UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

POVEČANJE ZRELOSTI MANAGEMENTA POSLOVNIH PROCESOV V RASTOČIH PODJETJIH: PRIMER PROCESA RAZVOJA PROGRAMSKE OPREME

Ljubljana, junij 2017

GAŠPER KARANTAN VOZEL

IZJAVA O AVTORSTVU

Spodaj podpisani Gašper Karantan Vozel, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Povečanje zrelosti managementa poslovnih procesov v rastočih podjetjih: Primer procesa razvoja programske opreme, pripravljenega v sodelovanju s svetovalcem red. prof. dr. Jurij Jakličem,

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 ____________________ Podpis študenta:______________________

KAZALO

UVOD **1**

1 RAZVOJ PROGRAMSKE OPREME KOT PROCES **3**
1.1 Proces in metodologija razvoja programske opreme 3
1.2 Motivi vzpostavitve in izboljšave procesov 8
1.3 Agilni proces 10

2 PROCESNI VIDIKI RAZVOJA PROGRAMSKE OPREME **13**
2.1 Slapovni model 14
2.1.1 Struktura slapovnega modela 14
2.1.2 Kritike slapovnega modela 16
2.2 Spiralni model 17
2.2.1 Struktura spiralnega modela 17
2.2.2 Kritike spiralnega modela 19
2.3 Ekstremno programiranje 20
2.3.1 Struktura ekstremnega programiranja 21
2.3.2 Kritike ekstremnega programiranja 23

3 ZRELOSTNI MODELI **23**
3.1 Najpogostejše težave v procesu razvoja programske opreme 25
3.2 Model CMM 27
3.3 Model CMM z agilnimi procesi za majhna rastoča podjetja 31

4 ANALIZA PROBLEMSKEGA STANJA **33**
4.1 Opis podjetja 33
4.2 Opis problemskega stanja 33
4.3 Metodologija analize stanja 34
4.3.1 Trenutno stanje razvojnih procesov 36
4.3.2 Ciljno stanje razvojnih procesov 38

5 POSTOPEK IZBOLJŠAVE ZRELOSTNEGA MODELA RAZVOJA PROGRAMSKE OPREME **38**
5.1 Opis testiranih projektov 39
5.2 Testiranje slapovnega modela 40
5.3 Testiranje spiralnega modela 43
5.4 Testiranje ekstremnega programiranja 47

6 REZULTATI RAZISKAVE **48**

SKLEP **50**

LITERATURA IN VIRI **52**

KAZALO TABEL

Tabela 1: Stopnje zrelosti 24
Tabela 2: Enostavni prikaz razreza zahtev za iteracije 44
Tabela 3: Enostavni prikaz razvojnega načrtovanja iteracij 45
Tabela 4: Prikaz nalog izvedenih po fazah 46

KAZALO SLIK

Slika 1: Abstraktni model procesa 4
Slika 2: Življenjski cikel managementa procesa 7
Slika 3: Prikaz faz, ki sledijo v prvotnem slapovnem modelu. 14
Slika 4: Predlagane izboljšave prvotnega slapovnega modela. 16
Slika 5: Spiralni model procesa razvoja programske opreme 18
Slika 6: Zanka povratnih informacij v modelu ekstremnega programiranja. 22
Slika 7: Ključna področja modela CMM v0.2 28
Slika 8: Ključna področja modela CMM v1.0 29

UVOD

Pri razvoju programske opreme obstaja splošno priznana potreba po boljši praksi upravljanja razvojnih procesov, ki bi znižala stroške in izboljšala rezultat kakovosti razvoja (Humphrey, 1991, str. 150). Proces razvoja upravljanja programske opreme velja za enega od glavnih dejavnikov, ki določajo uspeh ali neuspeh razvojnih projektov (Redmill, 1990). Mlada podjetja imajo procese razvoja programske opreme zelo slabo definirane oziroma jih v veliki večini nimajo definiranih. Razlog za to je, da na začetku ne čutijo potrebe po natančni opredelitvi teh procesov, saj je število vpletenih ljudi zelo majhno. Ključne informacije si tako delijo preko preprostejših metod, kot so: e-pošta, SMS in ustna komunikacija. Do težav začne prihajati, ko podjetje raste in se v proces vključuje večje število ljudi. Ključne informacije se vse pogosteje izgubljajo, prihaja do nesporazumov in odstopanj v razvoju, kar vodi do slabših produktov, višjih stroškov in daljšega časovnega roka izvedbe. Podjetje kmalu začuti potrebo po bolj kakovostnem pretoku informacij in natančnejši definiciji razvojnega procesa (Laesvirta & Ribière, 2008). Namen magistrske naloge je definirati razvojni proces izdelave programske opreme. Pri vzpostavitvi procesa razvoja programske opreme se bomo zgledovali po primerih dobre prakse. Leta 1992 je izšla prva verzija zrelostnega razvojnega modela za programsko opremo (angl. *Capability Maturity Model for Sotware,* v nadaljevanju CMM), ki jo je po naročilu ministrstva za obrambo v ZDA (angl. *US Department of Defense*) razvil inštitut za programsko opremo (angl. *Software Engineering Institute – SEI*) na univerzi Carnegie Mellon v Pittsburgu (Tong, 1994). Glavna ideja omenjenega modela je, da zrel razvojni proces ustvarja produkte v dogovorjenem času, v okviru proračuna in skladno z dogovorjenimi zahtevami ter visoko kakovostjo (Paulk, Curtis, Chrissis, & Weber, 1993).

V skladu s specifikacijami zrelostnega razvojnega modela bi podjetje potrebovalo najmanj deset let, da bi zvišalo stopnjo zrelostnega modela iz prve na peto. Po raziskavah inštituta za programsko opremo na univerzi Carnegie Mellon v Pittsburgu, na kateri so ocenjevali 59 najboljših podjetij na svojem področju, ima večina podjetij (89 %) prvo stopnjo zrelostnega modela, 12 % drugo in 7 % tretjo stopnjo. Nobeno podjetje ni dosegalo kriterijev za četrto in peto stopnjo zrelostnega modela razvoja programske opreme (Tong, 1994). Ta raziskava je bila opravljena, ko je bil omenjeni zrelostni model še v pripravi. Do danes se je stanje občutno izboljšalo, a podatki vseeno veliko povedo o zahtevnosti zrelostnih stopenj. Pregledali bomo tudi kritične poglede na izboljšave procesa razvoja programske opreme, ki trdijo, da je pri implementaciji zrelostnega modela bolj smiselno uporabljati navodila (angl. *recepies*) in ne načrtov (angl. *blueprints*). Zagovorniki tovrstnih pogledov trdijo, da se z agilnim razvojem lahko doseže enaka stopnja obvladovanja procesa razvoja kot pri klasičnem razvojnem procesu (Aaen, 2003). V tej točki se odločitev izbire metodologije procesa zaplete, saj zagovorniki klasičnega pristopa razvoja programske opreme trdijo, da je treba proces razvoja programske opreme opisati kot program. Kritiki temu nasprotujejo, saj menijo, da proces uporabljajo ljudje, in ne ravno

obratno, zato je treba sposobne udeležence procesov postaviti v središče razvojnega procesa in graditi enostavne odprte strukture razvojnih procesov (Aaen, 2003).

Eden izmed temeljnih problemov izboljšave zrelosti procesa razvoja programske opreme je, kako zagotoviti, da bodo vsi udeleženci procesa sam proces enako razumeli. To pogosto pomeni, da je treba dopolniti in razširiti takó znanje posameznika kot tudi celotne skupine, ki je vpletena v razvojni proces. Liam Fahey in Laurence Prusak (1998) sta identificirala 11 napak pri upravljanju znanja v povezavi z obstoječim znanjem v organizaciji in procesom učenja.

Zrelostni razvojni model ne omejuje podjetja z določanjem razvojnega modela, če ta izpolnjuje koncepte in zahteve zrelostnega modela. Skozi zgodovino razvoja programske opreme se je razvilo mnogo razvojnih modelov: model programiraj in popravljaj (angl. *code-and-fix*), postopni model (angl. *stepwise*), slapovni model (angl. *waterfall*), transformacijski model, spiralni model (Misra, Kumar, Kumar, Fantazy, & Akhter, 2012). Mnogo omenjenih razvojnih modelov je imelo ključne pomanjkljivosti, zato se jih danes ne uporablja več, a vsak model je imel vsaj nekaj pomembnih konceptov, ki so spremenili način razvoja. Na začetku tisočletja je prišla v ospredje nova filozofija razvoja, imenovana agilni razvoj. Temeljila je na dinamičnem prilagajanju razvijalcev in tesnem sodelovanju z naročniki.

Vsi omenjeni modeli govorijo o načinu razvoja in odnosu med razvijalci in managerji, a nobeden od modelov ne omenja, kako na razvoj vpliva naročnik. Ta vidik se mi zdi zelo pomemben, zato ga želim raziskati, saj menim, da vpliva na izbiro modela razvoja programske opreme. Nekateri naročniki se zavedajo, da naročilo programske opreme ni kot nakupovanje v trgovini, kjer so izdelki na policah, ampak so izdelki/produkti veliko bolj kompleksni, pri čemer je potrebno sodelovanje. To seveda vpliva na končno ceno produkta, ki jo večina naročnikov želi vedeti vnaprej, čeprav še ne definirajo vseh funkcionalnosti, ki jih želijo imeti.

Cilj magistrskega dela je s pomočjo zrelostnega razvojnega modela za programsko opremo (CMM) dvigniti stopnjo obvladovanja procesa razvoja programske opreme in s tem izboljšati doseganje časovnih rokov, kakovost produkta in znižati stroške razvoja. Pri tem bomo uporabili razvojni model, ki bo najprimernejši za velikost podjetja in dinamiko razvoja v povezavi s specifikacijami programske opreme. Majhna podjetja, predvsem pa start-up podjetja, so po naravi kreativna in fleksibilna in nerada uvajajo definirane procese in formalne modele, ki lahko ovirajo njihove naravne značilnosti (Sutton, 2000), zato se je pri le-teh treba osredotočiti na prilagodljivosti procesa, saj želimo ohraniti trenutno prilagodljivost, ki jo imamo pri razvoju. Zavedamo pa se, da bo zaradi tipičnih slabosti, ki jih imajo mlada podjetja (Sutton, 2000), to delo oteženo in bo treba pri vzpostavljanju procesnega modela spremeniti način mišljenja.

Magistrsko delo bo v prvem delu vsebovalo pregled in analizo teoretičnih modelov in metodologij izboljšave razvojnega procesa. V tem delu bo poudarek predvsem na kritičnem ovrednotenju modelov, saj vsak avtor predstavlja svoj model kot najboljši. Ob tem je treba imeti tudi v mislih, da izboljšujemo proces v majhnem rastočem podjetju, kar pomeni, da bodo določeni modeli manj primerni ali pa celo neustrezni. Za ovrednotenje ustreznosti metodologije si bomo pomagali s študijami primerov (Dangle, Larsen, Shaw, & Zelkowitz, 2005), izkušnjami izbranega podjetja in lastnimi izkušnjami iz zadnjih petih let. V tem času sem namreč pridobil dovolj izkušenj iz razvoja programske opreme, da bom lahko pripomogel k izboljšavi procesa.

Izpostavili bomo tudi obdobje, v katerem je podjetje začutilo potrebo po višji stopnji definiranosti procesov in razloge zanjo. Procese bomo primerjali z drugimi podjetji in poiskali podobnosti (Laesvirta & Ribière, 2008), saj bomo lahko tako lažje našli rešitve in izboljšave.

V drugem delu se bomo lotili problema na konkretnem primeru procesa razvoja programske opreme. Analizirali bomo trenutno stanje procesa razvoja in izpostavili glavne pomanjkljivosti. Naslednji korak bo načrtovanje izboljšave na ključnih področjih procesov, ki so potrebni, da se zrelostni model razvoja programske opreme dvigne na tretjo stopnjo. Pri tem si bomo pomagali z metodologijami razvoja, ki jih bomo v prvem delu analizirali in kritično ovrednotili. Za preverjanje uspešnosti in učinkovitosti modela in metodologije bomo uporabili interni sistem, v katerem prijavljamo napake, časovni okvir razvoja in zadovoljstvo naročnika.

Model CMM ima pet stopenj, pri čemer je prva stopnja t. i. ad hoc oziroma kaotično organizirani proces, peta stopnja pa je najvišja stopnja organiziranosti procesa. Cilj je dvigniti zrelostni model procesa razvoja na tretjo stopnjo, pri čemer je potrebno zadovoljiti 13 ključnih področij procesov (angl. *Key Process Areas,* v nadaljevanju KPA) (Paulk et al., 1993).

1 RAZVOJ PROGRAMSKE OPREME KOT PROCES

1.1 Proces in metodologija razvoja programske opreme

Preden začnemo pregledovati metodologije in procese razvoja programske opreme, moramo najprej razjasniti pojme.

Proces kot pojem je lažje razumeti, saj ni tako abstrakten kot pojem metodologija. Proces je strukturiran, merjen skupek aktivnosti, namenjen proizvodnji določenega produkta ali storitve za določenega kupca ali trg. Proces se močno osredotoča na to, kako delo v organizaciji poteka (Davenport, 2013). Splošno gledano so procesi ponovljivi, in če jih je mogoče ponoviti, potem jih je mogoče tudi do določene stopnje avtomatizirati. Ravno zato

je pomembno, da so procesi zelo natančno definirani, saj jih v nasprotnem primeru ni mogoče avtomatizirati.

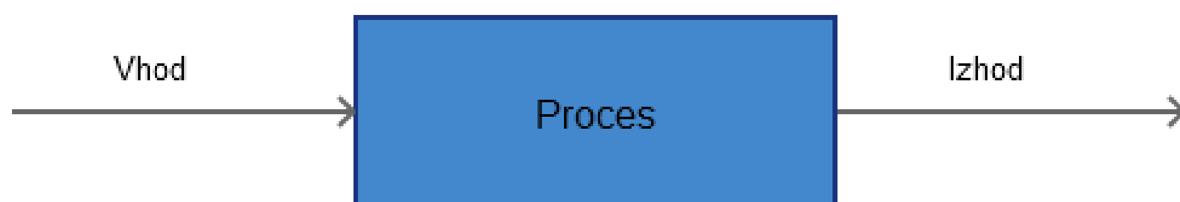
Običajno je procese, ki ustvarjajo fizični izdelek, lažje avtomatizirati, saj so drugače strukturirani kot procesi, katerih rezultat je virtualni izdelek (na primer programska oprema). Težavnost avtomatizacije procesov je zato odvisna od stopnje definiranosti in oblike strukturiranosti procesa (Tong, 1994).

Procesi so zaporedje logično povezanih korakov/aktivnosti, ki jih mora podjetje narediti, če želi doseči določen poslovni cilj (Davenport, 2013). Ta cilj je lahko produkt, storitev, razvoj produkta ali storitve, trženje, nabava ... Procesi nudijo pomoč pri načrtovanju razvoja in merjenju kakovosti proizvoda. Za enostavnejše razumevanje so procesi razvoja programske opreme predstavljeni v abstraktni obliki kot modeli procesa razvoja programske opreme (Coleman & O'Connor, 2008). Obstaja več različnih modelov, kot so Slapovni (angl. *Waterfall*), Spiralni (angl. *Spiral*), Agilni (angl. *Agile*) ...

Primarna funkcija modela procesa razvoja programske opreme je, da določi vrstni red faz, ki sodelujejo pri razvoju programske opreme, in vzpostavi prehodne kriterije za napredovanje iz ene faze v drugo. To vključuje dokončanje potrebnih zahtev za trenutno fazo in izbiro zahtev za prehod v naslednjo fazo. Model procesa programske opreme torej naslovi naslednji vprašanji (Boehm, 1988):

- Kaj bomo storili v naslednji fazi?
- Koliko časa bomo za to fazo porabili?

Slika 1: Abstraktni model procesa



Vir: I. Aaen, Software process improvement: Blueprints versus recipes, 2003.

Model procesa programske opreme je pomemben, ker nudi smernice o zaporedju aktivnosti, ki se morajo izvajati. Za vsak proces je pomembno, da je (Demmy & Petrini, 1989):

- dobro definiran,
- merljiv,
- ponovljiv in
- nadziran.

Za učinkovito in uspešno izvajanje procesa moramo razumeti namen procesa in učinke procesa. Torej moramo poznati njegove aktivnosti in imeti nadzor nad vhodnimi parametri, ki vstopajo v proces. Pomembno je, da poznamo vse elemente procesa.

Slika 1 prikazuje abstraktni model procesa, za katerega je značilno, da ima vhod in izhod. Rezultat vsakega procesa mora biti:

- diverzibilen, kar pomeni, da lahko razlikujemo med različnimi rezultati procesa,
- merljiv in
- bistven za delovanje podjetja.

V okviru managementa poslovnih procesov obstaja šest področij, ki so ključna za razvoj procesa:

1. identificiranje procesa,
2. odkrivanje procesa,
3. analiza procesa,
4. optimizacija procesa,
5. implementacija procesa in
6. spremljanje in nadzorovanje procesa.

1. Identificiranje procesa

Z identificiranjem procesa ugotovimo, kako so aktivnosti v procesu povezane in kakšno je njihovo zaporedje. Gre za opis procesa in vseh aktivnosti ter nalog zaposlenih v podjetju. Sodelovanje vseh zaposlenih je v tej fazi pomembno, saj se mora razumevanje procesa poenotiti in v razumljivi obliki zapisati trenutno stanje procesa. Identificiranje procesa torej pomeni, da natančno preučimo trenutno stanje delovanja procesa.

2. Odkrivanje procesa

Odkrivanje procesa je povezano s tehnikami odkrivanja procesov, ki lahko temeljijo na osnovi dokazov (dokumentacija, opazovanje, odkrivanje zakonitosti iz podatkov ...), intervjujev in delavnic.

3. Analiza procesa

Pri analizi procesa merimo učinkovitost trenutnega procesa, nato se na podlagi rezultatov meritev odločimo o dolgoročni strategiji izboljšave tega procesa. Rezultate primerjamo s cilji, ki so bili določeni na začetku procesa. Cilj analize je, da na podlagi pridobljenih podatkov lahko ugotovimo odstopanja od pričakovanih rezultatov in posledično pripravimo načrt, kako proces optimizirati. Vhodni parameter procesa analize procesa je

trenutni proces (»as-is«), izhodni parameter so slabosti trenutnega procesa, ki jih sprejme naslednja faza življenjskega cikla procesa.

4. Optimizacija procesa

Optimizacija procesa ali preoblikovanje procesa je postopek, pri katerem na podlagi analize procesa poskušamo odpraviti nepravilnosti in neoptimizirane elemente v procesu. Rezultat te faze je optimiziran proces (»to-be«), ki ga v naslednji fazi implementiramo.

5. Implementacija procesa

Na podlagi »to-be« modela procesa se izvede implementacija procesa v izvajalno okolje. Običajno se ta korak izvaja postopoma in se ga razdeli med različne udeležence procesa, saj se tako poveča uspešnost implementacije sprememb. Z različnimi orodji za vodenje poslovnih procesov lahko ta korak zelo poenostavimo.

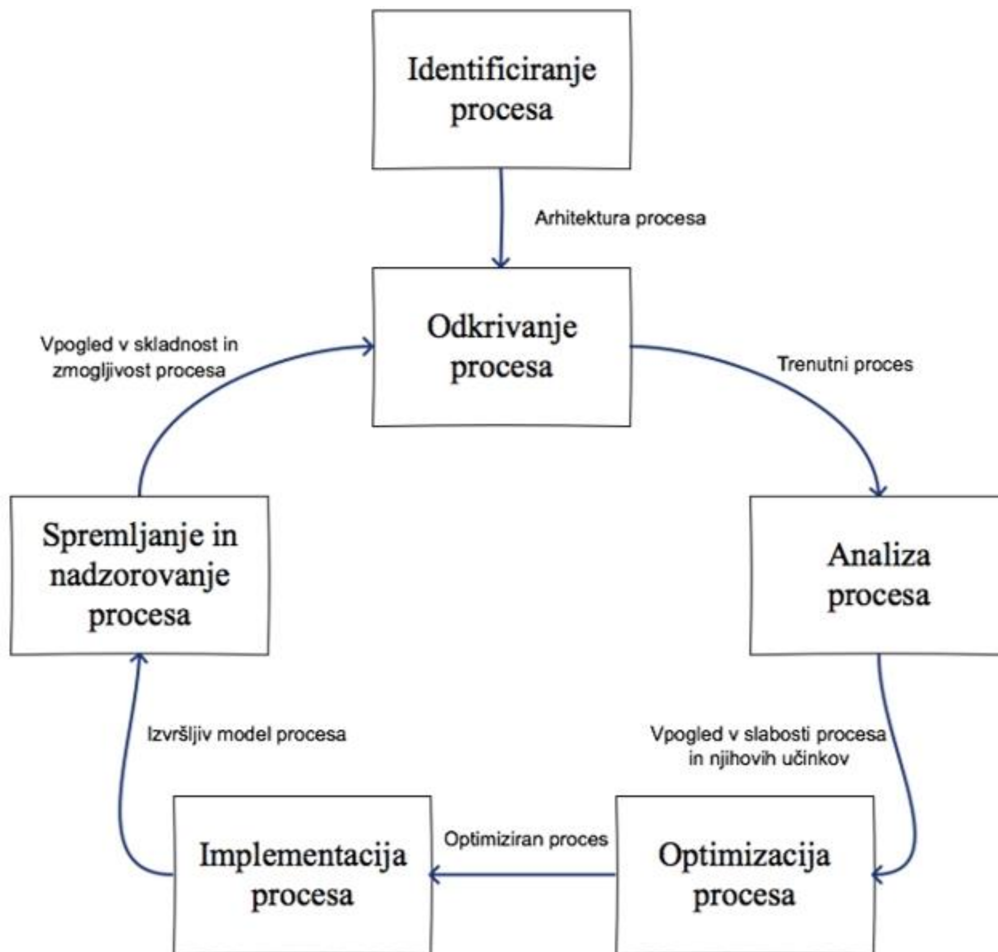
6. Spremljanje in obvladovanje procesa

Zadnja faza življenjskega cikla procesa je spremljanje in obvladovanje procesa. V tej fazi skrbimo, da pri procesu ne prihaja do napak. Vse težave in napake, ki se pojavijo, je potrebno hitro odpraviti in o njih poročati, da se preveri, kje v življenjskem ciklu procesa je prišlo do napake. S tem se zaključi življenjski cikel procesa, ki je prikazan na Sliki 2.

