UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

ANALIZA PRIMERNOSTI SCRUM METODOLOGIJE ZA RAZVOJ CELOVITIH INFORMACIJSKIH REŠITEV

Ljubljana, julij 2016 MATJAŽ ŠEGA

IZJAVA O AVTORSTVU

Podpisani Matjaž Šega, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom ANALIZA PRIMERNOSTI SCRUM METODOLOGIJE ZA RAZVOJ CELOVITIH INFORMACIJSKIH REŠITEV, pripravljenega v sodelovanju s svetovalcem dr. Mirom Gradišarjem

IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;

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

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

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

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

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

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

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

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

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

V Ljubljani, dne 12.7.2016 Podpis študenta:_____

KAZALO

KAZALO TABEL

KAZALO SLIK

UVOD

Problematika magistrskega dela ter predmet raziskave. Vse od zgodnjih 70-ih let nastajajo novi procesi razvoja programske opreme (v nadaljevanju programski procesi), ki poskušajo zajeti in organizirati znanje o razvoju programske opreme in sistemov. Tako lahko razvijalci programske opreme danes izberejo enega od številnih pristopov k razvoju, naj si bo to rigiden plansko-usmerjeni proces ali pa vitka agilna metodologija. Od začetka 21. stoletja srečujemo dva osnovna tokova. Tradicionalni (ali bogati) procesi naslavljajo celoten življenjski cikel programskega projekta, na primer z zagotavljanjem obsežnih smernic, standardiziranih procedur, predlog za planiranje projekta in vmesnikov do nadaljnjih organizacijskih procesov. Agilne metodologije pa želijo zmanjšati programski proces na minimum, v izogib "birokraciji" in da bi uporabnikom zagotovile le toliko pravil in smernic, kot jih je potrebno za izvedbo projekta (Theocharis, Kuhrmann, Münch, & Diebold, 2015)

Leta 2001 so se predstavniki vodilnih agilnih metodologij (Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming) in drugi simpatizerji potrebe po alternativi dokumentno-gnanemu, "težkemu" procesu razvoja programske opreme sestali na konferenci v Chicagu in spisali Manifest agilnega razvoja programske opreme (angl. *Agile Manifesto*). V tem manifestu so opredelili osnovne vrednote agilnega razvoja:

- posamezniki in interakcije pred procesi in orodji,
- delujoča programska oprema pred vseobsežno dokumentacijo,
- sodelovanje s stranko pred pogodbenimi pogajanji,
- odziv na spremembe pred togim sledenjem načrtom (Agile Alliance, 2001)

Agilne metodologije v razvoj vpeljujejo koncepte, ki na različnih točkah razvojnega procesa izboljšujejo možnosti za uspeh projekta. Osnovni koncept agilnih metodologij je razvoj v iteracijah, pri čemer je rezultat vsake iteracije delujoča rešitev. Naročnik lahko tako sproti preverja, ali gre razvoj produkta v pravi smeri, torej ali rešitev ustreza zahtevam, na podlagi delujočega prototipa pa tudi lažje preveri smiselnost samih zahtev in jih po potrebi prilagodi. Na ta način se zagotovi, da je rezultat razvoja programska rešitev, ki jo naročnik potrebuje, ne pa rešitev, ki jo je na podlagi pomanjkljivih informacij specificiral na začetku projekta.

Na nivoju razvojne ekipe je cilj agilnih metodologij izboljšati sodelovanje, koordinacijo in komunikacijo. Trenutno najbolj razširjena agilna tehnologija scrum (West, Gilpin, Grant, & Anderson, 2011) na primer predpisuje redne sestanke oziroma delavnice, na katerih se ekipa seznani s predvidenimi delovnimi nalogami, se o njih pogovori in skupaj oceni njihovo zahtevnost. Pred vsako iteracijo razvojna ekipa kot celota sprejme zavezo o delovnih nalogah, ki jih bodo opravili v iteraciji. V procesu iteracije se ekipa redno srečuje na kratkih sestankih, na katerih se vsi člani ekipe seznanijo, s čim se trenutno ukvarja posamezen član

ekipe. Poleg širjenja informacij je glavni cilj teh sestankov identificirati morebitne težave, tako da se jih lahko reši, še preden prerastejo v probleme.

Agilne metodologije prav gotovo niso primerne za vsak projekt. Boehm in Turner (2003) ter Hopkins in Jenkins (2008) ugotavljajo, da so najbolj primerne za enostavnejše projekte v dinamičnem in enostavnem okolju - tipično za popolnoma nove (angl. *greenfield*) projekte, torej projekte, pri katerih se ni potrebno ozirati na podedovane sisteme.

Po drugi strani Nerur, Cannon, Balijepally in Bond (2010) trdijo ravno nasprotno. Z rastjo pomembnosti informacijske tehnologije v moderni ekonomiji se razvijalci programske opreme spopadajo z vedno kompleksnejšimi izzivi. Zgodnji razvojni projekti so se ukvarjali predvsem z razvojem sistemov za avtomatizacijo rutinskih, dobro strukturiranih opravil. Primer takih projektov so sistemi za procesiranje transakcij in zgodnji vodstveni informacijski sistemi (angl. *management IS*). Prvi so bili namenjeni zbiranju in shranjevanju transakcijskih podatkov, drugi pa so na podlagi teh podatkov pripravljali različna poročila za izvajanje monitoringa in nadzora. Ti projekti so imeli malo interesnih skupin in še te so bile enake miselnosti, zato njihova obširna vključenost v razvoj ni bila niti pričakovana, niti potrebna. Danes razvojni projekti rešujejo izzive precej bolj nestrukturiranih področij. Obseg projektov se je razširil z ozkih in dobro definiranih domen (npr. obračuna plač) na sisteme, ki se dotikajo veliko ali celo vseh vidikov poslovanja tako znotraj (npr. ERP), kot tudi zunaj meja organizacije (npr. virtualne integracije, oskrbovalna veriga). Vsebinski kontekst je tako precej bolj strateški, kar zvišuje negotovost in kompleksnost razvoja. Prav tako se morajo pri takih projektih razvijalci usklajevati z večjim številom raznolikih interesnih skupin. Te niso nujno med seboj usklajene glede ciljev programske opreme, kar lahko vodi v podajanje konfliktnih informacij (Nerur et al., 2010). Z agilnim pristopom je usklajevanje med interesnimi skupinami lažje in posledično je produkt boljši.

Dejansko se agilni pristop uporablja na vse več projektih implementacije celovitih informacijskih rešitev. Trąbka in Soja (2014) sta v študiji dveh implementacij ERP sistemov primerjala tradicionalni in agilni pristop. Po njunem mnenju na izbiro pristopa najbolj vpliva temeljitost predimplementacijske analize. Če je v fazi analize pripravljeno podrobno oblikovanje rešitve, potem je smiseln tradicionalni slapovni (angl. *waterfall*) pristop implementacije. Če pa je v fazi analize izvedena zgolj analiza želenih funkcionalnosti in želenega obnašanja sistema (angl. *business analysis*), ne pa tudi oblikovanje rešitve, naročnik ne pridobi znanja o dejanskem sistemu, npr. videzu uporabniškega vmesnika. Tradicionalni slapovni pristop, pri katerem pridejo uporabniki prvič v stik s sistemom tik pred prehodom v produkcijo, tako navadno rezultira v dodatnih zahtevah, ki pa so velikokrat opuščene zaradi pomanjkanja virov. V najboljšem primeru naročnik uporablja sistem z vedenjem, da ne ustreza vsem njegovim potrebam, v najslabšem primeru pa lahko naročnik celo odstopi od projekta. V splošnem lahko uporaba agilnega pristopa zmanjša tveganje za neuspeh projekta implementacije ERP sistema.

Primernost agilne metodologije je odvisna tudi od sestave razvojne ekipe. Scrum metodologija (Schwaber, Sutherland, & Beedle, 2013) predvideva večfunkcijsko ekipo, v kateri so pokrita vsa znanja, potrebna za izvedbo naloge. Optimalna ekipa mora biti dovolj majhna, da je še okretna, ter dovolj velika za dokončanje zadostne količine dela v enem sprintu. Če ekipa vsebuje manj kot tri člane, zmanjšanje interakcije rezultira v manjših prirastkih produktivnosti. Ekipa z več kot devetimi člani pa zahteva preveč koordinacije in generira preveč kompleksnosti, da bi jo lahko upravljali z empiričnim procesom. Boehm in Turner (2003) ugotavljata, da agilne ekipe temeljijo na visoko usposobljenih članih in težko shajajo z večjim številom manj kvalificiranih članov.

Trąbka in Soja sta v svoji študiji (2014) poudarila še, da je izbira pristopa odvisna tudi od tipa naročnika, predvsem od panoge naročnikovega podjetja ter IT osveščenosti njegovih zaposlenih. Agilni pristop od projektne ekipe zahteva zavedanje življenjskega cikla razvoja sistema, saj je to nujno za zanesljivo ocenjevanje prototipov. Če so zaposleni dnevno v stiku z IT temami in poznajo relevantne koncepte in mehanizme, potem je to dobra osnova za uspešen agilni pristop. Avtorja sta v zaključku študije postavila hipotezo, da agilni pristop predstavlja večji izziv in je lahko povezan z večjim tveganjem v primeru naročnikov, ki delujejo v manj tehnološko intenzivnih panogah.

Več raziskav ugotavlja, da se na ERP projektih velikokrat uporablja hibridni pristop. Agilnost je vpeljana le v fazo razvoja, iz slapovne metodologije pa se ohranita koncepta faze analize in oblikovanja. West et al. (2011) so v svoji raziskavi ugotovili, da razvojna podjetja sicer usvojijo ideje agilnosti, vendar ne dosežejo vizije agilnih metodologij. Tako se dogaja, da posamezni člani ekipe delajo na več projektih hkrati, da se delovne naloge prestavljajo iz iteracije v iteracijo, da se za vsak projekt ekipa sestavi na novo itd. Razvojna podjetja so prek desetletij zgradila kopico tradicionalnih vodstvenih orodij - proračune, načrte, arhitekturne smernice, specifikacije zahtev - kar otežuje premik koncepta agilnosti od eksperimenta do načina, kako organizacija opravlja svoje delo. Tako se organizacije povrnejo k tradicionalnemu pristopu, saj:

- plani spodbudijo financiranje, agilni projekti pa sicer spodbujajo uporabo načrta, vendar je ta bistveno manj podroben kot tisti, ki je potreben za tradicionalni pristop;
- arhitekturo in oblikovanje delajo drugi oddelki, agilne ekipe pa naj bi za boljše in hitrejše delovanje vsebovale vse potrebne vloge;
- projektno vodstvo kot orodje za zmanjšanje tveganja zahteva pripravo seta dokumentov, medtem ko agilni pristop tveganje blaži tako, da razvojna ekipa čim prej dostavi verzijo produkta, na osnovi katere lahko potem primerne osebe ocenijo tveganje v kontekstu delujočega programa;
- razvijalci niso vključeni v proces zajema zahtev, kar je za enostavne probleme lahko dober pristop, vendar pa v veliko situacijah tehnologija bistveno vpliva na definicijo zahtev in je priprava dokumentov, ki opisujejo poslovni problem zgolj s strani poslovnih analitikov, praktično nesmiselna.

Namen raziskave. V okviru magistrskega dela želim preučiti primernost agilne metodologije scrum za razvoj celovitih informacijskih rešitev. Analiziral bom podjetje, v katerem sem zaposlen. Na enem od oddelkov, ki se ukvarjajo z implementacijo celovitih informacijskih rešitev, je bila agilna metodologija scrum vpeljana leta 2014, pred tem pa je bila v uporabi tradicionalna slapovna metodologija.

Zanima me, v kolikšni meri je oddelek usvojil različne vidike scrum metodologije, s kakšnimi izzivi se je pri tem srečal ter kako se je z njimi spopadel.

Zaradi narave razvoja oz. implementacije celovitih programskih rešitev pričakujem, da je bilo pri vpeljavi scrum metodologije sklenjenih več kompromisov oz. da se je kot najboljša izkazala kombinacija agilne in tradicionalne metodologije razvoja.

