

UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

**PRENOVA PROCESA RAZVOJA IN VZDRŽEVANJA
PROGRAMSKE OPREME V IZBRANEM PODJETJU**

Ljubljana, julij 2016

ROBERT GOLOB

IZJAVA O AVTORSTVU

Podpisani Robert Golob, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Prenova procesa razvoja in vzdrževanja programske opreme v izbranem podjetju, pripravljenega v sodelovanju s svetovalcem dr. Mirom Gradišarjem

IZJAVLJAM

1. da sem predloženo delo pripravil 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 poskrbel, 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 pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;
7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil 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 25.07.2016

Podpis študenta: _____

KAZALO

UVOD	1
1 MANAGEMENT KAKOVOSTI, PROCESOV IN PROJEKTOV.....	4
1.1 Poslovno okolje podjetja	4
1.2 Management kakovosti.....	4
1.3 Management poslovnih procesov	5
1.4 Projektni management	6
1.4.1 Procesi, področja znanja ter življenjski cikli.....	6
1.4.2 Management portfelja, programa in projektov	6
1.4.3 Obvladovanje stroškov projekta	7
1.4.4 PMI standardi za ocenjevanje projektov	8
2 RAZVOJ PROGRAMSKE OPREME	8
2.1 Vpliv informatike na uspešnost poslovanja podjetij	8
2.2 Kakovost programske opreme	9
2.3 Prenova procesov razvoja programske opreme s pomočjo standardov	10
2.4 Projekti razvoja programske opreme	13
2.5 Vzdrževanje programske opreme	13
2.6 Storitveno orientirana arhitektura.....	16
2.7 Značilnosti in trendi razvojnih modelov.....	17
3 OCENJEVANJE RAZVOJA PROGRAMSKE OPREME.....	18
3.1 Vpliv ocenjevanja razvoja programske opreme na uspešnost projektov.....	18
3.2 Omejitve pri ocenjevanju razvoja programske opreme.....	22
3.3 Tehnike ocenjevanja programske opreme	24
3.3.1 Izbor tehnik ocenjevanja glede na fazo razvoja programske opreme.....	24
3.3.2 Individualna strokovna presoja.....	24
3.3.3 Ocenjevanje z dekompozicijo.....	25
3.3.4 Aditivna in multiplikativna analogija	25
3.3.5 Ocenjevanje s pomočjo korelacije	26
3.3.6 Skupinska strokovna ocena (Delphi tehnika)	26
3.3.7 Uporaba orodij (Analogija z algoritemsko podporo)	26
3.4 Ocenjevanje obsega programske opreme	27
3.4.1 Karakteristike ter možni pristopi k ocenjevanju obsega.....	27
3.4.2 Funkcijske točke	28

3.4.3	Točke primerov uporabe	30
3.4.4	Aplikacijske točke	30
3.4.5	Objektno orientirane metode za merjenje obsega	30
3.5	Ocenjevanje truda.....	30
3.6	Ocenjevanje časa oz. trajanja	32
3.7	Ocenjevanje stroškov	34
3.8	Vzpostavitev procesa ocenjevanja	34
3.9	Ocenjevanje truda in časa po aktivnostih.....	35
3.10	Ocenjevanje stroškov v fazi vzdrževanja	37
4	AGILNI RAZVOJNI MODEL	38
4.1	Vpliv metodologije razvoja programske opreme na uspešnost projektov	38
4.2	Metodologija Scrum.....	39
4.3	Izvedba ocenjevanja pri uporabi agilnih metodologij	41
4.4	Praktični primeri uvajanja agilnih metodologij.....	44
5	PRENOVA RAZVOJA V IZBRANEM PODJETJU	47
5.1	Predstavitev Izbranega podjetja, d.d.	47
5.2	Predstavitev obravnavanega problema.....	50
5.3	Pregled in predstavitev možnih ukrepov	52
5.3.1	Strateške, organizacijske ter procesne spremembe	52
5.3.2	Upravljanje sprememb informacijskih sistemov.....	53
5.3.3	Prenova procesa razvoja programske opreme – 1. faza, izvedeno	55
5.3.4	Ugotovitve po izvedeni 1. fazi prenove	61
5.3.5	Prenova procesa razvoja programske opreme – 2. faza, predlog	64
	SKLEP	71
	LITERATURA IN VIRI	74

PRILOGE

KAZALO TABEL

Tabela 1:	Tipi vzdrževalnih aktivnosti po standardu ISO 14764.....	15
Tabela 2:	Razporeditev truda po tipih aktivnosti	36
Tabela 3:	Poraba časa po posameznih aktivnostih razvoja	37
Tabela 4:	Matrika tipov sprememb in izdaj (verzij).....	38
Tabela 5:	Uspešnost projektov glede na izbrano metodologijo razvoja.....	39

KAZALO SLIK

Slika 1: Pomen standardov v procesu stalnih izboljšav.....	11
Slika 2: Piramida zrelosti podjetja glede na stopnjo uvedbe standardov	12
Slika 3: Gibanje števila zahtev za spremembe po uvedbi novega IS	14
Slika 4: Vpliv nerealiziranih zahtev ob zaključku projekta.....	14
Slika 5: Zgodovina modelov razvoja programske opreme.....	17
Slika 6: Spremenljiv obseg ter fiksni čas in viri v agilnih modelih.....	18
Slika 7: Verjetnost zaključka projekta na predviden rok.....	21
Slika 8: Lijak negotovosti.....	23
Slika 9: Razvojni cikel novega produkta po metodologiji "Stage Gate"	35
Slika 10: Različni nivoji planiranja v agilnih modelih.....	42
Slika 11: Spremljanje statusa zahteve za spremembo	56
Slika 12: Zahtevki za spremembo informacijskega sistema (Confluence).....	57
Slika 13: Repozitorij funkcionalnih specifikacij po poslovnih področjih	58
Slika 14: Plan novih funkcionalnosti posamezne verzije	58
Slika 15: Grafičen pregled mesečnega plana.....	59
Slika 16: Povezava med Jiro in SVN	59
Slika 17: Obvladovanje razvoja preko Jire.....	60
Slika 18: Spremljanje realizacije planiranih nalog v verziji preko Jire.....	60
Slika 19: Pregled realiziranih nalog po razvijalcih.....	62
Slika 20: Pregled dela razvijalcev na podpori produkciji.....	63
Slika 21: Gibanje prejetih zahtevkov iz produkcije.....	63
Slika 22: Obremenitev razvijalcev in analitikov z nudenjem podpore na 2. nivoju.....	64
Slika 23: Podrobno planiranje dnevnih aktivnosti z dodatkom Activity Timeline	64
Slika 24: Ocenjevanje obsega agilnega razvoja s tehniko pokra planiranja.....	66
Slika 25: Usklajevanje ocene obsega posamezne zahteve med člani ekipe	66
Slika 26: Načrtovanje sprinta v Jiri	67
Slika 27: Spremljanje statusa realizacije na nadzorni plošči Scrum-a	67
Slika 28: Načrtovanje več iteracij razvoja po modelu Scrum v Jiri	68
Slika 29: Vzporedno spremljanje realizacije preko nadzorne plošče Scrum-a	68
Slika 30: Različne opcije grafičnega prikaza napredka Scrum projektov	69
Slika 31: Graf preostalega časa sprinta	69
Slika 32: Nadzorna plošča Kanban - status zahtevkov v delu.....	70
Slika 33: Primer grafičnega spremljanja zahtevkov pri uporabi modela Kanban	70

UVOD

Problematika. Vsakodnevno smo soočeni s problemi in izzivi, ki jih globalizacija poslovanja postavlja pred gospodarske subjekte. Obstaja več različnih strategij odziva na zaostrene pogoje poslovanja. Ena od možnosti je strategija preobrata, ki je mnogokrat pogojena z reorganizacijo poslovanja ter optimizacijo poslovnih procesov, kar vpliva tudi na zahteve po izboljšavah ali zamenjavi informacijske podpore. V sedanjem času pa ne zadošča več, da informacijski sistem (v nadaljevanju IS) samo zagotavlja izvajanje vizije in strategije podjetja, temveč jo hkrati tudi sooblikuje (Van Grembergen, 2009). Ob vse hitrejšem napredku informacijsko-komunikacijske tehnologije je namreč lahko sodoben in učinkovit IS celo eden od ključnih dejavnikov razlikovanja in zmagovanja. To še posebej velja za storitvene dejavnosti, kjer je zadovoljstvo uporabnikov in strank velikokrat odvisno prav od zanesljivosti in kakovosti informacijskih storitev (Cooper & Edgett, 1999). Zaradi nenehnih sprememb v poslovnem okolju in vse večje konkurence naraščajo zahteve po novi ter boljši programski opreми, hkrati pa se krajša razpoložljiv čas za prehod v produkcijo. Ker so v obdobju gospodarske recesije in zaostrenih pogojev poslovanja sredstva za investicije v informacijsko tehnologijo še dodatno omejena, je izrednega pomena, da se razpoložljiva sredstva namenijo za projekte z najvišjo dodano vrednostjo.

Za razvoj aplikativne programske opreme (v nadaljevanju PO) se torej namenja znatna finančna sredstva, hkrati pa se veliko projektov konča neuspešno. Glavne spremenljivke, ki determinirajo projekte razvoja PO so obseg projekta, stroški, čas razvoja in kvaliteta programske opreme (Bittner, 2007). Po zadnjem poročilu Standish Group (Hastie & Wojewoda, 2015) se samo 29 % projektov zaključi v predvidenem časovnem in finančnem obsegu ter z zahtevano kakovostjo in funkcionalnostmi. Kar 19 % projektov pa se zaključi neuspešno oziroma se jih predčasno prekine. Pri vseh ostalih projektih so preseženi predvideni roki ter stroški, obseg funkcionalnosti pa je znatno okrnjen. Ključno za uspeh projekta je sicer predvsem sodelovanje uporabnikov, podpora vodstva ter jasne zahteve, poznani pa so tudi razlogi za neuspeh. Ti so predvsem: neobjektivno ocenjevanje potrebnih virov in časa, uveljavljanje (napačnih) odločitev s pozicije moči ter namerno zavajanje glede stroškov in tveganj. Z verodostojno oceno predvidenega obsega projekta, potrebnega truda, časa ter stroškov projekta lahko neposredno izločimo vpliv negativnih dejavnikov, hkrati pa posredno pozitivno vplivamo tudi na najpomembnejše parametre, ki zagotavljajo uspešen zaključek projekta.

V strokovni literaturi zasledimo različne metodologije in napotke (McConnell, 1996), kako pristopiti k razvoju PO, da bo le-ta čim bolj zadostila poslovnim in uporabniškim zahtevam. Ključna ugotovitev je, da ne obstaja univerzalen pristop, ki bi dal najboljši rezultat v vseh možnih okoljih in primerih. Kljub temu pa je v zadnjem času opazna prevlada agilnih metodologij razvoja, ki poleg ostalih prednosti zagotavljajo tudi najhitrejšo povrnitev investicij v razvoj programske opreme (Leffingwell, 2011).

Na osnovi zgodovinskih podatkov je podana ocena, da se v celotnem življenjskem ciklu posamezne aplikacije za vzdrževalne aktivnosti porabi okoli 70 % sredstev (Pigoski, 1997). Zato je potrebno ocenjevanju truda ter stroškov nameniti ustrezno pozornost tudi v tej fazi (Stutzke, 2005) ter vzpostaviti učinkovit proces vzdrževanja programske opreme (April & Abran, 2008).

Poslovni izid podjetja je odvisen od izbire pravih informacijskih projektov ter njihovega uspešnega zaključka. Na uspešnost poslovanja, zadovoljstvo zaposlenih in strank prav tako vpliva nivo kvalitete informacijskih storitev skozi celoten življenjski cikel posamezne aplikacije. Hkrati so viri za izvajanje razvojnih in vzdrževalnih aktivnosti običajno omejeni, zahtevani roki izvedbe pa mnogokrat zelo kratki. Če že v fazi načrtovanja ni izvedena ocena potrebnega truda, se lahko zaradi nerealnih pričakovanj to odrazi na kvaliteti in okrnjenem obsegu končnega produkta, mnogokrat pa kljub povečanim kadrovskim virom tudi na zamujenih rokih. Zaradi tega je potrebno zagotoviti, da se vsi deležniki zavedajo pomena (Jones, 1998) in poznajo različne tehnike ocenjevanja obsega in stroškov razvoja programske opreme (Stutzke, 2005). Sočasno pa je potrebno uvesti čim bolj učinkovit proces razvoja programske opreme, ki bo omogočal hitro realizacijo poslovnih zahtev ter zagotavljal visok nivo kvalitete programske opreme.

Namen in cilj. Namen magistrske naloge je izboljšati proces načrtovanja informacijskih projektov predvsem na osnovi bolj verodostojne ocene obsega projekta ter potrebnega truda oziroma napora za realizacijo zahtev.

Cilji naloge so:

- uvedba formalnega ocenjevanja obsega in potrebnega truda za razvoj informacijskih sistemov v različnih razvojnih fazah,
- priprava izhodišč za oblikovanje portfelja informacijskih projektov ter določanje prioritet na osnovi ocenjenih stroškov in koristi,
- dopolnitev obstoječe metodologije razvoja programske opreme v opazovani organizaciji z najprimernejšimi pristopi iz proučenih metodologij in standardov.

Metodološki pristop. Pri izdelavi magistrske naloge sem uporabil znanje, pridobljeno med podiplomskim študijem dopolnjeno z večletnimi izkušnjami v informatiki.

V procesu priprave magistrskega dela sem najprej sistematično in temeljito predelal tujo in domačo strokovno literaturo z obravnavanih področij. S pomočjo teorije sestava, indukcije in sinteze sem nova spoznanja s proučevanih področij sestavil v celovito sliko obravnavane problematike.

S pomočjo analize in indukcije sem na osnovi poznavanja problemov v konkretni organizaciji podal predloge za možne izboljšave obstoječih procesov načrtovanja in razvoja programske opreme ter izvajanja informacijskih storitev.

Struktura poglavij. Vsebina naloge je razdeljena v pet poglavij, v katerih so po posameznih sklopih najprej predstavljena teoretična izhodišča posameznega področja znanja, ki sem jih nato na osnovi priporočil strokovnjakov s teh področij uporabil v praktičnem delu naloge.

Prenova procesa razvoja in vzdrževanja programske opreme še zdaleč ni samo tehničen problem, ki bi bil omejen zgolj na področje službe za informatiko. Za uspešno izvedbo prenove procesa je potrebno pridobiti podporo najvišjega managementa podjetja ter zagotoviti konstruktivno sodelovanje vseh ostalih strokovnih področij podjetja, ki v procesu razvoja in vzdrževanja PO nastopajo kot deležniki. V prvem poglavju so zato najprej opisani osnovni koncepti prenove procesov in managementa, ki lahko pripomorejo k učinkovitejšemu izvajanju procesov v informatiki ter višji poslovni vrednosti razvitih rešitev.

Za razvoj PO je značilno veliko število neuspešnih projektov, nepravilno delovanje aplikacij ter posledično nezadovoljstvo uporabnikov s kakovostjo izdelkov. Razvoj PO se lahko ostalim tradicionalnim inženirskim panogam po kakovosti izdelkov in učinkovitosti obvladovanja procesov približa predvsem z bolj dosledno rabo področnih standardov ter priporočenih metodologij. Osnovna izhodišča teh področij so opisana v drugem poglavju.

Osnovni namen razvoja PO je izdelava informacijske podpore, ki omogoča podjetju čim višje donose iz osnovne dejavnosti. Zaradi tega je potrebno tudi pri razvoju PO ocenjevati poslovno vrednost posameznih projektov razvoja ter učinkovito obvladovati portfelj informacijskih projektov. Predpogoj za vrednotenje donosnosti posameznega projekta razvoja je razmerje med poslovno vrednostjo ter stroški razvoja. Poslovno vrednost ocenijo naročniki, stroške razvoja in vzdrževanja pa oceni področje informatike. Strokovna izhodišča za vzpostavitev procesa ocenjevanja ter izvajanja ocenjevanj so opisana v tretjem poglavju.

Glede na najnovejše analize so agilne metodologije razvoja PO po deležu uspešno zaključenih projektov v znatni prednosti pred ostalimi ne glede na velikost projekta. Zaradi tega so v tem poglavju predstavljena osnovna izhodišča agilnega razvoja ter kratek pregled agilnih metodologij. Poudarek je na opisu najbolj razširjene agilne metodologije Scrum, opisani pa so še rezultati raziskav po uvedbi agilnih metodologij v dveh večjih podjetjih.

V petem poglavju so najprej opisane glavne karakteristike izbranega podjetja, nato pa je podrobno opisan problem, ki ga želimo s predlaganimi pristopi v magistrski nalogi

odpraviti oz. vsaj omiliti. Opisani so že izvedeni ukrepi ter njihove pomanjkljivosti, na koncu pa je podan predlog uvedbe nove metodologije razvoja PO v izbranem podjetju.

1 MANAGEMENT KAKOVOSTI, PROCESOV IN PROJEKTOV

1.1 Poslovno okolje podjetja

Na delovanje in uspešnost podjetja ključno vplivajo zunanji dejavniki. Pučko (2008) opozarja, da je podjetje potrebno obravnavati kot podsistem v okviru nekega širšega sistema. Vpliv okolja na podjetja proučuje sistemska teorija, ki organizacijo obravnava kot živ, dinamičen in v okolje odprt sistem, ki na vходу sprejema materijo, energijo in informacije, jih predela v želeno stanje in na izhodu ponovno odda okolju. Ključni element takšnega sistema je povratna zanka, ki s pomočjo kontrolnih mehanizmov primerja in regulira dejansko stanje na izhodu z želenimi ciljnim vrednostmi (Mihelič & Škafar, 2008). Mulej (2008) opozarja, da je sistemski pristop ključen za celovit vpogled v pričakovanja kupcev ter aktivnosti konkurence. Če se podjetja zadovoljijo s trenutnim stanjem in ne spodbujajo inovativnega razmišljanja, bodo slej kot prej postala nekonkurenčna ter bodo izrinjena s trga. Podobno meni tudi Collins (2001), ki trdi, da je dobro sovražnik odličnega. Bojnec (2007) navaja, da je primarni cilj podjetja sicer res dolgoročno preživetje na trgu, da pa mora za doseg tega cilja podjetje lastnikom ustvarjati pričakovan donos in je torej temeljni cilj maksimiranje dobička.

Za doseganje zastavljenih ciljev je v podjetju potrebno vzpostaviti ustrezno hierarhijo dejavnosti (Marolt & Gomišček, 2005). Vizija, poslanstvo, politike in vrednote podjetja vplivajo na oblikovanje strateških ciljev. Za doseg teh ciljev je potrebno pripraviti neko krovno strategijo, ki jo preko planov ter podrejenih ciljev pretvorimo v izvedbo. Prvi korak pri pretvorbi vizije v dosegljive plane je pretvorba vizije v nekaj ključnih strategij. Nato za vsako od teh opredelimo merljive strateške cilje ter jih glede na razpoložljive vire in pričakovane poslovne učinke umestimo v letne plane. Dolgoročne cilje je potrebno podrobneje opredeliti, jih umestiti v kratkoročne plane ter konkretne projekte, preko katerih bodo izvedeni. Pri umeščanju strateških ciljev v letne plane in pri izbiri projektov mora komunikacija potekati v obe smeri, tako da imajo tudi izvajalci možnost podati svoje videnje, opozoriti na potrebne vire ter izpostaviti tveganja (Juran & De Feo, 2010).

1.2 Management kakovosti

Na poslovni izid podjetij ima pomemben vpliv učinkovit management kakovosti (Kern-Pipan, 2010). Marolt (2005) management kakovosti definira kot vsoto dejavnosti, povezanih s kontrolo, zagotavljanjem in izboljševanjem kakovosti. Pri uvajanju managementa kakovosti je potrebno osredotočenje na kupca vgraditi v vizijo, poslanstvo, politike, strategije, dolgoročne in kratkoročne poslovne cilje in projekte. Na prvem mestu

pa je seveda zadovoljstvo lastnikov, za kar je potrebno vzpostaviti pravo razmerje med kvaliteto, stroški in dobičkom (Juran, 2010).

Ločimo več stopenj kakovosti (Juran, 2010):

- kontrola kakovosti (angl. *Quality Control*),
- zagotavljanje kakovosti (angl. *Quality Assurance*),
- stalno izboljševanje kakovosti (angl. *Continuous Quality Improvement*).

Prvi dve stopnji se ukvarjata s kakovostjo proizvoda, novejši pristopi pa se osredotočajo na procese, ki so ključni za doseganje boljše kvalitete končnih proizvodov. Za uvajanje stalnih izboljšav obstaja več uveljavljenih pristopov (ISO 9001:2000, BSC, Six sigma ...) ter nagrad, ki spodbujajo njihovo uporabo (Malcolm Baldrige National Quality Award, Europe Quality Award, Global Excellence Model Council ...).

1.3 Management poslovnih procesov

Kovačič, Jaklič, Indihar Štemberger in Groznik (2004) opisujejo management poslovnih procesov kot poslovni pristop obvladovanja sprememb celotnega življenjskega cikla procesa. Zaradi sprememb v okolju ter tehnološkega napredka je potrebno nekoč ločene naloge ponovno združiti in opazovati kot integrirane poslovne procese (Hammer & Champy, 1995). Pri prenovi poslovanja najprej preverimo obstoječe poslovne procese, nato pa z namenom zniževanja stroškov ter povečanja kakovosti izdelkov izvedemo njihovo korenito spremembo. Boutros (2014) proces definira kot zbir aktivnosti, ki porablja vire za pretvorbo vhodov v koristne izhode. Cilj izboljševanja procesov je kontinuirano povečevanje produktivnosti in kvalitete, kar v končni fazi vodi k doseganju poslovnih ciljev organizacije (večji dobiček, nižji stroški, hitrejši razvoj ...).

Boutros (2014) opozarja na pomanjkljivosti managementske funkcijske organizacijske strukture, kjer vsak oddelek oz. organizacijska enota stremi k lastnim ciljem in ni širšega pogleda na koristi podjetja. Ravno nasprotno pa procesna managementska organizacijska struktura poudarja potrebo po sodelovanju različnih funkcijskih enot ter je osredotočena na zahteve odjemalcev. Zaradi prednosti, ki jih prinaša tak pristop, vedno več organizacij uvaja procesno orientirano arhitekturo.

Ključne spremembe pri uvajanju procesne strukture so (Boutros, 2014):

- organizacijska struktura se spremeni iz funkcijske v procesno;
- zid med oddelki se poruši, pride do sodelovanja;
- enostavna opravila izginjajo, posameznik opravlja kompleksnejše naloge;
- ni več toliko nadzora, posameznik ima več pristojnosti;

- posameznik je bolj prilagodljiv, bolj učinkovit;
- ni več pomemben prispevek k funkcijski enoti, temveč vrednosti za kupca;
- kvaliteta se vgrajuje v izdelek;
- managerji niso več nadzorniki, temveč mentorji;
- rezultati dela so pomembnejši od vloženega dela.

1.4 Projektni management

1.4.1 Procesi, področja znanja ter življenjski cikli

Kot sredstvo za doseganje strateških ciljev podjetja se pogosto uporablja projektni način dela. Za učinkovito izvajanje projektov je potrebno imeti znanja s konkretnega vsebinskega področja, poznavanje standardov in predpisov, razumevanje projektnega okolja, znanje in spretnosti splošnega managementa ter spretnosti v medosebnih odnosih (PMI, 2008). Življenjski cikli projekta (študija izvedljivosti, začetna faza, ena ali več vmesnih faz, končna faza) se izvajajo preko 5 skupin procesov projektnega vodenja:

- skupina zagonskih procesov: opredelitev in odobritev projekta;
- skupina procesov planiranja: opredelitev in natančnejši opis ciljev ter ukrepov za doseg teh ciljev;
- skupina procesov izvajanja: zagotovitev in koordinacija virov, izvedba in integracija vseh aktivnost, potrebnih za uresničitev predvidenih ciljev projekta;
- skupina procesov spremljanja in kontroliranja: spremljanje in merjenje napredka, odmikov od plana, izvajanje korektivnih ukrepov;
- skupina procesov končanja: formalen sprejem izdelka, storitve ali drugega rezultata projekta ter zaključek projekta.

Področja znanja projektnega managementa so razdeljena v več sklopov, ki so prikazani v Prilogi 1. Pregled procesov projektnega managementa po skupinah ter glede na področje znanja pa je prikazan v Prilogi 2.

1.4.2 Management portfelja, programa in projektov

Za strateško skladnost, finančno uspešnost izbranih projektov ter zagotavljanje virov je potreben učinkovit management portfelja. Predpogoj za učinkovito obvladovanje portfelja projektov je, da so strateški cilji jasno izraženi. Biti morajo dovolj specifični, merljivi, izvedljivi, usmerjeni k rezultatom ter časovno opredeljeni (Moore, 2010). Pogosto je strateška skladnost projektov celo pomembnejša od donosnosti, ne sme pa se zanemariti niti tveganj. Zaželeno je, da je nabor idej in predlogov za nove projekte čim obsežnejši, saj imamo le tako možnost izbire najboljših. Porter (v Moore, 2010) trdi, da je bistvo strategije ravno v tem, da imaš možnost izbire, česa ne boš delal. Metrike uspešnosti projekta, način

spremljanja in izvajanja korektivnih aktivnosti je potrebno definirati vnaprej, zgrešene projekte pa čim prej ustaviti (Brown, 2008). Strateški in taktični nivo se prepletata pri izvajanju managementa programa, v sklopu katerega je potrebno projektno delo usklajevati z operativnim delom. Projektni management se primarno ukvarja z obvladovanjem obsega, časa in stroškov projekta. Obstaja več dobrih praks obvladovanja projektov, ki zagotavljajo učinkovitost in uspešnost projekta. Za vse dobre prakse pa je značilno, da dajejo velik poudarek upravljanju sprememb. Deležniki namreč pogosto spremenijo mnenje o zahtevah, obsegu ter ostalih parametrih projekta. Zaradi tega je potrebno vzpostaviti sistem, ki bo zagotavljal, da so sprejete samo spremembe, ki so nujne in podjetju prinašajo korist. Ključen element obvladovanja sprememb je nadzor nad spremembami in analiza vplivov posamezne spremembe (Saladis & Kerzner, 2009).

Saladis in Kerzner (2009) sta prepoznala 8 osnovnih elementov kontroliranja sprememb:

- opredelitev osnovnih izhodišč projekta,
- razumevanje okoliščin, ki vplivajo na pojav sprememb,
- identifikacija možnih virov sprememb,
- pregled in odobritev zahtev za spremembe,
- obvladovanje odobrenih sprememb,
- posodobitev vseh projektnih planov, na katere vpliva odobrena sprememba,
- dokumentiranje pridobljenih izkušenj ob izvajanju sprememb,
- zavrnitev zahtev za spremembe, ki nimajo zadostne dodane vrednosti, so predrage ali imajo druge negativne posledice.

1.4.3 Obvladovanje stroškov projekta

Znani so številni primeri, ko so se projekti bodisi zaradi nepravilne ocene ali neustreznega vodenja projekta končali z znatno preseženimi stroški, hkrati pa so običajno preseženi še predvideni roki in ni dosežena niti zahtevana kvaliteta (Moustafaev, 2011). Zaradi tega je obvladovanju stroškov projektov potrebno nameniti posebno pozornost. Saladis in Kerzner (2009) ločita 3 glavne procese:

- ocenjevanje stroškov (angl. *Cost Estimating*),
- budget (proračun) stroškov (angl. *Cost Budgeting*),
- kontrola stroškov (angl. *Cost Control*).

Med oceno potrebnega dela in želenim rokom običajno obstaja razhajanje. Oceno s pomočjo tehnične ekipe pripravi projektni manager, zahtevane roke pa postavijo stranke oz. vodilni management podjetja. Razlika med oceno potrebnega časa in zahtevanim rokom predstavlja projektno tveganje. Zaradi tega je potrebno uskladiti sprejemljiv rok za vse strani, ki je za izvajalce zavezujoč (Saladis & Kerzner, 2009).

1.4.4 PMI standardi za ocenjevanje projektov

Številnim napačnim ocenam projektov bi se lahko izognili, če bi se pred oz. med trajanjem projekta izvajale vse aktivnosti, ki so predlagane s strani PMI (angl. *Project Management Institute*). V sklopu ocenjevanja je potrebno ovrednotiti vire za aktivnosti, trajanje ter stroške projekta (PMI, 2011).

Poznati je potrebno življenjske cikle ocenjevanja (PMI, 2011):

- priprava na ocenjevanje (angl. *Prepare to Estimate*),
- oblikovanje ocen (angl. *Create Estimates*),
- obvladovanje ocen (angl. *Manage Estimates*),
- optimizacija procesa ocenjevanja (angl. *Improve Estimating Proces*).

Z ustreznim planiranjem lahko v veliki meri zmanjšamo tveganja, povezana z napačno oceno zaradi evolucije zahtev. Pred pričetkom ocenjevanja je potrebno dokumentirati uporabljen pristop. V tej fazi so identificirani potrebni viri, sprejmejo se odločitve o uporabljeni tehniki. Vse ocene projektov temeljijo na predpostavkah, ki vsebujejo določeno stopnjo negotovosti ter tveganja. Ocene je zaradi tega potrebno dopolnjevati, ko z razvojem projekta pridobivamo nove informacije. V inicialni fazi ima lahko projekt +/- 50 % natančnost, v kasnejših fazah pa lahko dosežemo natančnost do +/- 10 % (PMI, 2011).

Vzroki za razhajanje ocen so različni, nekaj najpomembnejših je (PMI, 2011):

- podrobnejše razumevanje zahtev (angl. *Evolution of Requirements*),
- potrjene zahteve za spremembe (angl. *Approved Change Requests*),
- problemi med izvajanjem projekta (angl. *Operational Problems*),
- napačne ocene (angl. *Faulty Estimating*).

2 RAZVOJ PROGRAMSKE OPREME

2.1 Vpliv informatike na uspešnost poslovanja podjetij

Sočasno z napredkom informacijske tehnologije se večja tudi obseg sredstev, ki jih podjetja vlagajo v IT. V zadnjih 50 letih so investicije podjetij v informacijsko tehnologijo narasle od začetnih 5 % do današnjih 50 % (Carr, 2003). Posebno intenzivna je bila rast internetnih podjetij med letoma 1995 in 2000. Nerealna pričakovanja (Howcroft, 2001) so se razblinila s počenjem »dot-com« balona v letih 2000 in 2001 ter strmim padcem vrednosti delnic teh podjetij. Pojavile so se trditve (Carr, 2003), da IT zaradi svoje razširjenosti in splošne dostopnosti ne zagotavlja več strateške prednosti pred konkurenco, ampak predstavlja samo še stroške in tveganja. To je spodbudilo številne raziskave, ki so dokazovale ravno

nasprotno. Kohli in Grover (2008) se tako strinjata, da samo uvedba sodobne informacijske tehnologije ne zadošča, temveč mora biti le-ta usklajena s strategijo podjetja, preko prenove poslovnih procesov pa mora omogočati višjo produktivnost. V raziskavi, ki je zajela več kot 400 globalnih podjetij (Mithas, Tafti, Bardhan, & Goh, 2012), so z analizo finančnih podatkov potrdili, da imajo naložbe v IT pozitiven vpliv na dobičkonosnost predvsem z omogočanjem višjih prihodkov. Ugotovili so, da imajo naložbe v IT večjo donosnost kot investicije v raziskave in razvoj ali marketing.

Read (2009) opozarja na rezultate raziskave, po kateri vodilni managerji menijo, da vodstvo informatike ne uporablja pravih meritev in kazalnikov za predstavitev uspešnosti informatike. Meri se zadovoljstvo uporabnikov, spremlja se poraba sredstev in produktivnost zaposlenih, premalo poudarka pa je namenjenega prikazu omogočanja novih poslovnih priložnosti in ekonomskih učinkov IT investicij.

2.2 Kakovost programske opreme

Kan (1995) opozarja, da kupci tudi od kompleksne PO zahtevajo visoko kakovost. Axelrod (2013) pa izpostavlja, da bi bilo potrebno več pozornosti pri zagotavljanju kakovosti nameniti tudi nefunkcionalnim lastnostim PO. Za izboljšanje kvalitete končnega izdelka je ključno, da optimiziramo že sam proces razvoja. Glede na specifično projekta je potrebno izbrati ustrezen razvojni proces, metode in pristope ter uporabiti primerna orodja in tehnologije. Nekateri modeli so sicer primernejši za specifične tipe projektov oz. za specifične okoliščine, vendar je uspeh primarno odvisen od zrelosti implementacije procesa.

Učinkovit razvoj PO lahko zagotovimo le s stalnim spremljanjem in izvajanjem meritev (Kan, 1995), ki so ključne za približevanje razvoja PO klasičnim inženirskim disciplinam. Najbolj razširjene so produktne, procesne in projektne metrike. Produktne metrike opisujejo posamezen produkt, npr. njegov obseg, kompleksnost, lastnosti, kvaliteto. Procesne metrike se uporabljajo za izboljšavo procesa, merimo npr. uspešnost odpravljanja napak, odzivni čas za odpravo napake ... Projektne metrike pa opisujejo značilnosti celotnega projekta, npr. število razvijalcev, stroške, produktivnost.

Napake se v končni produkt vnašajo skozi celoten cikel razvoja. Jones (2014) ocenjuje, da nastane v fazi zajema in specifikacij uporabniških zahtev 14 % napak, med načrtovanjem 30 %, v fazi kodiranja 40 %, po 7 % napak pa dodata še nekvalitetna dokumentacija ter slabi popravki. Z raznimi metodami testiranja lahko odpravimo od 80 do 90 % napak, za boljše rezultate pa je potrebno uporabiti tudi inšpekcije in statično analizo, ki ju izvajamo že pred samim testiranjem.

