UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

ANALIZA PROJEKTOV RAZVOJA PROGRAMSKE OPREME IZPELJANIH Z AGILNO METODOLOGIJO SCRUM V IZBRANEM PODJETJU

Ljubljana, oktober 2016 MELITA TRPIN

IZJAVA O AVTORSTVU

Podpisana Melita Trpin, študentka Ekonomske fakultete Univerze v Ljubljani, avtorica predloženega dela z naslovom Analiza projektov razvoja programske opreme izpeljanih z agilno metodologijo Scrum v izbranem podjetju, pripravljenega v sodelovanju s svetovalcem doc. dr. Aljažem Staretom

IZJAVLJAM

1. da sem predloženo delo pripravil/-a samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbela, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobila vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označila;

7. da sem pri pripravi predloženega dela ravnala v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobila soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne Podpis študentke:_____

KAZALO

KAZALO TABEL

KAZALO SLIK

UVOD

Programska oprema je postala nepogrešljiv del vsakdanjega delovnega procesa, njen razvoj pa je že nekaj časa ena med hitreje razvijajočimi se dejavnostmi. V zadnjih nekaj letih je programska oprema močno vplivala na delovanje večine podjetij, čeprav večina še vedno nameni le majhen delež prihodkov za programsko opremo. Najbolj odločilen dejavnik je vsakodnevna odvisnost dela od učinkovitega delovanja programske opreme. Za veliko podjetij je namreč programska oprema postala ključni dejavnik konkurenčne prednosti (Brandon, 2005).

Izdelati pravilno delujočo novo programsko opremo ali vzdrževati obstoječo? Ta odločitev je bistvenega pomena. Metode razvoja, s katerimi se kontrolirajo razvojni cikel, kakovost in stroški, se ob bliskovitem razvoju hitro izpopolnjujejo in spreminjajo. V primerjavi z drugimi industrijskimi panogami je razvoj programske opreme dosti težje planirati. Prav tako je težje kontrolirati kakovost izdelave izdelka. Zato so ljudje na začetku pri razvoju programske opreme uporabljali izkušnje pridobljene s prakso na drugih področjih, npr. pri gradnji strojne opreme. Ker ima programska oprema svoje posebnosti in se vodenje projekta razlikuje glede na organiziranost in cilje, so se skozi čas razvile številne metode, ki olajšujejo projektno vodenje (Solina, 1997).

Sodoben način dela zahteva, da se izvajalec hitro prilagaja in odziva na spremenjene zahteve naročnika. Izvajalec mora zagotoviti kakovosten izdelek, hkrati pa mora biti sposoben v kratkem času spremeniti način dela in dostaviti izdelek v dogovorjenem času. V svetu razvoja programske opreme je možno v grobem planirati smer projekta v bližnji prihodnosti, vendar je to zgolj pričakovana smer, ki je odvisna od zahtev stranke. To pomeni, da je pogosto zelo težko natančno napovedati in opredeliti aktivnosti za več mesecev vnaprej ter planirati potek projekta. V času celotnega razvoja izdelka se prioritete naročnika večkrat spremenijo, na kar mora biti izvajalec sposoben odreagirati in nemalokrat popolnoma spremeniti načrt dela. Zaradi tega je težje oceniti čas za izvedbo projekta, določiti potrebne vire za razvoj ter projekt ustrezno ovrednotiti.

Najstarejši in še vedno zelo pogost pristop razvoja programske opreme je tradicionalen oz. slapovni pristop, pri katerem si vse aktivnosti sledijo zaporedno. Zahteve morajo biti že na začetku jasno definirane, saj se ne moremo vrniti v predhodno fazo razvoja in projekt naknadno spreminjati. Naročnik dobi izdelek na koncu celotnega cikla in pogosto so pomanjkljivosti vidne šele takrat (Solina, 1997).

Za boljše obvladovanje nenehno spreminjajočih se zahtev in pričakovanj končnih uporabnikov izdelka so se sredi devetdesetih letih prejšnjega stoletja razvile agilne metodologije razvoja programskih rešitev. Novi trendi priporočajo uporabo agilne metodologije, ki je v ravno pravšnji meri predpisljiva, enostavna in hitro prilagodljiva. Metodologijo lahko na splošno opišemo kot vse, kar redno počnemo, da bi dosegli željen

rezultat, ki je dober izdelek. Pri razvoju programske opreme to ne predstavlja zgolj postopkov, ki so neposredno povezani z razvojem (npr. analiza, načrt, razvoj itd.), temveč sem sodijo tudi podporni postopki, način komunikacije med vsemi sodelujočimi, pravila odločanja itd. Zato lahko metodologijo označimo tudi kot množico odgovorov, s katerimi se projektni tim strinja (Bajec & Krisper, 2004).

Zadnja raziskava State of Agile organizacije VersionOne (2013) je pokazala, da je Scrum med vsemi agilnimi metodologijami daleč najbolj razširjen, njegov delež pa je v letu 2013 znašal kar 55 %. Glavni razlogi za uvedbo agilne metodologije so: hitrejše plasiranje izdelka na trg, lažje obvladovanje spremenjenih prioritet razvoja posameznih funkcij izdelka in lažje usklajevanje informacijskih in poslovnih ciljev. Laanti, Salo in Abrahamsson (2011) so opravili raziskavo za podjetje Nokia, s katero so potrdili, da se uporabniki agilne metodologije na splošno strinjajo s prednostmi, ki vključujejo večje zadovoljstvo, občutek učinkovitosti, boljšo kakovost, transparentnost in avtonomijo ter hitrejše odkrivanje pomanjkljivosti ali napak. Raziskava, ki jo je opravil Dr. Dobb's Journal (Ambler, 2008), kaže, da ljudje po uvedbi agilne metodologije na splošno verjamejo, da v povprečju proizvajajo izdelek višje kakovosti, da so bolj produktivni in da je zadovoljstvo udeležencev znotraj tima večje. Obenem imajo nižje stroške od tradicionalnih timov, ki ne uporabljajo agilne metodologije. Mahnič in Urevc (2012) sta preverjala uporabo, prednosti in pomanjkljivosti agilne metodologije Scrum in potrdila, da se uporabniki v večji meri strinjajo s trditvami o prednostih Scruma.

Schwaber in Sutherland (2011) definirata Scrum kot okvir, znotraj katerega lahko ljudje rešujejo kompleksne težave in obenem na kreativen in produktiven način dostavijo izdelek visoke vrednosti. Hkrati dodajata, da je Scrum nezahteven in enostaven za razumevanje, vendar težek za obvladovanje. Scrum omogoča jasen vpogled v projektni management in razvojne prakse ter tako ponuja prostor za izboljšave.

Temo magistrskega dela sem izbrala predvsem zato, ker sem bila tudi sama vključena v uvajanje in v proces prehoda iz tradicionalne na agilno metodologijo razvoja programske opreme. Izbrano podjetje se že več kot dvajset let ukvarja z izvedbo poslovnih informacijskih rešitev. Del podjetja se ukvarja z lastnim razvojem, drugi del pa implementira produkte svetovno priznanih dobaviteljev, kjer so potrebne le določene prilagoditve za posameznega naročnika. Pri manjših, manj zahtevnih projektih in z manjšim številom članov v timu je tradicionalni pristop razvoja programske opreme zadovoljivo deloval. Z večanjem obsega projektov in kompleksnejšim razvojem pa tradicionalni pristop ni omogočal potrebne agilnosti in preglednosti. Ravno zaradi dinamike razvoja programske opreme, kompleksnosti projektov in nestabilnega okolja, se je izbrano podjetje odločilo, da spremeni način dela in razmisli o uvedbi agilne metodologije. Z uvajanjem so začeli postopoma. Prvi je agilno metodologijo implementiral oddelek, ki razvija programsko opremo za zavarovalniške rešitve. S pomočjo zunanjega svetovalca, analize načina dela in pogovora z zaposlenimi se je podjetje odločilo, da uvede agilno metodologijo Scrum tudi na oddelku za

implementacijo ERP (angl. *Enterprise resource planning*). V razvojnih timih, ki se delijo glede na projekte (projekt je navadno razvoj izdelka za določeno podjetje oz. naročnika), je bil določen skrbnik metodologije in lastnik izdelka. V samem začetku uvajanja Scruma je bil določen tudi krovni skrbnik metodologije za vse projektne time. Za podjetje je značilno, da time ne sestavljajo le razvijalci in arhitekti poslovnih rešitev, temveč so člani tima tudi svetovalci in ponekod so v tim vključeni še testni inženirji programske opreme. V večini primerov ima razvojni tim v produkciji že delujočo programsko opremo, obenem pa se razvijajo novi sklopi funkcionalnosti ali se dopolnjuje že obstoječi deli izdelka, skladno z željami naročnika. To predstavlja še večji izziv za tim, ki mora novo različico testirati in preveriti, da vse funkcionalnosti še vedno pravilno delujejo.

Kljub uvedenemu Scrumu se še vedno pojavljajo težave pri razvoju in dodajanju novih funkcionalnosti. Nova funkcionalnost večkrat ni dovolj usklajena z naročnikom, zato so potrebne spremembe ali dopolnitve. Posledično prihaja do zamud pri predaji. Pogosto se v produkciji pojavijo tudi napake, ki so posledica slabega testiranja. Veliko birokracije, dokumentacije in novih pravil včasih časovno preobremeni tim.

Čeprav so agilne metodologije vse bolj priljubljene in imajo vedno bolj pomembno vlogo pri razvoju programske opreme, med razvijalci programske opreme še ni konsenza glede primernosti uporabe agilnih metodologij. Empiričnih raziskav, ki bi dejansko potrdile uspešno uvedbo oz. transformacijo iz tradicionalne v agilno metodologijo, je malo. Najbolj pogost razlog je, da podjetja nimajo primerljivih podatkov pred in po uvedbi (Laanti et al., 2011), zato o uspešnosti agilnih metodologij še vedno obstajajo dvomi, saj so nekateri mnenja, da se z uporabo tipičnih praks v ekstremni obliki povečuje nestabilnost in tveganje (Mahnič et al., 2009).

Raziskava organizacije VersionOne iz leta 2010 (VersionOne, 2010) ugotavlja, da sta glavna vzroka za neuspešno uvedbo agilne metodologije predvsem pomanjkanje izkušenj z agilnimi metodologijami in kultura podjetja. Dyba in Dingsoyr (2008) sta izvedla pregled empiričnih raziskav in ugotovila, da je bila opravljena samo ena študija metodologije Scrum, ki s kvantitativnimi podatki ovrednoti prednosti. Istega leta Dingsoyr et. al. (2008) navajajo, da glede na razširjenost uporabe agilnih metodologij obstaja zelo malo empiričnih raziskav in zato kot predmet raziskave predlagajo metodologijo Scrum, s katero imajo uporabniki več izkušenj. VersionOne (2013) še ugotavlja, da so poleg Scruma vedno bolj popularne tudi nekatere druge različice oz. sorodne agilne metodologije. Scrumu z 11 % sledi Scrum/XP Hybrid, Custom Hybrid z 10 % in Scrumban s 7 %. Raziskava je razkrila tudi vse večjo razširjenost agilne metodologije Kanban. Sjøberg et al. (2012) navaja, da je zamenjava Scruma s Kanbanom pozitivno vplivala na razvoj programske opreme v skandinavskem podjetju. Čas razvoja se je skoraj razpolovil, število napak se je zmanjšalo za 10 %, obenem pa se je izboljšala produktivnost. Nekateri drugi, npr. Terlecka (2012), navaja, da niso bili uspešni niti pri uporabi Scruma niti Kanbana in so zato za svoje potrebe razvili hibrid med Scrumom in Kanbanom. Mnogi ga imenujejo kar Scrumban. Tudi Kniberg in Skarin (2010)

v svoji knjigi predstavita tako ločeno uporabo Scruma in Kanbana, kot tudi kombinacijo obeh, ki nastane z namenom izboljšanja razvoja programske opreme.

Podjetje X se zaradi kompleksnosti okolja in dinamike razvoja izdelka še vedno dnevno srečuje s številnimi izzivi, zato je bil cilj magistrske naloge proučiti in preveriti, ali je bila uvedba agilne metodologije Scrum za razvoj programske opreme prava odločitev. Analiza o izboljšanju razvoja, uspešnosti uvedbe in zadovoljstvu uporabnikov zaenkrat še ni bila narejena znotraj podjetja. Na podlagi rezultatov analize, natančne in sistematične proučitve uporabe agilne metodologije Scrum sem želela pripraviti predloge izboljšav za podjetje, kar bi vodilo v še učinkovitejši razvoj izdelka in lažje obvladovanje poteka dela, kar je bil tudi ključni namen mojega dela. S tem sem želela pomagati podjetju pri nadaljnjih odločitvah o uporabi ali prilagoditvi agilnih metodologij. Nenazadnje bodo rezultati analize in predlogi v pomoč ostalim oddelkom znotraj podjetja, ki še ne uporabljajo nobene metodologije, in hčerinskim podjetjem pri analizi koristi in planiranju korakov uvedbe, ki jih morajo upoštevati za uspešno uvedbo in uporabo agilne metodologije Scrum ali katere druge prilagojene metodologije. Moj namen je tudi prispevati k razvoju stroke in pomagati raziskovalcem pri uveljavljanju novi agilnih pristopov v projektno delo.

Cilj magistrskega dela je bil analizirati primernost uporabe agilne metodologije Scrum za razvoj programske opreme v proučevanem podjetju. Zanimalo me je mnenje uporabnikov o prednostih metodologije Scrum. Želela sem tudi izvedeti, kako striktno se podjetje drži pravil in praks, ki jih predpisuje metodologija Scrum. Na koncu sem želela še raziskati, s katerimi težavami se podjetje še vedno srečuje pri svojem delu, in predstaviti možnosti nadaljnje izboljšave uporabe agilne metodologije Scrum.

Postavljena raziskovalna vprašanja so:

- Kaj so po mnenju uporabnikov prednosti in slabosti agilne metodologije Scrum?
- Kje se še vedno pojavljajo težave v delovnem procesu in kako jih je mogoče rešiti oz. kako se jim izogniti?
- Ali je v proučevanem podjetju možna kombinirana uporaba tradicionalnega in agilnega pristopa na enem projektu?
- Ali je agilna metodologija Scrum res primerna za razvoj vseh vrst programske opreme?

V prvem delu magistrskega dela sem na podlagi proučevanja relevantne domače in tuje literature podala splošno predstavitev projektnega managementa, posebnosti vodenja IT projektov in različne vrste razvoja programske opreme. V nadaljevanju sem podrobneje opisala razvoj agilne metodologije, kjer sem največ pozornosti namenila agilni metodologiji Scrum. Ta del sem opisala s pomočjo deskriptivne metode, komparativne metode in metode kompilacije. Predstavila sem tudi ugotovitve in spoznanja mnogih avtorjev, ki se ukvarjajo s tem področjem.

V raziskovalnem delu sem na podlagi proučevanja literature sestavila vprašalnik, ki mi je služil pri izvedbi raziskave v izbranem podjetju. Anketo sem izvedla v timih znotraj podjetja, ki projekt izvajajo po agilni metodologiji Scrum. Z anketo sem raziskovala uporabo Scruma. Poleg tega me je zanimalo mnenje uporabnikov o koristih, ki jih prinaša Scrum, in o morebitnih negativnih učinkih Scruma. Na koncu sem želela še izvedeti, s katerimi težavami se anketiranci še vedno srečujejo pri svojem delu in kaj so po njihovem mnenju ključni dejavniki za uspeh projekta. Anketa je vsebovala zaprta, pol-odprta in odprta vprašanja in je bila sestavljena iz štirih večjih sklopov. Prvi sklop je obravnaval uvedbo in uporabo Scruma na splošno (izobraževanja pred uvedbo, število članov v timu, dolžina cikla, sestanki, ocenjevanje uporabniških zgodb itd.). Namen drugega sklopa vprašanj je bil preveriti izvajanje metodologije Scrum v proučevanem podjetju. V tretjem sklopu sem želela ugotoviti, ali se uporabniki strinjajo s prednostmi Scruma, ki jih navaja literatura, in kaj se jim zdi po uvedbi Scruma boljše v primerjavi s prejšnjo metodologijo. V tem sklopu sem ugotavljala tudi, s katerimi težavami se uporabniki še vedno srečujejo v delovnem procesu. Zanimalo me je tudi, kateri dejavniki po mnenju uporabnikov najbolj vplivajo na uspeh projekta. Zadnji sklop je služil za segmentacijo anketirancev (spol in starost) ter nekaj splošnih vprašanj o izkušnjah (koliko let se že ukvarjajo z razvojem programske opreme, koliko let izkušenj že imajo z uporabo metodologije Scrum in na katerih vrstah IT projektov so že sodelovali).

Na podlagi analize odgovorov in dobrih praks iz literature sem na koncu iskala rešitve in izboljšave. V magistrskem delu sem uporabila tudi znanje, pridobljeno tekom študija, in znanje ter lastne izkušnje, pridobljene tekom uvajanja in uporabe agilne metodologije Scrum na delovnem mestu.

Po uvodu so v prvem in drugem poglavju sprva opisani osnovni pojmi, ki so predstavljali teoretično podlago za nadaljnjo raziskavo. V prvem poglavju sem se osredotočila na management IT projektov in izpostavila posebnosti, ki jih prinaša vodenje razvoja programske opreme. Drugo poglavje sem v celoti namenila agilnim metodologijam, s poudarkom na metodologiji Scrum. Sledila je še primerjava tradicionalne in agilne metodologije. V tretjem poglavju sem izvedla empirično raziskavo o primernosti rabe agilne metodologije Scrum za podjetje X. Temu so sledili rezultati raziskave ter na koncu povzetek ugotovitev empirične raziskave. Na podlagi ugotovitev analize in primerov dobrih praks iz literature sem v zaključku podala končno priporočilo podjetju o možnostih izboljšanja razvoja in managementa projekta. Magistrsko delo sem zaključila z viri in literaturo.

1 MANAGEMENT IT PROJEKTOV

V literaturi najdemo pod angleško besedo »management« štiri procesne korake, ki so potrebni za izvedbo projekta: **planiranje**, **organiziranje**, **vodenje** in **kontroliranje**. Pogosto pa je v slovenski literaturi zaslediti tudi zapis »vodenje projekta« ali »upravljanje projekta«, kar bi lahko v angleščino prevedli kot »leadership« za prvi in »governance« za

drugi zapis. To pa dejansko predstavlja enega od procesnih korakov. Rozman in Stare (2008) zato prevajata angleško besedo »management« kot »ravnateljevanje projektov« ali »ravnanje s projekti«. Tudi za besedo projekt lahko v literaturi zasledimo več definicij, ki so si vsebinsko podobne, razlikujejo se le v različnih poudarkih in stopnji zajemanja značilnosti projektov. Rozman in Stare (2008) tako projekt opredeljujeta kot podjem med seboj povezanih zaposlenih, sredstev in aktivnosti, ki so nujne, da se projekt lahko dokonča.

1.1 Tradicionalni projektni management

1.1.1 Projekt in faze projekta

Projekt lahko opredelimo kot enkraten, časovno in finančno omejen ter ciljno usmerjen kompleksen proces z logično povezanimi aktivnostmi. Cilj projekta je izdelava izdelka ali storitve, ki je v skladu z naročnikovimi zahtevami. Najznačilnejša lastnost projekta je njegova neponovljivost oz. enkratnost, kar pomeni, da se aktivnosti nikoli v celoti ne ponavljajo in da njihovo število ter zaporedje kot tudi udeleženci niso enaki kot pri drugih projektih. Druga pomembnejša lastnost je časovna omejenost projekta. Za vsak projekt se določita začetni in končni datum. Vsak projekt je usmerjen k cilju in vse aktivnosti so planirane in izpeljane, da se doseže postavljen cilj. Omeniti je potrebno še ostale lastnosti projekta, ki so: kompleksnost, konfliktnost, tveganost ter povezanost in soodvisnost projektnih aktivnosti. Glavna omejitev projekta so finančna sredstva in število udeležencev. Bolj kot so cilji kompleksni, bolj kompleksne so aktivnosti, ki vključujejo udeležence z različnih področij. Višja kot je kompleksnost, več pozornosti je potrebno nameniti koordiniranju aktivnosti, kontroliranju rokov, stroškov in sami izvedbi. Problem konfliktnosti pride hitreje do izraza v okolju, kjer se izvaja več projektov, ker prihaja do nasprotij interesov. Projektni managerji se v takih situacijah večkrat bojujejo za izvajalce in druge vire, ki so potrebni na projektu. S konfliktnostjo in enkratnostjo projekta je povezano tudi tveganje. Noben projekt ni enak prejšnjemu, zato obstaja tveganje nepričakovanega. Nekatere informacije lahko pridejo na dan šele, ko se projekt začne izvajati, in popolnoma spremenijo postavljen plan (Stare, 2011).

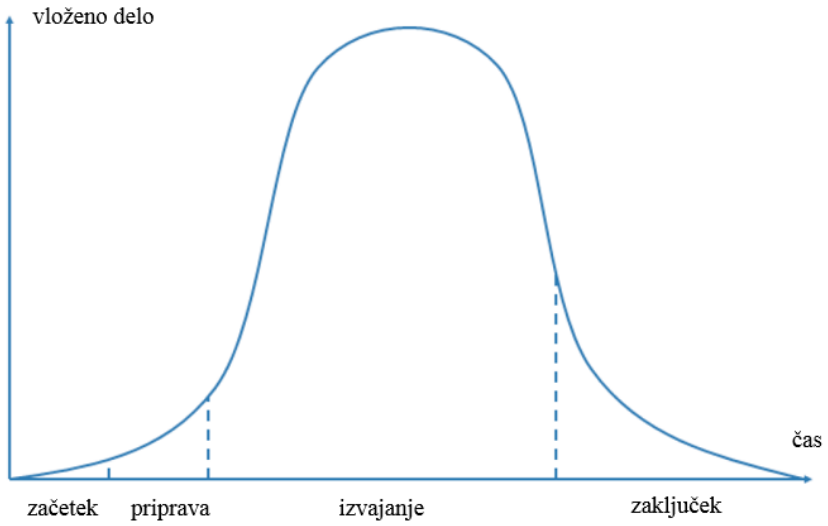
V grobem lahko projekte vsebinsko razdelimo v tri skupine: investicijski, raziskovalno-razvojni in organizacijski projekti. Druga pomembna delitev je na zunanje in notranje ali interne projekte. Če je projekt zunanji, je plačnik znan, projekt pa se izvaja po pravilih, ki jih določa predhodno podpisana pogodba. V tem primeru govorimo dejansko o izvedbenem projektu, kjer imamo 2 deležnika, investitorja in izvajalca. Projekte lahko razvrstimo tudi glede na naročnika. Naročnik je lahko zunanji ali notranji. Včasih je lahko naročnik projekta hkrati tudi izvajalec projekta, npr. športno društvo (Stare, 2011).

Ne glede na vrsto projekta gre vsak projekt skozi več faz, ki so pri večini projektov podobne, vendar nikoli popolnoma enake. Zaporedje faz predstavlja življenjski cikel projekta. Najbolj osnovne so štiri faze (Rozman & Stare, 2008; Stare, 2011):

1. **Začetek projekta**, kjer je v prvem delu potrebno pregledati vse naročnikove zahteve in izločiti tiste, ki nimajo dodane vrednosti. Identificirati se morajo potrebe in priložnosti. Na tem mestu je potrebno tudi zbrati in razčleniti ideje ter pripraviti okvirni plan aktivnosti.
2. **Priprava projekta** vsebuje podrobno planiranje projekta in določitev vseh ključnih aktivnosti. Tim pripravi bolj podroben in jasen terminski plan, plan virov, stroškov in tveganj.
3. **Izvedba projekta** je navadno najdaljša in najobsežnejša faza. Planirane aktivnosti se morajo izvajati točno po planu, ki je bil narejen v fazi priprave projekta. Za uspešno izvedbo te faze projekta so ključnega pomena ustrezno usklajevanje vseh udeležencev, vodenje tima in kontroliranje izvedbe.
4. **Zaključek projekta** zahteva, da tim konča projekt, pripravi dokumentacijo in preda rezultate projekta naročniku.

Delo, ki ga tim vloži, in stroški dela niso sorazmerni v teku projekta. Iz Slike 1 je razvidno, da so v fazi začetka in zaključka projekta vloženo delo in stroški relativno manjši kot v izvedbeni fazi (Rozman & Stare, 2008).

Slika 1: Obseg vloženega dela v odvisnosti od napredovanja projekta



Vir: R. Rozman & A. Stare, Projektni management ali ravnateljevanje projekta, 2008, str. 22

Opozoriti je potrebno, da faze projekta niso enako kot koraki managementa. Faza projekta je bolj vsebinska in predstavlja zaokrožen sklop aktivnosti, katerih rezultat je določen izdelek. V fazi priprave projekta je to npr. plan projekta. Koraki managementa predstavljajo planiranje, organiziranje, vodenje in kontroliranje, kot je bilo že zapisano v začetku prvega poglavja. Seveda obstaja povezava med fazami in koraki. Faza priprave projekta vključuje planiranje in kontroliranje, faza izvedbe projekta pa vključuje koraka vodenje in kontroliranje. V praksi se je sicer pogosto izkazalo, da je smiselno vključiti vodenje in kontroliranje že v fazo priprave projekta. Prav tako se pogosto zgodi, da je potrebno v fazi

izvedbe projekta še planirati, npr. v primeru odstopanj oz. sprememb glede na prvotni plan. Povezanost faz projekta in korakov managementa prikazuje Slika 2 (Stare, 2010).

Slika 2: Povezanost faz projekta in korakov managementa

Vir: A. Stare, Vodenje ali management projekta, 2010

1.1.2 Koraki projektnega managementa

Kot začetni korak projektnega managementa nastopi **planiranje**, kjer se pripravi terminski plan projekta, ki vključuje pripravo celotnega seznama aktivnosti projekta za doseganje ciljev. Aktivnosti si morajo slediti smiselno, upoštevati je potrebno njihovo odvisnost in razmisliti, ali se lahko nekatere aktivnosti izvajajo vzporedno. Upoštevati moramo čas za izvedbo posamezne aktivnosti, da lahko določimo končne roke za izvedbo posamezne aktivnosti in končni rok celega projekta. Pri tem je potrebno upoštevati tudi stroške in razpoložljive vire. Razpoložljivost človeških virov je potrebno zagotoviti dovolj zgodaj, da bodo na voljo, ko je na vrsti aktivnost, za katero so določeni. Temu sledi določitev izvajalca aktivnosti in plan morebitnih drugih potrebnih virov (znanje ali izkušnje druge osebe, oprema, material itd.). Namen planiranja je določiti čim krajši rok izvedbe projekta s čim nižjimi stroški, kjer je plan virov izhodišče za izračun. Seveda je ob tem potrebno upoštevati, da mora uspešna izvedba zagotoviti kakovost končnega izdelka oz. storitve. V tem koraku je potrebno pozornost nameniti tudi potencialnim tveganjem in že v naprej pripraviti ukrepe za zmanjšanje ali preprečitev tveganja. Kljub temu je potrebno pri dokončni določitvi končnega roka upoštevati še eno postavko, ki se večkrat pozabi pri planiranju. Gre za nepričakovane težave, ki se lahko pojavijo šele pri izvedbi in se jim zato ni mogoče izogniti niti z ukrepi, ki bi zmanjšali tveganje. Da se rok projekta ne zamakne, lahko v plan dodamo kontrolne aktivnosti ali pa nekatere aktivnosti ocenimo na daljši čas in s tem zmanjšamo tveganje zamika roka (Stare, 2011).

Korak, ki sledi planiranju, je **organiziranje**, kjer se določi tip organizacijske strukture in okvirne pristojnosti managerja projekta ter opredeli razmerja vseh udeležencev na projektu, ki bodo omogočili izvedbo plana. Izbrana mora biti organizacijska struktura, ki je čimbolj

prilagojena projektnemu delu, tako da omogoča dobro komuniciranje in enostavno izvrševanje nalog (Stare, 2011).

Korak **vodenja** se nanaša na vodenje projektnega tima, kjer člani tima največkrat niso stalno podrejeni projektnemu vodji, temveč so le začasno sposojeni za izvedbo ene ali več projektnih aktivnosti. Ravno zato je zelo pomembno gojiti dobre medsebojne odnose. Tukaj največjo vlogo igra projektni vodja, saj njegov stil vodenja (avtokratski, demokratični) vpliva na motiviranost članov, odnose in delovno vzdušje. Da se projekt lahko izpelje, mora tim združevati člane, ki imajo znanje in izkušnje na različnih strokovnih področjih, kar pa s sabo prinese različne osebne lastnosti, ki jih mora vodja uspešno obvladovati. Timsko delo nikoli ni enostavno in konflikti so neizogibni. Pomemben del vodenja je tudi vodenje projektnih sestankov (Stare, 2011).

Zadnji korak, ki sledi, je **kontroliranje**, kjer se preverja izvedbo vsake aktivnosti in skladnost izvedbe s planom. Najprej se ugotovi trenutno stanje, ki se nato primerja s pripravljenim planom. Na podlagi tega se ugotavljajo morebitna odstopanja. V kolikor so potrebni popravki oz. spremembe, se pripravijo korektivni ukrepi. Najbolj pogosto se kontrolira rezultate dela, roke, stroške, kakovost in tveganje. Kontroliranje se mora izvajati redno, kajti prej ko se odkrije odstopanje, lažje je odpraviti težave. Tukaj je ključnega pomena, da člani tima sodelujejo in dajejo prave podatke o stanju aktivnosti ter ne prikrivajo napak in težav (Stare, 2011).