Cilj raziskave. Cilji magistrskega dela so:

- analiza implementacije agilne metodologije scrum v ERP oddelku podjetja,
- ugotovitev, v kolikšni meri so odkloni od smernic metodologije posledica posebnosti razvoja celovitih informacijskih rešitev, in
- ocena smiselnosti vpeljave agilne metodologije tudi v drugi oddelek, ki se prav tako ukvarja z implementacijo celovitih informacijskih rešitev.

Metode dela. Pri izdelavi magistrskega dela sem uporabil znanje, pridobljeno na dodiplomskem študiju računalništva in informatike ter na podiplomskem študiju podjetništva, in večletne praktične izkušnje z implementacijo celovitih programskih rešitev.

Preučil sem domačo in tujo strokovno literaturo in spoznal teoretično podlago uporabe agilnih metodologij pri razvoju celovitih informacijskih rešitev. Z metodami analize stanja sem ugotovil stopnjo integracije agilne metodologije v razvojni proces ERP oddelka podjetja. Opravil sem več intervjujev s ključnimi osebami in na podlagi zbranih informacij našel razloge za odstopanja uporabljene metodologije od osnovne scrum metodologije.

Na koncu sem opravil še analizo razlik med obema oddelkoma za implementacijo celovitih informacijskih rešitev ter na podlagi teh razlik ocenil smiselnost in upravičenost vpeljave agilne metodologije tudi v drugi oddelek za celovite rešitve.

Struktura. Magistrska naloga je razdeljena na pet poglavij.

V uvodnem poglavju predstavim teoretična izhodišča metodologij razvoja programske opreme. Pri tem podrobneje predstavim agilno metodologijo scrum in pojem celovitih informacijskih rešitev.

V drugem poglavju najprej predstavim obravnavano podjetje ter oddelek AX, ki je vpeljavo agilne metodologije scrum že izvedel. Nato v treh podpoglavjih predstavim podrobnosti implementacije scruma v oddelek AX. V prvem in drugem podpoglavju na podlagi intervjujev s ključnima članoma predstavim dve ekipi oddelka in njune prilagoditve scruma. V tretjem podpoglavju predstavim pogled direktorja oddelka na implementacijo scruma v oddelek.

V tretjem poglavju opravim analizo metodologije, ki je nastala z vpeljavo scrum konceptov v oddelek AX. Podrobno analiziram odstopanja od pravil scruma in razloge zanje. Na koncu poglavja podam predloge oddelku AX za izboljšave metodologije.

V četrtem poglavju predstavim oddelek NAV ter analiziram metodologijo razvoja, ki jo oddelek uporablja.

V petem poglavju napravim primerjavo obeh oddelkov ter analizo smiselnosti vpeljave metodologije scrum v oddelek NAV. Na koncu podam priporočila oddelku NAV glede načina vpeljave metodologije scrum.

1 AGILNI RAZVOJ PROGRAMSKE OPREME

1.1 Metodologije razvoja programske opreme

Življenjski cikel razvoja programske opreme (angl. *software development lifecycle* ali SDLC) zajema programske procese, uporabljene za specifikacijo in transformacijo programskih zahtev v dostavljiv programski produkt. Neopredmetena in gnetljiva narava programske opreme dopušča široko paleto modelov življenjskega cikla razvoja programske opreme:

- linearne modele, v katerih si faze razvoja programske opreme sledijo v zaporedju s povratnimi informacijami in iteracijami po potrebi, čemur sledi integracija, testiranje in dostava celega produkta;
- iterativne modele, pri katerih je v ponavljajočih se ciklih programska oprema razvita v inkrementih razširitev funkcionalnosti;
- agilne modele, ki tipično vključujejo pogoste predstavitve delujoče programske opreme kupcu ali predstavniku uporabnikov, ki narekuje razvoj programske opreme v kratkih ciklih, ki proizvedejo majhen, za izdajo primeren inkrement delujoče programske opreme (Bourque & Feirley, 2014).

Inkrementalni, iterativni in agilni modeli lahko po želji dostavijo zgodnje različice delujoče programske opreme v uporabniško okolje. Linearnim SDLC modelom včasih pravimo

napovedni, iterativnim in agilnim SDLC modelom pa prilagodljivi modeli življenjskega cikla razvoja programske opreme (Bourque & Feirley, 2014).

Značilnost, po kateri se najbolj razlikujejo različni modeli SDLC, je način, na katerega upravljajo s programskimi zahtevami. Linearni razvojni modeli tipično v teku iniciacije projekta in faze načrtovanja razvijejo kar najbolj popoln nabor programskih zahtev, ki je potem striktno nadzorovan. Spremembe programskih zahtev temeljijo na zahtevkih za spremembo, ki jih odobri ali zavrne odbor za nadzor sprememb. Inkrementalni model proizvede zaporedne inkremente delujoče, dostavljive programske opreme z razdelitvijo programskih zahtev, ki naj bi se implementirale v vsakem od inkrementov. Tako kot v linearnem modelu so programske zahteve lahko striktno nadzorovane ali pa je dopuščena določena mera prožnosti za prilagoditev programskih zahtev med razvojem produkta. Agilni modeli sicer na začetku določijo okvir produkta in visoko nivojske funkcionalnosti, vendar so oblikovani tako, da omogočajo evolucijo programskih zahtev tekom projekta (Bourque & Feirley, 2014).

Potrebno je poudariti, da razpon SDLC-jev od linearnega do agilnega ni premočrten. V nek model so lahko vgrajeni elementi drugačnih pristopov. Inkrementalni SDLC model lahko na primer vključuje zaporedne faze zajema programskih zahtev in faze oblikovanja, vendar dovoljuje dobršno mero prožnosti pri spremembah programskih zahtev in arhitekture v procesu razvoja programske opreme (Bourque & Feirley, 2014).

1.1.1 Slapovni model SDLC

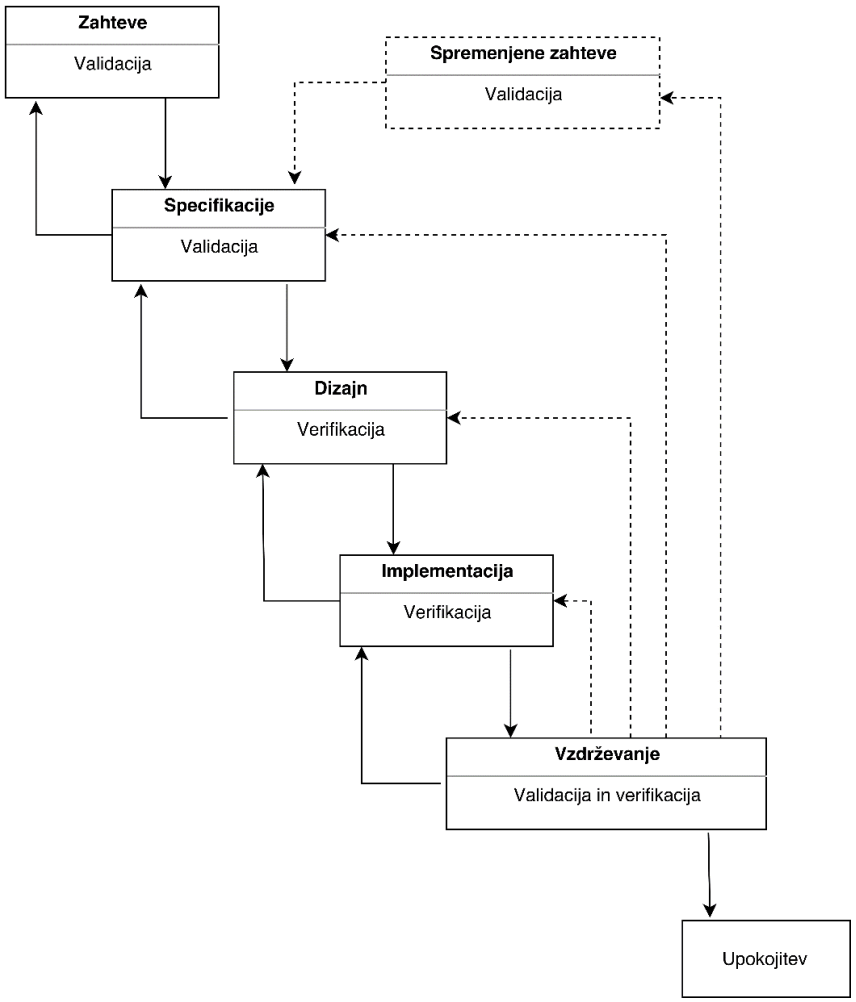
Slapovni (angl. *waterfall*) model je prvotno definiral Royce okoli leta 1970. Ta model ima prednost, da življenjski cikel razvoja programske opreme razdeli na faze, lahko razumljive vodstvu. Prvotno je bil razčlenjen na naslednje faze:

- zajem zahtev (zberejo se potrebe naročnika),
- specifikacija (pripravi se formalni dokument zahtev),
- načrtovanje (določi se module programa, algoritme in podatkovne strukture),
- implementacija (algoritmi in ostali elementi načrta se pretvorijo v kodo),
- testiranje enot (testira se pravilnost posameznih modulov),
- integracijsko testiranje (moduli se združijo in testira se, če je zadoščeno zahtevam),
- vzdrževanje po uvedbi (izvajajo se popravki ali dodelave kode) in
- upokojitev (produkta ni več moč vzdrževati oz. je zastarel, zato se ga umakne iz uporabe) (Sherrell, 2013).

Ta model se je izkazal kot neprilagodljiv, saj slabo prenese spremembe v zahtevah naročnika ali napake, odkrite pozno v življenjskem ciklu. Zato bolj sodobna različica slapovnega modela vsebuje povratne zanke in pogosto ne vsebuje testnih faz. S tem je poudarjeno, da se

mora teoretično (logični pregled) in/ali praktično (izvajanje implementirane kode) testiranje izvajati med vsako fazo SDLC. Na sliki 1 je predstavljen sodoben slapovni model s povratnimi zankami.

Slika 1: Sodoben slapovni model



Povzeto in prirejeno po L. Sherrell, Encyclopedia of Sciences and Religions, 2013, str. 2181.

Kljub temu, da ima slapovni model precej pomanjkljivosti, je še vedno precej uporabljan. Razlog za to je verjetno priljubljenost modela pri vodstvu, ker od razvijalcev že zgodaj v življenjskem ciklu zahteva pripravo projektnega načrta z mejniki in dostavami (Sherrell, 2013).

SDLC modelom pravimo tudi metodologije razvoja programske opreme. Bajec in Krisper (2003) metodologijo definirata tako: "Metodologija zajema vse, kar redno počnemo, da bi dosegli želen rezultat, torej izdelek ali storitev, ki je cilj našega dela. V primeru razvoja programske opreme to ne pomeni zgolj postopkov, ki so neposredno povezani z razvojem (npr. analiza, načrtovanje itd.), temveč zajema tudi podporne postopke, načine komunikacije

med sodelujočimi, pravila odločanja itd. V tem oziru lahko metodologijo opredelimo tudi kot množico dogovorov (konvencij), s katerimi se projektna skupina/organizacija strinja."

Metodologije imajo torej močno sociološko komponento. Ko organizacija privzame metodologijo, si jo vedno prilagodi po svoji meri, tako da ustreza njenemu načinu dela ter percepciji domene, za katere je vzpostavljena. Metodologija je sestavljena iz formalnih in neformalnih elementov. Formalni elementi so običajno zapisani v obliki postopkov, pravil, napotkov, smernic, standardov itd. Med neformalnimi elementi pa so ključnega pomena znanje in izkušnje, ki jih člani organizacije uporabljajo pri svojem delu. Predstavljajo tiste vrednote, ki so organizaciji lastne in so njena konkurenčna vrednost. Znanje se deli na eksplicitno in skrito znanje. Eksplicitno znanje je moč formalizirati, skrito znanje pa je bolj subjektivne narave in temelji na izkušnjah, idealih, čustvih, intuiciji, itd. Ko postane skrito znanje, ki ga uporabniki uporabljajo pri svojem delu, dovolj rutinsko, postane izrazljivo in lahko postane del formalne metodologije. Tako se metodologija bogati na osnovi skritega znanja posameznikov (Bajec & Krisper, 2003).

