UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

DIPLOMSKO DELO

UPORABA AGILNIH METOD PRI ORGANIZIRANJU RAZVOJA PROGRAMSKIH REŠITEV

Ljubljana, avgust 2015

BOR BRENTIN

IZJAVA O AVTORSTVU

Spodaj podpisani Bor Brentin, študent Ekonomske fakultete Univerze v Ljubljani, izjavljam, da sem avtor diplomskega dela z naslovom Uporaba agilnih metod pri organiziranju razvoja programskih rešitev**,** pripravljene v sodelovanju s svetovalcem prof. dr. Tomaž Turk.

Izrecno izjavljam, da v skladu z določili Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 21/1995 s spremembami) dovolim objavo diplomskega dela na fakultetnih spletnih straneh.

S svojim podpisom zagotavljam, da

- je predloženo besedilo rezultat izključno mojega lastnega raziskovalnega dela;
- je predloženo besedilo 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 v diplomskem delu, citirana oziroma navedena v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, in
    - pridobil vsa dovoljenja za uporabo avtorskih del, ki so v celoti (v pisni ali grafični obliki) uporabljena v tekstu, in sem to v besedilu tudi jasno zapisal;
- se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku (Ur. l. RS, št. 55/2008 s spremembami);
- se zavedam posledic, ki bi jih na osnovi predloženega diplomskega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom.

V Ljubljani, dne _____ Podpis avtorja:_____

1

KAZALO

KAZALO SLIK

KAZALO TABEL

UVOD

Hitri razvoj tehnološkega trga od dobaviteljev zahteva hitro odzivnost. Kupcem oziroma naročnikom je potrebno zagotovitvi izdelke v tistem trenutku, ko jih potrebujejo oziroma želijo, morajo pa biti tudi cenovno dovolj ugodni. Podjetja oziroma dobavitelji, ki to lahko zagotovijo, imajo na trgu prednost pred ostalimi ponudniki. Zato je tekma za zagotavljanje te prednosti tudi tekma v iskanju vedno novih delovnih procesov, prilagoditev v poslovanju podjetja in sprememb tehnologije, s katerimi zagotavljajo zanesljivost izvedbe celotnega projektnega organiziranja. V programskem inženiringu se uporabljajo pristopi, prisotni že iz 60. let. Programski inženirji danes povezujejo različne pristope v sodobne metode organiziranja projektov razvoja programskih rešitev.

Klasične metode, ki jih najpogosteje povezujemo s slapovnim načinom razvoja, so predstavljale dokaj dobro strukturo za organiziranje inženirskih projektov razvoja programskih rešitev kljub temu, da je v praksi zaradi samega načina organiziranja projektov pogosto prišlo do zastojev ali pa so projekti ostali nezaključeni.

Pri razvoju programskih rešitev se je pokazalo, da podjetja ob uporabi klasičnih metod programskih rešitev veliko časa porabijo za načrtovanje celotnega procesa. Pogosto je potrebno programsko rešitev v fazi razvoja – še posebej, če ta traja dalj časa – zaradi različnih razlogov še nekajkrat spremeniti oziroma popraviti. Pogosto se je med razvojem programske rešitve ugotovilo, da bo koda, ki naj bi jo končna programska rešitev vsebovala, prekompleksna in da je ne bo mogoče razviti, zato je bilo potrebno projekt tudi opustiti.

Leta 2001 so se v mestu Snowbird (Utah, ZDA) zbrali izkušeni inženirji programske opreme in napisali »Agilni manifest« (Manifesto for Agile Softare Development, 2001). Manifest je nabor principov in postopkov novih metod organiziranja razvoja programskih rešitev, ki so jih poimenovali agilne metode. S pomočjo agilnih metod naj bi končni rezultat pri razvoju programskih rešitev dosegel večjo funkcionalno kakovost v krajšem časovnem obdobju. O agilnih metodah je bilo do sedaj veliko napisanega.

Poglavitne prednosti, zaradi katerih naj bi se podjetja, ki se ukvarjajo z razvojem programskih rešitev, odločila za uporabo agilnih metod, so poenostavljanje procesov razvoja, večja pripravljenost na spremembe, samoorganizacijski timi in večja vloga naročnika pri sami organizaciji in razvoju projekta. Menim, da je odločitev za uporabo agilnih metod za

novonastala podjetja dobra odločitev, medtem ko je prevzem agilnih metod v podjetjih, ki uporabljajo klasične metode razvoja programskih rešitev, zahtevnejši organizacijski zalogaj.

Cilj

Ker menim, da je odločitev nekega podjetja za uporabo agilnih metod (še posebej, če je to do tedaj uporabljalo le klasične metode organizacije razvoja programskih rešitev) smela in da v celoti obrne na glavo dotedanjo prakso organizacije in način dela podjetja, sem se osredotočil ravno na to področje. Zanimalo me je, kaj je potrebno narediti, da se ta odločitev pokaže za optimalno v razvoju podjetja, kaj gre lahko narobe, katere so možne napake in tveganja ter kaj pomeni nova oblika organizacije notranjega okolja in načina dela.

V nalogi proučujem vprašanje, kako se mora podjetje, ki se ukvarja z razvojem programskih rešitev za znane naročnike, organizacijsko prilagoditi, da izvede projekt razvoja programske rešitve s pomočjo agilnih metod. Svoje ugotovitve sem zapisal na podlagi raziskave knjig, strokovnih publikacij in člankov. Ker so agilne oblike organiziranja projektov razvoja programskih rešitev najbolj opazna novost v zadnjih desetih letih, me je zanimalo, kaj sploh pomeni »biti agilen« in zakaj naj bi se podjetje odločilo voditi projekte s pomočjo agilnih metod. Ugotovil sem, kako se morajo podjetja, ki so se odločila, da bodo za razvoj programskih rešitev uporabljala agilne metode, organizirati, kako se spremembe v organiziranju kažejo v notranjem okolju in strukturi notranjega okolja, kakšna je vloga vodje projekta v podjetju, kjer so timi samoorganizacijski, in tudi, kako se izbere in organizira uspešen tim ter kakšno vlogo ima pri razvoju programskih rešitev naročnik.

Metode dela

Naloga je analiza del, za katero sem črpal informacije iz del različnih avtorjev knjig, člankov, strokovnih publikacij in spletnih dokumentov. Poskušal sem izpostaviti značilnosti agilnih metod, ki jih najpogosteje opisujejo ti avtorji. V diplomskem delu sem se osredotočil na projekte razvoja kompleksnejših programskih rešitev, ki od podjetja zahtevajo projektno organiziranje, torej programske rešitve, ki so namenjene poslovnim subjektom in so izdelani po naročilu in želji naročnika. S tem sem se izognil strategijam, kot sta marketing in partnerstvo, in tako večjo pozornost posvetil strategijam organiziranja projekta.

V delu predvidevam, da se programsko rešitev razvija za znanega kupca v obliki podjetja ali

organizacije. Kupca v nadaljevanju imenujem »naročnik«. Za podjetje, ki razvija del ali celoto programske opreme oziroma informacijskega sistema po naročilu, v nadaljevanju uporabljam izraz »podjetje«.

V svojem diplomskem delu najprej kratko predstavim program oziroma programsko rešitev kot produkt, saj se ta razlikuje od klasičnih inženirskih produktov. To je osnova za krajšo predstavitev značilnosti klasičnih metod, večjo pozornost pa posvečam predstavitvi agilnih metod. V nadaljevanju s pomočjo raziskav različnih avtorjev prikažem razlike med klasično in agilno vodenimi projekti razvoja programskih rešitev. Različne primerjave med klasičnimi in agilnimi metodami uporabljam skozi celotno diplomsko delo. V zadnjem delu se osredotočam na organizacijo podjetja, ki vpliva na organizacijo projekta razvoja programske rešitve s pomočjo agilnih metod. Podrobneje predstavim tudi različne vloge, ki jih imajo deležniki, soudeleženi v projektu, in kako vplivajo na uspeh projekta.

1 LASTNOSTI PROGRAMSKIH REŠITEV

Kittlaus (2004, str. 9) opisuje programsko rešitev ali program kot opredmeteno ekonomsko dobrino, ki pa se je ne da prijeti in ni vidna ali v kakršnekoli obliki opazna. Potrebujemo jo, da lahko uporabljamo računalnike oziroma strojno opremo. Opazna je na primer lahko le funkcionalnost programa, uporabniški vmesnik ali pa rezultat programsko vodene transakcije pri bančnih transakcijah. Programske rešitve so sestavljene iz jasno definiranih navodil, ki pri zagonu dajejo navodila strojni opremi, da izvede funkcijo, za katero je bil ustvarjen (Software, 2013). Čeprav so programske rešitve sicer res neotipljive in jih dojemamo bolj intuitivno, jih ne moremo primerjati z zrakom ali kakšno drugo nematerialno dobrino, meni Kittlaus (2004). Kot pri večini visokotehnoloških produktov tudi pri programskih rešitvah večina uporabnikov ne ve, kako delujejo. Lahko pa podjetju, ki jih uporablja, predstavljajo glavno sredstvo za doseganje profita, kot pravi. Razvoj kompleksnejših programskih rešitev je lahko povezan z zelo visokimi stroški, zato je pomembno, da se programske rešitve obravnava kot opredmetena sredstva, da jih lahko tudi amortiziramo. Prepričan je, da je potrebno programske rešitve dojemati drugače, predvsem s pravnega in prodajnega vidika, ker se neotipljivih stvari ne da obdavčiti ali prodajati. Programske rešitve so zato popolno nasprotje od drugih investicijskih ali uporabniških dobrin.

Posebnost je tudi, da kupec pravzaprav nikoli ne dobi produkta. Dobi le točno definirane in točno določene licenčne pravice. Kljub temu so investicije v programske rešitve v celotni

infrastrukturi informacijske tehnologije velike. Programske rešitve so sestavljene iz človeškega znanja, zapisanega v kodi, in imajo neprecenljivo lastnost, in sicer da se jih lahko zelo hitro kopira in hitro prenaša preko velikih razdalj. Kittlaus (2004, str. 18) opozarja tudi na specifične lastnosti programskih rešitev. Loči jih glede na:

- trg, na katerem so prisotni:
    - potrošniki (v nadaljevanju B2C) - primer uporabe so računalniške igre;
    - poslovni subjekti (v nadaljevanju B2B) – ločimo:
        - horizontalne (v panogi), primer je sistemski program,
        - vertikalne (specifične glede na panogo) - primer je aplikacija za trgovino z vrednostnimi papirji.
- funkcionalno področje, kot na primer: sistemski program, vmesno programje (angl. *Middleware*) in aplikacije;
- razvojno področje, kot na primer standardni programi, komponentni programi (angl. *Component software*) ali individualni razvoju;
- pogoje:
    - licenčni pogoji, primeri so: odprta koda (angl. *open Source*), brezplačna programska oprema (angl. *Freeware*), programiranje na poskus (angl. *shareware*) in plačljive licence,
    - razvoj v vrednosti plačila.

V svojem diplomskem delu sem se usmeril v delitev na programske rešitve, ki so namenjene B2B in ne toliko na tiste, ki so namenjene masovnemu trgu in so prilagojene za večjo količino končnih uporabnikov oziroma B2C. Kot pravi Kittlaus (2004), ima delitev na B2B in B2C lahko tudi skupni tok, saj so lahko operacijski sistemi ali programski paket Office uporabni tako v industriji kot za osebno uporabo posameznega potrošnika. Seveda pa lahko razvijalec programskih rešitev loči uporabne funkcije in jih prilagodi posamezni skupini uporabnikov. Primer je Windows, ki v svojem operacijskem sistemu omejuje funkcionalnosti in ga označuje z različnimi verzijami, te pa se prodajajo po različnih cenah. Med B2B in B2C so razlike velike. Programske rešitve, namenjene B2B, morajo biti v veliki meri prilagojene potrebam poslovnega subjekta. Število strank je manjše, so pa namestitve programskih rešitev na omreženih sistemih v podjetjih z velikim številom uporabnikov mnogo kompleksnejše. Kot pravi Kittlaus (2004), je naročniku programskih rešitev izdatek za prilagoditev in namestitev programske rešitve pogosto enak izdatku za razvoj programske rešitve, lahko pa ga celo presega, saj gre pri namestitvah kompleksnih programskih rešitev za procese, ki lahko

potekajo tudi več mesecev. Po končani namestitvi je nujno zagotoviti vzdrževanje nameščene programske rešitve, nemalokrat pa jo je potrebno tudi popraviti in dopolniti. Programske rešitve, namenjene osebni rabi, so prilagojene uporabi večjega števila posameznikov. Lahko so nameščene v nekaj minutah in takoj pripravljene za uporabo.

Pri razvoju programskih rešitev gre za avtorsko delo, zato jo je smiselno opredeliti kot storitev, saj gre za razvoj novih funkcij, funkcij in konceptov (Hogg, 2013). Vendar pa se lahko programsko rešitev patentira, kar ji med drugimi daje lastnost produkta. Kaj je torej programska rešitev in kako jo obravnavamo?

Veliko programskih rešitev lahko uvrstimo med produkte. Večina jih je namenjena B2C, to so igrice, operacijski sistemi, orodja Office in druge, ki jih kupimo, namestimo na svoj računalnik in uporabljamo. Pri razvoju, kot ga opisujem v nadaljevanju in je namenjen B2B, pa težko govorimo o produktu, saj je razvoj programske rešitve bolj podoben delu mehanika, ki težko fiksira strošek storitve servisa, saj se potrebe odkrivajo sproti. Tudi Meyer (2001) je zaključil, da so programske rešitve vedno bolj storitev, kljub poiskusom, da bi programiranje spremenili v objekte, ki bi na nek način sestavljali programske rešitve kot na tekočem traku. Podjetja, ki razvijajo programske rešitve, pa se zaradi veliko večjega tržnega deleža in prednosti, ki jih prinaša, vedno pogosteje poslužujejo »programskih rešitev kot storitev« (angl. *Software as a service*) ali skrajšano »SaaS«. To je nova oblika licenciranja in dobave programskih rešitev z naročniškimi razmerji. Baze so locirane na oblakih neodvisnih programskih ponudnikov (angl. *independent software vendor*) ali aplikacijskih programerskih vmesnikih (angl. *application service provider*), (Software as a service, 2013). Ta oblika distribucije daje poslovanju podjetjem, ki se ukvarjajo z razvojem programskih rešitev, večjo fleksibilnost, nižje cene, predvsem pa olajša končnemu kupcu uporabo. V svojem delu se osredotočam na projekte razvoja programskih rešitev po naročilu, zato načinov distribucije v nadaljevanju ne omenjam več.

Tako kot sta si medsebojno različni ciljni skupini, za katere podjetja, ki se ukvarjajo z razvojem informacijski tehnologij, razvijajo programske rešitve, tako so različne tudi poslovne in strateške usmeritve podjetij. Kittlaus (2004) pravi, da je za razvoj programskih rešitev B2P najpomembnejša strategija marketinga, sledi ji strategija partnerstva, na tretjem mestu pa je odločitev, ali in kako hitro plasirati produkt na trg. Pri poslovno orientiranih ali B2B-programskih rešitvah je na prvem mestu partnerska strategija, sledi ji strategija storitev, marketing pa se nahaja na trejem mestu.

Razvoj programskih rešitev je skupek kompleksnih aktivnosti, ki morajo združiti veliko variabilnih spremenljivk. Poleg tega jim moramo dodati še negotovost, ki jo prinašajo udeleženci v procesu. Za organizacijo projekta razvoja programske rešitve so se v desetletjih razvile različne metode, s katerimi poizkušajo podjetja dosegati višje poslovne uspehe projektov razvoja programskih rešitev. Izbira metod za organiziranje razvoja programskih rešitev pripomore k temu, da podjetja lažje in hitreje razvijejo uporabno kodo, ki končnemu uporabniku oziroma naročniku predstavlja vrednost.