2.3 Prenova procesov razvoja programske opreme s pomočjo standardov

S prenovo procesov razvoja PO se poskuša zagotoviti višjo kvaliteto, nižje stroške ter krajši čas razvoja. Kot pri vseh spremembah je tudi pri izboljšavah procesa razvoja PO možno pričakovati odpor sodelujočih, predvsem če se z uvedbo formalnih postopkov znatno poveča obseg birokratskih opravil. V primeru ustrezno izvedenih prenov procesa pa lahko pričakujemo številne pozitivne učinke (Lavallee & Robillard, 2012):

- manj incidentov in kritičnih situacij, manj izrednih naporov in nadur razvijalcev, manj odpovedi delovnega razmerja;
- manj izrednih situacij vodi k večjemu zadovoljstvu, višji morali;
- boljša porazdelitev odgovornosti.

Najbolj uveljavljeni okviri in modeli so CMM, CMMI, ISO/IEC 15504 (SPICE), Six-Sigma, PSP, TSP, QIP, TQM, IDEAL, PDCA. Lahko se odločimo tudi za kombinacijo več uveljavljenih modelov, nadgradnjo oz. prilagoditev uveljavljenih modelov ali pa izgradimo lasten model (Unterkalmsteiner, Gorschek &, Moinul, 2012).

Najpogostejše uporabljena metoda za izboljšavo procesa razvoja PO je CMMI. Najbolj poznan model implementacije je IDEAL, ki konkretnije opisuje korake za izvedbo. Aaen (2003) opozarja, da slepo sledenjem navodilom iz modela ne zagotavlja uspeha, ampak je potrebno prenovo procesa prilagoditi konkretni organizaciji. Pino, Pedreira in Garcia (2010) izpostavljajo, da model IDEAL ni primeren za manjše organizacije, ker zahteva preveč časa in napora, zato je zanje bolj primeren agilni pristop. Niazi, Wilson in Zowghi (2005) navajajo, da nezadovoljivo stanje na področju procesov razvoja PO ni posledica pomanjkanja standardov ali modelov, temveč nepoznavanja primernih strategij.

Uspešnost prenove procesa se presoja s pomočjo različnih metrik s področij kvalitete in stroškov (Unterkalmsteiner, Gorschek &, Moinul, 2012). Ker so za izvedbo prenove potrebna znatna sredstva, je smiselno meriti donosnost tega procesa. Glede na rezultate študij se donosnost prenove procesov razvoja PO giblje med 4 in 10 % (Solingen, 2004).

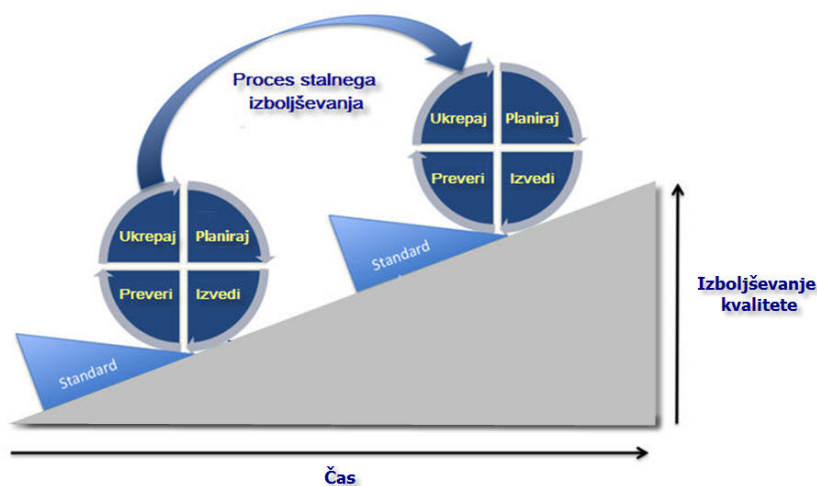
V strokovni javnosti so prisotni namigi, da razvoja PO ne moremo uvršati med inženirske discipline. Moore (2006) inženirski proces definira kot niz povezanih aktivnosti, ki s porabo virov ustvarijo izdelek, ki zadovolji vhodne potrebe. Kontrolo in korektivne aktivnosti izvajamo preko povratne zanke, ki temelji na meritvah karakteristik procesa, izdelka in zadovoljstva naročnika.

Uspešnost razvoja PO lahko znatno izboljšamo z uporabo standardov, v katerih so opisani procesi in prakse za učinkovitejši razvoj in vzdrževanje PO. Drug vir potrebnih znanj

predstavlja vodnik SWEBOK, v sklopu katerega so temeljna dognanja s tega področja zbrana v 10 področij znanja ter 8 povezanih disciplin, ki postavljajo razvoj PO v okvire inženirskih disciplin (Moore, 2006).

S področjem standardov razvoja PO se ukvarjata mednarodni organizaciji IEC (angl. *International Electrotechnical Commission*) ter ISO (angl. *International Organization for Standardization*). Z namenom harmonizacije standardov informacijske tehnologije sta ustanovili skupno delovno telo JTC 1 (angl. *Joint Technical Committee*), v sklopu katerega deluje pododbor SC 7, ki pokriva področje razvoja PO. Najpomembnejša organa na področju sistemov in programske opreme sta ISO/IEC JTC 1 / SC 7 ter IEEE S2ESC. Simbolično je pomen standardov v procesu stalnih izboljšav prikazan na Sliki 1.

Slika 1: Pomen standardov v procesu stalnih izboljšav



Povzeto in prirejeno po A. Monnappa, Pioneers of Project Management: Deming vs Juran vs Crosby, 2015, slika 3.

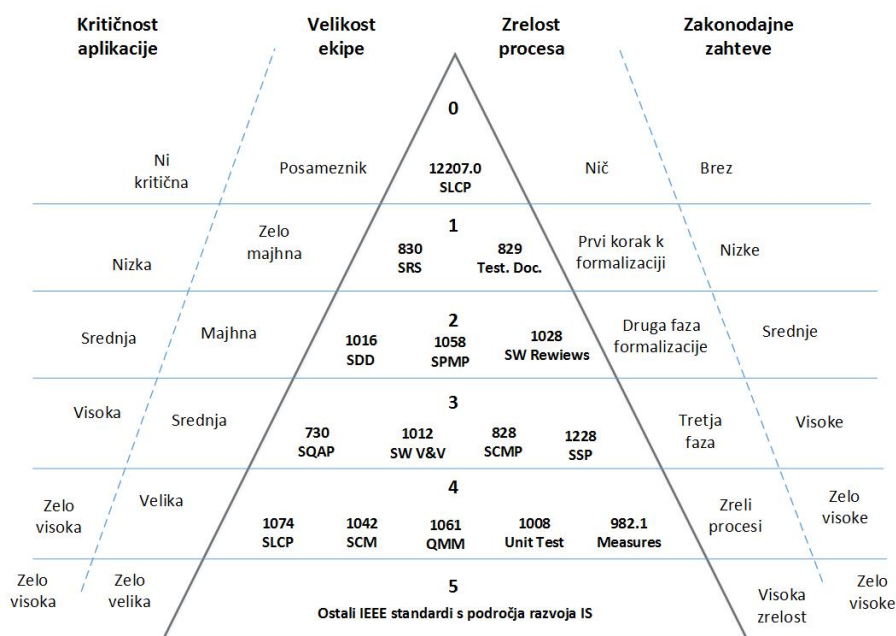
Eden od možnih pristopov k optimizaciji procesa razvoja PO je uvedba in uporaba standardov s tega področja. Opis IEEE standardov s področja razvoja PO je zbran v Prilogi 3. Vpliv stopnje uvedbe standardov na zrelost podjetja je prikazan na Sliki 2.

Nivo 0: Definirani so procesi na krovnem nivoju, vendar uporaba standardov ni formalizirana. Opremljeni so procesi za verifikacijo in validacijo ter upravljanje s konfiguracijami, vendar na mnogo nižjem nivoju kot v specializiranih standardih.

Nivo 1: Zahteve za programsko podporo so specificirane in zapisane, prav tako se pripravlja testna dokumentacija. Ta nivo je primeren samo za nekritične aplikacije, kjer ni strogih regulatornih zahtev. Primeren je kot prvi korak k formalizaciji procesa.

Nivo 2: Formalizirana ja priprava projektnih planov, dokumentiranja načrtovanja programske podpore ter sistematičnih pregledov končnega produkta. Predstavlja velik korak pri optimizaciji procesov glede na nivo 1. Ni priporočljivo, da bi hkrati poizkušali uvesti več standardov kot tiste, ki nastanejo s kombinacijo teh dveh nivojev. Zadovoljivo je opredeljena verifikacija in validacija, zagotavljanje kakovosti ter upravljanje s konfiguracijami. Opredeljeno je, kako bo programska oprema zagotavljala zahtevane lastnosti in funkcionalnosti na dovolj specificiran način, da je možno pričeti z razvojem oz. programiranjem ter pripravo testov za posamezen modul. Predstavlja tudi osnovo za pripravo integracijskih testov in testov enot. Podrobno je opredeljenih pet najpomembnejših tehnik izvajanj pregledov. Tehnični pregledi ter inšpekcije so izredno pomembni predvsem v zgodnjih fazah projekta ter so predpogoj, da lahko vodje kompetentno prevzamejo odgovornost za nadzor in usmerjanje projekta.

Slika 2: Piramida zrelosti podjetja glede na stopnjo uvedbe standardov



Povzeto in prirejeno po M.E.C. Schmid, Implementing the IEEE Software Engineering Standards, 2000, str. 27, slika 3.7.

Nivo 3: Ta nivo je potrebno doseči za vse kritične aplikacije. Uvaja standarde s področja zagotavljanja kvalitete, verifikacije in validacije (v nadaljevanju V&V), upravljanja konfiguracij ter varnosti. Poleg priprave predlog za dokumente planiranja teh aktivnosti je potrebno na tem nivoju zagotoviti tudi izvajanje planiranih aktivnosti.

Nivo 4: Na tem nivoju so podprti vsi ključni IEEE standardi, organizacija je dosegla potrebno zrelost.

Nivo 5: Na tem nivoju so implementirani vsi standardi s področja razvoja PO.

2.4 Projekti razvoja programske opreme

Obstajajo različni pogledi na prepletenost razvoja PO in projektnega managementa. Če sta trdno sklopljena, projektni management temelji oz. je odvisen od metodologij razvoja PO. Če pa sta šibko sklopljena, metodologije razvoja PO nimajo pomembnega vpliva na vodenje ter izvajanje projektov. Za kateri pristop se bomo odločili, je odvisno od velikosti organizacije, zrelosti obvladovanja informacijske tehnologije, uporabe dobrih praks razvoja PO, posameznega projekta, izkušenj in znanja članov ekip (Chemuturi, 2010).

Lahko pa uveljavljene principe iz domene znanja projektnega managementa s pridom uporabimo tudi pri razvoju PO. Investicije v informacijsko tehnologijo (v nadaljevanju IT) namreč predstavljajo znaten delež vseh investicij podjetij. Da začetne investicije skupaj s stroški vzdrževanja ne presežejo koristi, morajo biti aktivno upravljane in obvladovane preko celotnega življenjskega cikla. V sklopu obvladovanja IT portfelja je tako potrebno proaktivno identificirati, vrednotiti, razvrščati in upravljati razmerja med tveganji, stroški in koristmi (Mazlish & Handler, 2005).

Obvladovanje IT portfelja ima 8 korakov (Mazlish & Handler, 2005):

- priprava plana: postavitve ciljev in izhodišč za izvajanje procesa;
- planiranje: nadgradnja koraka ena in priprava investicijske strategije;
- kreiranje: priprava inventure trenutnih in predvidenih IT projektov;
- vrednotenje: izvajanje meritev in primerjava trenutnega stanja s postavljenimi cilji;
- uravnavanje: uravnava in razporeja resurse med posameznimi projekti;
- komuniciranje: pravočasno in pravilno obveščanje vseh deležnikov;
- nadgrajevanje sistema: identifikacija vlog, odgovornosti in procesov;
- vrednotenje procesa: preverjanje učinkovitosti izvajanja.

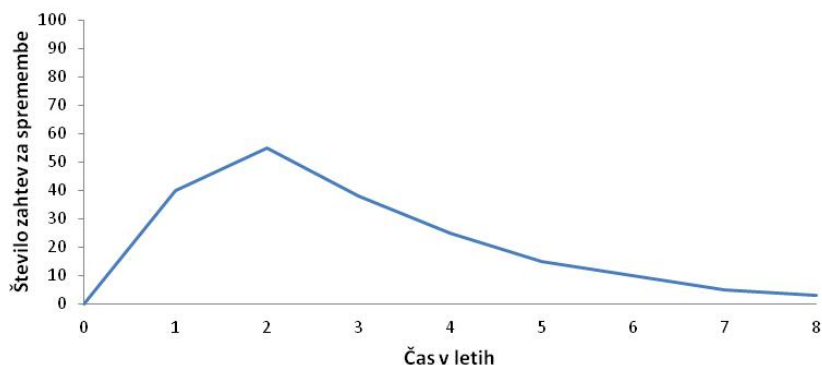
V ločenem vodniku za projektni management, ki je specializiran za področje vodenja projektov razvoja PO (PMI, 2013b) so podrobneje pojasnjene razlike med vložki, izločki ter uporabljenimi orodji in tehnikami. Osnovna področja znanja projektnega managementa za razvoj PO po PMI vodniku so prikazana v Prilogi 4.

2.5 Vzdrževanje programske opreme

Razvojne in vzdrževalne aktivnosti je dokaj težko ločiti. Glavna razlika je v tem, da so razvojne aktivnosti običajno vodene projektno, z znanim obsegom, opredeljenimi roki in sredstvi. Vzdrževalne aktivnosti pa temeljijo na naključnih, nepredvidljivih dnevnih zahtevkih s strani uporabnikov oz. naročnikov in morajo zagotavljati neprekinjeno izvajanje informacijskih storitev (April & Abran, 2008). Običajno imata tako naročnik kot izvajalec cilj, da se razvoj PO čim hitreje zaključi. Pri tem se pogosto pozablja, da se

(lahko) zaradi hitenja povečujejo dolgoročni stroški vzdrževanja (Schach, 1990). Po uvedbi novega IS v produkcijo je zaradi visokih pričakovanj uporabnikov značilno, da v prvih letih po uvedbi ter spoznavanju novega IS število zahtev po spremembah raste (Slika 3).

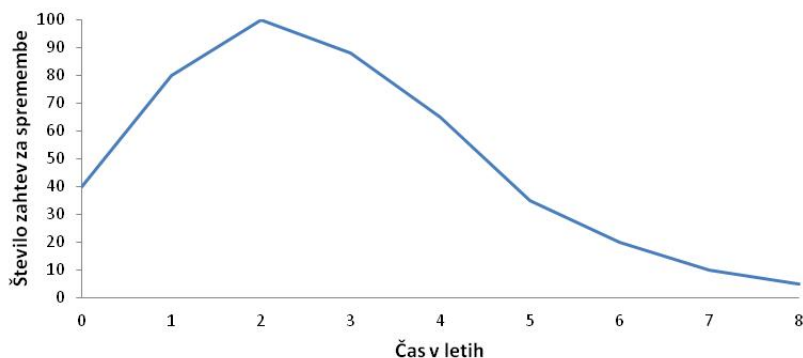
Slika 3: Gibanje števila zahtev za spremembe po uvedbi novega IS



Povzeto in prirejeno po T.M. Pigoski, Practical Software Maintenance, 1997, str. 126, slika 9.1.

Če ob prehodu v produkcijo niso bile realizirane vse projektne zahteve (Slika 4), pa je situacija lahko še slabša (Pigoski, 1997).

Slika 4: Vpliv nerealiziranih zahtev ob zaključku projekta



Povzeto in prirejeno po T.M. Pigoski, Practical Software Maintenance, 1997, str. 126, slika 9.2.

Zaradi tega je potrebno v prvih letih zagotoviti dovolj številčno ekipo za vzdrževanje, ki bo lahko zadostila pričakovanja uporabnikov. Po 2 do 3 letih število zahtev za spremembe običajno upade tudi do 50 % in se lahko število vzdrževalcev zmanjša. Eden od načinov gladkega prehoda iz razvoja v vzdrževanje je, da odgovorni za vzdrževanje že v fazi razvoja in prenosa v produkcijo prevzamejo oz. izvajajo upravljanje s konfiguracijami (Pigoski, 1997).

Vzdrževanje PO je zahtevno zaradi dveh razlogov. Naročniki običajno zahtevajo več novih funkcionalnosti kot jih je možno realizirati, zato mora management poskrbeti za presojo ekonomske upravičenosti posameznih zahtev. Še večji problem pa predstavlja dejstvo, da v fazi razvoja ni namenjeno dovolj pozornosti arhitekturni zasnovi ter standardom kodiranja, ki bi omogočali enostavno vzdrževanje. Večina razvijalcev namreč nima izkušenj s specifikami vzdrževalnih aktivnosti, prav tako pa se ne zaveda visokih stroškov, ki so posledica neoptimalne zasnove PO.

Po Swansonu (v Pigoski, 1997) delimo vzdrževanje na:

- korektivno (angl. *Corrective*): odprava napak;
- prilagoditve (angl. *Adaptive*): prilagoditve na spremembe v okolici;
- izboljšave (angl. *Perfective*): nove zahteve naročnikov in uporabnikov.

Standard ISO 14764 uporablja nekoliko drugačno razvrstitev tipov vzdrževalnih aktivnost, kar je prikazano v Tabeli 1.

Tabela 1: Tipi vzdrževalnih aktivnosti po standardu ISO 14764

	Popravki	Nadgradnje
Proaktivne	Preventivni	Izboljšave
Reaktivne	Korektivni	Prilagoditve

Povzeto in prirejeno po A. April & A. Abran, Software Maintenance Management, 2008, str. 23, slika 1.10.

Boehm (v April & Abran, 2008) navaja, da je kar 50-90 % vseh stroškov za programsko opremo porabljenih za vzdrževanje. Tudi po mnenju Pigoskega (1997) skupni stroški vzdrževanja dosegajo 60-80 % vseh stroškov za PO, od tega je preko 80 % stroškov namenjenih nadgradnjam oz. izboljšavam.

Po mnenju naročnikov so za vzdrževanje IS značilni visoki stroški, počasna realizacija, nejasne prioritete ter različen pogled na le-te s stališča uporabnikov in informatike (Pigoski, 1997). Po mnenju razvijalcev pa je glavna težava v pomanjkljivem načrtovanju arhitekture in slabih praksah programiranja ter pomanjkljivi oz. neobstoječi dokumentaciji. April in Abran (2008) pa sta mnenja, da ima najpomembnejši vpliv na učinkovito vzdrževanje PO management tega procesa.

Pri vzdrževanju PO morajo imeti napake, javljene iz produkcijskega okolja vedno prednost pred ostalimi zahtevami za spremembe. Vse ostale zahteve za spremembe bi morale iti pred realizacijo preko več stopenj ocen in potrjevanj. Najprej mora vzdrževalec oceniti potreben obseg zahteve ter potreben trud. V primeru, da je ocenjen trud večji od

dogovorjenega za vzdrževalne aktivnosti, se zahtevek preusmeri na razvojno ekipo, kjer se obravnava kot manjši razvojni projekt. Prag si lahko vsako podjetje izbere glede na svoje značilnosti. ISBSK (angl. *International Software Benchmarking Standards Group*) in UKSMA (angl. *United Kingdom Software Metrics Association*) predlagata kot ločnico med vzdrževalnim in razvojnim delom 5 č/d (April & Abran, 2008).

2.6 Storitveno orientirana arhitektura

Jurič in Pant (2008) opozarjata na pomanjkljivosti tradicionalnega življenjskega cikla razvoja aplikacij:

- zajem uporabniških zahtev in specifikacij (angl. *Requirements Specifications*),
- analiza in načrtovanje (angl. *Analysis and Design*),
- izvedba (angl. *Implementation*),
- testiranje (angl. *Testing*),
- nameščanje (angl. *Deployment*).

Te faze si običajno sledijo zaporedno, občasno delno vzporedno, uporabimo pa lahko tudi iterativni in/ali inkrementalni pristop. Vsi ti pristopi predvidevajo, da bodo specifikacije pripravljene vnaprej, zato niso najbolj ustrezni za situacije, ko se zahteve med razvojem spreminjajo. Ker z obstoječimi metodologijami in arhitekturo IS ne zmoremo dovolj hitro slediti zahtevam poslovnih uporabnikov, je potreben prehod na novejšo arhitekturo, ki to omogoča. Primer takšne arhitekture, ki zmanjšuje semantično vrzel med opisom poslovnih procesov ter programsko kodo, je SOA (angl. *Service Oriented Architecture*), (Jurič & Pant, 2008).

V podjetjih običajno obstaja mnogo ločenih IS (interni razvoj, nakup zunanjega IS po zahtevah naročnika, komercialni sistemi ERP, CRM ...), ki običajno nimajo skupne arhitekture in temeljijo na različnih tehnologijah. Zaradi poslovnih zahtev je potrebno vse te IS integrirati, z naraščanjem števila IS pa se eksponentno povečuje število teh integracij (Jurič & Pant, 2008).

Podjetja za uspeh in konkurenčno prednost na trgu potrebujejo učinkovite poslovne procese. Ker so poslovni procesi in IS običajno tesno sklopljeni, je ob spremembah poslovnih procesov potrebno spreminjati tudi IS. Sodobni IS so praviloma zelo kompleksni in je za implementacijo sprememb potreben določen čas, kar je v nasprotju s pričakovanji poslovnih uporabnikov. Zato so pogosto prisotni pritiski, da se čas razvoja skrajša na minimum. Poslovne zahteve imajo praviloma višjo prioriteto, kar onemogoča kvalitetno izvedbo vseh potrebnih aktivnosti v razvojnem ciklu programske opreme in lahko vodi k neoptimalnim arhitekturnim rešitvam. Zaradi tega je vzdrževanje takšnega IS vedno bolj kompleksno, kar še podaljšuje čas za uvedbo novih sprememb. Če ne prekinemo s takšno

prakso, obstaja velika verjetnost, da IS slej kot prej sploh ne bo več zmogel zadovoljevati zahtev poslovnih uporabnikov (Jurič & Pant, 2008).

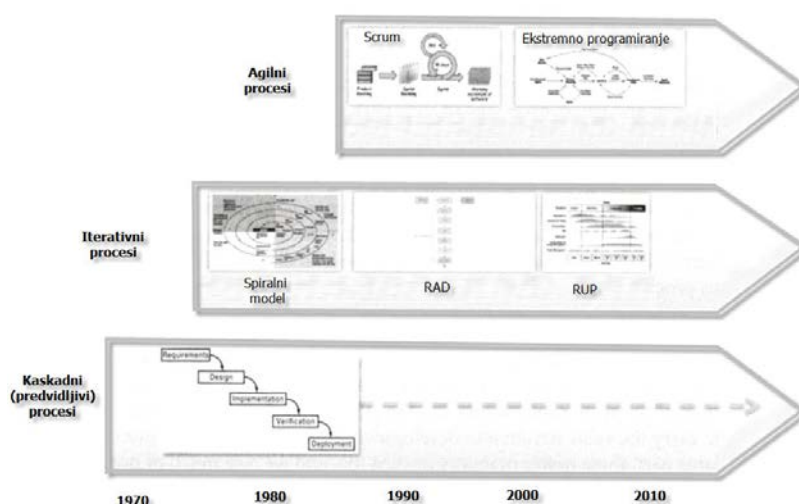
Storitveno orientirana arhitektura nam omogoča hitrejše prilagoditve programske podpore ob spremembah poslovnih procesov, ki jih običajno izvedemo preko naslednjih korakov (Jurič & Pant, 2008):

- identifikacija procesa, ki ga želimo avtomatizirati,
- modeliranje trenutnega procesa,
- avtomatizacija procesa brez večjih sprememb,
- optimizacija procesa,
- spremljanje in nadzor spremenjenega procesa.

2.7 Značilnosti in trendi razvojnih modelov

Najdaljšo zgodovino ima zaporedni oz. kaskadni razvojni model, ki je še vedno živ in se bo zagotovo uporabljal tudi še v bodoče (Slika 5). Kasneje so bili razviti iterativni modeli (Spiralni, Rad, RUP), ki jih podrobneje opisujeta Larman in Basili (2003). Zahtevajo manj dokumentacije za uporabniške zahteve, ki se po tej metodologiji zapisujejo s primeri uporabe (angl. *Use Cases*). Po letu 1990 se pojavijo agilni procesi. Z obširno raziskavo preko 1000 IT projektov je bilo potrjeno (Thomas, v Leffingwell, 2011), da podrobno opisovanje vseh zahtev na začetku daljših projektov ni več ustrezno. Zaradi hitro spreminjajočega okolja je namreč ravno tak pristop odgovoren za veliko število neuspešnih projektov.

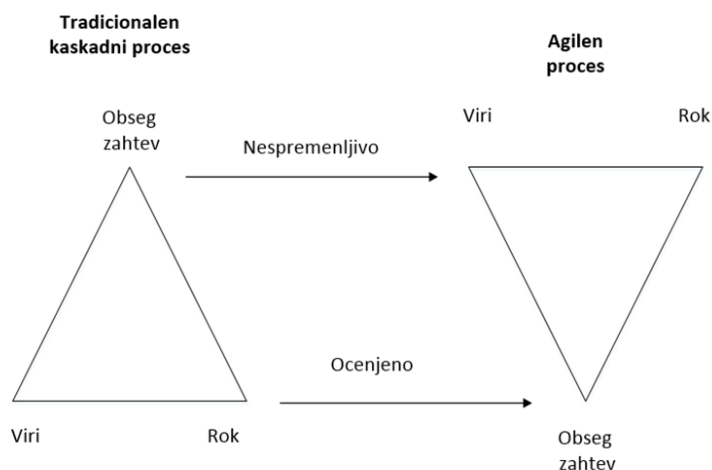
Slika 5: Zgodovina modelov razvoja programske opreme



Povzeto in prirejeno po D. Leffingwell, *Agile Software Requirements : Lean Requirements Practices for Teams, Programs, and the Enterprise*, 2001, str. 4, slika 1.1.

Za kaskadni model obstaja »jekleni trikotnik«, ki ga opredeljujejo obseg, stroški in čas. Ker je v jeklenem trikotniku edina spremenljivka kvaliteta, je seveda le-ta najbolj na udaru. Kot je prikazano na Sliki 6, pa so agilne metode »jekleni trikotnik« obrnile na glavo in dokazale, da so tanjši procesi razvoja PO bolj učinkoviti, skrajša se čas za povrnitev investicij, lahko pa dosežemo tudi dodatne pozitivne učinke, če smo z nekim produktom prvi na trgu (Leffingwell, 2011).

Slika 6: Spremenljiv obseg ter fiksni čas in viri v agilnih modelih



Povzeto in prirejeno po D. Leffingwell, Agile Software Requirements : Lean Requirements Practices for Teams, Programs, and the Enterprise, 2001, str. 16, slika 1.8.

3 OCENJEVANJE RAZVOJA PROGRAMSKE OPREME

3.1 Vpliv ocenjevanja razvoja programske opreme na uspešnost projektov

Za projekte z vseh področij je značilno, da pogosto prekoračijo zastavljene roke ter stroške, prav tako pa končni izdelek mnogokrat ne izpolnjuje vseh zahtev glede funkcionalnosti in kvalitete. Odstotek projektov, ki niso zaključeni v predvidenem roku, presegajo predvidene stroške ter ne izpolnjujejo v celoti pričakovanj naročnikov, je še posebej visok na področju razvoja programske opreme oziroma vseh sistemov, ki so kritično odvisni od njenega delovanja. Po mnenju Jonesa (1998) je to v veliko primerih posledica nekvalitetnih ter preveč optimističnih ocen, ki jih brez poznavanja ocenjevalskih tehnik in orodij izvajajo premalo usposobljeni kadri. Podobnega mnenja je tudi Venkataraman (2008), ki neupoštevanje vseh vrst stroškov projekta ter posledično nepravilno končno oceno navaja kot enega od temeljnih razlogov, zakaj IT projekti ne dosegajo pričakovanih ekonomskih ciljev. Berlin, Raz, Glezer in Zviran (2009) opozarjajo, da prenizke ocene vodijo v prekoračitve predvidenih stroškov projekta, nevarne pa so tudi previsoke ocene, zaradi katerih izpuščamo poslovne priložnosti.

Zaradi kritične odvisnosti uspeha projekta od pravilne ocene bi morali biti po mnenju Jonesa (1998) z osnovami ocenjevanja stroškov seznanjeni tako projektni managerji kot tudi specialisti za zagotavljanje kakovosti, analitiki, arhitekti in programerji. Običajno so večji projekti tako kompleksni, da ocenjevanja ni več možno izvajati ročno, zato predlaga uporabo specializiranih orodij za oceno stroškov. Projekti, pri katerih je bila uporabljena katera od formalnih metod ocenjevanja obsega, truda in stroškov, običajno odstopajo od predvidenih vrednosti zgolj minimalno. Jones (1998) opozarja še na nekaj področij, kjer je poznavanje ocenjevalskih tehnik ter priprava korektnih ocen izrednega pomena. Pogosto je pravilnost ocen oziroma vrednost programske opreme potrebno dokazovati na sodiščih v primerih sporov zaradi kršitev pogodbenih odnosov med dobaviteljem in naročnikom (prekoračeni stroški razvoja, povečani stroški zaradi dodatnih funkcionalnosti, ki niso bile opredeljene v pogodbi). Na sodiščih se obravnavajo tudi primeri, ko so bili določeni ponudniki izbrani prav na osnovi prvotno (namerno) nerealne ocene stroškov. Pravilna ocena je pomembna tudi z vidika davčne obravnave investicij v programsko opremo ter pravilnega načrtovanja denarnega toka za financiranje projekta.

Pomembnosti tega področja so se zavedali že v zgodnjih fazah razmaha informacijske tehnologije, ko so utemeljitelji tega strokovnega področja postavili temelje, na katerih še danes sloni večina najbolj uporabljenih metodologij. Že leta 1975 je A. Albrecht predstavil metodologijo funkcijskih točk (angl. *Function Point Metric*), leta 1981 pa je B. Boehm v knjigi »*Software Engineering Economics*« predstavil algoritme modela COCOMO (angl. *Constructive Cost Model*). Njuno delo sta v naslednjih letih dopolnila še T. DeMarco ter C. Simon, uporabo funkcijskih točk v praksi pa je populariziral predvsem C. Jones.

Potrebno je ločiti med meritvijo, oceno, ciljem, planom in zavezo. Z meritvami s pomočjo merilnih priprav dokaj točno določimo velikost posamezne lastnosti opazovanega objekta. Z oceno opredelimo približno vrednost veličine, ki opisuje določen objekt. Cilji so pravilom povezani s poslovnimi cilji oz. zahtevami (zakonodajna zahteva, finančne omejitve, pričakovanji trga, lastnikov ...). Tem ciljem pa so običajno podrejeni plani. Zaveza pomeni, da se z zahtevanimi roki strinjajo tudi izvajalci in potrjujejo, da bodo v predvidenem času zagotovili zahtevane funkcionalnosti z dogovorjenim nivojem kvalitete. V praksi se pogosto dogaja, da vodstvo podjetja od informatikov pričakuje zgolj plan za doseg poslovnih ciljev, brez predhodne izvedbe ocen izvedljivosti. Razlog je v tem, da poslovna stran nima dovolj tehničnih znanj, da bi prepoznala razliko med oceno, cilji, planom in zavezo (McConell, 2006).

Ocenjevanje mora biti nepristranski analitičen proces, medtem ko je planiranje že po svoji naravi pristransko, saj ga pripravljamo z namenom, da bi dosegli zastavljene poslovne cilje. Ocene so sicer osnova za pripravo planov, ni pa nujno, da so plani enaki ocenam. Od pravih ocen je odvisnih več planskih postavk: priprava terminskega plana, identifikacija kritične poti, členitev dela, določanje prioritet, opredeljevanje iteracij razvoja ... V

primeru, da se zahtevani cilji precej razlikujejo od ocen, je potrebno to v planih upoštevati kot visoko stopnjo tveganja (McConell, 2006).

Stutzke (2005) navaja, da je potrebno ocenjevanje in merjenje izvajati kot stalen proces. Z meritvami lahko vrednotimo in poročamo trenutno stanje proizvoda, procesa ali projekta ob določenem času. S ponavljanjem meritev pa lahko zaznamo trende in napovemo gibanje v bodoče. V primeru razhajanj med planiranimi in dejanskimi vrednostmi je to znak za ukrepanje.

Ocenjevanje in merjenje je potrebno izvajati tako pri načrtovanju in izgradnji kot tudi ob samem izvajanju ter predvsem ob morebitnih spremembah v času življenjske dobe izdelka. Najprej je potrebno definirati veličine, ki jih bomo ocenjevali oz. merili. V primeru sistemov, ki vsebujejo programsko opremo oz. v samostojnih projektih razvoja programske opreme se kot artefakti pojavljajo strojna oprema, programska oprema, dokumentacija, nastavitve konfiguracij, izobraževanja, pomoč pri uporabi ... Objekt opazovanja je lahko tudi proces ali aktivnost (planiranje, testiranje, nameščanje ...), (Stutzke, 2005).