Postavlja se torej vprašanje, kje postaviti mejo med formalnim in neformalnim delom metodologije. Zgodnje metodologije, ki so nastajale od začetka 70-ih let, so natančno predpisovale postopke, metode, tehnike in orodja, s katerimi so želele standardizirati postopke in izdelke. Take metodologije, ki jih danes označujemo kot tradicionalne (ali tudi bogate), so rigidne in plansko usmerjene. Naslavljajo celoten življenjski cikel projekta, na primer z zagotavljanjem obsežnih smernic, standardiziranih procedur, predlog za planiranje projekta in vmesnikov do nadaljnjih organizacijskih procesov (Theocharis et al., 2015).

Bajec in Krisper (2003) ugotavljata, da sledenje smernicam kompleksnih metodologij za uporabnike predstavlja veliko breme, saj morajo poleg svojega dela skrbeti še za različne dokumente in postopke, ki jih predpisuje metodologija. Poleg tega je kompleksno metodologijo težko vzdrževati in spreminjati z na novo formaliziranim skritim znanjem. Kljub temu pa je dolgo veljalo, da je le s takim pristopom možno razvijati učinkovite in obvladljive sisteme.

V sredini 90-ih let pa so se začele pojavljati vitke, agilne metodologije, katerih vodilo je zmanjšati proces na minimum, v izogib »birokraciji« in da bi uporabnikom zagotovile le toliko pravil in smernic, kot jih je potrebno za izvedbo projekta (Theocharis et al., 2015).

1.2 Agilne metodologije

Sprememba miselnosti v razvoju programske opreme je sledila spremembam v proizvodni industriji, kjer je Toyota s svojim konceptom vitke proizvodnje revolucionirala avtomobilsko industrijo (Womack, Jones, & Roos, 1990).

Medtem ko je večina industrije razvoja programske opreme še vedno uporabljala tradicionalne, »težke« metodologije, so se pojavljali posamezniki, ki so začeli načela vitkosti vpeljevati v svoje metodologije, s katerimi so reševali projekte v težavah in razvijali nove produkte (Williams, 2012).

Leta 1995 sta Sutherland in Schwaber (1995; 2010) na delavnici na konferenci OOPSLA formalizirala dve leti Sutherlandovih praktičnih izkušenj razvoja programskih produktov z uporabo scruma (Sutherland, 2004) v metodologijo »scrum«. Pod vplivom scruma je leta 1999 nastala metodologija »eXtreme Programming« (Beck, 2000), ki je bila prva metodologija, ki je postala širše znana. Izkazalo se je, da po svetu kar nekaj razvojnih organizacij uporablja »lažje« pristope, ki so se po zgledu scruma in eXtreme Programminga formalizirali v metodologije razvoja.

Leta 2001 so se predstavniki vodilnih agilnih metodologij (Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming) in drugi simpatizerji potrebe po alternativi dokumentno-gnanemu »težkemu« procesu razvoja programske opreme sestali na konferenci v Chicagu in spisali Manifest agilnega razvoja programske opreme (angl. *Agile Manifesto*). V tem manifestu so opredelili osnovne vrednote agilnega razvoja:

- Posamezniki in interakcije pred procesi in orodji.
- Delujoča programska oprema pred vseobsežno dokumentacijo.
- Sodelovanje s stranko pred pogodbenimi pogajanji.
- Odziv na spremembe pred togim sledenjem načrtom (Agile Alliance, 2001).

Podpisniki manifesta sicer cenijo dejavnike na desni, ampak vseeno cenijo bolj tiste na levi.

Zgornje vrednote se zrcalijo v dvanajstih principih, ki jim sledijo njihove metodologije:

- Naša najvišja prioriteta je zadovoljiti stranko z zgodnjim in nepretrganim izdajanjem vredne programske opreme.
- Sprejemamo spremembe zahtev, celo v poznih fazah razvoja. Agilni procesi vprežejo tovrstne spremembe v prid konkurenčnosti naše stranke.
- Delujočo programsko opremo izdajamo pogosto, znotraj obdobja nekaj tednov do nekaj mesecev, s preferenco po krajšem časovnem okvirju.
- Poslovneži in razvijalci morajo skozi celoten projekt dnevno sodelovati.
- Projekte gradimo okrog motiviranih posameznikov. Omogočimo jim delovno okolje, nudimo podporo in jim zaupamo, da bodo svoje delo opravili.
- Najboljša in najučinkovitejša metoda posredovanja informacij razvojni ekipi in znotraj ekipe same je pogovor iz oči v oči.
- Delujoča programska oprema je primarno merilo napredka.

- Agilni procesi promovirajo trajnostni razvoj. Sponzorji, razvijalci in uporabniki morajo biti zmožni konstantnega tempa za nedoločen čas.
- Nenehno težnje k tehnični odličnosti in k dobremu načrtovanju izboljša agilnost.
- Preprostost - umetnost zmanjševanja količine nepotrebnega dela - je bistvena.
- Najboljše arhitekture, zahteve in načrti izhajajo iz tistih ekip, ki so samoorganizirane.
- V rednih časovnih razdobjih ekipa išče načine, kako postati učinkovitejša ob rednem prilagajanju svojega delovanja (Agile Alliance, 2001).

Glavni guruji agilnih metodologij torej na prvo mesto postavljajo posameznike in odnose med njimi. Strokovni, sposobni in motivirani posamezniki, med katerimi poteka dobra komunikacija, so vredni več kot še tako dobro premišljen in natančno definiran proces, podprt z najsodobnejšimi orodji.

Poudarjajo tudi, da je delujoča programska oprema edino merilo napredka. Dokumentacija sicer je pomembna, vendar le dokler je je ravno prav in v ravno pravem trenutku. Veliko več kot še tako detajlna dokumentacija naročniku pomeni, da vidi delujočo funkcionalnost. Na podlagi manjših inkrementov naročnik dobi znanje o videzu in delovanju sistema ter tako odstopanje od lastnih potreb ugotovi dosti prej kot s prebiranjem in analizo dokumentacije. Naročnik lahko tako sproti preverja, ali gre razvoj produkta v pravi smeri, torej ali rešitev zahteva ustreza zahtevam, na podlagi delujočega prototipa pa tudi lažje preveri smiselnost samih zahtev in jih po potrebi prilagodi. Na ta način se zagotovi, da je rezultat razvoja programska rešitev, ki jo naročnik potrebuje, ne pa rešitev, ki jo je s pomanjkljivimi informacijami specificiral na začetku projekta.

Trąbka in Soja (2014) sta ugotovila, da tradicionalni slapovni pristop, pri katerem pridejo uporabniki prvič v stik s sistemom tik pred prehodom v produkcijo, navadno rezultira v dodatnih zahtevah, ki pa so velikokrat opuščene zaradi pomanjkanja virov. V najboljšem primeru naročnik uporablja sistem z vedenjem, da ne ustreza vsem njegovim potrebam, v najslabšem primeru pa lahko naročnik celo odstopi od projekta.

Sodelovanje z naročnikom produkta je bistvenega pomena, saj se le tako lahko zagotovi, da bo s končnim produktom zadovoljen. Sodelovanje naj poteka na dnevni ravni in na podlagi delujočih inkrementov produkta. Naročnik mora imeti pri razvoju aktivno vlogo kot vir informacij, potrjevalec razvitih funkcionalnosti in vir novih predlogov, ki bi dodali dodatno vrednost končnemu produktu.

Prav pozitivni odnos do dodatnih oz. spremenjenih zahtev je bistveni koncept agilnih metodologij. Dobro je imeti načrt, vendar mu ne smemo slepo slediti. Četudi pride do spremembe v zahtevah razmeroma pozno v razvoju, implementacija te spremembe pomeni pomembno vrednost (na primer konkurenčno prednost) za naročnika, zato je vsem v interesu spremembo realizirati.

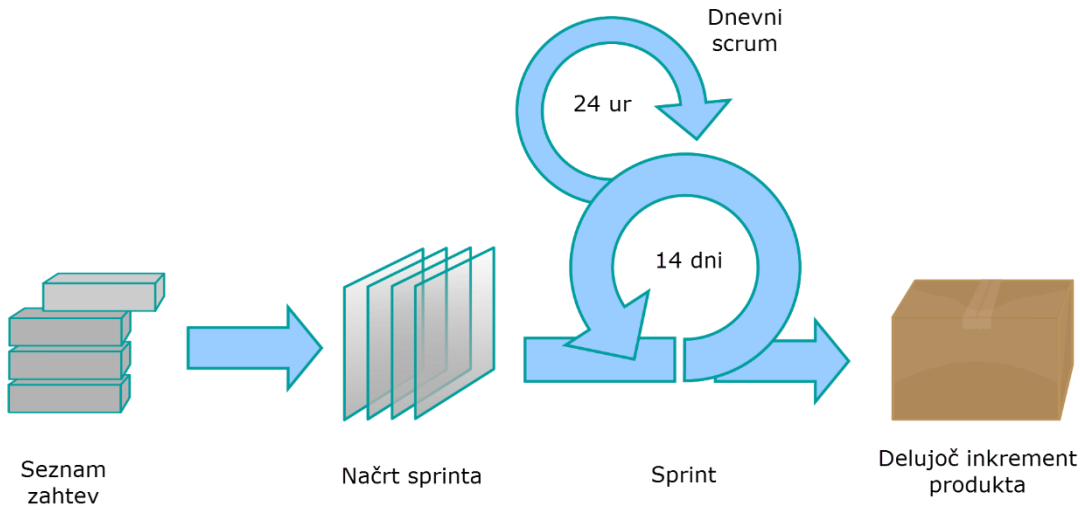
1.3 Scrum

Scrum je ena najbolj priljubljenih agilnih metodologij. Anketa, ki jo je leta 2010 izvedlo podjetje Forrester, kaže, da kar 38% razvijalcev, ki uporabljajo agilne metodologije, uporablja neko obliko scruma (West et al., 2011).

V uradnem priročniku scrum metodologije (Schwaber et al., 2013) je scrum definiran kot ogrodje, s katerim lahko ljudje pristopijo h kompleksnim adaptivnim problemom in produktivno ter kreativno dostavijo produkte najvišje možne vrednosti. Scrum ogrodje sestavljajo **scrum ekipe** in z njimi povezane **vloge**, **dogodki**, **izdelki** in **pravila**.

Na sliki 2 je prikazana poenostavljena shema scruma.

Slika 2: Enostavna shema scruma



Povzeto in prirejeno po C. Schmidt, Agile Software Development Teams, 2016, str. 17

Pri prevajanju angleških izrazov sem se naslonil na obstoječo literaturo v slovenskem jeziku (Plavše, 2015; Rojko, 2011), pri tem pa sem uporabil izraze, ki so se mi zdeli najbolj primerni. Izraz »dnevni scrum« sem namenoma pustil delno nepreveden, saj predstavlja izvor imena metodologije. Scrum je namreč način, kako pri ragbiju začneš igro – igralci se z glavami skupaj nagnetejo v kopico in poskušajo pridobiti posest žoge. To je dobra ponazoritev sestanka, kjer se vsa ekipa zbere in poskuša skupaj ugotoviti status nalog in sprinta.

1.3.1 Teorija scruma

Scrum temelji na teoriji empiričnega nadzora procesov ali empirizmu. Empirizem trdi, da znanje pride iz izkušenj in odločanja na podlagi znanih dejstev. Scrum uporablja iterativen,

inkrementalen pristop za optimizacijo predvidljivosti in nadzor tveganja. Vsako implementacijo empiričnega nadzora procesov podpirajo trije stebri: preglednost, nadzor in prilagoditev.

Preglednost je zagotovljena s skupnim standardom, tako da vsi opazovalci delijo skupno razumevanje, kaj vidijo. Tako morajo na primer vsi udeleženci v povezavi s procesom uporabljati skupen jezik; vsi izvajalci dela in vsi prejemniki produkta dela morajo deliti skupno definicijo pojma »zaključeno«.

Uporabniki scruma morajo redno nadzirati scrum izdelke in napredek v primerjavi s ciljem sprinta, da bi zaznali neželena odstopanja. Če se pri nadzoru opazijo odstopanja, je potrebno proces čim prej prilagoditi, tako da se minimizira nadaljnja deviacija.

