisa poslovnih potreba, obično opisana u smislu karakteristika;

8. Zadaci - zadaci su aktivnosti koje se obavljaju da bi se ispunila korisnička priča;

9. Tehnički dug - ovo se odnosi na stavke koje su bile ili na izostanke sa sastanaka za planiranje ili na odlaganja u korist prijevremene isporuke. isto

2.1.2.2. Historija razvoja SCRUM-a

Evolucija SCRUM.-a ilustrovana je kroz promjene i poboljšanja u načinu na koji je SCRUM definisan u različitim, utvrđenim izvorima. Za svaki izvor su opisane tri iste teme, pokazujući od čega se SCRUM sastojao u vrijeme "definicije" SCRUM-a: (1) Uloge, odgovornosti; (2) Kontrole, rezultati, artefakti; (3) Faze, sastanci, vremenski okviri i događaji (Schwaber & Sutherland, 2020).

O SCRUM-u je prvi raspravljao Ken Schwaber 1995. godine na OOPSLA1 događaju u radionici "Dizajn poslovnog objekta i implementacija" sa Jeffom Sutherlandom kao članom panela za taj događaj. Scrum se definiše kao da se sastoji od sljedećih glavnih komponenti: (1) Projektni tim - u njega spadaju mendažment i razvojni tim; (2) Backlog - u njega spadaju sljedećim redom Release/poboljšanja > paketi > izmjene > problemi > rizici > rješenja > problemi (3) tzv. "Pregame" (planiranje i dizajn) > Game (Razvojni sprintovi) > Postgame (kraj, završetak) (Schwaber, 2014).

Stoga će se u ovom poglavlju kratka historija SCRUM-a opisati kroz prezentaciju nekoliko radova o SCRUM-u koje je Schwaber izdao kroz godine te načina na koji je njihovo objavljivanje uticalo na viđenje i definiciju SCRUM-a. Prvi Schwaberov rad nije tačno iz 1995., ali je njegov sadržaj kodifikovan/prezentovan 1995. Ken Scwaber navodi da je SCRUM prvi put predstavio na OOPSLA 96. Ovdje je važno naglasiti da SCRUM nije akronim, niti je ikada bio (Schwaber i Sutherland, 2020).

SCRUM je opisan kao metodologija koja je fundamentalno drugačija od postojećih pristupa, i to na osnovu sljedećih karakteristika: (1) menadžment vodi menadžer proizvoda; (2) razvojni timovi su mali, međufunkcionalni timovi od tri do šest ljudi; (3) faza sprintova je empirijski proces; (4) konačnost rezultata faze sprintova je eksplicitno ostavljena otvorena (Schwaber i Sutherland, 2020).

Autori su također razradili tri obrasca SCRUMA: (1) SCRUM Meeting - dnevni sastanak od 15 minuta (u isto vrijeme i mjesto) za članove tima; (2) Sprint - period od približno 30 dana do stvaranja vidljive i upotrebljive isporuke; (3) Backlog - dinamična, prioritetna lista poslova koje treba izvesti na proizvodu. Također se uvode neki važni koncepti ili ideje: (1) Scrum Master - odgovorna uloga vođe tima za rad na sječi, pomaganje članovima tima i rješavanje blokova; (2) Scrum Tim - samoorganizovana struktura tima za proizvođenje korisnog rezultata u sprintu, neometani vanjskim zahtjevima; (3) Demo - sesija na kraju svakog Sprinta kojom se demonstrira nova funkcionalnost i pravi napredak (smanjenje zaostatka). Njega prati sesija "pregleda" za promjenu planiranja za budućnost (Schwaber, 2004).

U cijelom opisu Scrum obrasca izražene su sljedeće ideje: (a) postoji razlika u "globalnom" zaostatku odabranom za sprint, tj. razložen je na zadatke; (b) tim bira onaj dio zaostatka za koji se vrjeruje da se može završiti unutar iteracije sprinta i obavezuje se na to; (c) rad treba rezultirati kohezivnim osjećajem da je cilj Spirnta postignut kada se rad završi; (d)

prikazani vidljivi i upotrebljivi rezultati u Demo-u nazivaju se inkrementima; (e) menadžer proizvoda ili marketing menadžer spominje se kao "jedina osoba koja odlučuje o prioritetima zaostalih predmeta" (Schwaber, 2014).

Na SCRUM sastanku treba odgovoriti na tri pitanja: da li je završen? koji su blokovi? šta je sljedeće? Opšta svrha SCRUM sastanka jasno je opisana kao pružanje adaptivnog mehanizma "ekvivalentu termometra koji uzrokuje temperaturu tima". Nekoliko koncepata, čak i bez njihovog korištenja konačnog imenovanja, kasnije su priznati kao "značajan doprinos" oblikovanju SCRUM-a u njegovih prvih pet godina (Schwaber, 2014).

Prvu knjigu o SCRUM-u napisala su dva potpisnika "Agilnog manifesta" (2001). Ovdje je SCRUM definisan kao da se sastoji od sljedećeg: (1) Scrum Master + Vlasnik proizvoda + Scrum Tim; (2) Zaostatak proizvoda (>Release Backlog) + Sprint Zaostatak + Povećanje proizvoda; (3) Planiranje sprinta + Sprint + Dnevni Scrum + Sprint Pregled. Faze "pregame" i "postgame" ovdje nisu bile prisutne. SCRUM je tako obuhvatao samo sprintove. Po opisu složenosti, SCRUM je predstavljen kao zagovaranje nove paradigme koja napreduje u empirijskom upravljanju i samoorganizaciji (Schwaber, 2014).

Šest karakteristika "The New Product Development Game" rada Takeuchia i Nonaka (1986) spominju se kao fundamentalne za "Ispravnu implementaciju SCRUM-a". Preporučena veličina tima promijenjena je na sedam ljudi, plus ili minus dva. Na temu tima u kompoziciji se navodi da "SCRUM izbjegava vertikalne SCRUM timove".

U ovom radu neformalne prakse dobile su formalne nazive, dok su drugi nazivi preimenovani: (a) "Vlasnik proizvoda" dodaje se kao uloga; (b) Blokovi se sada nazivaju "prepreke"; (c) "Scrum Meeting" dobio je ime "Daily Scrum"; (d) Cilj sprinta sada se zove "Sprint Goal" uz dodavanje da Sprint Backlog po definiciji mora ispuniti cilj Sprita; (e) "Demo/review" mijenja se u "Sprint Review". Normalan prekid Sprinta moguć je ako ipak ima smisla nastaviti duže otkazivanje Sprinta. Scrum Master opisan je kao nova menadžment uloga uvedena u SCRUM (Schwaber, 2014).

Nakon uspostavljanja ScrumAlliance i Certified ScrumMaster kursa, Ken Schwaber obhavio je 2004. drugu knjigu o SCRUM-u. U ovoj knjizi SCRUM se definiše kao da se sastoji od: (1) Vlasnika proizvoda + Tima + ScrumMastera; (2) Zaostatka proizvoda (<vizija) + Sprint Backlog + Povećanje (funkcionalnosti proizvoda); (3) Planiranje sprinta + Sprint + Dnevni Scrum + Sprint Recenzija + Sprint retrospektiva.isto. Pored promjena u definiciji SCRUM-a, ova druga knjiga sadrži mnoge studije slučaja. sve ove knjige schwabera predstaviti kao faze razvoja scruma Treća knjiga Kena Schwabera teži opisivanju razloga za i načina za usvajanje Scrum-a, izazova i očekivanog uticaja. U ovoj je knjizi, objavljenoj 2007. godine, definicija SCRUM-a ostala ista onoj u prethodnoj (Schwaber, 2014).

Posljednji SCRUM vodič Kena Schwabera izašao je 2020. godine, te ga se smatra posljednjom od etapa razvoja SCRUM-a. U ovom vodiču SCRUM je definisan kao da se

sastoji od: (1) SCRUM tima = Vlasnik proizvoda + Programeri + SCRUM Master; (2) Sprint - Planiranje Spirnta + Dnevni Scrum + Recenzija Sprinta + Sprint retrospektiva; (3) Zaostatak proizvoda (<Cilj proizvoda) + Sprint Zaostatak (<Sprint cilj) + povećanje (=Gotovo).isto. Iako SCRUM vodič još uvijek nema vizualni prikaz SCRUM-a, jedina promjena koja utječe na prethodne slike jeste eksplicitna potvrda da Sprint može isporučiti višestruke inkremente, dakle više od običnog povećanja na kraju sprinta (Schwaber & Sutherland, 2020).

Historija SCRUM-a može se ukratko sumirati na sljedeći način. SCRUM je agilna metodologija razvoja softvera koja je nastala kao odgovor na potrebu za efikasnijim i fleksibilnijim pristupom razvoju projekata. Njegova priča datira unatrag u 1980-e godine kada je jezički stručnjak Jeff Sutherland počeo razmišljati o načinima kako poboljšati procese razvoja softvera (Hossain *et.al.,* 2009).

Sutherland je prvi put formalno predstavio SCRUM kao metodologiju 1995. godine zajedno s Kenom Schwaberom, stručnjakom za upravljanje projektima. Naziv "SCRUM" preuzet je iz ragbija, gdje označava situaciju u kojoj se igrači udružuju kako bi se natjecali za loptu. Analogija je bila jasna: timski rad, suradnja i prilagodljivost ključni su za uspješno upravljanje projektima, slično kao u ragbiju (Abdur *et.al.,* 2017).

SCRUM je prvi put formalno opisan u radu "SCRUM Development Process" objavljenom 2001. godine. Od tada se metoda kontinuirano razvijala i prilagođavala, kako bi se bolje odgovorilo na potrebe i izazove modernog razvoja softvera.

Osnovna načela SCRUM-a uključuju inkrementalni i iterativni pristup razvoju, česte inspekcije i prilagodbe, te fokus na transparentnost i timsku suradnju. SCRUM koristi okvir određenih uloga (Scrum Master, Product Owner, Development Team) i događaja (Sprint, Daily Scrum, Sprint Review, Sprint Retrospective) kako bi organizirao i vođenje projekta (Schwaber i Sutherland, 2020).

SCRUM je brzo postao popularan ne samo u razvoju softvera, već i u drugim industrijama koje su prepoznale vrijednost agilnog pristupa. Danas je SCRUM jedna od najpoznatijih i najrasprostranjenijih agilnih metodologija, pružajući organizacijama alate za efikasno upravljanje projektima, bolju komunikaciju i isporuku vrijednih proizvoda.

U zaključku, SCRUM je proizašao iz potrebe za poboljšanjem procesa razvoja softvera i postao je ključan alat za agilno upravljanje projektima. Njegova bogata povijest i evolucija svjedoče o trajnom utjecaju koji je imao na način na koji organizacije pristupaju razvoju i isporuci proizvoda.

2.1.2.3. Uloga SCRUM-a

Tri su ključne uloge definisane u SCRUM razvojnom pristupu: samoorganizovani razvojni SCRUM tim, Scrum Master i vlasnik proizvoda (Schwaber i Beedle, 2002). Vlasnik

proizvoda predstavlja interese eksternih zainteresovanih strana (kupac, korisnici, menadžment prooizvoda) i tako on predstavlja primarni interfejs između ovih zainteresovanih strana i tima za razvoj softvera (4). SCRUM tim odgovoran je za stvarni razvoj softvera. Daljnja uloga je mendažer proizvoda, koji definiše "početni sadržaj i vrijeme objavljivanja, zatim upravlja njihovom evolucijom kako projekat napreduje i varijable se mijenjaju, te se bavi zaostatkom, rizikom i sadržajem izdanja" (Schwaber i Sutherland, 2020).