Stutzke (2005) je mnenja, da je za pravočasno zaznavanje problemov na projektu ključna vloga ocenjevanj in meritev. Dobre ocene so ključ do uspešnega projekta/proizvoda. So osnova za sprejemanje odločitev, definiranje izvedljivih ciljev ter pripravo planov. Z meritvami zagotavljamo podatke za preverjanje skladnosti s specifikacijami in plani, sprejemanje odločitev, spreminjanje načrtov in planov ter izboljšanje bodočih ocen in procesov.

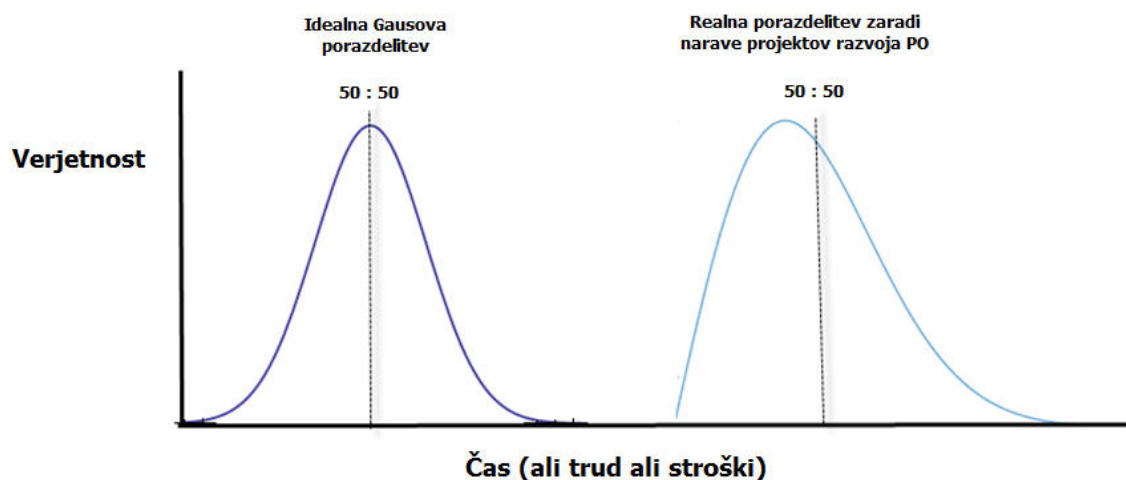
Ocenjevanje uporabimo, kadar zaradi objektivnih razlogov ni možni izvesti meritev. Ko je opazovani objekt nedosegljiv (npr. v stalni uporabi), ko bi bila meritev predraga ali prenevarna, ali pa objekt še niti ne obstaja. Za obstoječe objekte lahko izvedemo ocenitev direktno, ali pa jo izračunamo na osnovi drugih karakteristik, ki jih lahko direktno merimo ali vsaj lažje in bolj natančno ocenimo. Pri tem so nam v veliko pomoč zgodovinski podatki. Bolj težavno je ocenjevanje objektov, ki niso bili še nikoli izdelani ali aktivnosti, ki se še niso izvajale. To je žal običajno za vse nove proizvode in sisteme. V teh primerih mora ocenjevalec predvideti oz. napovedati prihodnost (Stutzke, 2005).

Kar tri četrtine projektov razvoja PO preseže ocenjen čas za zaključek projekta. Že iz tega lahko sklepamo, da je nemogoče s 100 % verjetnostjo podati oceno s točno določeno vrednostjo trajanja. Pri ocenjevanju trajanja projektov razvoja PO je prisotnih toliko negotovosti, da je nemogoče podati točno oceno. Zato obstaja samo določena verjetnost, da bo projekt zaključen ob predvidenem času. To lahko ponazorimo s klasičnim diagramom normalne porazdelitve verjetnosti (Gaussova funkcija). Ker pa takšna simetrična porazdelitev zaradi narave projektov ni možna, je krivulja na levi strani omejena. Ni pa možno napovedati, koliko se bo projekt podaljšal, zato se na desni strani krivulja razteza

zelo daleč. S črtkano črto je v tem diagramu prikazana vrednost, kjer je 50 : 50 % možnost, da bo projekt končan prej ali pozneje (Slika 7).

Mnenja o tem, kako natančno je možno oceniti trajanje projekta, so različna. Jones (1998) je mnenja, da je pri dobro obvladljivih projektih možno pripraviti oceno z 10 % natančnostjo. Conter, Dunsmore in Shen pa so že leta 1986 zapisali, da lahko ob pravilnem pristopu v 75 % primerov podamo oceno s +/- 25 % natančnostjo (v Stutzke, 2005).

Slika 7: Verjetnost zaključka projekta na predviden rok



Povzeto in prirejeno po S. McConell, Software Estimation, 2006, str.7-8, slike 1.2 in 1.3

Priprava kar najbolj točne ocene je pomembna z več vidikov. Pri preohlapni oceni smo izpostavljeni »Parkinsonovem zakonu« (delo bo zapolnilo ves razpoložljiv čas) ter »študentskemu sindromu« (z delom se odlaša do zadnjih trenutkov, nato kljub hitenju ni možno dokončati v roku). Po drugi strani pa nerealne ocene onemogočajo učinkovito planiranje. V primeru preveč optimističnih ocen se običajno pomanjkanje časa poskuša nadoknaditi že v začetnih fazah s površno pripravljenimi specifikacijami ter pomanjkljivim načrtovanjem, kar povzroča naknadne spremembe v fazi razvoja. Zaradi tega so stroški in čas razvoja še večji oz. daljši, kot če bi že v začetku pravilno ocenili potreben obseg in čas. Ko s projektom zamujamo, se namreč pojavijo še številne dodatne aktivnosti, ki jih sicer sploh ne bi bilo: sestanki z managementom, ponovne ocene, pojasnila ter opravičila naročnikom in uporabnikom, priprava začasnih nepopolnih rešitev, usklajevanje prioritete funkcionalnosti, popravljanje napak, povzročenih zaradi iskanja hitrih, tehnično neoptimalnih rešitev (McConell, 2006).

Iz navedenega sledi, da bomo najboljše rezultate projekta dosegli pri najbolj natančni oceni. Realne ocene namreč prinašajo mnogo koristi. Omogočajo realno spremljanje plana,

boljšo kvaliteto ter učinkovitejšo koordinacijo dela. Povečujejo kredibilnost razvojnih ekip ter omogočajo zgodnje prepoznavanje vrzeli med poslovnimi cilji in pričakovanimi rezultati projekta, s tem pa pravočasno izvedbo potrebnih korektivnih ukrepov (McConell, 2006).

3.2 Omejitve pri ocenjevanju razvoja programske opreme

V primerih, ko predvidevamo dogajanje v prihodnosti, se srečamo z več omejitvami (Stutzke, 2005).

Opustitev (angl. *Omission*). Obstaja možnost, da ocenjevalec ne identificira vseh dejavnikov, ki vplivajo na oceno oziroma upošteva napačne predpostavke. Ocenjevalec je odvisen od informacij, ki jih dobi od deležnikov v projektu. Naročniki pogosto ob začetku projekta ne znajo pojasniti vseh funkcionalnosti, ki jih potrebujejo oz. želijo. Specifikacije, zapisane v prostem tekstu, so nejasne in si jih je mogoče razlagati na različne načine. Pomanjkanje znanja s specifičnega področja lahko razvijalce zavede v preveč poenostavljeno razumevanje zahtev. Tudi managerji lahko napačno ocenijo vpliv stroškov dela, inflacije ... Zaradi tega so ocene pogosto prenizke, predvsem pri večjih in kompleksnejših sistemih pa verjetnost opustitve samo še narašča. V času projekta se te opustitve odkrijejo, ocenjene vrednosti pa narastejo.

Negotovost (angl. *Uncertainty*). Ocene temeljijo na nepopolnem oz. nezadostnem poznavanju lastnosti objekta ter vrednosti teh lastnosti. V najslabšem primeru niso poznane niti same lastnosti, bolj pogosto pa so lastnosti prepoznane, vendar ne znamo pravilno oceniti njihovih vrednosti. Negotovost v času projekta zaradi novih spoznanj upada.

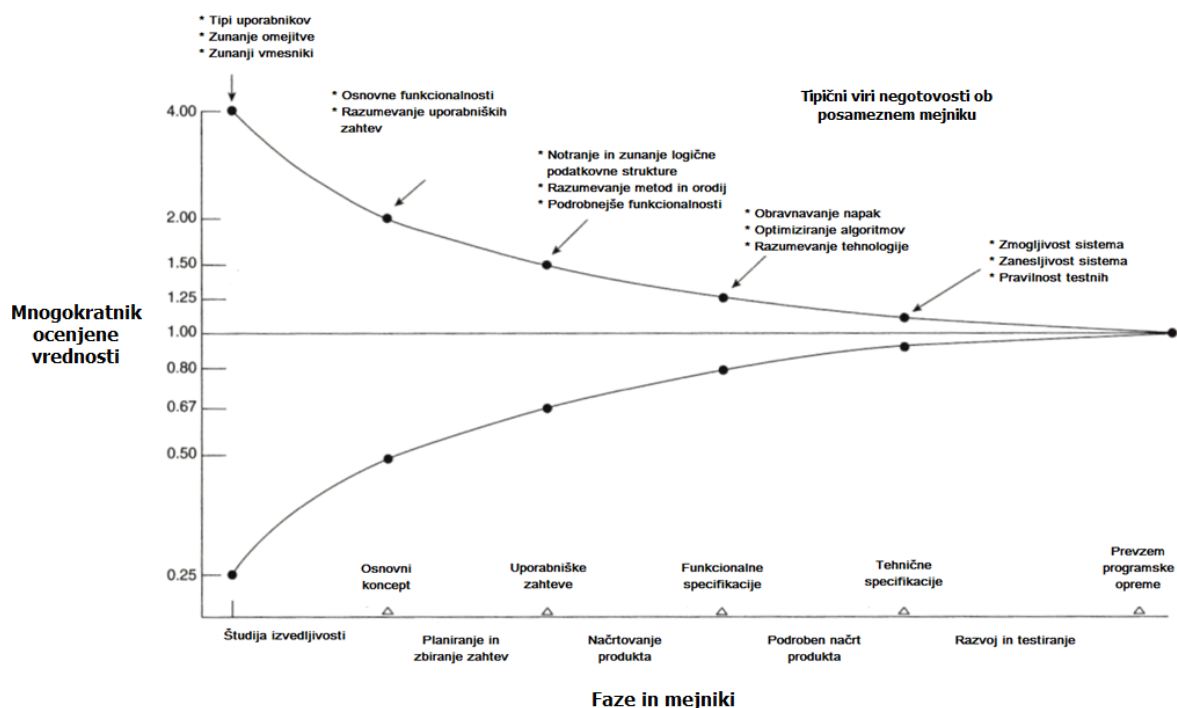
Spremembe (angl. *Change*). Predpostavke glede proizvoda, projekta ali procesa se lahko v času trajanja projekta spremenijo. Prav tako se lahko spremenijo tudi zunanji dejavniki. Torej se tudi identificirane in znane veličine spreminjajo, sprememb pa prav tako ni možno predvideti vnaprej. Število sprememb je sorazmerno s trajanjem projekta, spremembe, ki se uvajajo proti koncu projekta, pa so vedno dražje.

McConell (2006) pojasnjuje, da je proces razvoja PO proces postopnega izpopolnjevanja (Slika 8). V tem postopku je potrebno sprejemati številne odločitve, od katerih je odvisna stopnja negotovosti. Več odločitev kot je že sprejetih, manj negotovosti ostaja. Z opazovanjem tega procesa so raziskovalci prišli do ugotovitev, da je negotovost ocen odvisna od stopnje projekta ter da je glede na stopnjo projekta možno to negotovost ustrezno predvideti.

Na horizontalni osi so glavni mejniki oz. faze projekta, na vertikalni osi pa je stopnja napak, do katere pride v posamezni fazi projekta tudi pri izkušenih ocenjevalcih. V začetni fazi je tako lahko razlika med obema skrajnima vrednostma kar 16-kratna (+/- 4-krat).

Večje natančnosti v tej fazi ni mogoče doseči, tudi če bi ocenjevanju namenili več časa. Ker so mejniki projekta zgoščeni v zgodnjih fazah, se že po eni tretjini projekta negotovost zmanjša na +/- 1,25-krat. Potrebno se je zavedati, da je lijak negotovosti prisoten tudi v dobro vodenih projektih. V primeru neizkušenih ocenjevalcev ali slabo vodenega projekta pa se negotovost ne zmanjšuje tako hitro. Če se s pravilnim projektnim vodenjem ne odpravijo vse variabilnosti, je lahko negotovost vse do zaključka enaka ali pa se med samim projektom celo poveča. Pogosta napaka razvijalcev PO je, da podajajo zavezujoče ocene v prezgodnjih fazah projekta oz. že v začetnem stanju lijaka negotovosti (McConell, 2006).

Slika 8: Lijak negotovosti



Povzeto in prirejeno po R.D. Stutzke, *Estimating Software-Intensive Systems*, 2005, str. 12, slika 1.3.

McConell (2006) svetuje, da lahko negotovost do neke mere zmanjšamo z iterativnim modelom razvoja PO. V tem primeru imamo opravka z več mini lijaki negotovosti ter lahko hitreje pripravimo natančnejše ocene za nabor funkcionalnosti, ki bodo realizirane v posamezni iteraciji. Tak pristop pa ima pomanjkljivost, da v začetni fazi ne moremo oceniti celotnega obsega in trajanja projekta, ker takrat običajno še niso znane vse uporabniške zahteve. Veliko organizacij zato uporablja kombiniran pristop. Večina uporabniških zahtev se specificira že v začetku projekta, načrtovanje, razvoj, testiranje in izdajanje verzij pa se izvajajo v krajših iteracijah. Zaporedno se izvedejo začetne faze do definicije uporabniških vmesnikov, nato pa se preide na iterativni razvoj. Na ta način dokaj

hitro zmanjšamo negotovost na sprejemljiv nivo ter tako omogočimo dovolj natančno oceno trajanja in stroškov projekta. Dopuščamo pa tudi zadostno raven fleksibilnosti za manjše spremembe uporabniških zahtev, za katere je potrebno pri planiranju predvideti ustrezno rezervo.

Za okolja, v katerih se ni mogoče izogniti spremembam, so torej najprimernejši razvojni modeli s kratkimi iteracijami ter agilen pristop (Scrum, »*Extreme Programming*« ...). Tockey (2005) je z ekonomskimi izračuni dokazal, da so zaradi eksponentno naraščajoče cene sprememb med trajanjem projekta v takšnih primerih ustreznejši krajši razvojni cikli, saj fiksni stroški za obvladovanje posamezne iteracije upadajo mnogo počasneje. Zanimiva je tudi ugotovitev, da za projekte razvoja PO velja disekonomija obsega. Večji kot je obseg projekta, višji so stroški na posamezno enoto (McConell, 2006).

3.3 Tehnike ocenjevanja programske opreme

3.3.1 Izbor tehnik ocenjevanja glede na fazo razvoja programske opreme

McConell (2006) predlaga, da poizkusimo čim več parametrov, ki vplivajo na oceno prešteti ali izračunati. V zgodnjih fazah razvoja je možno šteti uporabniške zahteve, lastnosti /funkcionalnosti, primere uporabe, uporabniške zgodbe ... V nadaljevanju projekta je možno šteti tehnične zahteve, funkcijske točke, zahteve za spremembe, poročila, zaslonske maske, tabele v podatkovni bazi ... Proti koncu projekta pa je možno šteti na bolj podrobnem nivoju tudi npr. obseg že napisane kode, število javljenih napak, razrede ... Na osnovi zgodovinskih podatkov je mnogokrat iz prešteti veličin možno izračunati oceno potrebnega truda. Strokovno presojo naj bi se uporabljali samo takrat, kadar ni druge možnosti.

Najbolj znane so naslednje tehnike ocenjevanja (McConell, 2006):

- individualna strokovna presoja (angl. *Individual Expert Judgment*),
- ocenjevanje z dekompozicijo (angl. *Decomposition*),
- ocenjevanje z analogijo (primerjavo) (angl. *Estimation by Analogy*),
- ocenjevanje s pomočjo korelacije (angl. *Proxy Based Estimates*),
- skupinska strokovna presoja (angl. *Expert Judgment in Groups*),
- s pomočjo programskih orodij (angl. *Software Estimating Tools*).

3.3.2 Individualna strokovna presoja

Kljub drugačnim priporočilom so raziskave pokazale, da je prav individualna strokovna presoja najbolj pogost način ocenjevanja. Na nivoju posamezne naloge oz. kodiranja posamezne funkcionalnosti ali odprave konkretne napake lahko najbolj natančno oceno

podajo programerji sami. Potrebno pa se je zavedati, da nekdo, ki je strokovnjak na določenem tehničnem področju, ni nujno tudi strokovnjak za izvajanje ocenjevanj. Zato naj bi predvsem v začetnih fazah lijaka negotovosti pri ocenah sodelovali tudi strokovnjaki s področja ocenjevanja, najbolj izkušeni razvijalci ter analitiki. Pri ocenjevanju se je potrebno v čim večji meri izogibati intuiciji, kar lahko dosežemo z dekompozicijo celotnega problema na manjše naloge. V primeru bolj strukturiranih ocen se lahko le-te po natančnosti že primerjajo z ocenami, ki temeljijo na parametričnih modelih (McConell, 2006).

McConell (2006) priporoča, da naj na najnižjem nivoju posamezna naloga ne bi trajala več kot 2 dni. Prav tako je priporočljivo, da se ocena na nivoju posamezne naloge ne poda kot točna vrednost, temveč se oceni najboljši (angl. *Best Case*) in najslabši (angl. *Worst Case*), možen čas trajanja. Že samo s tem je ocenjevalec prisiljen k bolj poglobljenemu razmisleku, kar vodi k natančnejši oceni. Pričakovana (najverjetnejša vrednost) običajno ni ravno srednja vrednost. Ena od bolj poznanih tehnik, s katerimi izračunamo pričakovano vrednost, je PERT (angl. *Program Evaluation Review Technique*), kjer je za izračun pričakovane vrednosti potrebno podati še oceno najbolj verjetne vrednosti (angl. *Most Likely Case*). Pričakovano vrednost izračunamo po enačbi (1):

$$\text{Pričakovana vrednost} = \frac{(\text{Najboljša vred.} + (4 * \text{Najverjetnejša vred.}) + \text{Najslabša vred.})}{6} \quad (1)$$

Stutzke (v McConell, str. 2006) opozarja na predlog več strokovnjakov, da je pri razvoju PO zaradi praviloma preveč optimističnih ocen bolje uporabiti prirejeno enačbo (2):

$$\text{Pričakovana vrednost} = \frac{(\text{Najboljša vred.} + (3 * \text{Najverjetnejša vred.}) + (2 * \text{Najslabša vred.}))}{6} \quad (2)$$

3.3.3 Ocenjevanje z dekompozicijo

Razbijanje kompleksnih projektov na manjše, lažje obvladljive enote je eden od temeljnih pristopov pri ocenjevanju. Po izvedeni dekompoziciji se ločeno oceni vsaka elementarna naloga ter se nato vse ocene ponovno združijo v skupno oceno za celoten projekt. Nivo granulacije posamezne naloge je odvisen od stopnje projekta, temeljno orodje je strukturirana členitev dela (angl. *Work Breakdown Structure*) (McConnel, 2006).

3.3.4 Aditivna in multiplikativna analogija

Analogno ocenjevanje temelji na podatkih predhodno izvedenih projektov. Zato je pomembno, da po izvedbi vsakega projekta zberemo podatke, s katerimi postopno gradimo bazo znanja za ocenjevanje bodočih projektov. Aditivna metoda je nekoliko enostavnejša, saj ocenjena razmerja med preteklimi in novim projektom enostavno seštevamo, pri multiplikativni metodi pa upoštevamo tudi spremembe izhodiščne vrednosti, na katere so

vplivale predhodne prilagoditve. V določenih primerih lahko po mnenju W. Roetzheima (v Stutzke, 2005) uporabimo kar proporcionalno razmerje.

Ocene, pridobljene z analogijo, običajno ne zadoščajo za dovolj natančno oceno projekta, zato jih je smiselno izvajati v kombinaciji z ostalimi ocenjevalskimi tehnikami, npr. z uporabo parametričnih modelov ter metod Delphi ali PERT.

3.3.5 Ocenjevanje s pomočjo korelacije

Velikokrat je nemogoče podati direktno oceno veličine, ki nas zanima. V teh primerih si pomagamo z oceno veličine, ki jo je enostavneje oceniti ali izmeriti ter nato na osnovi znanih korelacij izračunamo vrednost spremenljivke, ki nas zanima. Na osnovi števila uporabniških zahtev npr. predvidimo število testnih primerov, na osnovi zahtevanih funkcionalnosti ocenimo število vrstic kode. Najbolj poznane in uporabljane so točke uporabniških zgodb (angl. *Story Points*).

3.3.6 Skupinska strokovna ocena (Delphi tehnika)

Pristop skupinskega odločanja najpogosteje uporabljamo v zgodnjih fazah projekta ter pri projektih z veliko neznankami. Skupinsko odločanje je lahko nestrukturirano ali strukturirano (McConell, 2006). Nestrukturiran je npr. skupen pregled (angl. *Group Review*), kjer vsak ocenjevalec pripravi individualno oceno, na prvem skupnem sestanku pa se nato sodelujoči uskladijo. Odločanje z glasovanjem ni dovoljeno, vsi sodelujoči se morajo strinjati z rezultatom, ki je v skrajnem primeru lahko tudi izračun srednje vrednosti. Najbolj razširjena tehnika strukturiranega odločanja je Delphi. Odločanje po tej tehniki je iterativen proces, ki sestoji iz več ponovitev. V vsakem krogu posamezen presojevalec anonimno in neodvisno oceni vrednost postavke. Nihče ne sme poznati ocen drugih, kar izniči možnost vplivanja ter političnih pritiskov. Na koncu vsake iteracije se udeleženci seznani z rezultati, eliminirajo se največja odstopanja ter se postopek ponovi, dokler se ne doseže soglasje. S pomočjo strukturirane obdelave podatkov ter povratnih informacij se lahko ocene običajno uskladijo v 3 do 4 iteracijah (Stutzke, 2005).

3.3.7 Uporaba orodij (Analogija z algoritemsko podporo)

Klasična ocena z analogijo temelji na podatkih zelo omejenega števila projektov, s katerimi razpolagajo v določeni organizaciji. Sodobna programska orodja za podporo ocenjevanja projektov razvoja PO uporabljajo podatkovne baze, v katerih so zbrani podatki več tisoč projektov ter se za njihovo interpretacijo uporabljajo namensko razviti algoritmi (angl. *Algorithmic Analogy Estimation*), ki temeljijo na znanju ekspertov s področja ocenjevanja ter posameznih strokovnih področij (Stutzke, 2005).

Vhodna točka za vsa komercialna orodja je obseg projekta, izražen na en ali več načinov (v logičnih stavkih, fizičnih vrsticah kode, funkcijskih točkah ...). Na izračun ostalih parametrov projekta vplivajo številni dejavniki, ki izvirajo iz ljudi, tehnologije, orodij ter okolja.

Obstajajo trije tipi parametričnih modelov (Stutzke, 2005):

- teoretični modeli obravnavajo proces razvoja PO preko matematičnih formul;
- statistični modeli uporabljajo linearne in nelinearne funkcije nad zgodovinskimi podatki;
- kompozitni modeli uporabljajo kombinacijo obeh predhodnih metod.

3.4 Ocenjevanje obsega programske opreme

3.4.1 Karakteristike ter možni pristopi k ocenjevanju obsega

Pri kompleksnih IS je končnim uporabnikom vidno samo okoli 20 % funkcionalnosti IS, ostale podporne komponente pa ostajajo skrite. Pri ocenjevanju pa je potrebno upoštevati vse komponente. Da je to možno, je potrebno razumeti namen produkta, način njegove uporabe ter celotno arhitekturo sistema (Stutzke, 2005).

Živkovič (2004) utemeljuje, da je pri razvoju PO ključno vprašanje obseg produkta, saj lahko iz njega določimo potreben trud, na podlagi truda pa oblikujemo terminski plan in izračunamo stroške. Loči dva temeljna pristopa za ocenjevanje obsega (ocenjevanje z analogijo in ocenjevanje z uporabo modelov) ter dva tipa metrik: neodvisne od tehnologije (funkcijske točke) ter odvisne od tehnologije rešitve (vrstice kode, število razredov).

McConell (2006) našteva podrobnejše pristope za merjenja obsega:

- lastnosti (angl. *Features*),
- uporabniške zgodbe (angl. *User Stories*),
- točke uporabniških zgodb (angl. *Story Points*),
- uporabniške zahteve (angl. *Requirements*),
- primeri uporabe (angl. *Use Cases*),
- funkcijske točke (angl. *Function Points*),
- Web strani (angl. *Web Pages*),
- komponente grafičnega vmesnika (angl. *GUI Components*),
- podatkovne tabele (angl. *Databases Tables*),
- integracije z drugimi IS (angl. *Interface Definitions*),
- razredi (angl. *Classes*),
- programske funkcije, rutine (angl. *Functions, Subroutines*),
- vrstice kode (angl. *Lines of Code*).

Kot najmanj primerno navaja število vrstic programske kode, kot sprejemljivo alternativo pa predlaga funkcijske točke, ki se na osnovi uporabniških zahtev lahko določijo v zgodnejših fazah projekta ter nam lahko posredno služijo tudi za določanje števila vrstic kode.

Karakteristike dobre meritve velikosti PO so (Stutzke, 2005):

- je v korelaciji s potrebnim trdom za razvoj,
- je neodvisna od tehnologije in proizvodnega procesa,
- se lahko oceni zgodaj v projektu,
- jo je možno enostavno oceniti (poceni in hitro).

V času projekta se razpoložljive ocenjevalske tehnike spreminjajo, saj s podrobnejšo dekompozicijo narašča nabor razpoložljivih podatkov.

Pri ocenjevanju obsega je potrebno upoštevati tudi rast velikosti ter nestabilnost zahtev. Jones (v Stutzke, 2005) navaja, da je pričakovana rast velikosti približno 1 % na mesec. Zaradi tega je potrebno zahteve v določeni fazi projekta zamrzniti, kar se običajno zgodi približno na polovici projekta. Na vložen trud ima velik vpliv tudi nestabilnost zahtev, ki povzročajo ponovno proučevanje in korigiranje že izdelane kode. Rast in nestabilnost sta največja v začetnih fazah projekta. Faktor rasti in faktor nestabilnosti sta odvisna od naročnika, stabilnosti zunanega okolja, značilnosti projekta, izkušenj razvijalcev, sposobnosti razvojnega procesa ter tipa IS.

Najpogosteje uporabljeni pristopi za merjenje obsega PO so:

- funkcijske točke,
- točke primerov uporabe,
- aplikacijske točke.

3.4.2 Funkcijske točke

Analiza funkcijskih točk spada med najbolj razširjene ter zanesljive metode za oceno obsega programskih projektov. Najprimernejša je za ocenjevanje poslovnih IS, kjer je poudarek na vnosu in obdelavi podatkov in niso potrebni kompleksni izračuni in algoritmi (Živkovič, 2004). Metoda funkcijskih točk ni najbolj uporabna v agilnih projektih, ker zaradi visokih stroškov in potrebnega časa ni skladna z osnovno filozofijo agilnega razvoja (Jones, 2008).

Formalno je za to metodologijo odgovorna skupina »*International Function Point User Group*« (IFPUG). Temelji na delitvi funkcionalnosti v dve skupini (podatkovna in transakcijska), ter znotraj teh na pet elementov (McConell, 2006):

- zunanji vhodi (angl. *External Inputs*): zaslone in forme, preko katerih končni uporabnik vnaša, spreminja ali dodaja podatke;
- zunanji izhodi (angl. *External Outputs*): zaslone, grafi in poročila, ki jih generira program (običajno zahteva procesiranje podatkov, kombiniranje več podatkov ...);
- zunanje poizvedbe (angl. *External Queries*): kombinacija vhodno /izhodnih zahtev, kjer vhodna poizvedba takoj pridobi enostaven odgovor (podatke se bere direktno iz baze);
- interne logične zbirke (angl. *Internal Logical Files*): glavne logične tabele podatkov, za vzdrževanje katerih je odgovoren uporabnik;
- zunanje vmesniške zbirke (angl. *External Interface Files*): podatki , ki jih generirajo drugi programi oz. se pridobivajo iz zunanjih podatkovnih zbirk, za katere uporabnik ni odgovoren.

Začetki funkcijskih točk segajo v leto 1975, trenutno pa obstaja preko 20 različic, ki bodisi dodajajo ali odvzemajo vhodne parametre, povečujejo obseg prilagoditev na osnovi kompleksnosti, dodajajo nove faktorje ali pa poenostavljajo izračune. Najbolj razširjena je uporaba metod IPFUG, COSMIC, MARK II ter NESMA (Jones, 2008).

Postopek izračuna funkcijskih točk ima sedem korakov (Živkovič, 2004):

- Določitev področja uporabe: opredelitev ali gre za razvoj, prenovo ali vzdrževanje.
- Določitev mej programskega sistema: stališče uporabnikov, kaj naj bo v sistemu zajeto in kaj ne.
- Določitev podatkovnih funkcij in njihove kompleksnosti: zajema vse interne logične zbirke in zunanje vmesniške zbirke. Kompleksnost se določi na osnovi tabele, kjer se upošteva število enostavnih podatkovnih elementov in število sestavljenih podatkovnih elementov.
- Določitev vseh transakcijskih funkcij in njihove kompleksnosti: pregled vseh zunanjih vhodov, zunanjih izhodov ter zunanjih poizvedb. Kompleksnost se določi na osnovi tabel, kjer se upošteva število podatkovnih funkcij in število podatkov, na katere se sklicujemo.
- Izračun neporavnanih funkcijskih točk: za vsako izmed petih funkcij oz. domen ovrednotimo nivoje (nizek, srednji, visok). Vsakemu nivoju kompleksnosti priredimo uteži, s katerimi pomnožimo število posameznih atributov in nato seštejemo zmnožke. Tako dobimo obseg, izražen v neporavnanih funkcijskih točkah (angl. *Unadjusted Function Points*).

- Izračun števila funkcijskih točk: način končnega izračuna funkcijskih točk je odvisen od področja uporabe, ki smo ga določili pri prvem koraku.

3.4.3 Točke primerov uporabe

S točkami primerov uporabe lahko ocenimo velikost in potreben trud že v fazi uporabniških zahtev, ne vsebujejo pa opisa nefunkcionalnih zahtev. Primeri uporabe (angl. *Use Case*) so ena od komponent standardne notacije UML.

3.4.4 Aplikacijske točke

Aplikacijske točke temeljijo na štetju zaslonov in poročil. Potreben trud za določen tip funkcionalnosti je tako bolj enakomeren, kot če bi temeljil zgolj na opisu uporabniških zahtev. Pristop je primeren predvsem, kadar se uporabljajo ICASE (angl. *Integrated Computer Aided Software Engineering*) razvojna okolja. Ta pristop je z nekaterimi dopolnitvami uporabil B. Boehm v COCOMO II. Metoda predvideva, da podatkovni vir že obstaja, kar drži v primeru »legacy« (obstoječih) sistemov. Ne upošteva namreč truda, ki bi ga bilo potrebno vložiti v definiranje ali spreminjanje podatkovnih struktur. Izkušveno se ocenjuje, da ta segment lahko predstavlja tudi 1/3 celotnega truda. S to metodo torej znatno podcenimo potreben trud, kadar je potrebno definirati tudi podatkovno bazo. Metoda se lahko uporabi v dokaj zgodnjih fazah, je pa odvisna od izbrane tehnologije.

3.4.5 Objektno orientirane metode za merjenje obsega

Analizo funkcijskih točk (angl. *Function Point Analysis*, v nadaljevanju FPA) je vpeljal Albrecht leta 1979. Glavne lastnosti so neodvisnost od uporabljene tehnologije in razširjenost, slabost pa diskretna zasnova funkcije, ki preslika lastnosti IS v obseg in slaba zasnova faktorjev tehnične kompleksnosti. Zato so se kasneje uveljavile še mnoge druge metode, ki vsaka na svoj način poskušajo preseči prvotne omejitve. Pri vseh so potrebna dodatne preslikave v primeru uporabe objektnih konceptov. Preslikave temeljijo na različnih diagramih (npr. na diagramih primerov uporabe, razrednih diagramih, diagramih zaporedja ...). Temeljna pomanjkljivost sodobnih metod je neprimerljivost z zgodovinskimi podatki ter možnost uporabe šele v poznih fazah razvojnega cikla, zaradi česar ni uporabna pri projektnem vodenju z namenom napovedovanja truda in trajanja projekta.

3.5 Ocenjevanje truda

McConell (2006) meni, da ima največji vpliv na trud velikost oz. obseg PO, drugi najpomembnejši faktor pa je produktivnost v posamezni organizaciji.

Hill (2001) med dejavniki, ki najbolj vplivajo na produktivnost, izpostavlja sposobnost skupinskega dela, izkušnje s tehnologijo, nivo tehnične inovativnosti, lastnosti produkta, zahtevano kakovost, delovno okolje, stabilnost zahtev, sodelovanje pogodbenih partnerjev ter vključenost naročnikov. Pomemben vpliv imajo strojna platforma, programski jezik ter uporabljena razvojna orodja. Prav tako je dokazano, da so učinkovitejše manjše ekipe (produktivnost prične upadati, če vključujejo več kot 5 članov).