2 METODE ORGANIZIRANJA RAZVOJA PROGRAMSKIH REŠITEV

Metode organiziranja razvoja programskih rešitev v programskem inženiringu pomenijo okvir, ki se ga uporablja za strukturo, načrt in kontrolo procesa razvoja programske rešitve (Software Development Methodology, 2013). Razvoj programskih rešitev je del industrije, ki je podvržena zelo hitrim spremembam. Je ena novejših industrij in ima tendenco, da se v branžo vključujejo mladi, kreativni in inovativni ljudje. Če združimo vse značilnosti, ugotovimo, da je programiranje razvijajoča se disciplina. Potreba po metodah je prišla s strani vodstva in svetovalcev in ne iz potrebe razvijalcev, ki metode ponavadi vidijo kot oviranje njihove svobode, kreativnosti in produktivnosti (Kittlaus, 2012).

V tem poglavju bom predstavil, kako in zakaj so metode nastale, komu so namenjene in zakaj se podjetja odločajo za izbiro agilnih metod organizacije razvoja programskih rešitev ter zakaj in v katerih primerih se podjetju, ki že razvija programske rešitve po klasičnih metodah, prehod na eno od agilnih metod izplača.

2.1 Pristopi k izbiri metode za organizacijo razvoja programskih rešitev

V inženiringu se uporablja razvoj s pomočjo življenjskega cikla razvoja projekta. Ta način predvideva, da so na voljo vse informacije, ki jih potrebujemo za uspešno izvedbo projekta. Vsi projekti se začnejo z načrtom izvedbe, ki mu sledi razvoj inženirskega produkta. Za izdelavo detajlnega načrta potrebuje podjetje zanesljive in kreativne ljudi, ki morajo imeti na voljo veliko količino informacij, da lahko predvidijo možne zastoje in napake. Pri splošnem inženiringu so ponavadi na voljo vse informacije, zato je strošek izdelave načrta nizek; verjetno znaša okoli 10 odstotkov celotnega stroška projekta. Pri programskem inženiringu pa

ponavadi zaradi pomanjkanja informacij in težko predvidljivih situacij, predvsem pa potreb po spremembah med samim razvojem, ta strošek lahko predstavlja tudi do 50 odstotkov celotnih stroškov razvoja. Tem stroškom pa se pridružijo še stroški testiranj, ki jih je potrebno izvesti ob koncu projekta, in so ponavadi visoki, saj je testiranje ena od zahtevnejših in težko izvedljivih faz programskega inženiringa. Zaradi teh razlik je programski inženiring drugačen in podjetja, ki se ukvarjajo z razvojem informacijske tehnologije, iščejo načine, kako standardizirati razvoj programskih rešitev. Potrebe po bolj discipliniranem razvoju programskih rešitev z usmeritvijo k večji predvidljivosti in učinkovitosti razvoja so vodile v razvoj metod, ki omogočajo večjo preglednost napredovanja projekta in posledično izboljšajo možnost uspeha projekta, pripomorejo k zmanjševanju stroškov izdelave načrta razvoja programske rešitve in k večji kakovosti napisane kode. Metode imajo svoje značilnosti, zagovornike in tudi nasprotnike. Večina metod izhaja iz osnovnih pristopov razvoja programskih rešitev ali programskega inženiringa.

V sodobnem svetu je povsem sprejemljivo, da naročniki nenehno spreminjajo svoje potrebe po predmetih in funkcijah naročene programske rešitve, lahko pa se spremenijo tudi razmere na ekonomskih, tehničnih, zakonodajnih ali drugih področjih. Kar je sedaj dobro, je lahko čez pol leta popolnoma neuporabno. Večji, kot je projekt, večja je količina sprememb. Agilne metode so se razvile predvsem zaradi vedno večjih razlik med programskimi zmogljivostmi in pričakovanji naročnikov.

Fowler (2005, str. 2) v svojem delu razdeli metode na inženirske oziroma klasične, ki so vodene plansko (angl. *Plan driven*), in agilne. Klasičnim ne pripisuje opaznih uspehov, ker meni, da so birokratsko zahtevne, kar upočasnjuje celoten razvoj programskih rešitev. Fowler (2006) pravi, da so agilne metode nastale kot odgovor na slabosti klasičnih, saj uporabljajo ravno toliko procesa, da lahko zvišajo kakovost programske rešitve.

Katero metodo bo podjetje izbralo za doseganje zastavljenih ciljev, je odvisno od tega, kaj podjetju ustreza glede na notranje okolje in tip projekta. S klasičnimi in agilnimi metodami podjetja izvajajo vrhunske projekte, za katere pa izbira druge metode ne bi bila primerna. Izbrana metoda vpliva na rezultat projekta, preseganje stroškov in tudi na zaključek projekta (Goldstein & Petrie, 1993). Možno je, da bi podjetje lahko za različne projekte uporabilo različne metode, a je tako zamenjevanje povezano z veliko učenja, prilagoditvami v organizaciji in velikimi denarnimi vložki.

2.2 Klasične metode

Razvoj programskih rešitev je bil na začetku zelo individualno pogojen. Veljalo je pravilo »piši in popravi« (angl. *code and fix*). To pravilo zasledimo tudi danes pri razvoju manj kompleksnih programskih rešitev. Projekti, ki so postajali vedno bolj kompleksni, pa niso mogli biti več izvedeni posamezno ali nestrukturirano. Z razvojem tehnologije je napredovala diverzifikacija uporabe programskih rešitev na skorajda vseh področjih. Pojavile so se potrebe po kompleksnejših programih, te pa lahko razvijajo timi, ki sledijo vzorcu, določenem s strani vodstva.

Kot odgovor na organiziranje razvoja programskih rešitev vedno večjega števila velikih in zahtevnih projektov so se v začetku 70. let razvile nove metode, ki jih danes imenujemo klasične metode. So tipični predstavniki linearnega pristopa. Po mnenju Benediktssona, Dachlerja in Thorbergssona (2004) so klasične metode sestavljene iz treh procesov, ki si med seboj delijo informacije: načrtovanje, nadziranje, izvajanje.

Klasične metode temeljijo na življenjskem ciklu (angl. *SPLC Software Product Life Cycle* in *SDLC System Development Life Cycle*) in najpogosteje je izbran način organiziranja razvoja programskih rešitev s tako imenovanim slapovnim načinom, pri katerem si faze sledijo ena za drugo. Uporaba besedne zveze »slapovni način« nekako opisuje neponovljivost sledenja faz. Po prehodu razvoja v naslednjo fazo življenjskega cikla se ni več mogoče vrniti na prejšnje faze razvoja.

Postopki, ki so značilni za klasične metode, zahtevajo veliko časa in denarja; obravnava se jih za razvojno ponavljanje. Dejansko se program v klasičnih metodah piše dvakrat, najprej na papirju in pozneje s programiranjem.

Prva naloga vodij projekta organizacije razvoja programske rešitve pri klasičnih metodah je sestava načrta in urnika ter sprotno kontroliranje procesa, ki sledi načrtu. Klasične metode obravnavajo načrt kot natančni oris procesa, kar pripelje do lažje izvedljivosti. Kontrola se izvaja s popravki, načrt pa ne dopušča učenja novih stvari, ki bi lahko pripeljale do izboljšanja projekta. Tako razmišljanje kaže na to, da vsak dober projektni vodja vsaj teoretično lahko vodi katerikoli projekt razvoja programske rešitve, ne glede na to, koliko se na vsebino projekta spozna, le slediti mora navodilom. Največkrat omenjene težave v procesih klasično organiziranih projektov razvoja programskih rešitev so:

- Motivacija ne prihaja od znotraj. Projekt se že v začetku načrtuje, tako da zadovolji različne zahteve strank oziroma naročnikov. Projektni vodja težko sam vloži svojo iniciativo pri projektu.
- Motivacija za načrtovanje je pogosto bolj rezultat želje po kontroli kot po končnem izdelku.
- Načrtovanje in kontrola sta osrednji točki organizacije.

V ospredju je fokus na inženirskem vodenju (angl. *management of engineering*). Proces realizacije projekta poteka v zaporedju faz od analize zahtev, načrtovanja in arhitekture, kodiranja, testiranja do vzdrževanja, vse pa je podprto s podrobno dokumentacijo. Ponavadi se naslednja faza začne šele, ko se prva konča (Kittlaus & Herde, 2011).

Klasične metode ali metode življenjskega cikla, ki so v 70. letih veljale za idealne, se danes povezuje z visokim tveganjem za neuspeh projekta in nizko produktivnostjo. Sčasoma se je pokazalo, da niso primerne za kompleksne in inovacijske projekte. Prav tako ne ustrezajo vedno večji dinamiki v razvoju informacijske tehnologije, ki od vodje projektov zahteva vedno večjo fleksibilnost. Danes je povsem običajno, da dve ali tri leta trajajoči projekti razvoja programskih rešitve zaradi tehnološkega in ekonomskega razvoja ter novih naročnikovih informacij doživljajo številne spremembe in popravke.

V današnjem času se spreminja tudi vloga naročnika. Pri klasičnih metodah se naročnik pojavlja bolj kot opazovalec, ki ga razvijalci vključijo pri začetnem pogovoru, na katerem določajo potrebne predmete in funkcije programskih rešitev. Možno je celo, da program pri implementaciji ne izpolnjuje naročnikovih pričakovanj.

Klasične metode so doživele veliko prilagoditev in sprememb. Današnje klasične metode se bistveno razlikujejo od tistih, ki so jih uporabljali pred več desetimi leti. Še vedno pa jih uporablja veliko podjetij (Benediktsson, Dachler, & Thorbergsson, 2005).

2.3 Agilne metode

Zaradi težav, ki so se pokazale pri klasičnih metodah, so začeli posamezniki premišljevati o novih metodah. Tako imenovane "lahke" metode (angl. *lightweight*) so bile predhodnica razvoja agilnih metod; te temeljijo na pomenu, ki ga opisuje izraz agilnost: spretnost, gibčnost, živahnost, delavnost (Verbinc, 1997).

Organizacijsko okolje podjetja se mora hitro prilagoditi na pričakovane in nepričakovane spremembe strank oziroma naročnikov, konkurence in trga (Highsmith, 2003). Bistvo agilnih metod pa je ravno ustvarjanje sprememb in odzivanje nanje. Ustvarjanje sprememb je tisto, kar zmede konkurenco, hkrati pa pomeni razvoj novih produktov, ustvarjanje novih tržnih poti, zmanjševanje proizvodnega časa in izdelavo programskih rešitev po naročilu za precej manjši tržni segment (Agilne metode razvoja programske opreme, 2014). Agilnost pomeni dinamičnost, vsebinsko eksaktnost, agresivno zaobjemanje sprememb in orientiranost k rasti projekta. Fowler (2005) pravi, da so agilne metode nastale zaradi prevelike birokracije klasičnih metod, ki upočasnjuje proces.

Dandanes agilne metode močno prevladujejo na trgu razvoja programskih rešitev. Leta 2012 je, kot kažejo rezultati raziskave Versionone (2012), imelo znanje o agilnih metodah že 81 % vprašanih. 25 % vprašanih je odgovorilo, da se z njimi srečujejo že več kot 5 let, ostali manj. Iz navedb v strokovni literaturi sklepam, da so osnove agilnih metod razvoja programskih rešitev nastale kot odgovor na japonske "lean" proizvodne metode, ko so znanstveniki pod okriljem ameriškega Ministrstva za obrambo (angl. *US Department of Defense*) v 80. letih izvajali raziskave uporabe agilnih metod proizvodnih procesov. Prvič pa se ime agilne metode navaja v manifestu (Manifesto for Agile Software Development, 2001), ki ga je leta 2001 na srečanju v Snowbirdu (Utah, ZDA) sestavilo 17 izkušenih inženirjev programske opreme. V Agilni manifest so zapisali principe in postopke, ki bi jih morali upoštevati vsi uporabniki agilnih metod.

Temeljna načela agilnih metod so (Agilne metode razvoja programske opreme, 2014):

- Vodenje projektov temelji na konstantnem prilagajanju in kontroliranju. Filozofija vodenja je takšna, da spodbuja timsko delo, samoorganizacijo in odgovornost.
- Niz najboljših inženirskih praks, ki omogoča hiter razvoj kakovostne programske rešitve, inposlovni pristop z večjo integracijo naročnikovih potreb in ciljev podjetja.
- Agilne metode so prilagodljive namesto predvidljive. Klasične metode načrtujejo razvoj projekta natančno, kar je dobro, dokler ne pride do sprememb. Spremembam se torej izogibajo. Agilne metode pa spremembe pozdravljajo, trudijo se jim prilagajati in v njih iskati izzive.
- Agilne metode so orientirane k ljudem namesto k procesu. Pri klasičnih metodah se trudijo najti proces razvoja, ki deluje dobro ne glede na to, kdo je uporabnik. Osnovna

teorija agilnih metod pa trdi, da ne more noben proces nadomestiti strokovnega znanja razvojnega tima, zato služi proces za podporo razvojnemu timu in njihovemu delu.

Nekaj najbolj zagnanih avtorjev agilnih teorij je za promoviranje agilnih metod ustanovilo celo neprofitno organizacijo "The Agile Alliance" (Manifesto for Agile Software Development, 2001), kjer poudarjajo naslednje prednosti:

- posameznik in interakcija namesto procesov in orodij;
- delujoč program namesto celovite dokumentacije;
- sodelovanje z naročniki namesto pogodbenih pogajanj;
- odzivanje na spremembe namesto sledenja načrtu.

Pri pojasnjevanju agilnih metod organizacije razvoja programskih rešitev se največkrat srečamo z izrazi: ponavljajoče (angl. *iterative*), postopno (angl. *incremental*), sodelovanje med naročniki in izvajalci, preprostost in prilagodljivost, ki opisujejo posamezne dele organizacije razvoja programskih rešitev pri agilnih metodah, *timebox*, *backlog* itd.

Pristop s ponovitvami ali iteracijami predstavlja prehod skozi vse aktivnosti procesa razvoja, analize, oblikovanja, načrtovanja testiranj in testa. Idealno traja dva do tri tedne. Zanimivo je, da podjetja pri razvoju nove programske rešitve pogosto niti ne vedo, katere lastnosti bodo vključila v naslednji ponovitvi; popolne vizije o končanem projektu pravzaprav ni. To pa ne pomeni, da ni vizije. Dobro mora biti znano, katere nujne predmete in funkcije naročena programska rešitev potrebuje.

Vodstvo, ki uporablja agilne metode organizacije razvoja programskih rešitev, za projekte še vedno pripravlja načrte izvedbe, ki pa niso detajlni. Vsaka dva ali tri tedne, odvisno od tega, kako si podjetje postavi časovni načrt, je potrebno zagotoviti testirano različico, ki jo praviloma preveri tudi naročnik programske rešitve. Povratne informacije naročnika so za nadaljnjo izvedbo projekta zelo pomembne. Šele ko je ponovitev končana in so izvedeni vsi testi, se načrtujejo funkcije, ki jih bodo programerji oziroma razvijalci razvijali v naslednji ponovitvi. Ti načrti so vezani na trenutno situacijo in ne na situacijo z začetka projekta, zato so bolj realni in prilagojeni. Z načrtom ponovitev prepoznavamo količino dela, ki je potrebna za razvoj sklopa funkcij v določenem časovnem okvirju (angl. *timebox*). Časovni okvir (v nadaljevanju *timebox*) predstavlja okvir, v katerega se zlaga predmete in funkcije, ki se jih

kodira v točno določenem časovnem obdobju oziroma ponovitvi. Različne agilne metode lahko časovni okvir poimenujejo tudi drugače, pri metodi Scrum so ga poimenovali “sprint”.