1.1.3 Udeleženci projektov in njihove vloge

Projekt združuje različne vrste udeležencev, ki igrajo pomembno vlogo pri določanju cilja projekta. Njihovi interesi in želje si lahko nasprotujejo in tako vplivajo na izvajanje in zaključek projekta. V kolikor se nekateri interesi preveč razlikujejo in s tem ogrožajo doseganje cilja, je smiselno razmisliti o tem, da se take udeležence izključi iz tima, v kolikor je to mogoče (Rozman & Stare, 2008). Med tipične udeležence projekta uvrščamo naročnika, uporabnike, skrbnika projekta, projektnega managerja, projektni tim in drugi vodstveni kadri podjetja, kjer se projekt izvaja: uprava, direktor in predsednik uprave (Stare, 2011).

Projektni manager je tisti, ki nosi odgovornost za uspešno in učinkovito izvedbo projekta. Predstavlja osrednjo osebo projekta, ki ima celovit pregled nad projektom, planom projekta, udeleženci in aktivnostmi, ki jih je potrebno izvesti (Stare, 2011). Odgovarjati in poročati mora trem osebam. Z naročnikom sodeluje in mu poroča o izvedenih aktivnostih ter skrbi za pravočasno izvedbo projekta. Podjetju, kjer je zaposlen, mora poročati o napredku projekta in opravičiti oz. argumentirati zamude na projektu. Z nadrejenim ali drugimi projektnimi managerji se mora dogovarjati glede razpoložljivosti udeležencev, v kolikor niso stalno prisotni v njegovem timu. Nenazadnje odgovarja tudi svojemu timu in skrbi, da delo na projektu teče gladko. Skrbeti mora za motiviranost, dobro delovno vzdušje, usposabljanje članov, prenos znanja in ustrezno razdelitev nalog. Projektni manager naj bi imel naslednje

lastnosti: ambicijo za zaključek projekta, vodstvene sposobnosti, poznavanje vsebinskega področja projekta, sposobnost reševanja konfliktov in sposobnost vodenja projekta v stresnih situacijah, ki so največkrat posledica tesnih začrtanih rokov (Rozman & Stare, 2008).

O usodi projekta odloča **vodstvo podjetja**, ki nadzoruje projekt od začetka do konca, skrbi za financiranje projekta, določa prioritete glede na ostale projekte, zagotavlja opremo, dodeljuje vire za projekt in skrbi za usmerjanje projekta, tako da preverja, ali so cilji projekta usklajeni s strateškimi cilji podjetja. Kadar so na projektu potrebne večje vsebinske odločitve, se v proces odločanja vključi tudi vodstvo podjetja (Stare 2011).

V podjetjih, kjer se izvaja več projektov hkrati, se običajno določi tudi nadzornika ali **skrbnika projekta**. Ta oseba ima izkušnje in je tista, ki izbere projektnega vodjo, ga usmerja in mu pomaga pri reševanju organizacijskih problemov. Sodeluje pri pomembnih odločitvah in potrjuje večje spremembe, hkrati pa tudi skrbi za napredek projekta, nadzira delo in rešuje konflikte. Poskrbeti mora tudi, da ima podjetje čim večje koristi od projekta (Stare, 2011).

Ključni udeleženci projekta so **naročnik** in končni uporabniki izdelka ali storitve. Projekt se izvaja za njih in zaradi njih, saj so oni tisti, ki bodo uporabljali končni izdelek oz. končno storitev. Naročnik je tisti, ki opredeli namen projekta in okvirno določi cilje. Poda osnovne zahteve in zamisli ter pri tem sodeluje s projektnim managerjem, tako da pojasnjuje zahteve in potrjuje vmesne rezultate dela (Stare, 2011).

Druga ključna vloga na projektu pa je **projektni tim**, ki je neposredni izvajalec aktivnosti. Avtorji obravnavajo tri ravni tima (Rozman & Stare, 2008; Stare, 2011):

1. **Ožji tim** običajno sestavljajo člani, ki so na projektu prisotni vsaj 60 % in pokrivajo določeno strokovno področje ter so najožji sodelavci projektnega vodje. Pri projektu so udeleženi že od samega začetka in zaradi izkušenj in strokovnega znanja igrajo pomembno vlogo pri celotni pripravi projekta; definirajo potrebne aktivnosti, določijo aktivnosti, ocenijo potrebne količine dela in tveganja, določijo ukrepe za zmanjšanje tveganja itd.
2. **Širši tim** sestavljajo preostali izvajalci, ki se jih določi med planiranjem, nekatere pa celo med izvajanjem projekta. Glede na izobrazbo in usposobljenost so določeni za aktivnosti, odvisno od njihovega trajanja in zahtevnosti. Ni nujno, da so prisotni skozi celotno trajanje projekta. Za nemoten potek timskega dela so pomembne tudi osebne lastnosti članov, ki jih je potrebno upoštevati, ko sestavljamo tim.
3. **Pogodbeni izvajalci**, ki so zunanji sodelavci.

1.2 Posebnost IT projektov

V uvodu smo že omenili, da ima programska oprema v današnjem svetu zelo pomembno vlogo in si težko predstavljamo življenje brez nje. Razvoj programske opreme je težko obvladovati in nadzorovati, saj se zahteve trga pogosto zelo hitro spreminjajo. Po eni strani

je potrebno vzdrževati obstoječo programsko opremo ali jo občasno izboljšati, po drugi strani pa je potrebno razvijati novo programsko opremo za nove dejavnosti in nove potrebe trga. Velikokrat se zgodi, da se IT projekti ne zaključijo.

Zdi se, da se IT projekti nikoli ne zaključijo v dogovorjenem roku in da so stroški izvedbe vedno višji od načrtovanih. Neodvisna ameriška organizacija The Standish Group je že leta 1994 opravila anketo v ZDA, na podlagi katere je bilo ugotovljeno, da je bilo na področju IT uspešno zaključenih le okoli 16 % vseh projektov. 53 % projektov je bilo nezaključenih in 31 % zaključenih z zamudo ter s proračunom, ki je presegal finančni plan. Podatki, ki jih je The Standish Group zbrala leta 2008, so že kazali napredek. Takrat je bilo uspešno zaključenih 32 % projektov, neuspešno pa le 24 % (Nasir & Sahibuddin, 2011). Poročilo, ki ga je The Standish Group pripravila v letu 2015, kjer so zajeli 50.000 projektov, sicer kaže, da procentualno število uspešno zaključenih projektov ne narašča. V letu 2015 je bilo takih le 29 %. Pri analizi sta bila dejavnika uspeha zaključen projekt znotraj dogovorjenih rokov in v skladu s proračunom. Nekoliko manj je sicer neuspešno zaključenih projektov; takih je bilo leta 2015 le 19 %. Rezultati kažejo, da je še vedno veliko manevrskega prostora za doseganje boljših rezultatov (Hastie & Wojewoda, 2015).

Wu et al. (2009) prav tako ugotavljajo, da je management IT projekta primarni dejavnik, ki vpliva na uspeh ali neuspeh projekta. Programsko opremo si je namreč težje predstavljati oz. vizualizirati kot strojno opremo. Zato je IT projekt tudi težje vizualno kontrolirati kot projekt, katerega končni izdelek lahko primemo v roke. Drugi izziv projektnega managerja je dostaviti programsko opremo znotraj dogovorjenih rokov in znotraj dogovorjenega proračuna.

Večino tehnik projektnega managementa je možno uporabiti tudi pri managementu IT projekta, vendar ima tak končni izdelek določene lastnosti, ki IT projekte razlikujejo od vseh drugih. Huges in Cotterell (1999) kot ključne lastnosti navajata: nevidnost, kompleksnost in fleksibilnost. Če opazujemo gradnjo ceste, je napredek očiten, pri razvoju programske opreme pa opravljeno delo ni takoj vidno. Vsaka denarna enota, porabljena za razvoj programske opreme, je bolj kompleksna od denarne enote, ki se porabi za razvoj katerega koli drugega izdelka. Končni uporabniki programsko opremo običajno uporabljajo pri vsakodnevnem delu, zato je velika verjetnost, da jo bo potrebno dopolnjevati in spreminjati.

Brandon (2005) navaja podobne ključne razlike med IT projekti in projekti na drugih področjih. Kot največjo razliko zopet omenja vizualizacijo, ki predstavlja največji izziv. Kot drugo ključno razliko omenja kakovost, saj je težje razvidno, če deli ne ustrezajo, ne delujejo in niso v skladu s celotno funkcionalnostjo. Spodaj je izpostavljenih še nekaj drugih razlik:

- največji strošek je delo z visoko stopnjo specializacije,
- obstaja velika razlika v stopnji produktivnosti človeških virov za enako delo,
- ocena stroškov in časa je bolj zapletena,

- projekti imajo navadno visoko stopnjo zahtevnosti,
- zahteve se pogosto spreminjajo,
- projekti imajo običajno visoko stopnjo tveganja zaradi novosti,
- pogosto so postavljeni nerealni cilji in vršijo se pritiski na projektne managerje ter projektni tim za čim hitrejšo in cenejšo izdelavo programske opreme.

V primerjavi z drugimi projekti, je za IT projekt značilna nujnost, edinstvenost, enkratnost, kratkoročnost, nepoznavanje vseh ciljev na začetku projekta in pogosta nestrokovnost naročnika ter negotovost. Liu & Liu (2010) ugotavljata, da igra pri managementu IT projekta kritično vlogo projektni manager, pri čemer so njegove management spretnosti pomembnejše od tehničnih. Liu & Liu (2010) kot glavne izzive za projektnega managerja navajata: stalno komunikacijo z naročnikom, aktivno komunikacijo z vodstvom podjetja in osredotočenost na same člane tima in ne le na rezultate.

1.2.1 Faze življenjskega cikla programske opreme

Projekti s področja IT in razvoja programskih rešitev so za projektni management še posebej velik izziv. Razvoj je pogosto kompleksen, zahteva veliko časa in ljudi. Ključno je hkratno obvladovanje različnih podrobnosti, kot so obvladovanje stroškov, planiranje ljudi in časa, določanje rokov, planiranje in usklajevanje dela itd. Posledično se je tudi na področju razvoja programskih rešitev uveljavil projektni način dela. Razvoj programske opreme sledi določenemu življenjskemu ciklu, ki navadno zajema naslednje aktivnosti: analiza zahtev programske opreme, planiranje, kodiranje in testiranje (Solina, 1997).

Cohen et al. (2010) še bolj podrobno razdelijo življenjski cikel programske opreme. Opredelijo 8 najznačilnejših faz, in sicer fazo zahteve, analize, kodiranja, testiranja, namestitev, delovanja in vzdrževanja. Omenjajo tudi deveto fazo, in sicer umik oz. prenehanje uporabljanja programske opreme.

Huges in Cotterell (1999) pred pričetkom projekta dodajata še eno fazo in sicer analizo izvedljivosti (angl. *feasibility*). Gre dejansko za odločitev, ali je projekt smiselno izpeljati ali ne. Opravi se popis osnovnih zahtev, ki bi jih nova programska oprema morala vsebovati, poda se groba ocena razvojnih in operativnih stroškov, kar se nato primerja s koristmi, ki bi jih nova programska oprema prinesla. Pri zelo velikih projektih lahko faza analize izvedljivosti preraste v samostojni projekt. Največkrat se to zgodi, ko ima podjetje več ponudb za razvoj nove programske opreme in se je potrebno odločiti za najboljšega ponudnika.

1.2.1.1 Analiza zahtev programske opreme

To je prva aktivnost, ki jo mora izvesti projektni tim. Predstavlja kritično fazo za uspeh projekta. Pričakovanja naročnika morajo biti jasno izražena in na koncu te faze tudi podrobno dokumentirana (Justin, 2013). Za projektni tim predstavlja ta aktivnost izziv, saj

mora tim zagotoviti, da se zajamejo vse zahteve in da so zahteve pravilno tolmačene. Naročnikom je potrebno postaviti prava vprašanja, saj imajo pogosto težave ravno s formuliranjem in izražanjem svojih zahtev. Velikokrat se pojavi težava, da na naročnikovi strani ni kompetentnega sogovornika. Kot je bilo že omenjeno, se zgodi, da naročnik šele ob testiranju funkcionalnosti zares razume, kaj je dobil, in prepozno ugotovi pomanjkljivosti. Za projektni tim je zato velik izziv, da na podlagi slabo definiranih in nejasnih, večkrat celo nasprotujočih si informacij ugotovi, kaj je realen problem (Solina, 1997).

Rezultat analize je specifikacija zahtev, v kateri se natančno opiše funkcionalnost, uporabnost in značilnost programske opreme. Dokument predstavlja vse, kar bo nova programska oprema omogočala (Huges & Cotterell, 1999). Opis mora biti jasen in razumljiv za oba, naročnika in razvijalca. Naročnik mora pred razvoj obvezno potrditi specifikacijo. Zahteve je smiselno razvrstiti po stopnji pomembnosti, da ne izgubljamo časa na zahtevah, ki niso nujne. V tem koraku je potrebna tudi analiza obstoječe programske opreme naročnika. Pogosto naročnik uporablja več programov, zato je potrebno preveriti, ali bo potrebna tudi integracija z obstoječimi programi, ki jih bo naročnik še naprej uporabljal (Solina, 1997).

Aktivnosti v tej fazi bi tako lahko razdelili na (Justin, 2013):

- Analizo tveganosti:
  - Podroben pregled nevarnosti in težav, ki lahko nastanejo pri integraciji z drugim sistemom.
  - Določanje pravil migracije, če gre za zaupne podatke. Pregledati je potrebno, kateri osebni podatki se zbirajo in kako se zbirajo ter doreči pooblastila, ki jih bo uporabnik imel v novem sistemu.
- Specifikacijo funkcionalnosti:
  - Vključuje opis zahtev za vmesnike, kot je npr. opredelitev vstopnih podatkovnih polj (izbira iz šifranta; določitev, če je polje lahko prazno itd.).
  - Pomembne podrobnosti, kot so: Ali je lahko datum vnesen pred današnjim datumom? Katera polja so lahko nastavljena kot privzeta?
  - Določitev delovnega toka skozi sistem, če funkcionalnost to zahteva. Kdo je pooblaščen za potrjevanje? Komu gre zahteva naprej v potrditev?
  - Zgodovinska sled za vsako posodobitev v bazi.
- Specifikacijo ostalih podrobnosti:
  - Možnost razširitve sistema ob naslednji izdaji oz. možnost nadgradnje programske opreme.
  - Ali obstajajo omejitve, ki jih je potrebno upoštevati v tej fazi?

Problem, ki se lahko pojavi v času analize, je komunikacija z naročnikom oz. ugotavljanje, katere zahteve so dejansko nujne. Naročnik ima velikokrat napačne predstave, ko se pogovarja o novi programski rešitvi. Največkrat se ni pripravljen prilagoditi in spremeniti

način dela ter pričakuje, da bo izgled nove programske rešitve enak stari, le z izboljšavami. Naročnik se mora zato zavedati, da bo z uporabo nove programske opreme najverjetneje prišlo tudi do sprememb v delovnem procesu, izvesti bo morda potrebno dodatna šolanja, premeščanje ljudi ali celo zaposlovanje novih (Solina, 1997). Justin (2013) zato predlaga, da se naredi popis vseh naročnikovih zahtev s pomočjo intervjujev in anket. Bolj zahtevna opcija je prikaz oz. priprava prototipa, ki po drugi strani omogoča, da si naročnik lažje predstavlja, kaj bo dobil ob zaključku projekta.

1.2.1.2 Planiranje

Po analizi zahtev sledi planiranje, ki je večstopenjski proces. Najprej se izdela načrt programske opreme, v katerem se predstavijo funkcionalnosti programske opreme. Osnova za pripravo načrta je specifikacija zahtev, ki je bila narejena v fazi analize. Velikokrat se teh dveh faz ne da strogo ločiti in se skozi planiranje postopoma dopolnjuje specifikacijo zahtev (Solina, 1997).

Planiranje je druga kritična aktivnost v razvoju programske opreme, saj je kakovost v največji meri odvisna ravno od dobrega planiranja. Ko je razvijalec pod časovnim pritiskom, to aktivnost včasih izpusti in se takoj loti pisanja programske kode. To se največkrat izkaže za napačno potezo, saj so kasneje potrebni popravki direktno na produkciji ali pa je potrebno programsko kodo še enkrat napisati. Izkaže se, da je tako vzdrževanje programske opreme tudi dosti dražje. Pripravljen načrt programske opreme je v nadaljevanju izhodišče za kodiranje (Solina, 1997).

Pri planiranju ni dovolj, da se pripravi načrt razvoja, temveč je pomembno, da se določijo testni scenariji, ki jih mora ta funkcionalnost pokriti. Če je predhodni popis zahtev naročnika natančen, je lažje definirati vse testne scenarije (Justin, 2013). Pripraviti je potrebno čimbolj realne scenarije in upoštevati tudi robne primere, kjer se lahko odkrije največ napak. Najbolje je, da so testni scenariji usklajeni tudi z naročnikovimi pričakovanji (Solina, 1997).

1.2.1.3 Kodiranje

V tem delu je potrebno specifikacijo in načrt programske opreme pretvoriti v programsko kodo in jo nato v celoti testirati. V kolikor je bila specifikacija dobro pripravljena in načrt vsebuje vse potrebne informacije, je kodiranje enostavnejši del celotnega procesa. Najbolj smiselno je, da razvijalec že med samim razvojem dokumentira programsko kodo, saj bo tako dokumentacija bolj natančna. Kasnejše vzdrževanje programske opreme je dosti lažje, če je bilo predhodno dokumentiranje programske kode na visokem nivoju. Pozornost je potrebno posvetiti tudi uporabniškemu vmesniku in zagotoviti, da je enostaven, razumljiv in uporabniku čimbolj prijazen. V nadaljevanju to olajša tudi predajo programske rešitve naročniku in zmanjšuje čas, potreben za izobraževanje. Smiselno je, da se že pri planiranju uporabniškega vmesnika vključi končne uporabnike (Solina, 1997).

Navadno je to najdaljša faza v življenjskem ciklu razvoja programske opreme. Po uspešno opravljenih testih s strani razvijalca je programska oprema pripravljena na testiranje s strani testnega inženirja (Justin, 2013).

1.2.1.4 Testiranje

Po zaključenem razvoju se izvede interno testiranje, nato sledi testiranje skupaj z naročnikom. S testiranjem se preveri kakovost programske opreme, seveda pa to ni edina aktivnost za preverjanje kakovosti. Pomanjkljivosti in napake je potrebno preprečevati in odkrivati sproti. Že analiza in planiranje lahko preprečita marsikatero napako. Pomanjkljivosti, ki jih odkrije naročnik, je praviloma dražje odpraviti, povzročajo nezadovoljstvo na strani naročnika, ki mora na popravek čakati, in na strani razvijalca, ki dela pod časovnim pritiskom. Vsako preverjanje mora odgovoriti na dve vprašanji: »Ali gre razvoj programske opreme v pravo smer?« in »Ali programska oprema ustreza osnovnim zahtevam?« (Solina, 1997)

Oseba, ki testira, opravi vse teste, ki so bili definirani v fazi planiranja (Justin, 2013). S testiranjem dejansko preverimo tudi delo, ki je bilo opravljeno med analizo, planiranjem in kodiranjem. V kolikor se napaka pojavi, jo je potrebno popraviti in nato zopet opraviti testiranje. Večkrat je iskanje vzroka in odpravljanje napake težavno in nepriljubljeno delo. Ugotovimo lahko, da popravek ne bo tako enostaven in bo vplival tudi na druge funkcionalnosti. Če se testira z zamudo in je bil razvoj opravljen že pred časom, razvijalec običajno potrebuje veliko več časa, da poišče napako, saj lahko že razvija novo funkcionalnost (Solina, 1997).

1.2.1.5 Namestitev programske opreme

Kompleksnost uvajanja in usposabljanja je odvisna od končnih uporabnikov in njihovega predhodnega znanja. Če gre za prvo uporabo nove programske opreme, potem so navadno izobraževanja obsežnejša. Uporabniki običajno že v fazi testiranja spoznajo novo funkcionalnost. Nameščanje programske opreme lahko poteka postopoma ali v enem kosu (James, 2013).

V življenjskem ciklu razvoja programske opreme je možno uporabljati enega od dveh pristopov razvoja programske opreme, o katerih bomo več govorili v naslednjih poglavjih. Slapovni pristop je tradicionalen in ima dobro strukturiran načrt ter zahteve, ki jih je potrebno upoštevati pri razvoju. Ta pristop dobro deluje pri velikih projektih, ki se izvajajo več mesecev. Agilni pristop je bolj prilagodljiv kar se tiče zahtev, planiranja, kodiranja in zato se skozi razvoj programske opreme aktivnosti (analiza, načrtovanje, razvoj, testiranje…) ponavljajo. Ta pristop se bolje obnese pri manjših projektih, kjer se pričakuje stalne izboljšave (James, 2013).

1.2.1.6 Vzdrževanje

Zadnja aktivnost služi vzdrževanju in izboljševanju programske opreme, ko je ta že v produkciji, torej jo naročnik že uporablja. Nadaljnje vzdrževanje, izboljševanje in odpravljanje napak v programski opremi je neizbežno (James, 2013).

Ker je razvoj programske opreme zelo hiter in prihajajo na trg vedno nove programske rešitve, stroški vzdrževanja stare programske opreme naraščajo. Ne glede na to je potrebno programsko opremo vedno deloma vzdrževati, če ne drugače, to od nas zahtevajo spremembe zakonodaje. Naslednje štiri aktivnosti spadajo pod vzdrževanje programske opreme (Solina, 1997):

1. **popravki za odpravljanje napak,**
2. **prilagajanje programske opreme** zaradi sprememb v okolju, ki ne vplivajo na funkcionalnost programske opreme,
3. **razširitev zmogljivosti programske opreme** zaradi novih ali spremenjenih zahtev,
4. **preventivno vzdrževanje programske opreme,** ki vključuje spremembe za lažje nadaljnje vzdrževanje.

Največ težav pri vzdrževanju programske opreme se pojavi zaradi slabe specifikacije in slabe ali neobstoječe dokumentacije. To je dostikrat tudi vzrok za nestrukturirano programsko kodo in nedoslednost pri razvoju. Težje je popravljati programsko kodo tujega razvijalca, zato jih večina raje piše nove programske rešitve. Največkrat osebe, ki vzdržujejo programsko opremo, niso osebe, ki so bile prisotne pri prvotnem razvoju programske rešitve. Pogosto je razvijalec tudi v časovni stiski in nove dokumentacije o spremembah ne naredi, kar se kasneje največkrat izkaže za slabost. Dokumentacija je namreč v pomoč pri testiranju spremenjene programske rešitve, saj tako testni inženir točno ve, kateri testni scenariji morajo biti pokriti (Solina, 1997).

Vzdrževanje programske opreme mora biti ustrezno organizirano, da ne prihaja do nasprotujočih sprememb v že obstoječi programski rešitvi. Vsak zahtevek naročnika se mora preveriti. V razvoj gredo le tisti zahtevki, ki so resnično smiselni in imajo dodano vrednost. Po koncu razvoja mora biti vsaka zahteva dokumentirana. Zahtevke, ki so del vzdrževanja, je smiselno razdeliti glede na tip, ki ima v nadaljevanju določen ustrezen postopek. Kritična napaka, ki ustavi normalno delovanje, ima najvišjo prioriteto (Solina, 1997).

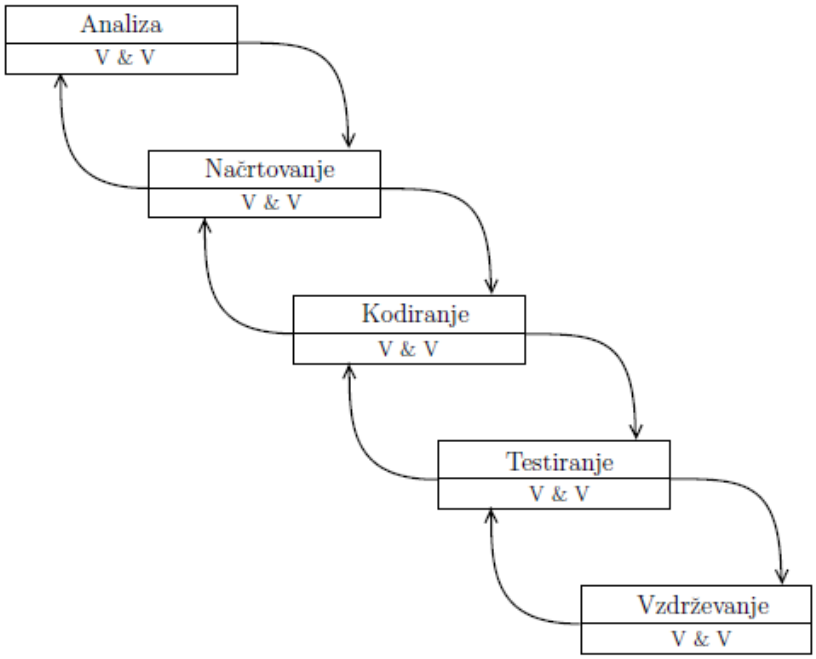
Vzdrževanje programske opreme je smiselno, dokler je ekonomsko bolj upravičeno kot iskanje in nadomestitev z novo programsko rešitvijo (Solina, 1997).

1.2.2 Različni pristopi razvoja programske opreme (razvojni modeli)

1.2.2.1 Klasičen ali slapovni razvoj programske opreme

Najstarejši in verjetno še vedno najbolj pogost postopek razvoja programske opreme je klasičen pristop razvoja programske opreme. V literaturi se uporablja tudi poimenovanje slapovni (angl. *waterfall*), zaporedni, linearni ali kaskadni pristop. Pri takem pristopu si vse aktivnosti sledijo zaporedno kot prikazuje Slika 3. Vsako aktivnost verificiramo in validiramo (V & V). To pomeni, da preverjamo pravilnost razvoja in ugotavljamo, ali razvita funkcionalnost ustreza naročnikovim potrebam (Krisper et al., 2003; Solina, 1997; Schwaber, 1997).

Slika 3: Klasičen (slapovni) razvoj programske opreme



Vir: F. Solina, Projektno vodenje razvoja programske opreme, 1997, str. 15

Pojavil se je v sedemdesetih letih in velja za prvi splošno sprejet pristop razvoja programske opreme. Aktivnosti in izdelki za vsako fazo so natančno definirani, prav tako pa je definirano, kaj vse mora biti izpolnjeno, da se lahko premaknemo v naslednjo fazo. To je bila novost, ki je razvijalcem ponudila disciplino in hrbtenico za kompleksnejši razvoj programske opreme. Največja slabost takega pristopa je, da ne omogoča vračanja v predhodno aktivnost, zato morajo biti vse zahteve naročnika natančno popisane in jasno definirane že na samem začetku. To se v praksi skoraj nikoli ne zgodi. Zelo redko se v celoti zaključi prvo fazo, nato drugo itd. Pogosto se moramo iz faze planiranja vrniti nazaj v fazo analize. Ob takšnem razvoju lahko končni izdelek naročniku dostavimo ob koncu razvoja rešitve in pogosto se šele takrat pokažejo bistvene pomanjkljivosti ali napake, ki so posledica

nerazumevanja zahtev. Kot smo že omenili pa so lahko spremembe že dokončane programske opreme zelo drage (Krisper et al., 2003; Solina, 1997; Schwaber, 1997).

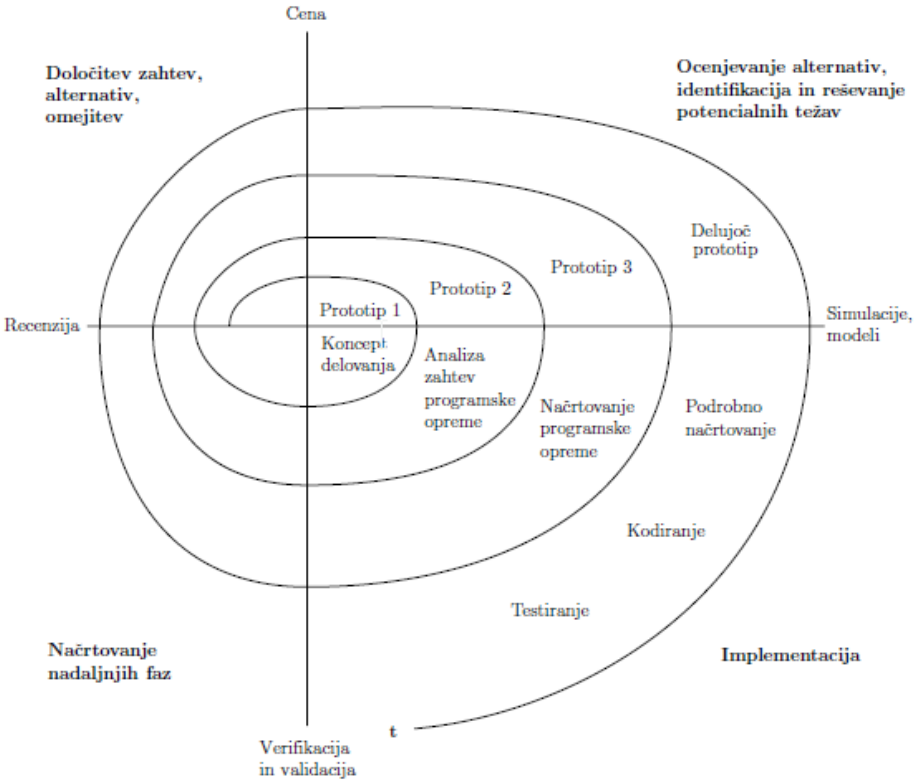
1.2.2.2 Razvoj programske opreme z uporabo prototipov

Največji problem klasičnega razvoja je, da so pred razvojem le redko vse zahteve nedvoumno in popolnoma definirane. Kot odziv na pomanjkljivost slapovnega pristopa se je razvil iterativni pristop, kjer se razvoj izvaja v več ciklih. Razvoj je postopen in za razliko od slapovnega pristopa se faze le delno zaključijo. Z iterativnim pristopom se je razvila uporaba prototipov, ki olajša komuniciranje z naročnikom. Na podlagi prototipa si naročnik lažje vizualno predstavlja končni izdelek. Pri tem pristopu se ponavljata dve fazi, izdelava in ocenjevanje prototipa. Sam proces razvoja prototipov traja toliko časa, dokler naročnik ni zadovoljen. Prototip mora biti testiran in ocenjen s strani naročnika. V nadaljevanju razvoja se prototipi največkrat zavržejo in se uporabijo le kot del specifikacije. Lahko pa se s pomočjo prototipov, ki so razviti skozi več ciklov na koncu razvije končni izdelek. Praksa je vseeno pokazala, da tak pristop ni najboljši. Zaradi hitrega razvoja prototipa ne moremo izvajati vseh aktivnosti, zato je dokumentacija običajno pomanjkljiva, izdelek je slabše kakovosti in manj fleksibilen, kar otežuje njegovo vzdrževanje (Krisper et al., 2003, Solina, 1997).