Za nadzor in prilagoditev scrum predpisuje štiri formalne dogodke, ki so opisani v poglavju 1.3.3: načrtovanje sprinta, dnevni scrum, pregled sprinta in retrospektivo sprinta.

1.3.2 Scrum ekipa

Scrum ekipo sestavljajo produktni vodja, razvojna ekipa in vodja scruma. Scrum ekipe so samoorganizirane in večfunkcijske. Samoorganizirane ekipe same izberejo najboljši način za izpolnitev svojega dela, namesto da so nadzorovane s strani drugih zunaj ekipe. Večfunkcijske ekipe imajo vse kompetence, ki so potrebne za izpolnitev dela, brez zanašanja na druge, ki niso člani ekipe.

Produktni vodja (angl. *product owner*) je odgovoren za maksimiranje pomena produkta in dela razvojne ekipe. Je edina oseba, odgovorna za upravljanje seznama zahtev, kar obsega:

- jasno izražanje postavk seznama zahtev;
- razvrščanje postavk seznama zahtev za najboljše doseganje ciljev in misij;
- optimiziranje pomena dela, ki ga opravi razvojna ekipa;
- zagotavljanje, da je seznam zahtev viden, pregleden in jasen vsem ter da prikazuje, kaj so naslednje naloge scrum ekipe; in
- zagotavljanje, da razvojna ekipa razume postavke seznama zahtev do potrebnega nivoja.

Da bi bil produktni vodja uspešen, mora celotna organizacija spoštovati njegove odločitve. Te odločitve so vidne v vsebini in urejenosti seznama zahtev. Kdorkoli želi spremembo prioritete postavke seznama zahtev, mora svojo željo nasloviti na produktnega vodjo. Nikomur ni dovoljeno naročiti razvojni ekipi, naj dela na podlagi drugega seta zahtev, razvojni ekipi pa ni dovoljeno delovati na podlagi tega, kar naroči nekdo drug.

Razvojna ekipa (angl. *development team*) je sestavljena iz strokovnjakov, ki na koncu vsakega sprinta dostavijo zaključen inkrement produkta, ki je potencialen kandidat za končno, produkcijsko različico. Razvojna ekipa se sama odloči, kako bo postavke seznama zahtev pretvorila v inkremente funkcionalnosti. Pomembna je tudi skupna odgovornost - kljub temu, da imajo lahko posamezni člani razvojne ekipe specializirane veščine in fokusna področja, odgovornost pripada razvojni ekipi kot celoti.

Optimalna ekipa je dovolj majhna, da je še okretna, ter dovolj velika za dokončanje zadostne količine dela v enem sprintu. Če ekipa vsebuje manj kot tri člane, zmanjšanje interakcije rezultira v manjših prirastkih produktivnosti. Manjše ekipe se lahko srečajo tudi z omejenostjo veščin in posledično nezmožnostjo dostave inkrementa. Ekipa z več kot devetimi člani pa zahteva preveč koordinacije in generira preveč kompleksnosti, da bi jo lahko upravljal z empiričnim procesom.

Vodja scruma (angl. *scrum master*) je odgovoren za razumevanje in izvajanje scruma. Po eni strani skrbi, da scrum ekipa spoštuje teorijo, prakso in pravila scruma, po drugi strani pa osebam zunaj scrum ekipe pomaga razumeti, katere od njihovih interakcij s scrum ekipo pomagajo in katere škodijo. Škodljive interakcije pomaga spremeniti in s tem omogočiti večjo produktivnost ekipe.

Delo vodje scruma je služiti produktnemu vodji, razvojni ekipi in organizaciji. Organizaciji v vlogi guruja pomaga pri uvedbi scruma ter planiranju scrum implementacij. Zaposlenim in interesnim skupinam pomaga razumeti in dosledno izvajati scrum ter empirični razvoj produkta, v scrum procesih pa tudi aktivno nastopa kot organizator dogodkov (na zahtevo in po potrebi), mediator in katalizator sprememb, ki povečajo produktivnost scrum ekipe. Produktnemu vodji pomaga pri iskanju tehnik za učinkovito vodenje in urejenost seznama zahtev. K urejenosti pripomore tudi z izobraževanjem scrum ekipe o pomembnosti čistosti in jedrnatosti postavk seznama zahtev. Produktnemu vodji pomaga pri razumevanju produktnega planiranja v empiričnem okolju ter pri razumevanju in prakticiranju agilnosti. Razvojni ekipi vodja scruma pomaga s treningom samoorganizacije in večfunkcijskosti ter pri odstranjevanju ovir za napredek. V primeru več scrumov znotraj organizacije sodeluje z ostalimi vodjami scruma za povečanje učinkovitosti scruma v organizaciji.

1.3.3 Scrum dogodki

Scrum predpisuje dogodke, s katerimi ustvari rutino in minimizira potrebo po sestankih, ki niso definirani znotraj scruma. Vsi dogodki so navzgor časovno omejeni, sprint pa je omejen tudi navzdol - ko se enkrat začne, je njegovo trajanje fiksno in ne more biti skrajšan ali podaljšan. Razen sprinta, ki je vsebnik za ostale dogodke, vsak dogodek v scrumu predstavlja formalno priložnost za nadzor in prilagoditev procesa.

Srce scruma je **sprint** – enomesečno ali krajše časovno okno, znotraj katerega se ustvari inkrement produkta. Vsak sprint lahko razumemo kot projekt z največ enomesečnim trajanjem. Vsak sprint ima definirane naloge in cilj. Med sprintom lahko produktni vodja na podlagi novega znanja vanj dodaja ali iz njega umika naloge, ne sme pa spreminjati ciljev glede kakovosti narejenega.

Omejitev trajanja sprinta na največ en mesec služi omejitvi tveganja. Če je konec sprinta predaleč, se lahko med razvojem spremenijo zahteve. Sprinti zagotovijo nadzor in prilagoditev najmanj enkrat mesečno, s tem pa tudi omejijo tveganje na en koledarski mesec stroškov. Če se okoliščine toliko spremenijo, da cilj sprinta postane zastarel, lahko produktni vodja sprint tudi predčasno prekliče. Do tega lahko pride, če podjetje spremeni smer ali če se stanje trga ali tehnologije spremeni.

Na sestanku **načrtovanje sprinta** (angleško *sprint planning*) se načrtuje delo, ki naj bi se opravilo v sprintu. Pri načrtovanju sodeluje celotna scrum ekipa, lahko pa so na načrtovanje sprinta povabljene tudi druge osebe, da nudijo tehnične ali domenske nasvete.

Razvojna ekipa na podlagi seznama zahtev, zadnjega inkrementa produkta, projicirane kapacitete razvojne ekipe v sprintu in pretekle uspešnosti razvojne ekipe poskuša napovedati funkcionalnosti, ki bodo razvite med sprintom. Število postavk seznama zahtev, ki bodo vključene v sprint, je odvisna izključno od razvojne ekipe, saj lahko samo razvojna ekipa oceni, kaj lahko opravi v prihajajočem sprintu.

Ko razvojna ekipa napove, katere postavke seznama zahtev bo dostavila v sprintu, scrum ekipa določi cilj sprinta. Cilj sprinta je cilj, ki bo dosežen znotraj sprinta z implementacijo postavk s seznama zahtev. Na koncu ekipa določi, kako bo v sprintu te funkcionalnosti zgradila v zaključen inkrement produkta. Postavke seznama zahtev, izbrane za ta sprint, ter načrt za implementacijo skupaj sestavljajo **načrt sprinta**.

Dnevni scrum (angleško *daily scrum*) je 15-minutni sestanek, ki je namenjen uskladitvi aktivnosti razvojne ekipe in določitvi načrta za naslednjih 24 ur. Med sestankom vsak član razvojne ekipe razloži:

- kaj je naredil včeraj,
- kaj bo naredil danes in
- ali vidi kako oviro, ki bi njemu ali razvojni ekipi preprečila dosego cilja sprinta.

Dnevni scrum izboljša komunikacijo, odpravi potrebo po drugih sestankih, pomaga prepoznati morebitne ovire pri razvoju, ki jih je potrebno odstraniti, poudarja in spodbuja hitro odločanje ter izboljšuje nivo znanja v razvojni ekipi. Je ključni sestanek za nadzor in prilagoditev.

Pregled sprinta (angleško *sprint review*) je dogodek ob koncu sprinta, na katerem se preveri inkrement in po potrebi prilagodi seznam zahtev. Namenjen je izzivanju povratne informacije in spodbujanju sodelovanja. Med pregledom sprinta scrum ekipa in ključni predstavniki interesnih skupin preučijo, kaj je bilo narejeno v sprintu. Produktni vodja razloži, katere postavke seznama zahtev so bile zaključene in katere ne. Razvojna ekipa predstavi delo, narejeno v sprintu, ter razloži, kaj je med sprintom šlo dobro, na katere probleme so naleteli in kako so bili ti problemi rešeni. Celotna skupina nato sodeluje pri razpravi glede naslednjih korakov, tako da pregled sprinta predstavlja dragocen prispevek k naslednjemu načrtovanju sprinta.

Retrospektiva sprinta (angleško *sprint retrospective)* je sestanek scrum ekipe, ki se zgodi po pregledu sprinta in pred naslednjim načrtovanjem sprinta. Predstavlja priložnost, da se scrum ekipa oceni in napravi načrt za izboljšave, ki bodo udejanjene v naslednjem sprintu. Namen retrospektive sprinta je:

- preučiti, kaj se je v zadnjem sprintu dogajalo z ozirom na ljudi, odnose, proces in orodja;
- identificirati in urediti glavne stvari, ki so šle dobro, in potencialne izboljšave; in
- napraviti načrt za implementacijo izboljšav v način delovanja scrum ekipe.

Pri retrospektivi sprinta ima pomembno vlogo vodja scruma, saj spodbuja scrum ekipo k izboljšanju znotraj ogrodja scrum procesa, svojih razvojnih procesov in praks, da bi bili bolj učinkoviti in prijetni za naslednji sprint. Čeprav so lahko izboljšave implementirane kadarkoli, predstavlja retrospektiva sprinta formalno priložnost za osredotočanje na nadzor in prilagoditev.

1.3.4 Scrum izdelki

Izdelki, ki jih definira scrum, so posebej oblikovani za kar največjo preglednost osnovnih informacij, tako da ima vsakdo enako razumevanje izdelka. S tem zagotavljajo preglednost in priložnost za nadzor in prilagoditev. Odločitve za optimizacijo vrednosti in kontrolo tveganja so narejene na podlagi zaznanega stanja izdelkov. Če je preglednost popolna, imajo te odločitve trdno podlago. V primeru nepreglednosti izdelkov so lahko odločitve napačne in tveganje se lahko poveča.

Seznam zahtev (angl. *product backlog*) popisuje vse funkcionalnosti, funkcije, zahteve, dodelave in popravke, ki sestavljajo predvidene spremembe na produktu v prihodnjih izdajah. Je edini vir zahtev za kakršnokoli spremembo na produktu. Za vsebino, razpoložljivost in urejenost seznama zahtev je odgovoren produktni vodja.

Seznam zahtev ni nikoli zaključen. Njegov zgodnji razvoj zgolj določi prvotno znane in najbolje razumljene zahteve, nato pa se razvija skupaj z razvojem produkta in okolja, v katerem bo produkt uporabljan. Seznam zahtev je namreč dinamičen - neprestano se

spreminja, da bi identificiral, kaj produkt potrebuje, da bi bil primeren, konkurenčen in uporaben. Dokler obstaja produkt, obstaja tudi njegov seznam zahtev.

Postavke seznama zahtev imajo naslednje atribute: opis, vrstni red, oceno in pomen. Izpopolnjevanje seznama zahtev je stalen proces, v katerem produktni vodja in razvojna ekipa postavkam seznama zahtev dodajajo podrobnosti in ocene, jih pregledujejo in posodabljajo ter jim določajo vrstni red. Višje uvrščene postavke seznama zahtev so navadno bolj jasne in podrobne kot nižje uvrščene, zato so tudi bolj natančno ocenjene. Postavke seznama zahtev, na katerih bo razvojna ekipa delala v naslednjem sprintu, se izpopolnijo, tako da se lahko katerakoli od njih razumno zaključi znotraj enega sprinta. Tako izpopolnjene postavke se razumejo kot pripravljene za izbiro v načrtovanju sprinta.