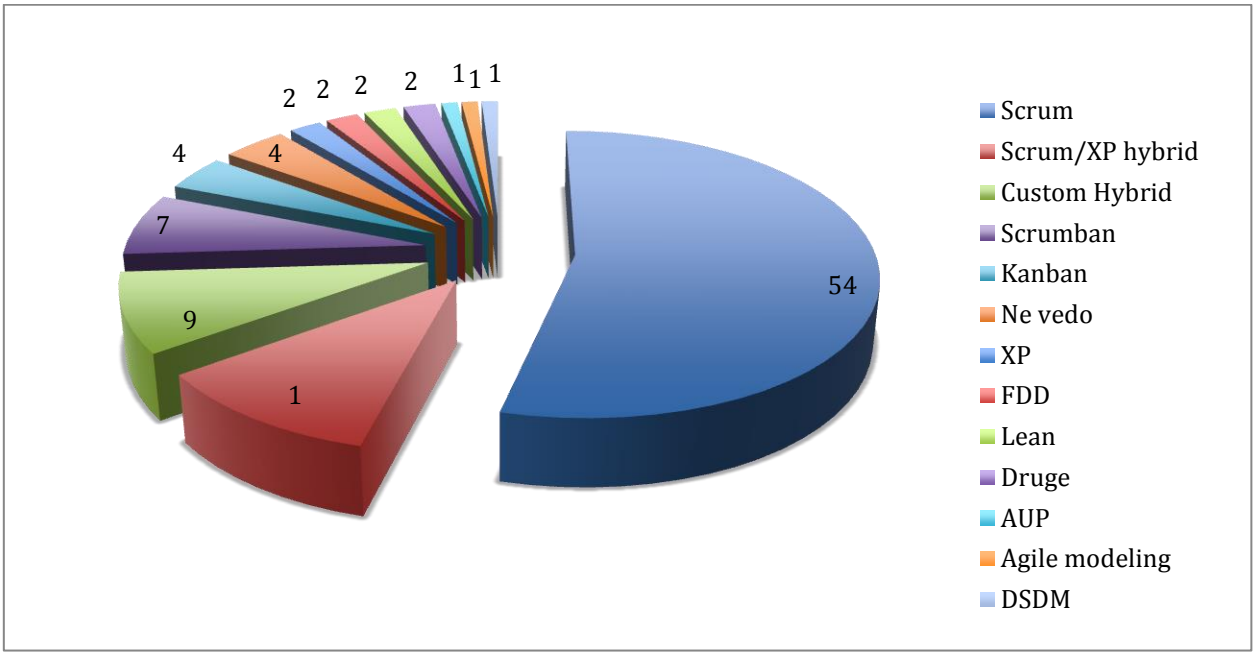
Postopni oziroma inkrementalni pristop omogoča, da se problemi razdelijo na podprobleme ali poduporabniške zgodbe, tako da se jih lahko razvija sočasno ali vzporedno. Ko se problem reši, se rešitev testira in integrira v ostali del programske rešitve. Agilne metode razvoja programskih rešitev uporabljajo ponovitvene in postopne (evolucijske) pristope, ki jih v veliki meri izvajajo samoorganizacijski timi s pomočjo efektivnih upravljavskih okvirjev, ki omogočajo ravno dovolj aktivnosti, da so rešitve visoke kakovosti s stroškovno učinkovitim razvojem, kar ustreza nenehno spreminjajočim se zahtevam naročnika.

Ena večjih razlik, ki sem jih opazil pri agilnih metodah, je manj dokumentacije. Klasične metode dokumentaciji posvečajo preveliko pozornost. Vsebovati mora detajlni načrt poteka projekta, kar je tudi jedro problemov, saj povzroča togost pri vnašanju kakršnihkoli sprememb in inovativnosti v projekt. Manj dokumentacije za izvedbo nalog po mojem mnenju vodi do bolj kodno usmerjenih projektov.

Odločitev za uporabo agilne metode ponavadi sprejmejo lastniki in svetovalci podjetij, ki se ukvarjajo z razvojem programskih rešitev. Teh metod je več. Pravzaprav gre za skupino metod, ki temeljijo na podobnih načelih. Ponavadi mora podjetje dobro premisliti, katera izmed njih je najbolj primerna njegovemu notranjemu okolju, možno pa jih je tudi kombinirati. Kaže, da podjetjem glede na notranje in zunanje okolje zelo ustrezajo mešani pristopi različnih agilnih metod, saj je popularnost hibridov velika.

Poglavitne agilne metode so Scrum, XP, Kanban, Dynamic Systems Development Method (v nadaljevanju DSDM), Adaptive Software Development (v nadaljevanju ASD), Feature-Driven Development (v nadaljevanju FDD), Crystal, med hibridi pa Scrum/XP in Scrumban. Daleč najbolj zaželena in uporabljena metoda je Scrum, ki daje konkretna navodila za organizacijo projektov z večanjem zmožnosti uspešnosti razvoja programske rešitve. Metodo je javnosti predstavil Ken Scwalber leta 1995. Letno poročilo o razvoju agilnih metod iz leta 2014 (Versionone, 2014) navaja, da v letu 2013 agilne metode pozna že več kot polovica vseh podjetij, ki razvijajo programske rešitve.

Slika 1: Agilne metode v uporabi leta 2013 v %



Vir: VersionOne, 7th Annual State of Agile Development Survey, 2013, str. 5.

2.4 Spremembe pri prehodu iz klasičnih na agilne metode

Prehod iz klasične metode na agilno metodo razvoja programske rešitve je zahtevna naloga, zahtevnejša od ustanovitve novega podjetja, ki bo organiziralo razvoj svojih projektov s pomočjo agilnih metod. Ne spremeni se namreč samo potek razvoja projekta, ampak tudi notranja organizacijska struktura. Gre za radikalno spremembo procesa projekta razvoja programske rešitve iz procesno orientiranega v kratek, ponavljajoč, testno usmerjen razvojni proces, usmerjen k ljudem (človeškim virom). Za podjetje je dobro, da se v naprej odloči, katero agilno metodo bodo uporabili, saj imajo nekatere agilne metode zelo specifične zahteve za potek razvoja projekta. Mnoga podjetja začnejo uporabljati pristope, značilne za agilne metode, kot so programiranje v dvojicah, prisotnost naročnika pri projektu, odločitve, sprejete s sodelovanjem med ljudmi, in deljenje znanja, to pa lahko naleti na neodobravanja zaposlenih, navajenih povsem drugačnega procesa dela.

Sprememba procesa projekta razvoja programske rešitve je strateška odločitev, ki je stresna za vse zaposlene v podjetju, zato mora vodstvo podjetja, preden se za to spremembo odloči, dobro preučiti, ali kulturno okolje podjetja sploh dopušča izbiro katere od agilnih metod. Vodstvo mora jasno in dobro določiti zahteve, ki so ključne za končni rezultat projekta. Ponazoriti morajo, koliko se lahko osnovni pogoji razvoja projekta spremenijo, koliko je

projekt kritičen za organizacijo in koliko so zaradi prehoda na drugačno organizacijo prizadete funkcije v ospredju projekta razvoja programske rešitve.

Menim, da je taka sprememba dosegljiva samo z znatnimi vložki časa, dela in finančnih sredstev. Pomembno je, da se podjetja na to dobro pripravijo in da se sprejeti metodi prepustijo zares in ne zgolj formalno. Pomembno je tudi, da se vodstvo podjetja odloči za strategijo, ki se je bo držalo do konca projekta, in da si postavi prioritete uvajanja sprememb. Namreč uvedba vseh sprememb naenkrat zna biti zelo veliko breme tako za projekt kot za zaposlene. S tem se zmanjšajo tveganja zaradi nerazumevanja zaposlenih in nastajanje motenj, ki jih takšna strateška odločitev potegne za seboj. Še posebej veliko pozornosti morajo tem spremembam nameniti podjetja, ki so dolga leta razvijala svoj način organiziranja razvoja projekta in dosegla visoke nivoje zmožnostno-zrelostnega modela.

Prav prehod s klasičnih na agilne metode organiziranja razvoja projekta je zaradi negotovega okolja ena težjih faz procesa. Pri klasičnih metodah se pred sestavo načrta dogovori o predmetih in funkcijah programske rešitve, te pa se določi s pogodbo, kjer se določi tudi strošek izdelave. To je zaradi sprememb, ki se jih pri agilnem razvoju pričakuje, nemogoče, zato mora podjetje prevzeti razvoj, ki je toleranten do sprememb. Zelo je pomembno, da se projektu zagotovi orodja, s katerimi dosega visoko sodelovanje med vodstvom, razvojem in naročnikom. Izvajati je potrebno veliko prototipinga in sprotnih testov, ki dajejo hitre povratne informacije, da lahko nemoteno potekajo hitre ponovitve ter izdaje predmetov in funkcij programske rešitve.

Za podjetje je prav tako pomembno, da preide iz razvojne kulture, ki daje prednost načelom in postopkom, v svobodni razvoj z organizacijo s strani članov tima. Pri agilnih metodah morajo pri odločitvah in sprejemanju odgovornosti sodelovati vsi člani tima. Tim mora pokazati dovolj znanja in pripravljenosti sodelovanja, predvsem pa mora sprejeti novo organizacijo kulture, ki je bolj prilagodljiva namesto usmerjena v predvidevanje.

V tem podpoglavju sem prikazal primerjavo sprememb med klasičnimi in agilnimi metodami, ki so po mojem mnenju pomembne za vodstvo podjetja, ki se ukvarja z razvojem programskih rešitev, ob prevzemu ene izmed agilnih metod. Predstavljene spremembe sem povzel iz več različnih člankov, strokovnih del in knjig, kar mi je dalo objektivnejši pogled na nujne spremembe, ki so zapisane v tabeli.

Tabela 1: Primerjava sprememb med klasičnimi in agilnimi metodami

	Klasične metode	**Agilne metode**
Osnovne predpostavke in cilji	Univerzalni pristop in rešitve za zagotavljanje predvidljivosti in varnosti/zanesljivosti. Razvoj je podrobno opredeljen, razvit je s podrobnim in obsežnim planom. Zahteve so definirane in zapisane vnaprej. Cilja sta predvidljivost in optimizacija. Niso nagnjene k spremembam.	Kakovostna in prilagodljiva programska rešitev je lahko razvita z razmeroma majhnim timom, uporablja princip kontinuiranega dizajna, popravkov in rednih testov, ki dajejo hitre povratne informacije za spremembe. Fleksibilen pristop, pričakuje se konceptna znanja za zagotavljanje hitrejšega razvoja. Percepcija tega je načelo »ne boš potreboval« - YAGNI (angl. *you aren't going to need it*). Cilja sta raziskovanje in prilagodljivost. Nagnjene k spremembam.
Dokumentacija	Veliko dokumentiranja, zahteva eksplicitna znanja.	Lahka, nadomesti se z osebno komunikacijo, tiho znanje.
Vodenje	Procesno orientirano.	Usmerjeno v ljudi.
Stil vodenja	Ukazovanje in nadziranje (angl. *command and control*).	Vodenje in sodelovanje.
Vodenje znanja	Eksplicitno/jasno.	Tiho.
Razvojni tim	Individualna vloga – zaželjena specializacija, plansko orientirani, strukturirani vnaprej.	Samo organizacijski timi – spodbujanje menjave vlog, kolocirani in sodelujoči.
Komunikacija	Formalna.	Neformalna.
Vloga naročnika	Pomembna, majhna in pasivna soudeležba v projektih.	Ključna, aktivna, naročnik velja za del razvojnega tima.
Razvojni stil	Predvidevanje.	Prilagodljiv.
Potek projekta	Usmerjan po nalogah in aktivnostih.	Usmerjan glede na funkcije programske rešitve.
Razvojni modeli	Življenjski cikel (slapovni, spiralni …).	Evolucijsko-dostavni model.
Merjenje uspeha	Skladnost z načrtom.	Podana poslovna vrednost.
Zaželjena organizacijska oblika/struktura	Mehanistična (birokratska z visoko formalizacijo).	Organska (fleksibilna, spodbuja sodelovanje in kooperativne socialne akcije).
Tehnologija	Ni omejitev.	Prednost imajo objektno orientirane tehnologije.

Vir: S. Nerur, R. Mahapatra, & G. Mangalaraj, Challenges of Migrating to Agile Methodologies, 2005, str. 5.; A. B. M. Moniruzzman, & S. A. Hossain, Comparative Study on Agile software development methodologies, 2012, str. 6.

Zelo je pomembno, da vodstva podjetij sprejmejo in razumejo človeški faktor. Avtorji, kot je Nerur (2005), pripisujejo človeškemu faktorju zelo pomembno vlogo pri prevzemu agilnih

metod. Pravzaprav je uspeh prehoda odvisen od odnosa ljudi v podjetju do pristopov, ki jih zahteva izbrana agilna metoda. Ena pomembnejših sprememb pri prehodu iz ene metode organizacije razvoja programskih rešitev je sprememba organizacijske kulture, saj zahteva spremembo navad in miselnosti, ki so jih zaposlenih gradili več let.

Lahko si predstavljamo težavo razvijalca, navajenega na samostojno in samotarsko delo pri klasičnih metodah, ko mora preiti na timsko usmerjeno razvijanje programskih rešitev. Običajno je, da se prevzemu novega načina dela na vse mogoče načine zoperstavlja in upira. Razvijalec, ki je navajen dela po navodilih, si najbrž težko predstavlja kreativno timsko delo, ki predpostavlja predano in dobro sodelovanje s sodelavci in naročniki, predajanje znanja, medsebojno zaupanje in spoštovanje.

Ker so razlike med klasično in agilno obliko razvoja in organiziranja projekta tako radikalne, je zelo pomembno, da pred prehodom podjetje zaposlene natančno seznani s spremembami, ki jih bo zahteval prehod na novo metodo organizacije razvoja programske rešitve. Udeleženci v procesu morajo biti izobraženi in ozaveščeni o novem načinu poteka organizacije projekta. To velja za vse ravni - od vodstva podjetja in vodenje projekta do razvojnikov in naročnikov. Spreminjajo se vse ravni organiziranja projekta. Za nastanek težave je dovolj že, če samo eden od zaposlenih ne razume ali noče razumeti novih relacij in načinov organiziranja razvoja. To se lahko pokaže kot velika ovira pri razvoju programske rešitve. Podjetju pomaga tudi izvedba računalniške simulacije izvedljivosti projekta. Uspeh prehoda je torej odvisen od pripravljenosti vseh ljudi v podjetju.

Pri uporabi klasičnih metod se nadzira in meri doseganje točno določenih ciljev v načrtu organiziranja razvoja programske rešitve. Pri agilnih metodah, kjer organizacija razvoja programskih rešitev temelji na predpostavkah in teoretiziranju, ki ni izkustveno ali praktično utemeljeno, kjer se načrti v toku organizacija razvoja lahko spremenijo in kjer pogon agilnih metod prihaja iz samoorganizacijskih timov, se podjetju pojavlja potreba po uvedbi nove oblike nadzora in merjenja napredka projekta razvoja programske rešitve. Preverjanje napredka projekta razvoja programske rešitve zmanjšuje tveganje za časovno, finančno in vsebinsko dobro izvedbo projekta razvoja programske rešitve. Kvantitativne meritve se pri agilnih metodah oddaljujejo od nadzora, ki gao izvaja vodja projekta pri klasičnih metodah. Ocenjevanje napredka projekta razvoja programske rešitve je del sprotnega preverjanja razvoja programske rešitve (izvajajo ga naročniki), tako da ga primerjajo s seznamom zahtev (angl. *product backlog*). Seznam zahtev (v nadaljevanju *backlog*) je seznam predmetov in

funkcij, ki jih naročnik želi imeti v končnem produktu. Ta seznam je odprtega tipa, na voljo je vsem deležnikom, vanj vpisujejo podatke o opravljenem delu, popravkih napak ali kakršnih koli spremembah. Kljub temu pa je za vsebino odgovoren naročnik, ki vanj vstavlja predmete in funkcije, določa prioritete in preverja potek dela. Vodenje *backloga* izničuje naročnikova tveganja, saj ima popoln pregled nad potekom dela.