1.2.2.3 Spiralni razvoj programske opreme

Spiralni model, prikazan na Sliki 4, je kombinacija klasičnega in prototipnega pristopa razvoja programske opreme. Prvi cikel razvoja programske opreme predstavlja prototip 1, kot je prikazano na Sliki 4. Ko se programska oprema preda v uporabo naročniku, se delo običajno še ne zaključi, saj se šele takrat ugotovijo pomanjkljivosti. Primopredaji sledi razvoj dodatnih sprememb. Poleg uvedbe sprememb pa je seveda potrebno programsko opremo tudi vzdrževati. Ker naročnik že uporablja novo programsko opremo, se z vzdrževalnimi deli največkrat mudi. Zopet se odvije cikel analize, planiranja, razvoja popravka v kodi in nato še testiranje. Zaradi pomanjkanja časa je dokumentacija večkrat izpuščena, kar se kasneje pokaže kot problem, saj ni nikjer zabeleženo, kaj je bilo dogovorjeno. To še nadalje oteži vzdrževanje. Spiralni model je med najbolj poznanimi iterativnimi pristopi razvoja programske opreme. Razvoj vsebuje več ciklov in vsak cikel vsebuje analizo, planiranje, kodiranje in testiranje. To so dejansko faze klasičnega razvoja programske opreme. Ker se programska oprema stalno razvija in izpopolnjuje, je potrebnih več ciklov razvoja (Krisper et al., 2003; Solina, 1997; Schwaber, 1997).

Slika 4: Spiralni razvoj programske opreme



Vir: F. Solina, Projektno vodenje razvoja programske opreme, 1997, str. 20

1.2.2.4 Inkrementalni ali postopni razvoj programske opreme

Inkrementalni ali postopni razvoj programske opreme razdeli celoten razvoj na manjše, neodvisne dele. Razvijajo se posamezne funkcionalnosti, ki so dovolj samostojne, da se jih lahko takoj preda naročniku. Vsak razvoj dela funkcionalnosti uporablja klasičen pristop in tako pokriva vse korake, od analize, planiranja, kodiranja, testiranja, implementacije do vzdrževanja. Različne dele funkcionalnosti lahko razvijamo tudi sočasno oz. vzporedno. Programska oprema na ta način postopoma raste in sčasoma se pokrije celotna začrtana oz. planirana funkcionalnost. Ko je predan še zadnji sklop funkcionalnosti, se razvoj lahko zaključi. Prednost takega pristopa je v tem, da dobi naročnik nekatere dele funkcionalnosti že zelo zgodaj v uporabo. Na ta način se morebitne napake in pomanjkljivosti hitreje odkrijejo. Takšen razvoj je cenejši od razvijanja funkcionalnosti v enem kosu (Krisper et al., 2003, Solina, 1997).

1.2.2.5 Kombiniran razvoj programske opreme

Osnova za kombiniran pristop razvoja programske opreme je klasičen pristop, s to razliko, da omogoča vračanje v predhodno fazo in vključuje uporabo prototipov. Dobre lastnosti klasičnega modela se ohranijo, medtem ko se pomanjkljivosti odpravijo z iterativnim pristopom. Kombiniran pristop je možno uporabljati tudi v kombinaciji z inkrementalnim

pristopom, kjer se posamezne funkcionalnosti razvijajo postopoma. Tak pristop je najbližje realnem procesu razvoja, kjer je večkrat potrebno vračanje v predhodno fazo. Vseeno so predpisane zaporedne aktivnosti (analiza, planiranje, kodiranje, testiranje itd.), ki se morajo izvajati, da ne bi bilo delo popolnoma neorganizirano. Pozitivna je tudi uporaba prototipov v vseh fazah razvoja, s čimer se izboljšuje komunikacija med naročnikom in razvijalcem (Krisper et al., 2003).

Zgoraj opisane metodologije so tradicionalne oz. klasične. Ker je potrebno slediti potrebam razvoja programske opreme, se razvijajo novi pristopi managementa projektov z namenom učinkovitejšega in boljšega obvladovanja nenehno spreminjajočih se zahtev. Razvile so se agilne metodologije, med katerimi najbolj izstopa agilna metodologija Scrum. Kompleksnost programske opreme zahteva maksimalno prilagodljivost in primerno kontrolo ter pregled nad celotnim projektom. Klasične metodologije definirajo vsebino in načrt razvoja izdelka na začetku projekta, kar onemogoča hitro odzivanje na nepričakovane dogodke. Za razliko od klasičnih metodologij, agilna metodologija Scrum celotno vsebino razdeli na manjše dele. Vse prakse Scrum so osnovane na iterativnem in inkrementalnem procesu. Ker se razvojne aktivnosti ponavljajo, se definirajo krajši cikli, kjer je vsak cikel projekt zase. Za vsak cikel se naredi seznam zahtev, vsako zahtevo se analizira in načrtuje. Sledita razvoj in testiranje ter nazadnje še implementacija (Schwaber, 2004). V nadaljevanju je na kratko opisan razvoj agilnih metodologij, bolj podrobno pa je predstavljena agilna metodologija Scrum.

2 AGILNA METODOLOGIJA SCRUM

2.1 Razvoj agilne metodologije

Prve agilne metodologije so se pojavile že v devetdesetih letih prejšnjega stoletja kot odgovor na tradicionalne metodologije z namenom obvladovanja vse pogostejših zahtev po spremembah programske opreme. Slovar slovenskega knjižnega jezika besedo agilen opredeljuje kot delaven in prizadeven. V literaturi največkrat zasledimo, da so agilne metode prilagodljive in usmerjene k ljudem in ne k procesom tako kot tradicionalne. Med bolj izstopajočimi so Ekstremno programiranje (XP), Razvoj narekovan z lastnostmi (FDD), Metoda dinamičnega razvoja sistemov (DSDM) in Scrum (Williams, 2010). VersionOne (2013) še ugotavlja, da so poleg Scruma vedno bolj popularne tudi nekatere njegove različice oz. sorodne agilne metodologije. Scrumu z 11 % sledi Scrum/XP Hybrid, Custom Hybrid z 10 % in Scrumban s 7 %. Raziskava je razkrila tudi vse bolj pogosto uporabo agilne metodologije Kanban.

Različni avtorji, ki so se povezali v organizacijo »Agile Aliance«, so februarja 2001 napisali Manifest agilnega razvoja programske opreme (Williams, 2010).V Manifest agilnosti so zapisali (Beck et al., 2001):

»Odkrivamo boljše načine razvoja programske opreme, tako da jo razvijamo in pri tem pomagamo tudi drugim. Naše vrednote so ob tem postale:

Posamezniki in interakcije pred procesi in orodji.
Delujoča programska oprema pred vseobsežno dokumentacijo.
Sodelovanje s stranko pred pogodbenimi pogajanji.
Odziv na spremembe pred togim sledenjem načrta.

Z drugimi besedami: četudi cenimo dejavnike na desni, vseeno bolj cenimo tiste na levi.«

V nadaljevanju so opredeljene najbolj aktualne agilne metodologije.

2.1.1 Ekstremno programiranje (XP)

Agilna metodologija ekstremno programiranje je bila na začetku namenjena manjšim skupinam, ki so se soočale z negotovimi in spreminjajočimi se zahtevami. Najpomembnejšo vlogo in največjo odgovornost ima razvijalec programske opreme. Glavna značilnost ekstremnega programiranja je programiranje v parih. To pomeni, da razvijalca sedita skupaj in sodelujeta pri pisanju programske kode. Na takšen način dobimo kakovostnejšo kodo, z manjšim številom napak in pomanjkljivosti. Razvijalca si lahko sproti izmenjujeta izkušnje, mnenja in informacije. Delati v paru ni enostavno, zato je priporočljivo, da se razvijalca najprej udeležita izobraževanja. Z naročnikom je potrebno stalno vzdrževati kontakt. Sprejemni testni scenariji morajo biti napisani in potrjeni s strani naročnika. Ekstremno programiranje predvideva tudi uporabo avtomatiziranih testov. Razvoj poteka v kratkih ciklih, ki so lahko celo dnevni (Williams, 2010).

Vsaka zahteva predstavlja svojo uporabniško zgodbo, ki jo navadno napiše naročnik sam ali v sodelovanju z razvojnim timom. Uporabniška zgodba nosi kratek opis, naslov zahteve, morebitne opombe in sprejemne teste. Ko se razvoj zaključi, se preveri, ali je uporabniška zgodba zaključena. Lastništvo nad kodo je skupno, tako lahko kdorkoli v skupini pregleda in popravi kodo drugega razvijalca (Williams, 2010).

Obstajajo različne vloge, ki jih definira metodologija ekstremnega programiranja. Te so: manager ali vodja, trener, sledilec, razvijalec ali programer, tester in naročnik. Manager je tisti, ki na podlagi komunikacije z ostalimi člani razvojnega tima sprejema odločitve, je pozoren na pomanjkljivosti in težave ter ocenjuje trenutno situacijo v timu. Trener (angl. *Coach*) je oseba, ki vodi razvojni tim in mora dobro poznati in razumeti metodologijo ekstremnega programiranja, saj je nenazadnje odgovoren za celoten proces. Oseba, ki zaseda vlogo sledilca, preverja napredek posameznega cikla in ocenjuje, ali so bili cilji doseženi znotraj zastavljenih časovnih okvirjev. Sledi ocena za izvedbo, ki je povratna informacija o natančnosti končnega izdelka. Te informacije se uporabijo pri naslednjem ocenjevanju nalog. Razvijalci morajo med sabo dobro komunicirati in koordinirati delo, zato da je programska koda enostavna in jasno definirana. Naročnik določa prioritete uporabniških

zgodb in pripravlja testne scenarije. Testni inženir pred predajo funkcionalnosti testira testne scenarije, ki so bili dogovorjeni z naročnikom (Williams, 2010).

Od vseh agilnih metodologij daje ekstremno programiranje največ poudarka na sodelovanje vseh članov projektnega tima in tudi vključevanje naročnika v proces izvajanja projekta. Odnos med naročnikom in projektnim timom je partnerski. Uporaba te metodologije je najbolj primerna za raziskovalno-razvojne projekte, pri katerih so zahteve in cilj nejasni (Stare, 2011).

2.1.2 Razvoj, osredotočen na funkcionalnosti (FDD)

Razvoj, osredotočen na funkcionalnosti (angl. *Feature Driven Development – FDD*), je procesno orientirana agilna metodologija, ki se osredotoča predvsem na fazo planiranja in fazo izvedbe. Proces dela poteka skozi pet iterativnih procesov. Definirana so priporočila glede časa za vsak proces. Definiran je tudi (omejen) čas za planiranje, pripravo specifikacije in načrta. Procesi ena, dva in tri se opravijo pred začetkom projekta in se kasneje dopolnjujejo skozi razvojni proces. Procesa štiri in pet se ponavljata v dvotedenskih ciklih. Spodaj so našteti vsi procesi (Williams, 2010):

1. **Razvoj splošnega modela**: Definirata se vsebina in obseg projekta.
2. **Priprava seznama funkcionalnosti programske opreme**: Sestavi se podroben seznam značilnosti, ki jih je potrebno združiti na logične vsebinske sklope.
3. **Planiranje po funkcionalnostih**: Pripravi se plan razvoja glede na tveganje, kompleksnost in prioritete.
4. **Priprava načrta funkcionalnosti**: Procesa tri in štiri sta najobsežnejša in najpomembnejša. V procesu štiri se pripravi podroben načrt razvoja vsake značilnosti.
5. **Razvoj funkcionalnosti.**

2.1.3 Kanban

Agilna metodologija Kanban se ravna po načelu »ravno v pravem času« (angl. *Just-In-Time*) in v času razvoja ne preobremenjuje članov razvojnega tima. Koncept izhaja iz podjetja Toyota, kjer je bil vpeljan kot sistem oskrbe delovnih mest z materialom. V programskem svetu je Kanban metodologija, ki nam pokaže, kaj, kdaj in koliko naj razvijamo. Anderson jo je prvič predstavil javnosti leta 2010 v knjigi Kanban: Successful Evolutionary Change to your Technology Bussiness. Vendar pa je bila Kanban metodologija dejansko uporabljena že leta 2004 v Microsoftu, kjer je Anderson delal. Metodo je formuliral kot odgovor na pritiske, ki jih je vršil vrhnji management glede postavljenih zahtev in rokov (Anderson, 2010).

V tem času je videl dva večja izziva (Anderson, 2010):

- Prvi izziv je bil, na kakšen način je mogoče zaščititi razvojni tim pred nenehnimi zahtevami vrhnjega managementa in hkrati doseči tempo razvoja, ki je vzdržljiv za člane razvojnega tima.
- Drugi izziv je bil vpeljati agilno metodologijo v podjetje in hkrati preseči neizbežen problem upora proti spremembam v načinu dela.

Pri iskanju rešitev za omenjena izziva je Anderson (2010) prišel do spoznanja, da je razvoj, ki je trenutno v teku, neposredno povezan s povprečnim časom izdelave. S tem je prišel do ugotovitve, da več kot je dela v teku, daljši je povprečni čas izdelave, kar slabo vpliva na kakovost. Z omejitvijo količine dela v teku se izboljša kakovost in skrajša povprečni čas, ki je potreben za izdelavo. Ravno omejitev obsega dela (angl. *Work-in-Progres – WIP*) je glavna značilnost metodologije Kanban, po kateri se Kanban tudi najbolj razlikuje od Scruma. Druga pomembnejša značilnost te metodologije je, da delo poteka po principu »prevzemi delo, ko si pripravljen« (angl. *Pull*), in ne po principu »delo ti nalagajo drugi« (angl. *Push*). Omogočeno je postopno uvajanje sprememb v sistem, zato metodologija ni tako strogo začrtana kot Scrum, kjer se lahko razvojni proces spremeni čez noč, da bi ustregel nekim vnaprej določenim pravilom.

Glavne lastnosti Kanbana so (Anderson, 2010):

- vizualizacija poteka dela (angl. *Visualize the workflow*),
- omejitev obsega dela,
- upravljanje delovnega toka (angl. *Manage flow*),
- jasna pravila v procesu,
- uvedba zank za povratne informacije (angl. *Implement feedback loops*),
- izboljšanje sodelovanja in spodbujanje eksperimentiranja z znanstvenimi metodami.

Metodologija Kanban skuša omejiti odpor na spremembe, ki so posledica uvedbe nove metodologije. Če je metodologija predstavljena na pravi način, bo podjetje z uvedbo le pridobilo (Anderson, 2010).

Anderson omenja 6 korakov, ki so potrebni za uspeh. Spodaj zapisane korake označuje kot recept za uspeh (Anderson, 2010):

1. osredotočenost na kakovost,
2. zmanjšanje količine dela v teku,
3. pogoste izdaje,
4. iskanje ravnovesja med povpraševanjem in zmogljivostjo,
5. postavljanje prioritet,
6. uravnavanje virov variabilnosti za povečanje predvidljivosti.

2.1.4 Scrumban

Omenili smo že, da je metodologija Scrum največkrat uporabljena metodologija. Sledijo ji metodologije, ki so hibridi med Scrumom in še eno metodologijo. Metodologiji, ki je kombinacija lastnosti Scruma in Kanbana, mnogi pravijo Scrumban. Mahnič (2014) navaja, da se je število uporabnikov Scrumbana v letu 2013 skoraj podvojilo. Veliko razvojnih timov, ki uporabljajo Scrum, ugotovi njegove pomanjkljivosti ali nepravilnosti. Ker pri razvoju že uporabljajo Scrum, to metodologijo raje prilagodijo, kot da bi uvajali popolnoma novo. Mahnič (2014) v svojem članku Improving Software Development through Combination of Scrum and Kanban združi najpomembnejše prakse metodologije Scrum, kot so planiranje cikla, redno izdajanje in podajanje povratnih informacij, z osnovnimi načeli Kanban, ki zahtevajo vizualizacijo delovnega procesa in omejevanje količine dela. Ta metoda je predvsem primerna za razvojni tim, ki že uporablja Scrum in želi doseči večjo vitkost razvoja, še boljšo preglednost in stalno optimizacijo.

Mahnič (2014) predlaga Kanban tablo s sledečimi kolonami za lažji prehod iz Scruma na Scrumban metodologijo:

- **Seznam zahtev izdelka** (angl. *Product Backlog*) je kolona, kjer so uporabniške zgodbe že definirane in imajo določeno prioriteto, ki jo je določil lastnik izdelka. Vse naloge morajo biti tudi ocenjene.
- **Seznam zahtev cikla** (angl. *Sprint Backlog*) je kolona, ki vsebuje vse uporabniške zgodbe, ki jih je razvojni tim skupaj z lastnikom izdelka določil v fazi planiranja. Način dela, ki se uporablja od razvoja in testiranja do predaje, je sistem vlečenja dela, ki je značilen za Kanban.
- **Naslednji** *(*angl. *Next*) je kolona namenjena omejeni količini nalog z visoko prioriteto. Ko razvijalec prevzame novo nalogo, jo iz kolone Naslednji premakne v kolono Analiza in Načrt.
- **Analiza in Načrt** je kolona, razdeljena v dve podkoloni: **V delu** in **Dokončano**. Na tej točki morajo biti vse uporabniške zgodbe, ki so v koloni Končano, analizirane in definirane tako, da je razvoj jasen in večjih neznank ni. Podkolona Dokončano dejansko služi kot izravnava toka (angl. *buffer*). Enako velja za kolone Razvoj, Testiranje in Dokumentacija, ki so razdeljene v podkoloni V delu in Končano.

Vsaka kolona ima določeno omejitev količine nalog v delu in se lahko po potrebi prilagaja. Na koncu lastnik izdelka potrdi nalogo, ki je pripravljena za stranko, ali jo zavrne, če ne ustreza definiciji statusa »dokončano«. Definicija statusa »dokončano« predstavlja seznam pogojev, ki morajo biti izpolnjeni, da je lahko ta del funkcionalnosti predan naročniku. Če naloga vseh pogojev ne izpolnjuje, se vrne na začetek, v kolono Naslednji in je označena kot naloga z višjo prioriteto (Mahnič, 2014).

2.2 Scrum

Ken Schwaber in Jeff Sutherland (Schwaber & Sutherland, 2011) sta razvila metodologijo Scrum in napisala Scrum vodič. Scrum opisujeta kot okvir, znotraj katerega je možno reševati kompleksne probleme, hkrati pa produktivno in kreativno razviti izdelek najvišje kakovosti. Scrum okvir sestavlja Scrum tim in znotraj tima dodeljene vloge, dogodki, pravila in potrebna orodja za izvajanje Scruma.

Schwaber in Sutherland (2011) metodologijo Scrum definirata kot:

- vitko,
- enostavno za razumevanje,
- zelo zahtevno metodologijo, če jo želimo resnično obvladati.

Glavno vodilo metodologije Scrum je preglednost in možnost prilagoditve (angl. *inspect and adapt*). Ker se v času razvoja lahko pojavijo presenečenja, ki lahko vplivajo na nadaljnji potek razvoja, in je skozi celoten proces potrebno stalno učenje ter spremljanje inovacij, Scrum poudarja nujnost kratkih ciklov razvoja, sprotno pregledovanje narejenega in nato prilagajanje ciljev izdelka in procesnih praks. Ta proces je potrebno ponavljati v neskončnost (Sutherland, 2010).

Scrum je zasnovan na osnovi teorije empirizma in združuje iterativni ter inkrementalni pristop. Teorija empirizma trdi, da se znanje pridobi z izkušnjami in odločitvami na podlagi znanih okoliščin. Postopek, ki vsakič zagotavlja sprejemljivo kakovost izdelka ali storitve, imenujemo definirano kontroliranje procesa (angl. *defined process control*). Definirano kontroliranje procesa uporabljamo tam, kjer pretirana kontrola ni potrebna, končni izdelek pa je dovolj kakovosten za trg. Če je izdelek nesprejemljive kakovosti, neuporaben in so stroški predelave previsoki, je potrebno zaradi kompleksnosti vmesnih aktivnosti uporabiti empirično kontroliranje procesa (ang. *empirical process control*). Na dolgi rok se izkaže, da je strošek uporabe empiričnega kontroliranja procesa cenejše kot predelava slabega izdelka, ki je nastala ob uporabi definiranega kontroliranja procesa. Obstajajo trije temelji, ki podpirajo izvajanje empiričnega kontroliranja procesa (Schwaber & Sutherland, 2011; Schwaber, 2004):

1. **Transparentnost:** Transparentnost pomeni, da morajo biti vsi vidiki procesa, ki vplivajo na rezultat, vidni vsem, ki so odgovorni za kontrolo. Na primer vsi udeleženci, tako tisti, ki delajo, kot tisti, ki sprejemajo rezultate dela, morajo razumeti, kaj pomeni definicija statusa »dokončano«.
2. **Pregled:** Različne vidike procesa je potrebno pogosto pregledati v smeri napredka glede na zastavljen cilj, zato da se lahko zaznajo neželena odstopanja. Takšni pregledi ne smejo ovirati delo, zato ne smejo biti prepogosti. Smiselno je, da jih na samem delovnem mestu izvajajo za to usposobljeni inšpektorji.

3. **Prilagoditev:** V kolikor so ugotovljena odstopanja izven sprejemljivih meja pri enem ali več vidikih procesa in bo zaradi tega končni izdelek nesprejemljiv, se mora proces spremeniti in prilagoditi. Po Scrumu obstajajo štiri možnosti pregleda in prilagoditve. Te možnosti so: **sestanek za planiranje cikla** (angl. *Planning*), **dnevni Scrum** (angl. *Daily Scrum*), **revizija cikla** (angl. *Sprint revision*) in **retrospektiva cikla** (angl. *Sprint retrospectiv*). (Schwaber & Sutherland, 2011). Več o tem v naslednjem poglavju, ki govori o procesu Scrum.

Primer kontrole empiričnega procesa je pregled napisane programske kode, ki se navadno zgodi, ko je neka zaključena enota funkcionalnosti končana. Koda se pregleda na podlagi določenih standardov in najboljših praks. Na podlagi podanih komentarjev in predlogov se lahko kodo popravi in izboljša. Kodo navado pregleduje razvijalec z izkušnjami, ki v celoti razume standarde in najboljše prakse (Schwaber, 2004).

Model Scrum je zasnovan tako, da čim bolj optimizira prilagodljivost, ustvarjalnost in produktivnost. Razvoj Scruma je strukturiran v ciklih (angl. *Iterations, Sprints*), ki so krajši od enega meseca. Cikli so fiksni in si sledijo eden za drugim. Na koncu vsakega cikla razvojni tim skupaj z lastnikom izdelka pregleda, kaj je bilo narejeno, in preveri, kaj bi bilo možno narediti v enem od naslednjih ciklov. Tim lahko na takšen način redno prejema povratne informacije od naročnika, saj mu izdelek v manjših delih ves čas predaja na vpogled. Za naročnika je najpomembnejša prednost metodologije Scrum ta, da se mu ob koncu vsakega cikla predstavi potencialni izdelek za izdajo in uporabo. Scrum torej omogoča predajo uporabnega izdelka ob koncu vsakega cikla, ki ima dejansko status »dokončano«, in naročniku zagotavlja potencialno delujočo verzijo izdelka ves čas razvoja (Sutherland, 2010).

V primeru programske opreme to pomeni (Sutherland, 2010), da:

- so zahteve integrirane v izdelek,
- je koda popolnoma testirana,
- je izdelek pripravljen za izdajo oz. predajo naročniku.

Scrum definira le tri Scrum vloge: **lastnik izdelka** (angl. *Product Owner*), **skrbnik metodologije** (angl. *Scrum Master*) in **razvojni tim** (angl. *Development Team*). Projektnega managerja Scrum ne predvideva, zato so naloge in odgovornosti managementa projekta porazdeljene med vse tri zgoraj naštete vloge. Osebe, ki te vloge opravljajo, so odgovorne za uspešno izvedbo projekta. Scrum največjo vlogo za odločanje in odgovornost nalaga razvojnemu timu. Planiranje posameznih zahtev, ki se opredelijo kot naloge, je tako popolnoma prepuščeno timu. Tim sam določi težavnost razvoja in potek reševanja naloge. Scrum tim je torej samoorganiziran tim, ki odločitve vedno sprejema na nivoju celotnega tima. Odgovornost nosi celotni tim in ne posameznik. Skrbnik metodologije, tudi Scrum mojster, je promotor metodologije Scrum in je zadolžen za njeno pravilno izvajanje,

razumevanje in upoštevanje s strani tima. Lastnik izdelka je zadolžen za urejanje in upravljanje seznama naročnikovih zahtev in ima celovit pogled na potek projekta in izdelek (Schwaber & Sutherland, 2011; Schwaber, 2004).

Prednosti metodologije Scrum so (Schwaber, 1997):

- V primerjavi s tradicionalnimi metodologijami, ki so oblikovane tako, da rešujejo nepredvidljivosti le v primeru nadgradnje izdelka, je Scrum v celoti prilagodljiva metodologija. Scrum omogoča obvladovanje sprememb tudi v času razvoja izdelka, kar podjetju omogoča, da spreminja izdelek in projekt v vseh fazah razvoja. Tako lahko izdelek predan naročniku pravočasno.
- Razvijalcem ponuja svobodo pri planiranju rešitev v celotnem razvoju izdelka, obenem pa nudi tudi priložnosti za učenje.
- Majhni timi so zaradi medsebojnega sodelovanja sposobni deliti tiho znanje o procesu razvoja. Na ta način je znotraj tima zagotovljeno okolje za prenos znanja.
- Objektno orientirana tehnologija predstavlja osnovo za metodologijo Scrum. Objekti in funkcionalnost ponujajo diskretno in obvladljivo okolje.

V nadaljevanju je podrobneje opisana metodologija Scrum skozi proces, vloge, dogodke, pravila in orodja, ki jih Scrum določa.

2.2.1 Scrum proces

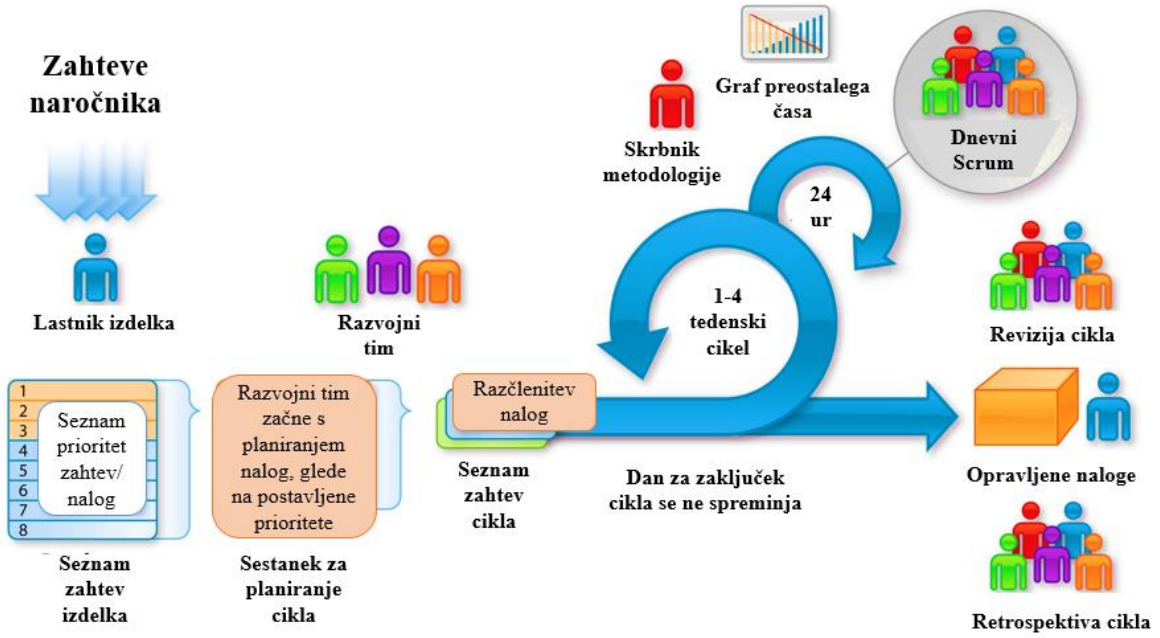
V tem poglavju je opisan proces dela po metodologiji Scrum in glavni dogodki, ki so definirani po teoriji. Predpisani dogodki se uporabljajo za ustvarjanje stalnosti oz. konstantnosti in zmanjševanje potreb po novih sestankih, ki jih Scrum ne določa. Dogodki so časovno omejeni oz. imajo maksimalno časovno okno, kar pomeni, da je za vsak dogodek predviden čas izvajanja. Dogodki ne smejo biti izpuščeni, saj to vodi v zmanjšanje transparentnosti in izgubo priložnost za pregled in prilagoditev (Schwaber & Sutherland, 2011).