Razvojna ekipa je odgovorna za vse ocene. Produktni vodja lahko sicer vpliva na razvojno ekipo s pomočjo pri razumevanju in izbiri kompromisov, vendar končno oceno podajo ljudje, ki bodo izvedli delo.

V kateremkoli trenutku se lahko sešteje preostalo delo do dosega cilja. Produktni vodja spremlja ta seštevek najmanj ob vsakem pregledu sprinta. Ta seštevek primerja s seštevkom ob prejšnjih pregledih sprinta in s tem oceni napredek v primerjavi s končanjem predvidenega dela do želenega časa za dosego cilja. Ta informacija je na voljo vsem interesnim skupinam.

Načrt sprinta (angl. *Sprint backlog*) vsebuje nabor postavk seznama zahtev, izbranih za realizacijo v določenem sprintu, ter načrt za dostavo zaključenega inkrementa produkta. Načrt sprinta je izdelek razvojne ekipe in le oni ga lahko spreminjajo tekom sprinta - ko je delo izvedeno ali zaključeno, se posodobi ocena preostalega dela. Če se elementi načrta spoznajo za nepotrebne, se odstranijo, lahko pa le-ti med sprintom tudi nastanejo. To se zgodi ko razvojna ekipa med delovnim procesom izve več o delu, ki je potrebno za dosego cilja sprinta.

V kateremkoli trenutku se lahko sešteje preostalo delo v načrtu sprinta. Razvojna ekipa spremlja ta seštevek najmanj ob vsakem dnevnem sprintu in s tem oceni verjetnost za dosega cilja sprinta. S spremljanjem preostalega dela tekom sprinta lahko razvojna ekipa upravlja svoj napredek.

Inkrement je vsota vseh postavk seznama zahtev, zaključenih v sprintu, in vrednosti inkrementov vseh prejšnjih sprintov. Na koncu sprinta mora biti novi inkrement zaključen, kar pomeni, da mora biti v uporabnem stanju in zadoščati definiciji zaključenosti scrum ekipe. Ne glede na to, ali se produktni vodja odloči, da bo inkrement izdal ali ne, mora biti inkrement v uporabnem stanju. Vsak inkrement je nadgradnja vseh predhodnih inkrementov in je temeljito testiran, da se zagotovi skladnost vseh inkrementov.

Ko sta postavka seznama zahtev ali inkrement opisana kot zaključena, mora vsakdo razumeti, kaj »zaključeno« pomeni. Čeprav se to močno razlikuje od scrum ekipe do scrum ekipe, morajo imeti člani za zagotovitev transparentnosti skupno razumevanje, kaj pomeni, da je delo zaključeno. To je **definicija zaključenosti** (angl. *definition of done*) scrum ekipe in se uporabi za določitev, kdaj je delo na produktnem inkrementu končano. Ista definicija vodi razvojno ekipo, ko se odloča, koliko postavk seznama zahtev lahko vključi v sprint med načrtovanjem sprinta.

Pričakovano je, da se z zrelostjo scrum ekip njihove definicije zaključenosti razširijo z bolj strogimi kriteriji za višjo kvaliteto. Katerikoli produkt ali sistem bi moral imeti definicijo zaključenosti, ki je standard za kakršnokoli delo na njem.

1.4 Celovite informacijske rešitve

Van Der Hoeven (2009) pojasnjuje, da ima pojem **Enterprise Resource Planning** (**ERP**) dva pomena:

- ERP kot aplikacija je standardni programski paket z zelo integriranimi funkcionalnostmi, ki se raztezajo preko vseh poslovnih procesov v podjetju. Sistem za planiranje virov, ki ga poganja ta programska oprema, uporablja centralno bazo podatkov.
- ERP kot fenomen pa je koncept integriranega načrtovanja in vodenja vseh virov podjetja kot celote. Gre za interakcijo med vsemi poslovnimi procesi organizacije in tudi z okoljem organizacije, na primer s kupci, dobavitelji in poslovnimi partnerji.

Islovar (terminološki slovar informatike, ki ga vzdržuje Slovensko društvo Informatika), *Enterprise Resource Planning* kot fenomen prevaja kot načrtovanje virov podjetja (2016). Za pojem ERP kot aplikacija pa obstaja več prevodov. Terminološki slovar informatike Islovar to prevaja kot »celovita programska rešitev« (2016). Sam sem se odločil, da bom uporabljal izraz celovita informacijska rešitev. Ta izraz po mojem mnenju dobro poudarja, da ne gre zgolj za program, ampak za celoten informacijski sistem. V primeru okrajšave pa bom uporabljal kar kratico ERP, saj je le-ta v strokovni literaturi že prerasla meje kratice in se uporablja kot samostojni izraz. Prevod kratice tako ni smiseln.

1.4.1 Zgodovina ERP

ERP je nastal v 90-ih letih kot naslednik logističnih izrazov MRP in MRP II. MRP je okrajšava za *Material Requirements Planning* ali po slovensko planiranje materiala (Islovar, 2016). To je metodologija, ki na podlagi pričakovane prodaje kupcem v proizvodnem procesu načrtuje potrebo po materialu. MRP orodja tistega časa so omogočala vnos kosovnice proizvoda ter pripravo naročila dobavitelju na podlagi: želene količine

proizvedenih kosov proizvoda, zaloge, dobavnih rokov, minimalne količine za naročanje idr. (Van Der Hoeven, 2009).

MRP II je metodologija načrtovanja proizvodnje, ki seže dlje kot MRP I. Osnovna ideja je ta, da za proizvodnjo proizvoda ni potreben zgolj material, ampak tudi delovni stroji, s katerimi material spremenimo v proizvod, ter ljudje, ki upravljajo s temi stroji. Kratica MRP tukaj pomeni Manufacturing Resources Planning in je v slovenščino prevedena kot planiranje proizvodnih virov. Zaradi potrebe po upoštevanju veliko več parametrov so bila MRP II orodja veliko kompleksnejša kot MRP I orodja (Van Der Hoeven, 2009).

V 80-ih je veliko organizacij za svoje delovanje uporabljalo več aplikacij - vsak oddelek je imel sebi namenjeno orodje, s katerim je opravljal svoje delo. Tako je imela na primer računovodska služba svoj računovodski sistem, skladišče pa svoj sistem za vodenje zalog. Ti sistemi med seboj niso bili povezani, zato je bilo potrebno izhodne podatke enega sistema ročno vnašati v drugega. To je seveda pomenilo dvojno delo, zaradi ročnega vnosa pa je prihajalo tudi do napak in zamud. Ker so dvojne knjižbe predstavljale probleme, se je pojavila potreba po integraciji vseh knjižb v enoten sistem, kjer bi različni moduli preprosto komunicirali med seboj. Poleg tega so podjetja želela, da bi sistem podpiral vse vire podjetja. Tako je nastal ERP ali celovita informacijska rešitev (Van Der Hoeven, 2009).

1.4.2 Lastnosti ERP sistemov

ERP sistemi so t.i. standardni programski paket, kar pomeni, da niso razviti za vsako podjetje posebej, ampak gre za standardno rešitev, ki jo implementator namesti in nastavi tako, da ustreza poslovnim procesom podjetja. ERP sistemi so bili spočetka namenjeni večjim podjetjem, sčasoma pa so bili prilagojeni tudi za srednja in majhna podjetja. Nekateri ponudniki sočasno razvijajo več produktov, vsakega namenjenega določeni velikosti podjetja, za posamezen produkt pa lahko obstaja tudi več panožnih rešitev - nadgradenj oz. modifikacij osnovnega produkta, namenjenih posameznim panogam - in dodatkov (angl. *add-on*) z določeno funkcionalnostjo. Vsak ERP sistem je možno programsko prilagoditi tudi za posamezno podjetje, pri čemer je potrebno dobro premisliti, na katerih mestih je prilagoditev sistema res potrebna, kje pa je smotrneje, da se prilagodijo procesi v podjetju. Če se poslovni procesi v podjetju bistveno razlikujejo od običajnih poslovnih procesov, potem je skoraj nujen razvoj sistema po meri.

1.4.3 Prednosti in slabosti ERP sistemov

ERP sistemi seveda niso vedno prava izbira za podjetje. Van Der Hoeven (2009) je identificiral več prednosti in slabosti ERP sistemov.

Prednosti:

- Integriranost - ERP brezšivno podpira koherenco med procesi in podatki v podatkovni bazi. Podatke je potrebno vnesti samo enkrat, da bi bilo dostopni kjerkoli v organizaciji.
- Procesna orientiranost - ključni deli organizacije so poslovni procesi in ne poslovne funkcije. Slabost procesnih funkcij je veliko prenosnikov, ki ovirajo poslovanje.
- Preprostost uporabe - ERP sistemi uporabljajo isti uporabniški vmesnik za vse module in s tem skrajšajo čas, potreben za učenje uporabe sistema.
- Odprtost - ERP sistemi zagotavljajo komunikacijske vmesnike za preprosto izmenjavo podatkov z drugimi sistemi.
- En dobavitelj - podjetju informacijsko podporo zagotavlja en sam dobavitelj, s čimer se olajša nakup in vzdrževanje sistema.
- Sodobnost - dobavitelji ERP sistemov so na vodilnem mestu pri razvoju novih rešitev in novih funkcionalnosti v svojem sistemu. So veliki igralci na trgu in v raziskave in razvoj vlagajo veliko denarja. Ko trg ali zakonodaja zahtevata novo funkcionalnost, bodo dobavitelji ERP sistemov takoj reagirali in zagotovili, da bo ta funkcionalnost na voljo v naslednji verziji sistema.
- Mednarodni vidik - prvi ERP sistemi na trgu so bili namenjeni multinacionalkam. S prilagoditvijo za srednja in majhna podjetja so ohranili večino mednarodnih vidikov, npr. podporo knjiženja v več valutah.

Slabosti:

- Dolgotrajne implementacije - v začetnih časih ERP sistemov ni bilo nič nenavadnega, če je projekt implementacije trajal več let. Od takrat so dobavitelji sistemov in svetovalna podjetja pridobili veliko izkušenj, tako da je danes implementacijo mogoče narediti v veliko krajšem času. Za trg srednje velikih in majhnih podjetij je večina dobaviteljev pripravila prednastavljene ERP sisteme, ki so skoraj pripravljeni za uporabo. S tem podjetje dobi kratko implementacijsko dobo, vendar je sistem manj prilagodljiv posebnim potrebam podjetja.
- Kompleksnost - ERP sistemi morajo integrirati celotno podjetje, zato so že po naravi kompleksni. Prepletenost funkcionalnosti in podatkov je težko osvojiti, zato morajo podjetja za implementacijo, nastavitev in prilagoditev ERP sistema najeti svetovalca. Kompleksnost botruje tudi neuspešnim implementacijam ERP projektov. Pogosto se svetovalci zaradi kompleksnosti ERP sistema usmerijo zgolj v posamezne panoge ali funkcijska področja, na primer v logistiko ali finance.
- Ni rešitev po meri - ERP sistem temelji na standardnem programskem paketu. Tak programski paket je seveda do neke mere prilagodljiv, da lažje pokrije potrebe podjetja, vendar nikoli ne bo po meri podjetja. Po drugi strani je razvoj posebnega, podjetju prilagojenega sistema drag in ne jamči dolgoživosti sistema. Vprašljivo je namreč, ali bosta oblikovalec in razvijalec še na voljo, ko se pojavi potreba po spremembah, recimo

ob spremembi zakonodaje. Obstaja tudi možnost razvoja sistema po meri kot dodatka standardnemu ERP programskem paketu. Veliko podjetij prvo implementacijo naredi na ta način, vendar kasneje opusti to idejo zaradi visokih stroškov nadgradnje ERP sistema na novo verzijo. Le-ta navadno vodi v novo aplikacijo po meri.

1.4.4 Implementacija ERP sistema

Implementacija ERP sistema vključuje vse aktivnosti, ki so potrebne, da se sistem vzpostavi in da se uporabniki izobrazijo za pravilno uporabo le-tega. Implementacijo večinoma izvajajo specializirani svetovalci, ki tesno sodelujejo s ključnimi uporabniki podjetja.