Zaželeno je, da se razvojni timi nahajajo v skupnem prostoru, saj se tako najlažje dela v dvojicah in medsebojno sodeluje z ostalimi udeleženci v projektu. To omogoča hitrejše odkrivanje in reševanje napak, hkrati pa pospešuje prehajanje znanja med člani timov, saj lahko težavo, ki je povzročila zastoj, rešijo z medsebojnim sodelovanjem takoj, ko ta nastane. Zelo je pomembno, da vodstva razumejo, da se napake dogajajo. Agilne metode poskušajo razumeti napake in jih izkoristiti za nadaljnji razvoj projekta. Razvijalci za narejene napake ne smejo biti kaznovani, saj se le tako lahko znanje in komunikacija širita med ljudmi, prisotnimi v projektu razvoja programskih rešitev.

Za podjetja, ki se ukvarjajo z razvojem programskih rešitev, so strategije organiziranja in upravljanja znanja nujno potrebne za uspeh. Pri klasičnih metodah je veliko dokumentacije, ki služi kot osnova za dobro komunikacijo in transparentnost oblikovanja. Dokumentacija torej predstavlja upravljanje z znanjem. Znanja, ki jih imajo razvijalci, se zato težko prenesejo na projekt, saj je dokumentiran načrt dela natančno določen in ne dovoljuje veliko inovativnih idej, ki nastajajo med razvojem projekta. Na drugi strani se pri agilnih metodah dokumentacijo postavlja na stranski tir in delo bolj usmerja k delujoči programski rešitvi, kar velja za vsako ponovitev. Znanje je akumulirano pri razvijalcih, nekaj informacij pa je skritih tudi v kodi projektov. Na ta način postajajo podjetja odvisna od razvojnih timov, del vodstvene moči pa se razdeli z vodstva projekta na druge člane tima. Mnogim podjetjem to ni všeč. Kako določati, katero znanje je dobro zapisati v kodo in katero naj ostane zakrito, je eden od novih fokusov organiziranja pri agilnih metodah razvoja projekta.

Klasični modeli uporabljajo razvoj projekta s pomočjo življenjskega cikla, proces je voden skladno z aktivnostmi in je mersko usmerjen, da zagotavlja točnost in popolnost. V nasprotju z življenjskim ciklom je pri agilnih metodah proces razvoja programske rešitve naravnan funkcijsko-evolucijsko, predvsem pa ponavljajoče.

Pomembnejše spremembe so še sprememba osebnostnih značilnosti, spremembe v odnosu z naročnikom in spremembe v znanju in izobraževanju. Agilne metode so naklonjene ljudem, ki

so samomotivirani in samostojni, saj morajo odločitve v ključnih trenutkih sprejemati sami. Menim, da je zelo dobro, če imajo udeleženci v procesu razvoja tudi določena znanja iz prodaje. Ta znanja namreč lahko uporabijo pri odnosu z naročniki, ti pa morajo razvojnim timom zaupati in jim biti na razpolago, kadar jih ti potrebujejo. Delujoča programska rešitev konec ponovitve je tista, ki povečuje medsebojno zaupanje.

3 ZNAČILNOST ORGANIZIRANJA PROJEKTOV RAZVOJA PROGRAMSKIH REŠITEV S POMOČJO AGILNIH METOD

S prevzemom agilnih metod je organiziranje projekta dobilo povsem novo obliko. Spremenjenim vlogam in nalogam sodelujočih v razvoju programske rešitve se morajo prilagoditi vse strukture zaposlenih v podjetju in tudi naročniki. Razvoj programskih rešitev s pomočjo agilnih metod zahteva nova znanja, ki jih morata vodstvo in vodja projekta upoštevati v praksi, če želita zagotoviti uspeh projekta in večanje poslovne uspešnosti. Vsako podjetje ima svojo kulturo in način vodenja, zato so rezultati med seboj lahko povsem drugačni. Naloga podjetja je razporediti znanje, ki je potrebno za uspeh.

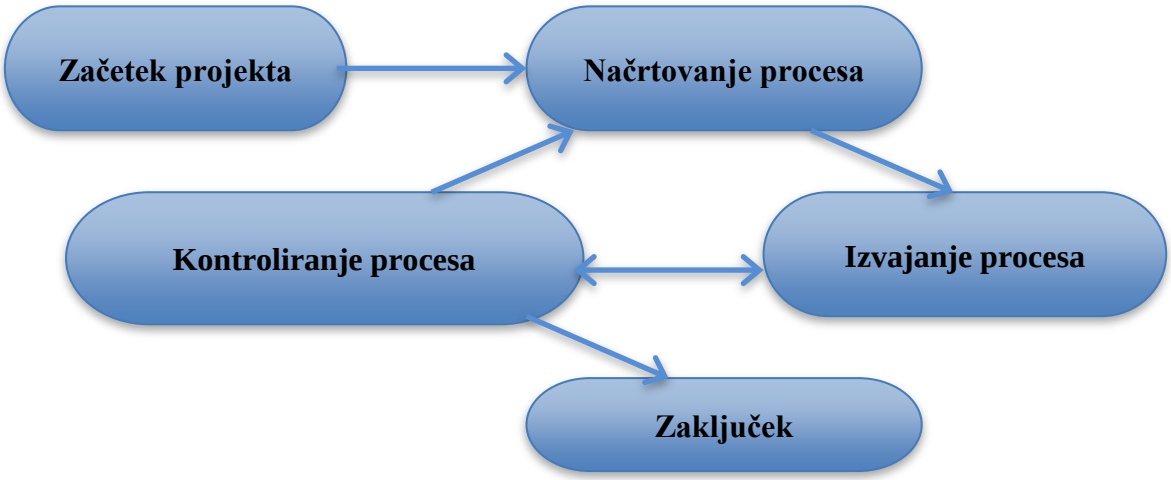
3.1 Organizacija projekta

Naročnik in podjetje, ki razvija programske rešitve, skleneta pogodbo z natančno opredeljenimi pogodbenimi tveganji. Ko je to narejeno, podjetje določi vodjo projekta in time, ki bodo delali na razvoju naročene kode oz. programske rešitve.

Pri večjih projektih se pogosto razvija celotne informacijske sisteme, kjer je potrebno uporabiti zelo različne sklope funkcij, ki zahtevajo drugačna znanja za razvoj. Pri takih projektih se svetuje organiziranje več podprojektov z izbiro podtimov, ki razvijajo posamezen sklop končne programske rešitve. V tem primeru je bolje določiti tudi več vodij teh timov in enega, ki skrbi za to, da se vsi podprojekti lažje združijo v celoto.

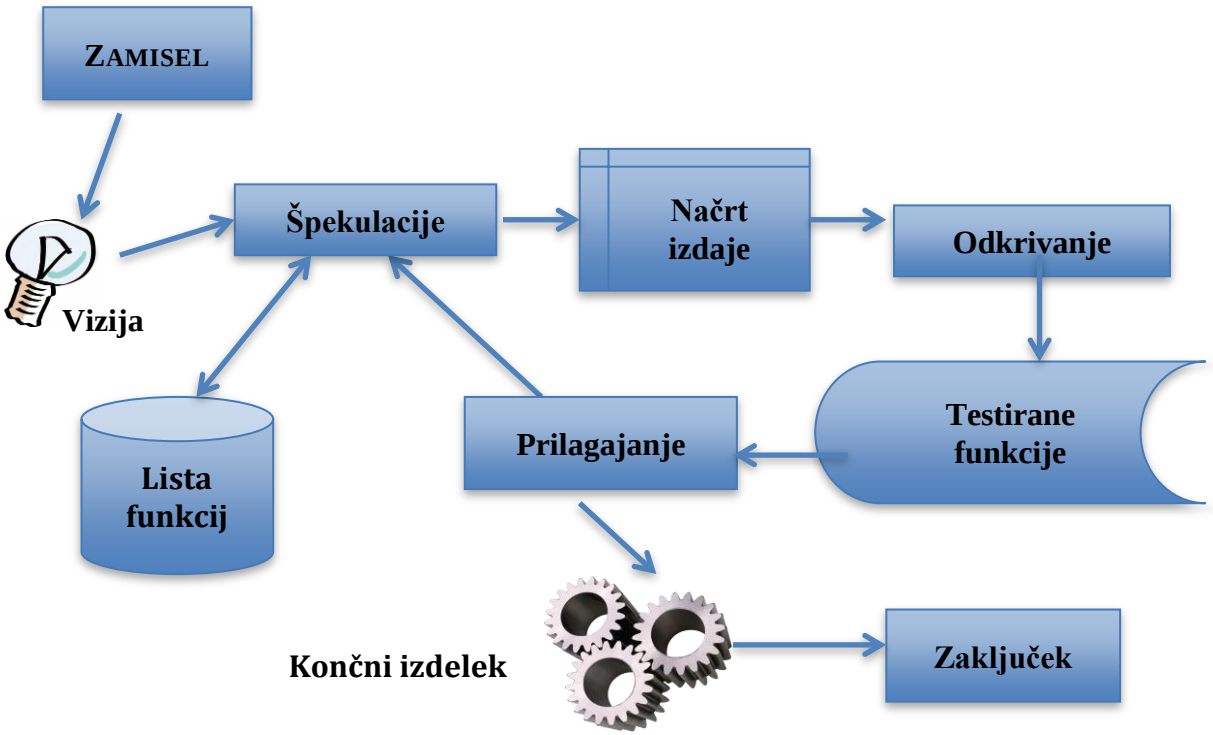
V *Project Management Book of Knowledge* (v nadaljevanju PMBOK), zbirki dobri praks, je opisan tudi model projektnega organiziranja, značilnega za klasično organizacijo razvoja programske rešitve. Highsmith (2003) je faze klasičnega modela razvoja projekta priredil tako, da bolje ustrezajo organizaciji razvoja programskih rešitev.

Slika 2: Model klasičnega projektnega organiziranja



Vir: A Guide to the Project Management Body of Knowledge, 2013, str. 39.

Slika 3: Agilno vodenje projektov po Highsmithu (2003)



Vir: J. Highsmith, Principles and Tools, 2003, str. 8.

Medtem ko je planiranje procesa prva faza modela projektnega organiziranja, značilnega za klasično organizacijo razvoja programske rešitve, Highsmith (2003) meni, da je prva faza pri agilnem vodenju projekta vizija. Vizija je zanj ključen del uspeha projekta. Brez nje organizacija razvoja programske rešitve praktično ni mogoča. V fazi, ki se imenuje vizija, se določi, kdo bo pri projektu sodeloval in kako bodo timi delovali, kaj bo potrebno razviti in v kakšnem obsegu. Podjetje mora v tej fazi pravilno razporediti znanje in določiti, kdo bo pri projektu sodeloval. Projekti imajo lahko več različnih funkcij, ki jih podjetje razvija sočasno,

zato se pri večjih projektih izbere več manjših timov, ki delajo na istem projektu. Razvojni timi v tej fazi vizualizirajo svoje delo pri projektu. Informacije, ki so v tej fazi dosegljive, lahko med projekti variirajo, vedno pa so jasni ključni predmeti in funkcije, za katere se vzajemno dogovorijo deležniki v projektu razvoja programske rešitve.

Pri manjših projektih je ta faza lahko končana že v kakšnem tednu. Pri večji projektih pa se dodatna izobraževanja, iskanje novih virov sredstev, arhitektura in obrazložitve analiz lahko zavlečejo v mesece. Od te faze naročnik nima nobene funkcionalne koristi, saj je namenjena usklajevanju znotraj podjetja.

Druga faza po Highsmithu je faza špekuliranja. Highsmith (2003) začne uporabljati besedo špekuliranje, da se izogne besedi načrt, saj meni, da je to beseda, ki jo preveč povezujemo s predvidevanjem in relativno gotovostjo, špekulacija pa po njegovem bolje ponazarja, da je projekt razvoja programskih rešitev spremenljiv.

Vizijo projekta se v tej fazi pretvori v *backloge*, ki naj bi jih programska rešitev vsebovala, ter predvidevane *timeboxe*, v katerih bodo funkcije izdelane. Rezultat te faze je načrt izdaje. Zasnova načrta je znanje, ki je trenutno na voljo deležnikom. To je ponovitveni načrt, ki temelji na funkcijah namesto na nalogah, ki se jih lahko izvede. Za izdelavo načrta izdaje je potrebno združiti informacije o lastnostih programske rešitve, arhitekturi platforme, virih, analizi tveganj, omejitvah podjetja, stroških in urnikih.

Namesto kontrole, ki je v tej fazi pri klasičnih metodah zaželena, Highsmith (2003) predvidi fazo odkrivanja. Gre za fazo, v kateri se ponavljajoče in postopno razvije potencialno delujočo kodo z načrtovanimi prevedbami in prilagoditvami med razvojem; koda vsebuje tudi proces in vsebine programske rešitve v razvoju. Vodja projekta se najprej osredotoča na izdajo delujoče kode in šele nato na skladnost programske rešitve. Bistvo te faze je zagotoviti delujoče testirane funkcije programske rešitve v kratkih časovnih okvirjih.

V fazi zaključka Highsmith (2003) predlaga, da se zberejo vsi notranji deležniki v projektu razvoja programske rešitve. Razvojni timi imajo tako možnost izmenjave znanja, pridobljenega med razvojem programske rešitve, med vsemi prisotnimi, vodstvo pa priložnost, da poudari dobre rešitve in uspehe. Taki motivacijski zaključki projektov dajejo deležnikom nov zagon in vzpodbudo.

Za organizacijo razvoja programskih rešitev s pomočjo agilnih metod je za Highsmitha (2003) pomembnih pet področij znanj, ki se jih lahko preslika v agilne metode. To so organiziranje integracije, organiziranje obsega in časa, vodenje kakovosti in organiziranje človeških virov. Ker so človeški viri za agilne metode področje, kjer so razlike najbolj vidne, v nadaljevanju prikazujem vloge, ki se formirajo pri razvoju programskih rešitev s pomočjo agilnih metod.

3.2 Organiziranje integracije projekta

Integracija je povezovanje in združevanje delov ali funkcij projekta v skupno delujočo celoto. Pri klasičnih metodah organiziranja projekta je integracija planirana vnaprej kot del načrta za organiziranje in delegiranje celotnega projekta razvoja programske rešitve.

Pri agilnih metodah je organiziranje integracije projekta kontinuirano (Fowler, 2006). Najprej se ponavljajoče in postopoma zbira zamisli in načrtuje. Vodja projekta se ne usmerja na vse elemente projekta že na začetku, ampak se raje posveča daljnosežnemu načrtovanju in z nizko stopnjo formalne dokumentacije dosega primerno komunikacijo med deležniki v projektu razvoja programske rešitve (Sliger, 2006). Predmeti in funkcije so v obliki uporabniških zgodb lahko napisane tudi na tablah v skupnem prostoru, kjer se nahajajo razvojni timi in potekajo skupni sestanki. Nekateri uporabljajo tudi raznobarvne samolepilne listke, ki jih lepijo na stene skupne sobe. V zadnjih letih pa se vedno bolj uporabljajo tudi primerne internetne programske rešitve, namenjene medsebojni komunikaciji. Te so primernejše za projekte, ki so geografsko razpršeni.

Razvojni timi si sami zastavijo načrt ponovitve in druge načrte, ki nastanejo v fazi zamisli in špekuliranj, in sami sebe zavezujejo k temu, da se jih držijo. Skupaj z deležniki sodelujejo pri pripravi izvedbenega načrta. V tem načrtu opredelijo čas dobave delujočega dela programa, funkcije programske rešitve in načrt ponovitve.