Schwaber in Sutherland (2011) predpisujeta naslednje dogodke v procesu Scrum:

- sestanek za planiranje cikla,
- cikel,
- dnevni Scrum,
- revizija cikla,
- retrospektiva cikla.

Na Sliki 5 je prikazan celoten Scrum proces od prejema naročnikovih zahtev, pripravljanja seznama zahtev izdelka, postavljanja prioritetnih nalog, pripravljanja seznama zahtev cikla, do razvoja skozi izbran časovni cikel in vseh ostalih dogodkov, ki jih predvideva metodologija Scrum.

Slika 5: Scrum proces



Vir: Agileforall, Introduction to Agile, 2016

2.2.1.1 Sestanek za planiranje cikla

Sestanek za planiranje cikla izdela celoten Scrum tim in je namenjen pripravi nalog, ki se bodo izvajale v ciklu. Za pripravo enomesečnega cikla ima Scrum tim na voljo 8 ur. Za krajše cikle je planiranje sorazmerno krajše. Planiranje dvotedenskega cikla tako traja le 4 ure. Planiranje je razdeljeno na dva dela, ki sta časovno omejena na polovico časa trajanja planiranja cikla. Tim si mora na tem mestu odgovoriti na dve vprašanji (Schwaber & Sutherland, 2011):

1. Kaj bo narejeno v tekočem ciklu?
2. Kako bo delo izvedeno?

V prvem delu razvojni tim napove funkcionalnosti, ki bodo v tekočem ciklu končane. Lastnik izdelka glede na želje naročnika razporedi naloge po prioriteti in jih predstavi razvojnemu timu, nakar celotni tim sodeluje in uredi naloge, tako da so razumljive. Da lahko razvojni tim predvidi, katere naloge s seznama zahtev izdelka bo lahko dokončal, mora najprej določiti število točk, ki jih lahko v trenutnem ciklu opravi. Le razvojni tim lahko poda oceno o količini točk, ki jih lahko opravi v ciklu. Več o izračunu števila točk, ki jih lahko razvojni tim v enem ciklu opravi, je pojasnjeno v poglavju 2.2.5 Točke uporabniških zgodb (Schwaber & Sutherland, 2011). Lastnik izdelka in razvojni tim definirata tudi definicijo statusa »dokončano«, ki jo mora vsaka zahteva na koncu cikla izpolnjevati. Več o definiciji statusa »dokončano« v poglavju 2.2.1.6 Definicija statusa dokončano (Sutherland, 2010).

V drugem delu se razvojni tim odloči, kako bo v času cikla vgradil izbrano funkcionalnost v že delujoči izdelek. Lastnik izdelka je lahko v tem delu prisoten, zato da pomaga pri razumevanju izbranih nalog s seznama zahtev izdelka. Če razvojni tim ugotovi, da je nabor točk premajhen ali prevelik za en cikel, se lahko z lastnikom izdelka ponovno pogaja glede nalog na seznamu zahtev cikla. Če razvojni tim potrebuje tehnične nasvete z različnih področji, lahko povabi tudi druge ljudi, da pomagajo. Na koncu planiranja cikla mora razvojni tim vedeti, kako namerava kot samoorganiziran tim doseči cilj tega cikla, in to predstaviti lastniku izdelka ter skrbniku metodologije, če tako zahtevata (Schwaber & Sutherland, 2011).

Da lahko razvojni tim začne z planiranjem naslednjega cikla, mora najprej vedeti svojo kapaciteto, ki jo izračuna glede na hitrosti (angl. *velocities*) preteklih ciklov. Ko ima tim enkrat določeno kapaciteto, lahko začne z zahtevo, ki je najvišje na seznamu. Vsako zahtevo, ki je bila predhodno ocenjena s točkami, analizira in razbije na manjše zgodbe ter poda bolj natančno oceno časa, ki je potreben za razvoj, ki je ponavadi definiran v dnevih ali urah. Ob koncu sestanka ima razvojni tim pripravljen seznam z zahtevami in približno oceno ur, ki jih potrebuje za razvoj (Sutherland, 2010).

2.2.1.2 Cikel

Najbolj pomemben dogodek je cikel, v času katerega se ustvari razširitev, ki je »dokončana«, uporabna in tudi potencialno pripravljena za predajo naročniku. Cikel je časovno omejen in traja v času razvoja vedno enako dolgo. Vsak Scrum tim se sam odloči, kako dolgi bodo njihovi cikli. Običajno trajajo cikli en mesec ali manj. S predolgimi cikli se namreč poveča kompleksnost in je zato večje tveganje, da se bo opredelitev končnega izdelka spremenila. Ravno časovna omejenost omogoča boljšo preglednost in hitrejše prilagajanje napredka proti cilju. V času enega cikla se odvijejo sestanek za planiranje cikla, revizija cikla, retrospektiva cikla in dnevni sestanki, katerih število je določeno z dolžino posameznega cikla (Schwaber & Sutherland, 2011).

Med samim ciklom (Schwaber & Sutherland, 2011):

- se ne sme uvajati sprememb, ki bi lahko vplivale na cilj,
- razvojni tim mora ostati isti,
- ne smejo se zmanjševati cilji glede na kakovost,
- ko zahteve v ciklu postanejo bolj jasne, se lahko lastnik izdelka in razvojni tim ponovno pogajata o obsegu dela.

Vsak cikel mora imeti definicijo končnega izdelka, fleksibilen načrt izdelave ter vsebovati opravljeno delo in končni izdelek. Zato lahko tudi rečemo, da je vsak cikel pravzaprav manjši projekt, v okviru katerega razvojni tim v določenem časovnem obdobju razvije razširitev izdelka (Schwaber & Sutherland, 2011).

Ena od pomembnih lastnosti Scrum metodologije je obveza razvojnega tima do razvoja oz. zahtev, ki so izbrane za tekoči cikel. Vse spremembe morajo biti predložene šele v naslednjem ciklu. To pomeni, da v primeru, da se na sredini cikla lastnik izdelka odloči, da je potrebna razširitev, mora novo zahtevo uvrstiti v naslednji cikel (Sutherland, 2010). Cikel lahko prekliče le lastnik izdelka, in sicer samo v redkih primerih in tudi pred iztekom časovnega okvirja. To se lahko zgodi v primeru, da cilj zastara, ker je podjetje spremenilo usmeritev, se je spremenil trg ali tehnologija. Torej cikel se prekliče takrat, ko ni več smiselno razvijati naloge, ki so bile določene v tem ciklu. Po preklicu se pregledajo vse zaključene naloge, potencialno pripravljene za izdajo pa lahko lastnik izdelka tudi sprejme. Nedokončane naloge se vrnejo na seznam zahtev izdelka, kjer se jih ponovno oceni (Schwaber & Sutherland, 2011).

Razvojni tim mora dnevno posodabljati čas, ki je še na voljo za dokončanje posamezne naloge. Na podlagi vnesenih ur se posodablja graf preostalega časa izdaje. Graf je grafična predstavitev dokončanja nalog. Idealna linija grafa je padajoča, kjer graf na koncu doseže 0, kar pomeni, da je bil razvoj dokončan. Največkrat linija ni idealna, kar je realnost razvoja izdelka. Pomembno je, da lahko razvojni tim iz grafa razbere napredovanje cilja cikla. Ni pomembno, koliko časa je bilo porabljenega v preteklosti za razvoj, temveč koliko dela še ostaja v prihodnosti in kako daleč je razvojni tim od cilja (Sutherland, 2010).

Ob koncu cikla so bile nekatere naloge končane, nekatere so bili dodane, nekatere imajo novo oceno in nekatere so izpadle iz paketa pripravljenega za izdajo oz. predajo stranki. Lastnik izdelka je odgovoren, da so vse te spremembe dokumentirane. Vidne morajo biti tudi na seznamu zahtev cikla in na seznamu zahtev izdelka (Sutherland, 2010).

2.2.1.3 Dnevni Scrum

Dnevni Scrum je dogodek, ki poteka enkrat na dan, vedno ob istem času in na istem kraju. Časovno je omejen na 15 minut. Namen dnevnega Scruma je pregled in uskladitev aktivnosti za naslednji dan. Tim dnevno oceni napredek v smeri cilja cikla in optimizira verjetnost, da se cilj cikla doseže. Dnevni Scrum razvojnemu timu omogoča, da lastniku izdelka in skrbniku metodologije predstavi, kako namerava delovati kot samoorganiziran tim za dosego cilja v tekočem ciklu. Za redno dnevno izvajanje sestanka skrbi skrbnik metodologije. Vsak član razvojnega tima odgovori na tri vprašanja (Schwaber & Sutherland, 2011):

- Kaj je bilo narejeno od prejšnjega dnevnega Scruma?
- Kaj bo narejeno do naslednjega dnevnega Scruma?
- Ali obstajajo težave pri opravljanju nalog v tekočem ciklu?

Poudariti je potrebno, da dnevni Scrum ni statusni sestanek ali poročanje managerju, temveč je namenjen članom samoorganiziranega tima, da izmenjajo vsebinske dileme in si

pomagajo, da dosežejo zastavljen cilj cikla (Sutherland, 2010). Z dnevnim Scrumom se izboljšuje komunikacija, izniči se potreba po drugih sestankih in hitreje se prepoznajo ter odstranijo ovire pri razvoju, hkrati pa se spodbudi hitro sprejemanje odločitev in izboljša stopnja znanja razvojnega tima (Schwaber & Sutherland, 2011).

2.2.1.4 Revizija cikla

Ob koncu vsakega cikla naredi razvojni tim revizijo cikla, kjer skupaj z lastnikom izdelka pregleda, kaj je bilo v tem ciklu narejeno. Člani razvojnega tim razpravljajo o vseh spremembah, ki so se zgodile v času cikla, in z lastnikom izdelka diskutirajo o morebitnih naslednjih oz. ugotovljenih novih zahtevah. Ta sestanek je neformalen, njegov namen je predstaviti delo, narejeno v tem ciklu, in spodbuditi razpravo med člani tima ter lastnikom izdelka. Za enomesečni cikel ima razvojni tim na voljo štiri ure, za krajše cikle sorazmerno manj (Schwaber & Sutherland, 2011).

Schwaber in Sutherland (2011) navajata naslednje elemente cikla:

- Lastnik izdelka ugotovi, kaj je bilo »dokončano« in kaj ni bilo »dokončano«.
- Razvojni tim diskutira o problemih, na katere je naletel tekom cikla, o tem, kako so bili ti problemi rešeni, in o stvareh, ki so potekale dobro.
- Razvojni tim predstavi »dokončano« delo in odgovarja na vprašanja glede razširitve.
- Lastnik izdelka članom tima predstavi trenutno stanje seznama zahtev izdelka in predvidi roke za dokončanje izdelka glede na dosedanji napredek.
- Celoten Scrum tim odloča, kaj narediti v prihodnje, in s tem revizija cikla poskrbi za dragoceni prispevek pri naslednjem planiranju cikla.

2.2.1.5 Retrospektiva cikla

Retrospektiva cikla sledi reviziji cikla in je časovno omejena na tri ure za enomesečni cikel. Na retrospektivi cikla ima tim priložnost poiskati izboljšave načina dela in jih upoštevati pri planiranju in izvajanju naslednjega cikla. Skrbnik metodologije na tem dogodku spodbuja razpravo med člani razvojnega tima v smeri izboljšav procesa in prakse. Cilj je z vsakim naslednjim ciklom izboljšati razvoj izdelka in olajšati delo vsem članom razvojnega tima. Na koncu retrospektive naj bi tim določil izboljšave, ki jih bo uvedel v naslednjem ciklu. Retrospektiva je tako priložnost za pregled in prilagoditev procesa (Schwaber & Sutherland, 2011).

Schwaber in Sutherland (2011) omenjata 3 namene retrospektive:

1. Celoten pregled poteka zadnjega cikla v zvezi z ljudmi, odnosi, procesi in orodji.
2. Prepoznavanje glavnih zadev, ki so šle v ciklu dobro, in iskanje potencialnih izboljšav za naslednje cikle.
3. Izdelava načrta za uvedbo izboljšav v načinu dela v Scrum timu.

Sutherland (2010) predlaga, da si razvojni tim na tablo nariše štiri kolone:

1. Kaj je bilo v tem ciklu dobro?
2. Kaj bi v tem ciklu lahko bilo še bolje?
3. Katere stvari bi bilo morda smiselno poskusiti?
4. Problemi, ki jih je potrebno izpostaviti?

Vsak član razvojnega tima seznam dopolni in na koncu se naredi pregled ponavljajočih se predlogov oz. problemov. Razvojni tim poskuša nato skozi diskusijo najti odgovore in poiskati rešitve. Sutherland še predlaga, da razvojni tim vsako nalogo iz tega seznama označi kot C (angl. *caused*), E (angl. *exposed*) ali U (angl. *unrealted*). C označuje, da je bila naloga na tem seznamu posledica uporabe metodologije Scrum oz. te težave ne bi bilo, če ne bi bilo Scruma. E označuje nalogo, ki je bila odkrita zaradi Scruma, torej problem bi nastal neodvisno od Scruma, vendar ga je razvojni tim odkril zaradi uporabe Scruma. U označuje nalogo, ki ni povezana s Scrumom (Sutherland, 2010).

Skrbnik metodologije je lahko učinkovit spodbujevalec retrospektive, vendar pa Sutherland (2010) navaja, da je morda bolje, če retrospektivo vodi nevtralna oseba, ki ni del tima in lahko vzpostavi bolj sproščeno vzdušje. Avtor še predlaga, da se skrbniki metodologije zamenjajo in vodijo retrospektivo drugega razvojnega tima.

2.2.1.6 Definicija statusa dokončano

Ko je naloga s seznama zahtev izdelka ali razširitev funkcionalnosti opravljena, mora nositi tudi oznako »dokončano«, tako da lahko vsi razumejo, kaj »dokončano« pomeni. Vsak tim ima nekoliko drugačno definicijo statusa »dokončano«, ključno pa je, da vsak član znotraj tima dobro razume, kaj vključuje definicija statusa »dokončano«. Splošna definicija statusa definicija »dokončano« pravi, da so izpolnjene vse točke, ki jih je določil Scrum tim. Navadno status »dokončano« pomeni, da je bila koda pregledana in implementirana, napisana v skladu s standardi, testirana in dokumentirana. Status »dokončano« zagotavlja transparentnost in kakovost izdelka. Na ta način tim ugotavlja, kdaj je bila naloga ali razširitev končana. Takšen način dela omogoča tudi boljšo preglednost (Schwaber & Sutherland, 2011; Sutherland, 2010).

Z definicijo statusa »dokončano« si razvojni tim pomaga pri določanju nalog s seznama zahtev izdelka, ki jih bo uvrstil na seznam zahtev za naslednji cikel. Namen je ustvariti razširitev funkcionalnosti končne programske opreme, ki mora biti pripravljena za potencialno izdajo ter v skladu s trenutno definicijo statusa »dokončano«. V vsakem ciklu se opravi razširitev funkcionalnosti in je dodatek k vsem prejšnjim razširitvam. Po temeljitem testiranju, ki zagotavlja, da razširitev deluje, se lastnik izdelka odloči, ali bo izdelana razširitev ob koncu cikla tudi objavljena (Schwaber & Sutherland, 2011).

Z vsakim naslednjim ciklom se Scrum tim razvija in izboljšuje. Skladno s tem se pričakuje, da se definicija statusa »dokončano« dopolnjuje in izpopolnjuje v smislu strožjih meril in višje kakovosti (Schwaber & Sutherland, 2011).

2.2.2 Scrum vloge

Kot že omenjeno v prejšnjem poglavju, je Scrum tim sestavljen iz treh ključnih akterjev. To so lastnik izdelka, skrbnik metodologije in razvojni tim. Scrum tim je samoorganiziran tim in zato sam določa, kako najbolje opraviti delo. Dela torej ne narekuje nekdo izven tima. Tim je sestavljen tako, da imajo znotraj tima člani vse sposobnosti za dokončanje dela in se zato ne zanašajo na druge, ki niso člani tima. Scrum je zasnovan tako, da optimizira fleksibilnost, kreativnost in produktivnost. Glavno vodilo je, da tim skozi ves razvoj izdelka zagotavlja razpoložljivost potencialno uporabne verzije izdelka (Schwaber & Sutherland 2013).

2.2.2.1 Lastnik izdelka

V slovenski literaturi je bilo mogoče zaslediti dva prevoda za besedno zvezo »Product Owner«. Mahnič (Mahnič, Georgiev & Jarc, 2009) to besedno zvezo prevaja kot »predstavnik naročnika«. V Scrum vodiču (Schwaber & Sutherland, 2011), ki ga je prevedel Klofutar, pa najdemo izraz »lastnik izdelka«. V svojem magistrskem delu bom uporabljala izraz lastnik izdelka.

Lastnik izdelka je odgovoren za maksimiranje vrednosti izdelka, če je izdelek namenjen trženju. Njegova odgovornost ni maksimiranje vrednosti, ki bi vplivala na prihodek podjetja v primeru razvoja internega izdelka, pomembna je le učinkovita izvedba nalog v ciklu z najnižjimi stroški (Sutherland, 2010).

Lastnik izdelka torej zastopa naročnika, in sicer tako, da upošteva njegove želje oz. zahteve, ki jih nato ustrezno pretvori v manjše uporabniške zgodbe, jim določi prioriteto in poskrbi, da je naloga narejena v dogovorjenem ciklu. Je edina oseba, ki lahko ureja seznam zahtev izdelka in določa ali spreminja prioriteto uporabniškim zgodbam (Pries & Quigley, 2010).

Naloge lastnika izdelka so (Schwaber & Sutherland, 2011):

- Jasna opredelitev uporabniških zgodb s seznama zahtev izdelka.
- Razvrstitev uporabniških zgodb s seznama zahtev izdelka, tako da se kar najbolje doseže cilje in poslanstvo.
- Zagotovitev, da ima delo, ki ga razvojni tim opravlja vrednost.
- Zagotovitev, da je seznam zahtev izdelka vsem viden, pregleden in razumljiv, ter da kaže prihodnje delo Scrum tima.
- Zagotovitev, da razvojni tim dovolj dobro razume uporabniške zgodbe s seznama zahtev izdelka.

Zgoraj opisane naloge lahko opravlja tudi razvojni tim, vendar pa odgovornost vedno ostaja na lastniku. Lastnik izdelka je lahko le ena oseba. Vse odločitve lastnika izdelka morajo biti spoštovane in upoštevane, saj le tako lahko lastnik izdelka doseže uspeh. Razvojnemu timu lahko naloge dodeljuje edino lastnik izdelka in le on ima pravico spreminjati vrstni red prioritet (Schwaber & Sutherland, 2011).

Lastnik izdelka je lahko tudi naročnik sam, torej ista oseba. Najpogosteje se tak primer pojavi pri notranjih projektih, ko se Scrum uporablja za potrebe lastnega podjetja. V nekaterih primerih, kjer je število strank veliko in so potrebe različne, je vloga lastnika izdelka dostikrat podobna projektnemu managerju. Lastnik izdelka v agilni metodologiji se od tradicionalnega vodja razlikuje po pogostosti interakcije z razvojnim timom. Komunikacija z razvojnim timom poteka vsaj enkrat ali dvakrat mesečno (Sutherland, 2010).

2.2.2.2 Skrbnik metodologije

Tudi za besedo »Scrum Master« lahko najdemo dva prevoda. Mahnič (Mahnič, Georgiev & Jarc, 2009) jo prevaja kot »skrbnik metodologije«, medtem ko je v Scrum vodiču beseda prevedena kot »Scrum učitelj«.

Skrbnik metodologije je promotor metodologije Scrum. Njegova glavna naloga je nuditi pomoč razvojnemu timu. Skrbeti mora, da je Scrum metodologija (Pries & Quigley, 2010):

- uporabljena tako kot predvideva teorija,
- pravilno razumljena,
- dobro sprejeta v timu,
- pravilno upoštevana (tim se drži pravil, praks in teorije Scruma).

Skrbnik metodologije nudi pomoč razvojnemu timu tako, da (Schwaber & Sutherland, 2011):

- poučuje razvojni tim na področju samoorganizacije,
- poučuje in vodi razvojni tim pri ustvarjanju visoko kakovostnih izdelkov,
- odstranjuje ovire, ki ovirajo prakticiranje Scruma kot predpisuje teorija,
- omogoča Scrum dogodke kot je zahtevano oz. potrebno,
- poučuje razvojne time, kjer Scrum še ni popolnoma razumljen in uveden.

Skrbnik metodologije služi tudi lastniku izdelka tako, da (Schwaber & Sutherland, 2011, str. 7):

- išče tehnike za učinkovito upravljanje seznama zahtev izdelka,
- jasno predstavi vizije, cilje in naloge s seznama zahtev izdelka razvojnemu timu,
- poučuje razvojni tim, kako ustvariti jasne uporabniške zgodbe na seznamu zahtev izdelka,

- razume dolgoročno planiranje v empiričnem okolju,
- razume in prakticira agilnost.

Za razliko od projektnega managerja, skrbnik metodologije ne izdaja nalog in ne razporeja ljudi na projektu oz. jim ne narekuje, kaj na počnejo. To počne Scrum tim sam, skrbnik metodologije timu le pomaga pri tem. Torej ni projektni manager, temveč je timu na voljo po potrebi. Skrbi za izobraževanje tima in ne dovoli vmešavanja tretjih oseb, ki niso člani tima. Njegova glavna naloga je, da poskrbi, da vsi člani Scrum tima razumejo in sledijo agilni metodologiji. Skrbnik metodologije mora biti predan svojemu delu in ga opravljati z zanosom. Le tako lahko učinkovito pomaga timu pri reševanju problemov. V nasprotnem primeru bo imel razvojni tim ali lastnik izdelka težave pri uspešnosti razvoja. Skrbnik metodologije nikoli ne more biti lastnik izdelka, lahko pa se zgodi, da je pri manjših timih to tudi oseba iz razvojnega tima z manjšim naborom dodeljenih nalog. To se običajno dogaja samo v manjših Scrum timih (Sutherland, 2010).

2.2.2.3 Razvojni tim

Razvojni tim skrbi, da se lastniku izdelka na koncu vsakega cikla zagotovijo nove razširitve izdelka, hkrati pa ves čas zagotavlja potencialno verzijo za izdajo. Le člani razvojnega tima ustvarjajo razširitve. Razvojni tim ima možnost, da se sam organizira in vodi svoje delo, kar zagotavlja večjo učinkovitost in uspešnost razvojnega tima (Schwaber & Sutherland, 2011).

Schwaber in Sutherland (2011) omenjata naslednje značilnosti razvojnega tima:

- Samoorganiziranost: Nihče (niti skrbnik metodologije) ne določa razvojnemu timu, kako uporabiti seznam zahtev izdelka za ustvarjanje razširitve funkcionalnosti, ki je potencialno pripravljena za izdajo.
- Razvojni timi so navzkrižno funkcionalni. Imajo vse sposobnosti, izkušnje in znanja, ki so potrebna za ustvarjanje razširitve izdelka.
- Scrum razvojnemu timu ne priznava nobenih nazivov, razen naziva razvijalec, ne glede na delo, ki ga oseba opravlja. Izjeme ne obstajajo.
- Posamezni člani razvojnega tima imajo lahko specializirane sposobnosti s področji na katera se osredotočajo, vendar je razvojni tim odgovoren kot celota.
- Razvojni timi ne vključujejo skupin, ki bi bile namenjene posebnim področjem, npr. testiranje ali poslovna analiza.

Schwaber in Sutherland (2011) predlagata, da je optimalna velikost razvojnega tima 3 do 9 članov. Če je razvojni tim majhen, lahko pride do nižje produktivnosti, ker se zmanjša interakcija med člani. Taki tim se srečujejo z omejitvami sposobnosti, kar jih nadalje omejuje pri ustvarjanju razširitve, ki je potencialno pripravljena na izdajo. V večjih ekipah pa se hitro pojavi problem koordinacije delovnega procesa. Lastnik izdelka in skrbnik metodologije sta všteta v velikost tima le takrat, ko sta vključena v reševanje zahtev s seznama zahtev izdelka.

Za razvoj programske opreme lahko razvojni tim sestavljajo razvijalci, arhitekti programskih rešitev in testerji. Tim razvija izdelek in predstavlja ideje lastniku izdelka, kako razviti ali izboljšati izdelek. Le 100 % zavezanost tima k delu na ciklu enega projekta lahko vodi do uspešnosti razvojnega tima, zato je nezaželeno delo na večjih projektih. Prav tako je nezaželeno menjavanje članov ekipe, saj lahko to vodi v nestabilnost ekipe in do zmanjšanja produktivnosti (Sutherland, 2010).

2.2.3 Scrum orodja

2.2.3.1 Seznam zahtev izdelka (angl. *Product backlog*)

Seznam zahtev izdelka, ki je narejen na podlagi zahtev naročnika, je osnova za razvoj. Lastnik izdelka, ki zastopa interese naročnika, je odgovoren za vzdrževanje, urejanje in razvrščanje zahtev na seznamu. V času izvajanja projekta se seznam zahtev izdelka dopolnjuje in spreminja, tako da določa tisto, kar izdelek potrebuje, in je na ta način lahko konkurenčen in uporaben. Lastnik izdelka po potrebi dodaja nove naloge in jih prioritetno razvršča skladno s trenutnimi potrebami uporabnikov (Schwaber & Sutherland, 2011).

Glavne tri lastnosti nalog na seznamu zahtev izdelka so opis, vrstni red in ocena težavnosti razvoja zahteve. Seznam je običajno razporejen po vrednosti, tveganju, prioriteti in nujnosti. Takojšne razvojne aktivnosti določajo naloge, ki so prve (najvišje) na seznamu. Naloge, ki so višje razvrščene, se podrobneje obravnavajo in o njih in njihovi vrednosti obstaja večje soglasje. Te naloge so tudi bolj jasne in bolj podrobno opisane ter posledično natančneje ocenjene. Zahtevki, ki bodo šli v razvoj v naslednjem ciklu, so še bolj podrobno opisani in razčlenjeni tako, da se lahko dokončajo v časovnem okviru enega cikla. Vse naloge, ki so lahko dokončane v okviru enega cikla, imajo oznako »pripravljeno« in se jih lahko izbere na sestanku za planiranje cikla. Kljub temu, da lahko številni Scrum timi delajo na istem izdelku, je seznam zahtev izdelka vedno le eden. Naloge se v tem primeru razporedi glede na določene lastnosti (Schwaber & Sutherland, 2011).

Torej naloga lastnika izdelka je, da seznam zahtev izdelka konstantno posodablja tako, da so prikazane trenutne spremembe naročnikovih želja, nove ideje, spoznanja in morebitne tehnične ovire, ki se pojavljajo (Sutherland, 2010). Razvojni tim je odgovoren za ocene nalog na seznamu in je tisti, ki poda začetno oceno napora. Lastnik izdelka te ocene ne sme spreminjati, timu lahko le pomaga pri razumevanju naloge in iskanju oz. sklepanju kompromisa. Lastnik izdelka in razvojni tim sodelujeta le pri podrobnostih nalog s seznama zahtev izdelka. Naloge skupaj pregledujeta in jih po potrebi dopolnjujeta (Schwaber & Sutherland, 2011). Poleg začetne ocene napora (angl. *initial estimate of effort*), ki jo lastnik izdelka pridobi od razvojnega tima, mora lastnik izdelka podati še poslovno oceno vrednosti (angl. *business value*) posamezne naloge na seznamu zahtev izdelka. Ta praksa je lastnikom izdelka pogosto nepoznana. Z začetno oceno napora in oceno vrednost, kot je prikazano na Sliki 6, in morda še z dodatno oceno tveganja, lastnik izdelka postavlja prioritete, tako da

maksimira donosnost (angl. *ROI*) projekta. Po drugi strani skuša na ta način omejiti večja tveganja (Sutherland, 2010).

Slika 6: Primer seznama zahtev izdelka

					Nove ocene napora		
					Preostali cikli		
Naloga	**Podrobno (wiki URL)**	**Prioriteta**	**Ocena vrednosti**	**Začetna ocena napora**	**1**	**2**	**3**
Kot kupec želim dodati knjigo v nakupovalno košarico (glej skico na UI na Wiki ...)	...	1	7	5			
Kot kupec želim odstraniti knjigo iz nakupovalne košarice	...	2	6	2			
Izboljšaj hitrost procesiranja transakcij (poglej matriko na wiki)	...	3	6	13			
Preuči rešitve za pohitritve validacije kreditnih kartic	...	4	6	20			

Vir: J. Sutherland, Scrum Handbook, 2010, str.26

Metodologija Scrum ne določa vrste ocenjevanja. Najpogosteje se uporablja relativno ocenjevanje, in sicer s točkovanjem namesto absolutnih vrednosti v obliki števila ur. Razvojni tim lahko na podlagi točkovanja skozi več ciklov izmeri hitrost dela v obliki števila povprečno rešenih točk na posamezen cikel. Tim tako lažje napove datum pričakovane izdaje končnega izdelka in roke za realizacijo posameznih funkcionalnosti. V praksi se pri uporabi Scruma razvojni tim sam odloči in zaveže, koliko zahtev bo razvil v določenem ciklu. Takšen način dela je bolj zanesljiv, saj razvojni tim na podlagi lastne analize in planiranja določi, kaj lahko v enem ciklu naredi. Zahtev torej ne narekuje tretja oseba. V okviru Scruma tudi ni določeno, kako podrobno je treba posamezno zahtevo definirati. O tem, kako podrobno je treba zahtevo zapisati, odločata lastnik izdelka in tim. Najbolj smiselno je zapisati zadosti, da lahko vsi člani tima razumejo, kaj se pri posamezni nalogi zahteva od posameznega člana tima (Sutherland, 2010).