Za vsak proces je tudi pomembno, da je modeliran, saj se s tem poveča razumevanje procesa. Ustvari se celovita slika, kako podjetje posluje, veliko lažje se odkrijejo tudi pomanjkljivosti procesa, ki jih je nato tudi lažje odpraviti. Tudi nadaljnje izboljševanje in testiranje procesov je veliko lažje, če je proces modeliran. Pri opisovanju procesa je priporočljivo, da je razumljiv za vse udeležence procesa in ne samo za osebe, ki proces modelirajo. Vsak model procesa mora vsebovati naslednje elemente:

- začetni dogodek, ki sproži proces,
- vhodne parametre (kaj, od kod, kako … vstopi v proces),
- opis,
- rezultat in
- lastnika procesa.

Slika 2: Življenjski cikel managementa procesa



Vir: A. Guide, Project Management Body of Knowledge third edition, 2004.

Za modeliranje procesa lahko uporabimo več tehnik in diagramov, kot so: diagrami eEPC, diagrami IDEF, podatkovni tokovi in notacija BPMN (angl. *business process modelling notation*), ki je standard za opis poslovnih procesov.

Razvoj programske opreme običajno poteka pri projektih, v katerih so procesi določeni, zaradi česar imajo ti procesi tudi določene omejitve, ki so značilne za samo vodenje projekta. Običajno pri vodenju projekta obstaja trojna omejitev (angl. *tripple constraint, iron triangle*), ki se nanaša na velikost, ceno in časovni okvir projekta. Omenjeni trije dejavniki vplivajo na kakovost produkta. Velikost projekta se nanaša na definirane okvirje sistema, ki ga bomo proizvajali, torej, kaj bomo naredili. Pri ceni projekta moramo paziti, da ne presežemo razpoložljivega proračuna. Časovni okvir je časovna omejitev vsakega projekta in s tem posledično tudi vseh procesov, s katerim se določijo roki izdelave projekta.

V knjigi PMBOK (Guide, 2004) so trikotnik trojne omejitve razširili in dodali še sredstva in tveganje. V klasičnem smislu omejitvenega trikotnika so sredstva spadala v ceno

projekta, v knjigi PMBOK pa so sredstva označili kot posebno omejitev, saj se ta bolj nanaša na produkcijske faktorje v podjetju, ki jih podjetje ne more kupiti po potrebi (na primer visoko izobražene kadre, zelo specifične naprave ...).

Razvoj programske opreme ni ponovljiv v taki meri kot na primer proizvodnja fizičnih produktov, a ima aktivnosti, ki se ponavljajo, in jih lahko zato tudi nadziramo. Vsebuje tudi elemente, kot so časovni okvir projekta, velikost projekta in pričakovana kakovost produkta, ki je rezultat procesa. Pri vsakem procesu lahko enostavno merimo tri lastnosti, ki so značilne za proces razvoja programske opreme (Janieszewski & George, 2003):

- čas – potreben čas za opravljanje določene naloge,
- velikost – velikost proizvedenega proizvoda,
- napake – število napak, tip napak, čas, potreben za odpravo napak itd.

Management poslovnih procesov močno vpliva na izvajanje in vodenje projekta in neposredno vpliva na to, ali se bo projekt pravočasno zaključil, dosegel dogovorjeno in pričakovano kakovost ter ostal v okvirih proračuna.

Metodologija je dobro premišljen in opredeljen ponovljiv pristop opravljanja "nečesa" z definiranim sklopom pravil, metod, testnimi aktivnostmi, končnim rezultatom in procesi, ki tipično služijo k rešitvi določenega problema. Metodologija je v organizacijskem smislu višje od procesa in lahko vsebuje več sočasnih ali zaporednih procesov. Glavna razlika med metodologijo in procesnim modelom je, da se metodologija osredotoča na zaporedje faz in kako to zaporedje vizualno predstaviti (drevesni diagrami, diagram prehodnih stanj ...). Metodologija definira tudi način dela, tehnike in orodja, ki se bodo uporabili v procesih in aktivnostih znotraj procesov. Pomembno je tudi, da so elementi znotraj metodologije usklajeni – na primer procesi morajo upoštevati načela in filozofijo metodologije. Elementi metodologije so torej zelo prepleteni in vplivajo eden na drugega (Avison & Fitzgerald, 2003).

Pri metodologiji ne gre samo za postopek, temveč se ta nanaša na vsako aktivnost, akcijo in tudi komunikacijo. Metodologija torej opredeljuje tudi, kako poteka komunikacija med zaposlenimi, in tako posledično vpliva tudi na kulturo podjetja (Cockburn, 2006).

Primer poslovne metodologije bi bil, kako načrtno v definiranem okolju nekaj testirati, preveriti rezultate, določiti končne rezultate in ugotoviti, kako izboljšati in nadzorovati vsakodnevne aktivnosti (Howell, 2012).

1.2 Motivi vzpostavitve in izboljšave procesov

Ena izmed pomembnejših aktivnosti, ki se vršijo nad samim procesom, je, da je proces nadziran, saj se s tem zaznajo nepravilnost in neoptimizirane aktivnosti v procesu. Pri

izdelavi programske opreme tej aktivnosti pravimo izboljšava procesa izdelave programske opreme (angl. *software process improvement,* v nadaljevanju SPI).

V 80. letih je Watts Humphrey s sodelavci začel razvijati SPI in te osnovne ideje so postale temelj modela CMM (več o tem v nadaljevanju) (Paulk et al., 1993). SPI je pomemben del življenjskega cikla procesa razvoja programske opreme. Bistvo SPI je, da v proces implementira spremembe, ki vplivajo na njegovo izboljšavo (Sharma & Sharma, 2009). SPI je namenjen organizacijam, ki imajo visoko usposobljene zaposlene in z uporabo različnih metod in orodij vršijo kompleksne aktivnosti. Glavna naloga SPI strokovnjakov je sodelovanje z managerji in vodji programerjev, saj imajo ti ljudje največ znanja o procesu (Aaen, 2003).

Izboljšava procesa izdelave programske opreme se nanaša na implementirane spremembe v procesu, ki povzročijo izboljšavo. Na trgu obstaja mnogo metod za izboljšavo procesa izdelave programske opreme, ampak vse le delno naslovijo dejavnike, ki so ključni za doseganje uspeha SPI (Sharma & Sharma, 2009). Karl Wiegers razloži SPI kot dosledno uporabo postopkov, ki prinašajo dobre rezultate, in spreminjanje postopkov, ki povzročajo težave (Weissfelner, 1999). Wiegers nadaljuje, da če nekdo resnično želi izboljšati proces, mora odkrito in kritično analizirati svoje delo in delo podjetja, saj drugače ne bo odkril pravih dejavnikov, ki jih je treba spremeniti. Tipični koraki SPI so naslednji (Sharma & Sharma, 2009):

- Preučiti trenutne tehnike ter očitne prednosti in slabosti.
- Nuditi smernice pri nastajajočih tehnikah in postopkih.
- Poročati o ugotovitvah, ki so uporabne za razvijalce in druge udeležence SPI.

Pri implementaciji SPI pride v podjetjih pogosto do odpora zaposlenih. Vzroki za odpor so različni glede na njihov položaj v podjetju. Razvijalci programske opreme po navadi navajajo vsakodnevne demotivacijske razloge, vodje projektov in managerji pa srednjeročne razloge (Baddoo & Hall, 2003). Mnogo od teh razlogov izhaja iz dolgoletnih pozitivnih in negativnih izkušenj, kot so:

- Ko se programer nauči razvijati programe, ki delujejo, si s tem izoblikuje tudi oseben pristop k procesu.
- Pogosteje ko programer uporablja ta pristop, močneje je vanj ukoreninjen.
- Predhodne slabe izkušnje uporabe novih tehnik in orodij spodbudijo v programerju dvom pri uvajanju novih metod dela.

Pri vzpostavljanju definiranih procesov managerji pogosto iščejo izgovore, zakaj se tega postopka ne bi lotili. Racionalizirajo, da se bodo izboljšav lotili, ko bodo projekti zaključeni in ko bodo imeli več časa. Tako razmišljanje ima mnogo pomanjkljivosti. Odlašanje in zanemarjanje izboljšav procesov lahko stanje poslabša. Težave, kot so slabo

načrtovanje in določanje meje projekta v povezavi z razpoložljivimi sredstvi in pomanjkanjem podpore vodstva, so vedno bremenile projekte, zato se jih ne sme uporabljati kot izgovor pri vzpostavljanju in izboljšavi procesov. Podjetja, ki sistematično implementirajo izboljšave svojih procesov, imajo uspešnejše procese. Da je izboljšava procesov dodatno delo, ki ni povezano s "pravim" delom, je pogosto zmotno razmišljanje (Yamamura, 1999).

Kljub številnim znanstvenim člankom, doktorskim nalogam, različnim raziskavam in splošnemu sprejetju procesa vzpostavitve in izboljšave razvoja programske opreme v podjetjih obstaja premalo dokazov, ki bi potrdili, da so se produkti izboljšali zaradi sprememb procesov (Kuilboer & Ashrafi, 2000; Glass, 1999). Pomanjkanje dokazov kaže na to, da SPI ne prinaša obljubljenih koristi. Pomanjkanje dokazov lahko interpretiramo tudi tako, da obstaja premalo neodvisnih analiz o učinkih SPI na kakovost produkta. Premajhna pozornost vpletenih prav tako lahko pripelje do manjše koristi SPI implementacije (Baddoo & Hall, 2003).

Delodajalec, ki daje navdih zaposlenim, je močno orodje za podjetje. Zaposleni, ki so motivirani, imajo več energije, ustvarjajo inovativne rešitve in imajo pozitiven odnos do dela. Vedno poskušajo izboljšati svoje delo in se radi javljajo za posebne zadolžitve. Managerji in zaposleni si med seboj zaupajo. Zaposleni spodbujajo vrednote in uporabo izboljšav, ko prevzamejo odgovornost za izboljšave, saj vanje verjamejo in so ponosni na dosežke, ki jih prinašajo (Yamamura, 1999).

Izboljšavo procesov lahko opredelimo kot razumevanje obstoječih procesov in spreminjanje tistih procesov, ki izboljšajo kakovost produkta ali storitve in/ali zmanjšujejo razvojne stroške in krajšajo čas razvoja (Sommerville, 1996). Izboljšava procesov je pomemben del vsakega procesa, pa naj gre za proizvodnjo, administrativno dejavnost ali razvoj programske opreme. Analiza in razčlenitev procesa sta zelo dober način, da podjetje odkrije ozka grla ali katera druga področja, ki se jih lahko bolj optimizira. S tem se izboljša obstoječi način delovanja.

Prednosti izboljšave procesa vključujejo višjo kakovost procesa kot tudi produkta. Čas, ki je potreben, da pride produkt na trg, se skrajša, poveča se učinkovitost znižanja stroškov produkta (Sommerville, 1996). Izboljšava procesa izdelave programske opreme je zato zelo pomembna za uspeh podjetja, saj pomaga izboljšati kakovost in učinkovitost tako procesa kot tudi produkta (programsko opremo).

1.3 Agilni proces

Tveganje predstavlja eno izmed večjih težav pri razvoju programske opreme. Tradicionalni razvojni procesi zelo slabo rešujejo tveganja (Beck, 2000), kot so:

- Prekoračeni roki. Programska oprema ni dokončana v dogovorjenem času.
- Odpoved projekta. Po prevelikem številu prekoračenih rokov se projekt odpove.
- Sistem odpove. Programska oprema postane po nekaj letih predraga za vzdrževanje in nadgrajevanje, zato jo je potrebno zamenjati.
- Pogostost napak. Programska oprema se ne uporablja, ker prepogosto prihaja do napak.
- Napačno razumevanje poslovnega problema. Programska oprema rešuje napačen problem.
- Sprememba poslovanja. Poslovni problem, ki ga rešuje programska oprema, je spremenjen.
- Neuporabne napredne lastnosti. Programska oprema vsebuje mnogo naprednih lastnosti, ki so bile zanimive za programiranje, a so za uporabnika neuporabne.
- Fluktuacija osebja. Programerji zasovražijo sistem in zapustijo projekt.

Agilni procesi imajo odgovore na zgornje težave in so zato zelo prepričljivi za uporabo. Agilni procesi zgornja tveganja rešujejo na naslednje načine (Beck, 2000):

- Prekoračeni roki. Krajši razvojni cikli in pogoste objave programske opreme.
- Odpoved projekta. Stranka izbere najmanjšo možno verzijo programske opreme, ki ji nudi največjo dodano vrednost. Tako se lahko v največji meri izogne napakam, obenem pa je vrednost programske opreme maksimalna.
- Sistem odpove. Po vsaki spremembi programske opreme se zaženejo obsežni testi kode, ki preverjajo delovanje programske opreme.
- Pogostost napak. Narejeni so testi, ki preverjajo funkcijsko delovanje (angl. *function tests*) kot tudi celovito delovanje (angl. *behaviour tests*).
- Napačno razumevanje poslovnega problema. Stranka je pomemben član razvojne ekipe. Specifikacije se nenehno izpopolnjujejo.
- Sprememba poslovanja. Zaradi krajših razvojnih ciklov in pogostejših objav programske opreme lahko stranka sporoči spremembo v poslovanju, ki vpliva na razvoj programske opreme.
- Neuporabne napredne lastnosti. Razvijajo se samo elementi z najvišjo prioriteto.
- Fluktuacija osebja. Programerji sodelujejo pri ocenjevanju zahtevnosti razvoja programske opreme in prevzamejo odgovornost za dokončanje svojega dela. Spodbuja se medosebna interakcija.

Agilni proces je relativno nov pristop razvoja programske opreme, ki vsebuje sklop principov, za katere se je prvotno zavzela skupina 17 programerjev, danes pa jih prakticirajo mnoga podjetja po vsem svetu. Glavna načela, ki jih zagovarjajo in ki so tudi vodila do nastanka agilne razvojne filozofije, so zasnovana na dolgoletnih uspešnih in neuspešnih izkušnjah z mnogimi projekti, na podlagi katerih so ugotovili, kateri dejavniki v procesu so pomembni in pripomorejo k boljšemu rezultatu (Misra et al., 2012). Čeprav je

imel vsak od programerjev svoj pristop pri razvoju programske opreme, so vsi, v nasprotju s klasičnim izoliranim razvojem, zagovarjali tesno sodelovanje med razvijalci programske opreme in naročnikom, kar pomeni, da so procesi veliko bolj prilagodljivi in dinamični. To je predvsem zelo pomembno za manjša podjetja, saj je to ena izmed njihovih konkurenčnih prednosti (Richardson & Von Wangenheim, 2007).

Glavna načela agilnih procesov so (Fowler & Highsmith, 2001):

- Večji poudarek je na osebni komunikaciji kot na pisnih dokumentih.
- Bolje je pogosteje objavljati nadgradnje programske opreme v kratkih razvojnih ciklih kot končno dostavljen projekt z vsemi prvotno dogovorjenimi zahtevami.
- Sprejemanje spreminjanja specifikacij na zahtevo naročnika.
- Prilagajanje organizacijskih zmogljivosti razvojne skupine glede na spreminjanje poslovnih zahtev.

Izčrpne razprave o agilnih procesih je mogoče najti v številni literaturi, na primer Abrahasson (2002), Ambler (2005), Reifer (2002), Cockburn and Highsmith (2001) ... Agilni procesi močno poudarjajo svobodo razvojne skupine in spodbujajo, da so v središču procesa sposobni in visoko motivirani posamezniki, ki jim je treba zagotoviti svobodo in fleksibilnost pri ustvarjanju, saj bodo tako najučinkovitejši in inovativni. Agilna filozofija zagovarja, da je potrebno proces implementirati okoli motiviranih posameznikov, ki jim je treba nuditi podporo in tako zagotoviti uspeh projekta. Pri agilnih procesih je zelo pomemben ustaljen in konstanten delovni čas, saj neodgovorno podaljševanje delovnika vpliva na zdravje razvijalcev. Primer: Agilna filozofija razvijalce spodbuja k 8-urnemu delovniku, saj z dodatnimi nadurami pri določenih težavah zmanjšujejo učinkovitost in znižujejo produktivnost pri prihajajočih aktivnostih v samem procesu (Misra et al., 2012).

V nasprotju z modelom CMM, o katerem bomo več govorili kasneje in ki je bolj osredotočen na težave managementa, ki je povezan z implementacijo učinkovitega procesa in sistematičnim izboljševanjem tega procesa, agilna metodologija bolj poudarja specifične aktivnosti, ki so povezane z majhnimi razvojnimi ekipami s hitrimi razvojnimi cikli (Nisse, 2000). Pomembno se je tudi zavedati, da je model CMM v organizacijskem smislu na višji stopnji, kot so agilni procesi, kar pomeni, da se model CMM in agilni procesi ne izključujejo.

Čeprav so agilno filozofijo mnogi strokovnjaki za programsko opremo sprejeli že v samem začetku, so se pojavile številne kritike s strani skupnosti razvoja programske opreme. Uspešnost predhodnih projektov, pri katerih so uporabljali agilne metode, je močno vplivala na sprejetje agilnih načel v industriji, še posebej v večjih podjetjih. Mnogo projektov poskuša v svojem razvoju uporabljati agilne metode, prav tako poročajo o zgodbah o uspehu (Misra et al., 2012). V literaturi nismo našli nobenega konkretnega primera neuspešne uporabe agilne metodologije, a vseeno je treba opozoriti, da čeprav ni

poročanj o neuspehih, obstajajo poročanja o omejitvah in naukih (angl. *lessions learned*) uporabe agilne filozofije.

Glavni očitek agilnim procesom za povečanje zrelosti razvojnih procesov je, da se komaj dotaknejo upravljalnih in organizacijskih težav, ki jih model CMM poudarja (Paulk, 2001).

Spodaj je naveden povzetek še nekaterih drugih kritik, ki se pojavljajo v literaturi.

- Agilni procesi so razumljeni zelo populistično in napačno (Irons, 2006).
- Vanje je težko je vključiti prave ljudi (Irons, 2006).
- Lahko povzročijo slabšo kakovost zaradi pomanjkanja strogosti (Irons, 2006).
- Omejena podpora pri podizvajalcih (Turk, France, & Rumpe, 2014).
- Omejena podpora pri razvoju z večjimi ekipami (Turk et al., 2014).
- Omejena podpora za razvoj velikih kompleksnih projektov (Mahanti, 2006).
- Ne deluje pri razvoju ekstremno zanesljivih in varnih sistemov (Lindvall et al., 2002).
- Najbolje deluje samo pri aplikacijah, ki se jih hitro razvije in niso zelo kompleksne (Lindvall et al., 2002).

2 PROCESNI VIDIKI RAZVOJA PROGRAMSKE OPREME

V tem poglavju bomo predstavili določene bolj razširjene modele razvoja programske opreme. Pomembno se je zavedati, da so določeni modeli bolj predvidljivi (na primer slapovni model), določeni pa bolj prilagodljivi (na primer ekstremno programiranje), a noben model ni popoln. Že s preprostim ovrednotenjem lahko hitro ugotovimo prednosti in slabosti teh modelov, zato mnoga podjetja pri projektih uporabljajo različne modele, saj so nekateri primernejši za določene tipe projektov kot drugi (Thummadi, Shiv, & Lyytinen, 2011).