Zanimiv je tudi vpliv pritiska, ki ga za doseganje rokov na razvijalce izvaja management. Neracionalni roki so eden najbolj destruktivnih dejavnikov, ki poleg demoraliziranja ekipe vplivajo tudi na fluktuacijo običajno najbolj sposobnih razvijalcev. Do neke stopnje pritisk pozitivno vpliva na produktivnost in motivacijo, nato pa pride do ostrega preobrata v negativno smer (Jones, 1998).

Stutzke (2005) opozarja, da mora za korektne izračune truda vsaka organizacija sama izmeriti oz. določiti produktivnost, ki temelji na meritvah njenega procesa. Izpostavlja tudi, da funkcijske točke izmerijo samo velikost končnega produkta, ne omogočajo pa ocene truda, ki je potreben za odzemanja in spreminjanje funkcionalnosti, ki so posledica sprememb uporabniških zahtev med izvajanjem projekta. Povečanje velikosti in potreben trud tako nista povsem sorazmerna, saj se predvsem zaradi nestabilnosti zahtev (spreminjanje, umikanje) v končni verziji ne odražajo vse že razvite funkcionalnosti. Na povečanje obsega namreč vplivajo samo dodane zahteve, minus odvzete, na potreben trud oz. inženirsko delo pa tako dodane, odvzete, kot tudi spremenjene.

McConnell (2006) ter Hill (2001) ločita naslednje možne pristope za izračun truda iz obsega:

- produktivnost ekipe lahko določimo s primerjavo s predhodnimi projekti (predpogoj je enak življenjski cikel, metodologija, orodja, kadri ...);
- z uporabo parametričnih komercialnih orodij (ki razpolagajo z zgodovinskimi podatki podobnih projektov ostalih organizacij);
- s pomočjo javno objavljenih podatkov potrebnega truda (povprečje v panogi) za posamezen tip aplikacije,
- s pomočjo izkustvenih enačb, objavljenih v repozitoriju »*Software Benchmarking Standards Group*« (ISBSG).

Na osnovi proučitve več sto projektov so v ISBSG pripravili nabor empiričnih formul, ki na osnovi znanih parametrov o obsegu projekta (izraženem v funkcijskih točkah), tipu razvojnega okolja (programskem jeziku) ter največji velikosti razvojne ekipe omogočajo izračun truda iz ocenjenega obsega.

Za specifične primere kombinacij osnovnih parametrov obstaja nabor enačb z različnimi utežmi, vsem pa je izhodišče splošna enačba (3):

$$\text{ČlovekMesecev} = 0,512 * \text{FunkcijskeTočke}^{0,392} * \text{NajvečjaVelikostEkipe}^{0,791} \quad (3)$$

3.6 Ocenjevanje časa oz. trajanja

Pravilna ocena časa, potrebnega za zaključek posameznega projekta ali pa zgolj posamezne funkcionalnosti je pomembna iz več razlogov. Rok za zahtevan zaključek lahko izvira iz zakonodajnih ali regulatornih zahtev, lahko je pogojen z zahtevami ali pričakovanji trga, internimi poslovnimi zahtevami ali zahtevami lastnikov. Po vzpostavitvi urejenega procesa ocenjevanja obsega in truda je izračun časa povsem matematično določljiv. McConell (2006) opisuje naslednje pristope:

Osnovna formula za izračun trajanja projekta.

Že v zelo zgodnjih fazah projekta lahko potreben čas za zaključek projekta izračunamo po izkustveni enačbi (4):

$$\text{Čas} = 3,0 * \text{Trud}^{1/3} \quad (4)$$

Vodilni faktor se glede na specifiko posamezne organizacije lahko giblje od 2 do 4, splošno pravilo pa je, da je koledarski čas funkcija tretjega korena potrebnega truda. Formula izvira že iz leta 1981, a jo novejša raziskava Boehma in Stutzkeja iz let 2000 in 2005 še vedno potrjujejo. Dejstvo, da je trajanje projekta v korelaciji s tretjim korenem truda, pomembno vpliva tudi na lijak negotovosti. Obseg, trud ter stroški se gibljejo v intervalu od - 0,25-krat do + 4-krat, čas trajanja pa samo od - 0,6-krat do + 1,6-krat.

Neodvisno od predhodnih ugotovitev so na osnovi raziskav repozitorija ISBSG do podobne empirične enačbe (5) za odvisnost med ocenjenim trudom in časom trajanja prišli tudi Oligny, Borquw in Abran (v Hill, 2001).

$$\text{Trajanje}_{(v\ koledarskih\ mesecih)} = 0,662 * \text{Trud}^{0,328} \quad (5)$$

Neformalna primerjava s predhodnimi projekti.

Trajanje projekta je možno oceniti tudi na osnovi razmerja trajanje/trud predhodnih projektov. Roetzheim (v McConell, 2006) za manjše projekte, ki uporabljajo hiter razvojni cikel, predlaga izračun po enačbi (6):

$$\text{Čas}_{nov} = \text{Čas}_{star} \left(\frac{\text{Trud}_{nov}}{\text{Trud}_{star}} \right)^{0,5} \quad (6)$$

Pri srednjih in velikih projektih pa je ustrežnejša uporaba enačbe (7):

$$\check{c}as_{nov} = \check{c}as_{star} \left(\frac{Trud_{nov}}{Trud_{star}} \right)^{1/3} \quad (7)$$

Hiter izračun na osnovi funkcijskih točk.

Na osnovi empiričnih izkušenj več tisoč projektov je Jones (v McConell, 2006) razvil enačbo (8), ki sicer ne more nadomestiti ostalih metod, lahko pa z njeno pomočjo že v zelo zgodnjih fazah projekta okvirno ocenimo potreben čas za zaključek.

$$Trajanje = FunkcijskeTočke^{Faktor} \quad (8)$$

Faktor uteži je odvisen od tipa IS ter uporabljene tehnologije razvoja.

Podoben pristop direktnega izračuna trajanja razvoja direktno iz funkcijskih točk opisuje tudi Hill (2001), ki kot enačbo (9) za programski jezik 4. generacije podaja:

$$Trajanje = 0,662 * FunkcijskeTočke^{0,404} \quad (9)$$

Pristop s pomočjo orodij.

S pomočjo vnosa zgodovinskih podatkov konkretne organizacije ter podatkovne baze projektov celotne panoge lahko znatno izboljšamo natančnost ocene.

Skrajševanje časa razvoja.

Pogosto se izračunan čas ne sklada s poslovnimi zahtevami in je potrebno iskati alternativne možnosti. Čas razvoja lahko poljubno zmanjšamo na račun okrnjenih funkcionalnosti ali z dodajanjem razvijalcev, a le do odločene mere. Z dodajanjem novih razvijalcev se trajanje ne krajša linearno oziroma se s krajšanjem predvidenega časa razvoja trud nesorazmerno večja. Da se čas razvoja skrajša za 10 %, je po eni od raziskav (Jensen v McConell, 2006) potrebno kar 30 % več vloženega truda. Trud narašča, ker je potrebno več koordinacije in managementa, zaradi več komunikacijskih poti je večja možnost za nesporazume, večje ekipe naredijo več napak. Ker se mora delo izvajati vzporedno, nedokončane ali funkcionalnosti z napako povzročajo več naknadnih popravkom drugim ekipam. Po mnenju mnogih raziskovalcev časa razvoja na noben način ni možno skrajšati za več kot 25 % glede na korektno izvedeno prvotno oceno (McConell, 2006).

3.7 Ocenjevanje stroškov

V primeru ročnih ocen stroškov veliko managerjev upošteva samo delo razvijalcev (programerjev), zanemari ali podceni pa vpliv vseh ostalih aktivnosti, kar predstavlja precejšen problem, saj je pri velikih sistemih največ stroškov povezanih z odkrivanjem in odpravljanjem napak. Pomembno težo imajo stroški, povezani s pripravo razne dokumentacije, specifikacij, uporabniških navodil, projektnim vodenjem. Veliki pa so lahko tudi potni stroški, stroški nastanitev in dnevnic (Jones, 1998).

3.8 Vzpostavitev procesa ocenjevanja

Postopek ocenjevanja mora biti natančno opredeljen že vnaprej, predvsem zaradi tega, da se izognemo prirejanju rezultatov. Standardiziran postopek ocenjevanja je dobro definiran proces, prilagojen posamezni organizaciji in predstavlja smernice za izvajanje ocenjevanja na nivoju posameznega projekta (McConell, 2006).

Dobro definiran postopek upošteva naslednje smernice (McConell, 2006):

- uporaba štetja in izračunavanja vedno, kadar je možno, presoja samo, kadar je nujno;
- uporaba več pristopov/tehnik ter primerjava rezultatov;
- priprava in vnaprejšnja objava plana ponovnih ocenjevanj;
- definiranje, na kak način se bo izvajalo ocenjevanje v posamezni fazi projekta;
- opis natančnosti ocene v posamezni fazi;
- definirati, kdaj je ocena dovolj natančna za pripravo budgeta (proračuna) projekta;
- definirati, kdaj je ocena dovolj natančna in stabilna, da je možno sprejeti zavezo;
- shranjevanje rezultatov ter stalno izboljševanje postopka.

Jones (1998) predlaga naslednji postopek izvedbe ocenjevanj:

- analiza uporabniških zahtev,
- pričetek ocenjevanja obsega,
- identifikacija aktivnosti, ki jih je potrebno izvesti,
- ocena vpliva napak v vseh razvojnih fazah,
- ocena potrebnih razvijalcev,
- prilagoditev predvidevanj dejanskim zmožnostim in izkušnjam,
- ocena truda in časa razvoja,
- ocena stroškov razvoja,
- ocena stroškov vzdrževanja in nadgrajevanja.

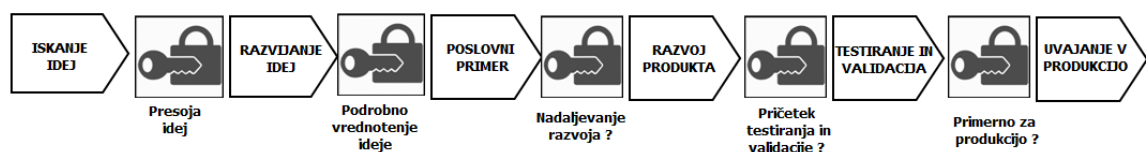
Stutzke (2005) loči dva možna pristopa k ocenjevanju, pogosto pa se v različnih fazah projekta uporabljata oba. S pristopom od zgoraj navzdol najprej pripravimo okvirno oceno obsega za celoten projekt. Nato se na osnovi lastnosti produkta ter poznane produktivnosti izračuna skupen trud in čas za celoten projekt, tega pa nato s pomočjo izkustvenih tabel razdelimo na posamezne aktivnosti. Pri pristopu od spodaj navzgor pa s pomočjo strukturirane členitve dela najprej prepoznamo posamezne naloge, ki jih nato s pomočjo ustreznih tehnik in metod samostojno ocenimo ter na koncu seštejemo na nivo projekta. Pri tem je pomembno, da s pomočjo povratnih zank v procesu ocenjevanja postopno izboljšujemo natančnost ocen.

Model procesa ocenjevanja naj bo formalen dokument, ki opisuje proces in je sestavljen iz:

- politik: visoko nivojska opredelitev ciljev organizacije ter področij odgovornosti;
- standardov: opisi artefaktov;
- postopkov: seznam korakov, s katerimi se izdelata artefakt, lahko vključuje opis tehnik, metod, pripomočkov;
- izvajalcev: seznam;
- vlog: odgovornost posameznikov za konkretne aktivnosti;
- pripomočkov: vnaprej pripravljenih predlog dokumentov, orodij ...

Razvoj PO je običajno sestavni del razvoja nekega produkta ali storitve. V uspešnih organizacijah tak razvoj poteka po formalnem procesu (ti. »Stage-Gate« proces), ki ga tvori zaporedje več faz in kontrolnih točk oz. vrat (Slika 9). Skladno z lijakom negotovosti so tudi ocene ob posameznih kontrolnih točkah oz. vratih po procesu »stage gate« določene z različno natančnostjo. Pri razvoju IS se soočimo s problemom, da je 85-90 % dela zajeto v fazi razvoja.

Slika 9: Razvojni cikel novega produkta po metodologiji "Stage Gate"



Povzeto in prirejeno po Cooper & Edget, Product development for the service sector, 1999, str. 78, slika 4.2.

3.9 Ocenjevanje truda in časa po aktivnostih

Ne glede na to, ali se projekt izvaja zaporedno ali iterativno, je potrebno oceniti trud, ki je potreben za izvedbo posamezne aktivnosti (zahteve, načrtovanje, izgradnja, test, management). McConell (2006) opozarja, da nekateri popularni modeli za pripravo ocen zaradi zelo različnih pristopov v praksi sploh ne zajemajo izračunov truda za zajem in

specifikacijo uporabniških zahtev ter za management projekta (npr. COCOMO II in Putnamov model). Poleg obsega projekta na potreben trud ter razporeditev po aktivnostih znatno vpliva tudi tip programske opreme (za poslovne namene, nadzorna ali vgrajena, podporna, uporabniška, sistemska ...).

Pomagamo si lahko z izkustvenimi tabelami (Tabela 2), ki podajajo približna razmerja podpornih aktivnosti glede na ocenjene razvojne aktivnosti (McConell, 2006).

Tabela 2: Razporeditev truda po tipih aktivnosti

Obseg [KLOC]	Razvojne aktivnosti				Razvojne in podporne aktivnosti					
	Načrtovanje arhitekture [%]	Izdelava (programiranje) [%]	Testiranje [%]	Skupaj razvoj [%]	Uporabniš ke zahteve [%]	Načrtovanje arhitekture [%]	Izdelava (programiranje) [%]	Testiranje [%]	Management [%]	Skupaj vse aktivnosti [%]
1	11	70	19	100	4	10	61	16	9	100
25	16	57	27	100	4	14	49	23	10	100
125	18	53	29	100	7	15	44	23	11	100
500	19	44	37	100	8	15	35	29	13	100

Povzeto in prirejeno po S. McConell, Software Estimation, 2006, str. 37-38, tabeli 21.1 in 21.4.

Jones (1998) uporablja nekoliko drugačno razporeditev po posameznih aktivnostih, ki jih deli na uporabniške zahteve (10 %), analizo in načrtovanje (20 %), kodiranje (30 %), testiranje (35 %) ter namestitve in uvajanja (5 %).

Ocenjevanje potrebnega časa za posamezne aktivnosti

Pri pretvorbi izračunanega truda in časa v planske postavke je potrebno upoštevati več dejavnikov, ki vplivajo na dejansko razpoložljivost virov (prazniki, bolniške, izobraževanja, delo na več projektih, sodelovanje na predstavitev, podpora uporabnikom, vzdrževanje obstoječih sistemov, sestanki ...). Samo v zelo razvojno naravnanih podjetjih je možno planirati 40-50 ur na teden (visoka motiviranost, majhne ekipe, plačane nadure ...). V velikih, strukturiranih organizacijah s pogostimi sestanki, kjer delo traja samo 40 ur tedensko, pa je efektivnega dela samo 20-30 ur tedensko, pogosto je sočasno delo na dveh ali več projektih (McConell, 2006). Zaradi neplaniranih aktivnosti torej ni nikoli na voljo 8 ur na delovni dan za izvajanje razvoja. Stutzke (2005) ugotavlja, da je tudi ob delu na samo enem projektu potrebno predvideti zgolj 140 delovnih ur na mesec. Nekatere organizacije sicer primarno računajo s 156 urami/mesec, a nato pri planiranju odzamejo rezervo (angl. *Buffer*), ki znaša od 10 do 30 %. Jones (1998) kot povprečen čas, ki je namenjen projektu, upošteva 132 ur/mesec, v modelu COCOMO II pa je upoštevanih 152 ur/mesec. Razporeditev časa med posamezne aktivnosti na osnovi izkustvenih tabel je prikazana v Tabeli 3.

Stutzke (2005) opozarja na neenotno terminologijo v literaturi in predlaga bolj dosledno poimenovanje ter razločevanje med posameznimi izrazi:

- čas razvoja (angl. *Development Time*, TDEV): čas potreben do priprave verzije, meri se v koledarskih dneh;
- trajanje razvoja (angl. *Duration Time*, TDUR): čas, ko se je opravljalo delo, meri se v delovnih dneh.

Tabela 3: Poraba časa po posameznih aktivnostih razvoja

Obseg [KLOC]	Aktivnost			Dodatno za aktivnost
	Načrtovanje arhitekture [%]	Izdelava (programiranje) [%]	Testiranje [%]	Uporabniške zahteve [%]
1	15-20	50-65	15-20	10-16
25	15-30	50-60	20-25	12-22
125	20-35	45-55	20-30	13-22
500	20-40	40-55	20-35	24-30

Povzeto in prirejeno po S. McConnell, Software Estimation, 2006, str. 239, tabeli 21.8 in 21.9.

V tuji literaturi se za oba pojma uporablja izraz »*Schedule*«, ki se v slovenščino pogosto prevaja kot terminski plan ali časovnica, čeprav je po izvornem pomenu bližje trajanju razvoja kot času v koledarskih dneh.

3.10 Ocenjevanje stroškov v fazi vzdrževanja

Na temelju opazovanj številnih IS je bilo ugotovljeno, da delež vzdrževalnih aktivnosti v celotnem življenjskem ciklu poslovnih aplikacij dosega kar 70 % (Stutzke, 2005).

Jones (1998) ugotavlja, da je ocenjevanje stroškov vzdrževanja še bolj kompleksno kot ocenjevanje razvoja. Na stroške vzdrževanja pomembno vpliva število uporabnikov sistema ter število napak v produktu v času pričetka produkcijske uporabe. Ocenjevanje stroškov nadgradenj je primarno odvisno od števila in obsega zahtev za dopolnitve funkcionalnosti. Običajno je takoj po prehodu v produkcijo več nadgradenj, nato se do naslednje večje verzije zahteve umirijo. Ocena stroškov vzdrževanja se običajno pripravlja za 3 do 5 let po uvedbi, izjemoma lahko tudi za 10 let, kjer pa je prisotnih že precej neznank.

Za oblikovanje modela za ocenjevanje vzdrževalnih aktivnosti je potrebno dobro poznati procese, ki jih specifična PO podpira. Tako razvoj kot vzdrževanja imata sicer iste cikle (analizo, načrtovanje, kodiranje, integracijo, testiranje in nameščanje), a se aktivnosti v njih precej razlikujejo. Velikost in kompleksnost IS s časom naraščata, kar zmanjšuje produktivnost razvijalcev (Stutzke, 2005).

Pri kompleksnih sistemih obstaja veliko tveganje, da se ob namestitvah popravkov pojavijo stranski efekti. Brooks (v Stutzke, 2005) ocenjuje, da je kar 20-50 % verjetnost, da ob odpravi napak injiciramo nove napake. Zaradi tega je potrebno skrbno načrtovati, kakšne spremembe v produkcijo uvajamo s posameznim tipom namestitve (Tabela 4).

Tabela 4: Matrika tipov sprememb in izdaj (verzij)

Tip spremembe	Tip verzije		
	Večja (angl. <i>Major</i>)	Manjša (angl. <i>Minor</i>)	Popravek (angl. <i>Patch</i>)
Nadgradnja (angl. " <i>Enhance</i> ")	DA		
Prilagoditev (angl. " <i>Adapt</i> ")	DA	Pogojno	
Izboljšava (angl. " <i>Perfect</i> ")	DA	Pogojno	
Manjši popravek (angl. " <i>Correct - minor</i> ")	Pogojno	DA	
Kritični popravek (angl. " <i>Correct - serious</i> ")			DA
Številčenje verzije	1.0	1.1	1.1.1

Povzeto in prirejeno po R.D. Stutzke, Estimating Software-Intensive Systems, 2005, str. 131.

Za mnoge starejše IS je bilo značilno, da je vzdrževanje izvajal isti razvijalec ali organizacija, ki ga je razvila. Podobno situacijo srečujemo še danes, čeprav je tak pristop problematičen z več vidikov. Vsak IS bi moral biti že načrtovan in razvit tako, da lahko vzdrževanje prevzame nekdo tretji. Če je IS tako kompleksen, da ga lahko vzdržuje le razvijalec, je to opozorilo, da je arhitekturno in izvedbeno z njim nekaj narobe. Narava dela v fazi vzdrževanja se razlikuje od razvoja, zato ni stroškovno učinkovito, če to delo opravlja ista oseba. Prav tako je frustrirajoče in negativno vpliva na karierni razvoj, če mora razvijalec več deset let skrbeti za isti program. Dodatno tveganje predstavlja tudi odvisnost od ključnega razvijalca, ki lahko zboli, umre, da odpoved ... (Jones, 1998).

4 AGILNI RAZVOJNI MODELI

4.1 Vpliv metodologije razvoja programske opreme na uspešnost projektov

Po zadnjem poročilu Standish Group (Hastie & Wojewoda, 2015) so agilne metodologije razvoja PO po deležu uspešno zaključenih projektov v znatni prednosti pred klasično »slapovno« metodologijo ne glede na velikost projekta (Tabela 5).

Čeprav imajo agilne metodologije daljšo zgodovino, so v strokovni javnosti postale bolj znane šele po letu 2001, ko so vodilni strokovnjaki s tega področja ustanovili zvezo »Agile Alliance« ter objavili temeljna načela agilnih metodologij (Beck, 1999):

- posamezniki in njihova komunikacija so pomembnejši kot procesi in orodja,
- delujoča programska oprema je pomembnejša kot popolna dokumentacija,
- sodelovanje uporabnika je pomembnejše kot pogajanje na osnovi pogodb,
- upoštevanje sprememb je pomembnejše od sledenja načrtu.

Tabela 5: Uspešnost projektov glede na izbrano metodologijo razvoja

Velikost projekta	Model	Uspešni [%]	Delno uspešni [%]	Neuspešni [%]
Vse velikosti	Agilen	39	52	9
	Slapovni	11	60	29

Povzeto in prirejeno po Hastie & Wojewoda, Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch, 2015, slika 3.

Poznanih je več različnih agilnih metodologij:

- ekstremno programiranja (angl. *Extreme Programming*),
- metoda DSDM (angl. *Dynamic Systems Development Method*),
- metoda FDD (angl. *Feature Driven Development*),
- metoda ASD (angl. *Adaptive Software Development*),
- RUP (angl. *Rational Unified Process*),
- Crystal Clear,
- Scrum,
- Kanban.

Metodologije so sicer le redko uvedene povsem skladno s teoretičnimi navodili. Scrum kot najbolj razširjeno metodologijo tako v celoti dosledno izvaja le 33 % uporabnikov, vseh 13 praks Ekstremnega programiranja pa samo 6 % (Kurapati, Manyam & Petersen, 2012) .

4.2 Metodologija Scrum

Scrum temelji na preglednosti, nadzoru in prilagoditvah ter uporablja iterativni in inkrementalni pristop (Schwaber, 2004). Temeljne značilnosti agilnega razvoja so velika povezanost članov ekipe, kratke iteracije z delujočim izdelkom, osredotočenost na poslovne prioritete ter poudarek na izboljševanju lastnega procesa dela (Cohn, 2006).

Vloge sodelujočih pri razvoju po metodologiji Scrum so razdeljene v skupini »prašičev« (angl. *Pigs*) ter kokoši (angl. *Chickens*). Na spodnjem seznamu prve tri vloge predstavljajo »prašiče«, ki so zavezani oz. odgovorni za doseganje rezultatov, ostale tri vloge pa so zgolj vključene (»kokoši«):

- skrbnik metodologije (angl. *Scrum Master*),
- predstavnik naročnika/lastnik izdelka (angl. *Product Owner*),
- razvojna ekipa (angl. *Team*).
- uporabniki (angl. *Users*),
- deležniki (angl. *Stakeholders*),
- vodstvo (angl. *Management*).

Scrum proces tvorijo aktivnosti:

- sestanek za načrtovanje sprinta (angl. *Sprint Planning Meeting*),
- sprint (tek, iteracija) (angl. *Sprint*),
- dnevni Scrum (angl. *Daily Scrum Meeting*),
- revizija sprinta (angl. *Sprint Review Meeting*),
- retrospektiva sprinta (angl. *Sprint Retrospective Meeting*).

Artefakti procesa pa so:

- seznam zahtev izdelka (angl. *Product Backlog*),
- seznam zahtev sprinta (angl. *Sprint Backlog*),
- graf preostalega časa (angl. *Burndown Chart*).

Načrtovanje sprinta se izvede na začetku vsakega sprinta in običajno traja cel delovni dan. Udeležijo se ga lastnik produkta, skrbnik metodologije ter celotna razvojna ekipa. Prva polovica dneva je namenjena predstavitvi in razlagi funkcionalnosti, ki imajo najvišjo prioriteto za realizacijo. V drugi polovici dneva razvojna ekipa podrobneje prouči zahteve in oceni, za katere zahteve lahko zagotovi realizacijo. Po začetku sprinta lahko samo ekipa doda novo delo. Nihče od vodilnih, niti lastnik produkta ne sme dodajati novih nalog, ki so bile ob planiranju pozabljene. Če ekipa napreduje hitreje, kot je bilo predvideno, se po dogovoru z lastnikom produkta vključijo dodatne uporabniške zgodbe. Če se pojavi zahteva z izredno visoko prioriteto, se sprint predčasno zaključi in se s sestankom za pripravo novega plana prične nov sprint (Cohn, 2004).

Dnevni SCRUM traja od 15 do 30 minut, sodelujejo člani razvojne ekipe, lastnik produkta in skrbnik metodologije. Predstavi se delo preteklega dne, predvideno delo v tekočem dnevu ter morebitne omejitve, ki preprečujejo izvajanje dela po planu. Uporaba programskih orodij za načrtovanje sprintov je posebej primerna, kadar ekipe ne delajo na isti lokaciji ali pa je potrebno zagotavljati visok nivo skladnosti s standardi, zakonodajo ali regulativo (Cohn, 2004).

Planiranje izdaj. Za večino projektov je najprimerneje planirati nove verzije oz. izdaje (angl. *Release*) za obdobje od 2 do 6 mesecev, posamezna iteracija pa običajno traja od 2

do 4 tedne. Dolžina iteracije je odvisna predvsem od časa za pripravo nove izdaje, stopnje negotovosti, stabilnosti prioritet ter potrebe po povratnih informacijah (Cohn, 2004).

Že na začetku razvoja produkta je smiselno definirati glavne sklope razvoja (angl. *Themes*) za vsaj nekaj naslednjih izdaj. Poleg rokov posameznih izdaj je potrebno opredeliti še prioritete posameznih uporabniških zgodb ter njihovo umestitev v posamezno verzijo. Pri razvrščanju prioritet po stopnjah visoka, srednja in nizka vedno ostaja nejasno, kakšna je meja med njimi. Primernejše je razvrščanje po metodi »MoSCoW« (angl. *Must Have, Should Have, Could Have, Wont't Have This Time*) (Cohn, 2004):

- zahtevano (angl. *Must*),
- potrebno (angl. *Shoud*),
- opcijsko (angl. *Could*),
- preloženo (angl. *Won't*).

Kot sta z raziskavo konkretnih implementacij ugotovila Rising in Janoff (2000), je metodologija Scrum še posebej primerna za majhne razvojne ekipe ter projekte, pri katerih ni mogoče pridobiti natančnih specifikacij.

Z razmahom agilnih metodologij v manjših podjetjih pa so se temu trendu v želji po krajšem času za vstop na trg pridružila tudi večja podjetja (Microsoft, Yahoo, Google, SAP ...). K agilnemu razvoju je večja podjetja prisilila tudi finančna kriza, saj so morali z manj sredstvi zagotavljati enako kakovosten in hiter razvoj. Agilen pristop pa ne predstavlja samo sprememb procesov, ampak tudi celotne organizacijske kulture, za kar je običajno potreben daljši čas (Waardenburg & Vliet, 2013). Največji izziv v velikih podjetjih, kjer so pričeli z uvajanjem agilnih metodologij, predstavlja integracija novih procesov v sicer strogo planski način dela (Lindvall et al., 2004).

4.3 Izvedba ocenjevanja pri uporabi agilnih metodologij

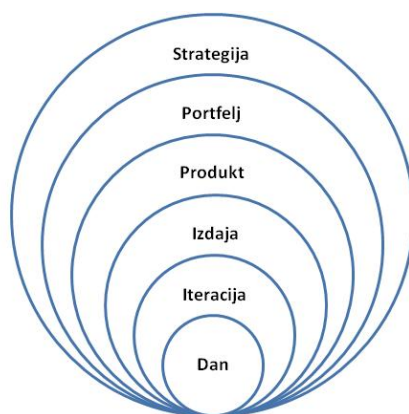
Tudi pri ocenjevanju agilnega razvoja je potrebno imeti osnovna znanja s področja ocenjevanja, saj je samo na ta način možno podati realne zaveze. Natančna ocena obsega in truda za razvoj je potrebna tudi za določanje prioritet ter planiranje iteracij in izdaj (Raith, Richter & Lindermeier, 2013).

Mnoge agilne ekipe uporabniške zgodbe razbijejo na naloge, preko katerih je nedvoumno jasno, kaj je treba sprogramirati. Nefunkcionalne zahteve (kvaliteta, zanesljivost, varnost ...) se obnašajo nekoliko drugače od uporabniških zgodb. Uvrščamo jih med omejitve (angl. *Constraints*) za funkcionalne zahteve. Nivo višje od uporabniških zgodb so funkcionalnosti, preko katerih sledimo viziji. Z dekompozicijo jih v uporabniške zgodbe pretvorimo ob načrtovanju izdaj. Pri planiranju izdaj temeljimo na funkcionalnostih, pri

planiranju iteracij pa na uporabniških zgodbah. Še en nivo višje od funkcionalnosti so epi (angl. *Epics*), s katerimi PO zagotavlja podjetju neko vrednost. Agilen razvoj je večkrat napačno razumljen kot razvoj brez formalnih zahtev, malo ali nič planiranja, brez koordinacije in dokumentacije. Leffingwell (2011) nasprotno trdi, da je agilen pristop najbolj discipliniran ter zagotavlja najvišjo kvaliteto. Slab sloves iz preteklosti mu tako preprečuje uporabo na poslovnih področjih, kjer bi lahko z njegovo uporabo imeli največ koristi.

Običajno se po Scrum-u planira samo naslednja iteracija. V določenih primerih pa je klub vsemu potrebno imeti tudi vpogled v celoten plan izvedbe, zato je bilo narejenih več raziskav ter teoretičnih modelov, kako tudi v metodologiji Scrum planirati za več iteracij ter kako omogočiti enostavno prilagajanje plana dejanskim razmeram (Golfarelli, Rizzi & Turricchia, 2013). Kot je prikazano na Sliki 10, se planiranje izvaja na več nivojih (Cohn, 2006):

Slika 10: Različni nivoji planiranja v agilnih modelih



Povzeto in prirejeno po M. Cohn, Agile estimating and planning, 2011, str. 28, slika 3.1

Večina ekip se ukvarja samo z notranjimi tremi nivoji. S planiranjem na nivoju produkta se ukvarja lastnik produkta. S planiranjem portfelja izbiramo produkte, ki imajo največjo vrednost in skladnost s strategijo podjetja.

Uporabniška zgodba predstavlja funkcionalnost, ki ima neko vrednost za naročnika oz. uporabnika. Sestavljajo jo tri komponente, in sicer pisni opis zgodbe, pogovor o zgodbi ter testni primeri. Pisni dokument služi planiranju in dokumentiranju, v pogovoru se pojasnijo podrobnosti in razjasnijo nejasnosti, s testnimi primeri pa določimo kriterije, po katerih lahko potrdimo, da je razvoj podpore za posamezno zgodbo dokončno zaključen. Uporabniške zgodbe morajo biti neodvisne druga od druge, opisi morajo biti dovolj prožni, da jih je med implementacijo možno prilagajati dejanski situaciji, imeti morajo neko vrednost za končnega uporabnika. Ne smejo biti preobsežne, omogočati pa morajo možnost ocene obsega ter pripravo testnih scenarijev. Uporabniške zgodbe so krajše od

primerov uporabe, praviloma pa naj ne bi zahtevale več kot 10 dni dela razvijalca. Uporabniške zgodbe niso skladne z zahtevami standarda za pripravo specifikacij IEEE Std. 830 (Cohn, 2004).

Ocenjevanje uporabniških zgodb poteka na način, ki dovoljuje spremembo ocene, če se med razvojem spremenijo izhodiščne predpostavke. Pristop mora biti dovolj prilagodljiv, da je uporaben tako za majhne uporabniške zgodbe kot za celotne »epe«. Omogočati mora hitro pripravo ter dovoljevati manjša odstopanja. Predvsem pa mora zagotavljati vpogled v napredovanje dela ter pripravo planov izdaj. Ocenjevanje vedno izvaja celotna ekipa, možnih pa je več pristopov. S pomočjo tehnik skupinskega odločanja ekipa v več iteracijah s pomočjo pojasnjevanj glede največjih odstopanj postopno konvergira k ciljni vrednosti. Po izvedbi ocen več zgodb je potrebno izvesti uravnoteženje ocen posameznih zgodb (angl. *Triangulation*). Na koncu vsake iteracije ekipa prešteje število točk, ki jih je uspešno realizirala ter na tak način določi hitrost (angl. *Velocity*) oz. zmogljivost (produktivnost) svojega razvoja (Cohn, 2004).

Načini ocenjevanja:

- poker planiranja (angl. *Planning Poker*),
- relativno ocenjevanje (angl. *Tabletop Relative Estimation*).