SCRUM Master je odgovoran za olakšavanje procesa razvoja, osiguravanje da tim koristi čitav niz odgovarajućih agilnih vrijednosti, praksi i pravila. SCRUM master održava dnevne koordinacione sastanke i otklanja sve prepreke timskih susreta. Šest aktivnosti Scrum mastera definisano je u širokom distribuiranom organizacionom kontekstu: sidro procesa, fasilitator ustajanja, otkalnjanje prepreka, planer sprinta, fasilitator scruma i integracijsko sidro6. Sidro procesa njeguje pridržavanje agilnim metodama. Facilitator ustajanja osigurava da članovi tima dijele informacije o statusu i smetnji tokom svakog sprinta. Uklanjanje prepreka osigurava programerima da napreduju u svom radu. Sprint planer podržava trijažu korisničke priče i planiranja radnog opterećenja koje se dešava prije razvojnog rada koji počinje u svakom sprintu. Scrum koordinatori rade sa ostalim Scrum majstorima u razvojnom programu. Integraciono sidro olakšava spajanje kodnih baza koje su razvili saradnički timovi koji rade paralelno (Abdur *et.al.,* 2017).

Prema Schwaberovim i Sutherlandovim Scrum smjernicama, Scrum master je "sluga-vođa za Scrum tim. Scrum Master pomaže onima izvan Scrum Tima da razumiju koje su njihove interkacija sa Scrum timom korisne, a koje nisu. Scrum Master pomaže svima da promijene ove interakcije kako bi makskimizirali vrijednost koju stvara Scrum tim" (Schwaber & Sutherland, The Definitive Guide to Scrum: The Rules of the Game, 2020). Prema tome, Scrum Master služi razvojnom timu. Ovo je u suprotnosti sa vlasnikom proizvoda, koji je odgovoran za maskimiziranje vrijednosti proizvoda i rada Scrum tima. Schwaber i Sutherland (2020) navode da iako postoji velika fleksibilnost u načinu na koji se to postiže, vlasnik proizvoda je jedina osoba odgovorna za upravljanje zaostatkom proizvoda.

Prema Schwaber i Sutherlandu (2020), zadaci upravljanja zaostatkom proizvoda uključuju: (1) naručivanje stavki u zaostatku proizvoda kako bi se na najbolji način postigli ciljevi i misije; (2) optimiziranje vrijednosti posla koji razvojni tim obavlja; (3) osiguravanje vidljivosti Product Backloga, te transparentnosti i jasnosti svima, uz pokazivanje onoga što će SCRUM tim i dalje raditi; (4) osiguravanje da razvojni tim razumije stavke u zaostatku proizvoda na potrebnom nivou.

Dokazi iz prakse pokazuju da se uloga SCRUM Mastera razvija. Na primjer, ova uloga se ponekad dijeli, dok su aktivnosti koje obavlja Scrum Master različite i donekle drugačije od prvobitne vizije. Ovo su uočili Gupta i Reddy (2016), koji su otkrili da su izazovi prilagođavanju Scruma u globalnom timu pomaganje više osoba koje dijele uloga Scrum mastera i vlasnika proizvoda. Gupta i ostali razvili su novu taksonomiju Scrum Mastera u

kojoj su kreirane tri nove uloge kako bi se održala složenost uključena u upravljanje globalnim razvojem softverskog tima.

Prema ISO/IEC/IEEE standardu o korisničkoj dokumentaciji u agilnim softverima, Scrum Master i menadžer proizvoda imaju slične odgovornosti kada su u pitanju objašnjavanje promjena ili novih zahtjeva. Scrum master i voditelj razvoja informacija ili menadžer projekta bi trebali dati smjernice tehničkim piscima i drugim članovima agilnih razvojnih timova kako se nositi s promjenjivim i novim zahtjevima. Možda je ovo miješanje uloga uglavnom zbog toga što organizacije pretvaraju tradicionalnu ulogu menadžera projetka u ulogu majstora Scrum-a (Abdur *et.al.,* 2017).

Prilagođavanje uloga Scrum-a i stvaranje novih uloga za upravljanje projektima veclikih razmjera uočeno je u drugim studijama, gdje je stvorena uloga "Area Product Owner" (APO); ovu ulogu dijelile su dvije osobe: arhitetka sistema i predstavnik menadžera proizvoda. Arhitekata sistema blisko je sarađivao s timom, dok predstavnik menadžera proizvoda nije imao direktnu interakciju sa timovima (Paasivaara & Lassenius, 2011). Ova kombinovana uloga (podijeljena između dvije osobe) dobro je funkcionisala za ovu organizaciju i prijavljena je kao jedan od uspjeha projekta. Ipak, u kasnijoj studiji isti autori primijetili su da su menadžeri imali dvostruku ulogu: ulogu Scrum Mastera i tradicionalnih dužnosti linijskog menadžmenta koji su uključivali kadrvoska pitanja kao što je evaluacija učinka. Prekomjerna upotreba uloge Scrum Mastera, koji je djelovao kao predstavnik tima sa zajedničkim sastancima, bila je problematična. Tim je smatrao da su ovi sastanci bili gubljenje vremena, te bi slao Scrum mastera da obavlja ove zadatke. Česti sastanci u SCRUM-u također su predstavljali problem (Stray *et.al.,* 2013). Uloga Scrum mastera je u fasilitiranju dnevnih koordinacionih sastanaka, gdje se koordinacioni sastanci koriste za komuniciranje statusa razvojnog rada unutar tima i vlasnika proizvoda. Međutim, efikasnost dnevnih koordinacionih sastanaka često je kompromitovala previše zainteresovanih strana, jer su se sastanci održavali prečesto da bi bili od koristi za učesnike (Abdur *et.al.,* 2017).

Narušavanje pažljive ravnoteže između Scrum uloga dovodi do drugih problema. Na primjer (Moe *et.al.,* 2008) primijetili su da Scrum masteri također vrše procjene, ali ne uključuju sav tim u raspravu o zadatku. Navedeno je dovelo do toga da programeri rade sami, što stvara siromašnu koheziju tima i probleme koji se pojavljuju na kraju sprinta, a ne na početku. Rečeno je da nedostatak temeljne rasprave umanjuje valjanost zajedničkog zaostatka "tako da se programeri više fokusiraju na sopstveni plan. Pošto je planiranje imalo slabosti i niko od programera nije smatrao da ima potpuni pregled, ovo je vjerovatno bio jedan od razloga za kasnije otkrivene probleme u dizajnu".

Ipak, u nedavnoj anketi koja je ispitivala da li menadžeri projekata i dalje postoje u agilnim razvojnim timovima, postoje rezultati prema kojima je 67% anketiranih organizacija izjavilo da i dalje ima ulogu menadžera projekta. Ovi autori pozvali su na više istraživanja o uzrocima zbog kojih su projekt menadžeri i dalje prisutni na projektima agilnog razvoja softvera i kako se njihova uloga možda promijenila. Prema konvencionalnoj mudrosti,

projektni menadžeri koriste komandni i kontrolni stil upravljanja, dok se Scrum majstori fokusiraju na vođenje i testiranje. Kao takvi, Scrum majstori nisu linijski menadžeri za članove sprinterskog tima. Nadalje, Scrum majstori ne dodjeljuju radne predmete članovima svog tima, jer se timovi samoorganizuju (Abdur *et.al.,* 2017)

Ukratko, u literaturi postoji tema koja se redovno pojavljuje, a to je da je original balans Scrum mastera, vlasnika proizvoda te da se uloga tima prilagođava i spaja kako bi odgovarala potrebama organizacija koje prelaze iz Waterfall modela na Scrum, ili kako bi se Scrum skalirao na velike razmjere (Abdur *et.al.,* 2017).

2.1.2.4. SCRUM artefakti

Scrum-ovi artefakti predstavljaju rad ili vrijednost. Dizajnirani su da maksimiziraju transparentnost ključa informacija. Dakle, svi koji ih pregledaju imaju istu osnovu za prilagođavanje. Svaki artefakt sadrži obavezu da osigura informacije koje poboljšavaju transparentnost i fokus prema kome se napredak može mjeriti: (a) Za zaostatak proizvoda to je cilj proizvoda; (b) Za zaostatak sprinta to je cilj sprinta; (c) Za inkrement to je Definicija Gotovog. Ove obaveze postoje kako bi ojačale empirizam i Scrum vrijednosti za Scrum tim i njegove zainteresovane strane (Schwaber & Sutherland, 2020).

Product Backlog predstavlja novu, uređenu listu onoga što je potrebno za poboljšanje proizvoda. To je jedan izvor rada koji je preduzeo Scrum tim. Stavke zaostatka proizvoda koje Scrum tim može uraditi u okviru jednog Sprinta smatraju se spremnim za izbor kada se vrši planiranje sprinta. One obično dobivaju ovaj stepen transparentnosti nakon rafiniranja aktivnosti. Prečišćavanje zaostatka proizvoda je čin razvijanja i daljeg definiranja zaostatka proizvoda u stavke manje preciznijih predmeta. Ovo predstavlja stalnu aktivnost za dodavanje detalja, kao što su opis, red i veličina. Atributi se često razlikuju u zavisnosti od domena posla. Za dimenzioniranje su odgovorni programeri koji će raditi posao. Vlasnik proizvoda može utjecati na programere pomažući im da shvate i izaberu kompromise (Schwaber i Sutherland, 2020).

Ovdje je važno naglasiti i pojam cilja proizvoda, koji opsiuje buduće stanje proizvoda koje može poslužiti kao cilj za planiranje Scrum tima. Cilj proizvoda je u zaostatku proizvoda. Ostatak zaostatka proizvoda pojavljuje se u definicija toga "šta" će ispuniti cilj proizvoda. Proizvod je sredstvo za isporuku vrijednosti. Ima jasne granice, poznate zainteresovane strane, dobro definisane korisnike ili kupce. Proizvod može biti usluga, fizički proizvod ili nešto apstraktnije. Cilj proizvoda predstavlja dugoročni cilj za Scrum tim. Potrebno je da Scrum tim ispuni (ili napusti) jedan cilj prije preuzimanja sljedećeg (Schwaber i Sutherland, 2020).

Što se tiče Spring Backloga, on se sastoji od Sprint cilja (zašto), skupa odabranih stavki zaostatka proizvoda Sprinta (šta), te akcijskog plana za isporuku inkrementa (kako). Sprint Backlog je plan za programere. To je vrlo vidljiva slika rada u realnom vremenu koje

preograмeri planiraju da ostvare tokom Sprinta kako bi postigli cilj Sprinta. Shodno tome, Sprint Backlog se ažurira tokom Sprinta kako se više uči. Također je potrebno više detalja kako bi se provjerio napredak u Daily Scrum-u (Schwaber, 2014).

Postoji i Sprint Goal, koji predstavlja jedini cilj za Sprint. Iako je cilj Spinta obaveza za programere, on pruža fleksibilnost u smislu tačnog posla potrebnog da se to postigne. Sprint Goal takođe stvara koherentnost i fokus, ohrabrujući Scrum tim da radi zajedničke, a ne odvojene inicijative. Cilj sprinta kreira se tijekom događaja planiranja Sprinta, što ima na umu Sprint cilj. Ako se ispostavi da je posao drugačiji nego što su očekivali, oni sarađuju sa vlasnikom proizvoda kako bi pregovarali o obimu Zaostatka sprinta unutar sprinta bez utjecaja na njegov cilj (Schwaber, 2004).