Zagotavljanje nadzora nad spremembami se pri integracijskem organiziranju močno razlikuje med agilnimi in klasičnimi metodami organiziranja projektov razvoja programskih rešitev. Agilne metode spodbujajo vsakodnevne približno 15-minutne sestanke, s katerimi dosežejo, da se spremembe in napake rešujejo sproti in da se zagotavlja minimum procesa za doseganje višje kakovosti programske rešitve (Sliger, 2006). Fowler (2006) ugotavlja, da se popravki izvajajo celo nekajkrat dnevno, saj se zaradi sprotnih testov napake vidijo zelo hitro. Nadzor se izvaja sprotno z *backlogom*, seznamom vseh uporabnih funkcij, ki jih je potrebno še razviti

do izdaje končne programske rešitve. Za *backlog* skrbi predvsem naročnik. Naročnik mora sam določati listo funkcij, ki morajo biti razvite, med njimi pa dajati prednost tistim predmetom in funkcijam, ki so najbolj potrebne. *Backlog* vsebuje tudi predmete, ki niso v povezavi samo s funkcijami, temveč vsebuje tudi delo tehnične podpore in napake, prav tako rangirane po pomembnosti. Višje rangirani predmeti in funkcije se v fazi sestave načrta ponovitve in izdaje prestavijo iz seznama *backlog* v ponovitev, kjer se jih kodira in testira, da jih lahko naročnik preveri in sprejme ali zavrne. Dnevni sestanki dajejo idealno možnost, da razvojni timi in naročnik predebatirajo in rešijo nepredvidene težave, hkrati pa predvidevajo možne težave, ki še lahko nastanejo. Po mnenju Fowlerja (2006) zaradi takšne oblike nadzora in sprotnih testiranj podjetja izdelujejo stabilne dele končnih programskih rešitev z zelo malo hrošči.

Na koncu vsake ponovitve razvojni timi predstavijo delujočo in testirano kodo naročniku, njegove povratne informacije pa so tiste, zaradi katerih se spremenijo odločitve v zvezi s sezamom *backlog* in rangiranjem predmetov v njem. Na koncu ponovitve se lahko spremeni tudi proces, tako da razvojni timi prilagodijo programsko rešitev ali način dela (Sliger, 2006). Celoten tim, ki dela na projektu, mora postati enakovreden pri kontroliranju sprememb, da se lahko odločitve sprejemajo hitro, s sodelovanjem in z malo ali nič formalizacije.

3.3 Organiziranje obsega in časa

Pri klasičnih metodah se spreminjanje obsega projekta ne odobrava. Projekti so organizirani tako, da sledijo začrtanemu cilju od začetka do konca brez oziroma z malo spremembami. Agilne metode spremembe pričakujejo in sprejemajo, saj se ponavadi fiksira stroške in urnike razvoja programske rešitve ter nato izdela delujočo rešitev, ki daje najvišjo funkcionalno vrednost za naročnika (Larman, 2003). Mislim, da je organizacija obsega in časa razvoja programske rešitve ena težjih sprememb znanj predvsem za vodjo projekta, še posebej, če njegovo znanje izhaja iz okolja klasičnih metod. Pri klasičnih metodah razvoja programske rešitve vodje projektov v želji, da bi bil projekt čim prej končan, pogosto preobremenijo razvojnike. Zaradi nepredvidenih težav prihaja do napak in nerešenih vprašanj.

Pri agilnih metodah je obseg dela določen le za predmete in funkcije, ki se v pravilnih odmerkih razporedijo v krajše časovne intervale ali *timeboxe*. Zaradi sprememb obsega razvoja kot posledice prilaganja vnajprej določenemu *timeboxu* je preglednost nad razvojem projekta zelo težka. Določitev konca projekta razvoja programske rešitve je praktično

nemogoča. Tudi pri klasičnih metodah je v razvojni fazi težko napovedati konec projekta, ker napovedi temeljijo na podlagi predvidevanj ali lastnih izkušenj vodje projekta (Sliger, 2006). Pri agilnih metodah se torej organiziranje obsega in časa odvija že pri organizaciji ponovitve. Zato se načrt ponovitve izdela samo za predmete in funkcije, ki se jih določi v *timeboxu*, in ne za celoten sistem.

3.4 Organiziranje vodenja kakovosti

Menim, da je organiziranje vodenja kakovosti in testiranj pri klasičnih metodah enako pomembno kot pri agilnih metodah. Težava nastaja, ker se ponavadi kontrolo kakovosti s testi izvaja v zadnjih dveh tednih pred izdajo programske rešitve naročniku. Testiranje preobsežnega dela v prekratkih rokih je že samo po sebi stresno delo. Odkrivajo se funkcije, ki niso v skladu z naročnikovo željo ali pa niso integrirane v sistem, a jih zaradi kratkega roka za testiranje ni mogoče popraviti in prilagoditi.

Agilne metode spodbujajo sprotno testiranje za zagotavljanje kakovosti razvite kode. Z njim se začne že po fazi planiranja ponovitve, ko so znane zahteve. To zmanjšuje možnost, da se pojavijo težave proti koncu ponovitve. Vsaka ponovitev daje naročniku novo funkcionalno vrednost in traja povprečno dva do tri tedne (Larman, 2003).

Kontrola kakovosti mora biti taka, da ustreza osnovam zagotavljanja kakovosti in je v skladu s politiko podjetja, njena implementacija in izvajanje pa sta stvar vodstva podjetja. Pri agilnih metodah se koda razvija ponavljajoče za vsako ponovitev, zato se zagotavljanje kakovosti začne že na začetku projekta s testi še nerazvite kode neke funkcije, kar omogoča večjo prisotnost ljudi, zadolženih za kakovost in teste skozi celoten projekt.

Zagotavljanje kakovosti je po mojem mnenju eden glavnih razlogov, da lahko agilne metode zagotavljajo višjo kakovost kode v krajšem časovnem intervalu. Celotna organizacija procesa razvoja programske rešitve se prilagodi tako, da zagotavlja testno voden razvoj (angl. *test-driven development*). Ta razvoj je značilen za metodo XP, nato pa se je od tu razširil na vse pomembnejše agilne metode (Sliger, 2006). Pomembno je avtomatizirano testiranje skoraj vsega, teste se piše, preden se razvije funkcionalna koda - podobno kot pri prototipnem pristopu, kjer se koda dejansko piše dvakrat. Gre za preprosto nalogo, kjer se testni primer in test enote enostavno označi s »sprejeto« ali »ni sprejeto«. Ko so testi končani, se lahko začne razvijati funkcionalno kodo, ki se konča z uspešno prestanim testiranjem primera; ta se mora

zaključiti še pred koncem ponovitve. Tako lahko zagotovimo najboljšo organizacijsko prakso načrtovanja, zagotavljanja in kontroliranja kakovosti funkcij programske rešitve (Sliger, 2006).

Značilnost agilnih metod je, da pri zagotavljanju kakovosti sodelujejo vsi deležniki v projektu razvoja programske rešitve. Testni timi se morajo povezovati z razvojnimi timi skozi celoten življenjski cikel projekta. Vedeti morajo, katere prijeme in tehnologije bodo uporabili za zagotavljanje najboljših rezultatov, ki olajšajo in razbremenijo delo razvojnih timov. Pri testih mora sodelovati tudi naročnik s testi sprejetja (angl. *acceptance test*) (Highmith, 2003).

Zagotavljanje kakovosti preprečuje nastanek hib in daje razvojnikom možnost izdelave kode višje kakovosti. Sestaviti je potrebno več scenarijev "kaj, če", da se izvajalci testiranja lažje usmerijo v tisto, kar razvojniki potrebujejo, namesto, da to poskušajo ugotoviti sami. Rezultat teh scenarijev olajša izvajalcem testiranja vpogled v funkcionalnost programske rešitve, hkrati pa lahko razvojniki in naročniki zanjo pišejo boljše testne primere. Pri kontroli kakovosti morajo testni timi odkrivati in prepoznavati hibe, ki so se že izmuznile v programsko rešitev s sodelovanjem z razvojnimi timi, ki te hibe izničijo. To imenujemo »iskanje hroščev« (angl. *bug-checking*) (Sliger, 2006). V sklopu ponovitev mora podjetje z različnimi tehnikami in sodelovanjem vseh deležnikov v projektu razvoja programskih rešitev zagotoviti, da razvita koda ustreza pričakovanjem naročnika.

3.5 Sprejemanje odločitev v podjetju, organiziranem s pomočjo agilnih metod

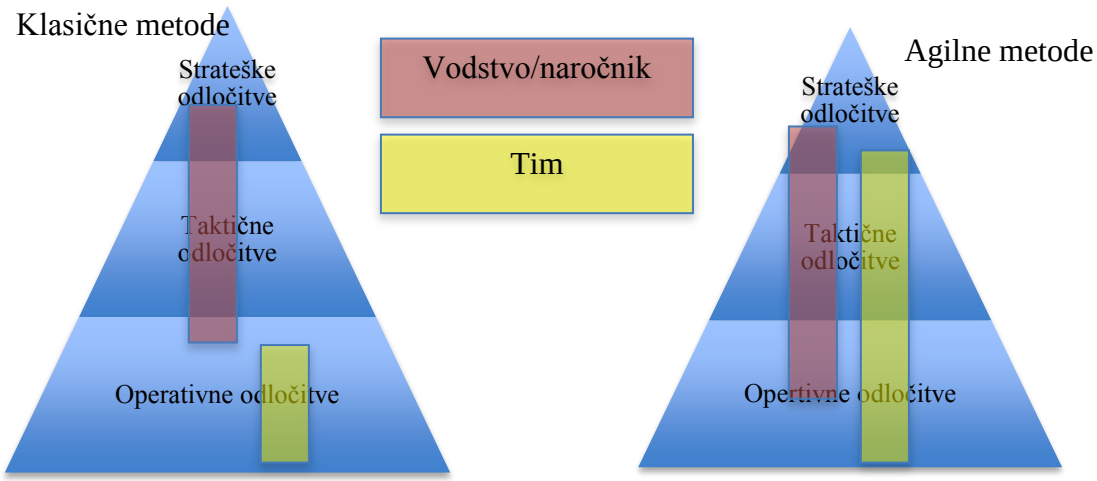
Agilne metode ne spreminjajo osnov, ki so potrebne za razvoj novih programskih rešitev, spreminjajo pa sodelovanje, koordinacijo in komunikacijo med sodelujočimi v procesu razvoja novih programskih rešitev (Moe, Aybüke, & Dyba, 2011). Ker je delujoča programska rešitev izdelana v zelo kratkem času, se morajo tudi odločitve sprejemati hitreje v primerjavi s klasičnimi, plansko organiziranimi metodami. Pomembne odločitve o tem, kaj in kako delati, se porazdeljujejo med več ljudi, ki se medsebojno dopolnjujejo. Vodenje ni več domena ene osebe ali vodstva. Sedaj se v marsikaterem podjetju v poslovne odločitve, ki so jih sprejemali srednje rangirani vodje, vključujejo tudi nižje rangirani zaposleni, udeleženi v procesu razvoja programske rešitve.

V vodstvene odločitve se morajo vključevati tudi razvijalci oziroma timi, ki se pri agilnih metodah organizirajo sami. Člani tima so odgovorni za taktične in operativne odločitve v različnih fazah projekta, o katerih imajo razpoložljiva tehnična znanja, ki lahko pripomorejo k uspehu projekta. Za sprejemanje operativnih odločitev se morajo timi pogosto prilagajati nastali situaciji, saj se sprejemajo sprotno ali inkrementalno. Pogosto se v odločanje vključuje tudi naročnik. Sodelovanje z naročnikom vodi k sprejemanju odločitev, ki so bolj usmerjene k optimalnemu uspehu projekta. To sicer v sprejemanje odločitev prinaša več težav kot pri klasičnih metodah, kjer večino teh odločitev sprejme projektni vodja.

V splošnem poznamo tri nivoje odločitev v podjetjih, ki so prilagojene vodstvenim aktivnostim, katerih meje pogosto niso dobro začrtane. Antonyjev model pri sprejemanju odločitev predvideva hierarhijo odločanja od zgoraj navzdol, agilne metode pa jo prestavijo od spodaj navzgor. Odločitvene procese Antony (1965) deli na:

- **strateške** odločitve, ki so skladne s strategijo in cilji podjetja;
- **taktične** odločitve, ki so usmerjene v istovetnost in porabo sredstev;
- **operativne** odločitve, ki se ukvarjajo z učinkovitimi dnevnimi odločitvami.

Slika 4: Antonyjev model procesa sprejemanja odločitev



Vir : N. B. Moe, A. Aybüke, & T. Dyba, Challenges of shared decision-making: A multiple case of agile software development, 2011, str 861.

Dolgo je Antonyjev model, ki utemeljuje hierarhijo odločanja, veljal za dobro obrazložitev sprejemanja odločitev v podjetju. S pojavom agilnih metod pa se začne model sprejemanja odločitev v podjetju spreminjati. Medtem ko pri klasičnih metodah odločitve sprejemajo predvsem vodstvo in vodje projekta, pri agilnih metodah dobršen del odločitev preide na nižje

rangirane delavce. Večino operativnih odločitev prevzamejo samoorganizacijski timi in naročnik, kar je ključnega pomena za organizacijo razvoja programskih rešitev s pomočjo agilnih metod (Moe et al., 2011).

Po mnenju Moe et al. (2011) podjetja, ki razvijajo programske rešitve, za razvoj programskih rešitev opravljajo rutinska in nerutinska dela, zato se jim zdi smiselna uporaba tako racionalnih kot omejenih racionalnih (angl. *boundet rational*) teorij o sprejemanju odločitev. Zato Moe et al. (2011) razdelijo odločitve pri razvoju programskih rešitev na:

- **Racionalne odločitve**, ki predstavljajo najbolj splošen pogled na sprejemanje odločitev. Sprejema se jih, kadar so na voljo vse informacije v zvezi s problemom. So bolj rutinske, ponavadi so odgovori na problem vnaprej znani Pri njih se odloča za tistega, ki je usmerjen k najboljšemu rezultatu. Ta proces se lahko rešuje tudi z racionalnimi modeli.
- **Nerutinske odločitve**, ki so lahko zelo kompleksne, saj se cilji pogosto spreminjajo, odločitve pa so omejene z informacijami, ki so na voljo. Zelo težko jih je načrtovati in standardizirati.
- **Naturalistične odločitve**, ki se jih najpogostje uporablja pri agilnih metodah. Gre za odločitve, ki jih sprejemamo s pomočjo znanj v realnem okolju. Za naturalistično odločitev so potrebni razumevanje problema, zaznavanje situacije in pravilen odziv, ne pa samo izbira med rešitvami, ki so na voljo.

Pri agilnih metodah gre za deljeno sprejemanje odločitev (angl. *shared decision making*) in samoorganiziranje (angl. *self-managment*). Pri deljenem sprejemanju odločitev ni nujno, da so udeleženi vsi člani samoorganizacijskega tima, je pa zaradi odločitev, ki jih morajo sprejeti vsi deležniki, pomembno, da so vsi obveščeni o vseh odločitvah – tudi o tistih, ki jih ne sprejemajo vsi (Moe et al., 2011).

Ugotavljam, da je dobra komunikacija pogoj, da so lahko vsi akterji sprotno in zadostno informirani o nadaljnjem razvoju projekta. Ideje in zamisli morajo biti vedno jasno in prepričljivo izražene, da lahko akterji v projektu razvoja programske rešitve izpolnijo svoje zadolžitve. Samoorganizacijski timi imajo lahko težavo pri sinhronizaciji odločitev, če jim primanjkuje informacij. Je pa sprejemanje odločitev pri vsakem projektu drugačno, kar velja tudi, če so si končni rezultati projektov razvoja programskih rešitev med seboj podobni.