2.2.3.2 Seznam zahtev cikla (angl. *Sprint Backlog*)

Naloge s seznama zahtev izdelka, ki so vsebinsko dopolnjene, se uvrstijo na seznam zahtev cikla glede na prioritete, ki jih določi lastnik izdelka. To so naloge, ki bodo narejene v naslednjem ciklu in bodo prinesle novo posodobitev končne programske rešitve oz. izdelka. Razvojni tim skozi celoten cikel spreminja seznam zahtev cikla, ko z načrtom spozna vse več o potrebnem delu za dokončanje posamezne naloge in dejavnike, ki so potrebni za doseganje cilja cikla. Če se zahteva novo delo, ga razvojni tim doda na seznam zahtev cikla. Ko je delo končano, se oceno preostalega časa posodobi. Če se ugotovi, da nekateri elementi

naloge niso potrebni za razvoj, jih razvojni tim odstrani z načrta. Tekom cikla lahko le razvojni tim spreminja seznam zahtev cikla. Seznam zahtev cikla podaja jasno sliko o delu, ki ga bo razvojni tim opravil v ciklu, in pripada izključno razvojnemu timu (Schwaber & Sutherland, 2011).

2.2.3.3 Uporabniške zgodbe

Scrum ne predpisuje oblike zahtev na seznamu zahtev izdelka. Uporabniške zgodbe izvirajo iz agilne metodologije ekstremnega programiranja. Vseeno lahko uporabniške zgodbe uporabimo za zahteve pri drugih procesih. Omogočajo boljši pregled in s tem lažjo razvrstitev prednostnih zahtev. Vse zahteve naročnika je potrebno pretvoriti v obliko uporabniških zgodb. Gre za kratek opis delovanja in možnosti uporabe funkcionalnosti tako, da je razumljiva razvijalcem in naročniku. Navadno je uporabniška zgodba sestavljena iz 3 delov (Cohn, 2004):

1. opis zahteve, ki služi za planiranje in kot opomnik,
2. razne opombe, ki služijo kot pomoč pri definiranju uporabniške zgodbe,
3. sprejemni testi, ki se lahko uporabijo za ugotavljanje, kdaj je razvoj uporabniške zgodbe končan.

Med sestankom za planiranje cikla se lastnik izdelka in razvojni tim pogovorita o najpomembnejših uporabniških zgodbah in sestavita seznam nalog, ki jih bo razvojni tim implementiral v naslednjem ciklu. Uporabniške zgodbe se razbije na manjše smiselne naloge, saj to omogoča boljši pregled in bolj sistematičen razvoj. Na dnevnem sestanku ima razvojni tim tako boljšo predstavo o tem, kaj je končni cilj cikla, in lažje ugotavlja, kaj je bilo že narejeno in kaj še ni dokončano (Cohn, 2004).

Uporabnik oz. naročnik zahteve mora biti prisoten pri celotnem razvoju uporabniške zgodbe. V sam proces je vpleten že od začetka, saj se od njega zahteva aktivno sodelovanje pri pisanju in definiciji uporabniških zgodb. Dostikrat vlogo pisanja prevzame lastnik izdelka (Cohn, 2004).

Pisanju uporabniških zgodb je največkrat namenjen poseben sestanek. Ko so uporabniške zahteve definirane, je potrebno počakati, da jih lastnik izdelka razvrsti glede na prioritete . Ob pisanju uporabniških zgodb je potrebno zahtevo čim bolje definirati in zapisati načrt razvoja ter vse morebitne dodatne opombe. Največkrat ne moremo takoj odgovoriti na vsa vprašanja in preveriti vse podrobnosti, zato se nadaljnje odgovore išče v času samega razvoja ali v obliki neformalnih pogovorov z lastnikom izdelka (Cohn, 2004).

2.2.4 Hitrost

Hitrost (angl. *Velocity*) je skupna vsota vseh uporabniških točk, ki jih lahko razvojni tim opravi v enem ciklu. Točkovanje je podrobneje opisano v naslednjem podpoglavju. Če lahko

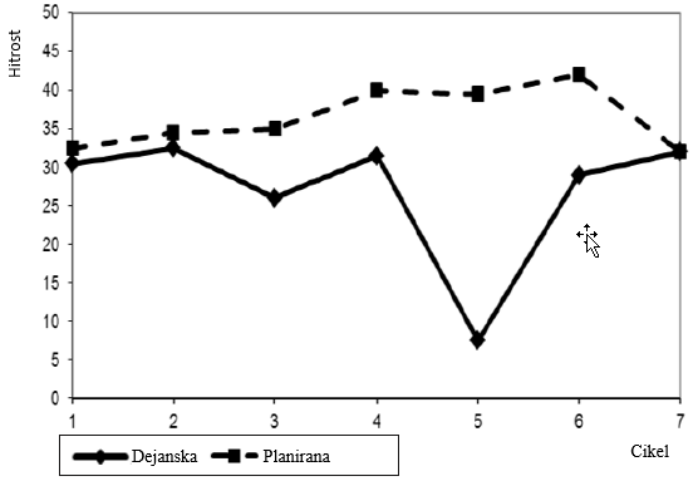
razvojni tim dokonča 3 uporabniške zgodbe z oceno 5, to pomeni, da je hitrost razvoja v tem ciklu 15. To je pomemben podatek, saj lahko iz tega razvojni tim sklepa, da bo tudi v naslednjem ciklu po vsej verjetnosti imel hitrost 15. Na ta način lahko tim lažje spremlja in napoveduje napredek dela na projektu (Cohn, 2005).

Če tim uporabi točke uporabniških zgodb in podatke o hitrosti razvoja, lahko lažje oceni, koliko uporabniških zgodb lahko vzame v naslednji cikel. V nadaljevanju lahko tim izračuna število ciklov, ki so potrebni za dokončanje projekta. Hitrost lahko določimo na 3 načine: uporabimo zgodovinske podatke, izmerimo glede na cikel ali naredimo predpostavko (Cohn, 2005).

Če lahko pridobimo zgodovinske podatke, je to najlažje. Težje pa je v tem primeru najti projekta, ki sta si podobna; enak projektni tim, enaka tehnologija, enaka orodja, podoben naročnik itd. V primeru neujemanja lahko pride do dileme, ali lahko zgodovinske podatke sploh uporabimo in kako zanesljiva je določitev hitrosti na podlagi zgodovinskih podatkov (Cohn, 2005).

Druga možnost je, da razvojni tim skozi cikle ugotovi svojo hitrost razvoja. V prvih ciklih napovedovanje hitrosti razvoja še nima smisla, temveč je bolje, da tim opazuje napredovanje in se odloči za napovedovanje šele takrat, ko opravi že nekaj ciklov. Tim se mora zavedati, da so odstopanja pri prvih ciklih večja kot sicer (Cohn, 2005). Primerjava med planirano hitrostjo in dejansko hitrostjo je prikazana na Sliki 7, iz katere je lepo razvidno, da je bil razvojni tim v prvih ciklih preveč optimističen, kar je razumljivo, saj razvojni tim ni imel predhodnih izkušenj (Mahnič & Žabkar, 2012).

Slika 7: Primerjava med planirano in dejansko hitrostjo

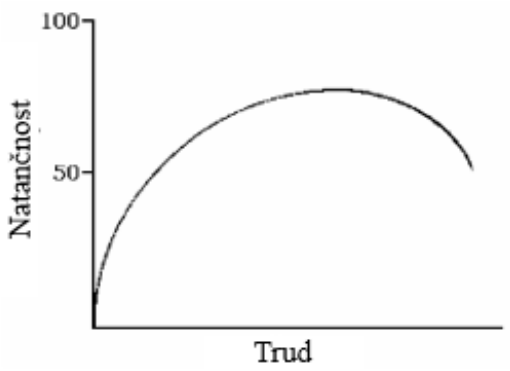


Vir: V. Mahnič & N. Žabkar, Measuring progress of scrum-based software projects, 2012, str.74

2.2.5 Točke uporabniških zgodb

S številom točk, ki so merska enota, ocenimo tehnično in časovno zahtevnost oz. velikost ene uporabniške zgodbe. Tim sam določi enoto merjenja (ponavadi so ure ali dnevi), dodeljena ocena pa predstavlja zahtevnost posamezne naloge in obseg dela. Če ima neka uporabniška zgodba oceno 4, druga pa 2, potem pomeni, da je prva dvakrat zahtevnejša od druge. Za dodeljevanje ocen obstaja več metod, ni pa predpisanega posebnega pravila ali točno določene formule. Ob ocenjevanju je potrebno upoštevati trud za razvoj in testiranje, kompleksnost naloge in tveganja ob razvoju. Ni nujno, da se s povečanjem časa ocenjevanja in vložka truda izboljša ocena natančnosti. Povezava med natančnostjo ocene (angl. *accuracy*) in vloženim trudom je prikazana na Sliki 8. Po določeni točki vsak dodaten trud doprinese zelo malo k večji natančnosti ocene. Ni važno, koliko truda je vloženega, ocena zahtevnosti nikoli ni na vrhu y osi. Prav tako pa je potrebnega zelo malo vloženega truda za izboljšanje natančnosti ocene. Zato je najbolje najti razmerje, ki razvojnemu timu doprinese kar se da natančno oceno za določeno uporabniško zgodbo. Koliko truda naj bo vloženega v ocenjevanje, je odvisno od samega namena ocenjevanja. Agilni timi sledijo ideji, da lahko z malo vloženega truda dosežejo veliko, zato se glede na Sliko 8 pozicionirajo bolj na levo. Zavedati se je potrebno, da je ne glede na to, koliko truda je vloženega v ocenjevanje, ocena na koncu le ocena, ki nikoli ni popolnoma točna (Cohn v Carlson, 2013; Cohn, 2005).

Slika 8: Povezava med natančnostjo ocene in vloženim trudom



Vir: M. Cohn, Agile Estimating and Planning, 2005, str. 50

Čeprav je splošno znano, da so najbolj natančne ocene podane s strani oseb, ki bodo nalogo tudi razvile, v agilnih timih ni vnaprej znano, kdo bo nalogo opravil. Ravno zaradi tega je pomembno, da vsi člani razvojnega tima sodelujejo pri ocenjevanju. Na koncu je važno, da ima razvojni tim vsaj neko okvirno oceno, lahko je to tudi interval. Iz ocene 30 lahko tim naredi interval 20-50. Ta informacija res ni natančna, vendar je bolje imeti ohlapno oceno, kot natančno oceno, ki se izkaže za napačno. Najbolj učinkovito je ocenjevanje stvari istega reda, torej ko ocene združimo v celoto. Najbolj pogosti izbiri za časovno ocenjevanje sta Fibonaccijevo zaporedje (1, 2, 3, 5, 8, 13, 20, 40, 100 itd.) in zaporedje, kjer je naslednik

dvakratnik predhodnika (1, 2, 4, 8, 16 itd.). Takšno ocenjevanje daje boljšo predstavo o zahtevnosti naloge (Cohn, 2005; Williams, 2010).

Splošno najbolj uporabljene metode za ocenjevanje so (Cohn, 2005):

- **Mnenje strokovnjaka**: Ta ocena je najhitrejša, saj je potrebno vprašati le eno osebo, ki ima strokovno znanje in lahko poda odgovor na podlagi svojih izkušenj, znanja in intuicije. Pri agilnih metodah je ta način malo v uporabi, saj so uporabniške zgodbe oblikovane na način, ki zahteva več funkcionalnosti, ki niso nujno v primarni domeni eksperta, saj lahko pri razvoju sodeluje več oseb. Metoda je bolj primerna za tradicionalne projekte.
- **Ocenjevanje po analogiji**: Pri tej metodi se za pridobitev ocene primerjata vsaj dve uporabniški zgodbi. Če uporabniške zgodbe niso enako obsežne, jih skušamo smiselno oceniti, glede na kriterij, ali je od neke zgodbe večja ali manjša. Pri ocenjevanju si pomagamo s tako imenovano triangulacijo, kar pomeni, da uporabniško zgodbo ocenimo na podlagi primerjave z več podobnimi zgodbami. Če želimo uporabniški zgodbi dati oceno 5, preverimo, ali je zgodba obsežnejša od tistih, ki že imajo oceno 3, in manj obsežna od tistih, ki imajo oceno 8.
- **Členitev**: V tem primeru uporabniške zgodbe razdelimo na več manjših obvladljivih delov, saj lahko tako z večjo gotovostjo napovemo oceno. Pri tem moramo biti pozorni, da deli niso premajhni, ker se lahko zgodi, da zgubimo sliko nad celotno oceno.

V nadaljevanju sta bolj podrobno opisani še dve metodi za ocenjevanje nalog, poker planiranje in igra ocenjevanja nalog.

2.2.5.1 Poker Planiranje

Poker planiranje (angl. *Planning poker*) je metoda za ocenjevanje uporabniških zgodb in je kombinacija vsega, kar smo našteli v prejšnjem poglavju. Pri ocenjevanju sodelujejo vsi člani razvojnega tima. Ocenjevanje poteka s pomočjo kart, na katerih so napisana števila, ki ponazarjajo zahtevnost uporabniške zgodbe. To je tudi najbolj zanesljiva metoda ocenjevanja (Cohn, 2005; Williams, 2010).

Ocenjevanje poteka na naslednji način (Cohn, 2005):

- Vsak od ocenjevalcev ima v rokah kup vnaprej pripravljenih kart z veljavnimi ocenami.
- Moderator, ki je ponavadi lastnik izdelka, prebere uporabniško zgodbo, ki je bila predhodno že definirana in zapisana glede na dogovorjeno predlogo.
- Temu navadno sledijo vprašanja o razvoju, implementaciji in morebitnih nejasnostih zahteve. Lahko se razvije krajša diskusija, katere namen ni do potankosti analizirati nalogo, temveč jo le na kratko razložiti.
- Ko so vsa vprašanja odgovorjena, sledi ocenjevanje, tako da vsak izbere oceno iz kupa kartic.

- Dvigovanje kartic poteka sočasno, zato da vsak poda oceno na podlagi lastnih občutkov in ni pod vplivom ocene drugega ocenjevalca.
- Največkrat se zgodi, da ne podajo vsi enake ocene, zato sledita argumentirani utemeljitvi ocenjevalca z najvišjo in ocenjevalca z najnižjo oceno.
- Na ta način lahko vsak poda svoj vidik ocenjevanja in velikokrat se zgodi, da kateri od ocenjevalcev ni upošteval vseh dejavnikov ali pa obstaja lažji način razvoja, na katerega ni noben pomislil.
- Po diskusiji se ocenjevanje za isto uporabniško zgodbo ponovi, dokler tim ne pride do poenotene ocene.

Cilj poker planiranja je, da se ocenjevalci zedinijo glede določene ocene, si izmenjajo mnenja in izpostavijo stvari, na katere drugi ni pomislil. Ni nujno, da vsi podajo enako oceno, dovolj je, če moderator opazi, da so si ocene zelo blizu in tako predlaga timu, da se sporazumno odločijo za končno oceno. Pogoj je, da se vsi strinjajo (Cohn, 2005).

Pogovor med ocenjevalci je ključnega pomena pri poker planiranju. Različni avtorji predlagajo različne časovne okvirje. Najbolj zanimiv način je omejevanje časa z dvominutno peščeno uro, ki se postavi na sredino mize. V primeru, da nekdo želi nekaj argumentirati, peščeno uro obrne. Zagovorniki te tehnike navajajo, da peščene ure ni potrebno obrniti več kot trikrat. Poker planiranje je časovno omejeno od ene do treh ur. V primeru, da je potrebno oceniti več uporabniških zgodb, je smiselno, da ima razvojni tim več sestankov, namenjenih ocenjevanju. Ocenjevanje navadno poteka na začetku projekta, ko so znane prve zahteve. Ko se tekom cikla pojavijo nove uporabniške zgodbe, jih razvojni tim ocenjuje po potrebi oz. pred začetkom novega cikla. Ocenjevanje je možno tudi v manjših skupinah, tako da se razvojni tim razdeli na manjše skupine, ki pa morajo imeti vsaj tri člane. Tak način je uporaben predvsem takrat, ko je potrebno oceniti veliko število uporabniških zgodb (Cohn, 2005).

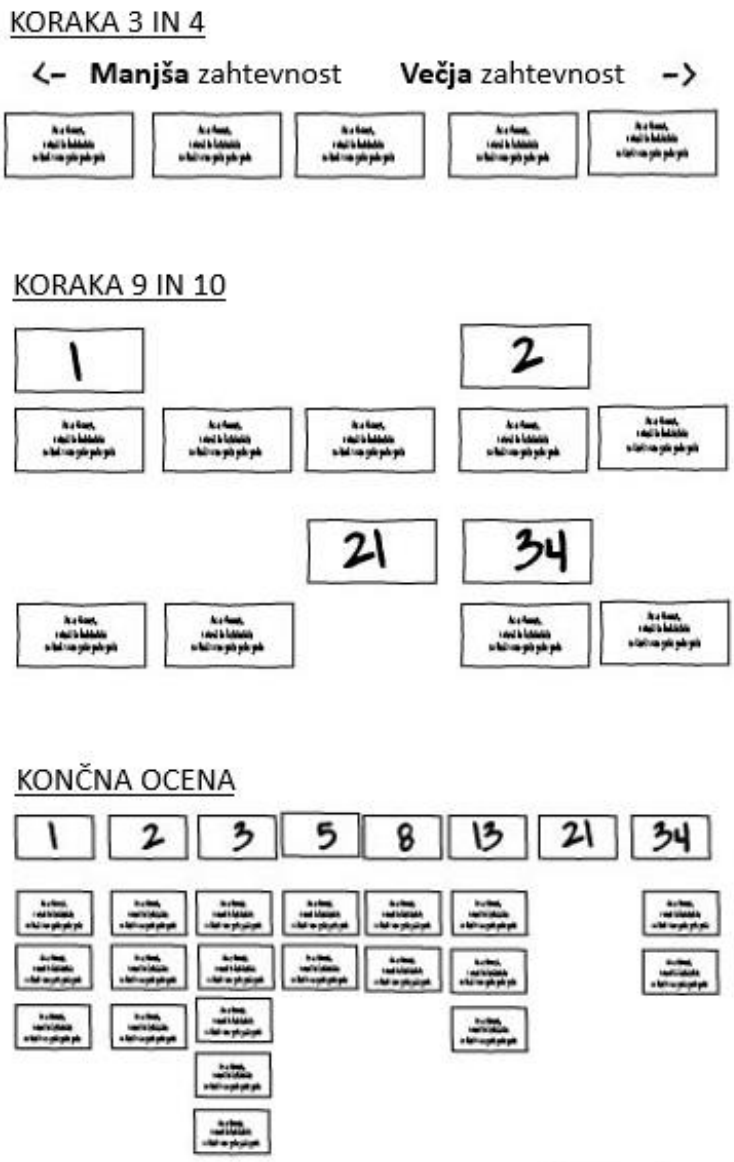
Taka tehnika ocenjevanja je dobra praksa iz 3 razlogov. Prvi razlog je nedvomno v tem, da metoda združi mnenja več strokovnjakov z različnih področij. S tem je ocenjevanje bolj natančno. Drugi razlog je v diskusiji, ki se vname med igranjem in ob utemeljevanju ocen, ki izstopajo. To pripomore k temu, da se odkrijejo skrite oz. manjkajoče informacije, na katere ostali niso pomislili. Zelo uporabna je takrat, kadar so uporabniške zgodbe napisane bolj splošno ali so nejasne. Kot tretje, pa so študije pokazale, da iskanje povprečja med posameznimi ocenami in skupinsko ocenjevanje prinašata boljše rezultate. Nenazadnje je uporaba poker planiranja lahko tudi zabaven pristop k ocenjevanju, ki je ponavadi nepriljubljena aktivnost med člani razvojnega tima (Mahnič, 2011).

2.2.5.2 Igra ocenjevanja nalog

Druga pogosta metoda ocenjevanja je igra ocenjevanja nalog (angl. *Team estimation game*). S to tehniko ocenjevanja lahko razvojni tim v eni uri oceni od 20 do 60 uporabniških zgodb.

Potek ocenjevanja je razdeljen na dva dela in si sledi po korakih, prikazanih na Sliki 9 (Johnson, 2012).

Slika 9: Prikaz poteka ocenjevanja po korakih



Vir: H.L. Johnson, How to play the Team Estimation Game, 2012

PRVI DEL OCENJEVANJA

1. Za ocenjevanje je potrebna ocenjevalna površina. To je lahko dolga prazna miza, tabla ali računalnik, če imamo na voljo primerno elektronsko orodje. Površino ponavadi pripravi skrbnik metodologije.
2. Lastnik izdelka pripravi vse uporabniške zgodbe, ki jih je potrebno oceniti.
3. Prvi izmed ocenjevalcev vzame uporabniško zgodbo, jo prebere ostalim članom razvojnega tima in jo nato postavi na sredino površine.

4. Drugi ocenjevalec vzame naslednjo uporabniško zgodbo in jo uvrsti levo ali desno, glede na že ocenjeno prvo uporabniško zgodbo. Levo pomeni, da je zahtevnost naloge manjša, desno, da je zahtevnost naloge večja od že ocenjene prve zgodbe. Uporabniško zgodbo je možno postaviti tudi med dve že ocenjeni uporabniški zgodbi, če je ocenjevalec mnenja, da je zahtevnost njegove uporabniške zgodbe ravno med njima.
5. Koraka 3 in 4 se ponavljata toliko časa, dokler niso razporejene vse uporabniške zgodbe.
6. Med ocenjevanjem uporabniških zgodb razvojni tim nadaljuje z ocenjevanjem, tako da premika vrstni red uporabniških zgodb, dokler se vsi ocenjevalci z vrstnim redom ne strinjajo. Med spreminjanjem vrstnega reda lahko vsak na kratko poda argument za spremembo. Če se ocenjevalec strinja s postavljenim vrstnim redom, lahko premikanje preskoči.
7. Točko 6 mora razvojni tim ponavljati toliko časa, dokler se vsi ocenjevalci z vrstnim redom ne strinjajo.

DRUGI DEL OCENJEVANJA

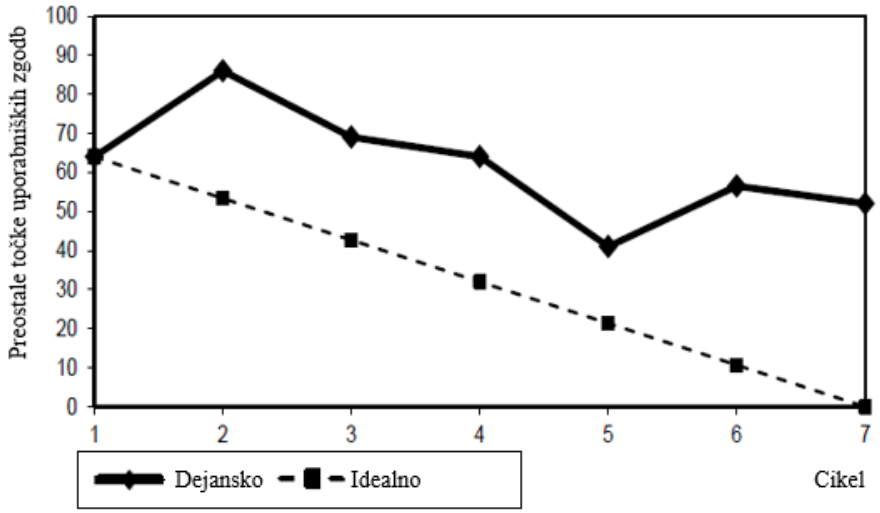
8. V drugem delu je potrebno vsem uporabniškim zgodbam določiti zahtevnost izvedbe naloge. V tem primeru se največkrat uporabi Fibonaccijevo zaporedje. Razvojni tim , enako kot pri poker planiranju, določi, kakšno zahtevnost predstavlja posamezna ocena.
9. Prvi ocenjevalec vzame kartico 1 in jo pristavi k uporabniški zgodbi, za katero meni, da velja zahtevnost 1.
10. Drugi ocenjevalec vzame kartico 2 in določi izbrani uporabniški zgodbi zahtevnost 2.
11. Koraka 9 in 10 se ponavljata toliko časa, dokler vse kartice z ocenami zahtevnosti niso porabljene. Pri dodeljevanju zahtevnosti se lahko vrstni red uporabniških zgodb zopet nekoliko spremeni, če je ocenjevalec mnenja, da si uporabniške zgodbe ne sledijo glede na zahtevnost.
12. Razvojni tim ponavlja točko 11, dokler vsi ocenjevalci niso zadovoljni z vrstnim redom ter ocenami zahtevnosti uporabniških zgodb.

2.2.6 Nadzor poteka dela

Za nadzor poteka dela razvojni tim uporablja naslednja merila: hitrost (angl. *Velocity*) in količina preostalega časa, ki ga predstavljata graf preostalega časa izdaje (angl. *Release burndown chart*) in graf preostalega časa cikla (angl. *Sprint burndown chart*) (Mahnič & Žabkar, 2012).

Kot omenjeno že v prejšnjem poglavju je hitrost vrednost, ki predstavlja količino zaključenega dela v vsakem ciklu in je izražena s seštevkom točk zaključenih uporabniških zgodb. Razvojni tim lahko na podlagi te vrednosti spremlja in napoveduje napredek na projektu. Razvojni tim oceni planirano hitrost in na podlagi te ocene na začetku vsakega cikla določi število uporabniških zgodb, ki jih lahko vzame v en cikel (Mahnič & Žabkar, 2012).

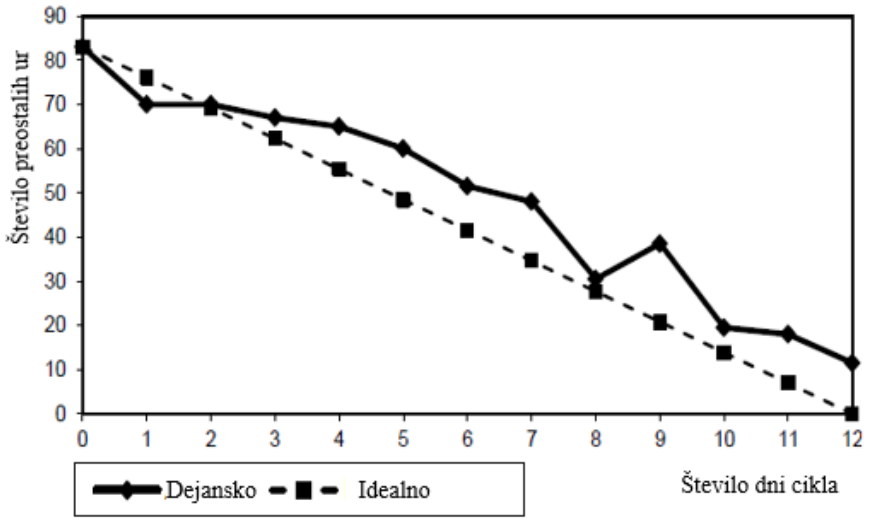
Slika 10: Primer grafa preostalega časa izdaje



Vir: V. Mahnič & N. Žabkar, Measuring progress of scrum-based software projects, 2012, str. 75

Graf preostalega časa izdaje na Sliki 10 prikazuje količino preostalega dela od začetka posameznega cikla do dokončanja vseh nalog na seznamu zahtev izdelka. Na grafu je izrisana vsota vseh točk uporabniških zgodb, ki še niso bile zaključene. Scrum tim je predvideval, da bodo vse naloge dokončane v 7 ciklih. Zaradi dodajanja novih nalog, ki niso bile previdene na začetku in niso bile del začetnega seznama zahtev izdelka, se število točk po prvem ciklu poveča. Graf prikazuje korelacijo med preostalim delom in napredkom razvojnega tima, ki zmanjšuje to delo (Mahnič & Žabkar, 2012).

Slika 11: Primer grafa preostalega časa cikla



Vir: V. Mahnič & N. Žabkar, Measuring progress of scrum-based software projects , 2012, str. 75

Graf preostalega časa cikla prikazuje Slika 11. Graf je podoben grafu preostalega časa izdaje, s to razliko, da prikazuje preostanek dela, ki ni bilo dokončano v tekočem ciklu. Horizontalna

os prikazuje število dni cikla, medtem ko je na vertikalni osi izrisano število preostalih ur. Razvojni tim mora graf dnevno posodabljati, zato da lahko spremlja napredek in preverja, ali bo tim uspel do konca cikla dokončati vse naloge (Mahnič & Žabkar, 2012).

2.3 Primerjava tradicionalne in agilne metodologije

V prejšnjih poglavjih je bil predstavljen projektni management in narejen pregled tradicionalnih ter agilnih metodologij za vodenje razvoja programske opreme. Bolj podrobno je bil predstavljen razvoj programske opreme z agilno metodologijo Scrum. V tem poglavju bomo naredili primerjavo tradicionalne in agilne metodologije in izpostavili argumente ZA in PROTI uporabi ene ali druge metodologije.

V literaturi lahko poleg delitve na tradicionalne in agilne metodologije zasledimo tudi delitev na težke (tradicionalne) in lahke (agilne) metodologije. Bistvena razlika med obema metodologijama se kaže v prilagodljivosti oz. sposobnosti odzivanja na spremembe. Ravno ta sposobnost največkrat odloča o uspehu ali neuspehu projekta za razvoj programske opreme. Pri tradicionalni metodologiji po pripravljenem planu za razvoj spremembe niso več dovoljene (Williams & Cockburn, 2003).