Najbolj pogosto uporabljena metoda za ocenjevanje obsega in časa trajanja projektov je ekspertna presoja. Poker planiranja je ena od novejših tehnik ekspertne presoje, ki izhaja iz Delphija in se pogosto uporablja v agilnih metodah, kot sta Scrum in Ekstremno programiranje. Prinaša boljše rezultate od individualne presoje, saj je prispevek vsakega od članov ekipe neodvisen od tega, kako glasen oz. vpliven je sicer (Mahnica & Hovelja, 2012). Če proces ocenjevanja (npr. poker planiranja) podpremo z orodji, lažje sledimo odklonom in pripravljamo še bolj natančne ocene (Cohn, 2006).

Poker planiranja je tehnika ocenjevanja, v kateri sodelujejo vsi člani razvojne ekipe, vključno z analitiki, preizkuševalci, načrtovalci podatkovne baze in grafičnih vmesnikov. Udeleženci s svojim ekspertnim znanjem, dekompozicijo in analogijo po pravilih igre pokra planiranja uskladijo končno oceno. Pravila so dokaj podobna metodi Delphy, uporabniki v nekaj iteracijah uskladijo oceno zgodbe v relativnih točkah. Ena točka lahko predstavlja idealen delovni dan ali pa ima bolj abstrakten pomen. Ko so ocene pripravljene, je potrebno uskladiti še prioritete posameznih zgodb. Pri tem je potrebno upoštevati finančni oz. poslovni vidik, stroške razvoja ter novo znanje in zmanjšanje tveganja, ki ga dosežemo z realizacijo posamezne zgodbe. Pri planiranju iteracij lahko že takoj na začetku vsako zgodbo razbijemo na manjše naloge, za katere sproti ocenjujemo kompleksnost in lahko z veliko gotovostjo sprejmemo zavezo, kaj bo realizirano. Če pa se ocenjevanje izvaja na višjem nivoju uporabniških zgodb in temelji predvidena hitrost razvoja na preteklih izkušnjah, lahko pride do večjih odstopanj.

Ocenjevanje hitrosti ekipe (angl. *Velocity*) lahko izvedemo na osnovi zgodovinskih podatkov, s spremljanjem hitrosti izvajanja začetnih iteracij ali pa zgolj ugibamo z neko stopnjo strokovnega znanja. Ko poznamo obseg zahtev in hitrost ekipe, lahko pripravimo terminski plan in ocenimo stroške.

Ocenjevanje obsega izvajamo s točkami uporabniških zgodb. Točke uporabniških zgodb so sicer relativne, kot enoto pa lahko izberemo tudi idealen delovni dan. Uteži ocenjevanja si lahko ekipa določi po lastni presoji, najbolj pogosti pa sta lestvici:

- Fibonaccijevo zaporedje : 1,2,3,5,8,13 ... ;
- potenca števila 2 : 1,2,4,8,16 ...

V primeru, da je za določene funkcionalnosti potrebno izvesti predhodno oceno stroškov, ali pa je realizacija predvidena v kasnejših obdobjih, se lahko pripravi bolj okviren opis zahtevanih funkcionalnosti na nivoju »epov«. Več medsebojno odvisnih uporabniških zgodb, za katere pripravljamo skupno oceno in planiranje izdaj, imenujemo »teme« (angl. *Themes*). »Epi« se že zaradi svoje velikost običajno obravnavajo kot »teme«.

Ocenjevanje izvajamo z:

- ekspertno presojo,
- analogijo,
- dekompozicijo,
- pokrom planiranja.

Za spremljanje napredka razvoja ter hitrosti ekipe uporabljamo:

- graf preostalega časa izdaje (angl. *Release Burndown Chart*),
- graf preostalega časa iteracije (angl. *Iteration Burndown Charts*),
- graf primerjave planirane in dejanske hitrosti (angl. *Planned and Actual Velocity*).

V primerih, ko imamo zelo dolgo čakalno vrsto (angl. *Backlog*), upravljanje s tem seznamom postane neučinkovito. Primerna rešitev v teh primerih je uporaba modela Kanban, ki temelji na principu omejene količine »dela v teku« (angl. *Work in Progress*). V delo se prevzame samo toliko nalog, za katere se lahko zagotovi realizacija. Ko je čakalna vrsta polna, ekipe niti ne razmišljajo več o novih zahtevah (Leffingwell, 2011).

4.4 Praktični primeri uvajanja agilnih metodologij

Izkušnje Microsoftove ekipe, ki je zaradi neučinkovitosti obstoječega procesa prešla na vitkejši proces Kanban, so v obsežnem članku opisali Anderson, Concas, Lunesu,

Marchesi in Zhang (2012). Ugotovitve temeljijo na izkušnjah ekipe, ki je bila več let odgovorna za izvajanje popravkov in manjših nadgradenj v sklopu vzdrževanja konkretnega produkta. Originalni proces je temeljil na priporočilih SEI PSP (angl. *Personal Software Process*) ter TSP (angl. *Team Software Process*) in je potekal po postopku:

- na začetku cikla se pripravi natančna ocena, ki jo morajo razvijalci poslovni strani podati v 48 urah po prejemu zahteve;
- sledi odločitev poslovnega lastnika storitve, ali se nadaljuje s postopkom;
- vse potrjene zahteve se po prioriteti evidentirajo v čakalno vrsto, enkrat mesečno lahko lastniki po potrebi preuredijo prioritete;
- 65 % inženirskega dela je namenjenega za kodiranje;
- 35 % inženirskega je predvidenega za testiranje.

Ker je bila hitrost razvoja manjša od prirasta novih zahtev, se je seznam zahtev na čakanju podaljšal na povprečno 80, vsak zahtev pa je čakal na realizacijo od 125 do 155 dni.

Ugotovljeno je bilo, da se največ časa izgubi za samo pripravo ocene. Preskakovanje z dela na trenutni nalogi na ocenjevanje novih zahtevkov pa je še dodatno zmanjšalo učinkovitost razvoja. Ključna sprememba je bila, da so omejili količino dela v teku (angl. *Work in Progress*) ter popolnoma opustili preliminarno pripravo ocen:

- Vsi zahtevki se takoj umestijo na seznam zahtev, brez priprave ocen.
- Enkrat tedensko lastniki odločijo, katere zahteve se umestijo v čakalno vrsto za izvedbo. Pred tem razvijalci ne podajajo nobenih ocen ali zavez.
- Število zahtev v delu je tako za razvijalce kot za preizkuševalce omejeno. Za vsako od teh 2 aktivnosti je lahko zahteva v čakalni vrsti, v izvajanju, ali pa je zaključena.
- Razvijalci prevzamejo zahteve s čakalne vrste in fokusirano delajo praviloma samo na eni zahtevi. Ko jo zaključijo, dobi status realizirano (angl. »Done«).
- Preizkuševalci prevzamejo zahteve s statusom »Done« v njihovo čakalno vrsto in prav tako hkrati delajo samo na eni zahtevi.
- Po zaključku testiranja se funkcionalnosti takoj objavijo.

Izkušnje z agilnim razvojnim modelom Scrum v podjetju s 30-letno zgodovino razvoja PO pa povzemajo Sjoberg, Johnsen in Solberg (2012). Po prehodu z zaporednega cikla na Scrum so v podjetju po nekaj letih spoznali, da jih časovno okviri ter obvezni sestanki na začetku in koncu vsakega sprinta omejujejo. Zaradi tega so prešli na Kanban ter po nekaj letih uporabe s pomočjo podatkov, ki so jih zbirali v času obeh metodologij izvedli znanstveno analizo. V času Scrum-a so uporabljali ocenjevanje s pokrom planiranja, sprinti so bili dolgi 3 tedne. Po 3 letih so ugotovili, da ta pristop ni primeren za vzdrževalne aktivnosti. Prešli so na model Kanban, ki so ga nekoliko priredili. Določili so količino dela

v teku, ocenjevanje in načrtovanje pa so izvedli, ko so prevzeli nalogo s seznama zahtev. Obdržali so dnevne stoječe sestanke, namesto demonstracij napredka na koncu sprinta pa so uvedli redne tedenske predstavitve napredka na projektu. Zadržali so enak nivo zagotavljanja kvalitete programske kode. Z analizo so ugotovili, da je število rešenih napak ostalo približno enako, za 3 krat pa se je povečala produktivnost in realizacija ostalih zahtev po dopolnitvah aplikacij.

V svoji knjigi s pragmatičnimi nasveti za ocenjevanje in agilen razvoj v nepredvidljivih razmerah Rothmanova (2015) med drugim podaja nekaj splošnih nasvetov:

- nikoli naj se ne ocenjuje točnega datuma brez možnega razpona ali ocenjene natančnosti;
- nujno je ozavestiti management, da se ocenjevanje izvaja v iteracijah;
- uporablja naj se postopno objavljane pomembnih funkcionalnost ter sprotne ocenjevanje preostalega dela;
- potrebno je zahtevati razvrstitev zahtev po vrstnem redu.

Na segmentu izvajanj ocenjevanj so zanimivi njeni naslednji nasveti:

- potrebno je dovolj razdelati uporabniške zgodbe;
- če ni možno podati ocene na osnovi empiričnih podatkov, lahko uporabiš »strokovno« uganje, vendar mora biti to jasno predstavljeno;
- če ni poznana hitrost ekipe, je smiselno uporabiti Kanban, vendar v tem primeru ni mogoče takoj oceniti celotnega seznama zahtev;
- če je določen tako nabor funkcionalnosti kot rok, nima smisla izvajati ocenjevanja. Bolje je takoj pričeti z delom, potrebno pa je dobiti vrstni red implementacije. V teh primerih je kot razvojni model predlagan Kanban, potrebno pa je ugotovi hitrost ekipe ter transparentno prikazovati napredek.

Podaja tudi nekaj praktičnih nasvetov za uspešnejše ocenjevanje in sam razvoj:

- Potrebno se je izogibati sočasnemu delu na več projektih, saj v teh primerih sploh ni mogoče podati ocene, ne glede na natančnost. Ni namreč znano, kako dolgo bo razvijalec delal na drugem projektu, koliko časa mu vzame, da se ponovno skoncentrira na osnovno nalogo ...
- Študentskemu sindromu, ki je na agilnih projektih sicer manj verjeten, se lahko izognemo z delom v paru, majhnimi nalogami ...
- Izbrana enota ocenjevanja je pomembna (človek/mesec, človek/dan, človek/ura). V takšnih enotah se namreč zamikajo tudi roki. Priporočena je dekompozicija na naloge, ki ne trajajo več kot dan ali dva in je možno takoj po zaključku prikazati delovanje. S tem se izognemo prelaganju realizacije na naslednji dan, naslednji teden ...

- Minimalna zahteva je, da je zgodbo možno realizirati znotraj ene iteracije (eno- do dvotedenske). Če tega nismo sposobni, še nismo zreli za agilen razvoj.
- Hitrost ekipe je odvisna od njenih članov, zato med projektom ne bi smeli spreminjati sestave ekipe.
- Zanesljivo oceno hitrosti ekipe se lahko poda šele po nekaj izvedenih iteracijah (npr. 3 dvotedenske iteracije). Pred tem je opredeljevanje hitrosti zgolj ugibanje.
- Kako dolgo traja testiranje, je odvisno od načina razvoja. Načeloma mora biti testiranju namenjenega vsaj toliko časa kot razvoju, pri čemer se testiranje prične že v zgodnjih fazah. Upoštevati je potrebno, da končni test vsebuje več korakov: testiranje programa, iskanje in odpravljanje napak, ponovno preverjanje popravka ...

5 PRENOVA RAZVOJA V IZBRANEM PODJETJU

5.1 Predstavitev Izbranega podjetja, d.d.

Strategija. Strateške umeritve izbranega podjetja so donosnost poslovanja, povečanje vrednosti, osredotočenost na stranko, enostavnost in stroškovna učinkovitost poslovnih procesov, zagotavljanje ustrezne kadrovske strukture in učinkovito upravljanje podjetja. S »Politiko upravljanja izbranega podjetja« (interno gradivo, 2014a) so podrobneje opredeljeni cilji, poslanstvo, vizija ter vrednote izbranega podjetja. V politiki so opisani tudi ključni deležniki in način sodelovanja z njimi, opredeljeni so organi družbe ter njihove pristojnosti in odgovornosti.

Tveganja. V izbranem podjetju se zavedamo operativnih tveganj, ki so povezana tako s spremembami v okolju kot z notranjo organiziranostjo ter internimi procesi. Poleg donosnosti in skladnosti s strategijo podjetja je potrebno pri izboru IT projektov ter večjih nadgradenj obstoječih IS upoštevati tudi tveganja, ki nastopijo bodisi v primeru odločitve za določen projekt oz. nadgradnjo, ali pa z njegovo zavrnitvijo. Aplikativna programska oprema predstavlja za podjetje operativno tveganje z vidika zagotavljanja neprekinjenega, pravilnega in varnega izvajanja poslovnih procesov, na katera lahko vpliva odpoved sistemov, njihovo nepravilno delovanje ali okvara opreme. Tveganje se ocenjuje na osnovi transformacijske matrike, ki temelji na kombinaciji verjetnosti dogodka ter njegovega učinka. Za odpravo operativnih tveganj je potrebno ugotoviti temeljne vzroke ter oceniti potrebne stroške, vire ter čas izvedbe, hkrati pa upoštevati predvidene učinke ukrepa. Pri izbiri najbolj učinkovitih ukrepov je potrebna presoja optimalnega razmerja med izpostavljenostjo tveganjem in donosnostjo ter ocena razmerja med stroški in koristmi. Najprej je seveda potrebno prepoznati ključna operativna tveganja, za katere v naslednjem koraku ugotavljamo temeljne vzroke. Možne ukrepe za odpravo temeljnih vzrokov lahko zberemo npr. s pomočjo tehnike viharjenja možganov (angl. *Brainstorming*). V sklopu presoje možnih ukrepov se ovrednoti vpliv na odpravo ključnega tveganja, pozitivni

finančni učinki posameznega ukrepa kot tudi stroški in potrebni viri za njegovo izvedbo (interno gradivo, 2015c).

Organizacija podjetja. Izbrano podjetje je hierarhično organizirana delniška družba s kolektivno upravo, ki ji podporo nudi več štabnih služb. Osnovna dejavnost je razdeljena med več poslovnih področij, podporo poslovanju pa poleg informatike izvajajo še ostala poslovna področja (finance, računovodstvo, ...).

Organizacijsko je področje informatike razdeljeno na več ločenih služb. Poleg služb, ki so odgovorne za podporo uporabnikom, sistemske storitve ter podporo upravljanju IT, je razvoj aplikativne programske opreme razdeljen med 3 službe, od katerih vsaka podpira delovanje enega od osnovnih poslovnih področij. Poslovna področja se med sabo razlikujejo predvsem glede na tip storitve ter trg, na katerem jih ponujajo. Določene storitve oz. produkti so tudi znotraj istega poslovnega področja podprti v več aplikacijah, saj je delo s posamezno aplikacijo prilagojeno naravi dela na določeni prodajni poti. Razvoj posamezne aplikacije je praviloma tudi organizacijsko ločen v samostojen oddelek.

Procesi. V izbranem podjetju je vzpostavljen »Sistem upravljanja poslovnih procesov« (interno gradivo, 2014b), v katerem so opredeljeni lastniki poslovnih procesov na strateškem nivoju, skrbniki poslovnih procesov na taktičnem nivoju ter skrbniki operativnih procesov na operativnem nivoju. Vsi procesi so evidentirani v registru poslovnih procesov, opredeljene so vloge in odgovornosti nosilcev aktivnosti, v teku pa je podrobnejše modeliranje trenutnih procesov ter priprava kazalnikov za merjenje uspešnosti in učinkovitosti posameznega procesa. Korenite spremembe poslovnih procesov, ob katerih je potrebno prilagoditi informacijsko podporo, se praviloma izvajajo kot projekti, bodisi z lastnim ali zunanjim razvojem, nakupom ali najemom. Za vse pobude za prenovu poslovnih procesov je potrebno preliminarно izvesti analizo in oceno izvedljivosti, skladnosti s strategijo podjetja ter ostalimi načrtovanimi projekti in projekti v delu. Prenovo procesa mora odobriti uprava ali drug ustrezen organ podjetja.

Izvajanje poslovnih in operativnih procesov je podrobneje opredeljeno v internih aktih. Na najvišjem nivoju so definirani zakoni, podzakonski predpisi in direktive, ki vplivajo na poslovanje podjetja. Sledijo jim statut družbe, kolektivna pogodba ter interni kodeksi (komunikacijski, etični ...) Na naslednjih nivojih so opredeljeni strategija ter politike družbe, poslovnik, pravilnik ter smernice in sistemi. Na najnižjem nivoju pa so podrobno dokumentirani poslovni ter operativni procesi s pripadajočimi navodili za delo.

Podporo upravljanju in linijskemu managementu nudi več odborov in komisij, ki jih lahko ustanovi nadzorni svet ali uprava. Odbori imajo običajno posvetovalno vlogo, lahko pa jim uprava dodeli tudi določene odločevalne pravice.

Poslovna področja svoje razvojne aktivnosti pretežno koordinirajo preko produktnega foruma. Ta organ je odgovoren za pripravo letnega plana novih produktov oz. storitev, ki jih posreduje v potrditev upravi, po uvedbi v produkcijo pa obravnava poročila o njihovi dobičkonosnosti. Produktni forum pri razvoju novih produktov oz. storitev sodeluje in usklajuje mnenja z vsemi poslovnimi področji, ki so povezana z novostmi, ter jih informira o sprejetih sklepih.

Projektno delo. V izbranem podjetju se projekti vodijo skladno s »Pravilnikom o projektne delu« (interno gradivo, 2016) ter priporočili in standardi PMI. Projektne pobude lahko poda vsak zaposleni, mora pa ustrezati določenim predpogojem, da se nato vodi skladno s projektne načinom dela. Projekti se izvajajo po projektne matrični organizacijski strukturi, specifično pa je, da glede na kompleksnost projekta poleg vodje projekta lahko obstajata še vlogi vsebinskega in tehničnega vodje projekta.

Faze izvedbe projekta:

- izdelava in finančna ocena projektne pobude,
- vrednotenje projektne pobude,
- potrditev projektne pobude,
- izdelava finančne ocene projektne listine,
- potrditev projektne listine,
- izvedba projekta,
- zaključek projekta in analiza projekta.

Že v projektne pobudi je potrebno izvesti okvirno oceno potrebnih virov in stroškov, pričakovanih koristi ter potrebnega dela, naknadno pa se izvede še podrobnejša finančna analiza projekta. Opredeliti je potrebno tudi proces ter metode merjenja stroškov in koristi. Vrednotenje pobude, ki obsega ocenjevanje poslovne vrednosti ter tveganja projekta, izvede poseben odbor. V sklopu vrednotenja poslovne vrednosti se poleg ostalih kriterijev presoja tudi finančna upravičenost investicije, tehnična izvedljivost pa se upošteva v okviru ocenjevanja tveganj. Pristojen odbor nato pripravi letni plan projektov, določa prioritete, usklajuje cilje s strategijo organizacije ter zagotavlja finančna sredstva. Odbor se sestane najmanj enkrat na četrtletje, projektne aktivnosti pa mora usklajevati z vsemi področji, ki so kakorkoli povezane s predvidenimi projekti.

Metodologija razvoja. Razvoj IS v opazovani službi se izvaja skladno z metodologijo, ki proces razvoja PO deli v faze zbiranja poslovnih in uporabniških zahtev, analize in načrtovanja, razvoja, testiranja, uvedbe v okolje ter naknadnega vzdrževanja. Metodologija navaja kot možne razvojne modele slapovni ali zaporedni model, iterativni model, iterativni model v kombinaciji s prototipnim ter inkrementalni model. Za vsako od faz je opredeljen proces, aktivnosti ter vloge, prav tako pa tudi izdelki in rezultati posamezne

faze. Predvidena so konkretna programska orodja, ne pa tudi standardi, ki bi jim moral razvoj slediti (interno gradivo, 2006).

Metodologija je bila pripravljena za namen prenove enega od večjih IS opazovanega podjetja, po zaključku prenove pa se ni več dopolnjevala in usklajevala z metodologijami ostalih področij razvoja v izbranem podjetju. Metodologija razvoja tako ni enotna za vsa razvojna področja, prav tako pa ne vsebuje sodobnejših agilnih razvojnih modelov. Delovanje informatike je sicer opredeljeno z enotnimi in formalno potrjenimi notranjimi akti na različnih nivojih. Na najvišjem nivoju so opisane politike, sledijo jim poslovniki in pravilniki ter smernice in sistemi. Na najnižjem nivoju so zapisana konkretna navodila za delo ter nekateri točno opredeljeni protokoli.

5.2 Predstavitev obravnavanega problema

Večino zahtev za nove funkcionalnosti in nadgradnje oz. spremembe funkcionalnosti v obstoječih IS generirajo posamezna poslovna področja. Največ zahtev izhaja iz razvojnih aktivnosti poslovnih področij, kjer uvedba novih produktov oz. storitev zahteva tudi sodelovanje informatike. Zahteva lahko izvira tudi iz potrebe po optimizaciji izvajanja poslovnega procesa, ki je že podprt z informacijsko podporo, ali pa uvajanjem nove informacijske podpore za obstoječe ali nove poslovne procese. Veliko predlogov za bolj enostavno, hitrejše in učinkovito delo s PO podajo končni uporabniki. Ostali generatorji zahtev so še zakonodajne in regulatorne zahteve, priporočila notranjih in zunanjih revizij ter zagotavljanja skladnosti. Ker se večje novosti v izbrano podjetje uvajajo preko projektnega načina dela, zahteve za prilagoditve ter integracijo obstoječih IS z novimi projekti prihajajo tudi s strani vodij posameznih projektov. Pomemben vir zahtev za spremembe tvorijo prilagoditve PO, ki so pogojene s spremembami strojne in systemske infrastrukture, preventivno vzdrževanje oz. odpravljanje latentnih napak, zagotavljanje varnosti in zaščite IS ter stabilizacija in optimizacija delovanja. Nezanemarljiv je tudi delež sprememb iz naslova korektivnih vzdrževalnih aktivnosti, kjer je za odpravo težav ali incidenta v produkciji potrebno spremeniti programsko kodo.

Z vidika tveganj in vpliva na uspešnost poslovanja ne moremo zanemariti nobenega področja, ki generira zahteve za spremembe. Ker pa ima večina zahtev, z izjemo povsem tehničnih prilagoditev ter programskih napak, v ozadju vedno tudi povsem strokovne odločitve, bo v nadaljevanju podrobneje opisan proces upravljanja s spremembami, ki jih generirajo posamezna poslovna področja.

V izbranem podjetju je bil v preteklih letih prenovljen oziroma na povsem novih arhitekturnih temeljih izgrajen celovit informacijski sistem za podporo poslovanja enega od treh ključnih poslovnih področij. Pred tem so bili poslovni procesi na tem področju podprti z več IS, ki so podpirali delovanje posamezne poslovne funkcije in so bili razviti še v programskih jezikih 3. generacije. Za vsako od aplikacij je bil odgovoren vodilni

inženir, ki je do potankosti poznal »svoj« produkt, ključne naročnike ter delno tudi strokovno področje in omejen nabor procesov, ki so bili podprti s posamezno aplikacijo. Kljub formalno predpisanim postopkom vlaganja zahtev za nove funkcionalnosti in dopolnitve PO, je v preteklosti veliko komunikacije potekalo direktno med naročniki in izvajalci. Prenova IS je trajala več let in se je izvajala na projekten način. Program prenove IS se je izvajal v obliki projektov, v sklopu katerih je bilo razvitih več aplikacij, ki pa so dokaj soodvisne ter uporabljajo enotno podatkovno bazo. Razvoj je po usmeritvah in navodilih strokovnih služb izbranega podjetja v večji meri potekal pri zunanjem izvajalcu, pri določenih aplikacijah pa so že od samega začetka sodelovali tudi interni razvijalci. Koordinacija razvojnih aktivnost je potekala na rednih tedenskih sestankih posameznih projektov ter usklajevalnih sestankih na nivoju celotnega programa, kjer so se zahteve za implementacijo novih funkcionalnosti v posamezni iteraciji vodile v obliki sklepov s teh sestankov, za spremljanje realizacije pa je skrbelo vodstvo projekta.

Po formalnem zaključku projektov ob koncu leta 2014 je bil ob prevzemu aplikacij v vzdrževanje poleg tehnične in uporabniške dokumentacije pripravljen tudi seznam nerealiziranih zahtev ter znanih napak. V prvih mesecih po prehodu v produkcijo je bilo, pričakovano, odkritih še precej novih napak, hkrati pa so tudi poslovna področja pričela vlagati zahteve za nove funkcionalnosti in dopolnitve obstoječih. Zahteve so dokaj hitro presegle zmožnosti internih razvojnih ekip in odobrenih sredstev za zunanje izvajalce, zato se je pričel podaljševati odzivni čas za odpravo napak, zahtevane funkcionalnosti pa niso bile realizirane v predvidenih rokih. Več kot je bilo zahtev na čakanju, težje je bilo upravljati s čakalno vrsto, povečala pa se je tudi zahteva vodstva za transparentnost nad stanjem posamezne zahteve.

Razvoj in vzdrževanje se je tudi po prevzemu IS izvajalo po že uveljavljeni metodologiji z enomesečnimi iteracijami. Ker je ob prehodu v produkcijo določeno število zahtev z začetka projektov ostalo nerealiziranih (med trajanjem projektov pa so nastajale še nove), je bilo v fazi vzdrževanja ob objavi verzij zaradi uvajanja novih funkcionalnosti praviloma potrebno nadgraditi podatkovni model. Zaradi kompleksnosti arhitekturne zasnove in soodvisnosti aplikacij so mesečne spremembe skupnega podatkovnega modela predstavljale veliko tveganje. Za končno izvedbo uporabniških, sistemskih in integracijskih testov je bilo na voljo namreč največ teden dni, kar ni zadoščalo za dovolj varno in stabilno delovanje.

Zahteve poslovnih naročnikov po novih funkcionalnostih so torej znatno presegle sposobnost razvojnih ekip za pravočasno in kakovostno realizacijo zahtev. Zaradi tega je bilo potrebno vzpostaviti sistem za transparentno ter učinkovito obvladovanje sprememb, ki temelji na strokovni presoji obsega, potrebnega truda ter časa za realizacijo zahtev. Hkrati pa je bilo potrebno poiskati tudi možnosti za spremembe poslovnih procesov razvoja in vzdrževanja PO, s katerimi se lahko hitreje odzivamo na zahteve naročnikov.

5.3 Pregled in predstavitev možnih ukrepov

5.3.1 Strateške, organizacijske ter procesne spremembe

Strategija. Pri doseganju strateških ciljev podjetja ima ključno vlogo tudi informatika, zato mora biti njena strategija skladna s strategijo celotnega podjetja. Ker lahko sodobne tehnologije poleg izvajanja strategije pomembno vplivajo tudi na njeno sooblikovanje, je smiselno obe strategiji pripravljati sočasno. Pri oblikovanju strategije informatike je potrebno upoštevati interese ključnih deležnikov ter na tem temelju pripraviti jasne cilje, ki jih želimo doseči. Za doseganje čim večje dodane vrednosti investicij v informacijsko tehnologijo je potrebno spremljati donosnosti posameznih IT projektov ter večjih nadgradenj obstoječih IS, prav tako pa tudi upravičenost stroškov ostalih vzdrževalnih aktivnosti. Poleg omejitve stroškov za informatiko bi bilo potrebno omejiti oz. spremljati tudi proračun posameznih poslovnih področij za zahtevane spremembe IS. Vodstvu podjetja ter posameznim poslovnim področjem je potrebno predstaviti naravo dela v informatiki, naše sposobnosti in omejitve. Na skupnih delavnicah je potrebno poiskati rešitve za učinkovitejše usklajevanje in boljšo komunikacijo ter poiskati optimalen način sodelovanja pri razvoju.

Tveganja. V izbranem podjetju smo v preteklem letu izvedli več delavnic, na katerih smo prepoznavali ključna operativna tveganja posamezne aplikacije ter izbirali najbolj učinkovite ukrepe za njihovo omilitev. Pri večini delovnih skupin so bili predlagani ukrepi podobni:

- izboljšanje komunikacije in informiranja,
- učinkovito in pregledno upravljanje sprememb,
- priprava nedvoumnih specifikacij,
- uporaba priporočenih praks in metodologij razvoja PO,
- dosledno izvajanje verifikacije in validacije PO,
- avtomatizacija testiranja in nameščanja,
- usposobljenost in motivacija razvojnih ekip, zadostno število razvijalcev in podpornih ekip,
- izboljšanje procesov razvoja PO ter njihove integracije v celoten proces razvoja novih produktov in storitev,
- optimizacija organizacije razvojnih služb v informatiki in poslovnih področjih.

Potrebno je zagotoviti finančne ter kadrovske vire ter pripraviti podrobnejši izvedbeni načrt za ukrepe, ki bodo dobili ustrezno potrditev.

Organizacija. Smiselno bi bilo preveriti, ali je obstoječa organizacijska struktura informatike, ki je usklajena z delitvijo ostalih poslovnih področij, najbolj optimalna. Izvira

namreč še iz časov, ko se je programska podpora pripravljala ločeno po posameznih poslovnih funkcijah, obdržala pa se je predvsem zaradi različnih tehnoloških platform. Po poenotenju arhitekture ob prehodu na nov IS opazovanega področja bi bilo potrebno ponovno preveriti pozitivne učinke konsolidacije ter tveganja, ki bi z združitvijo nastala. Dodatno bi bilo smiselno preveriti potrebo po ločenih oddelkih za razvoj posameznih aplikacij, ki temeljijo na skupni poslovni logiki. Z morebitno reorganizacijo bi lahko premestili kadrovske vire na segmente, kjer je trenutno najbolj potrebno. Pozitivne učinke bi bilo možno doseči tudi s spremenjenim razmerjem med številom sodelavcev, ki izvajajo podporo uporabnikom ter številom sodelavcev, ki so odgovorni za zagotavljanje kvalitete programske kode. Izvajanje razvoja in vzdrževanja v istih organizacijskih enotah ima sicer pozitivne učinke na segmentu izvajanja podpore produkciji, hkrati pa onemogoča pravilno planiranje razvoja ter znatno znižuje skupno zmožnost implementacije novih funkcionalnosti.

Procesi. Predvsem na segmentu upravljanja s spremembami so bile že uvedene določene spremembe, podrobneje bodo opisane v nadaljevanju. Za pripravo bolj natančnih ocen obsega, potrebnega truda, časa ter stroškov za razvoj IS bi bilo potrebno ta proces ocenjevanja formalizirati ter ga umestiti med ostale procese v informatiki ali pa te aktivnosti na ustrezen način integrirati v obstoječe procese razvoja. Potrebno je proučiti pozitivne učinke ter tveganja ob morebitni spremembi procesa razvoja, ki bi temeljila na agilnih metodologijah razvoja.

Projekti. Najbolj pogosto se sicer projekti zaženejo ob uvajanju novih produktov, smiselno pa je projektno voditi tudi spremembe poslovnih procesov. Predvsem kadar se zahtevajo znatna sredstva, je vpletenih več poslovnih področij, uvedba pa predstavlja večje tveganje za podjetje. Sprememba poslovnih procesov na segmentu razvoja PO ustreza vsem tem kriterijem. Pri projektnem delu se že sedaj vrednoti ekonomska upravičenost. V sklopu finančne analize se izračunava neto sedanja vrednost, notranja stopnja donosa, koeficient donosnosti naložbe ter doba povračila naložbe. Vhod v vse te analize pa je poleg poslovne vrednosti tudi strošek razvoja, ki v fazi presoje temelji predvsem na ocenah. Izboljšanje natančnosti podanih ocen razvoja IS je torej predpogoj za učinkovito upravljanje portfelja projektov.

Metodologija razvoja. Glede na trende sodobnih razvojnih metodologij bi bilo tudi v izbranem podjetju potrebno proučiti sodobne metodologije razvoja PO ter kjer je to mogoče, preiti na bolj agilen pristop. Konkreten predlog bo opisan v nadaljevanju.

5.3.2 Upravljanje sprememb informacijskih sistemov

Prvi ukrep, ki smo ga izvedli po prevzemu v vzdrževanje, je bil vzpostavitev sistema za upravljanje sprememb IS. V vodstvu informatike smo pripravili pregled vseh ključnih informacijskih storitev, formalno opredelili odgovorne vodilne inženirje za posamezno

aplikacijo ter v sodelovanju z naročniki podali tudi predloge vsebinskih lastnikov. Ker v procesu razvoja programske podpore za nove storitve nastopajo tako naročniki iz strokovnih služb kot uporabniki s področja prodaje in ostalih poslovnih funkcij, je bilo potrebno pripraviti interni akt, ki bo zavezujoč za vsa vpletena poslovna področja. S pomočjo sodelavcev, odgovornih za upravljanje poslovnih procesov, smo zmodelirali poslovni proces »Upravljanje sprememb IS« ter nato v informatiki pripravili krovni dokument »Sistem upravljanja sprememb na področju informacijskih sistemov« (interno gradivo, 2015b), ki ga je skupaj s sprejemom »Poslovnika o delovanju odbora za upravljanje sprememb« potrdila uprava. Odbor je tako postal formalen organ za podajanje zahtev za spremembe IS, ki presoja utemeljenost podanih zahtev ter določa prioriteto.