Modele razvoja programske opreme lahko razdelimo v dve kategoriji (Kumar & Bhatia, 2014): tradicionalni modeli in sodobni modeli. Med tradicionalne modele spadajo:

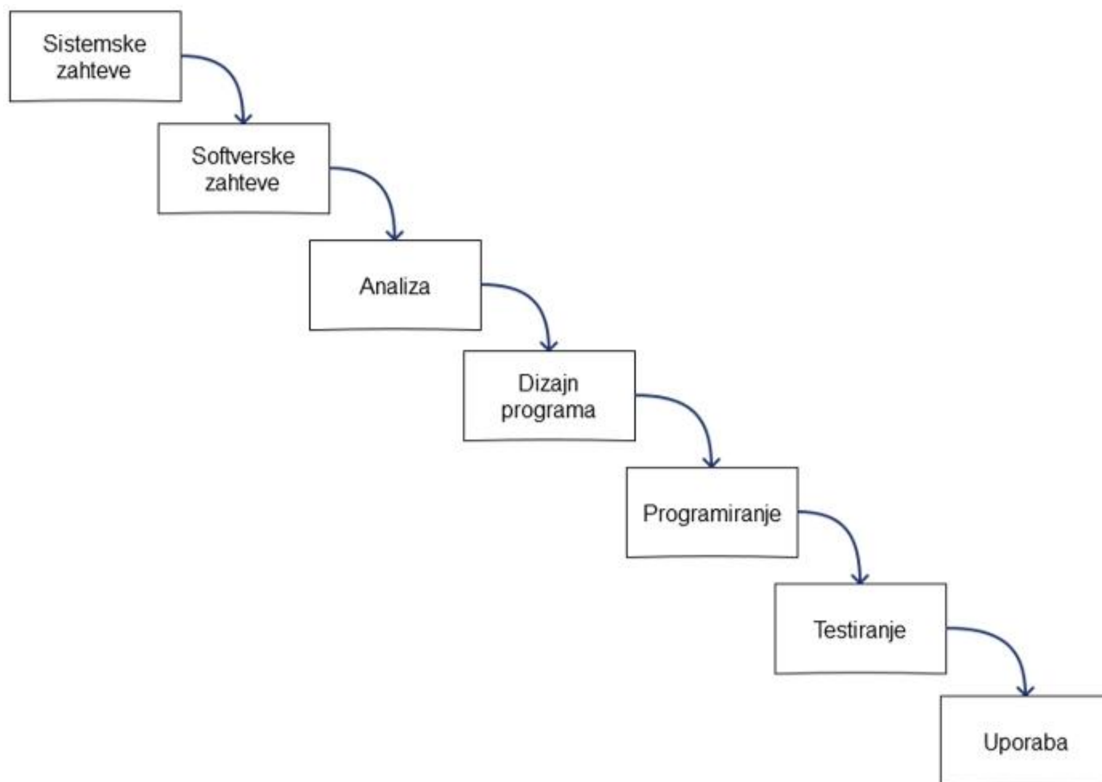
- slapovni model,
- spiralni model,
- transformacijski model,
- V-Model,
- programiraj in popravljaj.

Tipična predstavnika sodobnih razvojnih modelov sta ekstremno programiranje in model SCRUM.

2.1 Slapovni model

V sredini 60-ih so bili kot odgovor na neučinkovit način razvijanja programske opreme s stališča kakovosti, stroškov in časa prvič uvedeni modeli za razvoj programske opreme (Sommerville, 1996). Prvi primer tovrstnega modela je identificiral in predlagal Royce (1970), ta je temeljil na sistemu inženirskih načel (Royce, 1970). Model je bil kasneje znan kot tradicionalni oziroma slapovni model (angl. *Waterfall*). Slapovni model je tipično enosmerni model, pri katerem si aktivnosti sledijo ena za drugo. Model je zaradi nezmožnosti dostave stroškovno učinkovite in uporabniku prijazne programske opreme prejel mnogo kritik. Ravno to je bil povod, da so se kasneje razvili drugi modeli, ki so dali večji poudarek iterativnemu pristopu razvoja programske opreme (Boehm & Turner, 2003).

Slika 3: Prikaz faz, ki sledijo v prvotnem slapovnem modelu



Vir: W. W. Royce, Managing the development of large software systems, 1970.

2.1.1 Struktura slapovnega modela

Tradicionalna slapovna struktura naj bi vsebovala (Thummadi et al., 2011) objektno usmerjeno modeliranje podatkov, primere uporabe (angl. *use case scenario*) in arhitekturo programske opreme z uporabo objektno usmerjenega načrtovanja. Slika 3 prikazuje prvotni slapovni model, ki ga je Royce v svojem članku prvič zapisal leta 1970 (Royce, 1970).

Večina akademskih del se kot na očeta slapovnega modela sklicuje na Roycea in njegov članek Managing the development of large software systems, čeprav v samem članku Royce ne uporabi poimenovanja slapovni model. Prva, ki sta se v svojem članku sklicevala na Royceov model in uporabila izraz slapovni razvoj, sta bila Bell in Thayer (Bell & Thayer, 1976).

Royce (1970) v svojem delu identificira vzorec programiranja in predstavi njegove prednosti ter slabosti. Pravi, da sam verjame v proces, ki ga opisuje, a poudarja, da obstajajo velika tveganja pri njegovi implementaciji. Royce ne definira slapovnega modela, ampak ga identificira in prikaže njegove prednosti ter slabosti. Kot glavno slabost izpostavi dejstvo, da je faza testiranja na koncu razvojnega procesa. To tezo podkrepi z argumentacijo, da bo v veliki večini primerov, če se bodo težave odkrile v fazi testiranja, potrebna zelo obsežna prenova programske opreme. V tej točki predlaga spremembe slapovnega modela, ki so agilne narave (čeprav takrat agilna filozofija in agilni procesi še niso obstajali).

Winston Royce (1970) je predlagal 5 sprememb slapovnega modela:

- Najprej načrtuj programsko opremo.
- Načrtovanje programske opreme dokumentiraj.
- Naredi dvakrat.
- Načrtuj, kontroliraj in nadziraj testiranje.
- Vključi stranko.

Za prva dva predloga lahko rečemo, da sta zelo tradicionalna in da vsebujeta več dokumentacije. Ostali trije predlogi pa so precej podobni agilnim načelom (Fowler & Highsmith, 2001).

Naredi dvakrat – Royce predlaga, da če podjetje za stranko razvije določeno prvo programsko opremo, naj zanjo razvije tudi novo (drugo) verzijo te programske opreme, saj s tem izločimo napake, ki smo jih naredili v prvi verziji programske opreme. Prva verzija te programske opreme naj služi kot izoliran laboratorij, v katerem preizkušamo in testiramo funkcionalnosti.

Načrtuj, kontroliraj in nadziraj testiranje – V tej fazi ponovno navaja, da je slabo, če testiranje pride na koncu. Royce ne predlaga, da bi fazo testiranja prestavili na začetek razvoja, ampak močno poudarja pomembnost testiranja za uspeh procesa izdelave programske opreme. Pravi tudi, da bi morala programsko kodo pregledati tretja oseba, ki ni bila vključena v razvoj, saj se tako najlažje odkrije mnogo napak. To je tudi eno izmed glavnih načel agilne filozofije (Fowler & Highsmith, 2001).

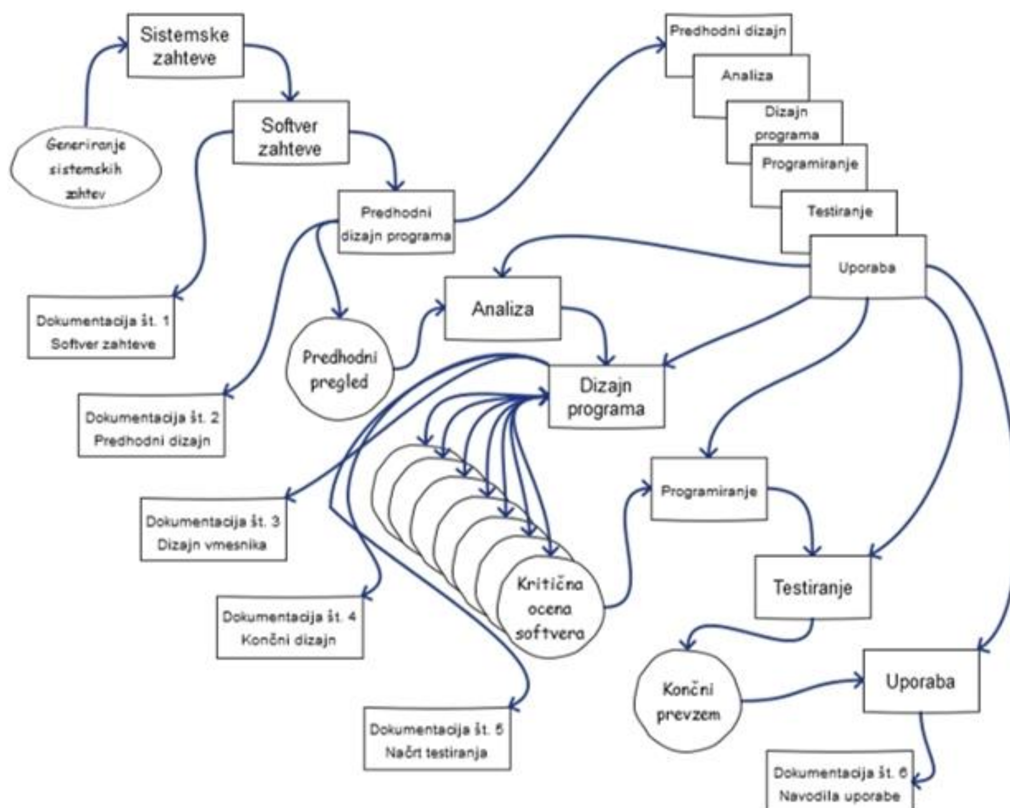
Vključi stranko – Eno izmed ključnih načel agilnega programiranja je, da je treba v proces razvoja programske opreme aktivno vključiti stranko. Royce je izključitev stranke izpostavil kot slabost takratnega razvoja in predlagal, da se ta aktivnost vključi v slapovni model. Pravi, da izključitev stranke iz procesa razvoja programske opreme vodi k težavam.

Slika 4 prikazuje Royceove predlagane izboljšave identificiranega modela (Slika 3), kasneje imenovanega slapovni model.

2.1.2 Kritike slapovnega modela

V tem delu bomo pregledali ostale kritike slapovnega modela, ki ga je identificiral Royce (Slika 3), in ne njegove verzije slapovnega modela, ki že vključuje določene izboljšave.

Slika 4: Predlagane izboljšave prvotnega slapovnega modela



Vir: W. W. Royce, Managing the development of large software systems, 1970.

Ena izmed glavnih težav slapovnega modela je pristop k izbiri tehnologije in orodij izdelave programske opreme (Cioch, Brabbs, & Kanter, 1994). Sem spadajo orodja za nadzor nad verzijami programske opreme (angl. *revision control*), orodje za nadzor in prijavo napak (angl. *bug tracking*), izbira programskega jezika, izbira programskega ogrodja (angl. *framework*), izbira knjižnic (angl. *library*) … Težava je v tem, da se pri razvoju programske opreme tehnologije zelo hitro spreminjajo, na kar razvijalci v podjetju

nimajo vpliva. V obdobju razvoja lahko pride do spremembe tehnologije in orodij, ki so bili izbrani na začetku projekta, ker so bili »najboljši«, a kasneje postanejo zastareli. Če bi proces izbire orodij potekal na koncu projekta, bi bil njihov izbor verjetno drugačen, ampak to seveda ni mogoče, še posebej ne pri uporabi slapovnega modela.
Druga večja težava pri slapovnem modelu je velika količina dokumentacije, ki je potrebna pri vsaki aktivnosti. Za razvoj večjih sistemov, še posebej tistih, pri katerih je bistveno, da so stabilni (na primer programska oprema v vesoljskih postajah), je dokumentiran način razvoja ključen tako za razvoj kot tudi za naročnika (Boehm, 1988), pri manjših projektih pa lahko ravno pretirana dokumentacija zavira celoten projekt.

Glavna težava, ki jo je izpostavil tudi Royce (1970), je neodzivnost na spremembe. Stranke pogosto spreminjajo zahteve, pri čemer je slapovni model zelo neprilagodljiv. Tudi če se spremembe na željo naročnika sprejmejo, je treba spremeniti velik del dokumentacije, ki je bila narejena na podlagi prvotnih zahtev.

2.2 Spiralni model

Leta 1987 je na pobudo ameriškega ministrstva za obrambo (angl. *Department of defense – DoD*) projektna skupina, zadolžena za nadzor razvoja programske opreme za vojsko, izdala poročilo, v katerem je poudarila glavne pomisleke o takratnem načinu razvoja programske opreme (Brooks, 1987). Ti pomisleki so se nanašali predvsem na slapovni model, omenjeni pa so bili tudi drugi, manj razširjeni modeli.

Odgovor na težave, ki so jih imeli ostali modeli, predvsem slapovni model, je bil spiralni model. Leta 1988 je Boehm predstavil spiralni model (Boehm, 1988), ki naj bi spremenil pristop k razvoju programske opreme. V nasprotju s slapovnim modelom, ki temelji na natančnem in obširnem dokumentiranju vseh korakov (angl. *document-driven* approach), spiralni model temelji na obvladovanju tveganja pri razvijanju programske opreme (angl. *risk-driven approach*) in nudi novo ogrodje za vodenje procesa razvoja programske opreme.

2.2.1 Struktura spiralnega modela

Spiralni model procesa razvoja programske opreme (Slika 5) se je razvijal mnogo let na podlagi izkušenj uporabe slapovnega modela, predvsem v povezavi z razvojem velikih sistemov, katerih naročnik je bila ameriška vlada (Boehm, 1988).

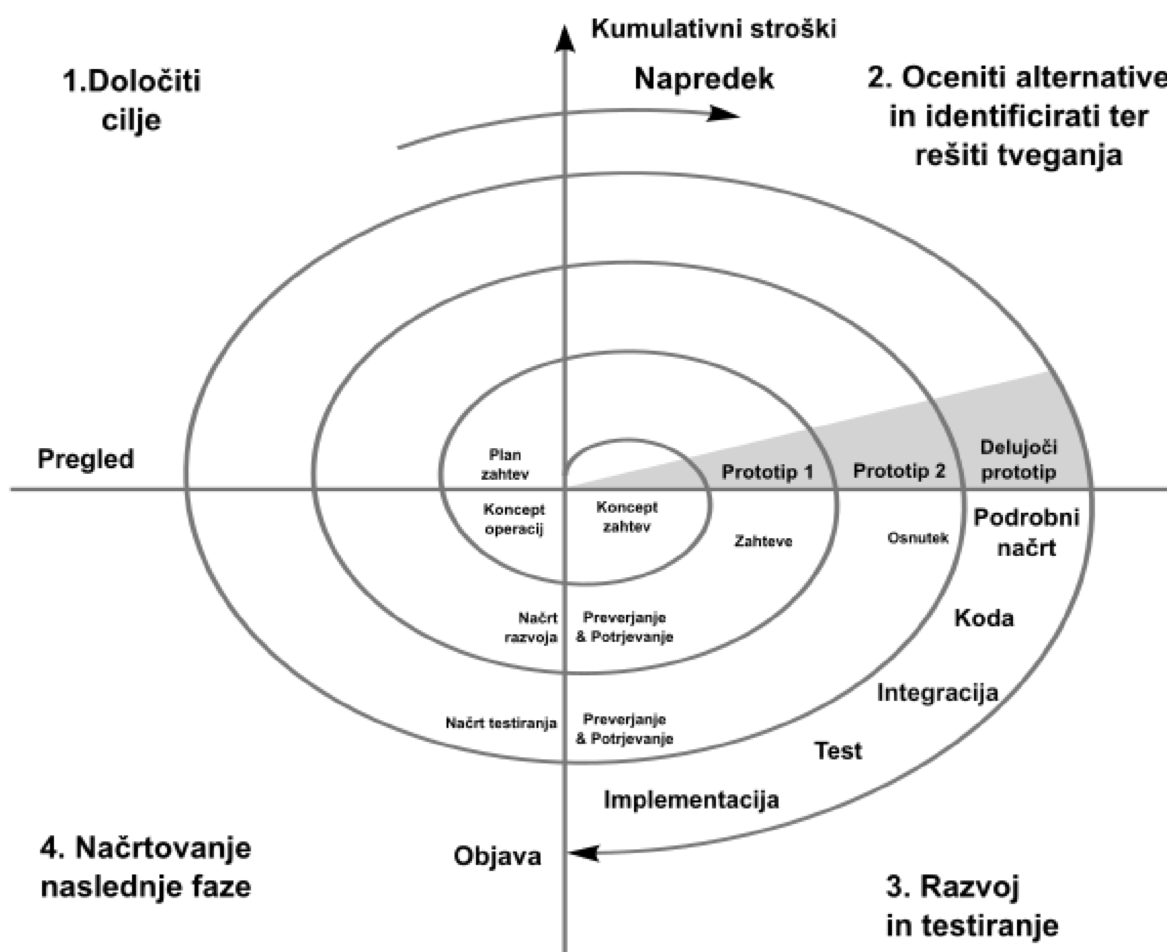
Tipični cikel spiralnega modela je sestavljen iz naslednjih faz (Slika 5):

- Določiti cilje, omejitve in alternative.
- Oceniti alternative in identificirati ter rešiti tveganja.
- Razvoj in testiranje.

- Načrtovanje naslednje faze.

V prvi fazi je treba izdelati seznam funkcionalnosti, zmogljivosti, v kolikšni meri se bo razvoj prilagajal spremembam itd. Pripravi se tudi več alternativnih načrtov programske opreme, ki se jo izdeluje, pregleda se trg in oceni, ali bi bilo katere od komponent smiselno kupiti ali za določene elemente uporabiti zunanji razvoj. Določene dele kode se lahko uporabi tudi iz prejšnjih projektov, zato se preveri, kolikšen del kode se lahko ponovno uporabi in kolikšen del kode bo lahko uporaben za naslednje projekte. Na koncu prve faze je potrebno pregledati omejitve ciljev, ki smo si jih zastavili (omejitve funkcionalnosti, zmogljivosti, časovna kompleksnost, stroški ...).

Slika 5: Spiralni model procesa razvoja programske opreme



Vir: B. W. Boehm, A spiral model of software development and enhancment, 1988.

Naslednja faza je ocena omejitve alternativ v povezavi s cilji in njihovimi omejitvami, ki smo jih določili v prvi fazi. Če se na primer odločimo, da bomo določene elemente programske opreme kupili, se moramo zavedati, da teh delov ne bomo mogli spreminjati in se jim bomo zato morali prilagajati. V tej fazi se prav tako identificirajo področja

negotovosti, ki so pri razvoju programske opreme velik vir tveganja. Pripravi se stroškovno učinkovita strategija za obvladovanje teh tveganj. Ta običajno vključuje izdelavo prototipa programske opreme in različne vrste simulacij ter primerjalnih testov (angl. *benchmarking*). Če se kakšno tveganje uresniči, se pregledajo alternative, ki so bile določene v prvi fazi in kritično ovrednotene v drugi fazi. V primeru, da se uporabijo alternative, se ponovi postopek identifikacije tveganj in ponovno naredi prototip programske opreme, na katerem se opravijo simulacije in primerjalni testi.

Tretja faza vključuje programiranje in testiranje programske opreme. Na podlagi prototipa, izdelanega v drugi fazi, se naredi operativni prototip, na katerem se opravijo obsežne simulacije in primerjalni testi. Nato se naredi podroben načrt, ki se ga uporabi pri programiranju programske opreme. Sledita testiranje in implementacija programske opreme. Testiranje vključuje testiranje posameznih enot programske opreme (angl. *unit test*), testiranje delovanja programske opreme kot celote (angl. *functional test*) in test sprejemljivosti za naročnika (angl. *user acceptance test*).

V zadnji fazi naročnik (naročnik je lahko tudi podjetje, ki izvaja razvoj te programske opreme) oceni rezultat projekta, pripravi se načrt razvoja za naslednji cikel razvoja programske opreme. V četrti fazi veliko bolje razumemo potrebe naročnika ter zmogljivosti orodij in tehnik, ki jih uporabljamo. Razumemo tudi njihove omejitve, zato se jim lahko prilagodimo. V določenih primerih se lahko oceni, da je treba spremeniti uporabo orodij in/ali tehnik razvoja programske opreme, kar pomeni, da je treba uporabiti alternative, ki smo jih definirali v prvem koraku in ovrednotili v drugem koraku. Do te faze razvoja zberemo veliko informacij, ki jih uporabimo pri načrtovanju naslednjega cikla spiralnega procesa.

2.2.2 Kritike spiralnega modela

Kljub naštetim prednostim ima tudi spiralni model slabosti. Ena izmed njih je, da se težko ugotovi izvor razdeljenih ciljev, omejitev in alternativ. Leta 1998 so Boehm, Egyed, Kwan, Port, Shah in Madachy (1998) predstavili izboljšavo spiralnega modela, ki so jo poimenovali spiralni model »WinWin«. Glavna sprememba je bila naslovljena ravno na omenjeno težavo. Kot rešitev so k vsakemu ciklu pred prvo fazo dodali tri faze:

- identificirati deležnike,
- identificirati cilje deležnikov,
- uskladiti cilje.

Kumar je izpostavil naslednje težave spiralnega modela (Kumar & Bhatia, 2014):

- Pomanjkanje eksplicitnih smernic razvoja programske opreme.
- Manjša uporabnost pri manjših projektih, saj imajo ta v večini manj tveganj pri

razvoju.
- Zanaša se na dobro ocenjevanje tveganj.
- Zaradi prototipnega pristopa se lahko poveča število ciklov, kar povzroči dražji razvoj, kot je bil predviden.
- Nudi izjemno veliko prilagodljivost, ki je za večino projektov nepotrebna.