Razvojni življenjski cikel tradicionalne metodologije torej predvideva popolno razumevanje zahtev naročnika, pripravo solidnega načrta, brezhiben razvoj in implementacijo programske opreme, ki v celoti zadovolji potrebe naročnika. Zato je velik poudarek na začetnem planiranju, kjer se predpostavlja, da so težave dobro opredeljene in da je možno razviti optimalno rešitev s planiranjem vnaprej. Prav tako se predvideva, da so procesi predvidljivi, ponovljivi in jih je mogoče optimizirati. Temelji na predpostavki, da se lahko proces ustrezno meri in da je možno vire sprememb identificirati in kontrolirati znotraj razvojnega življenjskega cikla. Tak pristop je zelo procesno osredotočen. Stil vodenja temelji na ukazu in kontroli z določeno hierarhijo. Značilni sta visoka formalizacija in standardizacija. Pripravi se velika količina dokumentacije, ki pojasnjuje programsko opremo, njene tehnične in oblikovne specifikacije. Naročnik je najbolj aktiven v fazi pripravljanja specifikacije in implementacije (Devi, 2013).

Agilna metodologija rešuje težave z uporabo empiričnega kontroliranja procesa (angl. *empirical process control*), ki zagotavlja kontrolo procesov z rednim nadzorom in prilagoditvijo tam, kjer zahteve niso popolnoma definirane, in se zato pojavlja negotovost v zvezi z izdelkom. Odločitve so narejene na podlagi opazovanja in eksperimentiranja z optimalno učinkovitostjo delovanja razvojnega tima. Kot že omenjeno na začetku poglavja, empiričen nadzor procesa temelji na transparentnosti, nadzoru in prilagodljivosti. Ker je razvoj programske opreme zelo zapleten in je tendenca spremenljivosti visoka, se zdi empirično kontroliranje procesa najbolj primerno za doseganje rezultatov. Kratki iterativni cikli, periodično razmišljanje, prilagajanje in stalno vključevanje kode v celoten sistem, ki je v razvoju, so najpomembnejše značilnosti agilne metodologije. Kratki ponavljajoči cikli

zagotavljajo prilagodljivost in omogočajo odzive na spremembe. Agilna metodologija Scrum ni postopek ali tehnika za izgradnjo ali razvoj izdelka in tako ne predpisuje posameznih tehničnih rešitev, temveč predstavlja okvir, znotraj katerega poteka razvoj izdelka oz. programske opreme. Komunikacija z naročnikom je stalna, kar daje razvojnemu timu pogoste povratne informacije glede razvite in predane funkcionalnosti (Devi, 2013; Sutherland & Schwaber, 2011).

V Tabeli 1 in 2 so prikazani argumenti ZA in PROTI uporabi tradicionalne ter agilne metodologije.

Tabela 1: Argumenti ZA uporabo in PROTI uporabi tradicionalne metodologije

ZA	PROTI
Enostavna za razumevanje, ker je postopek razvoja linearen.	Vse zahteve morajo biti znane na začetku, kar je največkrat težko dosegljivo.
Mejniki so jasni.	Arhitekti programske rešitve največkrat težko predvidijo vse težave, ki se lahko pokažejo šele tekom razvoja.
Zahteva stabilnost.	Nizka stopnja prilagodljivosti, ki otežuje vključitev sprememb v načrt, ko se je razvoj že začel.
Enostavno upravljanje s časom.	Zagotavlja malo preglednosti za končne uporabnike.
Napredek je lažje merljiv, saj je celoten obseg dela znan že na začetku.	Naročnik dobi celoten izdelek, ko se razvoj vseh dogovorjenih zahtev konča.
Obsežna dokumentacija daje občutek gotovosti.	
Naročnikova prisotnost ni nujno potrebna, ko se faza priprave specifikacije zahtev konča.	

Vir: Povzeto in prirejeno po Bowes, Agile vs. Waterfall: Comparing project management methods, 2014; Matt, Software Developmet Lifecycle: Waterfall vs. Agile, 2015; Lotz, Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?, 2013

Tabela 2: Argumenti ZA uporabo in PROTI uporabi agilne metodologije

ZA	PROTI
Zaradi zaporednih ponavljajočih se ciklov so deli programske opreme hitreje in redno dostavljeni naročniku.	Zaradi poudarka na razvoju programske opreme se lahko zgodi, da se dokumentacija zanemari.
Ob koncu vsakega cikla razvojni tim zagotavlja delujočo verzijo programske opreme.	Projekt se lahko izvaja dlje, kot je bilo pričakovano.
Naročnik pridobi močan občutek odgovornosti, ker je v stalnem stiku z razvojnim timom.	Zahteva 100 % zavezanost razvijalcev.
Redno pridobivanje povratnih informacij s strani naročnika.	Če se agilna metodologija izvaja slabo, lahko to povzroči dodatno neučinkovitost v velikih podjetjih in na koncu je lahko učinek ravno nasproten.

se nadaljuje

Tabela 2: Argumenti ZA uporabo in PROTI uporabi agilne metodologije (nad.)

ZA	PROTI
Tesnejše sodelovanje med razvijalci in podjetjem.	Pogosta vključenost stranke v razvoj je pozitivna, dokler ima naročnik čas in interes stalno sodelovati z razvojnim timom.
Zaradi stalne interakcije med člani tima je prenos znanja večji.	
Spremembe zahtev je mogoče vključiti na katerikoli točki procesa, tudi zelo pozno v razvoju.	
Omogoča oz. daje priložnost za nenehne izboljšave.	
Velika preglednost.	
Testiranje je vključeno od začetka do konca.	

Vir: Povzeto in prirejeno po Bowes, Agile vs. Waterfall: Comparing project management methods, 2014; Matt, Software Developmet Lifecycle: Waterfall vs. Agile, 2015; Lotz, Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?, 2013

3 ANALIZA UVEDBE METODOLOGIJE V PRAKSI

3.1 Priprava vprašalnika

Za iskanje odgovorov na zastavljena empirična vprašanja v uvodu sem uporabila spletni vprašalnik, ki sem ga posredovala timom v proučevanem podjetju, ki programsko opremo razvijajo skladno z agilno metodologijo Scrum. Za spletni vprašalnik sem se odločila, saj je struktura anketirancev računalniško pismena, poleg tega pa je to hitra metoda, ki omogoča avtomatsko shranjevanje odgovorov anketirancev na strežnik in skrajša čas med začetkom in koncem anketiranja (Bregar, Ograjenšek, & Bavdaž, 2005).

Pri sestavi spletnega vprašalnika sem si pomagala s teorijo, ki je predstavljena v teoretičnem delu magistrske naloge. Vprašalnik je sestavljen iz 4 večjih sklopov. Prvi vsebinski sklop vprašanj je bil splošen, glede same uporabe Scruma (vloga anketiranca v timu, število projektov, število članov v timu, dolžina cikla, sestanki, ocenjevanje uporabniških zgodb itd.). Namen drugega sklopa vprašanj je bil ugotoviti, kako strogo se v timih držijo pravil in praks, ki jih predpisuje metodologija Scrum.

V tretjem sklopu sem želela preveriti strinjanje uporabnikov s prednostmi in slabostmi metodologije Scrum, ki jih navaja teorija. Poleg tega me je zanimalo, če bi s Scrumom nadaljevali in ga priporočali drugim sodelavcem. V okviru tega sklopa sem želela tudi ugotoviti, s katerimi težavami se anketiranci pri svojem vsakodnevnem delu še vedno srečujejo. V tretjem sklopu me je prav tako zanimalo, kateri dejavniki so po mnenju anketirancev ključni za uspeh projekta. Pri sestavi vprašanj v tretjem sklopu sem si pomagala tudi s študijo Ocena prednosti metode Scrum in njenih tipičnih praks, ki sta jo opravila Mahnič & Urevc (2012). Četrti sklop je služil za segmentacijo anketirancev (spol in starost) Vključeval je tudi nekaj splošnih vprašanj o izkušnjah (koliko let se že ukvarjajo z razvojem

programske opreme, koliko let izkušenj že imajo z uporabo metodologije Scrum in na kakšnih projektih IT so že sodelovali).

Spletni anketni vprašalnik sem sestavila s pomočjo slovenskega brezplačnega orodja 1KA (EnKlikAnketa), ki se nahaja na domeni www.1ka.si. Povezavo do spletnega vprašalnika sem poslala po elektronski pošti osebam v podjetju, za katere vem, da pri vodenju projektov uporabljajo agilno metodologijo Scrum, saj metodologija ni uvedena v celotnem podjetju. Anketni vprašalnik se nahaja v Prilogi 1.

3.2 Rezultati raziskave

Tekom raziskave sem pridobila 56 veljavnih izpolnjenih vprašalnikov. 41 anketirancev je anketo izpolnilo v celoti. Tako število je bilo pričakovano, saj so anketiranci dobili anketo na služben elektronski naslov in jo izpolnjevali v službenem času, ko so zaposleni in nimajo veliko časa za reševanje ankete. Za potrebe analize rezultatov sem upoštevala vseh 56 izpolnjenih, saj sem si s podatki še vseeno lahko pomagala pri analizi rezultatov.

Ker so bila demografska vprašanja na koncu vprašalnika, so ta vprašanja izpolnili le tisti, ki so v celoti rešili anketo, zato bom tukaj prikazala strukturo anketirancev le za vseh 41 v celoti izpolnjenih vprašalnikov. Od 41 vprašanih, ki so v celoti rešili anketo, je bilo 29 % žensk in 71 % moških, kar je razumljivo, saj gre za IT podjetje, kjer prevladuje moška populacija. V podjetju prevladuje mlad kader, saj je bila večina vprašanih mlajših od 40 let. Največ vprašanih je bilo starih med 36 in 40 let. To so bili večinoma moški (glej Tabelo 3).

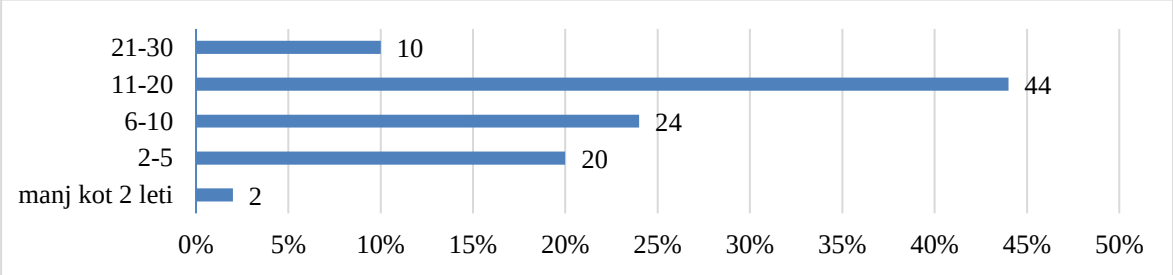
Na vprašanje »Koliko let se že ukvarjate z razvojem programske opreme?« je največ vprašanih odgovorilo, da se z razvojem programske opreme ukvarja med 11 in 21 let. Le 4 oz. 10 % vprašanih je odgovorilo, da imajo med 21 do 30 let izkušenj, kar je pričakovano, saj gre, kot sem že omenila, za mlad kader (glej Sliko 12).

Največ vprašanih ima med 2 in 2,5 leti izkušenj z uporabo metodologije Scrum. Takih je bilo 46 %, 27 % pa je takih, ki imajo več kot 2,5 leti izkušenj (glej Sliko 13).

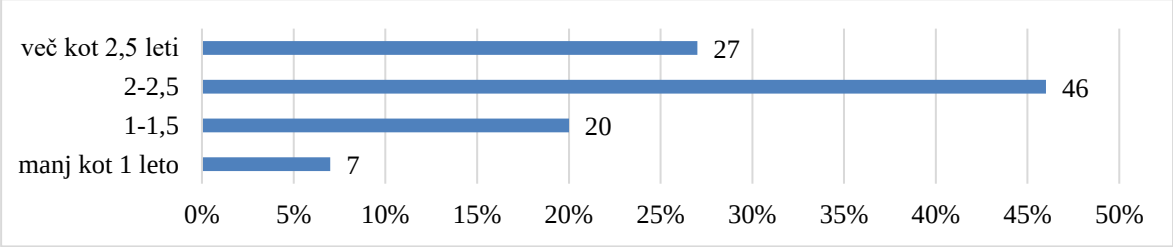
Tabela 3: Struktura anketirancev po spolu in starosti v %

		Spol		
		MOŠKI	ŽENSKI	SKUPAJ
Starost	Do 25 let	1	0	1
		3,0	0,0	3,0
	25-30	5	4	9
		17,0	33,0	22,0
	31-35	8	2	10
		28,0	17,0	24,0
	36-40	13	1	14
		45,0	8,0	34,0
	41-45	2	3	5
		7,0	25,0	12,0
	51-55	0	2	2
		0,0	17,0	5,0
	SKUPAJ	29	12	N = 41
		71,0 %	29,0 %	100,0 %

Slika 12: Število let ukvarjanja z razvojem programske opreme v %



Slika 13: Število let izkušenj z uporabo agilne metodologije Scrum v %



Pri vprašanju »Na katerih IT projektih ste v preteklosti že sodelovali?« je bilo možnih več odgovorov. Večina vprašanih je sodelovala na projektih razvoja informacijske podpore poslovnim procesom in projektih prilagoditve produkta poslovnim procesom naročnika, kar sta tudi glavni dejavnosti podjetja. Sledijo projekti implementacije tujega produkta, kot je npr. Microsoft, Qlik, Mecoms ipd., kar je prav tako ena od dejavnosti, ki jih proučevano podjetje opravlja. Nekaj izkušenj imajo anketiranci tudi z razvojem spletnih trgovin,

razvojem programa za izdelke in razvojem spletnih iger. V Tabeli 4 so IT projekti padajoče razvrščeni glede na pogostost sodelovanja na projektu.

Tabela 4: IT projekti, na katerih so anketiranci v preteklosti že sodelovali

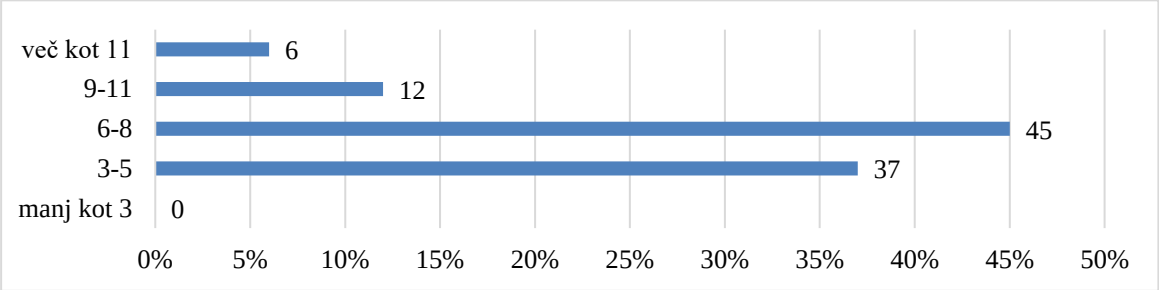
ODSTOTEK	VRSTE IT PROJEKTOV
90 %	Projekt razvoja informacijske podpore poslovnim procesom
83 %	Projekt prilagoditev produkta poslovnim procesom naročnika
59 %	Projekt implementacije produkta (Microsoft, Qlik, Mecoms...)
56 %	Projekt razvoja spletnih strani
41 %	Projekt razvoja informacijske podpore delu in zabavi (programi in aplikacije za računalnik/telefon)
34 %	Projekt lokalizacije produkta
12 %	Projekt razvoja spletnih trgovin
12 %	Projekt avtomatizacije proizvodnje (robotika, kontrola...)
12 %	Projekt razvoja programa za izdelke (programiranje mikroprocesorjev...)
5 %	Projekt razvoja spletnih iger

Pri vprašanju »Katera je njihova trenutna vloga v Scrum timu?« je bilo prav tako možnih več odgovorov, saj lahko ena oseba nastopa v več različnih vlogah. Na splošno je v podjetju največ razvijalcev programske opreme in svetovalcev, kar je bilo pričakovano. Tema vlogama sledijo arhitekt poslovnih rešitev, testni inženir, skrbnik metodologije in lastnik izdelka. Prav tako se pojavlja vloga projektnega managerja, ki ga metodologija Scrum ne predvideva. Pri pregledu individualnih anket je bilo ugotovljeno, da ponekod skrbnik metodologije in lastnik izdelka nastopata tudi v vlogi razvijalca programske opreme, torej hkrati opravljata dve vlogi.

V 70 % Scrum timov sodelujejo na 1 ali 2 projektih. Nekaj je tudi takih (20 %), ki delajo istočasno na 3 projektih. Ostali (10 %) so označili, da sodelujejo na več kot 3 projektih. V splošnem je povprečna ocena števila projektov 3,3. Večina vprašanih (39 %) dnevno nameni 5 do 6 ur za naloge, ki so definirane v posameznem ciklu. Nekaj manj (24 %) jih nameni 3 do 4 ure. Nekaj pa jih je tudi odgovorilo, da njihove ure niso vštete v kapaciteto. Večinoma so to projektni managerji, ki ne nastopajo v drugih vlogah, ki jih predvideva Scrum . Le 8 % je takih, ki svoj polni delovni čas, torej 7,5 ur, namenijo reševanju nalog v Scrumu.

Večina Scrum timov ima od 6 do 8 članov. Tako je odgovorilo 45 % vprašanih. 37 % vprašanih je odgovorilo, da imajo v timu 3 do 5 članov. Nekaj vprašanih (12 %) je označilo, da njihov Scrum tim šteje od 9 do 11 članov. 6 % jih je odgovorilo, da imajo več kot 11 članov (Slika 14). Vprašane sem prosila, da med člane tima ne štejejo lastnika izdelka, skrbnika metodologije in projektnega managerja, če le ti niso všteti tudi v kapaciteto za razvoj in testiranje nalog skozi Scrum.

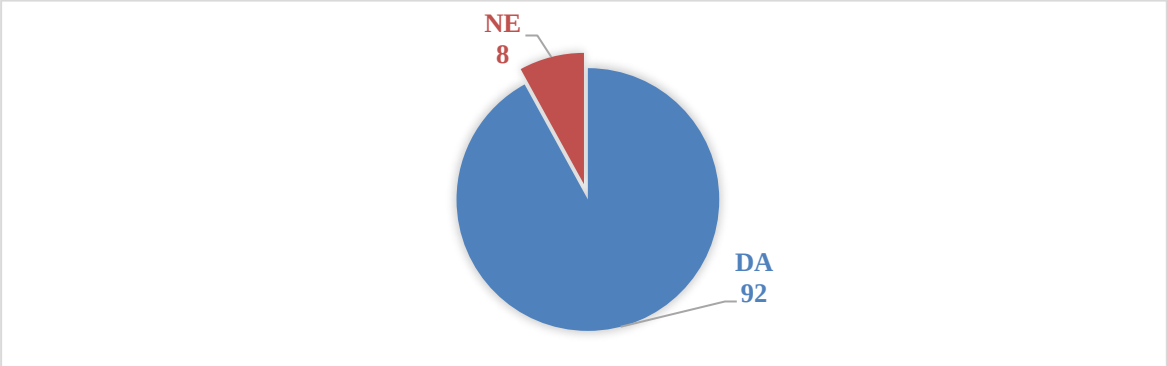
Slika 14: Število članov v Scrum timu v %



Kar 86 % vprašanih izvaja razvoj skozi 2-tedenske cikle. To je razumljivo, saj je najbolj smiselno, da imajo vsi sodelujoči timi enako dolge cikle. Nekateri imajo tudi 3- ali 4-tedenske cikle. Večina Scrum timov ima del funkcionalnosti tudi že v produkciji, kar pomeni, da naročnik programsko opremo že uporablja. Tako je odgovorilo 76 % vprašanih.

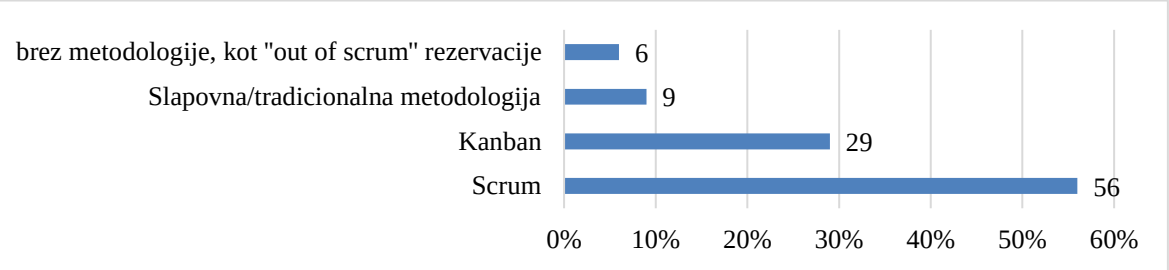
Vsi, ki so odgovorili z DA, so morali odgovoriti na dodatni vprašanji: »Ali vaš Scrum tim opravlja tudi vzdrževanje (*Support*) za funkcionalnost, ki je že v produkciji?« in »Kako izvajate/vodite vzdrževanje (*Support*)? Katero metodologijo uporabljate?«. Na prvo vprašanje je 92 % vprašanih odgovorilo z DA (Slika 15).

Slika 15: Odgovora na vprašanje: »Ali vaš Scrum tim opravlja tudi vzdrževanje za funkcionalnost, ki je že v produkciji?« v %



Najbolj pogosta metodologija, ki jo uporabljajo za vzdrževanje, je prav tako Scrum. Tako je odgovorilo 56 % vprašanih. Druga najbolj pogosta metodologija je Kanban. Le 9 % jih za izvedbo nalog, ki so del vzdrževanja, uporablja tradicionalen pristop. Vprašani so imeli tudi opcijo Drugo, kjer je vprašani zapisal »brez metodologije, kot *out of scrum* rezervacije«. To pomeni, da ima lahko en član ali več članov tima manjšo kapaciteto za izvedbo nalog skozi Scrum, ker si naredijo časovno rezervacijo za vzdrževanje (Slika 16).

Slika 16:Vrsta metodologije, ki jo Scrum tim uporablja za naloge, ki so del vzdrževanja v %



Naslednje vprašanje se je nanašalo na izvajanje metodologije Scrum. Zanimalo me je, kako strogo se držijo pravil, ki jih predvideva Scrum, zato sem postavila serijo trditev. Vprašani so morali na lestvici od 1 do 5 označiti svoje strinjanje oz. nestrinjanje s trditvijo. 1 pomeni »Sploh se ne strinjam« in 5 »Popolnoma se strinjam«. V Tabeli 5 je prikazana povprečna ocena, kjer razvidno, kako se v proučevanem podjetju Scrum izvaja in katera pravila se upoštevajo.

Tabela 5: Prikaz ravni upoštevanja pravil, ki jih priporoča metodologija Scrum

TRDITVE	POVPREČNA OCENA
Dnevni sestanek (*Daily Meeting*) se redno izvaja.	**4,4**
Uporabljene so vse vloge (lastnik izdelka, skrbnik metodologije in razvojni tim), ki jih Scrum predpisuje.	**4,2**
Sestanek za planiranje cikla (*Planning*) se dosledno izvaja pred pričetkom naslednjega cikla.	**4,0**
Za vsako uporabniško zgodbo se oceni tudi predvideno število dni/ur za dokončanje naloge.	**4,0**
Najprej se vedno oceni napor za posamezno nalogo.	**3,9**
Naš Scrum tim je samoorganiziran tim.	**3,8**
Znotraj tekočega cikla se prioritete zahtev (nalog) ne spreminjajo.	**3,0**
Prioritete nalog določa in spreminja le lastnik izdelka.	**2,9**
Pri planiranju naslednje iteracije cikla se dosledno upošteva hitrost (*Velocity)* razvoja tima.	**2,8**
Za vsako uporabniško zgodbo se dosledno upošteva definicija dokončano (*Definition of Done).*	**2,7**
Retrospektiva se izvaja redno ob koncu vsakega cikla.	**2,6**
Pred ocenjevanjem nalog so uporabniške zgodbe jasno definirane.	**2,6**
Sestanek za predstavitev iteracije (*how to demo= Sprint revision*) se izvaja redno ob koncu vsakega cikla.	**2,4**

Najbolj se vprašani držijo izvajanja dnevnega sestanka in imajo znotraj tima določene vse predvidene vloge, ki jih Scrum predpisuje. Glede spreminjanja zahtev znotraj tekočega cikla so bili vprašani neodločeni in so trditev ocenili s povprečno oceno 3. Najmanj se vprašani držijo pravila, da morajo biti pred ocenjevanjem nalog uporabniške zgodbe dobro definirane,

ter priporočila za izvajanje sestanka za predstavitev razvitega oz. dokončanega ob koncu vsakega cikla.

Na vprašanje »Ali ste pred uvedbo Scrum metodologije imeli tudi izobraževanje na temo Scruma?« jih je 53 % odgovorilo z DA in 27 % z NE. 20 % vprašanih je odgovorilo, da takrat še niso bili član tima. Vsem, ki so odgovorili z DA, sem zastavila dodatno vprašanje v zvezi z izobraževanjem, ki so ga bili deležni. V Tabeli 6 je prikazano strinjanje oz. nestrinjanje s trditvami.

Tabela 6: Prikaz povprečnih ocen trditev o izobraževanju na področju metodologije Scrum, ki so ga bili anketiranci deležni

TRDITVE	POVPREČNA OCENA
Izobraževanje, ki sem ga bil deležen je bilo ustrezno in kakovostno, da lahko agilno metodologijo Scrum izvajam pravilno.	**3,7**
Imel sem premalo znanja, ki bi mi pomagalo pri izvajanju Scrum metodologije.	**2,2**
Ves čas mi je bila na voljo oseba, ki ima izkušnje in znanje o Scrumu in mi je tako lahko pomagala pri prehodu na agilno metodologijo Scrum.	**2,7**

Vprašani so se v splošnem strinjali, da je bilo izobraževanje ustrezno in kakovostno in da so pridobili dovolj znanja za pravilno izvajanje metodologije Scrum. Pri samem uvajanju Scruma oz. pri prehodu na agilno metodologijo pa niso bili deležni podpore in pomoči izkušene osebe.

Pri naslednjih dveh vprašanjih sem želela, da vprašani ocenijo trditve o prednostih in slabostih metodologije Scrum. Pri ocenjevanju prednosti se anketiranci z nobeno trditvijo niso popolnoma strinjali, saj je bila najvišja povprečna ocena 4,0. Vprašani so se najbolj strinjali s trditvijo, »da je projekt mogoče razdeliti na zaporedje obvladljivih delov«. Vse ostale trditve so glede na povprečno oceno ocenili nižje od 4. 80 % vprašanih, ki ima s Scrumom več kot 2,5 leti izkušenj, se s to trditvijo popolnoma strinja. Najbolj so bili vprašani neodločeni pri trditvi, da »naročnik dobi novo funkcionalnost v dogovorjenem času« in da »uporaba Scruma prispeva k boljšim odnosom med naročnikom (uporabniki) in člani razvojnega tima (medsebojno zaupanje in presek znanja je večji)«. S trditvijo, da so zaradi Scruma naloge končane hitreje, pa se v večji meri niso strinjali. To trditev so ocenili s povprečno oceno 2,7 (glej Tabelo 7).

Tabela 7: Prikaz povprečnih ocen trditev, ki se nanašajo na prednosti metodologije Scrum

TRDITVE	POVPREČNA OCENA
Projekt je mogoče razdeliti na zaporedje obvladljivih delov.	**4,0**
Zaradi Scruma tim opravi planiranje in analizo nalog bolj temeljito.	**3,9**

se nadaljuje

Tabela 7: Prikaz povprečnih ocen trditev, ki se nanašajo na prednosti metodologije Scrum (nad.)

TRDITVE	POVPREČNA OCENA
Scrum izboljšuje kakovost razvoja in testiranja.	**3,7**
Scrum omogoča boljšo oceno časa, ki je potreben za dokončanje naloge.	**3,6**
Vsi člani tima imajo popoln vpogled v potek dela na projektu.	**3,5**
Zaradi Scruma je prenos znanja znotraj tima večji.	**3,3**
Zaradi Scruma so prioritete dela vedno jasne.	**3,2**
Zaradi Scruma je odzivnost na spremembe večja.	**3,1**
Scrum zmanjšuje število napak, ki so narejene v razvoju.	**3,1**
Naročnik dobi novo funkcionalnost v dogovorjenem času.	**3,0**
Uporaba Scruma je prispevala k boljšim odnosom med naročnikom (uporabniki) in člani razvojnega tima (medsebojno zaupanje in presek znanja je večji).	**3,0**
Zaradi Scruma so naloge končane hitreje.	**2,7**

Pri ocenjevanju slabosti je bilo največ strinjanja s trditvijo, da je »dokumentacija prevečkrat zanemarjena«, vendar je bila tudi ta povprečna ocena nižja od 4. Vse ostale ocene so bile blizu 3, kar pomeni, da se anketiranci v splošnem niti strinjajo niti ne strinjajo s to trditvijo. Najbolj so bili neodločeni pri trditvah »vodenje projekta z agilno metodologijo Scrum je zahtevnejše« in »sestanki pogosto trajajo predolgo in so izguba časa« (glej Tabelo 8).