Zaradi lažjega obvladovanja zahtev ter želje po čim bolj transparentnem prikazu statusa posamezne zahteve, smo se odločili, da se zahteve ne vodijo zgolj v obliki zapisnikov ter raznih preglednic. Za obvladovanje poslovnih zahtev smo kreirali nov projekt v programskem orodju, ki smo ga že pred tem uporabljali pri razvoju PO. Ker je orodje primarno namenjeno razvijalcem v informatiki, smo za management ter upravljavce sprememb z ostalih poslovnih področij pripravili prilagojene vpoglede v status njihovih zahtev ter stanje posamezne planirane verzije. Orodje, ki ga v razvoju uporabljamo za obvladovanje sprememb v posamezni aplikaciji, smo nadgradili z možnostjo vnosa ocen izvedljivosti. Ker pri razvoju sodelujejo tudi zunanji izvajalci, smo poleg ocen po kategoriji del predvideli tudi možnost zajema ponudbene vrednosti po načelu fiksne cene.

Po prevzemu novega IS v vzdrževanje, usmerjanje in nadzor razvoja ključnih informacijskih sistemov, ki ne poteka na projektni način, izvaja »Odbor za upravljanje sprememb informacijskih sistemov«. Sestava ter pristojnosti so opredeljene s poslovníkom, njegova temeljna naloga pa je, da je razvoj novih IS ter nadgrajevanje funkcionalnosti obstoječih IS skladen s strategijo, prioritetami in poslovnimi potrebami izbranega podjetja. Cilj upravljanja sprememb je, da so le-te planirane, ovrednotene, razvrščene po pomembnosti, dokumentirane ter pred uvedbo v produkcijsko okolje ustrezno verificirane in validirane. Pri potrjevanju določene zahteve je potrebno upoštevati oceno tveganj, potrebne vire za izvedbo spremembe, s tem povezane stroške ter pričakovane koristi.

Postopek uvajanja sprememb v IS izbranega podjetja:

- priprava zahtevka za spremembo informacijskega sistema,
- filtriranje (selekcija) zahtevkov za spremembe,
- priprava funkcionalnih specifikacij,
- izdelava ocene izvedljivosti ,
- priprava predlogov umestitve zahtevkov v verzijo posamezne aplikacije,
- proučitev predlogov, usklajevanje in predlog plana končne verzije,

- potrditev plana verzije,
- uresničevanje potrjenega plana verzij aplikacij,
- izvedba testov sprejemljivosti,
- testiranje novih funkcionalnosti,
- izvajanje uporabniških testov,
- izvajanje integracijskih testov,
- izvajanje performančnih testov,
- potrditev obsega verzije,
- izvajanje namestitvenih testov,
- potrditev primernosti za prehod v produkcijo,
- namestitev aplikacij v produkcijsko okolje.

5.3.3 Prenova procesa razvoja programske opreme – 1. faza, izvedeno

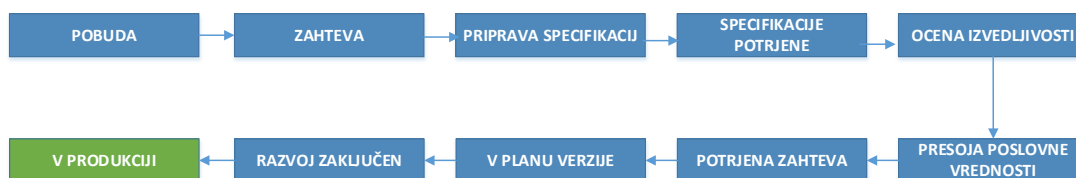
Po zaključku programa prenove IS obravnavanega poslovnega področja smo v informatiki ob prehodu v fazo vzdrževanja prevzeli dobre prakse s posameznih projektov ter jih predvsem z namenom obvladovanja stroškov ter boljše transparentnosti še dodatno nadgradili. V izbranem podjetju za obvladovanje opazovanega dela procesa razvoja PO (angl. *Application Lifecycle Management* – ALM) uporabljamo orodja podjetja Atlassian, ki se po mnenju skupine Gartner iz leta 2015 (objavljeno na njihovi spletni strani *Gartner Magic Quadrant for Application Development Life Cycle Management*), dokaj enakovredno postavljajo ob bok produktom IBM-a in Microsofta. Na voljo je še nekaj produktov istega proizvajalca, ki omogočajo celovito obvladovanje celotnega cikla razvoja PO, prav tako pa je na trgu dostopnih precej brezplačnih in plačljivih dodatkov različnih proizvajalcev, ki še poenostavijo in povečajo uporabnost osnovnih orodij. Confluence uporabljamo predvsem kot repozitorij dokumentacije, kot stično točko za izmenjavo informacij znotraj posamezne ekipe ter vizualizacijo pregledov in poročil iz Jire. V Jiri planiramo in spremljamo realizacijo posameznih zahtev, oblikujemo vsebino posamezne verzije ter načrtujemo izdaje verzij. Praviloma je za vsako aplikacijo oz. razvojno ekipo kreiran samostojen izvedbeni Jira projekt, za spremljanje zahtev naročnikov na krovnem nivoju pa uporabljamo ločen projekt za »Upravljanje sprememb«.

Trenutno zahteve za spremembe IS vlaga omejeno število oseb s posameznega poslovnega področja. Opis sprememb je običajno zapisan v obliki formalnih funkcionalnih specifikacij, pogosto zgolj opisno v samem zahtevku, občasno pa je zahteva v trenutku kreiranja še nepopolna in ne omogoča priprave realne ocene izvedljivosti. Za vse zahteve, ki so vložene do predpisanega roka in vsebujejo vsaj minimalno potreben obseg informacij, se v informatiki izvede ocena izvedljivosti. Ker je mnogokrat na osnovi ene poslovne zahteve potrebno izvesti spremembe v več aplikacijah, se takoj po izteku roka za vložitev zahtevkov za naslednjo iteracijo razvoja kreirajo naloge v Jira projektu posamezne

aplikacije, na osnovi katere vodja posamezne aplikacije izvede analizo zahteve in pripravi okvirno oceno izvedljivosti.

Trenutno se na enak način obravnavajo vse zahteve, ki jih odgovorne osebe poslovnih področij vnesejo v Jira projekt Upravljanje sprememb (v nadaljevanju US). Po sprejetem internem sistemu morajo biti vse te zahteve predhodno ovrednotene ter filtrirane. Ta korak se trenutno še ne izvaja v programskem okolju, zato se vse zahteve upoštevajo kot enakovredne, ne glede na celovitost funkcionalnih specifikacij pa se za vse pripravlja ocena izvedljivosti. Ker se tudi zaradi tega v čakalni vrsti za realizacijo novih funkcionalnosti nabira vedno več zahtev, je pripravljen predlog optimizacije vodenja zahtev v programskem orodju, s katerim bi dobili vpogled in nadzor nad prehajanjem statusa posamezne zahteve od pobude do umestitve v produkcijsko okolje (Slika 11).

Slika 11: Spremljanje statusa zahteve za spremembo



Pobude za nove produkte in storitve, ki jih generirajo razvojne službe posameznega strokovnega področja, se lahko v aplikacijo »Confluence« vnesejo s kreiranjem zahtevka za nov produkt (angl. *Product Requirements*). Vnos zahteve oz. pobude je omejen na odgovorne osebe s posameznega poslovnega področja. Predloga za vnos je prilagojena pravilom izbranega podjetja (Slika 12) ter zahteva vnos osnovnih podatkov, ki so potrebni za presojo na naslednjem nivoju (vsebinski opis, utemeljitev, namen in cilj ...).

Če pobuda za spremembo IS vsebuje vse potrebne podatke in koordinator za upravljanje sprememb na osnovi podane utemeljitve presodi, da je zahteva smiselna, kreira zahtevek v Jira projektu US. Zahtevek mora vsebovati dovolj podrobne funkcionalne specifikacije, na osnovi katerih je v informatiki možno pripraviti oceno izvedljivosti. Pri pripravi specifikacij poleg razvojne službe konkretnega poslovnega področja sodelujejo tudi tehnologi, ki poskrbijo za usklajenost zahtev še z ostalimi poslovnimi področji. Pri pripravi specifikacij po potrebi sodeluje tudi področje informatike, pred nadaljnjo obravnavo pa jo morajo praviloma potrditi odgovorne osebe vseh področij, na katere bo sprememba vplivala. Funkcionalne specifikacije so sestavni del Jira zahtevka, prav tako pa jih je možno shraniti v repozitorij dokumentacije v Confluence, kjer se pripnejo ali uvozijo v izvorni zahtevek oz. pobudo (Slika 13).

Po potrditvi ustreznosti funkcionalnih specifikacij lahko koordinator za upravljanje sprememb s kreiranjem podrejene zahteve v Jiri informatiki naroči izdelavo ocene izvedljivosti. Odgovorna oseba za posamezno aplikacijo izvede analizo ter poda oceno potrebnega truda v č/d.

Slika 12: Zahtevek za spremembo informacijskega sistema (Confluence)

On-line preverjanje terjatev

Predvidena objava	Februar 2015
Spremljanje razvoja	+ US-55 - INKASO - informatika knjižnice (zastopniki on_line) - prilagoditve v ITS analizi (posredovanje podatkov za on_line prilagoditev inkasov PP, v analizi viškov, prilagoditev zapiranja PP preko CKO (gl. plan CKO)) CLOSED + US-26 - On-line preverjanje terjatev ob inkasu v zalednem sistemu CLOSED
Status zahteve	DRAFT
Lastnik dokumenta	Robert Golob
Priprava specifikacij	M. Uršič
Vodilni razvijalec	P. Šušter, M. Rak
QA	P. Rak

Cilj

- manjša možnost napak ob izvajanju inkasa z on-line preverjanjem terjatev v zalednih sistemih z (sklici, zneski,...)
- poenostavljen in kontroliran proces oddaje gotovine

Namen

Trenutno so možne napake pri vpisu sklicev ter zneskov obrokov ob inkasu. S povezavo z zalednimi sistemi se tem napakam izognemo, hkrati so vzpostavljeni predpogoji za naknadno optimizacijo procesa zapiranja terjatev.

Na dveh večjih OE se še vedno izvaja oddaja gotovine zastopniškega inkasa na internih šaltirih, kar je prav tako povezano z možnimi napakami in veliko ročnega dela v zaledni podpori. S povezavo inkasa oddaje "zastopniške" gotovine s podatki z že oddanih spremnic se izognemo možnosti napak ter omogočimo avtomatizirano razknjiževanje terjatev na prodajni poti.

Uporabniške zahteve

#	Naziv	Kratek opis	Prioriteta	Opombe
1	Iskanje terjatev preko sklica	V prvi razvojni fazi mora biti podprto iskanje terjatev direktno preko sklica, omogočeno mora biti avtomatsko zapiranje terjatev v zalednem sistemu	visoka	
2	Iskanje terjatev po stranki	V drugi razvojni fazi je potrebno podpreti iskanje strank po več parametrih (DŠ, Ime in Priimek, Nalsov...) ter omogočiti izbor več terjatev iste stranke.	srednja	

Specifikacije za izdelavo programske podpore

[INKASO - dopolnitev podpore](#)

Glede na razpoložljive vire ter naravo zahtevka se odgovorna oseba v informatiki odloči, ali bo zahtevo realizirala interna razvojna ekipa ali pa bo posredovana zunanjim izvajalcem. V tem primeru se sprejme še odločitev, ali bo naročilo izvedeno po principu fiksne ponudbene cene ali pa glede na dejansko porabljene ure.

Področje informatike v sodelovanju s koordinatorjem za upravljanje sprememb ter vsebinskimi vodji aplikacij uskladi razmerje kadrovskih virov med razvojem novih funkcionalnosti in odpravo napak ter pripravi predlog plana verzij aplikacij. Odbor za upravljanje sprememb na osnovi prejetega predloga potrdi ali na seji uskladi obseg oziroma plan verzij aplikacij (Slika 14).

- > Upravljanje sprememb IS
- > Zahteve za spremembe in specifikacije po poslovnih področjih
 - > Informatika
 - > Korporacijski informacijski sistemi
 - > Prodaja
 - > Razvoj aplikacij
 - > Trženje
 - > Upravljanje sprememb in portfelja projektov
- > Zaledna podpora poslovanju
 - > Zahteve za spremembe ZPP
 - > On-line preverjanje terjatev
 - > Predloga za vlaganje zahteve za spremembo

On-line preverjanje terjatev

Created by Robert Golob, last modified on Jun 25, 2016

Predvidena objava	Februar 2015
Spremljanje razvoja	+ US-55 - INKASO - avtomatsko knjiženje CLOSED + US-26 - Online preverjanje terjatev ob inkasu v zalednem sistemu CLOSED
Status zahteve	DRAFT
Lastnik dokumenta	@Robert Golob
Priprava specifikacij	In Progress
Vodilni razvijalec	Robert Golob
QA	Robert Golob

Mesečni plan po aplikacijah		Predlog plana:						
		Key	T	Summary	Linked Issues	Labels	Status	Original Estimate
> 2016_01 Januar 2016								
> 2016_02 Februar 2016								
> 2016_03 Marec 2016								
> 2016_06 Junij 2016								
> 2016_09 September 2016								
> 2016_12 December 2016								
* ITB- [redacted] ka (2016_12)		ITBA-2667	+	US - [redacted] analitika - aktiviranje AF in PAF poslov - [redacted] in [redacted] in [redacted] analitika	US-696	Plan_201609 Plan_201612	IN PROGRESS	7 weeks, 1 da
* ITB- [redacted] le (2016_12)		ITBA-2670	+	US - Poročanje FURS - knjiga DDV - potrebno [redacted] [redacted] [redacted]	US-699	Plan_201609 Plan_201612	IN PROGRESS	4 days
* ITB- [redacted] (2016_12)		ITBA-2760	+	US - [redacted] [redacted] analitika - [redacted] na [redacted] (kasneje tudi ostale HD)	US-767	OUSIS Plan_201612	IN PROGRESS	3 days
* ITB- [redacted] r (2016_12)		ITBA-2779	+	ITB-Auf Prehod na WAS [redacted] Java 1.7, JSF [redacted]	US-230	Plan_201612	IN PROGRESS	2 weeks, 1 da
* ITB- [redacted] (2016_12)		ITBA-2785	+	US- Prilagoditev vseh [redacted] [redacted] za knjiženje v Navision po 1.1.2017	US-774	OUSIS Plan_201612	IN PROGRESS	1 week
* ITB- [redacted] r (2016_12)		ITBA-2786	+	US - [redacted] analitika - obračun tečajnih razlik - dopolnitev	US-773	OUSIS Plan_201612	IN PROGRESS	1 week, 2 day
> Arhiv 2015								

Vzpostavljena je integracija do repozitorija izvorne kode (Slika 16), tako da je možno povezavo med naročilom funkcionalnosti ter spremembo programske kode spremljati v obeh smereh.

58

Slika 15: Grafičen pregled mesečnega plana



Slika 16: Povezava med Jiro in SVN

US - Vnos preklincev (ITBP-2063)

Edit Comment Assign More Close Issue Reopen Issue Pošlji v potrjevanje

Sub-Tasks

1. Nova LDAP vloga za brisanje preklica **POTRJEN** *John Doe*
2. Poročilo o preklincih **POTRJEN** *Robert Golob*

Activity

Repository	Revision	Date	User	Message
SVN - shared	46275	Mon Apr 11 12:55:27 CEST 2016	Robert Golob	JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2064 Nova LDAP vloga za brisanje preklica JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2067 Kontrole na spreminjanje preklincev JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2093 Posebna oznaka partnerja za potrebe preklincev
Files Changed MODIFY /PartnerWeb/... /natPerson/generalDataTab.jsp MODIFY /PartnerEAF/... application.xml MODIFY /PartnerWeb/... pagecode/natPerson/NaturalPersonData.java MODIFY /PartnerWeb/... pagecode/natPerson/GeneralDataTab.java MODIFY /Common/tr... mmon/sifrant/Sifra.java MODIFY /Domain/tru... mon/vao/bp/NaturalPersonVAO.java MODIFY /PartnerWel... si/triglav/bp/beans/UserBean.java MODIFY /Common/tr... mmon/constants/Const.java MODIFY /PartnerWel... pagecode/natPerson/NaturalPerson.java MODIFY /PartnerWel... /WEB-INF/web.xml MODIFY /PartnerEAF/... m-application-bnd.xml				
SVN - shared	46284	Tue Apr 12 15:58:50 CEST 2016	Robert Golob	JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2064 Nova LDAP vloga za brisanje preklica JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2067 Kontrole na spreminjanje preklincev JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2093 Posebna oznaka partnerja za potrebe preklincev
Files Changed MODIFY /Common/branches/... av/common/sifrant/Sifra.java MODIFY /Common/branches/... av/common/constants/Const.java				
SVN - shared	46289	Tue Apr 12 16:00:30 CEST 2016	Robert Golob	JIRA ITBP-2063 US - Vnos preklincev <i>(dodaj komentar)</i> / ITBP-2064 Nova LDAP vloga za brisanje preklica
Files Changed				

Za vsako funkcionalnost je potrebno že ob planiranju vpisati verzijo, v kateri bo realizirana (Slika 17), po zaključku razvoja pa ustrezno spremeniti status realizacije.

Slika 17: Obvladovanje razvoja preko Jire

US - Izklopitev kontrolnega opozorila za OI

Details

Type:	New Feature	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	ITB-Partner 2.60.04
Component/s:	Aplikativne dograditve - PP		
Labels:	PP_Patch_201607		
Tip taska:	Nadgradnja		
Način obračuna:	FIX - Fixed price		
Izvajalec:			

People

Assignee: Matjaž
Reporter: Robert
Watchers: 1

Dates

Created: 10. May. 2016 12:14
Updated: 07. Jul. 2016 10:06
Resolved: 07. Jul. 2016 10:06

Preko teh podatkov (predvidene verzije ter statusa realizacije) lahko spremljamo napredek realizacije planiranih verzij že v času razvoja (Slika 18). Prav tako nam ta informacija služi za planiranje in izvedbo verifikacij in validacij, ko je posamezna verzija nameščena na testno okolje. Ob izdaji testne verzije namreč vodja testiranja v informatiki pozove odgovorne osebe naročnikov, da izvedejo validacijo realiziranih funkcionalnosti. Tudi potrjevanje pravilnosti implementacije posamezne funkcionalnosti se izvaja v Jiri, kar je omogočeno s prilagoditvijo oz. dodatnimi statusi za validacijo. Na osnovi potrjenih funkcionalnosti se pripravi verzija v dogovorjenem obsegu, po namestitvi na ustrezno testno okolje ter po izvedbi avtomatiziranih regresijskih in performančnih testov pa se organizirajo uporabniška testiranja. Rezultati uporabniških testiranj se v informatiki pregledajo, pripravi se seznam unikatnih napak, nato pa se izvede odpravljanje napak. V preteklosti je bil čas za odpravo odkritih napak znatno prekratek, po prehodu na daljši razvojni cikel pa je možno v tej fazi odpraviti večino odkritih napak. Kljub temu ugotavljamo, da bi bilo smiselno postopek verifikacije pričeti že pred samim testiranjem, saj se mnogokrat šele ob uporabniških testiranjih ugotovi, da določene specifikacije niso bile usklajene z vsemi deležniki, ali pa je v fazi načrtovanja tehnične implementacije prišlo do napačnega razumevanja zahtev iz funkcionalne specifikacije. Ker so popravki v tej fazi nekajkrat dražji od sprememb v predhodnih fazah, težimo k formalizaciji validacij funkcionalnih specifikacij ter sprotnem preverjanju tehničnih rešitev z naročniki.

Slika 18: Spremljanje realizacije planiranih nalog v verziji preko Jire

Version **REINSURE_1.66.03** UNRELEASED Release

☐ Start date not set ☐ Release date not set [Release Notes](#)

12 Issues in version 9 Issues done 2 Issues in progress 1 Issues to do

1-12 of 12 [View in Issue Navigator](#)

P	T	Key	Summary	Assignee	Status
↑	🔗	ITBRE-1466	Popravek datumov pri kopiranju obvestila o kršitvah	Primož	CLOSED

5.3.4 Ugotovitve po izvedeni 1. fazi prenove

V sklopu prve faze smo poleg uvedbe formalnega sistema za upravljanje sprememb, pričetka ocenjevanja truda in časa za realizacijo posamezne zahteve in celovitejše informatizacije obvladovanja procesa razvoja kot dodaten ukrep za stabilizacijo delovanja IS podaljšali razvojni cikel.

Ker je od delovanja novega IS ključno odvisno izvajanje osnovnega poslovnega procesa, smo po dogovoru z naročniki prešli na 3 mesečne razvojne cikle. Čas za izvedbo testiranja se je podaljšal, s tem pa smo dosegli tudi bolj stabilno delovanje IS. Po prehodu na daljši razvojni cikel pa smo se s prvimi težavami soočili že ob pripravi planov. Izkazalo se je, da je za tako dolgo obdobje skorajda nemogoče pripraviti plan, ki se med samim izvajanjem ne bi spremenil. Deloma je to pogojeno z zrelostjo poslovnih procesov vseh vpletenih področij, še bolj pa z dinamiko okolja, v katerem posluje izbrano podjetje. Dodaten problem, ki je ob daljšem razvojnem ciklu postal še očitnejši, pa je nezmožnost priprave natančne ocene truda in časa, potrebnega za razvoj posamezne funkcionalnosti. Tudi na tem segmentu obstaja več razlogov, zakaj se dejanski čas razvoja ne ujema z oceno. Prvi je zagotovo podajanje ocen v zgodnjih razvojnih fazah, ko specifikacije še niso popolnoma znane in je prisotnih preveč negotovosti, da bi se ocene lahko podale dovolj natančno. Drugi je dodajanje in spreminjanje zahtev med samim razvojem. Naslednji je podajanje nepravilnih ocen na osnovi subjektivne presoje, ki nima osnove v bolj poglobljenih analizah in poznavanju strokovnega področja ocenjevanja. Eden pomembnejših pa je tudi sedanji proces dela, zaradi katerega morajo razvijalci pogosto prekinjati trenutno delo pri implementaciji zahtevanih funkcionalnosti ter se posvetiti odpravi produkcijskih napak, pomoči uporabnikom, preskakovati z nedokončanih funkcionalnosti na nove zahteve. Kot največje probleme smo torej prepoznali:

- nepopolne specifikacije v trenutku podajanju zahtev, naknadno spreminjanje in dopolnjevanje specifikacij;
- dodajanje novih zahtev ter spreminjanje prioritet med razvojnim ciklom;
- nenatančne ocene obsega, truda ter časa za implementacijo posamezne zahteve;
- neučinkovita izraba delovnega časa razvijalcev zaradi preskakovanja na druge naloge (odpravljanje napak v produkciji, podpora uporabnikom ...);
- neustrezno planiranje na osnovi nenatančnih ocen ter nepredvidljivih kadrovskih virov.

Po letu in pol izkušenj s planskim načrtovanjem razvoja PO smo tako tudi v izbranem podjetju ugotovili določene pomanjkljivosti, ki se v veliki meri skladajo z ugotovitvami Microsoftove razvojne ekipe (Anderson et al., 2012).

V izbranem podjetju od internih razvijalcev ne zahtevamo, da bi na posamezno nalogo v Jiri vpisovali porabljen čas za realizacijo. Bolj stroga je zahteva do zunanjih izvajalcev, ki

morajo za posamezno zahtevo poleg porabljenega časa opredeliti še kategorijo zahtevnosti opravljenega dela, kar je osnova za pregled specifikacije prejetih računov.

Pregledi realiziranih zahtevkov (Slika 19) v določenem obdobju tako ne omogočajo neposredne primerjave produktivnosti posameznih izvajalcev, prav tako pa ne omogočajo sistematičnega izboljševanja ocen, saj nimamo primerjav med ocenjenim ter dejansko potrebnim delom.

Slika 19: Pregled realiziranih nalog po razvijalcih

Key	Summary	Assignee	Labels	Issue Type	Resolution	Status	Resolved
ITBI-8756	Preveriti delovanje aplikacije na ...	Iztok C...	Plan_201606	Task	Fixed	Potrjen	03. May. 2016 11:37
ITBI-9044	Preveriti delovanje aplikacije na ...	Branki...	Plan_201606	Sub-task	Fixed	Potrjen	19. May. 2016 12:18
ITBI-9037	Preveriti delovanje aplikacije na ...	Marko...	Plan_201606	Sub-task	Fixed	Resolved	26. May. 2016 11:20
ITBI-9198	Preveriti delovanje aplikacije na ...	Grego...	Remedy	Help Desk - Bug	Fixed	Resolved	06. May. 2016 12:00
ITBI-9200	Preveriti delovanje aplikacije na ...	Grego...	Remedy	Help Desk - Bug	Fixed	Resolved	06. May. 2016 13:11
ITBI-8990	Preveriti delovanje aplikacije na ...	Grego...	Remedy	Help Desk - Bug	Fixed	Resolved	06. May. 2016 13:18
ITBI-9226	Preveriti delovanje aplikacije na ...	Grego...	Remedy	Help Desk - Bug	Fixed	Resolved	10. May. 2016 11:59
ITBI-9228	Preveriti delovanje aplikacije na ...	Grego...	Remedy	Help Desk - Bug	Fixed	Resolved	10. May. 2016 12:09
ITBI-9256	Preveriti delovanje aplikacije na ...	Branki...	Remedy	Help Desk - Bug	Fixed	Closed	14. May. 2016 18:29
ITBI-9151	Preveriti delovanje aplikacije na ...	Marko...		Sub-task	Fixed	Resolved	02. May. 2016 08:40
ITBI-9152	Preveriti delovanje aplikacije na ...	Marko...		Sub-task	Fixed	Resolved	02. May. 2016 08:56
ITBI-8946	Preveriti delovanje aplikacije na ...	Marko...		Task	Fixed	Closed	02. May. 2016 08:57
ITBI-8931	Preveriti delovanje aplikacije na ...	Marko...		Bug	Fixed	Resolved	02. May. 2016 08:59
ITBI-9157	Preveriti delovanje aplikacije na ...	Branki...		Sub-task	Fixed	Resolved	03. May. 2016 10:35

Za boljše planiranje je, poleg natančnejšega ocenjevanja potrebnega truda, potrebno razpolagati tudi z natančnejšimi podatki o razpoložljivih kadrovskih virih. Ker je po trenutni organizacijski strukturi za izvajanje razvoja, vzdrževanja in podpore uporabnikov odgovorna ista ekipa, je namreč vnaprej nemogoče oceniti, koliko časa bo ta ekipa dejansko lahko namenila razvoju novih funkcionalnosti.

V izbranem podjetju za obvladovanje incidentov ter problemov v produkciji uporabljamo programsko orodje Remedy podjetja BMC, ki po Gartnerjevi analizi (objavljeni na njihovi spletni strani *Gartner Magic Quadrant for ITSSM*), prav tako spada med najboljša v segmentu obvladovanja IT storitev (angl. *IT Service Management*).

Vzpostavljeno imamo integracijo med orodjema Jiro in Remedy, tako da je možno slediti incidentu iz produkcije vse do programske kode, ki odpravlja to konkretno napako. Po odpravi napake, pripravi popravljene verzije in zahtevi za namestitev nove izdaje, je vzpostavljena tudi integracija v obratni smeri, ker se vse spremembe na produkcijskih okoljih spremljajo v orodju Remedy.

Tudi na segmentu podpore uporabnikom in odpravljanja napak ni zahtevano, da bi razvijalci, ki izvajajo podporo uporabnikom na 2. nivoju evidentirali porabljen čas za

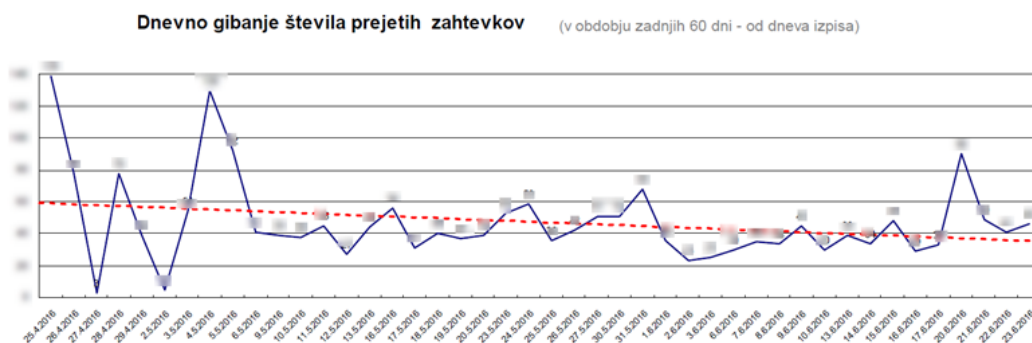
posamezen incident. Tako tudi trenutni pregledi rešenih Remedy zahtevkov po posameznem razvijalcu ne nudijo dovolj informacij, na osnovi katerih bi lahko ocenili čas, ki je bil na voljo za razvoj novih funkcionalnosti (Slika 20).

Slika 20: Pregled dela razvijalcev na podpori produkciji

Assigned Group*+	Assign Status*	Priority*	Incident ID*+	Summary*	Business Service*+
ITB	Matej E Assigned	High	INC000000344949	Na...	Prod
ITB	Matej E Assigned	High	INC000000344074	Avt	ITBU
ITB	Matej E Assigned	Medium	INC000000352788		Prod
ITB	Matej E Assigned	Medium	INC000000335319	e-w	GES
ITB	Matej E Assigned	Medium	INC000000335258	ITB	ITBU
ITB	Matej E Closed	High	INC000000352748	slu:	Prod
ITB	Matej E Closed	High	INC000000350117	V s	Prod
ITB	Matej E Closed	High	INC000000350107	nep	Prod
ITB	Matej E Closed	High	INC000000350007	NA	Prod
ITB	Matej E Closed	High	INC000000349500	spr	Prod
ITB	Matej E Closed	High	INC000000349423	vez	Upor
ITB	Matej E Closed	High	INC000000345565	neu	Prod
ITB	Matej E Closed	High	INC000000342754	spr	ITBU
ITB	Matej E Closed	High	INC000000336497	ni k	ITBU
ITB	Matej E Closed	High	INC000000332793	ITB	ITBU
ITB	Matej E Closed	High	INC000000332265	ITB	ITBU
ITB	Matej E Closed	High	INC000000330224	ITB	ITBU
ITB	Matej E Closed	High	INC000000329883	RA	ITBU
ITB	Matej E Closed	Low	INC000000349980	tež	Prod
ITB	Matej E Closed	Low	INC000000339646	pri	ITBU

Na osnovi spremljanja števila prejetih zahtevkov (Slika 21) ter porazdelitve po podpornih skupinah lahko sklepamo le, da razvijalci, analitiki oz. skrbniki aplikacij za operativno podporo uporabnikom namenijo znatno preveč časa (Slika 22).

Slika 21: Gibanje prejetih zahtevkov iz produkcije

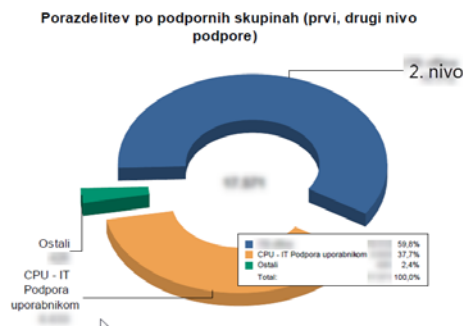


Če vztrajamo na trenutnem procesu dela, bi bilo torej natančnejše planiranje verzij ter spremljanje učinkovitosti razvijalcev možno samo ob doslednem vpisovanju porabljenega časa na posamezen Jira oz. Remedy zahtevek.

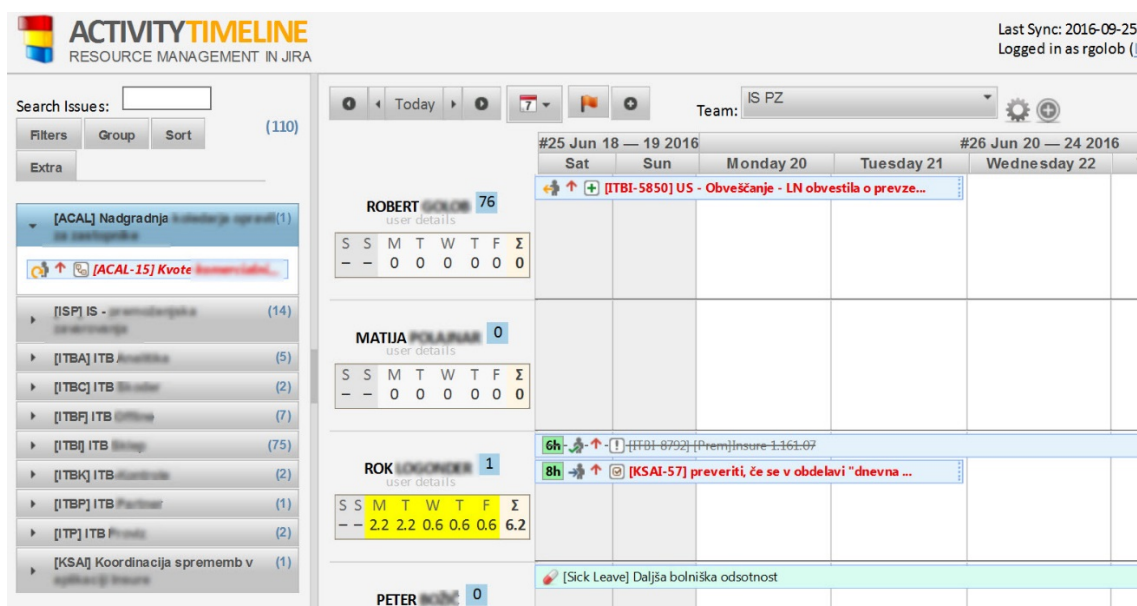
V tem primeru bi bilo možno izboljšati natančnost in pravilnost planiranja. Za razvojne ekipe s predvidljivim obsegom podpore uporabnikom na segmentih, kjer so tudi specifikacije dokaj stabilne, bi lahko za planiranje uporabili npr. dodatek Activity Timeline

(Slika 23). Aplikacija je integrirana z Jiro in omogoča kreiranje in prilagajanje plana dela preko enostavnega grafičnega vmesnika.