3.6 Organiziranje človeških virov

Pri agilnih metodah se vodenje in organiziranje človeških virov odraža v velikem številu različnih možnosti. Dejstvo pa je, da so človeški viri eden najpomembnejših delov organiziranja projekta razvoja programskih rešitev s pomočjo agilnih metod.

Pri agilnih metodah mora vodja projekta pravzaprav narediti preskok z vodje, ki poveljuje in kontrolira, v vodjo, ki sodeluje s člani celotnega tima in usmerja njihovo delo. Zagotoviti mora sodelovanje med vsemi člani tima, ki delajo na skupnem projektu razvoja programske rešitve, kar je za agilne metode ključnega pomena. Timi morajo biti medfunkcijski in samoupravni, s primernimi znanji in kompetencami vsakega posameznega člana tima, kar jim omogoča sodelovanje in doseganje kompromisov z naročnikom. Naročnik ali njegov predstavnik sta namreč pri agilnih metodah del celotnega tima. Naročnik in razvojni timi morajo sodelovati pri definiranju končnega rezultata in testih.

3.6.1 Vodja projekta

Odločitev za prehod iz organizacije projektov s pomočjo klasičnih v agilne metode zahteva tudi spremembo sloga vodenja projekta. Pri klasičnih metodah vodja projekta sestavi dobro definiran in detajlno dokumentiran načrt, ki ga nato uveljavlja in nadzira njegovo izvajanje. Razdeljuje naloge, daje napotke za delo in nadzoruje delo zaposlenih na projektu. Pri agilnih metodah pa razvojni timi sami sprejemajo večino odločitev, ki jih pri projektih, organiziranih s pomočjo klasičnih metod, sprejema vodja. Zato se lahko vprašamo, ali so vodje projektov v podjetju, ki uporablja agilne metode, sploh še potrebni in če so, kakšna je njihova vloga.

Na podlagi različnih strokovnih publikacij sem primerjal vlogo vodstva podjetja in vlogo vodje projekta razvoja programskih rešitev, kot ga poznajo klasične metode za organizacijo razvoja projekta, z vlogo, ki jo imajo vodstva podjetja in vodje projektov pri agilnih metodah. Čeprav so odločitve, ki jih sprejemajo razvojni timi, že prerasle iz operativnih v taktične in celo strateške, ugotavljam, da je vloga vodje projekta razvoja programskih rešitev za uspeh projektov tudi pri agilnih metodah še vedno zelo pomembna. Ugotavljam pa tudi, da obstaja velika povezava med uspehom projekta in dobrim vodenjem tima.

V nadaljevanju povzemam lastnosti, ki se najpogosteje navajajo pri opisu agilnega vodje projekta razvoja programske rešitve.

Vodja ima pomembne odgovornosti. Narediti mora vizijo organiziranja razvoja programske rešitve, nato pa voditi tim k njeni realizaciji. Resda vodja projekta v primeru uporabe agilnih metod ni več avtoritativen voditelj, organizator in nadzornik izvedbe projekta, je pa potreben kot podpornik članov tima, vplivnež, ki potiska zadeve naprej; je usklajevalec in povezovalec, ki sodeluje s člani tima, naročnikom, vodstvom in ostalimi možnimi deležniki projekta. Pri agilnih metodah mora vodja dopusti, da se člani tima odločajo samostojno. Delovati mora kot povezovalec, torej nekdo, ki usmerja in koordinira dosežke projekta razvoja programske rešitve s sodelovanjem. Vodja mora prepoznavati ustvarjalne zamisli vseh sodelujočih v projektu in jih vključiti v končno odločitev projekta. Highsmith (2003) je mnenja, da je v primeru slabe komunikacije med naročnikom naloga vodje, da to prepreči in zagotovi boljše pogoje za sodelovanje pri projektu. Rezultat takega sodelovanja je, da se združijo znanja in izkušnje iz več različnih ozadij in filozofij.

Menim, da se mora vodja pri agilnih metodah bolj posvetiti vprašanju, kako organizirati razvojne time, da bodo pripravljeni na to, da se predmeti in funkcije programske rešitve spreminjajo, oziroma da se kreativno odzivajo na vse mogoče motnje, ki se pojavijo v razvoju programskih rešitev.

Vodja projekta tudi še vedno načrtuje, s čimer zmanjšuje negotovosti, vendar se mora ob tem naučiti špekuliranja in prilagajanja namesto načrtovanja in grajenja, kot se vodi projekt pri klasičnih metodah. Načrtovati mora ravno toliko, da lahko projekt nemoteno napreduje, načrt pa mora znati organizirati in prilagajati tako, da je rezultat sprejemljiv za naročnika (Larman, 2003).

Biti mora pripravljen tvegati in preizkušati nove nepreverjene pristope, ki so pogosto zamisli posameznikov v projektnem timu. Uporabiti mora ponavljajoč, postopen in sočasen pristop za razvoj s fokusom na višanju poslovne vrednosti podjetja namesto določanja natančno definiranega načrta (Highsmith, 2003). Na težave in odločitve mora gledati bolj dolgoročno in široko s pomočjo analiz in načrtovanja ter dobro premisliti o tem, kako bodo današnje odločitve vplivale na razvojne time, podjetje in druge deležnike pri razvoju programskih rešitev.

Timi svoje dinamike ne morejo spremeniti ali ustvariti čez noč, za to so potrebni dolgi procesi prilagajanja in učenja. Pogosto je močan in karizmatičen vodja tisti, ki tim pripravi na

sodelovanje med člani in jih nauči sprejemanja skupinskih odločitev. Agilni vodje prenašajo znanje in veščine na člane tima, jih navdušujejo in motivirajo ter vodijo pri premoščanju morebitnih zastojev projekta in uspešnem prehodu ob spremembah predmeta in funkcije projekta. Njihov uspeh je zelo odvisen od motiviranosti, iznajdljivosti, ustvarjalnosti in kompatibilnosti vseh deležnikov v projektu, vključno z naročnikom (Scrum alliance – The Manager's Role in Agile, 2008). Pomembno je, da poznajo organizacijsko kulturo in okolje podjetja ter da so sposobni sprejemati načela, ki temeljijo na agilnih metodah. Šele ko vodja razume, kako je podjetje sposobno sprejemati agilne vrednote, lahko začne organizirati projekt s sestavo tima, ki bo najprimernejši za projekt razvoja programske rešitve (Kittlaus, & Herde, 2011). Iz opisov vodij in njihovega dela v različnih strokovnih publikacijah ugotavljam, da je dober vodja tisti, ki lahko pomembno vpliva na pozitiven izid projekta in delovanje članov tima.

3.6.2 Samoorganizacijski timi

Agilnim metodam predstavljajo timi osnovo za uspešno izveden projekt. Tim razumem kot skupino ljudi z določenimi znanji, ki mora čutiti odgovornost za uresničevanje in doseganje skupnega cilja. To dosegajo z visoko mero kooperativnosti. Ugotavljam, da je tim pri agilnih metodah zelo avtonomen, kar od članov poleg timskega duha zahteva visoko mero samoodgovornosti in sposobnost samoupravljanja. Tim, ki dela na projektu, mora biti sposoben spremeniti uporabnikovo zgodbo (angl. *user story*) v zaključek projekta. Podjetje pa mora poskrbeti, da je v timu dovolj akumuliranega znanja, da izvede razvoj delujoče kode za vsako ponovitev z vsemi funkcijami, načrtovanimi v *timeboxu* (Larman, 2003).

Naloga vodstva je, da timom predstavi najbolj potrebne zahteve, ki so pomembne za uspeh projekta. Ko imajo timi prioriteten seznam zahtev, postane tim del procesa. Tim je zato potrebno sestaviti medfunkcijsko in stabilno, njegovi člani pa morajo tudi zavestno sodelovati pri sprejemanju odločitev. Prevzemajo odgovornosti za različne zadolžitve v projektu razvoja programske rešitve, kot je načrtovanje, sestavljanje terminskih načrtov in sprejemanje odločitve, ki so pomembne tudi za ekonomski uspeh projekta.

Agilni razvojni timi so sestavljeni iz individualnih članov, ki si postavljajo svoje lastne delovne obremenitve, za katere so prepričani, da jih lahko opravijo v naslednji ponovitvi. Taki obliki organizacije pravimo samoorganizacijski timi. S tako organizacijo dosežemo, da se delo bolje razporeja med člane tima, tako da vsak prevzame delo, ki ga najbolje opravlja.

Samoorganizacijski timi morajo s skupnimi cilji dosegati skupno korist. To je mogoče, če imajo med seboj veliko mero zaupanja in sposobnost, da se hitro prilagajajo novim izzivom in znova organizirajo. Timi so samoorganizacijski takrat, ko znajo redno preverjati produkt in proces, tako da skupinsko preverjajo opravljeno delo in sprejemajo odločitve o nadaljevanju projekta razvoja programske rešitve. Če je tim pri tem uspešen, je velika verjetnost, da bodo timi lažje presegali tako fizične kot psihične prepreke med vsemi deležniki v projektu razvoja programske rešitve.

Število članov ne sme biti premajhno. Premalo članov omejuje zgoraj omenjene prednosti, ki jih prinese sodelovanje med člani tima. Najbolj razširjene agilne metode, kot sta SCRUM in XP, spodbujajo time z okoli sedmimi člani. V praksi je število članov tima odvisno od velikosti projekta razvoja programske rešitve. Majhni timi štejejo do 15 ljudi. Taki timi potrebujejo vodjo, ki formira razvojni tim in pomaga pri boljši komunikaciji med člani in sledenju vizije projekta. V večjih timih, ki so lahko sestavljeni tudi iz 50 ljudi, pa se ponavadi vse deležnike razdeli v manjše razvojne podtime. Vsak manjši podtim prevzame svoj poddel programske rešitve, za katero je zadolžen in odgovoren, da bo razvita koda delujoč del celotne programske rešitve. Ker vsak podtim skrbi za razvoj svojega dela, je lahko celotna slika arhitekture celotne programske rešitve precej slaba. Zato je dobro, če podjetje imenuje osebo, ki bo koordinirala njihovo delo in blažila arhitekturne dileme med različnimi podtimi. Oseba, ki jo določi vodstvo podjetja, mora poskrbeti, da podtimi razumejo širši kontekst projekta razvoja programske rešitve, zato da razvijejo kodo, ki je skladna s celotno programsko rešitvijo. Včasih podtimi potrebujejo še dodatno pomoč v obliki integratorja, ki skrbi, da se deli programske rešitve sestavijo v celoto. Intergratorji sodelujejo s podtimi od začetka projekta. Veliki projekti so bolj kompleksni, se pa dinamika v posameznem razvojnem podtimu od dinamike razvojnega tima pri manjših projektih razvoja programskih rešitev ne razlikuje veliko. Projekt razvoja programske rešitve lahko zaradi svoje specifičnosti in obsega potrebuje tudi dodatnega izvajalca testiranja, ki poskrbi za višjo raven kakovosti kode, ali tehničnega pisarja, ki dokumentira kodo in pripravi pomoč uporabnikom (Highsmith, 2003).

Agilne metode pogosto zahtevajo, da so v timih, ki so zadolženi za to, da se postopno razvije uporabna koda, poleg razvojnikov in izvajalcev testiranja še analitiki, arhitekti, tehniki, strokovnjaki za predmete urejanja, naročnik in vodja projekta (Slinger, 2006). Lastnost takih medfunkcijskih timov je v tem, da člani prepoznavajo in priznavajo znanje in veščine ostalih članov ter svoje znanje dopolnjujejo in kombinirajo z njihovim. Vsak od omenjenih ima v

timu svojo funkcijo, ki prispeva k razvoju programske rešitve. Razvoj programskih rešitev in njihova kakovost sta namreč močno odvisna prav od komuniciranja in kooperativnosti članov tima.

Timi morajo biti sposobni preučiti in razvrstiti zahteve ter napraviti obliko programske rešitve. Isti tim mora poskrbeti, da se razvije delujoča, testirana koda programske rešitve, tako da ustreza zahtevam, določenim s strani naročnika. Vse to pa mora potekati v kratki ponovitvi oziroma okvirju *timeboxa*. V vsaki ponovitvi morajo opraviti vse faze, ki jih klasične metode opravijo skozi celoten cikel razvoja programske rešitve.

Na koncu vsake ponovitve je potrebno organizirati srečanje, na katerem se izdela natančen pregled nad narejenim. Veliko avtorjev priporoča tudi ponovitvene oziroma izdajne sestanke. Na teh se predela uspehe in neuspehe, točke, kje so se pojavile težave, in zamisli, kako se bo te odpravilo v naslednjih ponovitvah. Ti sestanki morajo služiti tudi za sinhronizacijo z *backlogom*, ki ga sestavlja naročnik.

Menim, da je za podjetje primerno, če dopušča, da so člani v timu pretežno stalni. Člani, ki se poznajo, lažje sodelujejo in obvladajo medsebojna razmerja, kar povečuje njihovo zmožnost, da hitreje razvijajo programske rešitev, pa tudi, da so bolj kakovostne. Tak tim ne potrebuje dodatnega časa za vzpostavitev medsebojne komunikacije in pretoka znanja.

Da tim deluje v skladu z optimalnim poslovnim uspehom projekta, mora podjetje upoštevati, da imajo člani tima lastnosti, ki lahko prispevajo k uspehu ali neuspehu projekta razvoja programske rešitve. Prav tako morajo razumeti, da se lahko zmotijo ali da pri čem ne uspejo. Razvojni timi morajo čutiti, da jim podjetje zaupa. To pomaga k osebni rasti posameznih akterjev, predvsem pa se s tem dovoli rast in akumuliranje znanja. Ko se vzpostavi okolje zaupanja, se lahko začne dvigati zavest o moči tima in predanosti podjetju (Kittlaus, & Herde, 2011).

K uspehu projekta pripomore, če se organizira sistem dnevnih sestankov, kjer se srečujejo vsi člani tima, vključno s predstavnikom naročnika. Na teh sestankih se predstavi, kar je bilo narejeno, in načrt dela za isti dan. Pregled narejenega dela na dnevnih sestankih služi preverjanju in prilagoditvam kode programske rešitve v razvoju. To je pomembno za prepoznavanje možnih napak, ki bi potencialno ovirale izpeljavo začrtanega plana. Ti sestanki morajo ostati kratki. Na njih se ne rešuje potencialnih problemov.

3.6.3 Naročnik

Naročnik je tisti, ki se zaradi svojega notranjega okolja in organizacijske kulture izbere podjetje, ki organizira razvoj programskih rešitev po klasičnih ali agilnih metodah. Njegova vloga pri klasičnih metodah se bistveno razlikuje od vloge, ki jo ima pri agilnih metodah.

Naročnik ima pri klasičnih metodah možnost predaje svojih potreb in povratnih informacij le na začetku in koncu projekta, ko je program že nameščen v sistem, zato se pri uporabi klasičnih metod pogosto dogaja, da programske rešitve velikokrat ne vsebujejo funkcij, ki jih je naročnik pričakoval, oziroma da je končni produkt močno odstopal od naročnikovega naročila. Razvite programske rešitve šele pri implementaciji programa v sistem naročnika pokažejo, če in v kolikšni meri funkcije ustrezajo potrebam in željam naročnika.

Vsem agilnim metodam razvoja programske rešitve je skupna vključenost naročnika programske rešitve v projekt. Pričakuje se, da bo skozi celoten projekt zagotavljal sprotne povratne informacije in da bo za stalno sodelovanje z razvojnim timom zagotovil predstavnika oziroma kontaktno osebo, ki je pripravljena sodelovati s člani tima s polno predanostjo projektu. Imeti mora dovolj znanja o projektu in njegovih funkcijah, da lahko zagotavlja podporo razvojnim timom, posedovati mora dovolj pooblastil za sprejemanje odločitev pri projektu ter imeti dovolj znanja, da razume vprašanja razvojnih timov. Bohem in Turner (2004) imenujeta takega naročnika *"CRACK costumer"* (angl. *Collaborative, Representative, Authorized, Committed, Knowledgeable*).