Pri ocenjevanju in ovrednotenju projekta je ključno, da so prisotni izkušeni strokovnjaki na področju razvoja programske opreme, saj spiralni model temelji na obvladovanju tveganj pri razvoju programske opreme. To pomeni, da če so tveganja slabo ocenjena, obstaja velika verjetnost, da se bo spiralni cikel ponovil, ker prototip, izdelan v tretji fazi, ne bo ustrezal zahtevam.

2.3 Ekstremno programiranje

S hitro gospodarsko rastjo se je povečala tudi potreba po razvoju kompleksnejših programskih rešitev. To je predvsem vidno v prvih fazah razvoja programske opreme, v katerih se določajo zahteve (Li-li, Lian-feng, & Qin-ying, 2011). Slabo definirane zahteve vplivajo na vse nadaljnje faze razvoja programske opreme, zato je treba te zahteve zelo natančno opredeliti ali imeti razvojni model, ki je dovzeten za spreminjanje zahtev v procesu razvoja. Tradicionalni modeli se v času nastajanja s tem problemom niso srečevali, saj so bile takrat zahteve za programsko opremo veliko enostavnejše kot danes. Kot rešitev za tovrstne težave so prišli v ospredje agilni modeli. Med najbolj razširjene agilne modele spada ekstremno programiranje (angl. *eXtreme Programming – XP*). Ena izmed njegovih pomembnejših lastnosti je zmožnost prilagajanja hitrim spremembam. Dinamični pristop razvoja programske opreme je zelo primeren za manjše in srednje velike razvojne ekipe, ki vsebujejo do 10 članov (Paulk, 2001).

Slapovni model je dokumentno voden pristop, spiralni model uporablja pristop ocenjevanja tveganja, ekstremno programiranje pa uporablja pristop vrednot udeležencev procesa (angl. *value-driven approcah*). Ekstremno programiranje v samem jedru vključuje štiri osnovne aktivnosti tradicionalnega slapovnega modela: analizo, načrtovanje, kodiranje in testiranje. Te aktivnosti so tesno povezane s temeljnimi vrednotami ekstremnega programiranja (Paulk, 2001):

- Dobra komunikacija z naročnikom in znotraj razvojne ekipe.
- Pri reševanju problemov uporabiti enostavnejšo rešitev.
- Visoka stopnja povratnih informacij z uporabo testiranja osnovnih enot programske opreme (angl. *unit testing*) in testiranja uporabe programske opreme (angl. *functional testing*).
- Pogum in proaktivno soočanje s težavami.

Pomemben del ekstremnega programiranja je programiranje v parih. Programiranje v parih izboljša komunikacijo med člani razvojne ekipe in omogoča neprestan način pregledovanja in ovrednotenja kode. To je pomembno predvsem zato, ker iskanje in odpravljanje napak naraščata eksponentno s časom razvoja programske opreme. Programiranje v parih ustvarja tudi pozitivno motivacijo in pomaga doseči višjo kakovost programske opreme (Succi, Stefanovic, & Pedrycz, 2001).

2.3.1 Struktura ekstremnega programiranja

Razvojni proces, ki uporablja ekstremno programiranje, daje prednosti funkcionalnostim, ki so najpomembnejše za naročnika, jih implementira v najkrajšem možnem času in jih predloži naročniku za pregled. Pomembno je, da je ta cikel čim krajši, saj imajo v primeru, če pride do sprememb zahtev, le-te najmanjši možni vpliv na razvoj celotne programske opreme. S tem pristopom se tudi izboljša razumevanje želene programske opreme s strani naročnika, saj ta v preverjanje nenehno dobiva verzije programske opreme, kar je v nasprotju s tradicionalnim pristopom, kjer je naročnik na koncu razvojnega procesa prejel delujočo verzijo programske opreme, ki je lahko delovala povsem drugače, kot si je to zamislil. Slika 6 prikazuje model ekstremnega programiranja.

Najbolj tipičen in razširjen sklop primerov dobre prakse vsebuje 12 osnovnih elementov (Paulk, 2001), ki jih lahko razdelimo med management, razvojno ekipo in programerje.

Primeri dobrih praks za management:

- Na strani naročnika mora biti vedno dosegljiva določena oseba, če pride do nejasnosti glede zahtev.
- Programska oprema se pri ekstremnem programiranju razvija iterativno in inkrementalno, zato mora biti ves čas jasno določeno, kaj obsega naslednja iteracija, katere so prioritete in časovne ocene.
- Potrebno je objavljati majhne spremembe programske opreme v kratkih časovnih ciklih (dva tedna).

Primeri dobrih praks za razvojno ekipo:

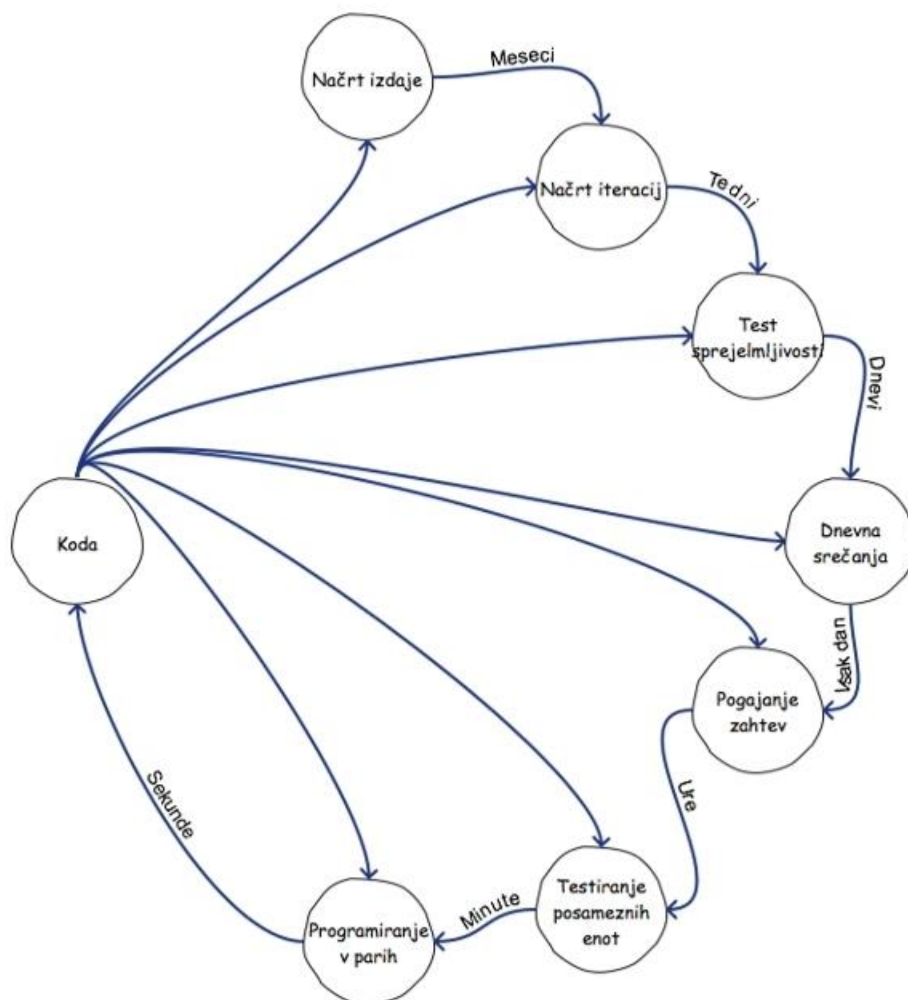
- Vsem razvijalcem mora biti jasno osnovno delovanje programske opreme, ki jo razvijajo.
- Vsakdo lahko popravi katerikoli del kode, ki se mu zdi potreben izboljšave.
- Vse spremembe in posodobitve programske opreme so takoj implementirane in testirane.
- Med razvijalci mora obstajati standard programiranja, ki se ga morajo držati vsi člani razvojne skupine. Na primer PEP 0008 (Van Rossum, Warsaw, & Coghlan, 2001).

- Ekstremno programiranje temelji na kreativnosti posameznih članov razvojne skupine, zato je pomemben 40-urni delavnik. Kreativnosti ni mogoče doseči, če so člani razvojne ekipe izčrpani.

Primeri dobrih praks za programerje:

- Vsak del kode je treba testirati.
- Sistem je treba zasnovati enostavno, saj ga je tako lažje razumeti, popravljati, testirati in vzdrževati.
- Refaktoriranje kode je potrebno opraviti takoj, ko je to potrebno.
- Programiranje v parih izboljša kvaliteto kode, znanje se hitreje prenaša med razvijalci.

Slika 6: Zanka povratnih informacij v modelu ekstremnega programiranja



Vir: K. Beck, Ekstreme programming explained: embrace change, 2000.

2.3.2 Kritike ekstremnega programiranja

Fleksibilnost ekstremnega programiranja temelji na štirih vrednotah, med katerimi sta komunikacija in povratna informacija najpomembnejši. Uspešna komunikacija je zelo pomemben dejavnik pri uspešno končanem projektu. Pri ekstremnem programiranju lahko komunikacijo razdelimo na tri dele (Li-li et al., 2011): interna komunikacija med člani razvojne ekipe, interna komunikacija med naročniki ter komunikacija med razvijalci in naročnikom. Vsi člani razvojne ekipe imajo dovolj znanja na tehničnem področju in tudi o samem projektu, da se lahko brez težav med sabo pogovarjajo in si uspešno prenašajo informacije. Tudi na strani naročnika imajo zelo jasne poslovne cilje, poslovne procese in zahteve za programsko opremo, ki so jo naročili, zato je komunikacija med njimi tudi učinkovita. Težava v komunikaciji nastane med razvijalci in naročnikom. Nekaj pogostih anomalij, ki se pojavijo v komunikaciji med naročnikom in razvijalci:

- **Nejasne zahteve.** Zaradi določenih razlogov naročniki ne morejo podati vseh zahtev ali pa so zahteve nejasne. Nejasne zahteve lahko celo povzročijo negativno spremembo v programski opremi.
- **Poplava zahtev.** Naročniki imajo prevelika pričakovanja, mislijo, da bo programska oprema rešila vse njihove težave, in upajo, da bodo razvijalci uspeli dokončati projekt v kratkem času. Naročnik ne razume tehnologije, zato slabo oceni razvoj, implementacijo in tveganja pri razvoju nove programske rešitve. Razvijalci zato ne zadovoljijo naročnikovih želja, kar vodi v nasprotovanja med naročnikom in razvijalci.
- **Pogoste spremembe.** Prepogoste spremembe zahtev vodijo v stagnacijo projekta. Spremembe končnega produkta povečajo obremenitev in podaljšajo cikel razvoja programske opreme.

Glavna kritika uporabe ekstremnega programiranja v namen izboljšave procesov (SPI) je, da daje premajhen poudarek managementu in organizacijskem delu razvoja programske opreme (Paulk, 2001). Ker ekstremno programiranje temelji na vrednotah, je tak model najtežje implementirati, saj so v podjetju potrebne velike spremembe sistema vrednot. Do težav lahko pride, če so v proces vpleteni ljudje, ki tem vrednotam nasprotujejo, zato se je treba zanašati na pozitivno vpletenost in aktivnost posameznikov v procesu.

3 ZRELOSTNI MODELI

Zrelostni modeli izvirajo s področja celovitega obvladovanja kakovosti (angl. *Total quality management*) (Cooke-Davies, 2002). Zahtevajo temeljito razumevanje trenutnega položaja organizacije in jasne dolgoročne cilje, saj so strateško povezani s procesi nenehnega izboljševanja. Zaveza za to spremembo je bistvena in potrebuje podporo ter sodelovanje vodstva (Hayes, 2014). Mnogo managerjev meni, da je njihovo znanje zadostno in da ne potrebujejo dodatnega znanja, kot so ga že pridobili z izkušnjami. To znanje aplicirajo na

procese razvoja programske opreme, od katerega je odvisno preživetje podjetja (Nisse, 2000). Do težav začne prihajati, če so trenutni procesi slabi in je njihov rezultat nepredvidljiv. Zrelostni modeli nudijo podjetju smernice, kako izboljšati procese, da postanejo predvidljivejši, nadzirani in jih je možno izboljševati. Ena izmed glavnih prednosti zrelostnih modelov je, da podjetju omogočajo prehod iz nezrelih procesov v zrelejše po korakih, saj je inkrementalne spremembe v procesu lažje aplicirati, kot vzpostaviti nov proces. Tabela 1 prikazuje tipične faze zrelostnih modelov (Paulk et al., 1993).

Večina zrelostnih modelov temelji na modelu CMM, ki je bil razvit med letoma 1986 in 1993 (Paulk et al., 1993). Od takrat se je razvilo že več kot 30 različnih modelov, ki rešujejo specifične poslovne težave (Brookes & Clark, 2009).

Tabela 1: Stopnje zrelosti

#	Stopnja zrelosti procesa	Opis
1)	Izvedljiv	Nepredvidljiv proces, ki je slabo nadzorovan.
2)	Voden	Procesi v projektu so opredeljeni, a se jih ne izvaja v celoti.
3)	Definiran	Procesi so definirani in se jih nadzorovano izvaja.
4)	Kvantitativno voden	Procesi so merljivi in nadzirani.
5)	Optimiziran	Poudarek na izboljšanju procesov.

Mnogo podjetij je porabilo veliko časa in denarja pri vzpostavljanju in izboljšavi razvojnega procesa »fizičnih« delov njihovih produktov, kot so nadzorne plošče, vezja itd. Malo podjetij pomisli, da bi bil tak pristop potreben tudi pri razvoju programske opreme. V zadnjih desetletjih je napredek v računalniški dejavnosti povzročil, da se je programska oprema postavila v ospredje razvoja produkta, saj je njegova pomembnost zelo narasla (Tong, 1994).

Produkti postajajo vse »pametnejši« (na primer pametni telefoni, računalniški sistemi v avtomobilih ...), zato so tudi uporabniki vse zahtevnejši. Večina managerjev in inženirjev ne razume specifičnih težav kakovosti programske opreme. V večini primerov je proces diagnosticiranja napak enak, a prognoza je običajno drugačna, saj je programska oprema nekoliko drugačna »vrsta« inženirstva. Drži, da sta programska oprema in strojna oprema inženirski »disciplini«, a inženirji strojne opreme imajo pri razvoju standardizirano strukturo in se morajo držati vrste pravil, katerih se razvijalcem programske opreme ni potrebno. Strojna oprema je omejena na fizični svet in mora upoštevati fizikalna pravila, medtem ko je programska oprema omejena le na posameznikovo domišljijo. Ravno zaradi te svobode prihaja do težav pri vodenju procesa razvoja programske opreme. Nek programer lahko s 100 vrsticami povzroči enak učinek, kot ga drugi programer s 50 vrsticam, zato je zelo težko meriti učinkovitost programerjev. Zaradi kratke zgodovine (programiranje je relativno novo področje) obstaja malo kazalcev kvalitete in standardov, po katerih bi lahko ocenili kakovost programske opreme (Tong, 1994).

Zrelostni modeli, kot je CMM, so v zadnjem času predstavljeni kot sredstvo, s katerim podjetja za izdelavo programske opreme privabijo več strank. Na to kaže povečano število implementacij modela CMM v organizacije (Shen & Ruan, 2008), saj podjetje, ki ima višjo stopnjo zrelostnih procesov, zagotavlja naročnikom, da bo njihov produkt pravočasno dostavljen, v okviru dogovorjenih zahtev in dogovorjenega proračuna. Vseeno se je potrebno izogibati neposredni implementaciji zrelostnega modela in raje uporabiti ustrezno asimilacijo načel in ciljev, ki so osnova teh modelov. Ravno zaradi te skrbi mnoga podjetja kažejo veliko zanimanje za agilne procese razvoja programske opreme v povezavi z zrelostnim modelom (Silva et al., 2015).

Zaradi velike potrebe po vzpostavitvi robustnega procesa in izboljšavi tega procesa se je razvilo več sodobnih modelov za reševanje teh težav. Obstajajo številne raziskave, ki kažejo na to, da uspešna implementacije procesnega modela izboljša produktivnost, kakovost, časovni načrt in poslovno vrednost. Pomembno je tudi poudariti, da obstajajo raziskave, ki kažejo na to, da veliko modelov za izboljšanje procesov ne deluje v celoti (Sharma & Sharma, 2009).

Poleg modela CMM obstaja še družina standardov ISO9000:2000, ISO9001:2000 in ISO9004:2000. Glavna kritika te družine standardov je, da vključuje preveč dokumentacije. Mnogim podjetjem je glavni namen implementacije teh standardov pridobitev certifikata zaradi trženja in ne zaradi vzpostavitve in izboljšave procesov. Načeloma lahko podjetje pridobi certifikat, če ima procese zelo dobro dokumentirane, kar pa ne pomeni, da so ti učinkoviti. Za implementacijo standarda ISO 9000 potrebuje podjetje tudi veliko sredstev (časa, denarja in truda) (Zahran, 1998).

Pri iskanju ustreznosti modelov za izboljšanje zrelosti procesov smo pregledali tudi druge, manj razširjene modele, kot so BOOTSTRAP (Evropa), SPIQ (Norveška) in ProPAM (Portugalska), a smo se odločili, da bomo zaradi razširjene uporabe in dobre dokumentacije za definiranje procesov uporabili zrelostni model zmogljivosti (CMM).

3.1 Najpogostejše težave v procesu razvoja programske opreme

Pred začetkom procesa povečanja zrelosti procesov je pomembno, da razumemo najpogostejše težave, s katerimi se srečujejo razvijalci programske opreme. Greg Tong je leta 1994 objavil članek, v katerem je izpostavil 6 glavnih problemov, s katerimi se srečujejo razvijalci programske opreme (Tong, 1994):

Analiza zahtev. Dokumentacija zahtev je nepopolna, sledljivost zahtev je »ad hoc«. Zagotavljanje kakovosti programske opreme. Zagotavljanje kakovosti bodisi ne obstaja ali pa je premalo neodvisno. Koda programske opreme in testi za to kodo niso analizirani, tehnična revizija je nezadostna, standardi programiranja ne obstajajo ali pa se jih ne uveljavlja, pomen testiranja kode običajno ni razumljen. Načrtovanje procesov in vodenje

projekta. Smernic za oceno velikosti, časovnega razporeda in stroškov običajno ni. Sledenje razvoju je premalo natančno.

Stranke so premalo vključene v proces razvoja. Management. Obljube so pogosto dane brez predhodnega posvetovanja z inženirji programske opreme. Časovni načrti so običajno nerealni. Višje vodstvo pogosto ni vpleteno v pregled stanja razvoja. Upravljanje konfiguracije. Upravljanje konfiguracije je večinoma omejeno samo na vodenje verzije programske kode (angl. *version control*). Politika uporabe knjižnic ni definirana. Nadzor nad spremembami je nepravilno implementiran. Izobraževanje. Priprava na projekte in stalno izobraževanje sta nezadostna. Managerji programske opreme običajno niso usposobljeni za vodenje projektov.

Od objave omenjenega članka je minilo že več kot 20 let, zato smo analizirali novejše raziskave in pregledali, ali so se v tem času glavni problemi razvoja programske opreme spremenili. Kaur in Sengupta (2013) sta v raziskavi, ki je trajala več let, razkrila naslednje ključne težave pri razvoju programske opreme:

- Analiza zahtev. Analiza zahtev je prva aktivnost procesa izdelave programske opreme. Že na začetku projekta so včasih cilji slabo definirani, zahteve niso jasno opredeljene, zato si morajo razvijalci sami razlagati, kako naj bi sistem deloval, kar pogosto vodi do napačno delujočega produkta.
- Pomanjkanje vključenosti. Pomanjkanje podpore managerjev in vključenosti zaposlenih sta dva največja izziva pri upravljanju IT projekta. Če zaposleni niso vključeni v projekt, ta postane z njihove strani celo osovražen.
- Velikost ekipe. Primerna velikost ekipe je bistvena pri razvoju programske opreme. Obstajajo tri velikosti ekip: majhna ekipa (do 10 ljudi), srednje velika ekipa (med 11 in 25 ljudi) in velika ekipa (nad 26 ljudi). Majhne ekipe so običajno fleksibilnejše in boljše komunicirajo. Tudi medsebojno usklajevanje je veliko lažje.
- Časovne dimenzije. Projekti z dolgim časovnim rokom običajno propadejo. Priporočljivo je skrajšati časovne roke.
- Prilagajanje spremembam. To je eno od glavnih načel agilnega razvoja (Fowler & Highsmith, 2001). Vseeno je nad spremembami treba imeti nadzor, saj preveliko število sprememb lahko prav tako vodi do neuspešnosti projektov.
- Testiranje. Glavni namen testiranja je, da se preventivno odkrijejo napake, ki se odpravijo. Pogosto se testov ne opravlja, ker projekt zamuja. Te napake se zato odkrijejo pri predaji programske opreme naročniku.
- Zagotavljanje kakovosti programske opreme. Periodično ocenjevanje kakovosti je obvezno, če želimo, da produkt ustreza zahtevam naročnika. V to ocenjevanje spada: ocena načrtovanja, pregled programske kode in različne vrste testiranja.