Povećanje ili inkrement predstavlja odskočnu dasku ka cilju proizvoda. Svaki inkrement je aditivan za sve prethodne inkremente i temeljno je verifikovan, što odgovara tome da svi inrekmenti rade zajedno. Kako bi pružio vrijednost, inkrement mora biti upotrebljiv12. Moguće je kreirati višestruke inkremente unutar Sprinta. Zbir povećanja je prikazan na Spirnt Reviewu na način na koji podržava empirizam. Međutim, inkrement se može dostaviti zainteresovanim stranama prije kraja Sprinta. Sprint Review se nikada ne bi trebao smatrati kapijom za oslobađanje vrijednosti. Rad se ne može smatrati dijelom inkrementa osim ako ne ispunjava definiciju završenog (Schwaber i Sutherland, 2020).

Ovdje je potrebno ukratko definisati i Definiciju Gotovog. Definicija Gotovog predstavlja formalni opis stanja prirasta kada on ispunjava kvalitet mjere potrebne za proizvod. U trenutku kada stavka Product Backlog ispuni definiciju Gotovog, rađa se inkrement. Definicija Gotovog stvara transparentnost pružajući svima zajedničko razumijevanje o tome koji su radovi završeni u sklopu Inkrementa. Ako stavka zaostatka proizvoda ne zadovoljava definiciju Gotovog, ne može biti objavljena niti predstavljena na Sprint Review. Umjesto toga, ona se vraća u Zaostatak Proizvoda za buduće razmatranje (Schwaber i Sutherland, 2020).

Ako je Definicija Gotovog za povećanje dijela standarda organizacije, svi Scrum timovi moraju ga slijediti kao minimum. Ako to nije organizacijski standard, Scrum tim mora kreirati Definiciju Gotovog koja je prikladna za proizvod. Od programera se traži da se pridržavaju Definicije Gotovog. Ako postoji više Scrum timova koji rade zajedno na proizvodu, oni se moraju međusobno definisati i biti u skladu sa istom Definicijom Gotovog (Schwaber, 2014).

2.1.2.5. SCRUM procesi

Scrum procesi koriste se za rješavanje složenih problema, tačnije onih problema koji se ponašaju nepredvidivo. Ne samo da su ovi problemi nepredvidivi, već je i načine na koje se pokazuju nemoguće predvidjeti. Drugim riječima, statistički uzorak rada ovih procesa nikada neće dati smislen uvid u njihov temeljni matematički model, te se pokušaj stvaranja

uzorka može napraviti isključivo sumiranjem njihovog rada do takvog stepena grubosti da bude irelevantan za one koji pokušavaju razumjeti ove procese ili upravljati njima. Veliki dio našeg društva zasniva se na procesima koji funkcionišu samo zato što je njihov stepen nepreciznosti prihvatljiv (Schwaber, 2004).

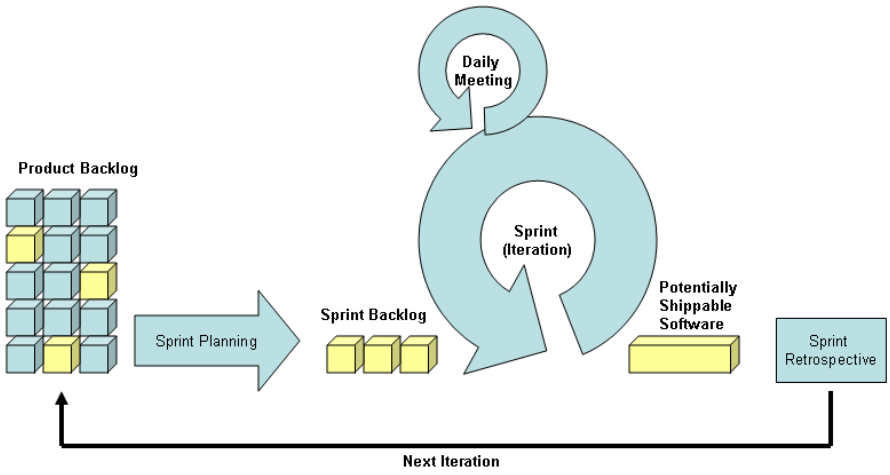
U slučajevima kada posotoji potreba da se poveća preciznost nekog procesa, isti je potrebno voditi korak po korak, osiguravajući da se on odvija na prihvatljivom stepenu preciznosti. U slučajevima kada do konvergencije ne dođe, potrebno je izvršiti prilagodbe kako bi se proces vratio u raspon prihvatljivih nivoa preciznosti. Postavljanje procesa koji će opetovano proizvoditi prihvatljiv kvalitet izlaza naziva se definisana kontrola procesa. Kada se definisana kontrola procea ne može postići zbog složenosti međuaktivnosti, potrebno je primijeniti nešto što se naziva empirijskom kontrolom procesa. isot

Definisani procesi koriste se kad je god navedeno moguće, budući da se njima može pokretnuti proizvodnja bez nadzora, sve do količine u kojoj je moguće proizvodnju ocijeniti kao robu. Ipak, ako je roba tako neprihvatljivog kvaliteta da je neupotrebljiva, ili je njena prerada prevelika da bi cijena bila prihvatljiva, ili je cijena neprihvatljivo niskih prinosa previsoka, potrebno je okrenuti se i prihvatiti veće troškove empirijske kontrole procesa. Dugoročno gledano, ispostavilo se da je pravljenje uspješnih proizvoda prvi put mnogo jeftinije putem korištenja empirijske kontrole procesa nego prerada neuspješnih proizvoda korištenjem definirane kontrole procesa. Postoje tri noge koje drže svaku implementaciju empirijske kontrole procesa: vidljivost, inspekcija i adaptacija. Vidljivost znači da oni aspekti procesa koji utiču na ishod moraju biti vidljivi onima koji kontrolišu proces. Ne samo da ti aspekti moraju biti vidljivi, već ono što je vidljivo također mora biti istinito. U empirijskoj kontroli procesa nema mjesta zavaravanju izgleda (Schwaber, 2004).

Druga etapa je etapa inspekcije. Različite aspekte procesa potrebno je dovoljno često provjeravati kako bi se otrkile neprihvatljive varijacije u procesu. Učestalost inspekcija mora uzeti u obzir i činjenicu da se procesi mijenjaju samim činom inspekcije. Zanimljivo je da potrebna učestalost pregleda često premašuje toleranciju inspekcija procesa. Srećom, to obično nije tačno u razvoju softvera. Drugi faktor u inspekciji jeste inspektor, koji mora posjedovati vještine da procijeni predmet inspekcije. Treći dio empirijske kontrole procesa jeste adaptacija. Ako inspektor inspekcijom utvrdi da je jedan ili više aspekata procesa izvan prihvatljivih granica te da će dobijeni proizvod biti neprihvatljiv, inspektor mora prilagoditi proces ili materijal koji se obrađuje. Podešavanje se mora izvršiti što je brže moguće kako bi se svelo na minimum daljeg odstupanja. Uzmimo pregled koda kao primjer empirijske kontrole procesa. Kod se pregleda u odnosu na standarde kodiranja i najbolje prakse u industriji. Svi uključeni u pregled u potpunosti i međusobno razumiju ove standarde i najbolje prakse. Pregled koda događa se kad god neko smatra da je dio koda koji predstavlja funkcionalno kompletan. Najiskusniji programeri pregledavaju kod, a njihovi komentari i prijedlozi dovode do toga da programer prilagođava svoj kod (Schwaber, 2004).

Scrum svoje prakse bazira na iterativnom, inkrementalnom skeletu procesa (Slika 1). Donji krug skeleta predstavlja iteraciju razvoja aktivnosti koje se javljaju jedna za drugom. Izlaz svake iteracije predstavlja povećanje proizvoda. Gornji krug predstavlja dnevnu inspekciju koja se dešava tokom iteracije, u kojoj se pojedini članovi tima sastaju kako bi međusobno pregledali aktivnosti i izvršili odgovarajuće prilagodbe. Pokretanje iteracije je lista zahtjeva. Ovaj ciklus se ponavlja do momenta kada se projekat više ne finansira (Schwaber i Sutherland, 2020).

Slika 1: Scrum Skeleton



Izvor: CODE Magazine

Kostur (skeleton) funkcioniše na ovaj način: na početku iteracije, tim pregleda šta mora da uradi. Zatim odabire ono što vjeruje da može pretvoriti u potencijalno povećanje za otpremu funckionalnosti do kraja iteracije. Tim tada ostaje sam kako bi uložio sve od sebe do kraja iteracije. Na kraju iteracije, tim predstavlja povećanje funkcionalnosti koju je izgradio na način da dionici mogu provjeriti funkcionalnost i mogu izvršiti pravovremene prilagodbe projektu. Srce Scrum-a leži u iteraciji. Tim razmatra zahtjeve, razmatra dostupnu tehnologiju i procjenjuje vlastite vještine i sposobnosti. Nakon toga tim kolektivno određuje kako će izgraditi funkcionalnost, svakodnevno mijenjajući svoj pristup kako se susreće s novim složenostima, poteškoćama i iznenađenjima. Tim smišlja šta treba da se uradi i bira najbolji način za to. Ovaj kreativni proces je srce Scrum-ove produktivnosti (Schwaber, 2004).

Ukratko, SCRUM procesi predstavljaju temeljne korake agilne metodologije razvoja softvera po imenu SCRUM. Ovi procesi omogućuju timovima da organiziraju svoj rad, prate napredak i kontinuirano poboljšavaju svoje aktivnosti kako bi isporučili visokokvalitetan softver. Glavni fokus SCRUM procesa je na fleksibilnosti, transparentnosti i suradnji unutar tima (Schwaber, 2014).

Jedan od osnovnih elemenata SCRUM procesa je "Sprint". Sprint je definirano razdoblje, obično trajanja od 2 do 4 tjedna, tijekom kojeg tim radi na razvoju softverskog proizvoda.

Svaki Sprint ima jasno definirane ciljeve i prioritete, koji su prepoznati i odobreni od strane Product Owner-a. Tijekom Sprinta, tim svakodnevno održava kratki sastanak pod nazivom "Daily Scrum" kako bi se usklađivali, dijelili informacije o napretku i identificirali eventualne prepreke (Abdur, *et.al.,* 2017).

Sprint počinje "Sprint Planning" sastankom, na kojem se odabiru i prioritetiziraju zadaci koji će se izvršavati tijekom Sprinta. Tijekom Sprinta, tim radi na razvoju tih zadataka i kontinuirano provodi "Sprint Review" kako bi demonstrirao postignuti napredak i dobio povratne informacije od Product Owner-a i potencijalnih korisnika (Schwaber & Sutherland, 2020).

Nakon završetka Sprinta, održava se "Sprint Retrospective", gdje tim reflektira o svojim performansama, identificira što je dobro funkcioniralo i što se može poboljšati te donosi planove za promjene u idućem Sprintu. Ovaj proces kontinuiranog poboljšavanja pomaže timu da se razvija i prilagođava izazovima (Schwaber i Sutherland, 2020).

Scrum Master igra ključnu ulogu u procesima tako da podržava tim u primjeni SCRUM principa, uklanja prepreke i osigurava da procesi teku glatko. Product Owner, s druge strane, odgovoran je za upravljanje backlogom zadataka, definiranje prioriteta i osiguravanje da tim radi na najvrijednijim zadaćama (Abdur, *et.al.,* 2017).