Sodelovanje z naročnikom je pri agilnih metodah ključnega pomena za uspešno organiziran projekt razvoja programske rešitve. Naročnika se vključuje v organizacijo projekta pri pisanju uporabnikovih zgodb z dialogom o potrebnih lastnostih programske rešitve in določanju funkcij, ki se morajo razviti v prihodnosti.

Agilne metode lahko vključujejo naročnika v proces razvoja na različne načine. Pri metodi Scrum je naročnik del celotnega procesa. Računajo, da ima dovolj znanja za določitev ključnih predmetov in funkcij programa, da bo poglobljeno preverjal opravljeno delo in predajal povratne informacije razvojnemu timu, ki bo podatke uporabil za razvoj ustreznih funkcij za vsako ponovitev. Pri metodi XP pa mora naročnik zagotoviti in določiti prioritete

zahtev, ki jih nato z razvojnim timom preudarno uskladijo, tako da ustrezajo tako razvoju kot naročnikovim prioritetam.

Različne agilne metode uporabljajo različna poimenovanja za naročnike. Pri dveh metodah, ki sta pogosteje uporabljeni XP in Scrum, ga prva ga poimenuje *costumer*, druga pa *product owner* (Larman, 2003).

Sodelovanje naročnika pripomore k izdelavi vizije funkcij in funkcionalnosti programske rešitve. Nujno pa je njegovo sodelovanje potrebno za redne in ustrezne izdaje predmetov in funkcij v vsaki ponovitvi. Njegova vloga je nujna tudi pri izdelavi *backloga* s predmeti in funkcijami, ki jih potrebuje, ter pri izdelavi plana. S tem se prepreči predvsem razvoj programske rešitve, ki ne bi ustrezala njegovim zahtevam. Najboljši način je pisanje *backloga*, kjer so vidni želeni predmeti in funkcije, ki morajo biti razviti v prihodnjih ponovitvah. *Backlog* služi timu za preverjanje narejenega in planiranje bodočih ponovitev oziroma *timeboxov*. Ob koncu vsake ponovitve mora naročnik preveriti razvito programsko rešitev in podati povratno informacijo. Naročnikove povratne informacije nikakor ne smejo biti prezrte (Hoda, Noble, & Marshall, 2010b).

Za podjetja, ki naročajo programsko rešitev in imajo zelo dobro razdelane vse potrebne funkcije, ki jih program mora vsebovati, je verjetno primerna tudi izbira podjetja, ki razvija programske rešitve s klasičnimi metodami. Včasih pa se naročniki odločajo za izbiro podjetij, ki uporabljajo klasične metode, ker jim bolj kot prilagajanja urnikov in proračuna, ki so nujne pri agilnih metodah, ustreza fiksno določen strošek z določenim rokom in obsegom izvedbe projekta, kar je značilno za klasične metode. Lahko jim je odveč tudi to, da morajo pri agilnih metodah aktivno sodelovati pri vseh fazah projekta in prevzemati določeno mero odgovornosti za določanje prioritet funkcij programske rešitve. Nekaterim naročnikom predstavlja preveliko breme tudi konstantno motenje njihovega delovnega procesa s pogostimi implementacijami, ki jih agilne metode nujno potrebujejo, saj tako pridobivajo povratne informacije, potrebne za nadaljnji razvoj projekta (Highsmith, 2003).

Možno je tudi, da si bo naročnik izbral podjetje, ki organizira razvoj programskih rešitev po agilnih metodah, a ne bo hotel sodelovati v projektu ali pa bo na to slabo pripravljen. Vključenost naročnika programske rešitve v projekt je izredno odgovorna in naporna. Predstavnik naročnika, ki sodeluje v projektu, si težko privošči, da bi se stoodstotno posvetil projektu razvoja programske rešitve. Običajno mora opravljati še svoje redno delo. Poleg tega

mora dobro razumeti predmete in funkcije ter način uporabe naročene programske rešitve. Vedeti mora tudi, kako deluje sistem sedaj in kaj pričakovati ob koncu projekta. V teh primerih je možnost, da projekt ne bo imel želenega rezultata, velika. Raziskava Hoda et al. (2010), ki obravnava vlogo naročnika v agilnih projektih, je pokazala, da so projekti, pri katerih naročnik ni sodeloval po pričakovanjih, imeli večinoma negativen rezultat.

Mogoče je tudi, da naročnik, zlasti če je veliko podjetje, kaže odpor do sodelovanja z manjšim podjetjem in njegovim notranjim razvojnim nivojem. Zgodi se tudi, da jim poskušajo vsiliti svoj bolj tradicionalni način dela s timom. Vsiljevanje navadno naleti na odpor s strani razvijalcev, ki pa pogosto nimajo dovolj pogajalske moči, da bi izboljšali komunikacijo.

Naročniki si lahko svojo vlogo predstavljajo narobe. Sodelovanje pri projektu si predstavljajo kot delegiranje razvoja programske rešitve v vsaki fazi razvoja, ne upoštevajo pa svoje odgovornosti za sprejemanje odločitev. Ne zavedajo se, da njihovo sodelovanje pomeni predvsem pomoč razvojnemu timu pri sprejemanju odločitev in ne delegiranje dela.

Dobra lastnost agilnih metod je, da lahko dokaj hitro določijo napake in z minimalnimi stroški identificirajo projekte, ki kažejo na neuspeh projekta. Po raziskavi Hoda et al. (2010a) pa to marsikateremu naročniku ni všeč, saj nočejo priznati svojih napak.

Poudaril bi še problem razdalje oziroma bližino izvajalca in naročnika. Njuna oddaljenosti ni nujno ovira, predstavlja pa težavo za aktivno sodelovanje naročnika pri projektu. Oddaljenost naročnika ovira neposreden kontakt, stalno in sprotno izmenjanje informacij ter preverjanje razumljenega. Naročnik tudi lažje zaupa timu, ki ga pozna, zato imajo lokalni naročniki prednost. Bližina omogoča, da se naročnik osebno zglasi pri razvijalcih, kjer se lahko o kakršnikoli situaciji v miru pogovorijo in sprejmejo primerne odločitve.

Nerazumevanje je povezano s slabim določanjem prioritetnih funkcij. Ko naročnik predstavi funkcije enakovredne, razvojni tim ni zmožen določiti, kaj razviti prej, to pa preprečuje organizacijo in izdelavo realnih planov za naslednjo ponovitev.

Vodja projekta je tisti, ki pomaga pri pravilnem razumevanju vizije projekta, tako da postane posrednik med naročnikom in timom, ki ga organizira. Dobro sodelovanje in prilagajanje pa lahko zelo poveča poslovno vrednost razvite programske rešitve (Kittlaus, & Herde, 2011).

3.7 Lastnosti agilnih pogodb

Sklepanje pogodb je osnova za začetek tako majhnega kot velikega projekta. Pogodbe med naročnikom in podjetjem, ki organizira razvoj programskih rešitev po agilnih metodah, se razlikujejo od pogodb, ki jih naročniki sklenejo, če se odločijo za klasično metodo. Pogodbe, podpisane za razvoj programske rešitve z agilno metodo, morajo vključevati tveganje ene in druge stranke.

Pogodbe pri klasičnih metodah temeljijo na strinjanju podpisnikov zato, da se v poteku razvoja spreminjajo funkcije in predmeti, kar po eni stani predstavlja tveganje za naročnika, da pogodba ne bo realizirana, po drugi pa tveganje za izvajalca, da bo prekoračen obseg dela in s tem tudi finančni okvir projekta. Pogodbe ponavadi vsebujejo fiksne cene in fiksen obseg dela. Zato to predstavlja veliko tveganje za naročnika, da projekt sploh ne bo izveden ali pa bo izveden le delno. Taka fiksiranja lahko zaradi omejitev povzročijo nižanja kakovosti napisane kode, saj se pogosto dogaja, da podjetja zaradi varčevanja izvajajo manj testiranj ipd. Neredko pa naročniki zaradi tega ne dobijo tistega, kar so potrebovali oziroma naročili.

Arbogast, Larman & Vodde (2012) pravijo, da se morajo podpisniki pogodb, podpisanih za razvoj programske rešitve, ki se bo izvedla s klasičnimi metodami, zavedati dolge dobe, ki preteče, preden se izda za naročnika koristno kodo, pozne povratne informacije, dolgega roka plačil in težav, če se projekt pred koncem izvedbe prekine.

Z agilnimi metodami se razvija delujočo kodo s hitrimi ponovitvami in skupnimi odločitvami, s čimer se pravzaprav že zmanjšuje tveganja. Projekt je razdeljen na več manjših, neodvisnih ponovitev in šele po vsaki ponovitvi se določi, ali se gre z razvojem naprej. Agilne pogodbe so formalni okvir za sodelovanje med naročnikom in podjetjem. Za uspeh projekta pa je bolj kot kjer koli drugje potrebno medsebojno zaupanje in sodelovanje. Agilne metode v zameno ponujajo zmanjševanje tveganj z deljeno odgovornostjo med naročnikom in podjetjem. Zato se mi zdi pomembno, da pisec pogodb razume agilne metode, saj mora agilno razmišljanje ohraniti tudi v pogodbah. Če ima dovolj znanja o agilnih metodah, lahko v pogodbi povzame tudi vizijo projekta. Poglavitno pa je, da se v pogodbe vključi spreminjajoče se okolje in da se ga primerno oceni.

Arbograst et al. (2012) meni, da je struktura agilnih pogodb enaka kot pri klasičnih metodah. Razlika je predvsem v razumevanju operativnih procesov in dobavnih rokih. Pri agilnem razvoju programskih rešitev ni mogoče točno določiti datuma izdaje končnega produkta, končne cene in obsega, kar je iz pravnega vidika težje sprejemljivo. To, da v pogodbah nista opredeljena niti čas niti strošek programske rešitve, lahko potencialnega naročnika od agilnih metod tudi odvrne.

Obstajajo prakse pri uporabi agilnih metod, kjer v pogodbe vključijo tudi stroške, obseg in čas, vendar se jih podjetja raje izogibajo. V resnici gre le za oceno stroškov, obsega in časa. Za določitev obsega Arbogast et al. (2012) priporočajo uporabo progresivnih pogodb (angl. *progressive contracts*), ki imajo povsem fleksibilen obseg ali uporabo pogodb s ciljnimi stroški (angl. *target-cost contracts*).

Pri projektih, za katere se sklepa pogodbe s ciljnimi stroški, se poizkuša v prvi fazi čim bolje oceniti obseg in podrobnosti razvoja programske rešitve, identificirati in analizirati, kaj vse v projektu je pomembno za stranko in kaj za podjetje, ter čim bolje oceniti možne stroške sprememb ali povečanja obsega projekta. Nato se lahko določi ciljne stroške. Pri tem si podjetje izračuna tudi ciljni dobiček (angl. *target profit*), ki ga vračuna v ciljne stroške. Podjetje naročniku sproti poroča o detajlih, ki sodijo v ciljne stroške. Razlika, ki nastane pri dejanskih stroških in ciljnih stroških, se obravnava kot prilagoditev naročnika in podjetja. Če so stroški v projektu višji, kot so bili predvideni, se sicer poveča strošek naročnika, hkrati pa se zmanjša dobiček podjetja. Naročnik in podjetje tako delita tveganja sprememb višine stroškov.

Za progresivne pogodbe je značilen povsem fleksibilen obseg projekta. Obseg projekta se opredeli za vsako ponovitev posebej. S temi pogodbami stranki dogovorita sistem za določanje cen, ne da bi pri tem omejili obseg projekta. Znanih je kar nekaj sistemov za določanja cene. Eden je fiksiranje cene na delovno enoto (angl. *fixed price per unit*), drugi je fiksiranje cene na ponovitev (angl. *fixed price per iteration*), najpogosteje pa se uporablja omejevanje časa in materiala (angl. *capped T&M*). Tako si podjetje zagotovi povračilo stroškov že zgodaj v projektu, stranka pa je proti koncu zavarovana pred stroški, ki presegajo postavljen limit.

Progresivne pogodbe so bolj značilne pri dolgoročnih sodelovanjih, ker temeljijo na zaupanju. Pogodbe se podpisuje postopno, odvisno od faze projekta, potreb naročnika in podjetja. Gre

za več pogodbenih faz oziroma tako imenovani večfazni model (angl. *multi-phase model*). Običajno so začetne pogodbe bolj definirane, sicer pa lahko variirajo glede na določanje fiksiranega časa, fiksirane cene in delno fiksiranega obsega projekta. Gre za uporabo katere koli omejitve v pogodbah, ki se jih podpisuje v različnih fazah projekta. S takim načinom poizkuša podjetje postopoma pridobiti zaupanje naročnika, kar vodi k zmanjševanju nadzora nad omejitvami, to pa pripomore k podpisovanju pogodb bolj progresivne narave.

Larman (2003, str. 18) opisuje primer uporabe večfaznega modela z dvema pogodbenima fazama. V prvi fazi, ko je na začetku projekta največ neznank in sprememb, svetuje podpis pogodbe s fiksiranim časom, fiksirano ceno in delno fiksiranim obsegom projekta. Tako se naročnik in podjetje lažje pripravita na drugo fazo, v kateri se cene in obseg fiksirajo na podlagi pridobljenih zanesljivih informacij iz prve faze. Kot pravi Larman (2003), pa se lahko v prvi fazi določi tudi pogodbo s fiksno ceno, fiksnim časom in prilagodljivim obsegom, v drugi fazi pa progresivno pogodbo, kjer se določi omejen čas in denar z neobvezujočim *backlogom*. Neobvezujoč *backlog* je dobro sestaviti, preden se podpiše pogodbo, saj s tem pripomoremo k postavljanju omejitev in jasnejši viziji projekta.

Pri velikih projektih je tveganj več, zato so tudi pogodbe kompleksnejše, za manjše projekte pa je mogoče uporabiti tudi standardizirane pogodbe, ki jih je mogoče najti tudi na spletu (Valtech, Tehch Works, Alistar Cockburn).

SKLEP

Agilne metode dajejo nove možnosti razvoja programskih rešitev. Zanimalo me je predvsem, kako se podjetje, ki razvija programske rešitve za znanega naročnika, organizacijsko prilagodi na agilne metode. Ugotavljam, da prehod na uporabo agilnih metod ni mogoč brez radikalnih prilagoditev in sprememb tako pri delu in razmišljanju zaposlenih kot tudi v organizacijski strukturi podjetja. Odločitev morajo sprejeti in razumeti vsi v podjetju. Projekti, organizirani s pomočjo agilnih metod, ne morejo biti uspešni zgolj s tehničnimi spremembami v organizaciji podjetja - spremeniti se mora predvsem kultura podjetja.

Podjetja morajo dobro pretehtati svoje zmožnosti in sredstva, preden se odločijo za organiziranje razvoja programskih rešitev z agilnimi metodami. Vodstvo mora metodo, za katero se odloči, popolnoma prevzeti, kar pa pomeni, da se spremenijo tudi relacije med zaposlenimi, njihove odgovornosti in kompetence ter tudi hierarhija sprejemanja odločitev.

Prav tako ugotavljam, da je za podjetje zelo pomembno, če ima sposobnost visoke organizacijske usmerjenosti k izpolnitvi in dosežkom projektov. Z visokimi pričakovanji lažje spodbuja sebe in druge k doseganju višjih vrednosti projektov razvoja in hkrati z iniciativo viša poslovno vrednost celotnemu podjetju.