Kumar je izpostavil naslednje parametre, ki najbolj vplivajo na programerje in razvoj programske opreme (Kumar & Bhatia, 2014):

- aktivno sodelovanje vseh interesnih skupin razvoja programske opreme,
- resnično izpolnjevanje zahtev naročnika,
- visoka stopnja vpletenosti naročnika v proces,
- periodične predstavitve programske opreme,
- hitri odzivni čas do naročnika,
- učinkovit način merjenja zahtev programske opreme,
- koordinacija različnih načrtov dela za različne vloge v procesu razvoja,
- kvaliteta končnega produkta,
- stroški prenosa programske opreme in usposabljanja uporabnikov programske opreme,
- prilagodljivost razvoja programske opreme,
- prilagodljivo načrtovanje,
- refakturiranje programske kode,
- sposobnost in izkušenost razvojne ekipe.

3.2 Model CMM

Zrelostni model zmogljivosti za programsko opremo (angl. *Capability Maturity Model – CMM*) je ogrodje za ovrednotenje procesov, ki se uporabljajo za razvoj programske opreme. Produktivnosti in kakovosti ni mogoče izboljšati le z uporabo novejše tehnologije in orodij. Procese razvoja programske opreme je treba dobro definirati in to je glavno bistvo modela CMM. Model CMM nudi organizacijam smernice o tem, kako pridobiti nadzor nad procesi za razvoj in vzdrževanje programske opreme ter razviti kulturo organizacije, ki bo odlična pri vodenju in razvijanju programske opreme. CMM je bil zasnovan za usmerjanje organizacij pri izbiri strategije za vzpostavljanje in izboljšanje procesov z analizo trenutnega stanja zrelosti procesov in identificiranjem največjih težav pri izdelavi programske opreme. Z osredotočenjem na omenjene dejavnosti in nenehnimi izboljšavami na teh področjih si organizacija zagotovi trajne koristi v zmogljivosti procesa razvoja programske opreme (Paulk et al., 1993).

Zgodovina modela CMM (Paulk, 2009):

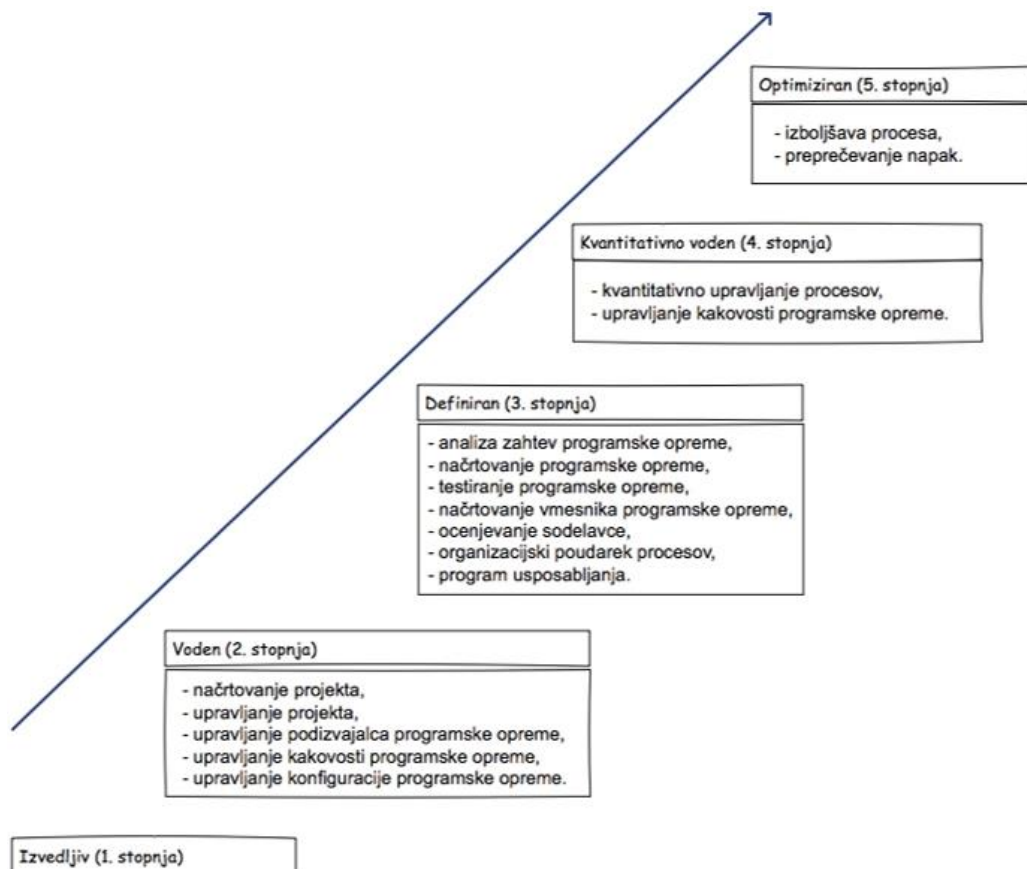
- 1990: CMM v0.2. Osnutek prve verzije modela (Slika 7).
- 1990: CMM v0.6. Dodana ključna področja.
- 1991: CMM v1.0. Slika 8 prikazuje ključna področja verzije 1.
- 1993: CMM v1.1. Manjše spremembe pri poimenovanju ključnih področij.
- 1997 CMM v2. Osnutek nove verzije modela CMM, vendar so nadaljnje delo ustavili, ker so prednost dali integraciji modela CMM.
- 2000 CMMI v1.0. Integracija zrelostnega modela zmogljivosti (angl. *Capability Maturity Model Integration*).

Eden izmed glavnih namenov modela CMM je, da pomaga organizacijam izboljšati procese razvoja programske opreme. Pri tem jim nudi smernice, po katerih se postopoma pomikajo od »ad-hoc« razvoja proti bolj nadzorovanemu procesu. Pri tem se ustvarjajo formalne in neformalne procedure, ki omogočajo ponovljivost in standardizacijo najboljših praks ter dvig zrelosti razvojnega procesa od kaotičnega do zrelega.

Za nezrele procese razvoja programske opreme je običajno značilno naslednje (Paulk, 1998):

- proces razvoja programske opreme je improviziran;
- cilji in obseg razvoja programske opreme so slabo določeni;
- roki in sredstva za izvedbo so običajno prekoračeni;
- managerji se večino časa ukvarjajo z reševanjem težav, ki nastanejo zaradi slabe organizacije;
- mehanizmi, ki bi preprečevali prekoračitev stroškov in časovnih okvirjev, niso vzpostavljeni;
- uspešnost projekta je v večini primerov odvisna od posameznikov.

Slika 7: Ključna področja modela CMM v0.2



Vir: M.C. Paulk, A history of the capability maturity model for software, 2009.

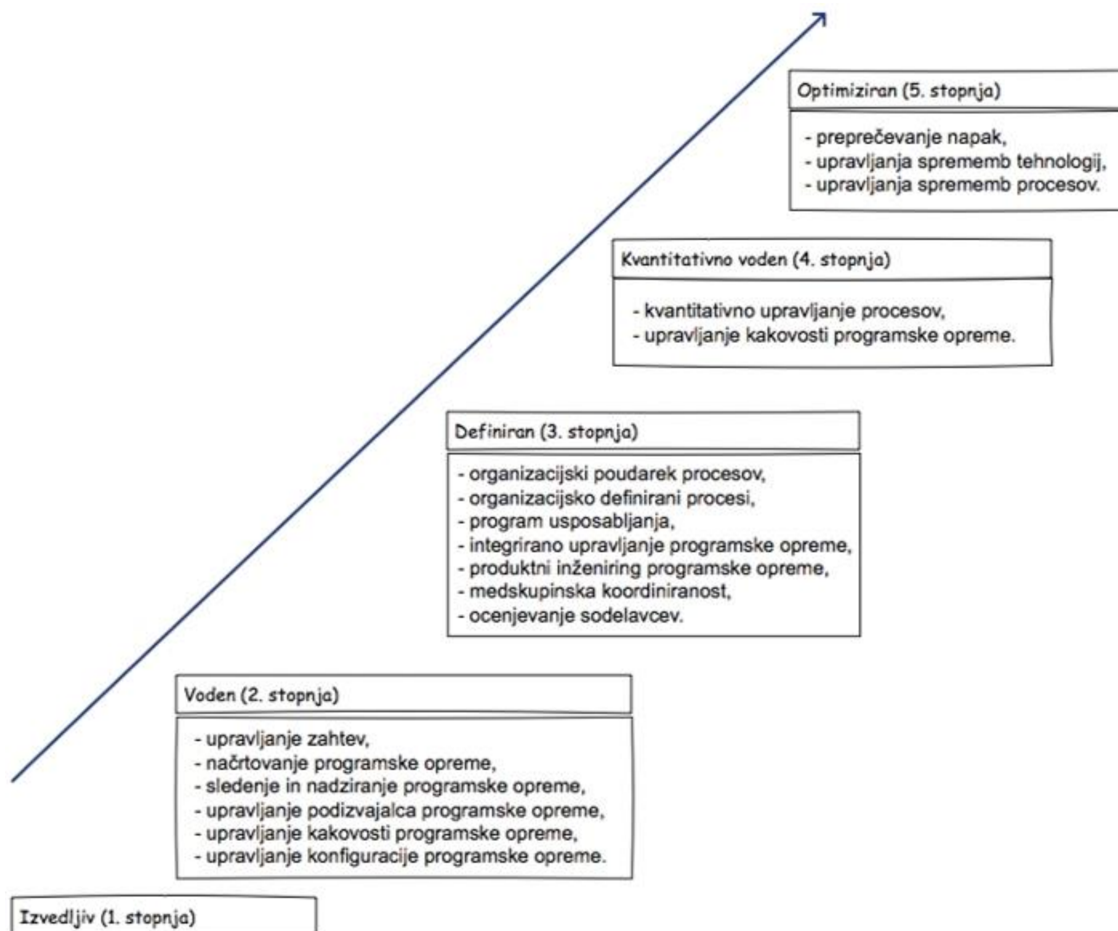
Pri implementaciji modela CMM se procesi postopno izboljšujejo in postanejo zrelejši, kar

pomeni, da (Silva et al., 2015; Sutton, 2000):

- delo poteka organizirano v skladu z definiranimi procesi razvoja programske opreme;
- udeleženci procesa razumejo celoten proces in imajo redna izobraževanja o nadaljnjem izboljševanju procesa;
- proces razvoja programske opreme je dokumentiran;
- vsak udeleženec procesa razume svojo vlogo, jasno mu je, kako si aktivnosti v procesu sledijo;

- izvedba procesa in aktivnosti v procesu so konsistentni z zahtevanimi postopki in v skladu z dokumentacijo procesa.

Slika 8: Ključna področja modela CMM v1.0



Vir: M.C. Paulk, A history of the capability maturity model for software, 2009.

Čeprav programerji in menedžerji pogosto poznajo težave pri procesu izdelave programske opreme, se mnenja o tem, katere izboljšave so najpomembnejše, pogosto razlikujejo. Ravno zaradi nesoglasja med vodstvom in strokovnim osebjem je težko doseči strategijo za

vzpostavitev ali izboljšanje procesa, zato je treba zagotoviti, da vsi udeleženci procesov sam proces enako razumejo (Aaen, 2003). Da dosežemo trajne učinke pri vzpostavljanju in izboljšavi procesov, je treba zasnovati inkrementalno pot, ki je povezana z organsko rastjo podjetja. Model CMM omogoča in tudi zahteva, da se podjetje izboljšav loti po fazah, s čimer se zagotovijo dobri temelji za naslednje stopnje izboljšave. Na tak način strategija za izboljšave, ki se črpa iz modela CMM, nudi načrt za nenehno izboljševanje procesov (Paulk et al., 1993).

Osnova modela CMM so sklopi najboljših praks, ki jih uporabljajo podjetja pri razvoju programskih rešitev in so organizirani na ključna področja procesov KPA. Kontroliranje sprememb in vodenje sledljivosti sprememb sta običajni aktivnosti projekta. Model CMM grupira take aktivnosti v sklopu upravljanja konfiguracije ključnih področij procesov, ki pa niso pri vseh projektih enaki in se lahko do določene mere razlikujejo. CMM model ovrednoti organizacijo na podlagi 18 ključnih področij procesov, glede na to, kako dobro jim uspe izpolnjevati zahteve teh področij (Dangle et al., 2005). Podjetje, ki ne izpolnjuje nobenega ključnega področja, je ocenjeno z najnižjo stopnjo, imenovano začetna stopnja (angl. *Initial*). Druga stopnja oziroma ponovljiv nivo vsebuje naslednja ključna področja procesov:

- upravljanje zahtev,
- načrtovanje programske opreme,
- sledenje in nadziranje programske opreme,
- upravljanje podizvajalca programske opreme,
- upravljanje kakovosti programske opreme,
- upravljanje konfiguracije programske opreme.

Če podjetje izpolnjuje vse zgoraj navedene kriterije, potem je to podjetje ocenjeno z drugo stopnjo modela CMM. Bistvo te stopnje je, da je proces organiziran do te mere, da se ga lahko pri podobnih projektih ponovi in s tem zagotovi uspešnejši rezultat projekta. Tretja stopnja, ki jo imenujemo definirani nivo, vsebuje dodatnih sedem ključnih področij:

- organizacijski poudarek procesov,
- organizacijsko definirani procesi,
- program usposabljanja,
- integrirano upravljanje programske opreme,
- produktni inženiring programske opreme,
- medskupinska koordiniranost,
- ocenjevanje sodelavcev.

Podjetja s tretjo stopnjo modela CMM imajo dokumentirane aktivnosti, standardizirane procese in natančno določene vhodne in izhodne parametre teh aktivnosti, ki so povezani v

celoten proces. Procesi so dobro definirani in stabilni ter veljajo za celotno organizacijo. V podjetju obstaja tudi skupina, ki je za te procese odgovorna, kar vključuje vzdrževanje in izboljševanje procesov ter skrb za izobraževanje na nivoju organizacije. Vodstvo razume pomembnost upravljanih procesov in nudi podporo osebam, ki so zadolžene za izvajanje te standardizacije.

Četrta stopnja, imenovana kvantitativno voden nivo, vsebuje naslednji ključni področji:

- kvantitativno upravljanje procesov,
- upravljanje kakovosti programske opreme.

Na četrtem nivoju ima podjetje vzpostavljene aktivnosti, ki kvantitativno in kvalitativno merijo produktivnost procesov na ravni vseh projektov v organizaciji. Zadnja, peta stopnja oziroma optimizirajoči nivo je sestavljena iz vseh 18 ključnih področij procesov, vključuje pa tudi:

- preprečevanje napak,
- upravljanja sprememb tehnologij,
- upravljanja sprememb procesov.

Model CMM je lahko uporabno orodje pri vzpostavljanju in izboljšavi procesov, saj je bil pri implementaciji konceptov celovitega obvladovanja kakovosti (angl. *Total Quality Management*) nadzorovan s strani programerske skupnosti, ki je skrbela, da je bil ta proces izveden z zdravo pametjo brez prekomernega balasta. Enostavna 5-nivojska struktura nudi podjetju možnost preproste integracije. V povezavi z inteligentno uporabo omenjenih konceptov zagotavlja učinkovito in robustno razvojno okolje (Paulk, 1998).

3.3 Model CMM z agilnimi procesi za majhna rastoča podjetja

Majhna razvojna podjetja, z manj kot 50 zaposlenimi, so temelj gospodarske rasti za mnoge države. V ZDA, Braziliji, Kanadi, na Kitajskem, v Indiji, na Finskem, Irskem, Madžarskem in še v mnogih drugih državah majhna podjetja predstavljajo tudi do 85 % vseh podjetij, ki razvijajo programsko opremo (Richardson & Von Wangenheim, 2007). V Sloveniji je to razmerje še večje in se giblje okoli 95 % (Car, 2010). Za nadaljevanje rasti potrebujejo majhna podjetja, ki razvijajo programsko opremo, učinkovite metode za razvoj programske opreme. Mnogo ljudi verjame, da so dobre prakse in rešitve drage in da so bolj namenjene večjim organizacijam, saj majhna podjetja nimajo dovolj sredstev za njihovo uporabo. Glede na to, kako številna so majhna razvojna podjetja, obstaja relativno malo programske opreme, ki bi jim pomagala pri teh težavah. Pri vzpostavljanju rešitev je potrebno paziti na omejitve podjetja, ki so po navadi povezane s sredstvi, ki so jih pripravljena porabiti za uvajanje novih procesov. Naslednji problem, s katerim se mnoga podjetja srečujejo, je, kako uporabiti nove procese, tehnike, primere dobre prakse in nova

orodja, ne da bi s tem zmanjšali produktivnost podjetja (Richardson & Von Wangenheim, 2007).

Majhna rastoča podjetja imajo težave pri implementaciji modela CMM, saj je ta zelo obsežen. Preobsežna dokumentacija, visoki stroški izobraževanja, visoke zahteve vloženih sredstev in pomanjkanje napotkov so le nekatere težave, s katerimi se manjša podjetja srečujejo pri implementaciji zrelostnega modela zmogljivosti. Opisano vodstveno strukturo v modelu CMM pogosto ni mogoče prenesti na manjša podjetja, prav tako ni mogoče povezati primerov uporabe, saj ustrezajo velikim podjetjem z dolgoletnimi izkušnjami in ustaljenimi procesi, četudi so ti slabi (Sharma & Sharma, 2009). Zaradi teh težav je pri omenjenih podjetjih napredek izboljšav veliko manjši, pa tudi rezultati nimajo tako močnega vpliva na poslovanje podjetja.

Pomembno je poudariti tudi razliko med majhnim podjetjem in rastočim podjetjem, saj eno ni vedno povezano z drugim. Podjetje je lahko majhno in hkrati staro, lahko pa je tudi veliko in rastoče. Razlika je pomembna pri implementaciji sprememb, saj so običajno mlada rastoča podjetja zanje veliko bolj dovzetna kot pa stara. Majhno podjetje ima lahko svoje procese na zelo visokem zrelostnem nivoju, kar pa običajno za mlada rastoča podjetja ne drži. Zaradi omenjenih razlogov bomo v magistrski nalogi majhno podjetje povezali z mladim rastočim podjetjem.

Običajno so ustanovitelji mnogih mladih podjetij, ki razvijajo programsko opremo, dobro podkovani z znanjem na področjih, ki niso povezana z razvojem programske opreme (Dangle et al., 2005). Tudi v primerih, ko imajo ustanovitelji izkušnje z razvojem programske opreme, so pogosto omejeni z razpoložljivimi sredstvi in odsotnostjo poslovnega modela (Voas, 1999).

Eden izmed prvih izzivov uporabe modela CMM je, da je glavni cilj majhnih rastočih podjetij preživetje (Coleman & O'Connor, 2008). Kljub zavedanju, da trenutno stanje ni zadovoljivo in da je vzpostavitev definiranih procesov nuja, je iskanje sredstev in dodeljevanje odgovornosti za izboljšavo procesov v smislu definiranja stabilnih procesov, ki se jih bo dejansko uporabljalo, težka poslovna odločitev (Paulk, 1998). Majhna podjetja pogosto delujejo v prepričanju, da:

- Smo vsi dovolj usposobljeni. Zaposlili smo ljudi, da opravljajo svoje delo in si z vidika časa in denarja ne moremo privoščiti usposabljanja (Paulk, 1998).
- Vsi komuniciramo drug z drugim. »Osmoza« deluje, ker smo si tako »blizu« (Paulk, 1998).
- Vsi smo heroji. Naredimo, kar je potrebno, pravila ne veljajo za nas in nas samo ovirajo (Paulk, 1998).

Potrebno pa se je zavedati, da majhna podjetja niso »pomanjšane« verzije velikih podjetij.

Običajno so zelo prilagodljiva in fleksibilna, kar je ena izmed njihovih glavnih prednosti. Zaradi fleksibilnosti se v majhnih podjetjih pogosto menja tehnologija, ki povzroča spremembo procesov in njihovo nestabilnost (Richardson & Von Wangenheim, 2007). To je tudi eden izmed demotivacijskih razlogov, zakaj podjetja ne uvedejo SPI (Baddoo & Hall, 2003). Baddoo in Hall (2003) v svoji raziskavi analizirata vzroke za demotivacijo zaposlenih.

Glavni skupni demotivacijski razlogi implementacije SPI pri razvijalcih so:

- premalo povratnih informacij,
- obremenitev,
- zmanjšanje kreativnosti,
- pomanjkanje vpletenosti managerjev.

Glavni demotivacijski razlogi pri projektnih vodjih so:

- pomanjkanje načinov kontroliranja projektov,
- »gašenje požarov«,
- fluktuacija osebja.