Dakle, SCRUM procesi predstavljaju strukturu i okvir za agilan razvoj softvera. Ova metodologija omogućava timovima da se brzo prilagode promjenama, kontinuirano uče iz svojih iskustava i isporuče vrijedne proizvode u skladu s potrebama klijenata. SCRUM procesi potiču suradnju, komunikaciju i transparentnost, čime pružaju efikasan način za postizanje uspjeha u dinamičnom svijetu razvoja softvera.

2.2. Komparacija između tradicionalnih i agilnih metodologija

Agilne metodologije zasnovane su na adaptivnim metodama razvoja softvera, dok su tradicionalni SDLC modeli (Waterfall model, na primjer), zasnovani na predviđanju pristupa. U tradicionalnim SDLC modelima, timovi rade sa detaljnim planom te imaju punu listu karakteristika i zadataka koji moraju biti završeni u narednih nekoliko mjeseci ili cijeli životni ciklus proizvoda. Prediktivne metode u potpunosti zavise od analize potreba i pažljivog planiranja na početku ciklusa. Bilo kakva potencijalna promjena prolazi kroz strogu kontrolu upravljanjem promjena i određivanje prioriteta. Agilni model korsti adaptivni pristup gdje nema detalja kao što je planiranje te su jasni samo budući zadaci vezani za karakteristike koje moraju biti razvijene. Tim se prilagođava dinamici promjena u zahtjevima proizvoda. Proizvod se često testira, minimizirajući rizik od velikih kvarova u budućnosti. Interakcija sa klijentima jača je strane agilne metodologije i otvorene komunikacije te je minimalna dokumentacija tipična karakteristika okruženja agilnog razvoja. Timovi blisko sarađuju i često se nalaze na istom geografskom području (Stoica *et.al.,* 2013).

Dok je agilni SDLC prikladniji za male i srednje projekte, tradicionalni SDLC velikih razmjera i dalje je bolji izbor. Stoga je važno da razvojni tim bira SDLC koji je najpogodniji za projekat koji je pri ruci. Postoje kriterijumi koje razvojni tim može koristiti za identifikaciju dimenzije željenog SDLC-a. Oni uključuju veličinu tima, geografsku lokaciju, veličinu i složenost softvera, tip projekta, poslovnu strategiju, inženjering sposobnosti itd. Takođe, veoma je važno za tim za proučavanje razlika, prednosti i nedostatak svakog SDLC-a prije izrade odluka. Štaviše, tim mora učiti kontekst poslovanja, industrijske zahtjeve i poslovnu strategiju prije ocjenjivanja kandidata SDLC-a. Pritom je važno imati SDLC evaluaciju i proces selekcije jer on maksimizira šanse za kreiranje uspješnog softvera. Stoga, odabir i usvajanje odgovarajućeg SDLC-a predstavljaju odluku menadžmenta sa duguročnim implikacijama (Stoica *et.al.,* 2013).

Iako agilne metodologije trijumfuju nad tradicionalnim u nekoliko aspekata, postoje mnoge poteškoće pri pokušajima da ih se učini radno sposobnima. Jedan od njih jeste značajno smanjenje dokumentacije i tvrdnja da bi sam izvor koda trebao predstavljati dokumentaciju8. Stoga su programeri navikli na umetanje više komentara u izvorni kod agilnih metodologija kako bi ih razjasnili i objasnili. Za početne programere ili nove članove tima teško je završiti svoje zadatke kada ne razumiju projekat u potpunosti. Primorani su postavljati puno pitanja iskusnim programerima što može odgoditi završetak iteracije te dovesti do povećanja troškova razvoja (Devi, 2013).

S druge strane, tradicionalne metode naglašavaju dokumentaciju u orijentaciji i pojašnjenje projekata za razvojni tim, tako da nema briga o nepoznavanju detalja projekta ili ne posjedovanju dobrog programera. Agilne metodologije poznate su po značaju koji se pridaje komunikaciji i implikacijama klijenata (Shinde *et.al.,* 2015).

Razvojni tim i klijenti će za svaku isporučenu verziju organizovati sastanak gdje če tim predstaviti obavljeni rad tekuće iteracije i klijenti će pružiti povratne informacije o isporučenom softveru (poboljšanja trenutnih karakteristika ili dodavanje novih). U većini slučajeva programeri će smatrati periodiku sastanka (obično se održavaju sedmično) dosadnom i zamornom, jer moraju predstaviti module više puta novim članovima i klijentima. Promjene se mogu desiti na svakoj iteraciji kao tražene od strane klijenata (Shinde *et.al.,* 2015).

Vremenski okvir za svaku iteraciju je kratak (obično traje nekoliko sedmica). Programeri često smatraju raspored preuskim za svaki modul koji zahtijeva složene algoritme obrade. Navedeno dovodio do kašnjenja u svim iteracijama i teškoćama u uspostavljanju efikasne komunikacije između tima članova i klijenata. isto, mijesati reference i izbacivati stvari (Shinde *et.al.,* 2015).

S druge strane, tradicionalne metodologije posjeduju dobro definisan model zahtijeva prije nego što počne proces implementacije i kodiranja, što služi kao referenca za razvojni tim tokom procesa kodiranja. Klijenti ne učestvuju u toj fazi životnog ciklusa razvoja. Razvojni tim će izvršiti kodiranje prema priloženoj dokumentaciji poslovnih analitičara

prije nego je sistem kompletan, i klijentima ga predstavljaju tek kao finalni proizvod. Programeri u ovom slučaju nisu zabrinuti zbog čestih sastanaka i imaju više vremena da završe sistem. Ovo im omogućava da obezbijede bolji proizvod (Shinde *et.al.,* 2015).

Činjenica da agilni razvoj dozvoljava promjene u zahtjevima u inkrementalnom načinu dovodi do dva problema kod dizajna: krutost i mobilnost. Krutost znači promjenu u sistemu koja dovodi do kaskade promjena u ostalim modulima, dok mobilnost znači nesposobnost sistema da uključi komponente za višekratnu upotrebu jer uključuju previše napora ili rizika. Kada su takvi problemi prisutni u cijelom sistemu, on mora biti na visokom nivou restrukturisanja kako bi se eliminisale neželjene ovisnosti (Stoica *et.al.,* 2013).

3. PROJEKT MENADŽMENT I NJEGOVA ULOGA

Termin menadžment tiče se skupa radnji koje imaju za cilj postizanje datih ciljeva na efikasan i djelotvoran način. Ovisno o ciljevima grupe ljudi kojoj su date ove aktivnosti, može se raditi o menadžmentu organizacije, kompanije, ali i projekta. Aktivnosti uključene u proces upravljanja uključuju resurse organizacije, a ovaj je proces ispunjen kombinovanjem i koordinacijom različitih vrsta resursa. Dakle, zadaci onih koji su odgovorni za realizaciju procesa, tzv. menadžera, uključuju: planiranje, donošenje odluka, organizovanje i kontrolu i jednog i drugog budžeta i, naročito, raspored radova. Iako postoje generalna pravila za upravljanje različitim subjektima ona su, bez obzira na njihovu prirodu, suštinski nepromijenjena. Mnoga od njih imaju svoje karakteristike, što uzrokuje da ti entiteti postanu predmetom posebnog istraživanja i analize. Ovo je također uočeno u slučaj IT projekata i procesa softverskog menadžmenta (Tytkowska *et.al.,* 2015).

Posljednjih godina se može vidjeti brza ekspanzija informatike koja se bavi upravljanjem IT projektima. Raznolikost opcija u softverskom modelu implementacije i kontrole, kao i sve veći broj alati koji podržavaju njihovo ispitivanje u praksi, vidljivi su dokazi ovog razvoja.

3.1. Povjerenje i komunikacija u upravljanju projektima

Komunikacija je bitan dio rada u svakom SCRUM timu. Mnogi od svakodnevnih događaja i aktivnosti usmenu se komuniciraju u tzv. "Dnevnom SCRUM-u" (eng. Daily Scrum). (Ratanotayanon, Kotak, & Sim, 2006) čak pokazuju u studiji slučaja da se znanje prenosi kroz duži vremenski period te da se njime može upravljati samo opsežnom komunikacijom. Međutim, što je tim veći, komunikacija je teža. Jedan od najvažnijih izazova u velikim SCRUM timovima je koordinacija među timovima Koordinacija zahtijeva komunikaciju između timova i unutar tima. Važni događaji koji se dešavaju tokom razvoja mogu biti relevantni za druge i često nisu dovoljno dokumentovani ili saopšteni3. Softver programeri ne ažuriraju relevantne dokumente, ne vide njihove

prednosti i želje za više automatskim generisanjem dokumentovanog sadržaja (Forward i Lethbridge, 2002).

U agilnim se timovima svakodnevno dešavaju mnoge aktivnosti i događaji za koje je važno da ih znaju i drugi članovi tima. Ovo je od posebne važnosti kada se donose odluke, pojavljuju se novi zadaci ili se dešavaju neočekivani incidenti i članovi tima nisu adekvatno informisani. Daily SCRUM članovima tima dozvoljava da budu u toku sa najnovijim aktivnostima svojih kolega (Wyrich *et.al.,* 2017).

Problem je također što svaki član tima ne učestvuje na svim sastancima. Ovdje je također važno naglasiti da se svaka važna aktivnost ili događaj ne prenosi u Dnevnom SCRUM-u. Dakle, čak ni dokumentacija Dnevnog SCRUM-a ne bi pokrila aktivnosti cijelog tima. Posljedica odsustva jednog od članova tima je to što on ili ona moraju tražiti ove informacije na različitim mjestima i moraju pitati svoje kolege. To košta vrijeme nekoliko članova tima i ne garantuje da će se tražilac informacija informisati o svakom važnom događaju (Wyrich *et.al.,* 2017).

Jednostavni komunikacijski alati koje koriste programeri poput Slack1, FlowDoc2 ili Gitter3 pružaju osnovne funkcije ćaskanja, ali ne podržavaju strukturisanje, određivanje prioriteta i sumiranje važnih događaja na odgovarajući način. Primjer alata s barem automatskim rezimeom za dnevne SCRUM sastanke predložio je Park (2007).

Moe i Dingsøyr (2008) su zaključili da timska komunikacija može biti jedan od ključnih alata za poboljšanje efikasnosti SCRUM tima. Međutim, rasprostranjena priroda globalnog SCRUM tima može stvoriti neke komunikacijske barijere. SCRUM sastanci kritični su za osiguranje projekta koji uspješno završava jer su ovi sastanci jedini način da tim ostane u kontaktu, pomažu jedni drugima, obavljaju svoje zadatke i razgovaraju o problemima sa kojima se bore. Gibson i Gibbs (2006) tvrdi da se komunikacijski izazovi među globalno raspoređenim članovima tima mogu manifestovati u obliku različitih maternjih jezika, razlike u kulturama, vremenskim zonama i velike fizičke udaljenosti jedan od drugog.

Fizička (lična) interakcija licem u lice može biti još važnija u smislu, ali se to dešava u rijetkim slučajevima. Espinosa i Carmel (2003) navode da timovi za razvoj softvera koji rade u istoj kancelariji pokazuju efikasnost u poređenju sa timovima koji nisu kolocirani. Međutim, zbog svoje neizvodljivosti u smislu globalno raspoređenog tima, interakcija licem u lice se općenito mora zamijeniti interakcijom na mreži. Video konferencije su popularne i predstavljaju pogodnu zamjenu za fizičku interakciju licem u lice i sastanke.