Slika 22: Obremenitev razvijalcev in analitikov z nudenjem podpore na 2. nivoju



Slika 23: Podrobno planiranje dnevnih aktivnosti z dodatkom Activity Timeline



5.3.5 Prenova procesa razvoja programske opreme – 2. faza, predlog

Ker lahko v informatiki vplivamo samo na nekaj parametrov, ki bi omogočili natančnejše planiranje, pa samo z več nadzora nad opravljenim delom razvijalcev ne bi dosegli zelenega cilja. Največji vpliv na razliko med ocenjenim in dejanskim časom razvoja ima namreč negotovost, ki izvira iz nepopolnega razumevanja zahtev, kar pa je v primeru razvoja PO dejstvo, ki se mu ni mogoče popolnoma izogniti. Tudi če se znatno izboljša kvaliteta funkcionalnih specifikacij ter natančnost ocen, še vedno nimamo vpliva na zunanje dejavnike, zaradi katerih prihaja do sprememb zahtev. Dinamika sprememb je sicer precej odvisna od same aplikacije oz. poslovnih procesov, ki jih podpira. Največja je

seveda na segmentu, kjer se tudi poslovni procesi in produkti najhitreje prilagajajo razmeram na trgu. Najmanjša pa je pri podpori poslovnim funkcijam, kjer so procesi zreli, so bile funkcionalnosti iz projektnih zahtev v celoti realizirane, zakonodajne spremembe pa so redke. Zato niti za vse aplikacije, ki jih razvijamo na obravnavanem področju, ne veljajo enotni kriteriji za izbiro najbolj optimalnega procesa razvoja.

Za področja razvoja, kjer so spremembe najbolj dinamične, specifikacije pa se hitro spreminjajo, je najprimernejša uporaba agilnih razvojnih modelov. Dolgo časa je prevladovalo mnenje, da agilni modeli razvoja niso primerni za poslovne aplikacije, kjer je potrebno dosledno zagotavljanje skladnosti z zakonodajo, regulativo ali zgolj internimi pravilniki. V zadnjih letih se ta pogled spreminja, saj je bilo na mnogih praktičnih primerih dokazano ravno nasprotno.

Temu se prilagajajo tudi razvojna orodja in tako je že nekaj časa na voljo dodatek Jira Agile, ki je zadnjih nekaj verzij za uporabnike Jire z aktivno vzdrževalno pogodbo brezplačen. Omogoča podporo za najbolj razširjeni agilni metodologiji Scrum in Kanban. Z nakupom dodatka »Agile Poker for JIRA« lahko avtomatiziramo postopek ocenjevanja, ki ga je sicer možno izvajati tudi ročno. Temelji na standardiziranem pristopu pokra planiranja, omogoča pa sodelovanje pri ocenjevanju ne glede na lokacijo.

Ker je izbrano podjetje večja gospodarska družba, za katero veljajo dokaj stroga pravila zagotavljanja skladnosti z zakonodajo in regulativo, se tudi spremembe poslovnih procesov izvajajo šele po tehtnem premisleku. Zaradi tega je v sklopu te naloge podan samo predlog možnih sprememb procesa razvoja PO, ki bi ga bilo v nadaljevanju potrebno še dodatno preveriti na manjšem pilotskem projektu ter šele nato formalizirati z novo metodologijo.

Razvoj po metodologiji Scrum

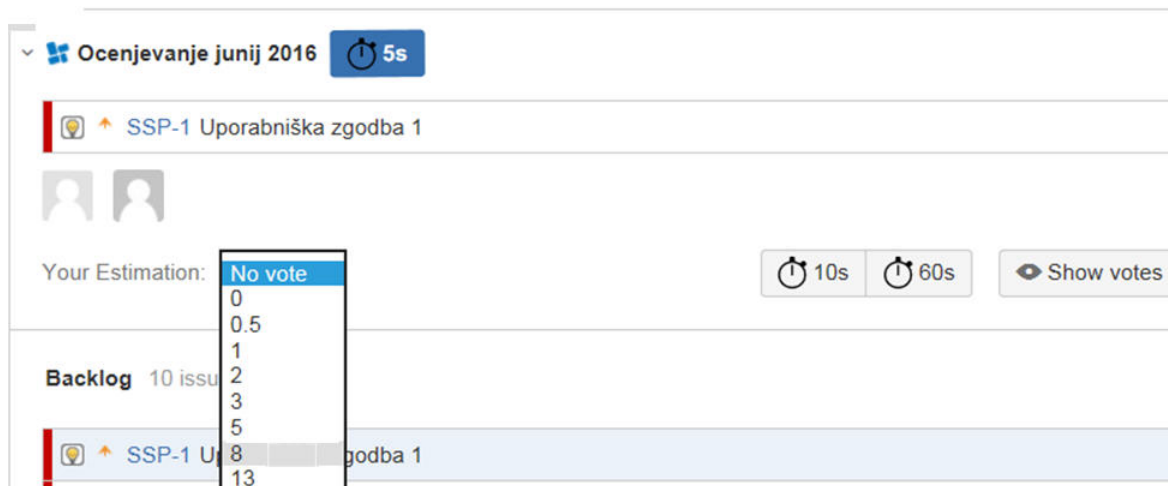
Za aplikacije, kjer so spremembe najbolj dinamične, specifikacije pa najbolj spremenljive, sem v predlogu proučil možnost uporabe razvojne metodologije Scrum. Ravno za te aplikacije pa veljajo tudi najbolj stroge zahteve za doseganje rokov, kdaj morajo biti novi produkti oz. storitve na voljo na trgu. Zaradi tega je potrebno ocenjevanju obsega, truda in potrebnega časa za razvoj nameniti vsaj toliko pozornosti kot v dosedanjem planskem načinu razvoja. Z uporabo programskih orodij ter izvedbo izobraževanj vseh razvijalcev o pomenu in tehnikah ocenjevanj pa želimo natančnost ocen še izboljšati.

Ocenjevanje, planiranje in spremljanje realizacije lahko sedaj v Jiri izvajamo tudi po principih agilnih metodologij.

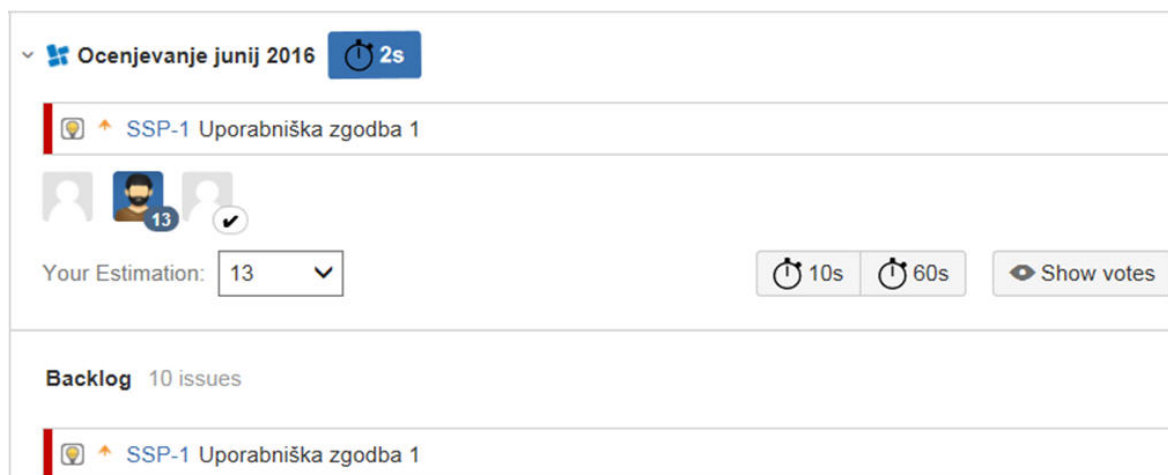
Ocenjevanje. V predlogu sem za izvedbo ocenjevanja uporabil poker planiranja. Obseg ocenjujem s točkami uporabniških zgodb, ki imajo za enoto idealen delovni dan, kot razpoložljiv nabor vrednosti pa sem izbral Fibonaccijevo zaporedje (Slika 24).

Ocenjevanje poteka po standardnih pravilih. Šele po zaključku ocenjevanja vseh sodelujočih oz po izteku predvidenega časa so ocene vidne vsem, nato se sodelujoči v nekaj iteracijah uskladijo glede končne vrednosti (Slika 25).

Slika 24: Ocenjevanje obsega agilnega razvoja s tehniko pokra planiranja



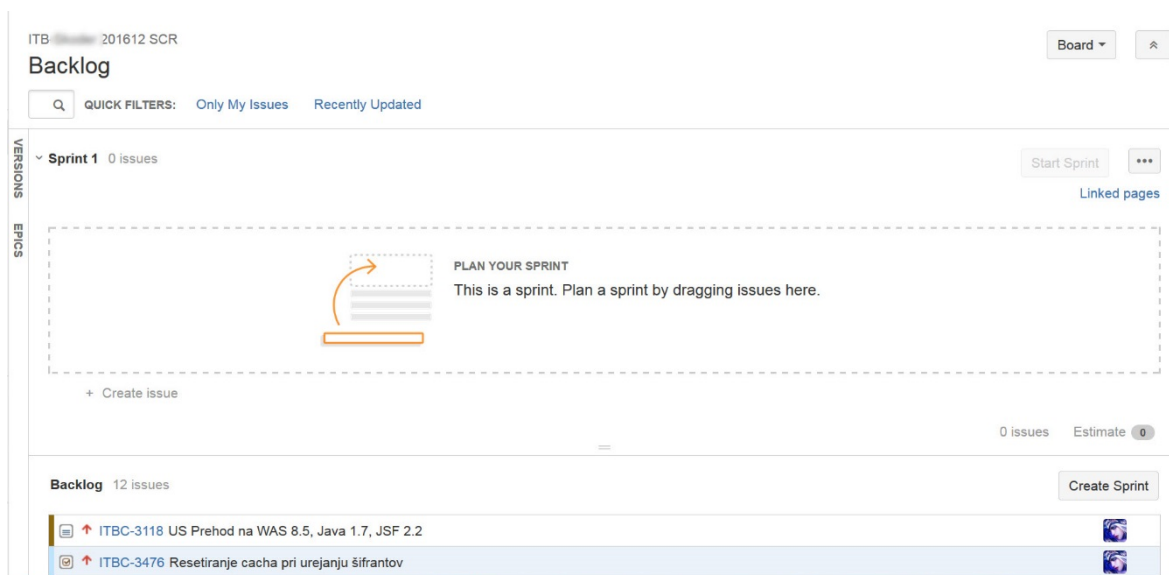
Slika 25: Usklajevanje ocene obsega posamezne zahteve med člani ekipe



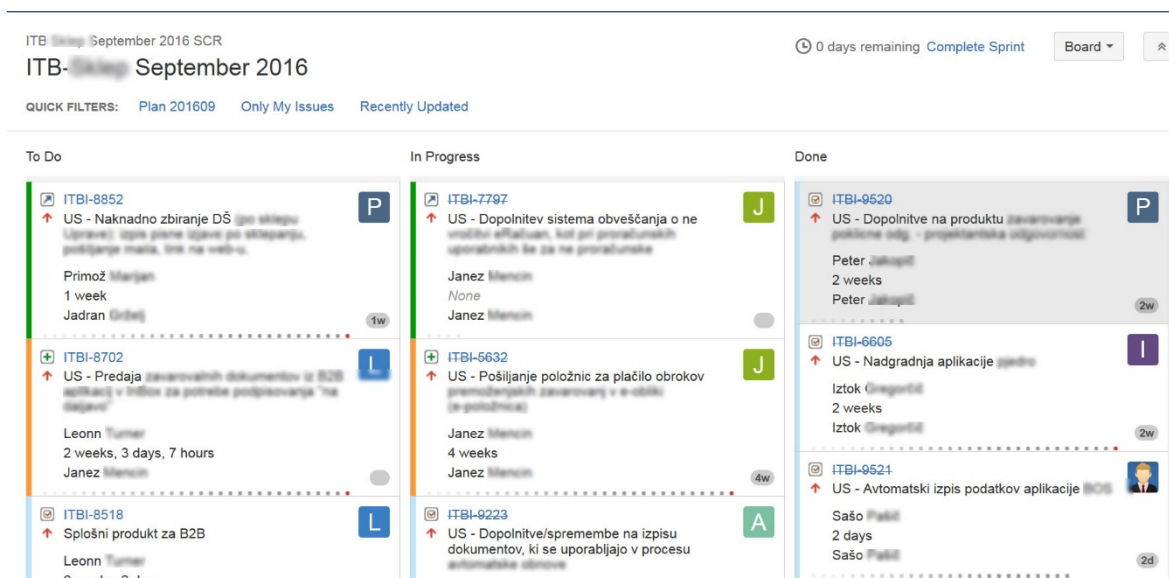
Planiranje. Na vsakem Jira projektu si lahko kreiramo nadzorno ploščo za vodenje razvoja po metodologiji Scrum ali Kanban. Uporabniške zgodbe ali posamezne zahteve lahko z načinom »povleci in spusti« umeščamo v posamezne sprinte, jim spreminjamo prioriteto (Slika 26) ter status realizacije (Slika 27).

Funkcionalnosti lahko planiramo za več iteracij oz. sprintov vnaprej. To je posebej pomembno v primerih obsežnejših dopolnitev, ko razvoja ni možno končati v okviru ene iteracije, potrebno pa je zagotoviti doseganje končnega roka (Slika 28).

Slika 26: Načrtovanje sprinta v Jiri

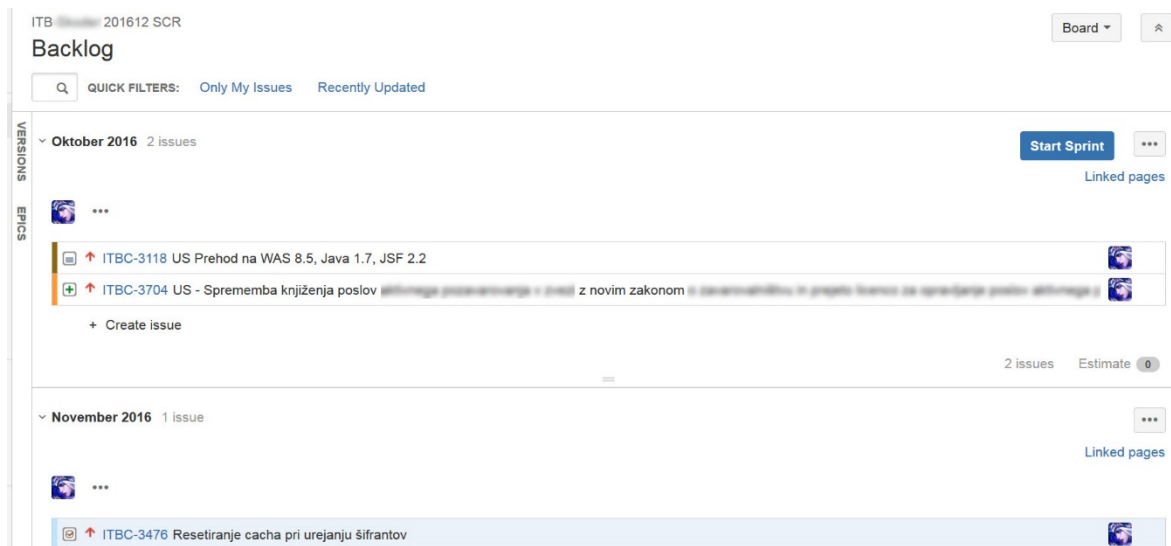


Slika 27: Spremljanje statusa realizacije na nadzorni plošči Scrum-a

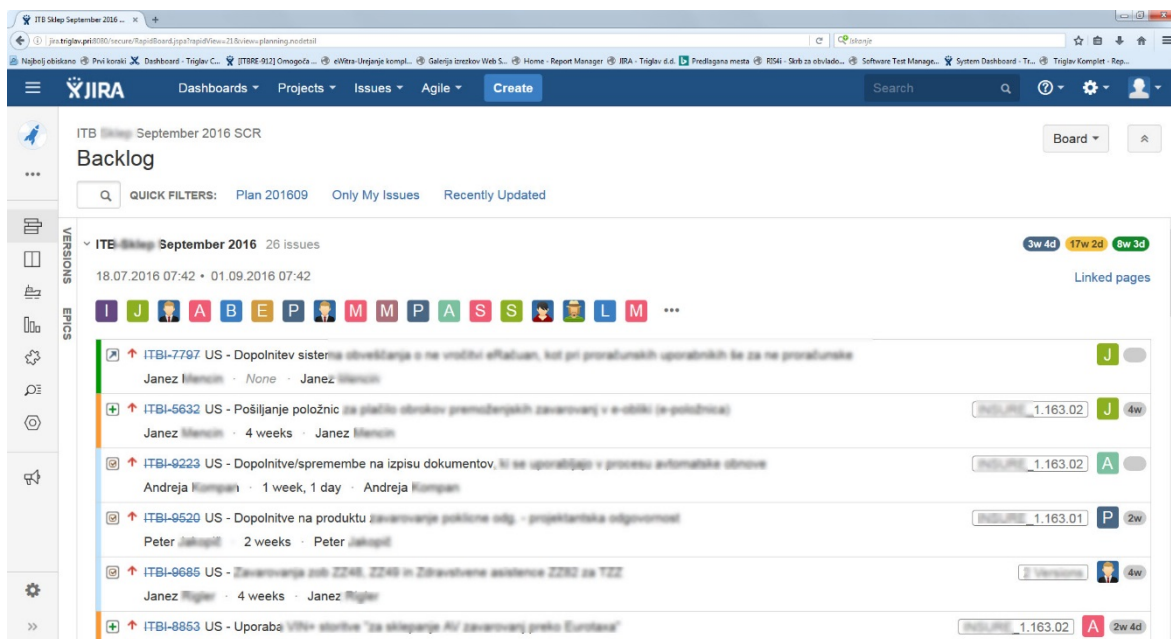


Ker je pred dejansko spremembo procesa in metodologije potrebno še dodatno preverjanje prednosti in slabosti, doslej še nobena produkcijska iteracija razvoja ni bila v celoti izvedena po tej metodologiji. Po metodologiji Scrum je bilo izvedeno vzporedno načrtovanje naslednje verzije pilotske aplikacije, ki bo v produkcijskem okolju objavljena po zaključku trenutne iteracije razvoja (Slika 29).

Slika 28: Načrtovanje več iteracij razvoja po modelu Scrum v Jiri



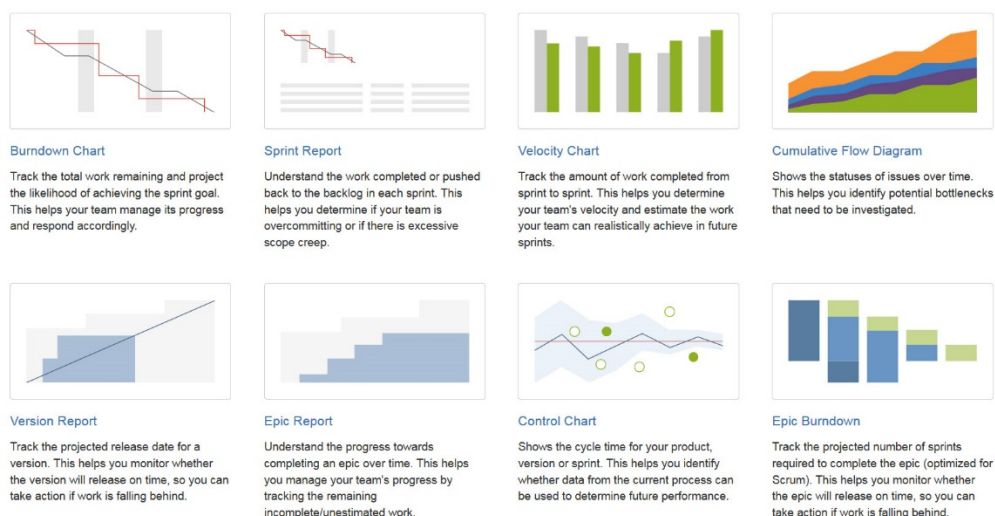
Slika 29: Vzoredno spremljanje realizacije preko nadzorne plošče Scrum-a



S tem so vzpostavljeni predpogoji, da bo možno napredek realizacije spremljati tudi preko izbranih grafičnih prikazov (Slika 30), ki jih podpira aplikacija Jira, npr.:

- graf preostalega časa (angl. *Burndown Chart*),
- poročila sprinta (angl. *Sprint Report*),
- grafični prikaz hitrosti ekipe (angl. *Velocity Chart*).

Slika 30: Različne opcije grafičnega prikaza napredka Scrum projektov



Z grafičnimi prikazi je smiselno podpreti predvsem poročila na managerskem nivoju, pri operativnem delu pa so še vedno bolj uporabna analitična poročila. Samo uvedba sodobnih orodij seveda še ne zagotavlja večje preglednosti in učinkovitosti, temveč je potrebno zagotoviti tudi dosledno izvajanje aktivnosti, ki so predvidene s posamezno metodologijo razvoja. Disciplinirano ažuriranje statusov izvedenih nalog je namreč predpogoj za pravi prikaz napredka dela (Slika 31) ter hitro ukrepanje v primerih odstopanj od zahtevanih rokov.

Slika 31: Graf preostalega časa sprinta



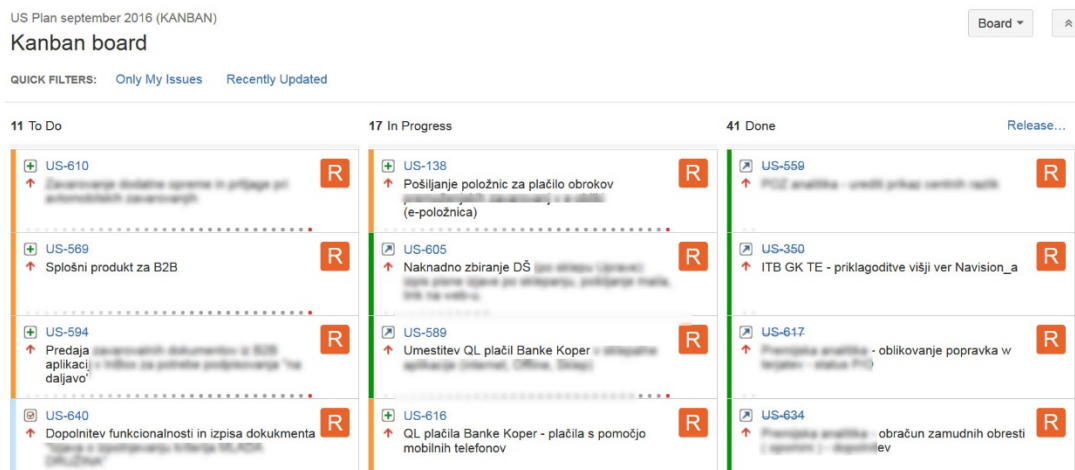
Razvoj po metodologiji Kanban

Glede na značilnosti posamezne aplikacije ter organizacijo dela posamezne ekipe je za določene primere primernejša uporaba metodologije Kanban, ki omejuje količino dela v teku. Razvojnim ekipam omogoča fokusiranje na trenutno aktualne zahteve in je lahko

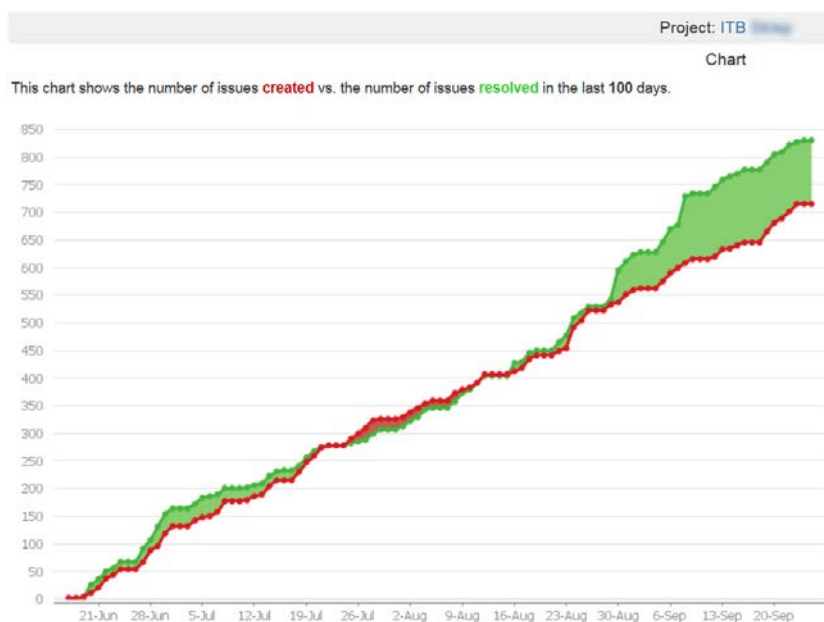
primerna izbira za vzdrževalne aktivnosti, kjer ne moremo vnaprej napovedati dinamike novih zahtev.

Podobno kot so ugotovili Sjoberg et al. (2012), tudi v izbranem podjetju obstajajo mnenja, da bi bila za vzdrževanje določenih aplikacij primerna razvojna metodologija Kanban. Ta pristop je aktualen predvsem za aplikacije, kjer ni potrebno predhodno usklajevanje prioritet med različnimi področji. Najprimernejši je za aplikacije, kjer morajo biti realizirane funkcionalnosti takoj po zaključenem razvoju na voljo za V&V ter je zahtevano, da se popravki ter novosti čim prej po potrditvi namestijo na produkcijsko okolje. Prikaz statusa zahtevkov v delu je prikazan na Sliki 32, na voljo pa so tudi vnaprej pripravljeni grafični pregledi (Slika 33).

Slika 32: Nadzorna plošča Kanban - status zahtevkov v delu



Slika 33: Primer grafičnega spremljanja zahtevkov pri uporabi modela Kanban



SKLEP

Sodobna informacijska tehnologija ima vedno pomembnejši vpliv na uspešnost in donosnost podjetij. Njena vloga ni več samo zagotavljanje izvajanja vizije in strategije podjetja, temveč ju tudi ključno sooblikuje ter podjetjem omogoča preboje na povsem novih poslovnih področjih. Ker pa morajo podjetja poleg donosnosti skrbeti tudi za dolgoročno preživetje, je potrebno ves čas spremljati dogajanje v okolju, ter zagotavljati zmožnost hitrega odziva tudi na nepričakovane spremembe. Zaradi tega je izrednega pomena, da podjetja dogajanje v poslovnem okolju spremljajo holistično ter odzive čim hitreje vključijo v svoje strategije.

Kot sredstvo za doseganje poslovnih ciljev podjetja pogosto uporabijo projekten način dela. Pri tem je ključno, da se strategija podjetja pretvori v konkretne, merljive projektne cilje. Čim obsežnejši nabor idej in predlogov za nove projekte je namreč predpogoj, da lahko vodstvo izbere res najboljše. Ker projekti pogosto presežejo predvidena sredstva, zamujajo predvidene roke in niso izvedeni v zahtevanem obsegu, je potrebno poskrbeti za učinkovito izvajanje managementa portfelja projektov. Razlogov za neuspešne projekte je seveda več, eden od pomembnejših pa je nepravilna ocena obsega, trajanja in stroškov, ki se izvede v začetnih fazah projekta oz. že pri pripravi projektne pobude. Nerealne ocene se pogosto odražijo v nizki kakovosti končnega produkta, razhajanje med realno oceno in krajšim rokom za izvedbo pa predstavlja projektno tveganje.

Za zagotavljanje ustreznega nivoja kakovosti je potrebno poskrbeti že z učinkovitimi poslovnimi procesi. Izboljševanje poslovnih procesov poleg višje kakovosti povečuje tudi produktivnost, kar skupaj omogoča doseganje zelenih poslovnih ciljev podjetij (nižji stroški, večji dobiček, hitrejši razvoj ...). Poslovni procesi so običajno podprti s tesno sklopljenimi informacijskimi sistemi, zato je ob vsaki spremembi procesa potrebna tudi zamudna nadgradnja programske opreme. Hitrejši odziv na spremembe omogoča servisno orientirana arhitektura, ki se ji vztrajno večja delež uporabe. Obstoječa funkcijska organizacijska struktura običajno ne zagotavlja maksimiranja vrednosti za kupce ter uspešnosti celotnega podjetja, saj vsaka organizacijska enota stremi k lastnim ciljem. Zaradi tega je potrebno v čim večji meri uvajati procesno managementsko organizacijo, ki poudarja potrebo po sodelovanju različnih funkcijskih enot in je osredotočena na odjemalce.

Informacijski sistemi so torej ključni za izboljševanje poslovnih procesov, prav tako so vsaj delno vključeni tudi v večino projektov. Ker so nadgradnje programske opreme potrebne tudi pri vseh spremembah in uvajanju novih produktov in storitev, delež investicij v informacijsko tehnologijo daleč presega vse ostale. Naložbe v informacijsko tehnologijo imajo sicer visoko donosnost, hkrati pa prinašajo tudi visoke stroške ter tveganja. Zaradi tega je za vodstvo podjetja potrebno vzpostaviti pregleden sistem nadzora nad sredstvi in

tveganji, hkrati pa z ustreznimi kazalniki prikazati tudi priložnosti in vrednost, ki jo informacijska podpora prinaša poslovni strani.

Ker je učinkovit razvoj PO ključen za doseganje poslovnih ciljev podjetja, je tudi na tem področju potrebno stalno izboljševanje procesov. Uspešnost razvoja PO je možno izboljšati z uvajanjem standardov s tega področja, pri čemer je potrebno celoten proces izpeljati postopno ter po skrbno načrtovanih korakih. Zaradi potreb po hitrem odzivu na zahteve trga se v zadnjem času vse bolj uveljavljajo agilne metodologije razvoja, ki so po deležu uspešno zaključenih projektov v znatni prednosti pred klasičnimi razvojnimi metodami. Pri obravnavi razvoja PO je pogosto premalo pozornosti namenjeno aktivnostim v fazi vzdrževanja, ki predstavljajo kar 70 % vseh stroškov v celotnem življenjskem ciklu posamezne aplikacije. Zato je tudi za to fazo potrebno izbrati pravilno razvojno metodologijo ter zagotoviti učinkovito obvladovanje zahtev za spremembe.

Odločitve za potrjevanje projektov ter določanje prioritet za razvoj in vzdrževanje programske opreme temeljijo na vrednotenju njihove ekonomske upravičenosti. Izhodišče za pripravo natančnih finančnih analiz pa je pravilna ocena projekta razvoja programske opreme oz. v primeru nadgradenj obstoječega informacijskega sistema ocena posamezne poslovne zahteve. Prenizke ocene ogrožajo doseg zastavljenih projektnih ciljev, previsoke pa vodijo k opuščanju poslovnih priložnosti. Zaradi tega bi morali osnove ocenjevanja poznati vsi managerji, odgovorni za pripravo ocen pa bi morali imeti poglobljena teoretična in praktična znanja s tega področja. Potrebno je jasno ločiti pomen in namen meritev, ocen, ciljev, plana in zaveze. Ocenjevanje mora biti nepristranski analitičen proces ne glede na zahtevane cilje, morebitna razhajanja pa je potrebno preko ocene tveganj vključiti v plane. Pri razvoju PO se ne moremo izogniti negotovostim in spremembam, zato je potrebno sprejeti dejstvo, da lahko ocene v začetnih fazah projekta znatno odstopajo od končnih vrednosti, z napredovanjem projekta pa se negotovost postopno zmanjšuje. Pogosta napaka razvijalcev PO opreme je, da podajajo preveč optimistične zaveze že v začetnem stanju lijaka negotovosti. Temu se lahko izognemo samo z dobrim poznavanjem tehnik ocenjevanja ter posebnosti, ki jih je potrebno upoštevati pri ocenjevanju obsega projekta, potrebnega truda, časa ter stroškov.

Na obravnavanem področju razvoja aplikativne PO smo k sistematičnemu ocenjevanju obsega, potrebnega truda ter časa bolj poglobljeno pristopili šele po prevzemu novega IS. V zadnjem letu in pol smo na tem področju naredili znaten napredek, saj se v tem trenutku ocenjevanje obsega, truda in časa, potrebnega za razvoj novih funkcionalnosti izvaja na vseh najpomembnejših aplikacijah opazovanega področja. Priprava ocen izvedljivosti v fazah razvoja in vzdrževanja PO je s potrditvijo Sistema za upravljanje sprememb na upravi tudi del formalno veljavnega internega akta. V aktualnem obdobju na opazovanem področju nismo sodelovali pri novih projektih, kjer bi lahko v celoti uvedli in preverili predlagane pristope na nivoju portfelja projektov. Strokovna izhodišča različnih tehnik ocenjevanja ter specifičnih pristopov za ocenjevanje obsega, truda, časa ter stroškov smo

uporabili pri vseh večjih nadgradnjah novega IS v sklopu vzdrževalnih aktivnosti. Posamezne zahteve so terjale tudi več človek/mesecev razvoja, večkrat je bilo potrebno usklajevati interese različnih poslovnih področij ter na osnovi poslovne vrednosti in predvidenih stroškov razvoja določati prioritete. Prva faza sprememb v procesu razvoja je bila primarno namenjena stabilizaciji novega IS. Uporabljeni so bili pristopi, s katerimi smo uvedli boljši nadzor nad uvajanjem sprememb ter z daljšimi razvojnimi cikli in temeljitejšim testiranjem zagotovili višjo kvaliteto PO. Po izvedenih spremembah in analizi rezultatov smo ugotovili, da z obstoječo organizacijo dela ter procesi razvoja ne uspemo realizirati vseh zahtev naročnikov. Zato sem v sklopu magistrske naloge proučil možne alternativne pristope. Na osnovi spoznanj, da se agilne metodologije uveljavljajo tudi v večjih podjetjih ter dosegajo najboljše rezultate glede uspešno zaključenih projektov, je podan predlog za pilotsko uporabo agilnih modelov razvoja Scrum in Kanban. Celotnega projekta po tej metodologiji na obravnavanem področju še nismo izvedli, smo pa pričeli paralelno spremljati razvoj obstoječega IS z orodji in pristopi iz agilnih modelov. Razen formalne dopolnitve obstoječe metodologije razvoja so cilji naloge doseženi. Ob že predvidenem poenotenju metodologij razvoja vseh aplikativnih področij pa bodo vključeni tudi najbolj uveljavljeni agilni modeli.