Nekatere od aktivnosti implementacije (Ray, 2011) so:

- modeliranje obstoječih procesov podjetja,
- prenova poslovnih procesov,
- nastavitev parametrov delovanja,
- razvoj dodelav,
- migracija podatkov,
- nastavitev uporabniških vlog in pravic,
- uporabniško testiranje,
- vpeljava v produkcijo.

Obstaja tudi več pristopov k uvedbi ERP sistemov. Ray (2011) glede na geografski in vsebinski obseg ločuje štiri pristope:

- pristop velikega poka (vse lokacije, vsi moduli),
- modularni pristop velikega poka (vse lokacije, izbrani moduli),
- zaporedni pristop (izbrane lokacije, vsi moduli) in
- modularni zaporedni pristop (izbrane lokacije, izbrani moduli).

Khanna in Arneja (2012) navajata, da obstaja veliko pristopov k uvedbi ERP sistema, vendar je večina teh pristopov različic petih osnovnih tipov:

- **Pristop velikega poka** (angl. *big bang*) predvideva vpeljavo vseh modulov ERP sistema v celotno organizacijo naenkrat. Če je pravilno in temeljito izveden, tak pristop prinaša prihranke zaradi odsotnosti potrebe po integraciji s podedovanimi sistemi. Slaba stran tega pristopa je ogromna potreba po virih za podporo ob prehodu sistema v produkcijo.
- **Fazni pristop** uvede module enega za drugim. V vsako organizacijsko enoto se namestijo samostojne instance ERP sistema, integracija med njimi pa se izvede kasneje v projektu. Do uvedbe vseh modulov za integracijo s podedovanimi sistemi skrbijo integracijski programi. Fazni pristop zmanjša tveganje z zmanjšanjem obsega uvedbe.

- **Vzporedni pristop** določeno obdobje po vpeljavi novega ERP sistema ohrani aktiven tudi podedovan sistem. Deli poslovnih aktivnosti hkrati potekajo tako v novem kot v starem sistemu. Tak pristop se uporablja na kritičnih projektih, kjer si naročnik ne more privoščiti nedelovanja sistema.
- **Pristop procesnih linij** postopno uvaja podobne poslovne procesne tokove ali proizvodne linije. Najprej prehod na nov ERP sistem izvede prva procesna linija in z njo povezane službe. Po uspešni uvedbi na nov sistem preide naslednja procesna linija. Začetni uspeh dvigne zaupanje organizacije v sistem in poveča verjetnost uspešne uvedbe.
- **Hibriden pristop** je kakršnakoli kombinacija prejšnjih pristopov. Konkreten pristop določijo člani implementacijske ekipe na podlagi preučitve informacij. Manjše in preprostejše ERP implementacije imajo navadno enostavnejši hibridni pristop kot organizacije z veliko različnimi organizacijskimi enotami. Tak pristop je pogosto uporabljen zaradi prožnosti in prilagodljivosti konkretnim situacijam.

1.4.5 Prilagajanje ERP sistema

ERP sisteme zaradi svoje zasnove vedno potrebujejo prilagoditve, preden jih lahko organizacija začne uporabljati. V kolikšni meri je potrebno ERP sistem prilagoditi, je odvisno od več faktorjev. Prvi faktor je stopnja prileganja funkcionalnosti ERP sistema in poslovnih procesov organizacije. Prve izdaje ERP sistemov niso bile prilagojene različnim panogam, ampak so bile razvite za generične, navadno proizvodne organizacije. To je pogosto botrovalo nizki stopnji prileganja in posledično je bilo ob implementaciji potrebno veliko prilagoditev. Danes za večino ERP sistemov obstajajo različne panožne rešitve in dodatki, vendar je v nekaterih primerih stopnja prileganja še vedno nizka. V takih primerih nastopi drugi faktor – pripravljenost organizacije na prilagoditev svojih procesov ERP sistemu (kadar je to mogoče). Poslovni procesi organizacije se razvijajo skozi čas, pri tem pa pridobivajo posebnosti, ki niso nujno potrebne ali učinkovite. Kljub temu jih organizacija včasih ni pripravljena opustiti (Brehm, Heinzl, & Markus, 2001).

Organizacija se mora torej v primeru odstopanja poslovnih procesov od funkcionalnosti ERP sistema odločiti za eno od treh možnosti:

- prilagoditev ERP sistema,
- prilagoditev procesov organizacije ali
- toleriranje neskladja (Brehm et al., 2001).

Naloga implementatorja je, da organizaciji pri tej odločitvi svetuje. Glavni argument proti prilagoditvi ERP sistema je težje vzdrževanje in nadgrajevanje. V tabeli 1 je prikazanih devet tipov prilagajanja ERP sistema, ki so jih identificirali Brehm et al. (2001), in vpliv vsakega od tipov prilagajanja na vzdrževanje in nadgrajevanje sistema.

Tabela 1: Tipi prilagoditev ERP sistema in njihov vpliv na vzdrževanje in nadgrajevanje

Tip prilagoditve	**Opis**	**Vpliv na vzdrževanje in nadgrajevanje**
Konfiguracija	nastavljanje parametrov, s katerimi se izbere drugačna pot skozi procese in funkcije sistema	ga ni / majhen
Panožne rešitve	paket dodelav ERP sistema, pripravljenih za zagotavljanje panožnih funkcionalnosti	odvisen od koordinacije med dobaviteljem ERP in dobaviteljem panožne rešitve
Zaslonske maske	priprava novih zaslonskih mask za vnos in prikaz podatkov	majhen / srednje velik
Razširjeno poročanje	priprava razširjenih izpisov podatkov in poročil	srednje velik / velik
Programiranje delovnih tokov	kreiranje nestandardnih delovnih tokov	srednje velik / velik
Uporabniški izhodni klici	programiranje dodatne programske kode v odprtem vmesniku	velik
ERP programiranje	programiranje dodatnih aplikacij v integriranem razvojnem okolju brez sprememb izvorne kode	velik, če so uporabljeni podatki iz ERP aplikacije
Razvoj programskih vmesnikov	programiranje vmesnikov do podedovanih sistemov ali produktov tretjih oseb	velik / zelo velik
Spreminjanje kode paketa	spreminjanje izvorne kode, od majhnih sprememb do sprememb celotnih modulov	zelo velik

Povzeto po L. Brehm et al., Tailoring ERP systems: a specrum of choices and their implications, 2001

Prilagajanje in razvoj dodelav za ERP sisteme zaradi vsebinske kompleksnosti zahtevata drugačen tip razvijalcev kot razvoj ostalih programskih rešitev.

1.5 Agilne metodologije in implementacije celovitih informacijskih rešitev

Kot sem omenil že v uvodu, agilne metodologije niso primerne za vsak projekt. Boehm in Turner (2003) sta identificirala pet dejavnikov, ki jih je potrebno upoštevati, ko se odločamo za metodologijo na projektu. V tabeli 2 so prikazane značilnosti slapovnih in agilnih metodologij glede na vsakega od petih dejavnikov.

Tabela 2: Značilnosti metodologij glede na dejavnike pri izbiri metodologije

Dejavnik	**Agilne metodologije...**	**Slapovne metodologije...**
Velikost	… so optimalne za majhne projekte in ekipe. Zanašajo se na skrito znanje.	… so prilagojene velikim projektom in ekipam.
Kritičnost cilja projekta	… na kritičnih projektih niso preverjene. Vlada splošno pomanjkanje dokumentacije.	… imajo dolgo zgodovino uporabe pri takih implementacijah.
Stabilnost in kompleksnost	… predvidevajo nenehne predelave. Primerne so za dinamične in preproste projekte.	… uporabljajo strukturirana izhodišča. Primerne so za bolj statične in kompleksne projekte.
Veščine	… predvidevajo nenehno vključenost visoko izurjenih posameznikov. Težko shajajo z večjim številom manj izurjenih posameznikov.	… potrebujejo visoko izurjene posameznike predvsem v začetnih fazah, v poznejših fazah pa shajajo z velikim številom manj izurjenih posameznikov.
Organizacijska kultura	… najbolje delujejo v kaotični, dinamični organizaciji, ki zagovarja opolnomočene posameznike.	… najbolje delujejo v organizaciji z dobro definiranimi vlogami in vzpostavljenimi postopki.

Povzeto in prirejeno po R. Hopkins in K. Jenkins, Eating the IT elephant, 2008

Na podlagi te metode za izbiro metodologije bi lahko s poznavanjem lastnosti projektov implementacije ERP sistemov ocenili, da je agilni pristop primeren le za preproste projekte, pri katerih naročnik potrebuje manjše število modulov, večinoma uporablja standardne procese (ni potrebe po obsežnih prilagoditvah) ter uporablja malo podedovanih sistemov, s katerimi se mora ERP integrirati. Čim pa je v uporabi več modulov, procesi niso standardni, je veliko integracij itd., kompleksnost projekta močno naraste, potrebna je večja implementacijska ekipa, večja pa je tudi kritičnost cilja projekta, saj ima lahko vsaka napaka obsežne posledice preko celotnega sistema. Prav tako je prehod v produkcijo obsežnejši, izšolati je potrebno več ljudi, za kar pa je nujna dobra dokumentacija. Po zgornjih merilih je torej za tak projekt primernejši bolj »discipliniran« slapovni pristop.

Nerur et al. (2010) pa prav agilne metodologije vidijo kot orodje za spopadanje s kompleksnimi projekti. Če so bili zgodnji začetki razvoja programske opreme usmerjeni k reševanju dobro definiranih problemov, danes razvojni projekti rešujejo izzive precej bolj

nestrukturiranih področij. Prav celovite informacijske rešitve so primer sistemov, ki se dotikajo veliko ali celo vseh vidikov poslovanja, tako znotraj kot tudi zunaj meja organizacije (npr. virtualne integracije, oskrbovalna veriga). Vsebinski kontekst je tako precej bolj strateški, kar zvišuje negotovost in kompleksnost razvoja. Prav tako se morajo pri takih projektih razvijalci usklajevati z večjim številom raznolikih interesnih skupin. Te niso nujno med seboj usklajene glede ciljev programske opreme, kar lahko vodi v podajanje konfliktnih informacij. Sistemi z največjo vrednostjo danes niso nujno tisti, ki so optimalni v tehnološkem smislu, pač pa tisti, ki so privlačni zaradi minimiziranja konflikta, ki izvira iz različnih želja in potreb interesnih skupin. Če pri implementaciji ERP-ja uporabimo tradicionalno slapovno metodologijo, se lahko zgodi, da spregledamo negativni vpliv, ki ga ima dodelava v enem modulu na drugi modul ali na integracijo. Zaradi inkrementalnega pristopa, stalne integracije ter sprotnega preverjanja rešitve je z agilnim pristopom usklajevanje med interesnimi skupinami lažje in posledično je produkt boljši.

Johansson in Carvalho (2009) ugotavljata, da je osnovna težava prilagajanja ERP sistemov razhajanje med zahtevami naročnika in dostavljeno funkcionalnostjo. Ta težava izhaja iz načina komunikacije med razvijalci in naročnikom, saj le-ta poteka v slapovnem stilu preko dokumentov in ne neposredno med ljudmi. Poleg tega ERP sistemi dostavijo ogromno funkcionalnosti, ki uporabnike preobremenijo z novimi informacijami. To uporabnikom otežuje razumevanje delovanja sistema in prepoznavanje že obstoječe funkcionalnosti, zato zahtevajo nepotrebno prilagoditev. Carvalho, Johansson in Manhães (2009; 2010) vidijo agilne metodologije s svojim načinom dostavljanja funkcionalnosti v majhnih inkrementih kot idealne za doseganje učinkovite komunikacije in hitrega odziva na spremembe. Če jih uspešno vpeljemo v proces prilagajanja ERP sistema, se bodo poslovni procesi in ERP sistem razvijali v tandemu.