Međutim, istovremeno, izazovi u komunikaciji globalno distribuiranog tima ovdje mogu postati još uticajnijima. Istraživanje Espinose i Carmela (2003) pokazuje da nekim članovima tima možda nedostaju adekvatne vještine engleskog govora (engleski je svjetski jezik koji se koristi u razvoju softvera), da kulturološke razlike mogu gurnuti ljude jedne od drugih, dok razlike u vremenskim zonama mogu učiniti izazovnim učešće na sastancima za neke od članova tima. Distribucija globalnog tima također iziskuje pripremu

telefonskih/video konferencija, što oduzima dosta vremena. Navedeno znači da svakodnevni SCRUM sastanci, planiranje sprintova i recenzija sprintova (koji su uobičajeni SCRUM sastanci) moraju barem ukratko izvještavati. budući da ovi sastanci predstavljaju važne komunikacijske sesije među članovima tima.

3.2. Znanja i vještine projektnog menadžera

Ulogu projektnih menadžera u SCRUM-u igraju tzv. Scrum masteri. Unutar SCRUM-a postoje samo tri uloge: Vlasnik proizvoda, član SCRUM tima i Scrum master. Ovo rezultira ravnotežom između kupca, korisnika i drugih dionika interesa, koje zastupa vlasnik proizvoda, te tehničke realnosti razvoja softvera, koje predstavlja SCRUM tim. Scrum Master olakšava interakciju između ova dva interesa, a služi i za izolaciju tima u cjelini od vanjskih smetnji (otuda i opis "sluga-vođa" koji se čest koristi za opisivanje Scrum mastera) (Abdur, *et.al.,* 2017).

Tri Scrum Master aktivnosti (facilitiranje procesa, facilitiranje ceremonije i uklanjanje prepreka) mogu se smatrati tradicionalnim, tvrde Schwaber i Beedle (2002). Određivanje prioriteta, s druge strane, trebalo bi da bude odgovornost vlasnika proizvoda, dok bi procjenu trebali izvršiti članovi Scrum tima. Iako Scrum Master može facilitirati ove aktivnosti, on ili ona ne bi ih trebali obavljati; to je zato što se Scrum oslanja na ravnotežu moći između "poslovnih" i "tehničkih" interesa kako bi se postavili realni sprint ciljevi . Data uloga Scrum Mastera kao facilitatora i posrednika između vlasnika proizvoda i SCRUM tima, dovodi do preopterećenja uloge Scrum Mastera upravljanjem projektima, te uvodi sukob interesa koji može ugroziti sposobnost Scrum Mastera da osigura ravnotežu između interesa vanjskih dionika i Scrum tima: Scrum master trebao bi izolirati tim i ukloniti prepreke, ali, kao menadžer projekta, on ili ona također imaju odgovornost za postizanje ciljeva postavljenih na višim nivoima organizacije. Stray i kolege primijetili su da, kada se Scrum Master gleda kao menadžer, a ne fasilitator, Dnevni Scrum postaje vježbe izvještaja menadžmenta umjesto timskog komunikacijskog sastanka (Wyrich *et.al.,* 2017).

Ako nastanu tenzije kada se aktivnosti Scrum Mastera kombinuju sa aktivnostima Projekt Menadžera, postavlja se pitanje o tome koja uloga Scrum-a je prava uloga za obavljanje aktivnosti menadžera projekta. Kako bi se odgovorilo na ovo pitanje, korisno je razmotriti tačno šta uključuje upravljanje projektima u Scrum-u, posebno imajući u vidu da bi Scrum timovi trebali biti samoorganizovani. Schwaber definira pet aktivnosti upravljanja projektima koje se moraju provesti kada poduzima razvoj uz korištenje SCRUM pristupa: (1) Upravljanje vizijom - uspostavljanje, njegovanje i prenošenje vizije proizvoda; (2) Upravljanje ROI - praćenje napretka projekta u odnosu na povrat ulaganja u ciljeve, uključujući ažuriranje i određivanje prioriteta zaostatka proizvoda kako bi odražavali ove ciljeve; (3) Upravljanje iteracijom razvoja - proširenje stavki u zaostatku proizvoda u stavke za Sprint Backlog, a zatim implementacija tih stavki po prioritetu; (4) Upravljanje procesom - olakšavanje ceremonija, uklanjanje prepreka i zaštita tima od vanjskih uticaja;

(5) Upravljanje izdanjima - odlučivanje kada se kreira službeno izdanje, kao odgovor na tržišne pritiske i druge realnosti ulaganja (Abdur *et.al.,* 2017).

Od svih navedenih samo je upravljanje procesom odgovornost Scrum Mastera; Upravljanje razvojnim iteracijama je odgovornost razvojnog tima, dok su preostale aktivnosti (upravljanje vizijom, upravljanje povratom ulaganja i upravljanje izdanjima) odgovornosti vlasnika proizvoda (Abdur *et.al.,* 2017).

Navedeno sugeriše da, kada organizacije odluče da usvoje Scrum, njihov postojeći menadžer projekta trebao bi zauzeti ulogu vlasnika proizvoda. Prednosti navedenog su dvostruke: prvo, menadžeri projekta kao vlasnici proizvoda mogu se zalagati za poslovne zahtjeve bez osjećaja napetosti u vezi sa svojim odgovornostima vlasnika proizvoda, jer je takvo zagovaranje u skladu sa ulogom vlasnika proizvoda (Abdur *et.al.,* 2017).

Drugo, Scrum Master bio bi slobodan da podrži Scrum tim kada se radi o zahtjevima koji su u suprotnosti sa tehničkom stvarnošću te da podrži vlasnika proizvoda kada se poslovni prioriteti razlikuju od preferencija članova Scrum tima (na primjer, kada određene svjetovne funkcionalnosti moraju biti razvijene kako bi mapa puta proizvoda napredovala, nauštrb tehnički zanimljivijih karakteristika), te da bi se oboje podržalo kada pritisak višeg menadžmenta prijeti da nadjača ili kompromituje odluku vlastitog tima (Abdur *et.al.,* 2017).

Naš uvid u tenzije i konflikte nastale kombinacijom Scrum Mastera i uloge projektnog menadžera zasnovane su na zapažanjima jednog razvojnog tima i intervjua sa jednim Scrum Masterom/Projekt menadžerom. Zbog toga je potreban izuzetan oprez pri generalizovanju rezultata. Ipak, zapažanja autora taj i taj () sugerišu dvije tvrdnje koje mogu poslužiti kao osnov za daljnja istraživanja: (1) Prilikom usvajanja SCRUM-a, timovi će biti uspješniji ako prethodni Projekt menadžer preuzima ulogu vlasnika proizvoda umjesto uloge Scrum Mastera, te (2) Prilikom usvajanja SCRUM-a, timovi koji kombinuju uloge Scrum Master-a i Project Manager-a doživjet će napetost kao rezultat sukoba interesa između ove dvije uloge (Stray *et.al*., 2013).

Pregled literature koja pokriva upotrebu SCRUM-a u globalno distribuiranom timu pokazuje ograničenu količinu dostupne literature, iz čega se može izvući zaključak da to nije dobro proučeno polje. Ipak, postoji istraživanje Pries-Heje & Pries-Heje (2011) koje je izmalo za cilj proučiti zašto SCRUM radi u globalno distribuiranom timu. Istraživanje je dalo 9 ključnih poena, a to su: (1) SCRUM može razviti odnose i veze između članova tima; (2) SCRUM pomaže u izgradnji povjerenja, čak i u raspoređenim timovima; (3) SCRUM pruža timu jednostavan jezik i jasne ciljeve; (4) SCRUM je veoma zgodan za upravljanje timskim zadacima; (5) SCRUM kreira granične objekte i uloge ključa; (6) SCRUM preporučuje organizovanje sastanaka u skladu sa određenim rasporedom; (7) SCRUM ima uobičajene, ali efikasne alate za kontrolu napretka rada; (8) SCRUM ima procese za osiguranje kvaliteta; (9) Primjena SCRUM-a motiviše tim (Kostin, 2021).

Dakle, ova studija odražava ključne prednosti implementacije SCRUM-a u globalno distribuisanim timovima u razvoju softvera. Drugi značajan rad u ovom polju je "Upotreba SCRUM-a u globalnom razvoju softvera", što je sistematski pregled literature koji su ponudili Hossain, Babar & Paik (2009). Ovo istraživanje je razmatralo 20 radova koji su pokušali istaržiti poteškoće pri korištenju SCRUM-a u globalno distribuiranom timu. Na osnovu Izvještaja o industrijskom iskustvu, ova studija ima nekoliko relevantnih zaključaka koji su važni za potporu trenutnog istraživanja: (1) Potrebno je baviti se pitanjima vezanim za timsku distribuciju, kao što su komunikacija, koordinacija i saradnja; (2) SCRUM je potrebno prilagoditi usvajanju u globalno distribuiranom timu; (3) Moguće je da usvajanje SCRUM-a ima ograničenja zbog globalno distribuiranog konteksta timskih faktora. Ovi zaključci iz 2009. godine naglašavaju glavne probleme zajednice dok SCRUM usvajanje može biti relevantno za trenutnu industriju.

3.3. Organizacija projektnog tima

Projektni tim predstavlja samoupravnu grupu sposobnu da samostalno isporučuje i zadovolji zahtjeve kupaca. Kao rezultat navedenoga, timu je potrebna unakrsna funkcionalna zastupljenost vještine i znanja u domenima podataka, alata i infrastrukture. Tipičan projektni tim može se sastojati od sljedećeg: (a) Vlasnik proizvoda; (b) Scrum master; (c) Arhitekte/Analitičari; (d) Dizajneri i programeri (Kostin, 2021).

Karakteristike i sposobnosti projektnog tima su sljedeće: (a) 5 do 9 resursa koji su raspoređeni na puno radno vrijeme za Sprint; (b) Unakrsne, funkcionalne sposobnosti kao što su vještine, podaci i organizacijsko znanje, (c) Samoosnaživanje; (d) Odgovornost za isporuku proizvoda; (e) Određivanje zadataka potrebnih za isporuku svake funkcije; (f) Procjena truda za svaki zadatak; (g) Razvijanje karakteristika; (h) Rješavanje problema (Kostin, 2021).

Jedan od faktora SCRUM-a jeste i njegova timska struktura. SCRUM tim uglavnom sadrži 4-10 profesionalaca. U stvari, SCRUM ne predstavlja magičan proces, već on radi jer tim radi kako treba i svaki član sudjeluje jedan za drugim. Pored toga, dokazano je da mali timovi u razvoju softvera rade mnogo bolje u poređenju5 sa velikim timovima (Schwaber, 2004).

Tim u SCRUM- u se takođe formira na poseban način, i obično se sastoji od 3-9 profesionalaca, iako postoje različiti pogledi na broj ljudi u timu. Scwaber (2004) predlaže formiranje jednog ili više timova od tri i šest osoba, te da svakom od njih bude dodijeljen set paketa (ili objekata. Jeff Sutherland predlaže korištenje tima od sedam ljudi pri razvoju SCRUM-a. Pomenuti naučnici ove brojke su predložili na osnovu njihovog iskustva na nekiliko SCRUM projekata. Jedna stvar je zajednička u svim gore navedenim sugestijama, a to je da veličina tima ne bi trebala biti velika.

3.4. Uloga u projektnom timu

Dakle, osnovna jedinica SCRUM-a je mali tim ljudi, SCRUM tim. SCRUM tim sastoji se od jednog Scrum mastera, jednog vlasnika proizvoda i programera. Unutar SCRUM tima ne postoje podtimovi ili hijerarhije. On predstavlja kohezivnu jedinicu profesionalaca fokusiranih na jedan po jedan cilj, ciljeve proizvoda. SCRUM timovi su međufunkcionalni, što znači da članovi tima imaju sve vještine potrebne za stvaranje vrijednosti svakog sprinta. Oni su također samoupravni, što znači da interno odlučuju ko šta radi, kada i kako (Schwaber & Sutherland, 2020).