4 ANALIZA PROBLEMSKEGA STANJA

4.1 Opis podjetja

Podjetje je bilo ustanovljeno leta 2008, ustanovitelja sta bila takrat tudi edina zaposlena. S svojim znanjem in takratnimi izkušnjami sta razširila podjetje in kmalu zaposlila nove ljudi. Trenutno je v podjetju 7 zaposlenih in 2 aktivna zunanja izvajalca (Agencija Republike Slovenije za javnopravne evidence in storitve, 2015). Glavni produkti podjetja so spletne strani, spletne trgovine in spletni sistemi po naročilu. Podjetje je razvilo lasten sistem *Content Management System – CMS*, ki se uporablja za izdelavo naprednejše prilagojenih spletnih strani. Za spletne trgovine podjetje uporablja platformo Magento, pri sistemih po naročilu pa se prilagaja zahtevam in trenutno najbolj napredni tehnologiji. Podjetje poudarja zadovoljstvo strank in verjame v dolgoročno sodelovanje. Pri samem razvoju neprestano stremi po uporabi novejše tehnologije in modernem dizajnu, saj si tako pridobiva konkurenčno prednost.

4.2 Opis problemskega stanja

Za primerjavo problemskega stanja smo kot referenco vzeli podjetje The SysSec Group (Laesvirta & Ribière, 2008). Podjetje The SysSec Group se je s problemom povečanja zrelosti managementa poslovnih procesov soočilo, ko je z željo po sodelovanju do njih

pristopilo podjetje, za katero je moralo razviti produkt, ki ga trg v tistem trenutku še ni ponujal. Ravno ta produkt je podjetju The SysSec Group omogočil hitro rast, ki pa je razkrila tudi slabosti njihovih procesov.
Zaradi kompleksnega delovnega okolja so odkrili dve ključni težavi, ki ju s takratnim načinom dela, niso mogli rešiti. Prva težava je bila, kako pridobiti talentiran kader z dovolj izkušnjami in znanjem za delo, druga pa, kako zagotoviti 2-urno podporo ključnim uporabnikom.

Pri reševanju problema povečanja zrelosti managementa poslovnih procesov v izbranem podjetju smo našli povezavo z razlogi, ki so vodili do razkritja težav v podjetju The SysSec Group (Laesvirta & Ribière, 2008):

- pomanjkanje prenosa in izmenjave znanja,
- nezadostna infrastruktura,
- nerazviti in neučinkoviti procesi,
- neustrezno izobraževanje,
- neučinkovita interna komunikacija,
- nejasni cilji.

Te težave sta v svoji raziskavi, ki je potekala v obdobju petih let in vključevala 200 majhnih podjetij, identificirala tudi Brodman in Johnson (1997). Ugotovila sta, da je za majhna podjetja potreben prilagojen model CMM, saj celoten model zahteva prekomerno količino dokumentacije, previsoke stroške usposabljanja, preveliko sredstev ... V ta namen sta razvila prilagojeno verzijo modela, imenovano LOGOS Tailored CMM. Glavne značilnosti (Paulk, 1998) tega modela so:

- pojasnitev obstoječih praks,
- pretiravanje v očitnih stvareh,
- primeri alternativnih praks,
- uskladitev praks z majhnimi podjetji in majhnimi projekti v povezavi z njihovo strukturo in razpoložljivimi sredstvi.

4.3 Metodologija analize stanja

V podjetju smo se zavedali, da ima trenutno stanje razvojnih procesov številne pomanjkljivosti, a preden smo začeli s konkretnimi spremembami, smo morali najprej natančno ugotoviti, kje so glavne težave in zakaj do njih prihaja. Potrebno je bilo temeljito analizirati trenutno stanje razvojnega procesa (angl. *as-is*), da smo lahko na podlagi pridobljenih informacij pripravili načrt za spremembo razvojnih procesov (angl. *to-be*). Odločili smo se, da z vsakim zaposlenim opravimo poglobljen intervju. Pri izbiri metodologije raziskave smo se za poglobljen intervju odločili zato, ker smo od majhnega

števila ljudi (7) želeli pridobiti točno določene informacije. Intervju smo uporabili tudi zato, ker smo želeli izvedeti, kaj menijo zaposleni, česar preko ankete ne bi izvedeli (Driscoll, 2011).

Za intervju sem pripravil vprašanja, ki so bila prilagojena vsakemu zaposlenemu glede na njegovo vlogo v podjetju. Pregledali smo tudi zadolžitve intervjuvanca v elektronski obliki, saj preko sistema, ki med drugim vodi elektronske zadolžitve, najlažje opazimo težave v procesu. Ta sistem za vodenje projektov in sledenje napakam je v tem trenutku tudi edini sistem, ki se uporablja v vseh procesih in preko katerega se sledi napakam ter stanju aktivnosti. Uporablja se ga tudi za beleženje časa, ki se ga porabi za določeno aktivnost.

V praksi se analiza procesov razvoja programske opreme močno opira na razumevanje procesov z vidika udeležencev procesa razvoja programske opreme in interpretacije tega razumevanja, ki je lahko drugačna od realnega stanja (Coleman & O'Connor, 2008). Posledično je tak način raziskovanja kvalitativen, saj je usmerjen na posameznika in analizira njegov pogled in izkušnje s procesom. Ena izmed prednosti kvalitativnega raziskovanja je tudi, da lažje ugotovimo, kaj se v samem podjetju dogaja (Avison, Lau, Myers, & Nielsen, 1999).

V intervjuju smo sodelovali direktor, jaz in intervjuvanec. Intervju je bil sestavljen iz dveh delov. V prvem delu je vsak zaposleni prejel splošna vprašanja, ki so bila povezana z njegovim pogledom na proces in njegovim zadovoljstvom s procesom. Drugi del je vseboval specifična vprašanja, ki so bila neposredno povezana z njegovo vlogo v procesu.

Pri sestavi vprašanj smo se zgledovali po literaturi (Mahanti & Evans, 2012) in na podlagi zadolžitev v elektronski obliki intervjuvanca. Splošna vprašanja so bila:

- Z oceno 1–5 označi stopnjo definiranosti procesa razvoja programske opreme in argumentiraj svojo oceno.
- Z oceno 1–5 označi stopnjo zrelosti procesa razvoja programske opreme in argumentiraj svojo oceno.
- Naštej 3 glavne težave, ki jih srečuješ pri procesu razvoja programske opreme, in jih razloži.
- Kdaj je do teh težav začelo prihajati?
- Opiši celotni proces razvoja programske opreme v podjetju. Za kateri del procesa se ti zdi, da potrebuje največ pozornosti v smislu izboljšave?
- Ali si s trenutnim stanjem procesa razvoja programske opreme zadovoljen? Argumentiraj.

Ker smo želeli čim natančnejšo sliko trenutnih procesov z vseh vidikov (Aaen, 2003), smo v drugem delu intervjuja izpostavili določene konkretne težave, ki so bile zabeležene v

sistemu za vodenje projektov, ter se pogovorili, zakaj je do njih prišlo in kako bi jih v prihodnje preprečili. V tem delu je bil glavni poudarek na tem, kako intervjuvanec vidi težave in kako si predstavlja, da bi se jih lahko odpravilo. Zanimalo nas je torej, kakšen je njegov pogled na celoten razvojni proces.

4.3.1 Trenutno stanje razvojnih procesov

Po opravljeni raziskavi smo lahko potrdili težave trenutnih »ad-hoc« procesov, ki so se začele pojavljati z rastjo podjetja. Ko je bilo v podjetju manj zaposlenih, se je namreč z ustno komunikacijo lahko prenesla večina potrebnih informacij. Če se je kakšna informacija izgubila, je prišlo do napake v procesu, a so se zaradi majhnega števila vpletenih v proces te napake tudi hitro odpravile (običajno sta bila to dva zaposlena).

Druga težava, ki smo jo odkrili pri analizi stanja, je obvladovanje kompleksnejših projektov, ki se jih je podjetje začelo lotevati. Do te točke je podjetje projekt zaključilo v največ mesecu dni, kar pomeni, da projekti niso bili kompleksni. Sedaj se ukvarjamo s projekti, ki jih razvijamo več kot pol leta in jih nato aktivno nadgrajujemo. Trenutno največji projekt, ki smo ga razvili v 7 mesecih, nadgrajujemo z različnimi posodobitvami na željo naročnika že več kot 18 mesecev. V tem času se je izvorna koda sistema potrojila, kar pomeni, da je kompleksnost sistema narasla. To se kaže predvsem pri iskanju in odpravljanju napak ter implementaciji nadgradenj. Tudi ocene nadgradenj so zato manj natančne, kar vodi do nezadovoljstva naročnikov.

V podjetju uporabljamo spletno orodje za vodenje projektov in sledenje aktivnostim, imenovano Trac (Software, 2001). Orodje smo začeli uporabljati ravno z namenom, da bi zmanjšali neučinkovito komunikacijo, a je njegova uporaba zelo različna glede na vlogo udeleženca v podjetju. V naši raziskavi smo prav tako odkrili problem različnega razumevanja procesov udeležencev le-teh. Problem razumevanja procesa različnih udeležencev je bil naveden kot eden izmed glavnih izzivov, ko se podjetje odloči, da bo začelo vzpostavljati definirane procese oziroma implementirati SPI (Aaen, 2003). Vodilne v podjetju v večini primerov zanima časovna komponenta aktivnosti, v nasprotju z razvijalci, katerim je ključna vsebina. Baddoo in Hall (2003) sta raziskovala te različne poglede in jih opisala kot demotivacijske razloge za povečanje zrelosti razvojnih procesov. Tudi komunikacija je izrazito drugačna glede na udeleženca procesa. Razvijalci v podjetju potrebujejo večino informacij v pisni obliki še posebej pri večjih projektih, saj se tako izločita nejasnost in kasnejše pregovarjanje o napačno delujoči funkcionalnosti. S tem, ko se zahteve pretvorijo v pisno obliko, se tudi razkrijejo pomanjkljivosti v specifikaciji zahtev. Pomanjkljivosti lahko vodijo do zaustavitve procesa, saj je potrebno to informacijo pridobiti od naročnika. Nepopolna dokumentacija zahtev je ena izmed najpogostejših težav, s katerimi se srečujejo razvijalci programske opreme (Tong, 1994). Tak način dokumentiranja zahtev terja veliko več časa kot ustno izročanje ali pa posredovano e-sporočilo z naročnikovimi željami, zato se ga pogosto preloži na stran razvijalca. Tu

nastane težava, saj razvijalci v našem podjetju običajno ne komunicirajo z naročnikom, kar pomeni, da imajo veliko manj informacij o njegovih zahtevah, željah in pričakovanjih. Operirati morajo le z informacijami, ki so jih prejeli preko e-pošte in ustnega izročila projektnega vodja. Prekomerno zanašanje na ustno komunikacijo pogostokrat povzroča težave. Z njimi se srečuje mnogo manjših podjetij, saj verjamejo, da si lahko, ker so dovolj majhni in so si dovolj »blizu«, vse informacije posredujejo ustno (Paulk, 1998).

Težava trenutnih procesov je tudi ta, da udeleženci procesov ne vedo vedno, v kateri fazi so in katera faza sledi. Kljub temu da so bili zelo podobni projekti že zaključeni, si aktivnosti v procesu ne sledijo vedno v istem vrstnem redu. To povzroči daljšo časovno izvedbo in več napak v produktih. Pri pregledovanju konkretnih težav, ki so zabeležene v sistemu za vodenje projektov, se je izpostavil tudi problem popisovanja funkcionalnosti sistemov. Funkcionalnosti in njihovo delovanje namreč niso popisani, kar pomeni, da pri napakah, ki se jih opazi pri nadgradnji sistema, zelo težko ugotovimo, ali je sama nadgradnja povzročila na novo nastalo delovanje ali je bilo takšno stanje že pred njo.

V intervjujih z vodji projektov, ki imajo največ stika z naročniki, se je ugotovilo, da večina naročnikov ne razume kompleksnosti razvoja programske opreme. Spletne sisteme si predstavljajo vizualno, kot elemente na elektronskih vezjih, ki jih razvijalci zlagamo skupaj. V večini primerov je tako poenostavljena predstava zelo napačna in vodi do nezadovoljstva strank, saj ne razumejo, zakaj razvoj traja toliko časa. Naša naloga je, da naročniku predstavimo natančnejšo sliko razvoja programske opreme in mu razložimo, da se za vizualnimi elementi skriva funkcionalnost, ki je kompleksnejša od njihove predstave. To predstavitev običajno opravi vodja projekta, ki tudi popiše naročnikove zahteve. Na tej točki je ključno, da je vodji projekta proces izdelave programske opreme popolnoma jasen, saj mora od naročnika pridobiti vse informacije, ki so potrebne za razvojni proces. Postavljati mora vprašanja namesto razvijalcev in oblikovalcev, saj jim bo moral kasneje te informacije posredovati. Že na začetku procesa izdelave programske opreme, ko vodja projekta dobi prve informacije s strani naročnika, je pomembno, da so procesi definirani in da je njihovo razumevanje enotno. V nasprotnem primeru se od naročnika ne pridobi dovolj informacij, zato jih je potrebno pridobiti kasneje v razvojnem procesu.

V najboljšem primeru se proces ustavi, potrebno je čakati naročnika, da sporoči manjkajoče dele, v najslabšem primeru pa je potrebno po prejetju manjkajočih informacij programsko opremo korenito spremeniti, ker je bila napačno zasnovana. Sam proces se zato skupaj s stroški razvoja podaljša. Naročnik dobi produkt kasneje, kot mu je bilo obljubljeno, zato je nezadovoljstvo stranke razumljivo.

Druga težava, ki jo občutijo stranke, je povezana s posodobitvijo in nadgradnjo obstoječe programske opreme, ki jo je razvilo izbrano podjetje. V procesu razvoja se sistemov in njihovih funkcionalnosti ne popisuje, kar kasneje pri iskanju napak, posodobitvah in nadgradnjah povzroča težave. Podjetje nima definiranega procesa za upravljanje sprememb

(angl. *Change management*), kar povzroči, da stranke zahtevo za nadgradnjo ali posodobitev sistema pomanjkljivo definirajo, zato je potrebno te informacije pridobiti naknadno. Zaradi nerazumevanja procesa udeležencev procesa v izbranem podjetju se običajno te zahteve, ki so bile posredovane preko e-pošte, posreduje razvijalcem, ki ne razumejo, kaj je potrebno narediti. Proces se tako podaljša ali celo ustavi.

4.3.2 Ciljno stanje razvojnih procesov

Prvi cilj, ki smo si ga v izbranem podjetju zastavili, je dvigniti zrelost poslovnega procesa na drugo stopnjo v modelu CMM. Dolgoročna strategija, ki jo v okviru te magistrske ne bomo obravnavali, je, da podjetje doseže tretjo stopnjo modela CMM, a se zavedamo, da je to zelo zahtevna naloga, ki je ni mogoče doseči v kratkem obdobju. Tretjo stopnjo modela CMM lahko dosežemo le, če podjetje že posluje z drugo stopnjo modela CMM, zato je zelo pomembno, da izberemo najprimernejši model razvoja programske opreme, ki nam bo to omogočal.

Za majhno podjetje je pomembno, da uporablja prilagodljiv razvojni model (Paulk, 1998), a ker želimo dvigniti zrelost procesov, mora biti model tudi predvidljiv. Zaradi tega moramo izbrati razvojni model, ki bo dovolj predvidljiv, da bomo lahko na sam proces aplicirali SPI, a kljub temu dovolj prilagodljiv, saj je to ena izmed najpomembnejših lastnosti majhnih rastočih podjetij (Sutton, 2000).

5 POSTOPEK IZBOLJŠAVE ZRELOSTNEGA MODELA RAZVOJA PROGRAMSKE OPREME

Da bi dosegli ciljno stanje razvojnih procesov, smo se odločili, da vse tri modele razvoja programske opreme (Slapovni model, Spiralni model in Ekstremno programiranje), ki smo jih v teoretičnem delu pregledali in analizirali, testiramo na konkretnih projektih.

Za vsak model razvoja programske opreme smo izbrali primeren projekt, ki smo ga upravljali z izbranim modelom. Projekti so si bili med seboj karseda podobni, saj smo tako lažje primerjali razvojne modele. Pri procesu izbiranja modelov razvoja programske opreme smo bili omejeni s sredstvi podjetja. Zavedali smo se, da je en projekt za ovrednotenje modela za razvoj programske opreme zelo malo, a glavni namen tega procesa izbora ni bil, da odkrijemo »najboljši« model razvoja programske opreme, ampak najprimernejši model za izbrano podjetje. Tudi v primeru, če bi bil izbrani model na dolgi rok manj primeren, je podjetje kljub temu dvignilo zrelostni model razvoja programske opreme, kar je tudi cilj magistrske naloge.

Pred začetkom testiranja modelov razvoja programske opreme smo morali pregledati ključna področja, ki so zahtevana za model CMM druge stopnje, in opredeliti, kako jih bomo dosegli s konkretnim testiranim modelom razvoja programske opreme. Kriterije, po

katerih smo ocenjevali in merili model razvoja programske opreme, lahko razdelimo v dve skupini: kvantitativne in kvalitativne.

Za razvoj programske opreme so značilne tri enostavne meritve: čas, velikost in število napak (Mahanti & Evans, 2012). Na podlagi teh meritev smo izbrali kvantitativne dejavnike. Kvalitativne smo izbrali na podlagi Boehmovih petih dimenzij izbire razvojnega modela (Boehm & Turner, 2003).

Kvantitativni dejavniki:

- Število napak v programski opremi zaradi slabo definiranih zahtev.
- Število zaustavitev procesa razvoja programske opreme zaradi slabo definiranih zahtev (angl. *process blocker*).
- Časovna komponenta, ki bo delno odvisna od kompleksnosti projekta, a ker si bodo izbrani projekti po kompleksnosti zelo podobni, bodo tudi ti podatki primerljivi.

Kvalitativni dejavniki:

- Kako močno razvojni model posega v naše delo.
- Koliko dela imamo z razvojnim modelom.
- Koliko nas razvojni model ovira pri inovativnosti in razvoju.
- Kompleksnost razvojnega modela in s tem povezana težavnost izvedbe ter implementacije razvojnega modela.
- Prilagodljivost razvojnega modela glede na spremembe v fazi razvoja.

Po zaključenem projektu smo za vsak razvojni model ponovno opravili poglobljen intervju z vsakim zaposlenim v izbranem podjetju in raziskali, kakšni so bili učinki kvalitativnih dejavnikov. Kvantitativne dejavnike smo že vnaprej analizirali ter jih med intervjujem komentirali in preverjali, ali odražajo realno stanje. Analizirali smo jih tako, da smo najprej zbrali podatke zadnjih petih projektov, ki so bili opravljeni pred začetkom testiranja razvojnih modelov, in izračunali povprečno število napak zaradi slabo definiranih zahtev, povprečno število zaustavitev razvojnega procesa zaradi slabo definiranih zahtev in povprečno časovno komponento. Ti podatki so bili osnova za primerjanje s podatki, ki smo jih dobili po opravljenem testiranju vsakega razvojnega modela (slapovni model, spiralni model, ekstremno programiranje).

5.1 Opis testiranih projektov

Projekti, ki smo jih izbrali za testiranje vseh modelov, so si bili v osnovi zelo podobni. Temeljili so na spletni strani z naprednim dodatkom, ki ga je bilo potrebno prilagojeno razviti v skladu z naročnikovimi zahtevami. V omenjenih projektih sodeluje celotno

podjetje, zato je bilo pomembno, da se je testiranje opravilo na projektih takega tipa. Pri izbiri kriterijev za izbor primernih projektov smo se zgledovali po preteklih študijah primerov (Dangle et al., 2005).

Običajno taki projekti trajajo od dva do štiri mesece. V skrajnih primerih se lahko zavlečejo tudi dlje. Najpogostejše težave, ki so povzročile, da se je projekt podaljšal, so nastale zaradi nejasnih zahtev naročnika, kar je lahko tudi posledica naših neopredeljenih procesov.

5.2 Testiranje slapovnega modela

Pred začetkom razvoja je bilo potrebno za slapovni model izbrati in dodelati primerno predlogo (angl. *template*) dokumenta za projektno vodenje, ki je ustrezala naslednjima kriterijema: 1) Vključevati mora rešitve za zagotavljanje ključnih področij, ki so potrebna za doseganje druge stopnje modela CMM (Paulk et al., 1993, str. 2) Dokumentacija mora biti skladna s slapovnim modelom, kar pomeni, da mora vključevati faze razvoja, ki so tipične za slapovni model (Royce, 1970). Pomembno je omeniti, da sta oba kriterija enako pomembna, zato pri iskanju predlog nismo želeli pristati na kompromise v smislu, da bi izključili določene elemente slapovnega modela ali izključili določena ključna področja modela CMM.

Oba kriterija smo prilagodili in odstranili dele dokumentacije, ki so za izbrano podjetje manj primerni, kot je na primer ključno področje upravljanje podizvajalca programske opreme. Pri prilagajanju dokumentacije smo se poleg na lastne potrebe močno zanašali tudi na nasvete, ki smo jih našli v literaturi (Richardson & Von Wangenheim, 2007; Dangle et al., 2005; Tong, 1994; Paulk, 1998).

Predloga dokumenta za projektno vodenje je organizirana v skladu s procesi PMBOK (Guide, 2004), ki v osnovi vključuje 18 poglavij. Izločili smo 5 poglavij, to so: Načrt upravljanja stroškov, Načrt upravljanja javnih naročil, Načrt upravljanja tveganja, Načrt upravljanja kakovosti in Načrt upravljanja zaposlovanja.