Dejansko se agilni pristop uporablja na vse več projektih implementacije celovitih informacijskih rešitev. Trąbka in Soja (2014) sta v študiji dveh implementacij ERP sistemov primerjala tradicionalni in agilni pristop. Po njunem mnenju na izbiro pristopa najbolj vpliva temeljitost predimplementacijske analize. Če je v fazi analize pripravljeno podrobno oblikovanje rešitve, potem je smiseln tradicionalni slapovni pristop implementacije. Če pa je v fazi analize izvedena zgolj analiza želenih funkcionalnosti in želenega obnašanja sistema (angl. *business analysis*), ne pa tudi oblikovanje rešitve, naročnik ne pridobi znanja o dejanskem sistemu, npr. videzu uporabniškega vmesnika. Tradicionalni slapovni pristop, pri katerem pridejo uporabniki prvič v stik s sistemom tik pred prehodom v produkcijo, tako navadno rezultira v dodatnih zahtevah, ki pa so velikokrat opuščene zaradi pomanjkanja virov. V najboljšem primeru naročnik uporablja sistem z vedenjem, da ne ustreza vsem njegovim potrebam, v najslabšem primeru pa lahko naročnik celo odstopi od projekta. V splošnem lahko uporaba agilnega pristopa zmanjša tveganje za neuspeh projekta implementacije ERP sistema.

Mnoge agilne metodologije predvidevajo ali vsaj močno priporočajo stalno prisotnost predstavnika naročnika na lokaciji razvijalca, tako da lahko ves čas neposredno sodeluje s scrum ekipo. Coram in Bohner (2005) opozarjata, da je včasih težko najti stranko, ki je pripravljena na tolikšno vpletenost v razvojni proces. Zato je potrebno pri izbiri metodologije razvoja upoštevati tudi razpoložljivost predstavnikov naročnika.

Že Boehm in Turner (2003) sta napovedala nastanek hibridnih metodologij, ki uravnotežujejo agilnost in disciplino, in številne raziskave potrjujejo, da je dandanes redko katero podjetje res popolnoma agilno.

West et al. (2011) so ugotovili, da večina razvojnih organizacij, ki so usvojile koncepte agilnosti, v resnici uporablja hibridno metodologijo, ki jo avtorji poimenujejo »Water-Scrum-Fall«. Razlogi za to so delno v dejstvu, da uvedbo agilne metodologije pogosto vodijo njeni uporabniki, ki pa se osredotočajo predvsem na domene, na katere imajo vpliv, večinoma na samo ekipo. Področja izven njihovega vpliva, na primer poslovna analiza in upravljanje izdaj, še naprej sledijo bolj tradicionalnim pristopom, kar pomeni, da je agilnost večinoma vpeljana le na nivo razvojne ekipe. Na tradicionalni pristop pred začetkom razvoja imajo močan vpliv upravljavska pravila, ki v mnogih organizacijah pred začetkom dela zahtevajo definirane zahteve in načrte. Ti načrti pogosto predstavljajo osnovo za pogodbo med poslovnimi uporabniki in IT ekipo, ki določi usmeritev projekta, terminski načrt in proračun. Pogoste izdaje inkrementov po vsakem sprintu pa onemogoča predvsem dejstvo, da večina organizacij nima arhitekture, ki bi podpirala dinamične, prožne izdaje. Zato so izdaje novih verzij redke in povezane z dolgotrajnimi procesi preverjanja skladnosti.

Do podobnih ugotovitev so z ekstenzivno študijo literature prišli tudi Theocharis et al. (2015). »Water-Scrum-Fall« in podobne kombinacije so postale realnost. Glavno pobudo takih integracij vidijo v pričakovanjih, ki jih imajo različne interesne skupine do »optimalnega« pristopa k razvoju programske opreme. Projektni vodje potrebujejo določeno stabilnost za opravljanje običajnih nalog projektnega vodenja, kot so ocenjevanje, načrtovanje in nadzor. Projektni vodje imajo poleg tega tudi odgovornost za uskladitev projektov s strategijo podjetja, kar pomeni, da morajo biti projekti povezani z ostalimi poslovnimi procesi, kot so kadrovanje, prodaja, sklepanje pogodb itd. Ker agilne metodologije navadno naslavljajo le sistemske in razvojne naloge, takšne povezave pogosto manjkajo. Tako so tradicionalni pristopi uporabljeni za zagotovitev osnovne strukture in ogrodja za projektno organizacijo ter za zagotovitev vmesnikov do procesov podjetja. Po drugi strani razvijalci želijo pristop, ki nudi podporo razvojnim nalogam, obenem pa nudi svobodo pri izbiri najboljših praks v posamezni situaciji. To pomeni, da hibridna metodologija, v kateri so združeni agilni in tradicionalni pristopi, na videz nudi situacijo, najboljšo za vse udeležence.

2 SCRUM V PODJETU ADACTA D.O.O.

2.1 Predstavitev podjetja Adacta d.o.o.

Podjetje Adacta d.o.o., Ljubljana je član skupine Adacta Group, ki jo sestavlja šest podjetij (Adacta Ljubljana, Adacta Zagreb, Adacta Beograd, Adacta Brno, Adacta Moskva in Adacta Ciper), katerih osnovna dejavnost je razvoj, implementacija in vzdrževanje poslovnih informacijskih rešitev ter s tem povezano svetovanje.

Poslovanje Adacte lahko delimo na tri enakovredne segmente, ki pa so v različnem obsegu zastopani v posameznih hčerinskih podjetjih:

- lastna rešitev AdInsure, osnovna rešitev za zavarovalnice, ter rešitve za finančni sektor, s katerimi je Adacta prisotna na širšem področju CEE in v Rusiji;
- poslovno-informacijske rešitve Microsoft Dynamics NAV, AX in CRM;
- rešitve poslovne inteligence ponudnika Qlik.

Podjetje Adacta se je začelo z implementacijo celovitih informacijskih rešitev ukvarjati leta 1995. Leta 1997 je Adacta postala partner podjetja Microsoft in začela implementirati njegov ERP za mala in srednja podjetja Microsoft Navision (kasneje preimenovan v Microsoft Dynamics NAV). Leta 2011 je Adacta kot prvo podjetje v regiji začela implementirati ERP rešitve za srednja in velika podjetja Microsoft Axapta (kasneje preimenovan v Microsoft Dynamics AX).

Danes je Adacta s preko 370 zaposlenimi v šestih državah in na osmih poslovnih lokacijah (Ljubljana, Maribor, Zagreb, Skopje, Beograd, Brno, Moskva, Nicosia) eden največjih ponudnikov poslovno informacijskih rešitev in sistemov v regiji. Najštevilčnejši tim strokovnjakov, specializiranih za rešitve Dynamics, ki šteje preko 100 projektnih in strokovnih vodij, svetovalcev in programerjev (od tega 50 v Sloveniji), se je izkazal z uspešno implementacijo in vzdrževanjem preko 200 projektov uvedbe rešitev Dynamics kar je približno 40 odstotkov vseh implementacij v regiji (Adacta d.o.o., 2016).

2.2 Predstavitev Dynamics AX oddelka

Dynamics AX oddelek v Adacti se ukvarja z implementacijo in vzdrževanjem ERP rešitve Microsoft Dynamics AX. Oddelek je bil ustanovljen leta 2011, kmalu po uspešno zaključenem pilotskem projektu. Od takrat je bilo izpeljanih že 12 uspešnih implementacij, v času pisanja te naloge pa so v različnih fazah implementacije še trije projekti.

Oddelek AX v vseh podjetjih skupine Adacta zaposluje skupaj nekaj več kot 70 ljudi. V oddelek se je v začetku leta 2014 uvedel proces agilnih metodologij. Takrat se je izvedla tudi

večja reorganizacija in na novo so se postavile ekipe. Oddelek je razdeljen v osem ekip, sedem projektnih in eno ekipo za podporo strankam. V Ljubljani so locirane štiri ekipe, dve sta locirani v Zagrebu, ena pa v Beogradu.

2.2.1 Celovita informacijska rešitev Microsoft Dynamics AX

Celovita informacijska rešitev Microsoft Dynamics AX (v nadaljevanju AX) se je izvorno imenovala Axapta. Razvijalo jo je dansko podjetje Damgaard A/S. Prva različica rešitve, Axapta 1.0, je izšla na ameriškem in danskem trgu leta 1998. Prvotno je rešitev obsegala module za knjigovodstvo, upravljanje zalog, proizvodnjo, logistiko in prodajo. Razvoj se je nadaljeval z različicami Axapta 1.5, 2.0 in 2.1, nato pa se je podjetje Damgaard A/S leta 2000 združilo s podjetjem Navision Software A/S. Novo podjetje Navision-Damgaard je izdalo različico Axapta 2.5.

Podjetje Navision-Damgaard je bilo leta 2002 prevzeto s strani podjetja Microsoft. Produkt je bil sprva preimenovan v Microsoft Business Solutions Axapta, v procesu posodobitve blagovnih znamk Microsoftovih ERP produktov pa je dobil končno ime Microsoft Dynamics AX. Axapto 3.0 je tako julija 2006 nadomestil Microsoft Dynamics AX 4.0. Microsoft ni le širil ERP funkcionalnosti, temveč je tudi postopoma prilagajal rešitev videzu in občutku Microsoft Office produktov ter jo integriral z ostalimi rešitvami. Poleti 2008 je izšla različica Microsoft Dynamics AX 2009 z vmesnikom, temelječim na vlogah, z razširjenimi funkcionalnostmi in novimi moduli.

Trenutna različica Microsoft Dynamics 2012 je bila izdana avgusta 2011. Prinesla je razširitev uporabniškega vmesnika s številnimi funkcijami poslovnega obveščanja in poročanja, panožne razširitve, izboljšano integracijo z Office produkti, tesnejše povezave z ostalimi Microsoft rešitvami (SharePoint, Lync) ter drugo (Sycor Group, brez datuma).

Microsoft Dynamics AX 2012 rešitev je trinivojska aplikacija. Sestavljajo jo:

- Microsoft SQL podatkovna baza, v kateri so shranjeni podatki,
- servis Application Object Server ali AOS, ki izvaja vso poslovno logiko programa in
- klienti (samostojna aplikacija, spletna aplikacija, mobilna aplikacija, …) (The Microsoft Dynamics AX Team, 2014).

Dodelava in predelava AX kode se izvaja v integriranem razvojnem okolju MorphX, ki je dostopno neposredno iz klientske aplikacije, od verzije 2012 dalje pa je z uporabo vtičnika možen tudi razvoj v okolju Microsoft Visual Studio. Koda je napisana v lastnem programskem jeziku X++. Le-ta je objektno orientiran in temelji na razredih, omogoča pa tudi neposredne SQL poizvedbe (The Microsoft Dynamics AX Team, 2014).

2.3 Opis uporabljene metodologije

Za analizo oddelka in metodologije razvoja, ki jo uporabljajo pri svojem delu, sem opravil intervjuje s ključnimi osebami na oddelku. Pri izbiri intervjuvancev so me vodila temeljna vprašanja, na katera sem želel dobiti odgovore:

- Kako je scrum implementiran v različnih ekipah?
- Kako usmeritev ekipe vpliva na implementacijo scruma?
- Kako scrum vidijo osebe z različnimi scrum vlogami?
- Kako na scrum gleda vodstvo oddelka?

Ker sem želel pridobiti podrobne informacije o delovanju ekip, sem se raje kot za večje število krajših pogovorov z zaposlenimi odločil za poglobljene intervjuje z manjšim številom ključnih oseb na oddelku. Osebe sem izbral tako, da sem dobil odgovore na vsa zgornja vprašanja. Odločil sem se, da intervjuvam tri ključne osebe. Prvi intervju sem opravil s svetovalko Manco, ki v ekipi A nastopa v vlogi produktnega vodje. Drugi intervju sem opravil s Petrom, ki je v ekipi B skoraj dve leti opravljal naloge vodje scruma. Tretja oseba, ki sem jo intervjuval, je direktor AX oddelka, ki je bil tudi pobudnik vpeljave scruma v oddelek. Zaradi zagotavljanja anonimnosti so imena oseb spremenjena.

V podpoglavjih 2.3.1 in 2.3.2 bom najprej opisal delovanje obeh ekip. Nato bom v podpoglavju 2.3.3 opisal pogled direktorja področja na vpeljavo scruma v oddelek. V poglavju 3 bom opravil analizo uvedene metodologije in jo primerjal z načeli scruma in agilnih metodologij na splošno.