SCRUM tim dovoljno je mali da ostane okretan i dovoljan za obavljanje značajnog posla unutar vlastitih sprintova, što obično obavlja 10 ili manje ljudi. Generalno, autori Schwaber & Sutherland (2020) su otkrili da su manji timovi produktivniji i da bolje komuniciraju. Ako SCRUM timovi postanu preveliki, trebali bi razmotriti reorganizaciju u više kohezivnih SCRUM timova, od kojih je svaki fokusiran na isti proizvod. To je razlog zbog koga bi trebali dijeliti isti cilj proizvoda, zaostatak proizvoda, te vlasnika proizvoda.

SCRUM tim odgovoran je za sve aktivnosti koje su vezane za proizvode, počevši od saradnje dionika, verifikacije, održavanja, rada, eskperimentsianja, istraživanja, razvoja i bilo čega drugog što bi moglo biti potrebno. Organizacija ove timove strukturiše i ovlaštava da upravljaju svojim radom. Rad u sprintovima održivim tempom poboljšava fokus i dosljednost SCRUM tima. Cijeli SCRUM tim je odgovoran za stvaranje vrijednog, korisnog inkrementa svakog sprinta. SCRUM definiše tri specifične odgovornosti unutar SCRUM tima: programere, vlasnika proizvoda i Scrum mastera (Schwaber, 2014).

Developeri ili programeri su ljudi u SCRUM timu koji su posvećeni stvaranju bilo kojeg aspekta upotrebljivog povećanja svakog sprinta. Specfične vještine potrebne programerima često su široke i variraju u zavisnosti od domena posla. Međutim, programeri su uvijek odgovorni za: (a) Kreiranje plana za Sprint, Sprint Backlog; (b) Uvođenje kvaliteta pridržavanjem definicije gotovog; (c) Svakodnevno prilagođavanje plana ka cilju Sprinta, te; (d) Smatranje jednih i drugih odgovornim kao profesionalaca (Schwaber i Sutherland, 2020).

Što se tiče vlasnika proizvoda, oni su odgovorni za maksimiziranje vrijednosti proizvoda koji je rezultat rada SCRUM tima. Način na koji se to radi uvelike se može razlikovati među organizacijama, SCRUM timovima i pojedincima. Vlasnik proizvoda je također odgovoran za efikasno upravljanje zaostatkom proizvoda, što uključuje: (a) Razvijanje i eksplicitno komuniciranje cilja proizvoda; (b) Kreiranje i jasnu komunikaciju stavki zaostatka proizvoda; (c) Naručivanje stavki zaostatka proizvoda; (d) Osiguravanje da je zaostatak proizvoda transparentan, vidljiv i razumljiv (Schwaber, 2004).

Vlasnik proizvoda može obaviti gore navedeni posao ili može delegirati odgovornost na druge. Bez obzira na to, vlasnik proizvoda ostaje odgovoran. Da bi vlasnici proizvoda uspjeli, cijela organizacija mora poštovati njihove odluke. Ove odluke vidljive su u

sadržaju i redoslijedu zaostataka proizvoda, te kroz provjerljivi inkrement na Sprint Reviewu. Vlasnik proizvoda je jedna osoba a ne komisija. Vlasnik proizvoda može predstavljati potrebe mnogih dionika u zaostatku proizvoda. Oni koji žele promijeniti zaostatak proizvoda mogu to učiniti pokušavajući uvjeriti vlasnika proizvoda (Schwaber & Sutherland, 2020).

Scrum Master je odgovora za uspostavljanje SCRUM-a na način koji je definisan u SCRUM vodiču. Oni to rade pomažući svima da shvate SCRUM teoriju i praksu, kako unutar SCRUM tima, tako i organizacije. Scrum Master odgovoran je za efikasnost SCRUM tima. Oni to rade tako što omogućavaju SCRUM timu da poboljša svoju praksu, u okviru SCRUM-a. Scrum majstori su pravi lideri koji služe SCRUM timu i širokoj organizaciji (Abdur *et.al.,* 2017).

Scrum Master SCRUM timu služi na nekoliko načina, uključujući: (a) Obučavanje članova tima u samoupravljanju i međufunkcionalnosti; (b) Pomaganje SCRUM timu da se fokusira na kreiranje inrekmenata visoke vrijednosti koji zadovoljavaju definiciju gotovog; (c) Izazivanje otklanjanja prepreka napretku SCRUM tima, i; (d) Osigurati da se svi SCRUM događaji odvijaju i da su pozitivni, produktivni i održavani unutar vremenskog okvira (Abdur *et.al.,* 2017).

SCRUM Master služi vlasniku proizvoda na nekoliko načina, uključujući: (a) Pomaganje u pronalaženju tehnika za efikasno definisanje ciljeva proizvoda i upravljanje zaostatkom proizvoda; (b) Pomaganje SCRUM timu da shvati potrebu za jasnim i konciznim stavkama zaostatka proizvoda; (c) Pomaganje u uspostavljanju empirijskog planiranja proizvoda za složeno okruženje, te (d) Omogućavanje saradnje zainteresovanih strana prema zahtjevu ili potrebi (Abdur *et.al.,* 2017).

Scrum Master služi organizaciji na nekoliko načina, uključujući: (a) Vođenje, obuka i obučavanje organizacije u njenom usvajanju SCRUM-a; (b) Planiranje i savjetovanje implementacije SCRUM-a unutar organizacije; (c) Pomaganje zaposlenima i zainteresovanim stranama da shvate i primjene empirijski pristup za kompleks rad, te (d) Uklanjanje barijera između zainteresovanih strana i SCRUM timova . (Abdur *et.al.,* 2017).

4. ANALIZA REZULTATA ISTRAŽIVANJA

4.1. Opis uzorka istraživanja

Uzorak istraživanja je bio 100 ispitanika, članova projektnog tima u kompaniji Mistral BiH. Istraživanje je urađeno u periodu jul - august 2023. Anketni upitnik je napravljen uz pomoć alata Google forms i kao takav je dostavljen ispitanicima. Prema podacima u Tabeli 1 anketu je ispunilo 100 ispitanika, i to 66 muškaraca i 33 žene (Tabela 1).

Tabela 1: Spolna struktura ispitanika

Navedite Vaš spol!				
	Frequency - uzorak	Percent - postotak	Valid Percent - važeći postotak	Cumulative Percent - kumulativni postotak
M	67	67,0	67,0	67,0
Ž	33	33,0	33,0	100,0
Total - ukupno	100	100,0	100,0	

Izvor: rad autorice

Najveći broj ispitanika njih 45, ima između 20 i 30, godina te njih 44 između 30 i 40 godina, dok 10 ispitanika ima između 40 i 50 godina. Najmanji broj ispitanika, samo jedna osoba ima između 50 i 60 godina (Tabela 2).

Tabela 2: Starosna struktura ispitanika

Navedite Vašu dob!				
	Frequency - uzorak	Percent - postotak	Valid Percent - važeći postotak	Cumulative Percent - kumulativni postotak
20-30 godina	45	45,0	45,0	45,0
30-40 godina	44	44,0	44,0	89,0
40-50 godina	10	10,0	10,0	99,0
50-60 godina	1	1,0	1,0	100,0
Total - ukupno	100	100,0	100,0	

Izvor: rad autorice

Najviše ispitanika njih 76 ima VSS, 16 ispitanika ima SSS dok osam ispitanika ima VŠS (Tabela 3).

Tabela 3: Obrazovna struktura ispitanika

Navedite stepen Vašeg obrazovanja!				
	Frequency - uzorak	Percent - postotak	Valid Percen - važeći postotakt	Cumulative Percent - kumulativni postotak
SSS	16	16,0	16,0	16,0
VŠS	8	8,0	8,0	24,0
VSS	76	76,0	76,0	100,0
Total - ukupno	100	100,0	100,0	

Izvor: rad autorice

Većina ispitanika njih 57 ima između jedne i pet godina iskustva u upravljanju projektima, dok 30 ispitanika ima između pet i 10 godina iskustva. Najmanji broj ispitanika, njih 6 ima između 10 i 15 godina iskustva u upravljanju projektima dok njih 7 ima 15 godina i više iskustva u upravljanju projektima (Tabela 4).

Tabela 4: Iskustvo u upravljanju projektima

Navedite iskustvo u upravljanju projektima!				
	Frequency - uzorak	Percent - postotak	Valid Percent - važeći postotak	Cumulative Percent - kumulativni postotak
1-5 godina	57	57,0	57,0	57,0
5-10 godina	30	30,0	30,0	87,0
10-15 godina	6	6,0	6,0	93,0
15 godina i više	7	7,0	7,0	100,0
Total - ukupno	100	100,0	100,0	

Izvor: rad autorice

U pogledu pozicije u firmi 66 ispitanika su zaposlenici, dok su 34 ispitanika voditelji projekta (Tabela 5).

Tabela 5: Funkcija u firmi

Navedite Vašu funkciju u firmi!				
	Frequency - uzorak	Percent - postotak	Valid Percent - važeći postotak	Cumulative Percent - kumulativni postotak
Zaposlenik/ica	66	66,0	66,0	66,0
Voditelj/ica projekta	34	34,0	34,0	100,0
Total - ukupno	100	100,0	100,0	

Izvor: rad autorice

4.2. Analiza rezultata dobijenih anketom

4.2.1. Pouzdanost mjernog instrumenta

Analiza pouzdanosti mjernog instrumenta je bio prvi korak kod analize rezultata istraživanja. Test za mjerenje koeficijenta Cronbach alfa se koristi za mjerenje pouzdanosti mjerne skale. Koeficijent je 1951. godine razvio Cronbach i od samog početka je predstavljao najčešće korišten indeks koji se koristio za procjenu pouzdanosti mjernog instrumenta. Najčešće se koristi za ocjenjivanje stavova/tvrdnji u kojima se koristi Likert-ova skala s pet mogućnosti, a koristi se kako bi se ocijenila pouzdanost iste. Koeficijent Cronbach alfa u osnovi predstavlja ukupni koeficijent pouzdanosti za skup varijabli (stavovi/izjave, pitanja) (Raykov, 2015). Veći rezultat u osnovi predstavlja veću pouzdanost korištene skale. Najmanja prihvatljiva vrijednost skale se kreće između 0.7 i 0.8. Vrijednosti manje od 0.7 označavaju nepouzdanu mjernu skalu. Važnost Cronbach alfa koeficijenta se ogleda u tome jer mjeri pouzdanost skale korištene u anketnom upitniku. Naime, bez pouzdanih podataka nemoguće je imati pouzdane rezultate.

Vrijednost Cronbach alpha koeficijenta za korištenu skalu iznosila je 0,897 što upućuje na dobru konzistentnost i pouzdanost mjernog instrumenta (Tabela 6).

Tabela 6: Pouzdanost mjernog instrumenta

Reliability Statistics		
Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
,897	,895	25

Izvor: rad autorice

4.3. Analiza rezultata nakon obavljenog intervjua

Intervju je obavljen s (Ime i prezime osobe), voditeljem projektnog tima u Mistralu. Intervju je obavljen preko online aplikacije Zoom, 31.07.2023. godine.

Pitanje 1: Koje su glavne prednosti korištenja agilne metodologije?