Dve ključni področji druge stopnje modela CMM smo izvzeli iz omenjenega dokumenta: upravljanje podizvajalca programske opreme in upravljanje konfiguracije programske opreme. Prvo ključno področje smo izvzeli zato, ker izbrano podjetje v tem trenutku ne uporablja podizvajalca za izdelavo programske opreme. Konfiguracija programske opreme je izredno pomemben element v procesu razvoja programske opreme (Richardson & Von Wangenheim, 2007). Za izključitev konfiguracije programske opreme iz dokumentacije smo se odločili zato, ker ima izbrano podjetje v tem trenutku to področje že zelo dobro urejeno in smo se odločili, da bomo ta čas in energijo raje vložili v bolj kritična področja. Način konfiguracije programske opreme je v kulturi podjetja, zato menimo, da bi nas strogo definiranje in dokumentiranje konfiguracije omejevalo in bi izgubili določen del

prilagodljivosti. Na to je že opozoril tudi Sutton (2000) v svojem članku, v katerem opisuje vlogo procesov v mladih podjetjih.

Prilagojen dokument za projektno vodenje je imel na koncu naslednja poglavja:

1. Uvod
2. Pristop projektnega vodenja
3. Obseg projekta
4. Seznam mejnikov
5. Osnovni časovni načrt in struktura razdelitve del
6. Načrt upravljanja sprememb
7. Načrt upravljanja komunikacij
8. Načrt upravljanja obsega projekta
9. Načrt upravljanja terminskega plana
10. Koledar sredstev
11. Izhodiščni stroški
12. Minimalne zahteve
13. Sprejetje naročnika

Uvod vsebuje pregled projekta na visoki ravni in kaj je vključeno v načrt projektnega vodenja. Vključuje abstraktni opis projekta, namen projekta in končne rezultate (angl. *deliverables*). V naslednjem poglavju je predstavljen splošen pristop k upravljanju projekta. Opisane so splošne vloge in pooblastila članov projektne skupine. Vključuje tudi omejitve sredstev, ki jih projekt lahko ima. Izjava Obsega projekta je izhodiščna točka za izdelavo seznama mejnikov. Vključuje podrobnosti zahtev, ki so vključene in izključene iz projekta. S tem se razjasni, kaj je vključeno v projekt, in se pomaga izogniti kasnejšim nejasnostim med člani projektne ekipe in deležniki. Na podlagi informacij, ki smo jih pridobili v poglavju Obseg projekta, se lahko izdela seznam mejnikov (angl. *milestones*). Potrebna dela se razdelijo na segmente, določi se, kateri sklopi bodo opravljeni do določenega datuma. V tem poglavju je tudi jasno opredeljeno, kako se bodo mejniki spremenili v primeru, če bi prišlo do sprememb obsega projekta. V petem poglavju je pripravljen osnovni časovni načrt, ki je povezan s strukturo razdelitve del. Struktura razdelitve del nudi delovni paket, ki se izvede za projekt. Poglavje Načrt upravljanja sprememb opisuje proces nadzora nad spremembami projekta. Idealno bi bilo, če bi se ta proces izvajal na nivoju organizacije, a ker v tem trenutku v izbranem podjetju še vzpostavljamo definirane procese, smo ta del obdržali v tem dokumentu. Ker je izbrani projekt srednje kompleksen, lahko ta del obdržimo v svojem poglavju. V nasprotnem primeru, torej če bi šlo za večji, kompleksnejši projekt, je priporočljivo to poglavje ločiti v poseben dokument. Z dobro komunikacijo se je mogoče izogniti številnim težavam.

Način pretoka informacij smo opredelili v sedmem poglavju, ki vsebuje naslednje ključne elemente:

- komunikacijske zahteve, ki temeljijo na vlogah;
- katere informacije bodo sporočene;
- način sporočanja informacij;
- kdaj se bodo informacije sporočale;
- kdo sporoča in kdo prejema informacije;
- ravnanje z informacijami.

Temu poglavju smo namenili večjo pozornost, saj imamo v izbranem podjetju visoko stopnjo komunikacij, zato je bistvenega pomena, da je ta komunikacija kvalitetna (Aaen, 2003). V poglavju Načrt upravljanja obsega projekta je pomembno, da je obseg jasno opredeljen in dobro dokumentiran. Slabo opredeljen in dokumentiran obseg v večini primerov vodi do zamude projekta, nepotrebnega dodatnega dela, nedoseganja rezultatov, prekoračitve stroškov, slabe volje naročnika in izvajalca (Sharma & Sharma, 2009) ... Tu smo zato jasno opredelili, kdo ima pristojnosti in kdo odgovarja za upravljanje obsega projekta in kako je obseg projekta definiran. Natančneje ko je vsebina tega poglavja opisana, manj težav imamo pri končni predaji projekta (Paulk, 1998). V naslednjem poglavju smo opredelili splošno ogrodje, ki se ga uporablja pri oblikovanju terminskega plana projekta. V nasprotju od vsebine v poglavju Seznam mejnikov smo tu dali večji poudarek dnevnemu in tedenskemu razporejanju sredstev. S tem smo učinkoviteje zagotovili, da so naloge in mejniki doseženi pravočasno. V Koledar sredstev smo vključili ključna sredstva, ki so potrebna za projekt, in časovno komponento, ki je vezana na sredstva. V Izhodiščnih stroških smo za vsako fazo razvoja dodelili okvirne stroške razvoja. Te faze razvoja so skladne s slapovnim modelom (Royce, 1970). V poglavju Minimalne zahteve smo opisali minimalne zahteve, ki so sprejemljive za naročnika. Te zahteve se nanašajo predvsem na kakovost izdelka in ne na obseg oziroma specifikacije produkta. Na koncu je potrebno, da dokument podpišeta naročnik in odgovorni v izbranem podjetju. S podpisom obe strani potrdita, da sta seznanjeni z vsebino dokumenta in da se z njo strinjata.

Pred začetkom razvoja je bilo potrebno pri slapovnem modelu izpolniti veliko večjo količino dokumentacije, kar je povzročilo, da se je sam razvoj pričel kasneje, saj smo za dokumentacijo porabili štiri tedne. V prvem tednu smo iskali primerne predloge dokumenta za vodenje projekta, ga dodelali in prilagodili, preverili skladnost z našimi potrebami in skladnost z modelom CMM (Brookes & Clark, 2009) ter slapovnim modelom (Royce, 1970). V naslednjih treh tednih smo izpolnjevali vsebino dokumenta. Pri tem smo se morali naučiti uporabljati tudi določena nova orodja, ki so nam omogočila izpolnitev tega dokumenta (na primer Microsoft Project).

Zaradi zelo dobre dokumentacije, v kateri smo opredelili vse potrebne elemente, ki smo jih potrebovali v času razvoja, je bila prijavljena samo ena zaustavitev procesa razvoja zaradi slabo definiranih zahtev. Večja količina dokumentacije je povzročila, da se je čas razvoja občutno povečal. Namesto povprečno treh mesecev smo za ta projekt porabili štiri mesece.

Časovno komponento razvoja je vsekakor podaljšala začetna faza, v kateri smo izpolnjevali potrebno dokumentacijo, pa tudi natančnejše popisovanje nalog.

V poglobljenem intervjuju z zaposlenimi smo dobili zelo enotne rezultate, saj je slapovni model vplival na začetni del načrtovanja in dogovarjanja, ko se je izpolnjevala dokumentacija, kot tudi na sam proces razvoja. Vodja projekta je imel pri slapovnem modelu veliko več dela na začetku projekta, kar je skladno s fazami, ki si sledijo v slapovnem modelu (Royce, 1970). V nadaljevanju smo bili pri razvoju vsi enotnega mnenja, da je delo potekalo občutno bolj tekoče, saj so bile zahteve zelo jasne, terminski načrt je bil določen in v primeru, ko je prihajalo do zamud, smo to zelo hitro ugotovili in se temu prilagodili. V fazi razvoja se je bilo potrebno natančno držati določenih terminov, in če je prišlo do prekoračitve začrtanih ciljev, smo morali prilagoditi urnik. Od začetka do konca projekta smo porabili veliko več časa za dokumentiranje celotnega procesa, kar je povzročilo tudi, da se je projekt izvajal dalj časa.

Zaradi narave projekta nas model pri razvoju ni oviral, a v primeru, da bi šlo za kompleksnejši projekt, bi bila večja verjetnost, da bi čutili omejitve modela, saj v razvoju dopušča zelo malo prilagajanja (Cioch et al., 1994). Težava bi se pojavila tudi, če naročnik ne bi imel jasne predstave o produktu, ki ga želi, saj ne bi mogli izpolniti potrebne dokumentacije, ki se pripravi na začetku projekta.

5.3 Testiranje spiralnega modela

Rezultati slapovnega modela so bili zelo spodbudni, saj je model zelo stabilen in robusten. Glavna težava, ki smo jo prepoznali, je, da prihaja do večjih težav ob uporabi nove tehnologije, kar predstavlja večje tveganje za razvoj. Skupaj s težjim prilagajanjem spremembam pri zahtevah predstavlja slapovni model težavo pri sami izvedbi. Vse omenjene slabosti je izpostavil že Royce (1970) v svojem prvotnem članku, v katerem je predlagal tudi spremembe. Kasneje so na te slabosti opozorili tudi drugi (Cioch et al., 1994), zaradi česar se je razvil spiralni model (Boehm, 1988), ki naslavlja omenjene težave.

Tudi pri spiralnem modelu smo najprej poiskali primerno predlogo dokumenta, s katero bomo vodili projekt in planirali iterativne faze razvoja. Preverili smo skladnost dokumenta s spiralnim razvojnim modelom in ključnimi področji druge stopnje modela CMM. Tudi tukaj smo, zaradi enakih razlogov, izvzeli ključni področji: upravljanje podizvajalca programske opreme in upravljanje konfiguracije programske opreme.

Nastali dokument je bil veliko krajši in preprostejši od dokumenta, ki smo ga uporabili pri slapovnem modelu, saj je vseboval le abstraktno ozadje projekta in dve tabeli, v katerih so bile predstavljene iteracije in razvojni terminski plani. Prenašanje znanja, pridobljenega iz predhodne iteracije v naslednjo iteracijo, je ena izmed glavnih prednosti spiralnega modela

(Boehm, 1988), zato je pomembno poudariti, da je bilo pri vsaki iteraciji potrebno omenjeni dokument posodobiti.

Dokument je bil sestavljen iz treh delov. Prvi del je vseboval:

1. Opis projekta.
2. Časovne omejitve projekta.
3. Zgodnje zahteve in načrtovanje ocen.
4. Rezultati.

Tabela 2: Enostavni prikaz razreza zahtev za iteracije

Funkcionalnosti na visoki ravni	**Testiranje razčlenjevanja (1-28)**	**Prototip 1 (2-20)**	**Prototip 2 (25-1)**	**Objava (3-31)**
Razčlenjevanje dokumentov	✓	✓	✓	✓
Struktura podatkovne baze	Da, s testnimi podatki	✓	✓	✓
Poročila		Zaželeno	✓	✓
Iskalnik		✓	✓	✓
Uvoz in izvoz podatkov		Osnovno	✓	✓
Avtomatična varnostna kopija		Ne, samo v meniju	Zaželeno	Ročno
Mrežne operacije			Osnovno	✓

V Opisu projekta so se na kratko po alinejah napisale glavne značilnosti projekta. V nasprotju s slapovnim modelom smo tu dali tu večji poudarek jedrnatosti, katere smo se nato držali skozi celoten dokument. V naslednjem delu smo zapisali datume, ki so pomembni za projekt.

Na primer »28. november – testiranje, razčlenjevanja pdf dokumenta«; »končna objava v najkrajšem možnem času, a ne kasneje kot v prvem kvartalu 2015«.

V Zgodnjih zahtevah in načrtovanju ocen smo poizkušali karseda realno oceniti in vključiti funkcionalnosti, potrebne za vsako iteracijo. Tu smo se držali načel, ki so tipična za spiralni model (Boehm, 1988), in sicer predvsem tega, da smo v prvih iteracijah razvijali tiste funkcionalnosti, ki so za projekt predstavljale največje tveganje. To tveganje se je v večini primerov nanašalo na nerazumevanje strankinih želja in/ali na uporabo novih tehnologij, ki se jih v izbranem podjetju zelo pogosto poslužujejo. Na koncu smo zapisali končne ugotovitve, povezane z zbranimi informacijami o projektu, predvsem v povezavi s tveganji.

Naslednji del dokumenta je predstavljala tabela, v kateri je bil naveden seznam funkcionalnosti in iteracij z okvirnimi datumi. Enostavni primer omenjene tabele predstavlja Tabela 2.

V tretjo tabelo smo zapisali razvojne iteracije, ki jih je planirala razvojna ekipa. V tej tabeli

je večji poudarek na iteracijah – kdaj se bodo iteracije izvajale in kaj se bo v njih izvajalo. Tabela 3 prikazuje enostavno verzijo omenjene tabele.

Tabela 3: Enostavni prikaz razvojnega načrtovanja iteracij

# iteracije	Obdobje	Dela
	1. avgust	1. Sestavljena predstava celotnega projekta. 2. Projektno delo je bilo razdeljeno na iteracije. 3. V iteracijah so bile postavljene prioritete 4. Razčlenjevanje dokumentov je bilo identificirano kot največje tveganje, zato se bo to funkcionalnost prvo razvilo.
1.	1. do 15. avgust	1. Začetek specifikacij manj tveganih funkcionalnosti. 2. Objava prve verzije prototipa razčlenjevalnika dokumentov. Potrebno nadgraditi prototip, da se identificira vse možne težave. 3. Načrtovanje, kodiranje in testiranje funkcionalnosti za predstavitev.
2.	15. avgust do 29. avgust	1. Kritično ovrednotenje časa, ki je bil porabljen za funkcionalnosti, ki so bile narejene do 28. avgusta (kaj smo se naučili v prejšnji iteraciji). 2. Ugotoviti katera tveganja so še vedno prisotna. Načrtovanje in razdeljevanje dela. 3. Revizija seznama funkcionalnosti za trenutno iteracijo. Prilagoditev projekta po potrebi. 4. Načrtovanje, kodiranje in testiranje funkcionalnosti za naslednjo predstavitev.
3.	1. september do 30. september	1. Kritično ovrednotenje časa, ki je bil porabljen za funkcionalnosti, ki so bile narejene do 14. septembra. 2. Pregled rezultatov testiranj. 3. Pridobiti povratno informacijo stranke na podlagi predstavitve. 4. Dokončati potrebna dela pri razčlenjevalniku dokumentov. 5. Dokončati načrtovanje, kodiranje, testiranje za zgodnjo beta različico konec novembra.
Beta različica	1. oktober do 25. oktober	1. Pridobiti povratne informacije od stranke na podlagi beta različice. 2. Odpraviti napake. 3. Dokončati ostala projektna dela. 4. Objava.

Razbijanje projekta na manjše mini projekte in izdelava prototipov najbolj tveganih funkcionalnosti sta se izkazala za zelo učinkovit pristop. Nikoli nismo prišli do nepredvidljivih situacij, saj smo ugotovitve prenašali iz iteracije v iteracijo, tako kot narekuje spiralni model (Boehm, 1988). Kot zelo učinkovit pristop se je izkazalo tudi to, da so se najprej začele razvijati najbolj tvegane funkcionalnosti. Tako smo v primeru napačne implementacije dobili od naročnika povratno informacijo, na podlagi katere smo opravili potrebne popravke. Celoten načrt se je ob vsaki iteraciji posodobil in prilagodil potrebam naročnika in naravi tehnologije, ki smo jo uporabljali. Pri testiranju spiralnega modela smo tako potrdili večino prednosti, ki so zanj značilne (Boehm, 1988).

Dokumentacije je bilo v primerjavi s slapovnim modelom občutno manj, zato se je lahko razvoj pričel veliko prej. Za iskanje primerne predloge in izpolnitev celotne začetne dokumentacije smo potrebovali en teden, kar je tri tedne manj kot pri slapovnem modelu. Pri spiralnem modelu smo morali dokumentacijo med projektom posodabljati na začetku vsake iteracije, z izjemo prve, za kar smo skupaj porabili še dodatnih šest delovnih dni, kar skupaj znaša še vedno občutno manj kot pri slapovnem modelu. Naše testiranje modela je skladno tudi s teorijo (Cioch et al., 1994; Boehm et al., 1998), in sicer v smislu, da je slapovni model dokumentno voden, spiralni model pa temelji na obvladovanju tveganj, kar posredno pomeni, da člani projekta veliko manj časa porabijo za izpolnjevanje dokumentacije in se bolj osredotočajo na sam projekt.

V poglobljenih intervjujih z zaposlenimi smo pregledali kvantitativne dejavnike, ki smo jih določili za ocenjevanje modela. Število napak je bilo zaradi slabo definiranih zahtev v primerjavi s slapovnim modelom višje, a jih v času razvoja nismo občutili kot kritične napake, ampak kot del razvoja. Te napake so se razrešile v naslednji iteraciji, ko jih je naročnik pri predstavitvi izpostavil. Ključnega pomena pa je, da po objavi projekta teh napak ni bilo več.

Ker slapovni model temelji na obvladovanju tveganj (Boehm, 1988), so se najprej začele razvijati najbolj tvegane funkcionalnosti. Posledično se je v vsaki iteraciji razvil prototip najbolj tveganih funkcionalnosti, ki jih je naročnik pri predstavitvi pregledal. V primeru, da si je naročnik to funkcionalnost zamislil drugače, smo jo v naslednji iteraciji spremenili, a v večini primerov se je zgodilo, da je naročnik pri pregledu prototipa dobil in nam predstavil novo, boljšo idejo te funkcionalnosti. Zaradi opisanega procesa pri razvoju ni nikoli prišlo do zaustavitve razvojnega procesa, saj se je v vsaki iteraciji razvijal le del celote, tako kot narekuje spiralni model (Boehm, 1988).

V intervjujih so zaposleni menili, da je spiralni model veliko manj posegal v njihovo delo in da tudi s samim modelom niso imeli veliko dela. Zaposlenim se je izvedba modela zdela zelo enostavna in učinkovita. Z izdelavo prototipov je podjetje občutilo veliko razliko pri prilagajanju naročnikovim zahtevam v času razvoja, kar je pozitivno pozdravil tudi naročnik projekta, na katerem se je spiralni model testiral.

Tabela 4: Prikaz nalog izvedenih po fazah

Faza	Dela
Analiza	• Dogovarjanje glede potrebnih funkcionalnostih. • Virtualna dokumentacija (JIRA), do katere ima dostop tudi naročnik. • Dokončni dogovor o potrebnih zahtevah.
Načrtovanje	• Razdelitev del na manjše enote. • Ustvarjanje elektronskih zadolžitev v sistemu JIRA. • Načrtovanje dvo tedenskih objav.
Kodiranje	• Kodiranje v parih.

Tabela 5: Prikaz nalog izvedenih po fazah (nad.)

Faza	Dela
Kodiranje	• Usklajevanje kodiranja na dnevnih sestankih.
Testiranje	• Izvajanje popravkov. • Redno objavljanje kode.

5.4 Testiranje ekstremnega programiranja

Od vseh testiranih modelov razvoja programske opreme je najtežje implementirati ekstremno programiranje, saj temelji na vrednotah zaposlenih (Liu & Lu, 2012), te pa seveda ni mogoče spremeniti čez noč. Druga občutnejša razlika, ki loči ekstremno programiranje od slapovnega in spiralnega modela, je ta, da ekstremno programiranje ne vsebuje nobene dokumentacije, ampak se drži temeljnih načel in vrednot agilnih procesov (Misra et al., 2012).

Vrednote smo morali skupaj z načeli uporabiti pri razvoju, a zavedali smo se, da jih bo treba, vsaj na začetku, močneje podkrepiti. Odločili smo se, da v vsako pisarno na vidno mesto obesimo tablo, na kateri bodo napisani temeljne vrednote in temeljna načela agilnih procesov, med katere spada tudi ekstremno programiranje (Liu & Lu, 2012). Za pomoč pri vodenju razvoja smo uporabili spletno orodje JIRA (ltd, 2016), ki je posebej prilagojeno za agilne procese. JIRA nam je služila kot nadomestilo dokumentov za vodenje projektov, preko katerega smo lahko ustvarjali zadolžitve, jih prenašali, prijavljali napake itd. Dostop do omenjenega sistema je imel tudi naročnik, kateremu je omogočal nenehen pregled nad našim delom.

Izvedba nalog v vsaki fazi je prikazana v Tabeli 4. Pri implementaciji ekstremnega programiranja smo morali spremeniti tudi naše vsakodnevne dejavnosti. Uvedli smo programiranje v parih in kratke dnevne sestanke, na katerih je vsak odgovoril na 3 vprašanja:

- Kaj si naredil včeraj?
- Kaj boš naredil danes?
- Ali pričakuješ kakšne težave?