Namen in cilji magistrske naloge so omejeni na aktivnosti, ki se izvajajo na področju informatike. Da bi z novimi projekti s področja informacijske tehnologije ter razvojem in vzdrževanjem IS dosegli primarne cilje celotnega podjetja (čim večji dobiček, dolgoročno preživetje ...) pa je potrebno na tehtnico postaviti še poslovno vrednost posameznega projekta oziroma zahteve za novo funkcionalnost. Ob omejenih finančnih virih je namreč samo na ta način možno zagotoviti izbor in izvedbo projektov z najvišjo dodano vrednostjo, upoštevati pa je seveda potrebno še skladnost s strategijo ter zakonodajne zahteve.

Problematika ocenjevanja ključnih parametrov projektov oz. produktov razvoja PO je v nalogi obravnavana predvsem z vidika internega razvoja programskih rešitev v večjem podjetju, v katerem področje informatike izvaja zgolj podporno funkcijo. Poznavanje tehnik in principov ocenjevanja PO pa je vsaj v enaki meri pomembno tudi za vsa podjetja, v katerih je razvoj PO primarna dejavnost. Ključna spoznanja iz naloge so torej enako zanimiva za vse razvijalce PO, ne glede na to, v kakšni vlogi nastopajo. Kot so ugotovili že številni avtorji z obravnavanega področja, je tudi v izbranem podjetju največji problem podajanje zavezujočih ocen v zgodnjih fazah projekta oz. na osnovi nepopolnih specifikacij za nadgradnje obstoječih IS. Dognanje, da je lijak negotovosti dejstvo, ki se mu pri razvoju PO ni mogoče popolnoma izogniti, je potrebno predstaviti tudi poslovni strani. Sprejeti je potrebno dejstvo, da število novih zahtev narašča ter da se spremembam specifikacij ni mogoče izogniti. Zato se je nanje potrebno prilagoditi z izbiro ustreznega razvojnega modela ter prenoviti procese razvoja PO.

LITERATURA IN VIRI

1. Aaen, I. (2003). Software process improvement: Blueprints versus recipes. *IEEE Software*, 20(5), 86-93.
2. Anderson, D.J., Concas, G., Lunesu, M.I., Marchesi, M., & Zhang, H. (2012). A Comparative Study of Scrum and Kanban Approaches on a Real Case Study Using Simulation. *Agile Processes in Software Engineering and Extreme Programming*, Malmö, Springer, 111, 123-137.
3. April, A., & Abran, A. (2008). *Software Maintenance Management*. Hoboken: John Wiley & Sons.
4. Axelrod, C.W.(2013). *Engineering Safe and Secure Software Systems*. Norwood: Artech House.
5. Berlin, S., Raz, T., Glezer, C., & Zviran, M. (2009). Comparison of estimation methods of cost and duration in IT projects. *Information and Software Technology*, 5(4), 738-748.
6. Bittner, K., & Spence, I. (2007). *Managing Iterative Software Development Projects*. Crawfordsville: Addison-Wesley.
7. Bojnec, Š., Čepar, Ž., Kosi, T. & Nastav, B. (2007). *Ekonomika podjetja*. Koper: Fakulteta za management Koper.
8. Boutros, T., & Purdie, T. (2014). *The process improvement handbook: a blueprint for managing change and increasing organizational performance*. New York: McGraw-Hill.
9. Brown, J.T. (2008). *The Handbook of Program Management*. USA: McGraw-Hill.
10. Carr, N.G. (2003, maj). IT Doesn't Matter. Najdeno 11.5.2016 na spletnem naslovu <https://hbr.org/2003/05/it-doesnt-matter>
11. Chemuturi, M., & Cagley T.M. Jr. (2010). *Mastering Software Project Management*. New Delphy: J. Ross Publishing.
12. Cohn, M. (2011a). *Agile estimating and planning (11th ed.)* Upper Saddle River: Prentice Hall Professional Technical Reference.
13. Cohn, M. (2011b). *User Stories Applied: For Agile Software Development (16th ed.)* Boston: Addison-Wesley.
14. Collins, J.C. (2001). *Good to Great: Why Some Companies Make the Leap ... and Others Don't*. New York: Harper Business.
15. Cooper, R.G., & Edget, S.J. (1999). *Product development for the service sector*. b.k.: Basic Books.
16. Gartner Magic Quadrant for ITSSM. Najdeno 1.6.2016 na spletnem naslovu <https://www.gartner.com/doc/3118319?ref=SiteSearch&stkw=Magic%20Quadrant%20for%20ITSSM&fnl=search&srcId=1-3478922254>
17. Gartner Magic Quadrant for Application Development Life Cycle Management. Najdeno 1.6.2016 na spletnem naslovu <https://www.gartner.com/doc/2980720/magic-quadrant-application-development-life>

18. Golfarelli, M. , Rizzi, S., & Turricchia, E. (2013). Multi-sprint planning and smooth replanning: An optimization model. *Journal of systems and software*, 86(9), 2357-2370.
19. Hammer, M. & Champy, J. (1995). *Preurejanje podjetja*. Ljubljana: Gospodarski vestnik.
20. Hastie, S. & Wojewoda, S. Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. Najdeno 4.6.2016 na spletnem naslovu <https://www.infoq.com/articles/standish-chaos-2015>
21. Hill, P.R. (2001). *Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration*. Australia : International Software Benchmarking Standards Group Ltd.
22. Howcroft, D (2001). After the goldrush: deconstructing the myths of the dot.com market. *Journal of information technology*, 16(4), 195-204.
23. Izbrano podjetje, d.d. (2006). *Metodologija razvoja PO* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
24. Izbrano podjetje, d.d. (2014a). *Politika Upravljanja izbrane družbe* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
25. Izbrano podjetje, d.d. (2014b). *Sistem upravljanja poslovnih procesov* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
26. Izbrano podjetje, d.d. (2015a). *Poslovník o delovanju odbora za upravljanje sprememb* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
27. Izbrano podjetje, d.d. (2015b). *Sistem upravljanja sprememb na področju informacijskih sistemov* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
28. Izbrano podjetje, d.d. (2015c). *Analiza stanja in predlogi ukrepov za znižanje ključnih operativnih tveganj pri uporabi ali razvoju aplikacij* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
29. Izbrano podjetje, d.d. (2016). *Pravilnik o projektnem delu* (interno gradivo). Ljubljana: Izbrano podjetje, d.d.
30. Jones, C. (1998). *Estimating Software Costs*. New York: McGraw-Hill.
31. Jones, C. (2008). *Applied Software Measurment*. New Delphy: Tata McGraw-Hill.
32. Jones, C. (2014, avgust). *Achieving Software Excellence*. Najdeno 26.3.2016 na spletnem naslovu <http://static1.1.sqspcdn.com/static/f/702523/25136918/1404242669467/201407-Jones.pdf?token=8qLCSClvQ22QQCslQJsg6d2Wna4%3D>
33. Juran, J.M., & De Feo, J.A. (2010). *Juran's quality handbook: the complete guide to performance excellence*. USA: McGraw-Hill.
34. Jurič, B.M. & Pant, K. (2008). *Business Proces Driven SOA using BPMN and BPEL*. Birmingham: Packt Publishing.
35. Kan, S.H. (1995). *Metrics and Models in Software Quality Engineering*. Reading: Addison-Wesley.
36. Kern Pipan, K. (2010). *Vpliv stalnih izboljšav in človeškega kapitala na poslovno odličnost organizacije*. Kranj: Fakulteta za organizacijske vede.

37. Kohli, R., & Grover, V. (2008). Business value of IT: An essay on expanding research directions to keep up with the times. *Journal of the association for information systems*, 9(1), 23-39.
38. Kovačič, A., Jaklič, J., Indihar Štemberger, M. & Groznik, A. (2004). *Prenova in informatizacija poslovanja*. Ljubljana: Ekonomska fakulteta.
39. Kurapati, N., Manyam, V.S.C., & Petersen, K. (2012). *Agile Software Development Practice Adoption Survey. Agile Processes in Software Engineering and Extreme Programming*, Malmö, Springer, 111, 16-30.
40. Larman, C., & Basili, VR. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6), 47-56.
41. Lavallee, M., & Robillard, P.N. (2012). The Impacts of Software Process Improvement on Developers: A Systematic Review. *34th International Conference on Software Engineering*, Zurich, International Conference on Software Engineering, 113-122.
42. Leffingwell, D. (2001). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Upper Saddle River: Addison-Wesley.
43. Mahnic, V., & Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of systems and software*, 85(9), 2086-2095.
44. Marolt, J. & Gomišček, B. (2005). *Managent kakovosti*. Kranj: Založba Moderna organizacija.
45. Mc Connell, S. (1996). *Rapid development*. Redmond: Microsoft Press.
46. McConnell, S. (2006). *Software Estimation*. Redmond: Microsoft Press.
47. Mihelič, A. & Škafar, B. (2008). *Poslovni procesi*. Ljubljana: Zavod IRC.
48. Mithas, S., Tafti, A., Bardhan, I., & Goh, J.M. (2012). Information technology and firm profitability: mechanisms and empirical evidence. *MIS Quarterly*, 36(1), 205-224.
49. Monnappa A. (2015, julij). Pioneers of Project Management: Deming vs Juran vs Crosby. Najdeno 2.1.2016 na spletnem naslovu <http://www.simplilearn.com/deming-vs-juran-vs-crosby-comparison-article>
50. Moore, J.W. (2006). *The Road Map to Software Engineering*. Hoboken: John Wiley & Sons.
51. Moore, S. (2010). *Strategic project Portfolio Management*. Hoboken: John Wiley & Sons.
52. Moustafaev, J. (2011). *Delivering Exceptional Project Results*. USA: J.Ross Publishing.
53. Mulej, M. (2008). *Invencijsko-inovacijski management z uporabo dialektične teorije sistemov*. Ljubljana: Korona plus.
54. Niazi, M., Wilson, D., & Zowghi, D. (2005). A maturity model for the implementation of software process improvement: an empirical study. *Journal of systems and software*, 74(2), 155-174.
55. Phillips, D. (2004). *The Software Project Manager's Handbook (2nd ed.)*. Hoboken: John Wiley & Sons.

56. Pigoski, T.M. (1997). *Practical Software Maintenance*. Canada: John Wiley & Sons.
57. Pino, F.J., Pedreira, O., & Garcia, F. (2010). Using Scrum to guide the execution of software process improvement in small organizations. *Journal of systems and software*, 83(10), 1662-1667.
58. PMI. (2004). *Vodnik po znanju projektnega vodenja (3. izdaja)*. Kranj:Moderna organizacija.
59. PMI. (2011). *Practice standard for Project Estimating*. Newton Square, Pennsylvania: PMI.
60. PMI. (2013a). *PMBOK Guide. (5th. ed.)* Newton Square, Pennsylvania: PMI.
61. PMI. (2013b). *Software Extension to the PMBOK Guide (5th. ed.)*. Newton Square, Pennsylvania: PMI.
62. Pučko, D. (2008). *Strateški management 1*. Ljubljana: Ekonomska fakulteta.
63. Raith, F., Richter, I., & Lindermeier, R. (2013). Identification of inaccurate effort estimates in agile software development. *20th Asia-Pacific software engineering conference (APSEC 2013)*, 2, 67-72.
64. Read, T.J. (2009). *IT Value Network*. New Yersey: John Wiley & Sons.
65. Read, T.J. (2009). *The IT value network*. Hoboken: John Wiley & Sons.
66. Rising, L., & Janoff, N.S. (2000). The Scrum software development process for small teams. *IEEE Software*, 17(4), 26-32.
67. Rothman, J. (2015). *Predicting the Unpredictable: Pragmatic Approaches to Estimating Cost or Schedule*. b.k.: Practical Ink.
68. Saladis, F.P., & Kerzner, H. (2009). *Bringing the PMBOK Guide to Life*. Hoboken: John Wiley & Sons.
69. Schach, S.R. (1996). *Classical and Object-Oriented Software Engineering (3rd ed.)*. Chicago: Irwin.
70. Schmid, M. E.C. (2000). *Implementing the IEEE Software Engineering Standards*. Indianapolis: Sams Publishing.
71. Schwaber, K. (2004). *Agile Project Management with Scrum*. Redmond: Microsoft Press.
72. Sjoberg, D.I.K., Johnsen, A., & Solberg, J. (2012). Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software*, 29(5), 47-53.
73. Stutzke, R.D. (2005). *Estimating Software-Intensive Systems*. Upper Saddle River: Addison Wesley.
74. Tockey, S. (2005). *Return on software*. Boston: Addison-Wesley.
75. Unterkalmsteiner, M., Gorschek, T., & Islam Moinul, A.K.M. (2012). Evaluation and Measurement of Software Process Improvement-A Systematic Literature Review. *IEEE transactions on software engineering*, 38(2), 398-424.
76. Van Grembergen, W. (2009). *Enterprise governance of information technology: achieving strategic alignment and value*. New York: Springer Science & Business Media.

77. van Solingen, R. (2004). Measuring the ROI of software process improvement. *IEEE Software*, 21(3), 32-38.
78. van Waardenburg, G., & van Vliet, H. (2013). When agile meets the enterprise. *Information and software technology*, 55(12), 2154-2171.
79. Venkataraman, R.R., & Pinto, J.K. (2008). *Cost and value management in projects*. Hoboken: John Wiley & Sons.
80. Živkovič, A. (2004). *Metoda za določanje obsega objektno-orientiranih programskih projektov*. Maribor: FERI.

PRILOGE

KAZALO PRILOG

Priloga 1: Področja znanja projektnega managementa	1
Priloga 2: Procesi projektnega managementa po skupinah ter področjih znanja.....	2
Priloga 3: IEEE standardi s področja razvoja PO	3
Priloga 4: Področja znanja projektnega vodenja razvoja PO	8

PRILOGA 1: Področja znanja projektnega managementa

Obvladovanje integracije projekta: · priprava projektne listine · izdelava predhodne opredelitve projekta · priprava in izdelava plana za obvladovanje projekta · usmerjanje in obvladovanje izvajanja projekta · spremljanje in kontroliranje projektnega dela · celovito kontroliranje sprememb · zaključevanje projekta	Obvladovanje človeških virov v projektu: · planiranje človeških virov · pridobivanje projektne ekipe · razvoj projektne ekipe · obvladovanje projektne ekipe
Obvladovanje obsega projekta: · planiranje obsega · opredeljevanje obsega · oblikovanje WBS · overjanje obsega · kontroliranje obsega	Obvladovanje komuniciranja v projektu: · planiranje komuniciranja · posredovanje informacij · poročanje o opravljenem delu · obvladovanje udeležencev projekta
Obvladovanje časa projekta · opredeljevanje aktivnosti · razvrščanje aktivnosti · ocenjevanje virov za aktivnosti · ocenjevanje trajanja aktivnosti · pripravljanje terminskega plana · kontroliranje terminskega plana	Obvladovanje tveganj projekta: · planiranje obvladovanja tveganj · prepoznavanje tveganj · kvalitativna analiza tveganj · kvantitativna analiza tveganj · planiranje odzivov na tveganja · spremljanje in kontroliranje tveganj
Obvladovanje stroškov projekta: · ocenjevanje stroškov · planiranje stroškov · kontroliranje stroškov	Obvladovanje oskrbovanja projekta: · planiranje nakupov in nabav · planiranje pogodb · pridobivanje odzivov prodajalcev · izbiranje prodajalcev · zaključevanje pogodb
Obvladovanje kakovosti projekta: · planiranje kakovosti · izvajanje zagotavljanja kakovosti · izvajanje kontroliranja kakovosti	

PRILOGA 2: Procesi projektnega managementa po skupinah ter področjih znanja

Področje znanja	Skupine procesov projektnega managementa				
	Skupina zagonskih procesov	Skupina procesov planiranja	Skupina procesov izvajanja	Skupina procesov spremljanja in kontroliranja	Skupina procesov končanja
4. Obvladovanje integracije projekta	4.1 Priprava projektne listine	4.2 Priprava plana obvladovanja projekta	4.3 Usmerjanje in obvladovanje izvajanja projekta	4.4 Spremljanje in kontroliranje dela na projektu 4.5 Izvajanje celovite kontrole sprememb	4.6 Zapiranje (končanje) projekta
5. Obvladovanje obsega projekta		5.1 Priprava plana za obvladovanje obsega 5.2 Zbiranje (uporabniških) zahtev 5.3 Opredelitev obsega projekta 5.4 Oblikovanje WBS		5.5 Validacija (overjanje) obsega 5.6 Kontroliranje obsega	
6. Obvladovanje časa projekta		6.1 Priprava plana za obvladovanje časa projekta 6.2 Opredeljevanje aktivnosti 6.3 Razvrščanje aktivnosti 6.4 Ocenjevanje virov za aktivnosti 6.5 Ocenjevanje trajanja aktivnosti 6.6 Priprava terminskega plana			
7. Obvladovanje stroškov projekta		7.1 Priprava plana za obvladovanje stroškov projekta 7.2 Ocenjevanje stroškov projekta 7.3 Opredelitev proračuna (budgeta) projekta		7.4 Kontroliranje stroškov	
8. Obvladovanje kakovosti		8.1 Priprava plana za zagotavljanje kakovosti	8.2 Izvajanje zagotavljanja kakovosti	8.3 Kontroliranje kvalitete	
9. Obvladovanje človeških virov		9.1 Planiranje človeških virov	9.2 Zbiranje projektne ekipe 9.3 Razvoj projektne ekipe 9.4 Obvladovanje projektne ekipe		
10. Obvladovanje komunikacije	Identificiraj deležnike	10.2 Planiranje komuniciranja	10.2 Obvladovanje komuniciranja	10.3 Kontroliranje komuniciranja	
11. Obvladovanje tveganj projekta		11.1 Planiranje obvladovanja tveganj 11.2 Prepoznavanje tveganj 11.3 Kvalitativna analiza tveganj 11.4 Kvantitativna analiza tveganj 11.5 Planiranje odzivov na tveganja		11.6 Kontroliranje tveganj	
12. Obvladovanje oskrbovanja		12.1 Planiranje oskrbovanja projekta (nakupov in nabav)	12.2 Vodenje postopkov oskrbe projekta	12.3 Kontroliranje oskrbovanja projekta	12.4 Zaključevanje pogodb za oskrbovanje projekta
13. Obvladovanje deležnikov	13.1 Identifikacija deležnikov	13.2 Priprava plana za obvladovanje deležnikov	13.3 Obvladovanje dogovorov med deležniki	13.4 Kontroliranje dogovorov med deležniki	

Povzeto in prirejeno po PMI, PMBOK Guide, 2013, str. 61, Tabela 3-1

PRILOGA 3: IEEE standardi s področja razvoja PO

Tip	Oznaka	Naziv	Opis
Terminologija	IEEE Std 610.12-1990 (R2002)	IEEE Standard Glossary of Software Engineering Terminology	Slovar izrazov s področja programskega inženirstva.
Planiranje	IEEE Std 730-2002	IEEE Standard for Software Quality Assurance Plans	Opredeljuje strukturo in vsebino dokumentov za planiranje in zagotavljanje kvalitete programske opreme.
Planiranje	IEEE Std 828-1998	IEEE Standard for Software Configuration Management Plan	Opredeljuje vsebino dokumenta za planiranje in upravljanje s konfiguracijami programske opreme ter navaja zahteve za izvajanje specifičnih aktivnosti.
Dokumentacija	IEEE Std 829-1998	IEEE Standard for Software Test Documentation	Opisuje strukturo in vsebino dokumentov za planiranje, izvajanje in poročanje s področja testiranja programske opreme.
Dokumentacija	IEEE Std 830-1998	IEEE Recommended Practice for Software Requirements Specifications	Predlogi vsebine in karakteristike specifikacij (uporabniških) zahtev za programsko opremo.
Meritve	IEEE Std 982.1-1988	IEEE Standard Dictionary of Measures to Produce Reliable Software	Nabor meritev, s katerimi ugotavljamo oz. že v zgodnjih fazah razvoja ocenjujemo zanesljivost programske opreme.
Procesi	IEEE Std 1008-1987 (R2003)	IEEE Standard for Software Unit Testing	Opisuje pristop, osnovne koncepte in predpostavke pri izvajanju testov enot.
Procesi	IEEE Std 1012-2004	IEEE Standard for Software Verification and Validation	Opisuje procese verifikacije in validacije programske opreme, s katerimi preverjamo, ali rezultati posamezne aktivnosti ustrezajo zahtevam za to določeno aktivnost ter ali končni proizvod zadovoljuje potrebe uporabnikov. Tako z vidika procesov kot z vidika končnega proizvoda.
Dokumentacija	IEEE Std 1016-1998	IEEE Recommended Practice for Software Design Description	Opisuje strukturo in vsebino dokumentacije, ki jo je potrebno pripraviti v fazi načrtovanja programske opreme.
Procesi	IEEE Std 1028-1997 (R2002)	IEEE Standard for Software Reviews	Opisanih je pet pregledov ter postopkov za njihovo izvajanje (managerski pregled, tehnični pregled, inšpekcija, "hitri-pregled", revizija)
Orodja	IEEE Std 1044-1993 (R2002)	IEEE Standard Classification for Software Anomalies	Podaja enoten pristop in predloge klasifikacij anomalij v programski podpori in pripadajoči dokumentaciji.
Meritve	IEEE Std 1045-1992 (R2002)	IEEE Standard for Software productivity Metrics	Opredeljuje terminologijo ter definira pristope merjenja produktivnosti programske opreme.
Planiranje	IEEE Std 1058-1998	IEEE Standard for Software Project Management Plan	Opredeljuje strukturo in vsebino dokumenta za planiranje managementa projektov razvoja programske opreme.

Tip	Oznaka	Naziv	Opis
Meritve	IEEE Std 1061-1998(R2004)	IEEE Standard for a Software Quality Metrics Methodology	Opisuje metodologijo za vzpostavitev ustreznih kontrolnih mehanizmov ter meritve parametrov.
Procesi	IEEE Std 1062, 1998 Edition (R2002)	IEEE Recommended Practice for Software Acquisition	Predlog dobrih praks pri nabavi programske opreme (predvsem za razvoj in nadgradnje PO po naročilu).
Dokumenti	IEEE Std 1063-2001	IEEE Standard for Software User Documentation	Opredeljuje minimalne zahteve glede strukture in vsebine uporabniške dokumentacije. Tako za tiskane dokumente kot dokumentacijo v elektronski obliki.
Procesi	IEEE Std 1074-1997	IEEE Standard for Developing Software Life Cycle Processes	Opredeljuje definiranje življenjskega cikla razvoja programske opreme.
Orodja	IEEE Std 1175.1-2002	IEEE Guide for CASE Tool Interconnections-Classification and Description	Opisuje temeljne koncepte integracije CASE orodij na področju informacijskega inženirstva.
Procesi	IEEE Std 1219-1998	IEEE Standard for Software Maintenance	Opisuje proces vzdrževanja programske opreme ter management tega procesa.
Procesi	IEEE Std 1220-1998	IEEE Standard for Application and Management of the Systems Engineering Process	Opisuje aktivnosti in procese sistemskega inženirstva v celotnem življenjskem ciklu razvoja sistema.
Planiranje	IEEE Std 1228-1994 (R2002)	IEEE Standard for Software Safety Plans	Opisuje minimalno vsebino planov za razvoj, nabavo, vzdrževanje in odstranitev programske opreme varnostno kritičnih sistemov.
Dokumenti	IEEE Std 1233, 1998 Edition (R2002)	IEEE Guide for Developing System Requirements Specifications	Podaja navodila za pripravo specifikacij zahtev za sistem in zajema identifikacijo, organizacijo, predstavitev ter spremembe teh specifikacij. Opredeljuje tudi karakteristike in potrebno kvaliteto opisa teh zahtev.
Orodja	IEEE Std 1320.1-1998 (R2004)	IEEE Standard for Functional Modeling Language-Syntax and Semantics for IDEF0	Definira IDEF0 kot jezik za modeliranje odločitev, akcij in aktivnosti organizacije ali sistema.
Orodja	IEEE Std 1320.2-1998 (R2004)	IEEE Standard for Conceptual Modeling Language-Syntax and Semantics for IDEF1X 97 (IDEF object)	Definira dva konceptualna modelirna jezika (IDEX97), ki podpirata delo z relacijskimi in objektnimi bazami ter objektnimi modeli.
Dokumenti	IEEE Std 1362-1998	IEEE Guide for Information Technology-System definition-Concept of Operations (ConOps) Document	Podaja predlog strukture in vsebine dokumenta za delovanje oz. uporabo sistema z uporabnikovega vidika.

Tip	Oznaka	Naziv	Opis
Orodja	IEEE Std 1420.1-1995 (R2002)	IEEE Standard for Information Technology-Software Reuse-Data Model for reuse Library Interoperability:Basic Interoperability Data Model (BIDM)	Opisuje informacije, ki naj jih vsebujejo knjižnice s kodo, namenjeno ponovni/večkratni uporabi.
Orodja	IEEE Std 1420.1a-1996 (R2002)	Supplement to IEEE Standard for Information Technology-Software Reuse-Data Model for Reuse Library Interoperability: Asset Certification Framework	Dopolnjuje standard 1420.1 z dodatnimi informacijami (certificiranje sredstev).
Orodja	IEEE Std 1420.1b-1999 (R2002)	IEEE Trial-Use Supplement to IEEE Standard for Information Technology-Software Reuse-Data Model for Reuse Library Interoperability:Intellectual property Rights Framework	Dopolnjuje standard 1420.1 z dodatnimi informacijami (intelektualno lastništvo).
Orodja	IEEE Std 1462-1998 (R2004)	IEEE Standard: Adoption of International Standard ISO/IEC 14102:1995, Information Technology-Guideline for the Evaluation and Selection of CASE tools	Usmeritve za evalvacijo in izbiro CASE orodij.
Orodja	IEEE Std 1465-1998 (R2004)	IEEE Standard: Adoption of International Standard ISP/IEC 12119:1994, Informational Technology-Software Packages-Quality Requirements and Testing	Podaja usmeritev za pripravo zahtev za nivo kakovosti standardnih programskih paketov ter testiranje teh produktov.
Dokumenti	IEEE Std 1471-2000	IEEE recommended Practice for Architectural Description of Software Intensive Systems	Podaja konceptualni okvir ter priporočeno vsebino opisa arhitekture sistemov z intenzivno uporabo programske opreme.

Tip	Oznaka	Naziv	Opis
Terminologija	IEEE Std 1490-2003	IEEE Guide: Adoption of PMI Standard; A Guide to the Project Management Body of Knowledge (PMBOK® Guide)	Opredeljuje sprejetje standarda PMI kot IEEE standard.
Procesi	IEEE Std 1517-1999 (R2004)	IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes	Opredeljuje življenjske cikle procesa z vidika ponovne uporabe programskih komponent. Povezan s standardom IEEE/EIA 12207.
Procesi	IEEE Std 1540-2001	IEEE Standard for Software Life Cycle Process-Risk Management	Opredeljuje management tveganj z vidika življenjskega cikla procesa, vezano na standard IEEE/EIA 12207.
World Wide Web	IEEE Std 2001-2002	IEEE Recommended Practice for the Internet-Web Site Engineering, Web Site Management, and Web Site Life Cycle	Predlaga najboljše prakse pri načrtovanju internetnih in extranetnih strani.
Tehnologija	ISO 9000:2000	Quality Management Systems-Fundamentals and Vocabulary	Sistemi vodenja kakovosti - Osnove in slovar.
Procesi	ISO 9001:2000	Quality Management Systems-Requirements	Sistemi vodenja kakovosti – Zahteve.
Procesi	ISO 9004:2000	Quality Management Systems-Guidelines for Performance Improvement	Pristop z vodenjem kakovosti za zagotavljanje trajne uspešnosti organizacije.
Terminologija	ISO/IEC 9126-1:2001	Software Engineering-Product Quality - Part 1: Quality Model	Opredeljuje model, s katerim se ocenjuje kvaliteta produkta z več vidikov.
Procesi	IEEE/EIA 12207.0-1996	Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology-Software Life Cycle Processes	Podaja okvirni opis vseh procesov v celotnem življenjskem ciklu programske opreme.
Dokumenti	IEEE/EIA 12207.1-1996	Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology-Software Life Cycle Processes-Life Cycle Data	Podaja usmeritve za dokumentiranje podatkov, ki izvirajo iz procesov po standardu IEEE/EIA 12207.0.

Tip	Oznaka	Naziv	Opis
Procesi	IEEE/EIA 12207.2-1997	Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology-Software Life Cycle Processes-Implementation Considerations	Podaja usmeritve za implementacijo procesov, opisanih v IEEE/EIA 12207.0.
Meritve	IEEE Std 14143.1-2000	IEEE Adoption of ISO/IEC 14143-1:1998, Information Technology-Software Measurement-Functional Size Measurement-Part 1: Definition of Concept	Opisuje koncept meritev obsega programskih produktov po načelu funkcijske velikosti.
Procesi	ISO/IEC 14764:1999	Information Technology-Software Maintenance	Vsebuje usmeritve za implementacijo procesa vzdrževanja, opisanega v ISO/IEC 12207.
Orodja	ISO/IEC 15026:1998	Information Technology-System and Software Integrity Levels	Opisuje koncept nivojev integracije, opredelitev zahtev za integracijo ter proces za določanje le-teh.
Procesi	ISO/IEC 15288:2002	System Engineering-System Life Cycle Processes	Podaja okvirni opis vseh procesov v celotnem življenjskem ciklu sistemov, ki jih gradimo ljudje.
Procesi	ISO/IEC (Draft) 15504 (five parts)	Information Technology-Process Assessment	Opis metod za oceno procesov z namenom določitve zrelosti ter možnih izboljšav.
Procesi	ISO/IEC TR 15504 (nine parts)	Information Technology-Software Process Assessment	Opis metod za oceno procesov razvoja programske opreme z namenom določitve zrelosti ter možnih izboljšav.
Procesi	ISO/IEC 15939:2002	Software Engineering-Software Measurement Process	Opisuje izvajanje meritev v življenjskem ciklu programske opreme, opisanem v IEEE/EIA 12207.
Procesi	ISO/IEC 9003:2004	Software Engineering-Guidelines for the Application of ISO 9001:2000 to Computer Software	Usmeritve organizacijam za implementacijo standarda ISO 9001:2000 na področju nabave, prodaje, razvoja, uporabe ter vzdrževanja programske opreme.

PRILOGA 4: Področja znanja projektnega vodenja razvoja PO

Obvladovanje integracije projekta 4.1 Priprava projektne listine 4.2 Priprava plana za obvladovanje projekta 4.3 Usmerjanje in obvladovanje izvajanja projekta 4.4 Spremljanje in kontroliranje projektnega dela 4.5 Celovito kontroliranje sprememb 4.6 Zaključevanje projekta	Obvladovanje kakovosti projekta 8.1 Planiranje kakovosti 8.2 Izvajanje zagotavljanja kakovosti 8.3 Kontroliranje kakovosti
Obvladovanje obsega projekta 5.1 Planiranje obsega 5.2 Zbiranje zahtev 5.3 Opredeljevanje obsega 5.4 Oblikovanje WBS 5.5 Overjanje obsega 5.6 Kontroliranje obsega	Obvladovanje človeških virov v projektu 9.1 Planiranje človeških virov 9.2 Pridobivanje projektnega tima 9.3 Razvoj projektnega tima 9.4 Obvladovanje projektnega tima
Obvladovanje časa projekta 6.1 Planiranje časa 6.2 Opredeljevanje aktivnosti 6.3 Razvrščanje aktivnosti 6.4 Ocenjevanje virov za aktivnosti 6.5 Ocenjevanje trajanja aktivnosti 6.6 Pripravljanje terminskega plana 6.7 Kontroliranje terminskega plana	Obvladovanje komuniciranja v projektu 10.1 Planiranje komuniciranja 10.2 Obvladovanje komunikacij 10.3 Kontroliranje komunikacij
Obvladovanje stroškov projekta 7.1 Planiranje stroškov 7.2 Ocenjevanje stroškov 7.3 Dodeljevanje (določanja) finančnih sredstev 7.4 Kontroliranje stroškov	Obvladovanje tveganj projekta 11.1 Planiranje obvladovanja tveganj 11.2 Prepoznavanje tveganj 11.3 Kvalitativna analiza tveganj 11.4 Kvantitativna analiza tveganj 11.5 Planiranje odzivov na tveganja 11.6 Kontroliranje tveganj
	Obvladovanje oskrbe projekta 12.1 Planiranje oskrbe 12.2 Vodenje oskrbe 12.3 Kontroliranje oskrbe 12.4 Zaključevanje oskrbe