Mislim da su glavne prednosti korištenja agilne metodologije, stalno i kontinuirano praćenje projekata, pojedinačno praćenje rada svakog učesnika u projektu. Dalje, mislim da uz korištenje agilne metodologije brzina i rokovi izvršavanja projekta postaju sve brži.

Pitanje 2: Kako se može upravljati komunikacijom i povjerenjem u razvoju softvera velikih razmjera?

navedeno se postiže isključivo korištenjem projektnih metodologija. U većini slučajeva to su: SCRUM, Kanban i Scrunban.

Pitanje 3: Koje su to prepreke i izazovi kada se govori o komunikaciji i povjerenju kod upravljanja projektima velikih razmjera?

Glavna prepreka je nekorištenje agilnih metodologija. U cilju poboljšanja komunikacije i povjerenja kod upravljanja projektima velikih razmjera agilne metodologije i to na prvom jmjestu SCRUM.

Pitanje 4: Koji su efekti nedostataka komunikacije i povjerenja?

Glavni efekti se ogledaju u tome da se projekti sporo izvršavaju, jer ne postoji centralni mehanizam upravljanja nad projektima, a samim tim ne postoji ni povjerenje među članovima projekta.

Pitanje 5: Koja su to moguća rješenja kao bi se prevladali prethodno spomenuti izazovi?

Odgovor je jednostavan, a to je korištenje agilne metodologije i to SCRM, Kanban ili Scrunban.

Pitanje 6: Po Vašem mišljenju kako primjena SCRUM-a utiče na zadovoljstvo i produktivnost zaposlenih?

Po mom mišljenju SCRUM povećava performanse zaposlenika. što posljedično utiče na zadovoljstvo i produktivnost.

Pitanje 7: Da li primjena SCRUM-a pomaže zaposlenicima da kreiraju bolji proizvod?

Da, posebno u smislu smanjivanja stresa i boljeg balansa između radnih obaveza i privatnog života.

Pitanje 8: Da li primjena SCRUM-a ima pozitivan efekat na radnu kulturu?

Da, svakako posebno na transparentnost što ovu metodologiju čini prikladnim okvirom za upravljanje projektima.

5. ANALIZA REZULTATA S OBZIROM NA POSTAVLJENE HIPOTEZE

5.1. Analiza rezultata s obzirom na hipotezu 1

Hipoteza 1: Primjena SCRUM metode ima pozitivan uticaj na zadovoljstvo zaposlenika u kompaniji.

Za ocjenu istinitosti prve polazne hipoteze korištene su deskriptivne statistike. koristila se skala s pet mogućnosti odgovora čija je srednja vrijednost broj tri. Stavovi koji se odnose na prvu hipotezu su: 5, 11, 12, 13, 16, 18, 19 20 i 21. Prema rezultatima predstavljenim u Tabeli većina analiziranih stavova iz upitnika koje se odnose na prvu hipotezu ovog rada imale su iznos srednje vrijednosti veći od prosjeka (veći od tri) što potvrđuje istinitost prve hipoteze. Iznos srednje vrijednosti manji od prosjeka (manji od tri) imao je stav "korištenje SCRUM-a povećava moj nivo stresa" ($\bar{x}$ = 2,58) što potvrđuje istinitost prve hipoteze, jer korištenje SCRUM-a ne povećava nivo stresa. Također, iznos srednje vrijednosti manji od prosjeka je imao stav "korištenje SCRUMA pomaže uspostavljanju balansa između obaveza na poslu i privatnog života" ($\bar{x}$ = 2,85). Najveći iznos srednje vrijednosti imao je stav "SCRUM pomaže stvoriti prijateljsko okruženje s članovima tima" ($\bar{x}$ = 3,51), zatim slijedi stav "SCRUM omogućuje da moje mišljenje čuju menadžeri" ($\bar{x}$ = 3,42). Na trećem mjestu je bio stav "SCRUM omogućava vrijednim zaposlenicima da budu prepoznati ($\bar{x}$ = 3,25). Dalje slijede stavovi "SCRUM mi pruža više mogućnosti za poboljšanje mojih vještina osim obavljanja redovnih zadataka na poslu" i "osjećam da dobijam više pohvala za svoj rada kada kompanija koristi SCRUM" sa iznosom srednje vrijednosti ($\bar{x}$ = 3,10) Tabela 7). Dakle, od ukupno devet analiziranih stavova koji se odnose na prvu hipotezu, prema rezultatima samo jedan stav nije išao u prilog istinitosti prve hipoteze. Dakle, navedeni rezultati impliciraju kako primjena SCRUM-a ima pozitivan uticaj na zadovoljstvo zaposlenika čime je potvrđena istinitost prve polazne hipoteze ovoga rada.

Tabela 7: Deskriptivna statistika - hipoteza 1

Descriptive Statistics - deskriptivna statistika

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation-standardna devijacija
SCRUM pomaže	100	1	5	3,51	,927

stvoriti prijateljsko okruženje s drugim članovima tima.					
SCRUM me motiviše da prelazim svoje granice kako bih bio produktivniji/ ija.	100	1	5	3,08	1,032
SCRUM omogućava vrijednim zaposlenicima da budu prepoznati	100	1	5	3,25	,925
SCRUM omogućava da moje mišljenje čuju menadžeri.	100	1	5	3,42	,955
SCRUM mi pruža više mogućnosti za poboljšanje mojih vještina osim obavljanja mojih redovnih zadataka na poslu.	100	1	5	3,10	,980
Osjećam da dobijam više pohvala za svoj rad kada kompanija koristi SCRUM.	100	1	5	3,10	,882
SCRUM pomaže uspostavljanj u balansa između obaveza na poslu i privatnog života.	100	1	5	2,85	1,067
Korištenje SCRUM-a povećava moj nivo stresa.	100	1	5	2,58	,912
Korištenje SCRUM-a omogućava ravnopravniji tretman zaposlenika.	100	1	5	3,26	,917

Valid N (listwise - važeće vrijednosti)	100

Izvor: rad autorice

5.2. Analiza rezultata s obzirom na hipotezu 2

Hipoteza 2: SCRUM metoda poboljšava saradnju među članovima tima i pozitivno utiče na stvaranje prijateljskog radnog okruženja u kompaniji.

Za ocjenu istinitosti druge hipoteze korištene su deskriptivne statistike. Za analisu su korišteni slijedeći stavovi iz upitnika: 5, 8 i 9. Prema rezultatima predstavljenim u Tabeli 8, sve analizirana pitanja iz upitnika koje se odnose na prvu hipotezu ovog rada imale su iznos srednje vrijednosti veći od prosjeka (veći od tri) što potvrđuje istinitost prve hipoteze. Najveći iznos srednje vrijednosti imao je stav "SCRUM sastanci pružaju više informacija članovima tima kako bi donijeli ispravne odluke" ($\bar{x}$ = 3,88), zatim slijedi stav "SCRUM mi pruža više mogućnosti da podijelim svoje stavove s članovima tima" ($\bar{x}$ = 3,69). Najmanji iznos srednje vrijednosti imao je stav "SCRUM pomaže stvoriti prijateljsko okruženje s drugim članovima tima ($\bar{x}$ = 3,51) (Tabela 8).

Tabela 8: Deskriptivna statistika - hipoteza 2

Descriptive Statistics - deskriptivna statistika hipoteza 2

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation -s standardna devijacija
SCRUM pomaže stvoriti prijateljsko okruženje s drugim članovima tima.	100	1	5	3,51	,927
SCRUM mi pruža više mogućnosti da podijelim svoje stavove s članovima tima.	100	1	5	3,69	,849
SCRUM sastanci pružaju više informacija članovima tima kako bi donijeli bolje odluke.	100	1	5	3,88	,879

Valid N (listwise) - važeće vrijednosti	100

Izvor: rad autorice

5.3. Analiza rezultata s obzirom na hipotezu 3

Hipoteza 3: SCRUM metoda olakšava dobijanje konstruktivnijih povratnih infirmacija od kolega u timu.

Za analizu rezultata (deskriptivna statistika) s obzirom na treću hipotezu korišteni su stavovi: 8, 9 i 17. Rezultati su pokazali da su analizirani stavovi imali iznos srednje vrijednosti veći od tri čime je potvrđena istinitost treće hipoteze ovoga rada. Najveći iznos srednje vrijednosti imao je stav "SCRUM sastanci pružaju više informacija članovima tima kako bo donijeli ispravne odluke ($\bar{x}$ = 3,88), zatim slijedi stav "SCRUM sastanci mi omogućavaju da dobijem bolje povratne informacije od drugih članova tima" ($\bar{x}$ = 3,78). Najmanji iznos srednje vrijednosti je imao stav "SCRUM mi pruža više mogućnosti da podijelim svoje stavove s drugim članovima tima" ($\bar{x}$ = 3,69) (Tabela 9).

Tabela 9: Deskriptivna statistika - treća hipoteza

Descriptive Statistics - deskriptivna statistika treća hipoteza

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation standardna devijacija
SCRUM mi pruža više mogućnosti da podijelim svoje stavove s članovima tima.	100	1	5	3,69	,849
SCRUM sastanci pružaju više informacija članovima tima kako bi donijeli bolje odluke.	100	1	5	3,88	,879
SCRUM sastanci mi omogućavaju da dobijem bolje povratne informacije od drugih članova tima.	100	1	5	3,78	,991
Valid N (listwise) - važeće vrijednosti	100				

Izvor: rad autorice

5.4. Analiza rezultata s obzirom na hipotezu 4

Hipoteza 4: SCRUM metoda pomaže prepoznavanje truda i zalaganja kod zaposlenika, pomaže da se čuju njihova mišljenja i prepozna njihov trud.

Za ocjenu istinistosti četvrte hipoteze korištena je opet deskriptivna statistika. U analizi su korišteni slijedeći stavovi iz upitnika: 12 i 18. I ovdje su rezultati pokazali da analizirai stavovi imaju iznos srednje vrijednosti veći od prosjeka (broja 3) što potvrđuje četvrtu hipotezu ovoga rada (Tabela 10).

Tabela 10: Deskriptivna statistika - četvrta hipoteza

Descriptive Statistics - deskriptiva statistika četvrta hipoteza

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation - standardna devijacija
SCRUM omogućava vrijednim zaposlenicima da budu prepoznati	100	1	5	3,25	,925
Osjećam da dobijam više pohvala za svoj rad kada kompanija koristi SCRUM.	100	1	5	3,10	,882
Valid N (listwise) - važeće vrijednosti	100				

Izvor: rad autorice

5.5. Analiza rezultata s obzirom na petu hipotezu

Hipoteza 5: SCRUM omugućava zaposlenicima više slobode kod procesa donošenja odluka u kompaniji Mistral BiH. Naime, na taj način zaposlenici su direktno uključeni u proces donošenja odluka, odnosno svoja mišljenja ili kritike mogu dijeliti s drugim članovima tima.

Za ocjenu istinitosti pete hipoteze korišteni su stavovi: 6, 7 i 15. Prema rezultatima svi analizirani stavovi imali su iznos srednje vrijednosti veći od prosjeka čime je potvrđena peta hipoteza ovog rada. najveći iznos srednje vrijednosti imao je stav "SCRUM povećava transparentnost" ($\bar{x}$ = 3,99), zatim slijedi sta "osjećam se više uključenim u proces donošenja odluka kada kompanija za koju radim koristi SCRUM" ($\bar{x}$ = 3,46). Na trećem

mjestu je bio stav "SCRUM mi daje više slobode da donesem ispravne odluke ($\bar{x}$ = 3,45) (Tabela 11).