Med programerji smo določili stil programiranja. Za kodo Python smo uporabili smernice PEP 0008, za JavaScript, HTML in CSS smo uporabili smernice, ki jih uporablja Google. Vsak del kode je bilo treba obvezno testirati, preden se je objavil. Znižali smo toleranco za obstoj zastarele kode. To je pomenilo, da ko je programer opazil kodo, ki ni bila več potrebna ali bi jo bilo potrebno zaradi novih sprememb spremeniti, je bil to dolžan narediti. Slednje se je uspešno izvajalo zaradi programiranja v parih.

Začetni proces je bil zelo podoben kot pri slapovnem modelu, le da smo zahteve skupaj s predvidenimi roki vnesli v sistem JIRA, preko katerega je lahko naročnik spremljal proces. Za popis funkcionalnosti in potrebnih del smo potrebovali sedem delovnih dni. V sistemu JIRA so bili postavljeni tudi mejniki, ki so določali, kdaj se bodo izvajale redne objave posodobljene kode. Največje spremembe je bilo občutiti v samem razvoju, saj je potekal v parih. Potrebno je bilo napisati tudi več programske kode, saj se je za vsak del funkcionalne kode napisal test, ki je testiral ta del kode (angl. *unit test*). Ta del se je izkazal za časovno potratnega. Ko je namreč naročnik pri pregledovanju funkcionalnosti želel spremembo, je bilo poleg kode, ki je odgovorna za to spremembo, treba spremeniti še novi test, ki testira novo oziroma popravljeno funkcionalnost. Ker smo znižali tudi toleranco zastarele kode, smo imeli tudi s tem veliko več dela.

Programiranje v parih se je do določene mere izkazalo kot uspešen pristop, saj se je znanje med programerji hitreje prenašalo, zmanjšala pa se je celotna produktivnost razvojne ekipe. Verjetno lahko z neko gotovostjo trdimo, da se je zmanjšalo tudi število programerskih napak, a ker tega dejavnika nismo dodali med kriterije, s katerimi določamo uspešnost razvojnega modela, ga ne moremo upoštevati.

V poglobljenih intervjujih so bila mnenja glede vseh vprašanj zelo deljena. Najslabše je model ocenila razvojna ekipa, saj je ekstremno programiranje najmočneje posegalo v njihovo vsakdanje delo. Spremeniti so morali način razmišljanja, ki je moral biti skladen z vrednotami ekstremnega programiranja. Ravno zaradi močnega poudarka na vrednotah in načelih se je razvojni ekipi zdel model težak pri izvajanju, saj je nudil zelo malo oprijemljivih navodil. Večina navodil je bila abstraktnih. Pri vprašanju glede prilagodljivosti ni bilo nobenih pripomb. V izbranem podjetju so enotno menili, da je model zelo prilagodljiv v vseh fazah razvoja, kar je tudi skladno s teorijo, ki smo jo preučili (Thummadi et al., 2011).

Čas izvajanja projekta je bil zelo podoben kot pri spiralnem modelu, le da je bilo več časa porabljenega za programiranje in manj za načrtovanje. Število napak zaradi slabo definiranih zahtev je bilo primerljivo s spiralnim modelom. Pri številu zaustavitev procesa zaradi nejasnih zahtev sta bili prijavljeni 2 napaki več kot pri spiralnem modelu. Te so bile tudi posledica naročnikovega neupoštevanja dogovora, saj je eno izmed načel ekstremnega programiranja tudi to, da mora biti na strani naročnika vedno dosegljiva oseba, v primeru, če pride do nejasnih zahtev (Paulk, 2001). S tega stališča je model ekstremnega programiranja tudi težje izvajati, saj se mora določenih dogovorov držati tudi naročnik.

6 REZULTATI RAZISKAVE

V okviru magistrske naloge smo uspešno analizirali tri glavne razvojne modele in jih testirali v izbranem podjetju na konkretnih projektih. S tem smo poglobili razumevanje razvojnih modelov ter poglobili svoje znanje o managementu procesov. Na podlagi

preučitve literature smo spoznali, da je treba pri implementaciji zrelostnega modela upoštevati tudi velikost podjetja. Pri testiranju razvojnih modelov smo tako lahko izločili določene elemente, ki so manj primerni za manjša podjetja, kar je zmanjšalo kompleksnost implementacije razvojnega modela. S tem smo tudi povečali verjetnost uspeha same izvedbe. Pri testiranju razvojnih modelov so bili pomembni tudi demotivacijski razlogi implementacije izboljšave procesa izdelave programske opreme, saj smo bili nanje pripravljeni in smo jih lahko obvladovali. Pri vzpostavljanju prenovljenega procesa se je pozitivno izkazalo, da se je v središče procesa postavilo posameznika in ne obratno. Čeprav so v literaturi o tem mnenja deljena, smo se v izbranem podjetju odločili, da bomo proces prilagodili posameznikom, tako kot to svetuje Aaen (2003). Pomembno je tudi, da se pri definiranju procesa zajamejo pomembne značilnosti procesa in ne drobnih podrobnosti.

Pri testiranju razvojnih modelov smo največje razlike ugotovili med ekstremnim programiranjem in slapovnim modelom. Čeprav je že Royce želel slapovni model posodobiti in vanj implementirati številne lastnosti ekstremnega programiranja, so bile njegove ideje napačno razumljene, zaradi česar je slapovni model postal še bolj tog. Pri uporabi ekstremnega programiranja je veliko večja težava, ki smo jo pri testiranju tega modela potrdili, ta, da je potrebno spremeniti kulturo podjetja, saj model temelji na načelih in vrednotah zaposlenih. Te pa je veliko težje uvesti, zato bi lahko rekli, da nam ekstremnega programiranja ni uspelo v celoti testirati, saj bi za to morali spremeniti načela in vrednote zaposlenih.

Spiralni model se je izkazal kot zelo uravnotežen model. V primerjavi s slapovnim modelom pri njem ni bilo potrebno izpolniti tolikšne količine dokumentacije, kar je pomenilo, da se je delo hitreje začelo. Glavna prednost, ki smo jo prepoznali in ki je ostala dva modela nista imela, je način obvladovanja tveganja. Koncept, da se najprej začne razvijati najtežji in najbolj tvegan del programske opreme, se je izkazal za zelo učinkovitega. V povezavi z iteracijami razvojnih faz je ta koncept še toliko močnejši, saj se po izvedbi prvega prototipa znanje prenese v naslednjo iteracijo. Spiralni model te tako usmerja in na nek način prisili, da se že na začetku projekta začnejo reševati glavne težave projekta, saj se te veliko hitreje razkrijejo. Pri slapovnem modelu in ekstremnem programiranju pa se najbolj kritičen del velikokrat naredi na koncu, zato se izvedbeni roki prestavijo, kar lahko ogrozi celoten projekt.

Z implementacijo spiralnega razvojnega modela smo tako v izbranem podjetju povečali zrelost razvojnih procesov, kar je bil tudi cilj magistrske naloge. Temu sledi faza izboljšave procesa izdelave programske opreme z namenom, da bi se v izbranem podjetju model CMM dvignil na tretjo stopnjo.

SKLEP

Povečanje zrelosti razvojnega procesa je zelo kompleksen in dolgotrajen proces. Zelo pomembno je, da pri prenovi sodelujejo ključni nosilci procesa. Izkazalo se je, da je za dvig zrelosti razvojnih procesov potrebno sodelovanje vseh zaposlenih v podjetju. V nekaterih primerih je bilo potrebno v razvojni proces vključiti tudi naročnika. Na koncu testiranja razvojnih modelov smo v podjetju ugotovili, da je bilo zelo učinkovito najprej pregledati razvojne modele in kriterije, ki so pomembni za dvig zrelosti procesov. Še posebej za dobro se je izkazala odločitev za dejansko testiranje razvojnih modelov na projektih, saj smo tako vsi v podjetju dobili dober vpogled v njihovo delovanje. Pomembno je bilo, da smo tako tudi spoznali, kakšne so razlike v praksi. Pri izbiri razvojnega modela je sodelovalo celotno podjetje in tudi Spiralni model je bil odločitev vseh zaposlenih. S tem smo dosegli, da so zaposleni vzeli novi razvojni model za svojega, saj je bil tudi ta njihova odločitev. Pogosto se zgodi, da vodstvo vsili določen razvojni model, a je neuspešno implementiran, saj ga udeleženci (zaposleni) ne izvajajo, zato je zelo pomembno, da zaposleni vedo, da njihovo delo vpliva na razvojni proces in s tem povezan razvojni model (Yamamura, 1999).

Pri raziskavi smo bili omejeni s časom in sredstvi. Idealno bi bilo, če bi v podjetju oblikovali tri ekipe, vsaka bi uporabila en razvojni model in ga testirala vsaj 6 mesecev. Po tem obdobju bi lahko primerjali razvojni proces vseh treh ekip in tako bi učinkoviteje in natančnejše ugotovili, katere so prednosti in slabosti razvojnih modelov ter kakšni so njihovih učinki na dvig zrelosti razvojnih procesov. Na ta način bi tudi učinkoviteje testirali ekstremno programiranje, saj bi lahko v tem času ekipa spremenila vrednote udeležencev procesa, kar je bistveno za omenjeni razvojni model.

Uporaba modela CMM se je izkazala kot smiselna odločitev, saj smo s tem prihranili čas in trud, ki bi ju v nasprotnem primeru porabili, da bi sami ugotovili, katera področja so pomembna za dvig zrelosti razvoja programske opreme. Tudi če bi se odločili, da bomo sami raziskovali ključna področja, bi potrebovali več let, a vseeno bi zelo težko tako dobro definirali elemente, ki so pomembni za zrel razvoj programske opreme, saj so avtorji model CMM razvijali mnogo let. Ravno to je tudi namen modela CMM, in sicer, da podjetjem ni potrebno samim ugotavljati, katera področja so pomembna za zrel razvoj in zakaj. Naše ugotovitve pa so potrdile številne trditve v literaturi (Richardson & Von Wangenheim, 2007; Brodman & Johnson, 1997; Aaen, 2003; Sutton, 2000), da je za majhna podjetja model CMM potrebno prilagoditi. V izbranem podjetju smo to storili, kar je pospešilo celotno testiranje in implementacijo razvojnega modela, s tem pa posledično tudi dvig zrelosti razvojnih procesov v izbranem podjetju. Če bi model CMM implementirali brez modifikacij, bi bil postopek daljši in dražji, kar je posebej nevarno za majhna podjetja, saj nimajo toliko zalednega kapitala.

Po uvedbi spiralnega razvojnega modela in dvigu zrelosti razvojnih procesov je delo v

podjetju veliko bolj tekoče in nemoteno. Opazili smo tudi, da se vse več težav, s katerimi smo se soočali pred prenovo procesov, opazi vnaprej in se zato lahko nanje bolje pripravimo. Posledično smo lahko tako bolj samozavestni pri ocenah časa in stroškov, kar stranke tudi opazijo. S tem se tudi hitreje gradi obojestransko zaupanje. Lahko bi rekli, da so se pozitivni učinki implementacije druge stopnje modela CMM prelili tudi na druga področja, ki niso neposredno povezana s samim razvojem programske opreme.

LITERATURA IN VIRI

1. Aaen, I. (2003). Software process improvement: Blueprints versus recipes. *IEEE Software*, *20*(5), 86–93.
2. Abrahasson, P. (2002). Agile Software Development Methods: Review and Analysis. *VTT Publications, 478,* 107.
3. Agencija Republike Slovenije za javnopravne evidence in storitve. (2015). *Iskalnik poslovnih subjektov.* Najdeno 17. maja 2016 na spletnem naslovu https://www.ajpes.si
4. Ambler, S. (2005). *Quality in an agile world. Software Quality Professional, 7*(4), 34.
5. Avison, D. E., Lau, F., Myers, M. D., & Nielsen, P. A. (1999). Action research. *Communications of the ACM*, *42*(1), 94–97.
6. Avison, D., & Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools*. New York: McGraw Hill.
7. Baddoo, N., & Hall, T. (2003). De-motivators for software process improvement: an analysis of practitioners' views. *Journal of Systems and Software*, *66*(1), 23–33.
8. Beck, K. (2000). *Extreme programming explained: embrace change*. Melbourne: Addison-Wesley Professional.
9. Bell, T. E., & Thayer, T. A. (1976). Software requirements: Are they really a problem? *Proceedings of the 2nd international conference on Software engineering* (str. 61–68). Washington: IEEE Computer Society Press.
10. Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61–72.
11. Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed.* Melbourne: Addison-Wesley Professional.
12. Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., & Madachy, R. (1998). Using the WinWin spiral model: a case study. *IEEE Computer*, *31*(7), 33–44.
13. Brodman, J. G., & Johnson, D. L. (1997). A software process improvement approach tailored for small organizations and small projects (tutorial). *Proceedings of the 19th international conference on Software engineering, ACM* (str. 661–662). New York: Institute of Electrical and Electronics Engineers.
14. Brookes, N., & Clark, R. (2009). Using maturity models to improve project management practice. *POMS 20th Annual Conference*. Orlando: The Centre for Project Management Practice.
15. Brooks, F. (1987). *Defense science board task force report on military software*. Washington: Office of the Under Secretary of Defense for Acquisition.
16. Car, N. P. (2010). Poslovanje podjetij po dejavnosti in velikostnih razredih, podrobni podatki. Najdeno 5. junija 2016 na spletnem naslovu http://www.stat.si/StatWeb/glavna navigacija/podatki/prikazistaronovico
17. Cioch, F. A., Brabbs, J., & Kanter, S. (1994). Using the spiral model to assess, select and integrate software development tools. *Assessment of Quality Software Development Tools* (str. 14–28). New York: Institute of Electrical and Electronics Engineers.

18. Cockburn, A. (2006). *Agile software development: the cooperative game*. New Jersey: Pearson Education.
19. Cockburn, A., & Highsmith, J. (2001). *Agile software development: The people factor. Computer*, *34*(11), 131–133.
20. Coleman, G., & O'Connor, R. V. (2008). An investigation into software development process formation in software start-ups. *Journal of Enterprise Information Management*, *21*(6), 633–648.
21. Cooke-Davies, T. (2002). The "real" success factors on projects. *International journal of project management*, *20*(3), 185–190.
22. Dangle, K. C., Larsen, P., Shaw, M., & Zelkowitz, M. V. (2005). Software process improvement in small organizations: a case study. *IEEE Software, 22*(6), 68–75.
23. Davenport, T. H. (2013). *Process innovation: reengineering work through information technology*. Brighton, MA: Harvard Business Press.
24. Demmy, W. S., & Petrini, A. B. (1989). Statistical process control in software quality assurance. *Aerospace and Electronics Conference* (str. 1585–1590). New York: Institute of Electrical and Electronics Engineers.
25. Driscoll, D. L. (2011). Introduction to primary research: Observations, surveys, and interviews. *Writing Spaces: Readings on Writing*, *2*, 153–174.
26. Fahey, L., & Prusak, L. (1998). The eleven deadliest sins of knowledge management. *California management review*, *40*(3), 265–276.
27. Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development, 9*(8), *8*.
28. Glass, R. L. (1999). The realities of software technology payoffs. *Communications of the ACM*, *42*(2), 74–79.
29. Guide, A. (2004). *Project Management Body of Knowledge third edition (PMBOK GUIDE)*. Phoenix: Project Management Institute.
30. Hayes, J. (2014). *The theory and practice of change management*. New York: Palgrave Macmillan.
31. Howell, K. E. (2012). *An introduction to the philosophy of methodology*. London: Sage.
32. Humphrey, W. S. (1991). *Managing the software process (Hardcover).* Melbourne: Addison-Wesley Professional.
33. Irons, A. (2006). *Agile methods silver bullet or red herring.* Newcastle: Northumbria University.
34. Janieszewski, S., & George, E. (2003). *Six sigma and software process improvement*. Washington: PS&J Software Six Sigma.
35. Kaur, R., & Sengupta, J. (2013). Software process models and analysis on failure of software development projects. *International Journal of Scientific & Engineering Research, 2*(2), 1–4.
36. Kuilboer, J., & Ashrafi, N. (2000). Software process and product improvement: an empirical assessment. *Information and software technology*, *42*(1), 27–34.
37. Kumar, G., & Bhatia, P. K. (2014). Comparative analysis of software engineering

models from traditional to modern methodologies. *Fourth International Conference on Advanced Computing & Communication Technologies* (str. 189–196). New York: Institute of Electrical and Electronics Engineers.

38. Laesvirta, O., & Ribière, V. M. (2008). Km in a fast-growing global IT company: A case study. *VINE*, *38*(2), 254–266.
39. Li-li, Z., Lian-feng, H., & Qin-ying, S. (2011). Research on requirement for high-quality model of extreme programming. *International Conference on Information Management, Innovation Management and Industrial Engineering*. New York: Institute of Electrical and Electronics Engineers.
40. Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., & Zelkowitz, M. (2002). Empirical findings in agile methods. *Conference on Extreme Programming and Agile Methods* (str. 197–207). Berlin: Springer Heidelberg.
41. Liu, L., & Lu, Y. (2012). Application of agile method in the enterprise website backstage management system: Practices for extreme programming. *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on* (str. 2412–2415). New York: Institute of Electrical and Electronics Engineers.
42. ltd, A. (2016). Jira software - issue & project tracking for software teams. Najdeno 17. aprila 2016 na spletnem naslovu https://www.atlassian.com/software/jira
43. Mahanti, A. (2006). Challenges in enterprise adoption of agile methods-A survey. *Journal of computing and information technology, 14*(3), 197–206.
44. Mahanti, R., & Evans, J. R. (2012). Critical success factors for implementing statistical process control in the software industry. *Benchmarking: An International Journal*, *19*(3), 374–394.
45. Misra, S., Kumar, V., Kumar, U., Fantazy, K., & Akhter, M. (2012). Agile software development practices: evolution, principles, and criticisms. *International Journal of Quality & Reliability Management*, *29*(9), 972–980.
46. Nisse, D. (2000). Leadership, army style. *IEEE Software, 17*(82), 92–94.
47. Paulk, M. C. (1998). Using the software CMM in small organizations. *Pacific Northwest Software Quality Conference.* Portland: Oregon Convention Center.
48. Paulk, M. C. (2001). Extreme programming from a CMM perspective. *IEEE Software, 18*(6), 19–26.
49. Paulk, M. C. (2009). A history of the capability maturity model for software. *ASQ Software Quality Professional*, *12*(1), 5–19.
50. Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability maturity model (version 1. 1.). *IEEE Software, 1*(4), 18–27.
51. Redmill, F. (1990). Considering quality in the management of software-based development projects. *Information and Software Technology*, *32*(1), 18–22.
52. Reifer, D. J. (2002). *How good are agile methods? IEEE Software, 19*(4), 4.
53. Richardson, I., & Von Wangenheim, G. C. (2007). Guest Editors' Introduction: Why are Small Software Organizations Different? *IEEE Software, 24*(1),18–22.

54. Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON* (str. 3285–338). New York: Institute of Electrical and Electronics Engineers.
55. Sharma, B. P., & Sharma, N. (2009). Software Process Improvement: A Comparative Analysis of SPI models. *Emerging Trends in Engineering and Technology (ICETET)* (str. 1019–1024). New York: Institute of Electrical and Electronics Engineers.
56. Shen, B., & Ruan, T. (2008, December). A case study of software process improvement in a chinese small company. *Computer science and software engineering*, *12*.
57. Silva, F. S., Soares, F. S. F., Peres, A. L., de Azevedo, I. M., Vasconcelos, A. P. L., Kamei, F. K., & de Lemos Meira, S. R. (2015). Using CMMI together with agile software development: A systematic review. Information and Software Technology. *Information and Software Technology*, *58*, 20–43.
58. Software, E. (2001). The trac project - integrated scm and project management. Najdeno 15. februarja 2016 na spletnem naslovu https://trac.edgewall.org/
59. Sommerville, I. (1996). Software process models. *ACM Computing Surveys (CSUR)*, *28*(1), 269–271.
60. Succi, G., Stefanovic, M., & Pedrycz, W. (2001). Quantitative assessment of extreme programming practices. *Electrical and Computer Engineering* (str. 81–86). New York: Institute of Electrical and Electronics Engineers.
61. Sutton, S. M. (2000). The role of process in a software start-up. *IEEE Software,* 33–39.
62. Thummadi, B. V., Shiv, O., & Lyytinen, K. (2011). Enacted routines in agile and waterfall processes. *Agile Conference (AGILE)* (str. 67–76). New York: Institute of Electrical and Electronics Engineers.
63. Tong, G. (1994). Software development process improvement: The forgotten son? *World Class Design to Manufacture*, *1*(5), 21–25.
64. Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. *arXiv preprint arXiv, 1409, 6600.*
65. Van Rossum, G., Warsaw, B., & Coghlan, N. (2001). Pep 8 – style guide for python code. Najdeno 27. maja 2016 na spletnem naslovu https://www.python.org/dev/peps/pep-0008
66. Voas, J. (1999). Advice for those bitten by the startup bug. *IT professional*, *1*(3), 38–45.
67. Weissfelner, S. (1999). ISO 9001 for software organizations. *Elements of Software Process Assessment and Improvement* (str. 77–100). California: IEEE Computer Society.
68. Yamamura, G. (1999). Process improvement satisfies employees. *IEEE Software, 16*(5), 83–85.
69. Zahran, S.(1998). *Software process improvement: practical guidelines for business susccess*. Melbourne: Addison-Wesley Longman Ltd.