Tabela 8: Prikaz povprečnih ocen trditev, ki se nanašajo na slabosti metodologije Scrum

TRDITVE	POVPREČNA OCENA
Prevečkrat je dokumentacija zanemarjena.	**3,8**
Prevelika preobremenjenost s sestanki, ki jih Scrum metodologija predvideva za vse člane Scrum tima.	**3,2**
Naročnik nima dovolj časa za stalno interakcijo s Scrum timom.	**3,3**
Vodenje projekta z agilno metodologijo Scrum je zahtevnejše.	**3,0**
Sestanki pogosto trajajo predolgo in so izguba časa.	**3,0**
Scrum določa preveč pravil.	**2,9**
Scrum je premalo agilen glede na zahteve naročnika.	**2,8**
Daljše trajanje projekta kot predvideno.	**2,8**

Na vprašanje »Ali menite, da je Scrum primeren za razvoj vseh vrst programske opreme?« je bilo možno odgovoriti z Zagotovo da, Da, Mogoče, Ne in Zagotovo ne. Glede na Tabelo 9 je razvidno, da so se vprašani v večji meri strinjali in odgovorili z DA, saj so odstotki v tej koloni najvišji. Tudi izračun povprečne ocene kaže na to, da menijo, da je Scrum primeren za vse vrste razvoja. Najnižjo oceno so dodelili avtomatizaciji proizvodnje, najvišjo pa razvoju informacijske podpore delu, zabavi in poslovnim procesom. To je tudi razumljivo, saj imajo vprašani veliko več izkušenj z razvojem podpore delu, zabavi in poslovnim procesom kot z razvojem avtomatizacije proizvodnje.

Tabela 9: Odgovori na vprašanje: »Ali menite, da je Scrum primeren za razvoj vseh vrst programske opreme?«

VRSTE RAZVOJA PROGRAMSKE OPREME	Zagotovo ne	Ne	Mogoče	Da	Zagotovo da	POVPREČNA OCENA
Razvoj informacijske podpore delu in zabavi (programi in aplikacije za računalnik/telefon)	0%	7%	18%	49%	27%	4,0
Razvoj informacijske podpore poslovnim procesom	0%	2%	18%	53%	27%	4,0
Razvoj programa za izdelke (programiranje mikroprocesorjev)	4%	4%	29%	44%	18%	3,7
Implementacija produkta (Microsoft, Qlik, Mecoms...)	0%	4%	18%	53%	24%	4,0
Prilagajanje produkta poslovnim procesom naročnika	2%	7%	20%	53%	18%	3,8
Razvoj spletnih strani	0%	4%	31%	38%	27%	3,9
Razvoj spletnih trgovin	0%	4%	24%	44%	27%	3,9
Razvoj spletnih iger	0%	4%	29%	44%	22%	3,8
Avtomatizacija proizvodnje (robotika, kontrola...)	5%	11%	31%	36%	18%	3,5

Na vprašanje »Ali bi Scrum priporočili tudi drugim sodelavcem?« je 91 % vprašanih odgovorilo z DA. Tiste, ki so odgovorili z NE, sem prosila, da svojo oceno pojasnijo. Ker je bil odgovor neobvezen, sem dobila le 3 odgovore, ki so prikazani v Tabeli 10. Dva vprašana sta pojasnila, da Scrum ni primeren za vse faze projekta oz. da je primeren takrat, ko se produkt razvija na novo in ga naročnik še ne uporablja, torej še ni v produkciji.

Tabela 10: Odgovori na vprašanje: »Zakaj Scrum metodologije ne bi priporočali tudi drugim sodelavcem?«

ODGOVORI
• Ker lahko enake rezultate dosežemo s pomočjo drugih metodologij in ker Scrum ni primeren za vse faze projekta. • Zaradi zahtevnosti organizacije procesa s Scrum metodologijo. • Primerna za specifične vrste dela, predvsem za nove izdelke, ki še niso v produkciji.

Naslednje vprašanje »Da bi lahko svoje delo kar najbolje opravljali, kaj bi si želeli spremeniti v zvezi s Scrumom in svojim delom na splošno?« ni bilo obvezno, zato je nanj odgovorilo le nekaj anketirancev (glej Tabelo 11). Le 2, ki sta na to vprašanje odgovorila, sta zapisala, da nič ne bi spremenila. Veliko odgovorov je bilo na temo uporabniških zgodb, predvsem v smislu definiranja, analiziranja in specifikacije zahtev, preden le te pridejo v cikel. Vprašani bi želeli prejeti od lastnika izdelka in naročnika bolj izčrpne specifikacije in definicije nalog. Tako bi lahko bolj natančno ocenili in planirali ure, ki so potrebne za dokončanje naloge.

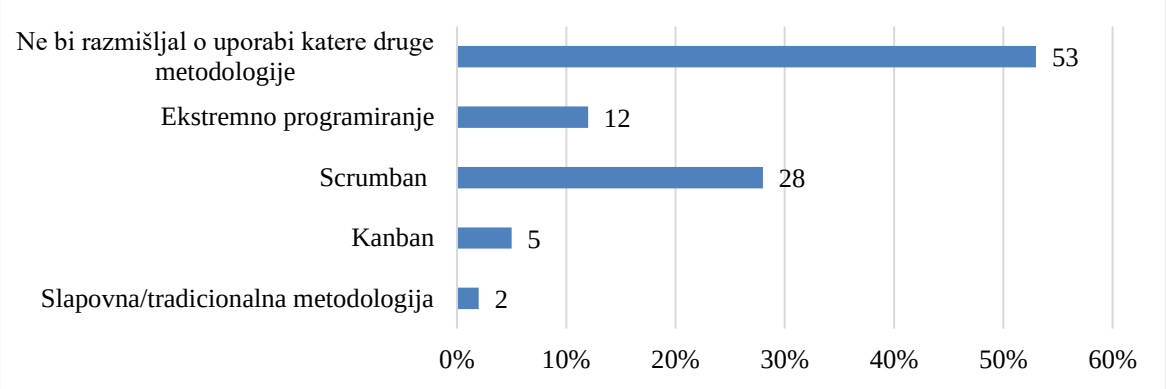
Posledično bi lahko z večjo gotovostjo napovedali, kdaj bo naročnik dobil funkcionalnost. Izpostavili so tudi željo po intenzivnejši komunikaciji z naročnikom in skupnem določanju prioritet, ki bi omogočalo jasno in fiksno določitev prednostnih nalog. Med pomembne dejavnike so uvrstili tudi razumevanje in upoštevanje pravil uporabe metodologije Scrum s strani naročnika. Izpostavili so tudi potrato časa s sestanki. Menijo, da je potrebno število sestankov zmanjšati oz. optimizirati vsebino sestankov.

Tabela 11: Kaj bi vprašani želeli spremeniti v zvezi z metodologijo Scrum in s svojim delom na splošno?

SPLOŠNO GLEDE ORGANIZACIJE PROJEKTA
• Predvsem mora biti vzdrževanje ločeno od razvoja. • Dosledna implementacija dogovorov. • Postavitev realističnih rokov s strani vodstva/organizacije podjetja. Boljše spremljanje spreminjanja zahtev stranke in postavljanje novih rokov.
SPLOŠNO GLEDE SCRUMA
• Redefiniranje Scruma v podjetju, sledenje vsem pravilom ali nobenemu. • Pomembno je, da se Scruma ne držiš kot pribito, temveč ga prilagodiš svojim razvojnim specifikam. • Scrum deluje, če sodelujejo tudi člani, ki ga uporabljajo. Treba je motivirati člane, da sledijo metodologiji in rezultati bodo prišli sami. • Dosledno upoštevanje pravil Scruma. • Zmanjšati število sestankov.
SODELOVANJE Z NAROČNIKOM
• Bolj jasne in fiksne prioritete s strani stranke. • Večja vpletenost naročnika. • Konsenz vseh deležnikov (predvsem strank) o pravilih uporabe metode Scrum. Določitev fiksnih prioritet in obsega (vsebine) kot predpogoj za uspešno izvedbo cikla.
UPORABNIŠKE ZGODBE
• Več časa za analizo. • Boljša specifikacija nalog in bolj natančna ocena planiranih ur za opravilo nalog. • Bolj definirane zahteve stranke in tako lažje ocenjevanje posamezne zgodbe → s tem se lažje napove oceno in napor za nalogo. Tako lahko z večjo gotovostjo trdimo, kdaj bo stranka prejela funkcionalnost. • Več manjših nalog, ocena posamezne naloge bi morala biti največ en dan. • Bolj definirane naloge s strani lastnika izdelka. • Uporabniške zgodbe bi morale biti natančno definirane pred planiranjem cikla. • Postopek predpriprave oz. specificiranja uporabniških zgodb po Scrumu je pri manjših nalogah preveč časovno potraten, saj zahteva prisotnost vseh članov tima. Za manjše naloge bi lahko ocenjevanje prevzel en sam usposobljen razvijalec. • Prenos znanja bi lahko bil večji. Morali bi si vzeti več časa za predstavitev cikla (*how to demo*). • Uporabniške zgodbe bi morale biti bolj natančno določene, tako da bi načrt dela vzel manj časa med ciklom.

Na vprašanje »Ali bi razmišljali o uporabi katere druge metodologije?« jih je največ (53 %) odgovorilo z NE. Če bi, bila to Scrumban, ki je kombinacija Scruma in Kanbana. Tako je odgovorilo 28 % vprašanih. 12 % vprašanih bi se odločilo za Ekstremno programiranje in 5 % za Kanban. Za tradicionalen pristop bi se odločilo le 2 % vprašanih (glej Sliko 17).

Slika 17: Odgovori na vprašanje: »Ali bi razmišljali o uporabi katere druge metodologije?« v %



V nadaljevanju je sledilo vprašanje »Ali menite, da je možna kombinirana uporaba tradicionalnega in agilnega pristopa na enem projektu? Vpišite vašo oceno od 1 do 5, kjer 1 predstavlja povsem tradicionalen pristop, 5 pa popolnoma agilen pristop«. Nihče od vprašanih ni odgovoril z 1. Skupna povprečna ocena vseh vprašanih je 3,7, kar kaže na to, da se večina vprašanih bolj nagiba k agilnemu pristopu. Tukaj sem postavila še neobvezno vprašanje: »Ali lahko pojasnite svojo oceno?«. V Tabeli 12 so prikazani odgovori. Vprašanim se zdi tradicionalen pristop najbolj smiseln v začetni fazi projekta, torej pri začetnem zajemu zahtev, analiziranju in definiranju uporabniških zgodb, ki so ključnega pomena za dober razvoj in uspešen projekt. 2 sta zapisala, da se že sedaj do neke mere uporablja kombiniran pristop, saj se znotraj cikla izvajajo tradicionalne aktivnosti, kot so analiza, načrtovanje, razvoj, testiranje in implementacija. Nekaj vprašanih je bilo proti uporabi kombiniranega pristopa, ker poveča kompleksnost projekta, podaljša celoten projekt in prinese zmedo glede dela v timu.

Tabela 12: Pojasnilo vprašanih glede možnosti uporabe kombiniranega in tradicionalnega pristopa

ZA
• Tudi trenutno uporabljamo kombinacijo. • Včasih bi morali imeti več informacij o zahtevah/potrebah, ki jih ima stranka, da bi lahko podali točno oceno in tudi lažje definirali zahteve, ki jih ima stranka. • Potrebno se je izogibati ekstremom, najboljše je pobrati najboljše prakse iz vsake tehnologije. • Tradicionalen zajem zahtev in nato agilen razvoj. • Popolnoma agilen pristop pri razvoju za znanega naročnika po mojem mnenju ni realen.

se nadaljuje

Tabela 12: Pojasnilo vprašanih glede možnosti uporabe kombiniranega in tradicionalnega pristopa (nad.)

ZA
• Kombinacija obojega lahko izkoristi prednosti enega, ki so slabosti drugega. Na primer, če se ugotovijo večje prioritete tekom cikla, bi se lahko dodajale naloge. Število sestankov bi se zmanjšalo... • Za del projekta, ki ima natančno specifikacijo in vemo, da se med izvedbo projekta ne bo spreminjal, bi bil tradicionalni pristop. Za del projekta, za katerega se bo specifikacija izoblikovala tekom izvedbe, je bolj primeren agilen pristop. • Agilni pristop za ekipo, ki se ukvarja neposredno s stranko (vzdrževanje, popravljanje napak,..), tradicionalni pristop za razvoj novih funkcionalnosti. • V začetni fazi (plasiranje projekta) je vseeno treba na projekt pogledati kot celoto s tradicionalnim pristopom, nadaljevanje pa je lahko agilno. • Za večje kose novih funkcionalnosti bi se lahko uporabljal tradicionalen pristop. • Kljub agilnemu pristopu, se znotraj vsakega Scrum cikla izvajajo analiza, načrtovanje, implementacija, testiranje → torej že izvajamo tradicionalen proces. • Tradicionalen v smislu, da se za pripravo uporabniške zgodbe pridobi veliko podrobnosti, ki jih stranka zahteva.
PROTI
• Kombinirana raba je nesmiselna. Tradicionalni pristop temelji na daljšem obdobju analiziranja in planiranja, medtem ko agilne metode težijo k čim hitrejšemu izdelku in sprotnemu prilagajanju le tega. Sočasna uporaba dveh tako različnih metod bi bila škodljiva. • Kombinacija pristopov lahko poveča kompleksnost projekta. Uporaba enega pristopa tako zmanjšuje kompleksnost. • Menim, da je najbolje uporabljati prilagojeno verzijo Scruma, brez mešanja različnih pristopov. • Po mojem mnenju se vsak projekt lahko vodi z uporabo agilnega pristopa, kar pripomore k hitrejšem razvoju in tudi hitrejši dostavi manjših in zaključenih celot produkta.
DRUGO
• Agilen pristop je pogosto neučinkovit med svetovanjem v fazi implementacije. • Metodologijo dela na projektu je potrebno prilagoditi fazi, v kateri se projekt nahaja. • Lahko kombiniraš pristope, ni pa to več Scrum.

Vprašani so morali še pri dveh vprašanjih ocenjevati trenuten način izvajanja projektov. Postavila sem serijo trditev, kjer so morali vprašani izraziti svoje strinjanje oz. nestrinjanje z oceno od 1 do 5, kjer je 1 pomenilo »Sploh se ne strinjam« in 5 »Popolnoma se strinjam«. Prvo vprašanje se je nanašalo na težave, s katerimi se pri svojem delu še vedno srečujejo. Vprašani so kot največji težavi izpostavili nerealistične roke, ki so postavljeni s strani vodstva, in primanjkovanje časa za pripravo dobre specifikacije razvoja posamezne naloge. Trditvi so ocenili s povprečnima ocenama 3,9 in 3,8. Prav tako opažajo, da so naloge, ki pridejo v cikel, slabo definirane ter da dogovorjeni roki niso uresničeni. Neodločeni so bili le pri trditvi »Ni pregleda nad celotnim projektom«. Nestrinjanje pa so pokazali pri trditvi »S strani vodstva ni dovolj podpore za ustrezno vodenje Scruma«. To trditev so ocenili s povprečno oceno 2,3 (glej Tabelo 13).

Tabela 13: Povprečna ocena trditev, ki se navezujejo na težave, s katerimi se anketiranci srečujejo pri svojem delu

TRDITVE	POVPREČNA OCENA
S strani vodstva so postavljeni nerealistični roki.	**3,9**
Premalo časa za pripravo dobre specifikacije razvoja posamezne naloge.	**3,8**
Dogovorjeni roki niso uresničeni.	**3,5**
Naloge, ki pridejo v cikel, so slabo definirane.	**3,5**
Na splošno se zahteve naročnika spreminjajo hitreje kot je dolžina cikla.	**3,4**
Ker ima naročnik del funkcionalnosti že v produkciji, je usklajevanje dela težje.	**3,4**
Dražji projekt kot predvideno.	**3,3**
Funkcionalnost, ki je predana stranki je pogosto slabo testirana.	**3,3**
Zahteve se spreminjajo znotraj posameznega cikla	**3,2**
Ni pregleda nad celotnim projektom.	**3,0**
Slabo postavljene prioritete in zaradi tega posledično pogosto spreminjanje prioritet.	**2,8**
S strani vodstva ni dovolj podpore za ustrezno vodenje Scruma.	**2,3**

Drugo vprašanje se je nanašalo na uspeh projekta. Zanimalo me je, kateri dejavniki so po mnenju anketirancev nujni za uspeh projekta. Vprašani nobenega dejavnika niso ocenili kot nepomembnega. Popolnoma neodločeni so bili pri trditvi, da je za uspeh nujno »dosledno izvajanje retrospektive«. Najbolj pomembne se jim zdijo »dobro analizirane in definirane uporabniške zgodbe«. Ta dejavnik so ocenili s povprečno oceno 4,7. S povprečno oceno 4,6, 4,5 in 4,4 so sledili »realistični roki za dostavo programske opreme«, »realistične predstave naročnika« in »jasno zastavljen seznam nalog za prihajajoči cikel«, ki so prav tako nujni za uspešnost projekta (glej Tabelo 14).

Tabela 14: Povprečna ocena pomembnosti dejavnikov za uspeh projekta

TRDITVE	POVPREČNA OCENA
Dobro analizirane in definirane uporabniške zgodbe.	**4,7**
Postavljeni realistični roki za dostavo funkcionalnosti programske opreme.	**4,6**
Realistične predstave naročnika glede programske opreme.	**4,5**
Jasno zastavljen seznam nalog za prihajajoči cikel.	**4,4**
Pravilne ocene zahtevnosti posameznih uporabniških zgodb.	**4,0**
Dosledno upoštevanje koncepta dokončanosti (*Definition of done*).	**4,0**
Redno sodelovanje naročnika z razvojnim timom.	**4,0**
Pravilna ocena hitrosti (*Velocity*) razvoja tima na začetku vsake iteracije.	**3,9**
Dosledno izvajanje sestankov za načrtovanje (*Planning*) cikla.	**3,9**
Podpora vodstva je ključni dejavnik za uspeh agilnega razvoja.	**3,9**
Dosledno izvajanje vsakodnevnih sestankov (*Daily Meetings*).	**3,7**
Dosledno izvajanje sestankov za predstavitev rezultatov cikla (*How to Demo*).	**3,3**
Dosledno izvajanje retrospektive.	**3,0**

3.3 Ugotovitve in razprava

Prvo raziskovalno vprašanje, ki sem si ga zastavila v uvodnem delu, je bilo: »**Kaj so po mnenju uporabnikov prednosti in slabosti agilne metodologije Scrum?**«

Za ugotavljanje mnenja uporabnikov sem postavila trditve o prednostih in slabostih metodologije Scrum ter prosila vprašane za oceno na lestvici od 1 do 5, kjer je 1 pomenilo »Sploh se ne strinjam« in 5 »Popolnoma se strinjam«. Trditve so prikazane v Tabelah 7 in 8. Raziskava prednosti je pokazala, da se povprečna ocena za 5 od 12 trditev nahaja na intervalu med 3,5 in 4,0. Za ostalih 7 trditev se ocena giblje okoli povprečne ocene 3, kar pomeni, da se vprašani niti ne strinjajo niti strinjajo z navedenimi trditvami. Ocene vprašanih kažejo na to, da se uporabniki strinjajo z manj kot polovico naštetih prednosti oz. so neodločeni. Najvišjo povprečno oceno 4,0 je dobila trditev »Projekt je mogoče razdeliti na zaporedje obvladljivih delov«, kar sta v svoji raziskavi ugotovila tudi Mahnič in Urevc (2012). Druga trditev, s katero so se vprašani strinjali, je: »Zaradi Scruma tim opravi planiranje in analizo nalog bolj temeljito«. Ta trditev je bila ocenjena s povprečno oceno 3,9. Najmanj so se vprašani strinjali s trditvijo, da »so naloge zaradi Scruma končane hitreje«. Anketa je prinesla nekoliko presenetljive rezultate glede prednosti Scruma, ki se nekoliko razlikujejo od vsesplošno pričakovanih rezultatov, ki naj bi jih zagotavljala agilna metodologija Scrum. Verjeten vzrok za odstopanje je nedosledno upoštevanje priporočil metodologije Scrum.

Kot največjo slabost so anketiranci izpostavili »zanemarjeno dokumentacijo«. To trditev so ocenili s povprečno oceno 3,8. Vse ostale trditve so bile nižje od 3,5, zato bi težko rekli, da se vprašani s trditvami o slabostih strinjajo. Sledita trditvi »naročnik nima dovolj časa za stalno interakcijo« s Scrum timom (3,3) in »prevelika preobremenjenost s sestanki« (3,2). Povprečne ocene trditev o slabostih Scruma kažejo, da se uporabniki z njimi ne strinjajo oz. so prav tako neodločeni.

Na splošno bi večina vprašanih, kar 91 %, metodologijo Scrum priporočalo drugim sodelavcem, kar kaže na to, da je bila uvedba metodologije Scrum prava odločitev. Prav tako jih je večina odgovorila, da ne bi razmišljali o uporabi druge metodologije, in če že, bi se večina odločila za Scrumban, ki je kombinacija Scruma in Kanbana. Zaključimo lahko, da so prednosti, ki jih je metodologija Scrum prinesla, večje od slabosti.

Moje drugo raziskovalno vprašanje je bilo: »**Kje se še vedno pojavljajo težave v delovnem procesu in kako jih je mogoče rešiti ali se jim izogniti?**«

Največji problem, ki so ga vprašani izpostavili, je postavljanje nerealističnih rokov za izvedbo oz. razvoj funkcionalnosti s strani vodstva. Prav tako so izpostavili, da je premalo časa za pripravo specifikacije razvoja posamezne naloge. Trditvi »dogovorjeni roki niso uresničeni« in »naloge, ki pridejo v cikel so slabo definirane« sta prejeli povprečno oceno

3,5. Tudi iz odgovorov na vprašanje »Da bi lahko svoje delo kar najbolje opravljali, kaj bi si želeli spremeniti v zvezi s Scrumom in svojim delom na splošno?« je bilo razbrati, da si vprašani želijo več redne komunikacije z naročnikom, postavljanje realističnih rokov ter bolj natančno analizirane in definirane uporabniške zgodbe. Trditve so prikazane v Tabeli 13.

Moje tretje raziskovalno vprašanje je bilo: »**Ali je v proučevanem podjetju možna kombinirana uporaba tradicionalnega in agilnega pristopa na enem projektu?«**

Glede kombinirane uporabe agilnega in tradicionalnega pristopa so bila mnenja vprašanih deljena. Vprašani so morali določiti oceno od 1 do 5, kjer je 1 predstavljala povsem tradicionalen pristop in 5 popolnoma agilen pristop. Vprašani so se v povprečju bolj nagibali k samostojni rabi agilnega pristopa, saj je skupna povprečna ocena dosegla 3,7. Pri pojasnjevanju svoje ocene so bila mnenja deljena. Tradicionalni pristop se jim je zdel najbolj smiseln v prvi fazi projekta, torej v času spoznavanja naročnika in njegovih zahtev ter med pripravo specifikacije in predloga rešitve. Večina vprašanih namreč ocenjuje, da sta zajem in specifikacija zahtev najbolj pomemben in ključni del projekta. Pojasnila vprašanih so prikazana v Tabeli 12.

Zadnje raziskovalno vprašanje je bilo sledeče: »**Ali je agilna metodologija Scrum res primerna za razvoj vseh vrst programske opreme?«**

Pri odgovoru na vprašanje »Ali menite, da je Scrum primeren za razvoj vseh vrst programske opreme?« so se večinoma vsi strinjali, da je Scrum primerna metodologija, saj so se vse ocene nahajale na intervalu med 3,5 in 4,0. Najbolj neodločeni so bili pri razvoju avtomatizacije proizvodnje (robotika, kontrola...), saj večina vprašanih še ni delala na tovrstnih projektih. Izkušnje na tem področju je imelo le 12 % anketirancev.

3.4 Predlogi za podjetje

Glede na rezultate raziskave so vprašani zadovoljni in vidijo več prednosti kot slabosti metodologije Scrum. Pozitivno je tudi to, da bi večina Scrum priporočala svojim sodelavcem, kot tudi to, da ne razmišljajo o uporabi druge metodologije.

Največ problemov oz. potrebnih izboljšav vprašani vidijo pri zbiranju zahtev naročnika in analiziranju ter definiranju uporabniških zgodb. Vprašani si želijo boljše in bolj pogoste komunikacije z naročnikom, tako zaradi skupnega definiranja uporabniških zahtev kot tudi zaradi čim manjšega spreminjanja zahtev, ko so te že definirane ali celo dodane v trenutni ali naslednji cikel. Po mnenju vprašanih imata največji vpliv na uspeh projekta analiza in definicija uporabniške zgodbe. Uporabniška zgodba mora biti izčrpno analizirana in temeljito definirana pred sprejetjem v cikel in začetkom razvoja. Na drugo mesto so postavili določitev realističnega roka za dostavo funkcionalnosti programske opreme, iz česar lahko sklepamo, da so roki pogosto prekratki, kar povzroča naglico, ki ima negativen vpliv na razvoj in testiranje. Po mnenju anketirancev je od vseh sestankov najpomembnejši sestanek

za načrtovanje cikla. To je v skladu z dejavnikom dobro analizirane in definirane uporabniške zgodbe, ki so ga postavili na prvo mesto.

Določiti bi bilo potrebno bolj jasna pravila, ki bi določala, kdaj je uporabniška zgodba dovolj dobro analizirana in definirana, preden gre lahko v razvoj. Vsaka zahteva, ki ne bi zadoščala pravilom, ne bi smela biti sprejeta v cikel. Podobno kot definicijo statusa dokončano (angl. *definition of done*), bi vsaka uporabniška zgodba imela definicijo statusa pripravljeno (angl. *definition of ready*). Na tem mestu bi bilo potrebno obrazložiti naročniku, zakaj je tako pomembno, da je naloga dobro analizirana in specificirana, in zakaj si je za to vredno vzeti čas, četudi to podaljša rok dostave. Namreč pod časovnim pritiskom in med hitenjem, da bo naloga čim hitreje izvedena, lahko tudi hitro pozabimo, kakšen vpliv ima lahko samo manjši popravek v programski kodi na že delujočo naročnikovo funkcionalnost. Za naročnika, vodstvo podjetja in Scrum tim bi morala imeti kakovost predane funkcionalnosti prednost pred hitrim (slabim) razvojem.

Najbolj smiselno bi bilo uvesti ločene time za razvoj in vzdrževanje, kjer bi bil verjetno najbolj primeren Kanban, saj je tukaj prioriteta na sprotnem reševanju manjših zahtev. To je navadno tudi normalna praksa, vendar je pogosta težava v tem, da so nekateri projekti preveč specifični, pisani na kožo le enemu naročniku. To otežuje predajo projekta timu za vzdrževanje. Nekateri timi že izvajajo vzdrževanje po metodologiji Kanban, a le znotraj istega Scrum tima. Ena od možnosti bi bila, ob predpostavki, da je tim dovolj velik, da se tim razbije in vzpostavi dva ločena tima, enega za razvoj in drugega za vzdrževanje. Tim za razvoj uporablja metodologijo Scrum, tim za vzdrževanje pa Kanban. Ta opcija bi bila najbolj smiselna tam, kjer je število članov v timu od 9 do 11 ali več kot 11, saj je priporočilo Scrum, da je največje število članov v timu med 6 in 8. Druga možnost je rotacija. V tem primeru člani tima rotirajo in so enkrat v razvojnem timu, drugič pa v timu za vzdrževanje. Takšna rešitev odpira nekaj vprašanj. Po kakšnem časovnem ključu bi se člani tima menjali? Ali lahko posameznik vsebinsko preklaplja med vzdrževanjem in razvojem? Če se večje time razbije na manjše skupine, je smiselno uvesti tako imenovani sestanek »Scrum of scrums«. To je dnevni Scrum, ki se izvaja na dveh ravneh, in sicer dnevni sestanek na ravni posameznih razvojnih timov in dnevni sestanek vodij razvojnih timov. Oba sestanka morata biti kratka in konkretna (Cohn, 2007).

Če tim ostaja isti, bi bilo vredno razmisliti o uvedbi metodologije Scrumban, ki jo je omenilo kar nekaj (28 %) vprašanih. Terlecka (2012) v svojem članku, kjer opisuje uporabo Scrumbana, navaja, da je ta metodologija primerna za tiste time, ki imajo veliko vzdrževanja in obenem razvoj po meri (»ad-hoc«). Veliko razvojnih timov skozi uporabo Scruma ugotovi pomanjkljivosti in išče izboljšave. Ker se v podjetju že uporablja Scrum, je lažje spreminjati in ustvarjati hibrid med dvema metodologijama, kot pa uvajati popolnoma novo metodologijo.

Vprašani so sestanek za retrospektivo ocenili kot dejavnik, ki ni pomemben za uspeh projekta (povprečna ocena 2,9), in ga prav tako ne izvajajo pogosto. Povprečna ocena trditve »Retrospektiva se izvaja redno ob koncu vsake iteracije cikla« je bila ocenjena z 2,7. Schwaber (2004) navaja, da retrospektiva med drugim izboljšuje produktivnost, saj se lahko člani tima učijo na napakah drugih. Prav tako je sestanek za retrospektivo čas, ko lahko člani tima izrazijo svoje mnenje o tem, kaj se jim je zdelo v prejšnjem ciklu dobro, s čim bi morali nadaljevati, kaj bi morali izboljšati, kaj se ni izkazalo kot dobra praksa itd. To je tudi primeren čas za pogovor o slabo ali premalo definiranih uporabniških zgodbah, ki so jih vprašani izpostavili kot enega ključnih problemov. Torej ključni dve vprašanji na retrospektivi sta: »Kaj je šlo v tem ciklu dobro?« in »Kaj bi lahko izboljšali v naslednjem ciklu?«

Podobno so vprašani ocenili sestanek za revizijo cikla. Glede na pomembnost za uspeh projekta so ta dejavnik ocenili s povprečno oceno 3,3, obenem pa ga tudi ne izvajajo pogosto, saj je bila trditev »Sestanek za predstavitev iteracije (*how to demo = Sprint revision*) se izvaja redno ob koncu vsake iteracije« ocenjena s povprečno oceno 2,4. Ravno ta sestanek omogoča večji prenos znanja med člani tima, kar je bil je bil tudi eden od odgovorov na vprašanje »Da bi lahko svoje delo kar najbolje opravljali, kaj bi si želeli spremeniti v zvezi s Scrumom in svojim delom na splošno?« (»Prenos znanja bi lahko bil večji → morali bi si vzeti več časa za *how to demo*; uporabniške zgodbe bi morale biti bolj natančno določene, tako da bi načrt dela vzel manj časa med sprintom«.) Menim, da bi morali timi Scrum več pomembnosti in časa nameniti sestanku za retrospektivo in reviziji cikla.