Tabela 11: Deskriptivna statistika - peta hipoteza

Descriptive Statistics - deskriptiva statistika peta hipoteza

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation - standardna devijacija
SCRUM mi daje više slobode da donesem ispravne odluke.	100	1	5	3,45	,857
SCRUM povećava transparentnost.	100	1	5	3,99	,859
Osjećam se više uključenim u proces donošenja odluka kada kompanija za koju radim koristi SCRUM.	100	1	5	3,46	,904
Valid N (listwise) - važeće vrijednosti	100				

Izvor: rad autorice

5.6. Analiza rezultata s obzirom na šestu hipotezu

Hipoteza 6: SCRUM metoda pomaže ravnomjernoj raspodjeli zadataka među članovima tima u kompaniji.

Iznos srednje vrijednosti za stav 14, a koji se odnosi na ocjenu istinitosti pete hipoteze je bio veći od prosjeka ($\bar{x}$ - 3,93) čime je potvrđena istinitost pete hipoteze ovoga rada, donosno rezultati impliciraju da SCRUM metoda pomaže ravnomjernoj raspodjeli zadataka među članovima tima (Tabela 12).

Tabela 12: Deskriptivna statistika - šesta hipoteza

Descriptive Statistics - deskriptivna statistika šesta hipoteza

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation - standardna devijacija
SCRUM pomaže boljoj raspodjeli zadataka među članovima tima.	100	1	5	3,93	,868

Valid N (listwise) - važeće vrijednosti	100

Izvor: rad autorice

5.7. Analiza rezultata s obzirom na sedmu hipotezu rada

Hipoteza 7: SCRUM metoda odnosno njegova primjena ima pozitiva efekat na usavršavanje zaposlenika i njhovu produktivnost i lični razvoj.

Za ocjenu sedme hipoteze korišteni su slijedeći stavovi iz anketnog upitnika: 1, 11 i 16. Iznos srednje vrijednosti za sve analizirane stavove koji su se odnosili na sedmu hipotezu ovoga rada bio je veći od prosjeka čime je potvrđena istinitost sedme polazne hipoteze ovoga rada. Najveći iznos srednje vrijednosti je imao stav "SCRUM aktivnosti mi pomažu da bolje razumijem viziju i ciljeve svakog projekta" ($\bar{x}$ = 3,77), zatim slijede stavovi "SCRUM me motiviše da prelazim svoje granice kao bih bio produktivniji/a" ($\bar{x}$ = 3,08) i "SCRUM mi pruža više mogućnosti za poboljšanje kojih vještina osim obavljanja mojih redovnih zadataka na poslu" ($\bar{x}$ = 3,10) (Tabela 12).

Tabela 13: Deskriptivna statistika - sedma hipoteza

Descriptive Statistics - deskriptivna statistika sedma hipoteza

	N - uzorak	Minimum	Maximum	Mean - srednja vrijednost	Std. Deviation - standardna devijacija
SCRUM aktivnosti mi pomažu da bolje razumijem viziju i ciljeve svakog projekta.	100	1	5	3,77	,897
SCRUM me motiviše da prelazim svoje granice kako bih bio produktivniji/ija.	100	1	5	3,08	1,032
SCRUM mi pruža više mogućnosti za poboljšanje mojih vještina osim obavljanja mojih redovnih zadataka na poslu.	100	1	5	3,10	,980
Valid N (listwise) - važeće vrijednosti	100				

Izvor: rad autorice

6. ZAKLJUČAK

U zaključku, primjena SCRUM-a kao agilne metode razvoja softvera donosi brojne prednosti i poboljšava proces razvoja u današnjem dinamičnom okruženju. SCRUM promiče suradnju među timovima, omogućava brzu prilagodbu promjenama i fokusira se na isporuku visokokvalitetnog softvera. Ova metodologija temelji se na transparentnosti, inspekciji i prilagodbi, što pomaže timovima da bolje razumiju zahtjeve klijenata i da kontinuirano poboljšavaju svoj rad.

Jedna od ključnih prednosti SCRUM-a je fleksibilnost koju pruža. Ciklusi iteracija omogućavaju timu da brzo reagira na promjene u zahtjevima, tehnologijama ili prioritetima. Ovaj pristup minimizira rizik od razvoja nepotrebnih funkcionalnosti i omogućava kontinuiranu isporuku vrijednih komada softvera. Također, SCRUM potiče komunikaciju unutar tima i sa klijentima, čime se osigurava bolje razumijevanje zahtjeva i smanjuje mogućnost nesporazuma.

Kroz redovite retrospektive i refleksiju, SCRUM promiče kontinuirano poboljšavanje procesa i timskih performansi. Ovaj fokus na učenje iz prethodnih iskustava omogućava timu da se prilagodi i raste s vremenom, što rezultira boljom produktivnošću i kvalitetom softvera.

Ipak, valja istaknuti da uspješna primjena SCRUM-a zahtijeva posvećenost tima i organizacije. Svi sudionici moraju biti angažirani i aktivno sudjelovati u procesu kako bi se ostvarile prednosti metode. Također, SCRUM nije uvijek najbolji izbor za svaki projekt ili organizaciju; važno je uzeti u obzir specifične potrebe i okolnosti kako bi se odabrala odgovarajuća agilna metoda.

U konačnici, SCRUM je moćan alat koji omogućava timovima da se agilno prilagode promjenama i brže isporučuju vrijedan softver. Njegova fleksibilnost, naglasak na komunikaciju i kontinuirano poboljšavanje čine ga relevantnom opcijom u modernom svijetu razvoja softvera.

REFERENCE

1. Abdur, R., Bass, J., & Beecham, S. (2017). *A Study of the Scrum Master's Role.* London: University of East London.

2. Alian, M., & Javanmard, M. (2015). *Comparison between Agile and Traditional software development methodologies.* Teheran: Payama Noor University.

3. Awad, M. (2005). *A Comparison between Agile and Traditional Software Development Methodologies.* Perth: School of Computer Science and software Engineering, The University of Western Australia.

4. Boehm, B. (2005). *A Spiral Model of Software Development and Enhancement.* TRW Defense Systems Group.

5. Cervone, V. H. (2011). Understanding agile project management methods using Scrum. *OCLC Systems & Services: International Digital Library Perspectives , 27* (1), str. 18-22.

6. Devi, V. (2013). *Traditional and Agile Methods: An Interpretation.* Scrum Alliance.

7. Dingsøyr, T. M. (2013). Research challenges in large-scale agile software development. *SIGSOFT Softw. Eng. Notes 38(5)* , 38-39.

8. Espinosa, J. A., & Carmel, E. (2003). The impact of time separation on coordination in global software teams: A conceptual foundation. *Software Process: Improvement and Practice, 8(4)* , 249-266.

9. Forward, A., & Lethbridge, T. (2002). *A survey.* New York, USA: Proceedings of the 2002 ACM Symposium on Document Engineering, DocEng 2002.

10. Gibson, C. B., & Gibbs, J. L. (2006). Unpacking the concept of virtuality: The effects of geographic dispersion, electronic dependence, dynamic structure, and national diversity on team innovation. *Administrative Science Quarterly, 51(3)* , 451-495.

11. Gupta, R., & Reddy, P. (2016). *Adapting agile in a globally distributed software development.* Hawaii International Conference on System Sciences.

12. Hicks, M., & Foster, J. S. (2010). Score: Agile research group management,. *Communications of the ACM , 53* (10), str. 30-31.

13. Hicks, M., & Foster, J. (2009). *SCORE: Agile Research Group Management.* Maryland: University of Maryland, College Park.

14. Hossain, E., Babar, M. A., & Paik, H. (2009). Using Scrum in Global Software Development: A Systematic Literature Review. *2009 Fourth IEEE International Conference on Global Software Engineering* , 175-184.

15. Kostin, D. (2021). *Effective communication in globally distributed Scrum teams.* Porirua, New Zealand: Whitireia Polytechnic.

16. Moe, N., Dingsyr, T., & Dyb, T. (2008). *Understanding self-organizing teams in agile software development.* 19th Australian Conference on Software Engineering.

17. Mundra, A., Misra, S., & Dawale, C. (2015). *Practical Scrum-Scrum Team: Way to Produce Successful and Quality Software.* New Delhi, India: Amrawati University.

18. Owen, R., Koskela, L., Heinrich, G., & Codinhoto, R. (2006). Is agile project management applicable to construction? *Salford Centre for Research and Innovation* , str. 51-56.

19. Paasivaara, M., & Lassenius, C. (2011). *Scaling Scrum in a large distributed project.* 2011 International Symposium on Empirical Software Engineering and Measurement.

20. Pareliya, M. (2018). Implementing Agile Project Management (SCRUM) Approach in Development of Building Projects. *Master Thesis* . Cept University Faculty of Technology.

21. Park, S. (2007). *A daily Scrum meeting summarizer for agile software development teams - Master thesis.* Calgari, Canada: University of Calgary, Canada.

22. Pries-Heje, L., & Pries-Heje, J. (2011). Why Scrum Works: A Case Study from an Agile Distributed Project in Denmark and India. *2011 Agile Conference* , 20-28.

23. Ratanotayanon, S., Kotak, J., & Sim, S. (2006). *After the Scrum: twenty years of working without documentation.* San Francisco, California: Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), 5–7 July 2006.

24. Raykov, T. (2015). Scale Reliability, Cronbach's Coefficient Alpha, and Violations of Essential Tau-Equivalence with Fixed Congeneric Components. *Multivariate Behavioral Research , 4*, str. 329-353.

25. Sapathy, T. (2013). *SBOK Guide.*

26. Schwaber, K. (2004). *Agile Project Management with Scrum.* Microsoft Press.

27. Schwaber, K. (2014). *SCRUM Development Process.* Burlington, MA.

28. Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum. Volume 1.* Prentice Hall Upper Saddle River.

29. Schwaber, K., & Sutherland, J. (2020). *The Definitive Guide to Scrum: The Rules of the Game.*

30. Shinde, L., Tangde, J., & Kulkarni, R. (2015). TRADITIONAL Vs. MODERN SOFTWARE ENGINEERING - AN OVERVIEW OF SIMILARITIES AND DIFFERENCES. *Advances in Computational Research, Volume 7, Issue 1* , 187-190.

31. SNSCT. (2010). *Software Engineering - Spiral Model.* Toronto: University of Toronto.

32. Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional. *Informatica Economică vol. 17, no. 4* , 64-77.

33. Stray, V., Lindsjorn, Y., & Sjoberg, D. (2013). *Obstacles to efficient daily meetings in agile development projects: A case study.* 2013 ACM/IEEE International Symposium on, IEEE.

34. Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review, January-February 1986.*

35. TutorialsPoint. (2. Januar 2009). *SDLC - Spiral Model*. Preuzeto 1. August 2023 iz https://www.tutorialspoint.com/sdlc/pdf/sdlc_spiral_model.pdf

36. Tytkowska, M., Bach, M., & Werner, A. (2015). *Project Management In The Scrum Methodology.* Warsaw, Poland: Silesian University of Technology.

37. Wyrich, M., Bogicevic, I., & Wagner, S. (2017). *Improving Communication in Scrum Teams.* Stuttgart, Germany: Institute of Software Technology, University of Stuttgart.

38. Zahidul - Islam, A., & Ferworn, A. (2020). A Comparison between Agile and Traditional Software Development Methodologies. *Global Journal of Computer Science and Technology: C Software & Data Engineering, Volume 20 Issue 2 Version 1.0* , 7-44.