Vsako podjetje se sooča s spremembami drugače, težje pa jih prevzamejo tista, ki spreminjajo svojo obliko organizacije projektov razvoja programskih rešitev, kot podjetja, ki se formirajo na novo. Podjetja, ki se reorganizirajo, morajo dobro razumeti agilne metode in svoje notranje okolje, da lahko uspešno organizirajo projekt in ga pripeljejo do višje poslovne vrednosti. Organizacija projekta s pomočjo agilnih metod predstavlja podjetju vsakič nove izzive, saj se vsakokrat vključijo nove spremenljivke, tudi če gre za projekt s podobnim zaključkom.

Ugotavljam, da je uporaba agilnih metod v praksi pokazala velike prednosti, predvsem pa pridobila zaupanje podjetij ter naročnikov. Po podatkih raziskave Verison1 (2013) število podjetij, ki se odločijo razvoj programskih rešitev z uporabo agilnih metod, raste. Leta 2012 se je za agilne metode odločilo že več kot 50 odstotkov podjetij. Najpogosteje uporabljena agilna metoda je Scrum, se pa v zadnjih letih povečuje priljubljenost tako imenovanih hibridov. Menim, da rast uporabe agilnih metod dokazuje, da se zaradi zmanjšanja tveganj in izboljšav v poslovni vrednosti napisane kode uporaba agilnih metod za razvoj programskih rešitev izplača.

V svoji nalogi sem analiziral različna dela in se osredotočil na programske rešitve, ki se razvijajo za znanega naročnika. V prvem delu sem opisal lastnosti programskih rešitev. V nadaljevanju sem opredelil uporabo metod za organizacijo razvojno-programskih rešitev v podjetju ter klasične in agilne metode. Skozi delo sem primerjal uporabo agilnih in klasičnih metod, kar me je vodilo do ugotovitve, da sta to dve povsem različni obliki organiziranja razvoja programskih rešitev.

Pri agilnih metodah uporaba ponovitev in postopnega razvoja zmanjšuje tveganja tako za naročnika kot podjetje. Projekt se lahko ustavi po vsaki ponovitvi. Druga stvar, ki je pri agilnih metodah zelo pomembna, je sodelovanje in prilagajanje med naročnikom in razvojnimi timi. Za uspeh projekta pa je bolj kot kjer koli drugje potrebno medsebojno zaupanje in sodelovanje.

Proces integracije poteka skozi celoten projekt. Reševanje nastalih težav se odvija sproti na dnevnih sestankih. V primeru neprimernih rešitev se proces lahko spremeni po končani ponovitvi z dopolnivami v *backlogu*. Prednost se daje uporabi *timeboxov*, v katerih se v eni ponovitvi razvije uporabna koda v omejenem obsegu in času. Idealno se ponovitev odvije v dveh do treh tednih. Veliko zmanjšanje tveganja predstavlja tudi sprotno testiranje in vodenje kakovosti, ta se začne že v fazi planiranja ponovitve, ko so znane zahteve. To zmanjšuje možnost, da se pojavijo zastoji proti koncu ponovitve. Vsaka ponovitev daje naročniku novo funkcionalno vrednost in traja povprečno dva do tri tedne.

Agilne metode na novo postavljajo organizacijo hierarhije, ki postane bolj odprta. Vloge zaposlenih se spremenijo. Vodja projekta postane bolj posrednik, ki skrbi, da se sledi viziji projekta. Razvojniki postanejo člani samoorganizacijskih razvojnih timov. Samoorganizacijski timi sodelujejo pri sprejemanju odločitev, kar jim daje večjo vlogo pri projektu. Agilne metode ne dajejo veliko poudarka dokumentaciji. Znanje je akumulirano predvsem v glavah članov razvojnega tima. Novo vlogo pri organizaciji razvoja programske rešitve z uporabo agilnih metod ima tudi naročnik. Postati mora del tima, z vodenjem *backloga* pa postane vir informacij za razvoj. Njegovo sodelovanje pogojuje končni rezultat.

Z uporabo agilnih metod pogodbe niso več glavni vir zmanjševanja tveganj. Menim, da si morajo stranke, ki pogodbo podpisujejo, zaupati, saj so namreč bolj formalni okvir za sodelovanje med naročnikom in podjetjem. Priporoča se spreminjanje pogodb glede na stopnjo razvoja programske rešitve, saj se zaupanje ponavadi gradi skozi projekt. Prednost se daje prilagodljivosti med naročnikom in timi.

Menim, da je odločitev uporabe agilnih metod za razvoj programskih rešitev dobra izbira. Ugotavljam, da lahko podjetja z uporabo agilnih metod zagotavljajo kakovostno rešitev v krajšem času kot z uporabo klasičnih metod. V teoriji se zdijo nekateri pristopi nekoliko utopični, vendar praksa kaže, da so rezultati uporabe agilnih metod zelo dobri, kar nakazuje na večjo uporabo agilnih metod.

Podjetjem, ki se lotevajo kompleksnejših projektov in zahtevajo organizacijo večjega tima, bi pri razvoju programske rešitve predlagal uporabo agilnih metod. Njihove dobre strani so, da podpirajo okolje, kjer lahko vsi deležniki v timu prispevajo k razvoju, in da mnenje vsakega člana tima šteje, kljub temu pa ostaja dovolj herarhije, da so projekti izvedeni kakovostno.

Ocenjujem, da je sprejetje agilne metode za podjetja na začetku zelo težko, ko pa celotno podjetje začne »dihati« agilno in se poveča zaupanje do agilnih metod med zaposlenimi, so hitro vidni tudi boljši rezultati in večje zadovoljstvo strank. Raste tudi zaupanje naročnikov, zaradi česar pričakujem, da se bo v prihodnosti med podjetji, ki razvijajo programske rešitve, uporaba agilnih metod razširila, verjetno tudi na organizacijo drugačnih inženirskih ali celo neinženirskih projektov.

LITERATURA IN VIRI

1. Abrahamson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods. Review and Analysis. V *VVT Publications 478*. Najdeno 2. februarja na spletnem naslovu http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf.

2. Agilne metode razvoja programske opreme. (b.l.). V *Wikipedia the free enciklopedia*. Najdeno 11. aprila 2014 na spletni strani http://sl.wikipedia.org/wiki/Agilne_metode_razvoja_programske_opreme.

3. Antony, R. N. (1965). *Planning and Control Systems: A Framework for Analysis. Division of Research.* Boston: Division of Research, Harvard Business Scool.

4. Arbogast, T., Larman, C., & Vodde, B. (2012). *Practices for Scaling Lean & Agile Development: Large, Multisite, & Offshore Product Development with Large-Scale Scrum* (5^{th}ed.). b.k.: Addison Wesley.

5. Beck, K., & Fowler, M. (2000). *Planning Extreme Programming*. b.k.: Addison Wesley.

6. Benediktsson, O., Dalcher, D., & Thorbergsson, H. (2004). Working With Alternative Development Lifecycles: A Multiproject Experiment. *Information Systems Devleopment, 1*(1), 463-476.

7. Benediktsson, O., Dalcher, D., & Thorbergsson, H. (2005). Development Life Cycle Management: A Multiproject Experiment. *Engineering of Computer-Based Systems* (str. 289-296).

8. Boehm, B., & Turner, R. (2005*).* Management Challenges to Implementing Agile Processes in Traditional Development Organisations. *Software, 22*(5), 30-39.

9. Boehm, B.(2002). Get Ready for Agile Methods, With Care. *Computer 32*(1), 64-69.

10. Carr, N. G. (2003). IT doesnt matter. Najdeno 26 maja 2013 na spletnem naslovu https://hbr.org/2003/05/it-doesnt-matter.

11. Chan, F. K. Y., & Thong J. Y. L. (2008). Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems, 46*(4), 803-804.

12. Chow, T., & Cao, D. (2008). A survey study of critical success factors in agile software projects. *Journal Of Systems and Software, 81*(6), 961-971.

Dalcher, D., Benediktsson, O., & Thorbergsson, H. (2005).Development life cycle: A multiproject experiment. V *Engineering of Computer-Based Systems, 12(1),* 289-296.

13. Dimitrijevic, D. (2010). Modeli Softverskog Procesa Razvoja- Softversko inzenjerstvo. Najdeno 26. maja 2013 na spletnem naslovu http://www.slideshare.net/thecoco/poslovna-inteligencija-15556794.

14. Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems and Software 85*(6), 1213-1221.

15. Drury, M., Conboy, K., & Power, K. (2012). Obstacles to decision making in Agile software development teams. *The Journal Of Systems and Software, 85*(6), 1239-1254.

16. Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile development: A systematic review. V *Information Software Technology, 50*(9/10), 833-859.

17. Fowler, M. (2005).The New Methodology. Najdeno 15. februarja 2013 na spletni strani http://www.martinfowler.com/articles/newMethodology.html.

18. Fowler, M. (2006). Continuous Integration. Najdeno 20. marca 2014 na spletni strani http://martinfowler.com/articles/continuousIntegration.html.

19. Goldman, S., Nagel, K., & Preiss, K. (1994). *Agile Competitors and Virtual Organisations*. b.k.: Wieley.

20. Goldstein, Z., &Petrie, D. (1993) Finding The Middle Ground Management Strategy for Software Development. V *Information Management & Computer Security, 18*(3), 185-197.

21. Highsmith, J. (2003). Principles and Tools, Agile Project Managment. *Principles and Tools, 4*(2).

22. Hoda, R., Noble, J., & Marshall, S. (2010a). The impact of inadequate costumer collaboration on self-organizing Agile teams. *Information and Software Technology, 53* (5), 521-534.

23. Hoda, R., Noble, J., & Marshall, S. (2010b). Organising Self-Organizing Teams. Najdeno 12. marca 2014 na spletnem naslovu http://homepages.mcs.vuw.ac.nz/~elvis/twikipub/Main/RashinaHoda/OSOT.pdf.

24. Hogg, M. (2013). SW development: product or service? Najdeno 18. aprila 2014 na spletnem naslovu http://www.embedded.com/electronics-blogs/agile-adventures-in-an-embedded-world/4407269/Software-development--product-or-service-.

25. Iivari, J., & Iivari, N. (2010). The relationship between organizational culture and the deployment of agile methods. *Information and Software Technology, 53*(5), 509-520.

26. Jermić, Z. (2011). Životni ciklus i metodologije razvoja softvera. Najdeno 26. maja 2013 na spletnem naslovu http://www.slideshare.net/borbrentin/savedfiles?s_title=t-2-zivotni-ciklus-i-metodologije-razvoja-softvera&user_login=jeremycod.

27. Kane, M. (2013). An overview of Agile Contracts. V *Scrumology.* Najdeno 1. marca 2014 na spletnem naslovu http://scrumology.com/an-overview-of-agile-contracts/.

28. Karlström, D., & Runeson, P. (2006). Integrating Agile Software Development into Stage-Gate managed product Development. *Empirical Software Engineering, 11*(2), 203-225.

29. Kittlaus, H. B. (2004). *Software-Produkt Managment*. Berlin: Springer-Verlag Berlin Heidelberg New York.

30. Kittlaus, H. B. (2012). Software Product Management and Agile Software Development - Conflicts and Solutions. Najdeno 1. marca 2014 na spletnem naslovu http://www.innotivum.de/Kittlaus_Software_Product_Management_and_Agile_Development_120818.pdf.

31. Kittlaus, H. B., & Herde, H. (2011). Agile Software-Entwicklung braucht ein agiles Projektmanagment. Najdeno 15. marca 2013 na spletnem naslovu http://www.heise.de/developer/artikel/Agile-Software-Entwicklung-braucht-auch-ein-agiles-Projektmanagement-1598135.html.

32. Larman, C. (2003). *Agile and Iterative Development - A Manager's Guide*. Boston: Addison-Wesley.

33. *Manifesto for Agile Software development*. Najdeno 1. marca 2013 na spletni strani http://agilemanifesto.org/.

34. McAvoy, J., & Butler, T. (2009). The role of project management in ineffective decision making within Agile software development projects. V *European Journal of Information Systems, 18*(1), 372-383.

35. Melo, C., de O., Cruizes, D., S., Kon, F., & Conradi, R. (2012). Interpretative case studies on agile team productivity and management. *Information and Software Technology*, 55(2), 412-427.

36. Meyer, B. (2001, Oktober). Software: Product or Service?. Najdeno 18. Aprila 2014 na spletnem naslovu http://se.ethz.ch/~meyer/publications/softdev/product-service.pdf.

37. Mistra, S., C., Kumar, V., & Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management, 27*(4), 451-474.
38. Moe, N. B., Aybüke, A., & Dyba, T. (2011). Challenges of shared decision-making: A multiple case of agile software development. *Information and Software Technology, 54*(8), 853-865.

39. Moe, N., B., Dybå, T., & Dingsøyr, T. (2009). A teamwork model for understanding an agile team: A case study of Scrum project. *Information and Software Technology, 52*(5), 480-491.

40. Moniruzzman, A. B. M. & Hossain, S. A., (2012). Comparative Study on Agile softwaredevelopment methodologies. Najdeno 19. Februarja 2013 na spletnem naslovu http://arxiv.org/pdf/1307.3356.pdf.

41. Nerrur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. V *Comunications of the ACM 48*(5), 72-78.

42. Project Managment Institute (2013). *A Guide to the Project Management Body of Knowledge (5th ed.).* Pennsylvania: Project Managment Institute.

43. Scrum (development). (b.l.). V *Wikipedia the free enciklopedia*. Najdeno 19. februarja 2013 na spletni strani http://en.wikipedia.org/wiki/Scrum_(development).

44. Scrum Alliance – The Manager's Role in Agile (2008). Najdeno 08. maja 2013 ma spletni strani http://www.scrumalliance.org/community/articles/2008/july/the-manager-s-role-in-agile.

45. Sliger, M. (2006). A Project Manager Survival Guide to Going Agile. *Rally Software Development Corporation.* Najdeno 14. marca 2013 na spletnem naslovu http://www.rallydev.com/documents/rally_survival_guide_0307.pdf.

46. Software as a service. (b.l.). V *Wikipedia the free enciklopedia*. Najdeno dne 19. februarja 2013 na spletni strani http://en.wikipedia.org/wiki/Software_as_a_service.

47. Software Development Methodology. (b.l.). V *Wikipedia the free enciklopedia*. Najdeno dne 19. februarja 2013 na spletni strani http://en.wikipedia.org/wiki/Software_development_methodologies.

48. Software development process. (b.l.). V *Wikipedia the free enciklopedia*. Najdeno dne 19. februarja 2013 na spletni strani http://en.wikipedia.org/wiki/Software_development_life_cycle.

49. Software. (b.l.). V *Wikipedia the free enciklopedia*. Najdeno dne 19. februarja 2013 na spletni strani http://en.wikipedia.org/wiki/Software.

50. Spann, D. (2006). Agile Manager Behaviors: What to Look For and Develop. *Agile Project Managment Advisory Service, 7*(9).

51. Stare, A. (2011). Projektni Management. Najdeno 17. maja 2013 na spletnem naslovu http://projektni-management.si/author/aljazstare/.

52. Subhas, C. M., Kumar, V. ,& Kumar, U. (2012) Agile Software Development Practices: Evolution, Principles and Criticisms. *International Journal of Quality and Reliability Management 29*(9), 972-980.

53. Verbinc, F. (1997). *Slovar tujk (12*). Ljubljana: Cankarjeva založba.

54. Versionone. (2013). 7th Annual State of Agile development survey. Najdeno 28. januarja 2014 na spletnem naslovu http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

55. Vinekar, V., Slinkman, C. W. & Nerur, S. (2006). Can Agile and Traditional Systems Development Approaches Coexsist? An Ambidextrous View. V *Information Systems Development, 23*(3), 31-42.

56. Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2010). The agile requrements refinery: Applying SCRUM principles to software product management. *Information and Software Technology, 53*(1), 58-70.

57. Williams, L. (2010). Agile Software Development - Methodologies and practices. *Advances in computers, 80*(1), 1-44.