SKLEP

V prvem delu magistrske naloge sem opisala tradicionalni projektni management skozi faze in korake projektnega managementa. Izpostavila sem posebnosti managementa IT projektov in prikazala različne tradicionalne pristope razvoja programske opreme. V nadaljevanju so bili predstavljeni agilni pristopi za vodenje projektov razvoja programske opreme, ki se, sodeč po številnih raziskavah, vse bolj uveljavljajo. Najbolj podrobno je bila opisana agilna metodologija Scrum, ki je v proučevanem podjetju tudi najbolj razširjena in se uporablja že dlje časa.

Kompleksno, turbulentno in spreminjajoče se okolje od managementa zahteva hitro prilagajanje in ravno zato so se pojavili novi agilni pristopi, med katerimi po trenutnih podatkih najbolj izstopa prav agilna metodologija Scrum. Za Scrum je značilen manjši samoorganiziran tim, osredotočenost na ljudi, redno sodelovanje in komunikacija z naročnikom, kratki razvojni cikli in s tem tudi preprostost, jasnost, razumljivost ter preglednost. Definirane so tri vloge: skrbnik metodologije, razvojni tim in lastnik izdelka oz. predstavnik naročnika. Skrbnik metodologije je oseba, ki skrbi, da vsi nemoteno opravljajo svoje delo, tako kot predvideva Scrum. Lastnik izdelka, ki je lahko tudi naročnik, skrbi, da so želje naročnika jasno predstavljene razvojnemu timu, in je tisti, ki naročniku

zagotavlja kakovost izdelka. Scrum predvideva 4 dogodke, ki se izvajajo tekom enega cikla. Ti dogodki so dnevni Scrum, sestanek za na načrtovanje cikla, revizija cikla in retrospektiva cikla. Na dnevnem Scrum dogodku je priporočeno, da vsi člani tima stojijo in na hitro predstavijo dogajanje prejšnjega dne, predstavijo plan današnjega dne in izpostavijo morebiten problem, s katerim se trenutno soočajo. Sestanek za planiranje se izvede pred naslednjim ciklom in je namenjen podrobnejšemu definiranju in ocenjevanju nalog. Na reviziji cikla, ki se izvede na koncu cikla, razvojni tim lastniku izdelka predstavi izvedene naloge. Na retrospektivi cikla se razvojni tim pogovori o poteku prejšnjega cikla in poišče izboljšave za naslednjega. Poleg dogodkov Scrum vpeljuje še naslednje artefakte: seznam zahtev izdelka, seznam zahtev cikla, graf preostalega časa in definicijo statusa »dokončano«.

Kljub trenutno največji priljubljenosti Scruma, pa to še ne pomeni, da je to edina agilna metodologija oz. da so tradicionalne metodologije v celoti izrinjene. Tuje raziskave so pokazale, da je agilna metodologija Kanban v vzponu, saj je enostavnejša, preglednejša in določa manj pravil. Tudi naša raziskava je pokazala, da je Kanban druga najbolj pogosta agilna metodologija za vodenje vzdrževanja, takoj za Scrumom. Kanban predpisuje manj pravil in praks kot Scrum in je zato bolj enostavna za uporabo. Glavna praksa metodologije Kanban je omejitev obsega dela. Kanban zahteva, da pred reševanjem nove naloge zaključimo prejšnjo. Raziskava, ki so jo opravili Lei et.al. (2015), kjer so analizirali vpliv Scruma in Kanbana na projekte IT, je pokazala, da obe metodologiji vodita k uspešno zaključenemu projektu, le da je Kanban bolj uspešen pri doseganju zastavljenih rokov.

Vse bolj priljubljena je tudi agilna metodologija Scrumban (kombinacija Scruma in Kabana). Sodeč po naši raziskavi je Scrumban tista agilna metodologija, ki bi lahko nadomestila Scrum. Mahnič (2014) navaja, da Scrum in Kanban nista izključujoči se metodologiji, temveč lahko druga drugo dopolnjujeta in timu olajšata doseganje zastavljenih ciljev.

Če podamo vsesplošno oceno metodologije Scrum, lahko rečemo, da je bila uvedba Scruma v proučevanem podjetju smotrna, saj bi več kot 90 % vprašanih Scrum priporočilo svojim sodelavcem. Iz tega lahko sklepamo, da ima Scrum več prednost kot slabosti. Mnenje vprašanih je tudi, da lahko metodologijo Scrum uporabimo pri skoraj vseh vrstah razvoja programske opreme. Glede uporabe kombiniranega pristopa pa se vsi bolj nagibajo k agilnemu pristopu. Za uspeh projekta, vsaj glede na podatke raziskave, je najbolj pomembna predvsem prva faza, faza priprave (analiziranje, planiranje in organiziranje), in ravno tukaj se pojavlja največ težav. Ta faza tako predstavlja področje, kjer je največ možnosti za uvedbo nadaljnjih izboljšav. Eden od izzivov je tudi prepričati naročnika o prednostih Scruma in mu razložiti, zakaj se mora držati določenih pravil. Za naročnika je najtežji prehod s starega načina dela na novega. Dejstvo je, da je metodologija le ogrodje, ki pomaga na poti do zastavljenih ciljev, in da metodologija sama po sebi še ne zagotavlja stoodstotnega uspeha projekta. Nenazadnje je pomembno zagotoviti podporo vodstva. Vodstvo mora podpirati nov način dela, spodbujati zaposlene, da vestno sledijo konceptom Scrum, in jim pri tem aktivno pomagati.

Da bi dobili boljše rezultate, bi morali v raziskavo vključiti večje število anketirancev iz sorodnih podjetij z zelo dinamičnim delovnim okoljem in zahtevnim naročnikom, ki ima del funkcionalnosti že v produkciji in celoten razvoj programske opreme poteka več let. V takšnih pogojih obstaja večja verjetnost, da se bodo zahteve naročnika spremenile in da se bo pojavila potreba po rednem vzdrževanju programske opreme. Za podrobnejši vpogled v delo posameznih timov bi bilo potrebno vsak tim opazovati pri vsakdanjem delu, podrobneje spoznati delovanje tima in potek projekta. Podrobnejši vpogled v delovanje tima bi lahko pridobili tudi z intervjuji nekaterih članov tima, ki nastopajo v različnih vlogah.

LITERATURA IN VIRI

1. Agileforall. (2016). *Introduction to Agile*. Najdemo 1. decembra 2015 na spletnem naslovu http://agileforall.com/resources/introduction-to-agile/
2. Ambler, S.W. (2008). Has agile peaked? Let's look at the numbers. *Dr. Dobb's Journal.* Najdeno 1. junija 2015 na spletnem naslovu http://www.drdobbs.com/architecture-and-design/has-agile-peaked/207600615
3. Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. United States of America: Blue Hole Press.
4. Bajec, M., & Krisper, M. (2003). Agilne metodologije razvoja informacijskih sistemov. *Uporabna informatika, 11*(2), 68-76.
5. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, C.R., Mellor, M., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifest agilnega razvoja programske opreme. Najdeno 5. junija 2015 na spletnem naslovu http://agilemanifesto.org/iso/sl/
6. Bowes, J. (2014). Agile vs Waterfall: Comparing project management methods. *MANIFESTO*. Najdeno 1. marca 2016 na spletnem naslovu https://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/
7. Brandon, D. M. (2005). *Project management for modern information systems*. United States of America: IRM Press.
8. Bregar, L., Ograjenšek, I., & Bavdaž, M. (2005). *Metode raziskovalnega dela za ekonomiste: izbrane teme*. Ljubljana: Ekonomska fakulteta.
9. Carlson, R. (2013). Story Point Estimating. *Agile & Lean Education Associates*. Najdemo 1. novembra 2015 na spletnem naslovu http://www.a2zalea.com/wp-content/uploads/2014/02/Story-Point-Estimating_20140213.pdf
10. Cohen, S., Dori, D., de Haan, U., Cohen, S., De Haan, U., & Dori, D. (2010). A software system development life cycle model for improved stakeholders' communication and collaboration. *International Journal of Computers, Communications & Control*, *1*, 23-44.
11. Cohn, M. (2004). *User stories applied: For agile software development*. Boston: Addison-Wesley Professional.
12. Cohn, M. (2005). *Agile estimating and planning*. Pearson Education Inc: Prentice Hall.
13. Cohn, M. (2007). Advice on Conducting the Scrum of Scrums Meeting. *Scrum Alliance.* Najdeno 15. junija na spletnem naslovu https://www.Scrumalliance.org/community/articles/2007/may/advice-on-conducting-the-scrum-of-scrums-meeting
14. Devi, V. (2013). Traditional and Agile Methods: An Interpretation. *Scrumalliance.* Najdemo 30. marca 2016 na spletnem naslovu

https://www.Scrumalliance.org/community/articles/2013/january/traditional-and-agile-methods-an-interpretation
15. Dingsoyr, T., Dyba, T., & Abrahamsson, P. (2008, avgust). A preliminary roadmap for empirical research on agile software development. *IEEE*, 83-94.
16. Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology, 50*(9), 833-859.
17. Hastie, S. & Wojewoda, S. (2015). Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. *InfoQ.* Najdeno 11. maja 2016 na spletnem naslovu http://www.infoq.com/articles/standish-chaos-2015
18. Hughes, B., & Cotterell, M. (1999). *Software project management* (2nd ed.). Berkshire: McGraw-Hill International.
19. Johnson, H.L. (2012). How to play the Team Estimation Game. *AGILE LEARNING LABS.* Najdemo 12. decembra na spletnem naslovu http://www.agilelearninglabs.com/2012/05/how-to-play-the-team-estimation-game/.
20. Justin (2013). What is the Software Development Life Cycle (SLDC)? *Airbrake Blog*. Najdeno 12. maja 2016 na spletnem naslovu https://airbrake.io/blog/insight/what-is-the-software-development-life-cycle
21. Kniberg, H., & Skarin, M. (2010). *Kanban and Scrum-making the most of both.* United States of America: C4Media, Publisher of InfoQ.com.
22. Krisper, M., Rupnik, R., Zrnec, A., Bajec, M., Osojnik, R., Tomažič, R., & Silič, M. (2003). Enotna metodologija razvoja informacijskih sistemov. *Center Vlade RS za informatiko.* Najdeno 1. decembra 2015 na spletnem naslovu http://www2.gov.si/mju/emris.nsf/.
23. Laanti, M., Salo, O., & Abrahamsson, P. (2011). Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology, 53*(3), 276-290.
24. Lei, H., Ganjeizadeh, F., Jayachandran, P. K., & Ozcan, P. (2015). *A statistical analysis of the effects of Scrum and Kanban on software development projects.* United States of America: Robotics and Computer-Integrated Manufacturing.
25. Liu, S., & Liu, C. (2010, november). Management innovation of it project managers. *IEEE*, 62-65.
26. Lotz, M. (2013). Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?. Najdeno 1. marca 2016 na spletnem naslovu http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology
27. Mahnič, V., & Žabkar, N. (2012). Measuring progress of scrum-based software projects. *Elektronika ir Elektrotechnika, 18*(8), 73-76.
28. Mahnič, V. (2011). A case study on agile estimating and planning using scrum. *Elektronika ir Elektrotechnika, 111*(5), 123-128.

29. Mahnič, V. (2014). Improving Software Development Through Combination of Scrum and Kanban. *Recent Advances in Computer Engineering, Communications and Information Technology*, 281-288.
30. Mahnič, V., & Urevc, J. (2012). Ocena prednosti metode Scrum in njenih tipičnih praks. *Uporabna informatika, 10*(3), 184-194.
31. Mahnič, V., Georgiev S., & Jarc, T. (2009). Poučevanje metode Scrum v sodelovanju s podjetjem za razvoj programske opreme. *Mednarodna multikonferenca Informacijska družba*, 12(*A*), 243-255.
32. Covalent Marketing (2015). Software Developmet Lifecycle: Waterfall vs. Agile. *CovalentMarketing*. Najdeno 1. marca 2016 na spletnem naslovu http://covalentmarketing.com/blog/2015/10/19/software-development-lifecycle-waterfall-vs-agile-2/
33. Nasir, M. H. N., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific research and essays*, *6*(10), 2174-2186.
34. Pries, K. & Quigley, J. (2010). *Scrum Project Management* (1th ed.). Boca Raton: CRC Press.
35. Rozman, R., & Stare, A. (2008). *Projektni management ali ravnateljevanje projekta.* Ljubljana: Ekonomska fakulteta.
36. Schwaber, K. & Sutherland, J. (2011). Scrum Guide. Najdeno 20. maja 2015 na spletnem naslovu http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100
37. Schwaber, K. (1997). Scrum development process. *In Business Object Design and Implementation*, 117-134. London: Springer London.
38. Schwaber, K. (2004). *Agile project management with Scrum*. Washington: Microsoft press.
39. Sjøberg, D. I., Johnsen, A., & Solberg, J. (2012). Quantifying the effect of using kanban versus scrum: A case study. *IEEE, 29*(5), 47-53.
40. Solina, F. (1997). *Projektno vodenje razvoja programske opreme*. Ljubljana: Fakulteta za računalništvo in informatiko.
41. Stare, A. (2010). Vodenje ali management projekta. Najdeno 3. februarja 2016 na spletnem naslovu http://projektni-management.si/2010/09/30/vodenje-ali-management-projekta/
42. Stare, A. (2011). *Projektni management: teorija in praksa.* Ljubljana: Agencija Poti.
43. Sutherland, J. (2010). Scrum handbook. Najdeno 10. junija 2015 na spletnem naslovu http://www.scruminc.com/w p-content/uploads/2014/07/The-Scrum-Handbook.pdf
44. Terlecka, K. (2012, avgust). Combining Kanban and Scrum - Lessons from a Team of Sysadmins. *IEEE,* 99-102.
45. VersionOne. (2010). *State of agile Survey 2010*. Najdeno 8. marca 2016 na spletnem naslovu https://www.versionone.com/pdf/2010-state-of-agile-survey.pdf
46. VersionOne. (2013). *State of agile Survey 2013*. Najdeno 2. junija 2015 na spletnem naslovu http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

47. Williams, L. (2010). Agile Software Development Methodologies and Practices. *Advance In Computers*, *1*(80), 1-44.
48. Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. *IEEE, 36*(6), 39-43.
49. Wu, C. s., Chang, W. c., & Sethi, I. K. (2009). A Metric-Based Multi-Agent System for Software Project Management. *IEEE*, 3-8.

PRILOGA

KAZALO PRILOG

PRILOGA 1: Anketni vprašalnik

Pozdravljeni,

Sem študentka Ekonomske fakultete v Ljubljani in izvajam analizo projektov razvoja programske opreme izpeljanih z agilno metodologijo Scrum. Tvoje sodelovanje mi bo v pomoč pri izdelavi magistrskega dela, za kar se ti že v naprej iskreno zahvaljujem. Anketa je anonimna in ti ne bo vzela več kot 10 minut časa. Podatki bodo uporabljeni izključno za mojo raziskavo.

Melita Trpin

Q1 - Katera je vaša trenutna vloga v Scrum timu?
V kolikor imate več vlog, prosim označite vse vaše vloge.

- ☐ Lastnik izdelka (" ProductOwner")
- ☐ Skrbnik metodologije (" ScrumMaster")
- ☐ Razvijalec programske opreme =razvojni tim
- ☐ Arhitekt poslovnih rešitev = razvojni tim
- ☐ Svetovalec = razvojni tim
- ☐ Testni inženir = razvojni tim
- ☐ Project manager
- ☐ Drugo (vpiši):

Q2 - Na koliko projektih trenutno sodeluje vaš Scrum tim (za koliko strank/naročnikov razvijate programsko opremo znotraj vašega Scrum tima)?Prosim vpišite število projektov.

Q3 - Koliko ur dnevno namenite za naloge, ki so vodene skozi Scrum metodologijo (kolikšna je vaša kapaciteta)?

- ◯ moje ure niso vštete v kapaciteto
- ◯ do 2 uri
- ◯ 3 do 4 ure
- ◯ 5 do 6 ur
- ◯ 7,5 ur

Q4 - Koliko članov šteje vaš trenutni Scrum tim?(Ne upoštevajte Product Ownerja, Scrum Masterja in Project Managerja, če niso všteti v kapaciteto.)

- ◯ manj kot 3
- ◯ 3 -5
- ◯ 6 -8
- ◯ 9-11
- ◯ več kot 11

Q5 - Kako dolgi so vaši Sprinti?

◯ 1 teden
◯ 2 tedna
◯ 3 tedni
◯ 4 tedni
◯ Kombinacija tednov (prosim vpišite številki ločeni z vejico):
◯ Drugo (vpišite število tednov):

Q6 - Zakaj uporabljate kombinacijo tednov?

Q7 - Trenutno znotraj vašega Scrum tima razvijate programsko opremo tudi za stranko/naročnika, ki ima del funkcionalnosti že v produkciji?

◯ DA
◯ NE

Q8 - Ali vaš Scrum tim opravlja tudi vzdrževanje ("Support") za funkcionalnost, ki je že v produkciji?

◯ DA
◯ NE

Q9 - Kako izvajate/vodite vzdrževanje ("Support")? Katero metodologijo uporabljate?

◯ Scrum
◯ Kanban
◯ Slapovna/tradicionalna ("Waterfall") metodologija
◯ Drugo (vpiši):

Q10 - V kolikšni meri vaš tim sledi priporočilom/praksam, ki jih Scrum predpisuje? ***Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 "Sploh se ne*** *strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Naš Scrum tim je samoorganiziran tim.	◯	◯	◯	◯	◯
Uporabljene so vse vloge (Product Owner, Scrum Master in razvojni tim), ki jih Scrum predpisuje.	◯	◯	◯	◯	◯
Prioritete nalog določa in spreminja le Product Owner.	◯	◯	◯	◯	◯

	1	2	3	4	5
Pred ocenjevanjem nalog so uporabniške zgodbe jasno definirane.	◯	◯	◯	◯	◯
Za vsako uporabniško zgodbo se dosledno upošteva definicija dokončano ("definition of done").	◯	◯	◯	◯	◯
Pri planiranju naslednje iteracije Cikla se dosledno upošteva hitrost ("Velocity") razvoja tima.	◯	◯	◯	◯	◯
Znotraj tekočega Cikla se prioritete zahtev (nalog) ne spreminjajo.	◯	◯	◯	◯	◯
Dnevni sestanek ("Daily Meeting") se redno izvaja.	◯	◯	◯	◯	◯
Sestanek za planiranje Cikla ("Planning") se dosledno izvaja pred pričetkom naslednjega Cikla.	◯	◯	◯	◯	◯
Retrospektiva se izvaja redno ob koncu vsake iteracije Cikla.	◯	◯	◯	◯	◯
Sestanek za predstavitev iteracije ("how to demo" = "Sprint revision") se izvaja redno ob koncu vsake iteracije.	◯	◯	◯	◯	◯
Najprej se vedno oceni napor za posamezno nalogo.	◯	◯	◯	◯	◯
Za vsako uporabniško zgodbo se oceni tudi predvideno število dni/ur za dokončanje naloge.	◯	◯	◯	◯	◯

Q11 - Ali ste pred uvedbo Scrum metodologije imeli izobraževanje na temo Scruma?

◯ DA
◯ NE
◯ V času uvajanja Scrum metodologije še nisem bil član tima.

Q12 - Spodnje trditve se nanašajo na izobraževanje, ki ste ga bili deležni. *Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 "Sploh se ne strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Izobraževanje,ki sem ga bil deležen je bilo ustrezno in kakovostno, da lahko agilno metodologijo Scrum izvajam pravilno.	◯	◯	◯	◯	◯
Imel sem premalo znanja, ki bi mi pomagalo pri izvajanju Scrum metodologije.	◯	◯	◯	◯	◯
Ves čas mi je bila na voljo oseba, ki ima izkušnje in znanje o Scrumu in mi je tako lahko pomagala pri prehodu na agilno metodologijo Scrum.	◯	◯	◯	◯	◯

Q13 - Spodnje trditve se nanašajo na vašo oceno prednosti metode Scrum. ***Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 ''Sploh se ne*** *strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Scrum omogoča boljše oceno časa, ki je potreben za dokončanje naloge.	○	○	○	○	○
Scrum izboljšuje kakovost razvoja in testiranja.	○	○	○	○	○
Projekt je mogoče razdeliti na zaporedje obvladljivih delov.	○	○	○	○	○
Scrum zmanjšuje število napak, ki so narejene v razvoju.	○	○	○	○	○
Zaradi Scruma je prenos znanja znotraj tima večji.	○	○	○	○	○
Zaradi Scruma tim opravi planiranje in analizo nalog bolj temeljito.	○	○	○	○	○
Vsi člani tima imajo popoln vpogled v potek dela na projektu.	○	○	○	○	○
Zaradi Scruma so naloge končane hitreje.	○	○	○	○	○
Zaradi Scruma so prioritete dela vedno jasne.	○	○	○	○	○
Zaradi Scruma je odzivnost na spremembe večja.	○	○	○	○	○
Uporaba Scruma je prispevala k boljšim odnosom med naročnikom (uporabniki) in člani razvojnega tima (medsebojno zaupanje in presek znanja je večji).	○	○	○	○	○
Naročnik dobi novo funkcionalnost v dogovorjenem času.	○	○	○	○	○

Q14 - Ali menite, da ima Scrum metodologija katere slabosti? S kakšnimi težavami, vezanimi na Scrum metodologijo, ste se srečali pri delu? ***Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 ''Sploh se ne*** *strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Scrum določa preveč pravil.	○	○	○	○	○
Vodenje projekta z agilno metodologijo Scrum je zahtevnejše.	○	○	○	○	○

	1	2	3	4	5
Daljše trajanje projekta kot predvideno.	◯	◯	◯	◯	◯
Naročnik nima dovolj časa za stalno interakcijo s Scrum timom.	◯	◯	◯	◯	◯
Scrum je premalo agilen glede na zahteve naročnika.	◯	◯	◯	◯	◯
Prevelika preobremenjenost s sestanki, ki jih Scrum metodologija predvideva za vse člane Scrum tima.	◯	◯	◯	◯	◯
Sestanki pogosto trajajo predolgo in so izguba časa.	◯	◯	◯	◯	◯
Prevečkrat je dokumentacija zanemarjena.	◯	◯	◯	◯	◯

Q15 - Ali menite, da je Scrum primeren za razvoj vseh vrst programske opreme?

	Zagotovo ne	Ne	Mogoče	Da	Zagotovo da
Razvoj informacijske podpore delu in zabavi (programi in aplikacije za računalnik/telefon)	◯	◯	◯	◯	◯
Razvoj informacijske podpore poslovnim procesom	◯	◯	◯	◯	◯
Razvoj programa za izdelke (programiranje mikroprocesorjev)	◯	◯	◯	◯	◯
Implementacija produkta (Microsoft, Qlik, Mecoms...)	◯	◯	◯	◯	◯
Prilagajanje produkta poslovnim procesom naročnika	◯	◯	◯	◯	◯
Razvoj spletnih strani	◯	◯	◯	◯	◯
Razvoj spletnih trgovin	◯	◯	◯	◯	◯
Razvoj spletnih iger	◯	◯	◯	◯	◯
Avtomatizacija proizvodnje (robotika, kontrola...)	◯	◯	◯	◯	◯

Q16 - Da bi lahko svoje delo kar najbolje opravljali, kaj bi si želeli spremeniti v zvezi s Scrumom in svojim delom na splošno?

Q17 - Bi Scrum priporočili tudi drugim sodelavcem?

◯
◯ DA
NE

Q18 - Zakaj Scrum metodologije ne bi priporočali tudi drugim sodelavcem?

Q19 - Bi razmišljali o uporabi katere druge metodologije?

◯ Slapovna/tradicionalna ("Waterfall") metodologija
◯ Kanban
◯ Scrumban (kombinacija Scruma in Kanbana)
◯ Ekstremno programiranje
◯ Drugo:
◯ Ne bi razmišljal o uporabi katere druge metodologije

Q20 - Ali menite, da je možna kombinirana uporaba tradicionalnega in agilnega pristopa na enem projektu? Vpišite vašo oceno od 1 do 5, kjer 1 predstavlja povsem **tradicionalen** pristop, **5** pa popolnoma **agilen** pristop.

Q21 - Lahko pojasnite svojo oceno?

Q22 - S kakšnimi težavami na splošno se še vedno srečujete pri svojem delu? *Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 "Sploh se ne strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Dogovorjeni roki niso uresničeni.	◯	◯	◯	◯	◯
S strani vodstva so postavljeni nerealistični roki.	◯	◯	◯	◯	◯
Dražji projekt kot predvideno.	◯	◯	◯	◯	◯
Premalo časa za pripravo dobre specifikacije razvoja	◯	◯	◯	◯	◯

	1	2	3	4	5
posamezne naloge.					
Funkcionalnost, ki je predana stranki je pogosto slabo testirana.	◯	◯	◯	◯	◯
Ni pregleda nad celotnim projektom.	◯	◯	◯	◯	◯
S strani vodstva ni dovolj podpore za ustrezno vodenje Scruma.	◯	◯	◯	◯	◯
Zahteve se spreminjajo znotraj posameznega Cikla.	◯	◯	◯	◯	◯
Slabo postavljene prioritete in zaradi tega posledično pogosto spreminjanje prioritet.	◯	◯	◯	◯	◯
Na splošno se zahteve naročnika spreminjajo hitreje kot je dolžina Cikla.	◯	◯	◯	◯	◯
Naloge, ki pridejo v Sprint so slabo definirane.	◯	◯	◯	◯	◯
Ker ima naročnik del funkcionalnosti že v produkciji, je usklajevanje dela težje.	◯	◯	◯	◯	◯

Q23 - Kaj po vašem mnenju vpliva na uspeh projekta? ***Prosim odgovorite na spodnje trditve, kjer na lestvici od 1 do 5 pomeni 1 "Sploh se ne*** *strinjam" in 5 "Popolnoma se strinjam".*

	1	2	3	4	5
Jasno zastavljen seznam nalog za prihajajoči Sprint.	◯	◯	◯	◯	◯
Dobro analizirane in definirane uporabniške zgodbe.	◯	◯	◯	◯	◯
Pravilne ocene zahtevnosti posameznih uporabniških zgodb.	◯	◯	◯	◯	◯
Pravilna ocena hitrosti ("Velocity) razvoja tima na začetku vsake iteracije.	◯	◯	◯	◯	◯
Dosledno izvajanje sestankov za načrtovanje ("Planning") iteracije.	◯	◯	◯	◯	◯
Dosledno izvajanje vsakodnevnih sestankov ("Daily Meetings").	◯	◯	◯	◯	◯
Dosledno upoštevanje koncepta dokončanosti ("Definition of done").	◯	◯	◯	◯	◯
Dosledno izvajanje sestankov za predstavitev rezultatov iteracije ('How to Demo').	◯	◯	◯	◯	◯
Dosledno izvajanje retrospektive.	◯	◯	◯	◯	◯

	1	2	3	4	5
Redno sodelovanje naročnika z razvojnim timom.	◯	◯	◯	◯	◯
Realistične predstave naročnika glede programske opreme.	◯	◯	◯	◯	◯
Podpora vodstva je ključni dejavnik za uspeh agilnega razvoja.	◯	◯	◯	◯	◯
Postavljeni realistični roki za dostavo funkcionalnosti programske opreme.	◯	◯	◯	◯	◯

Q24 - Spol:

◯ Moški
◯ Ženski

Q25 - V katero starostno skupino spadate?

◯ do 25 let
◯ 25 -30
◯ 31 - 35
◯ 36 - 40
◯ 41 - 45
◯ 46 -50
◯ 51 - 55
◯ 56 - 60
◯ nad 60 let

Q26 - Koliko let se že ukvarjate z razvojem programske opreme?

◯ manj kot 2 leti
◯ 2 - 5
◯ 6 -10
◯ 11 - 20
◯ 21 - 30
◯ več kot 30 let

Q27 - Od tega z uporabo metodologije Scrum(koliko izkušenj že imate s Scrumom)?

◯ manj kot 1 leto
◯ 1 - 1,5

◯ 2 - 2,5

◯ več kot 2,5 leti

Q28 - Na katerih IT projektih ste v preteklosti že sodelovali?

Možnih je več odgovorov

- ☐ Projekt razvoja informacijske podpore delu in zabavi (programi in aplikacije za računalnik/telefon)
- ☐ Projekt razvoja informacijske podpore poslovnim procesom
- ☐ Projekt razvoja programa za izdelke (programiranje mikroprocesorjev...)
- ☐ Projekt implementacije produkta (Microsoft, Qlik, Mecoms...)
- ☐ Projekt lokalizacije produkta
- ☐ Projekt prilagoditev produkta poslovnim procesom naročnika
- ☐ Projekt razvoja spletnih strani
- ☐ Projekt razvoja spletnih trgovin
- ☐ Projekt razvoja spletnih iger
- ☐ Projekt avtomatizacije proizvodnje (robotika, kontrola...)