2.3.1 Scrum v ekipi A

O scrumu v ekipi A sem govoril z Manco, ki v ekipi nastopa v vlogi produktnega vodje. Produktni vodja ekipe A je po izobrazbi magistrica poslovno-informacijskih ved. Na področju ERP ima šestnajst let delovnih izkušenj. Deset let je delala v podjetju, ki je razvijalo svojo (manjšo) ERP rešitev. Tam je pokrivala vsa področja (finance, zaloge, plače itd.). Na Adacti je kot svetovalka zaposlena pet let, zadnje leto pa nastopa v vlogi produktnega vodje. Kot svetovalka načeloma pokriva področje financ, ker pa je narava dela taka, da finance pridejo na vrsto na koncu procesa, pozna in pokriva tudi področja nabave, zalog, prodaje, planiranja, kadrovanja in plač, torej praktično vse module, ki na koncu rezultirajo v finančnem modulu.

Ekipo A imenujejo tudi produktna ekipa, saj se v osnovi ukvarja z razvojem dveh produktov, Inita in Lokalizacije. Produkt Init je skupek dodelav, ki standardni paket AX razširjuje z naborom funkcionalnosti, ki so bile razvite v sklopu katere od prejšnjih implementacij, pa so zanimive za vse stranke. Nekatere funkcionalnosti so razvite tudi posebej za Init. Init je

verzija AX, ki jo ekipe vzamejo kot osnovo za implementacijske projekte. Produkt Lokalizacija je prav tako skupek dodelav standardnega paketa AX. Vsebuje funkcionalnosti, ki so potrebne za skladnost osnovne verzije AX z zakonodajo vseh podprtih držav. Trenutno Lokalizacija vsebuje podporo za zakonodaje Slovenije, Hrvaške ter Srbije in Črne gore. Ko produktna ekipa izda novo verzijo Inita ali Lokalizacije, ekipa za podporo strankam izvede nadgradnjo sistema pri vseh obstoječih strankah.

Adacta za vsako podprto verzijo AX vzdržuje kompatibilno verzijo Inita in Lokalizacije. Ko Microsoft izda novo različico AX, produktna ekipa začne projekta priprave različic Inita in Lokalizacije, ki bosta skladni s to novo različico AX-a. Pri tem se lahko nekatere funkcionalnosti v novo verzijo prenesejo iz prejšnje, druge pa morajo biti predelane. Zadnja taka projekta sta bila razvoj oz. nadgradnja Inita in Lokalizacije na različico AX 2012 R3.

Ekipa A se ukvarja tudi s pomočjo ekipi za podporo strankam, predvsem pri podpori strankam, pri katerih so implementacijo izvajali določeni člani ekipe pred vzpostavitvijo produktne ekipe.

Tretja naloga ekipe A pa je razvoj za druge ekipe oddelka. Trenutno se največ časa ukvarjajo z razvojem za projekt EDP (elektro-distribucijska podjetja).

Projekt EDP

Cilj tega projekta je dodelave, pripravljene v okviru implementacij AX v dve elektro-distribucijski podjetji, zbrati v produkt, ki bo osnova za nadaljnje implementacije v podjetja te panoge. Rešitvi ne moremo reči vertikala, saj ne gre za nov modul, pač pa za skupek dodelav preko vsega ERP sistema, ki podpirajo specifike elektro-distribucijskih podjetij.

Projekt EDP vodi neka druga ekipa AX oddelka, ki jo bom v tej analizi imenoval projektna ekipa. Projektna ekipa je prav tako scrum ekipa, ki šteje sedem članov. Od teh sta dva razvijalca, ki razvijata manjše, bolj preproste dodelave. Večina razvoja je predana v izdelavo ekipi A, tako da le-ta pravzaprav nastopa v vlogi podizvajalca.

Projekt EDP je pravzaprav sestavljen iz dveh projektov - projekta analize in projekta implementacije. Analiza je trajala tri mesece in jo je vodila projektna ekipa, pri njej pa je sodeloval tudi produktni vodja ekipe A. V analizo so bili vključeni predvsem svetovalci. Pri analizi so se zelo trudili opraviti svetovalni del, torej ne zgolj zapisovati strankinih zahtev, pač pa jim tudi svetovati in najti boljše rešitve. Začeli so s popisom zahtev, pri čemer so takoj ocenjevali, ali je zahteva že pokrita s standardno rešitvijo ali gre za dodelavo. Že v tej fazi so v sodelovanju s stranko poskusili prilagoditi procese, zato nastali dokument funkcionalnih zahtev ne popisuje obstoječega, ampak želeno stanje. Nato so naredili dizajn, pri čemer je bil nastali dokument funkcionalnega dizajna obsežen in dejansko v obliki navodil. Vsebina, ki jo je potrebno na projektu implementacije pokriti, je torej zelo dobro

definirana. Na podlagi dizajna se je ovrednotilo vrednost dodelav. Ta podatek so predstavili vodstvu naročnika. To je imelo na voljo določen znesek za implementacijo, vendar je bila vrednost vseh dodelav večja od tega zneska. Zato so se z vodstvom dogovorili, katere dodelave niso nujne oz. kje se lahko uporablja standardna funkcionalnost. S tem so določili pogodbeno vrednost projekta.

Sestava ekipe

V ekipi je deset članov - štiri svetovalke in šest razvijalcev. Posebnost ekipe je dejstvo, da trije člani ne živijo v Sloveniji, temveč v ekipi večinoma sodelujejo na daljavo, fizično pa so prisotni le občasno. Ena svetovalka in en razvijalec sta tako zaposlena v Adacti Zagreb, v Ljubljani pa sta prisotna enkrat na dva tedna. Ena svetovalka pa je zaposlena v Adacti Beograd in je v Ljubljani prisotna le izjemoma. Ta svetovalka trenutno ne opravlja nobenih nalog, pač pa nastopa bolj v vlogi kontakta, ki spremlja lokalno zakonodajo in opozarja na spremembe.

Ekipa obstaja manj kot eno leto. Je stalna, vendar nekateri člani ekipe del svojega časa delajo tudi na projektih drugih ekip. Kljub temu sodelujejo na vseh sestankih ekipe. Čas za projekte je planiran v sprintu z zmanjšano kapaciteto, včasih pa se aktivnosti za projekt dodajo kar v plan ekipe, s čimer se nadzira obremenjenost člana.

Ekipa je samozadostna, kar pomeni, da ima vsa potrebna znanja, ki jih potrebuje za opravljanje svojega dela. Pokrivajo vsa tehnična in vsebinska znanja, razen mogoče zelo specifičnih področij. V ekipi je približno polovica članov zelo izkušenih, polovica pa manj izkušenih. Večina razvijalcev dovolj dobro pozna vse module, da jih lahko suvereno dodelujejo. Nekateri pa so bolj ozko specializirani in večinoma opravljajo naloge s svojega področja. Tudi take pa se poskuša počasi vključevati tudi v druga področja. Podobno se obravnava tudi manj izkušene člane - začnejo s spoznavanjem enega področja, postopoma pa dobivajo naloge tudi z drugih področij. Pri tem se jim seveda pomaga in se jim nudi mentorstvo. Taka mentorstva in pomoč znotraj ekipe so v planu navadno upoštevana z zmanjšanjem kapacitete, ponekod pa se pri oceni naloge upošteva še določen čas za pomoč in mentorstvo.

Kapaciteta

Kot je že bilo omenjeno, člani ekipe pomagajo tudi pri nalogah ekipe za podporo strankam. Gre za stranke, ki rešitev uporabljajo v produkciji, zato imajo take naloge seveda prioriteto. Pri tem ne gre za pomoč pri vseh podpornih nalogah, saj ekipa za podporo strankam veliko problemov reši sama. Pogosto samo prosijo za kak nasvet ali pomoč. Takrat si član ekipe vzame recimo pol ure, da jim pomaga, potem pa nadaljujejo sami. Včasih pa se na ekipo A obrnejo s kompleksnejšim problemom, in sicer zato, ker je član ekipe delal na implementaciji projekta in funkcionalnost najbolje pozna. Take naloge lahko vzamejo tudi dan ali dva.

Take naloge niso redne in jih je mogoče načrtovati le okvirno z manjšo kapaciteto. Kapaciteta članov ekipe je povprečno štiri ure na dan, produktni vodja ima kapaciteto približno tri ure dnevno, nekateri razvijalci pet ur, en razvijalec pa veliko sodeluje pri podpori stranki po kompleksni implementaciji in ima zato nižjo kapaciteto.

Če produktni vodja ekipe A ugotovi, da je bilo motenj v sprintu manj od pričakovanih in da jim bo uspelo narediti več od plana, se s člani ekipe dogovorijo za naslednje postavke, ki jih bodo po vrsti jemali v razvoj. Poskrbljeno je, da je na seznamu zahtev na zalogi vedno dovolj izpopolnjenih postavk.

Scrum izdelki

Za vodenje seznama zahtev in načrta sprinta uporabljajo orodje Microsoft Team Foundation Server (TFS). Ker ekipa razvija produkta Init in Lokalizacijo, ki imata poseben pomen za celoten oddelek, ima do seznama zahtev ekipe dostop več oseb, med drugim produktni vodje ostalih ekip ter nekateri člani ekipe za podpore strankam. Te osebe imajo tudi pravico v seznam zahtev dodajati nove postavke. V načrt sprinta lahko posega le produktni vodja ekipe A. Projektna ekipa, ki je interni naročnik EDP projekta, ima dostop do seznama zahtev in načrta sprinta, končna stranka pa dostopa nima.

Definicija zaključenosti, ki si jo je predpisala ekipa, poleg razvoja in testiranja predvideva še pripravo dokumentacije in pregled kode. Za produkta Init in Lokalizacijo je dokumentacija bistvenega pomena, zato je vsaka dodelava dobro dokumentirana. Dokumentacijo za projekt piše projektna ekipa.

Scrum dogodki

Ekipa je začela z enomesečnimi sprinti, kar so po petih sprintih spremenili in zdaj delajo v dvotedenskih sprintih. V času intervjuja je potekal deseti sprint.

Načrtovanje sprinta se zgodi enkrat na dva tedna, in sicer na začetku sprinta v ponedeljek. Načrtovanje traja skoraj cel dan, kar je več kot v ostalih ekipah. Razlog za to je, da na ta dan v Ljubljano prideta tudi hrvaška člana ekipe, tako da dan izkoristijo tudi za izpopolnjevanje seznama zahtev, učenje, pregled narejenega in ostalo.

Izpopolnjevanje seznama zahtev se zgodi vsak teden. Na njem gredo skozi vse postavke seznama zahtev, ki še nimajo ocene potrebnega truda. Zaradi velikega števila postavk, ki izvirajo iz različnih virov, dajejo planiranju in obrazložitvi postavk velik pomen. Mančina naloga je imeti pregled nad novimi in starimi postavkami seznama zahtev ter sprotno razvrščanje po prioriteti, tako da so med izpopolnjevanjem seznama zahtev čim bolj učinkoviti.

Dnevni scrum se zgodi vsak dan, s tem da na njem sodelujejo samo slovenski člani ekipe. Hrvaška člana sodelujeta po potrebi preko telekonference.

Pregledu sprinta in retrospektivi dajejo manj poudarka. Več poudarka je na načrtovanju oz. definiciji zahtev ter iskanju rešitev in dizajnu. Je pa ekipa uvedla preglede kode, ki jih razvijalci izvajajo vzajemno. V ta proces so vključili tudi oba razvijalca projektne ekipe.

Komunikacija z naročnikom

Na projektu EDP ekipi A naročnika predstavlja projektni vodja oz. projektna ekipa. Le-ta ima svoj seznam zahtev in določene postavke tega seznama so dali v razvoj ekipi A. Na začetku sta si obe ekipi vzeli precej časa, da so postavke natančno pregledali in ekipa A je že v tistem trenutku izpostavila potencialne probleme in naredila spremembe dizajna. Te spremembe so se uskladile tudi z naročnikom. Nato sta ekipi uskladili še pričakovanja glede razpoložljive kapacitete, torej kdaj lahko želeno dobijo narejeno. Od začetka razvoja se redno usklajujejo, pregledujejo novonastale postavke ter spremembe prioritete.

Ko ekipa A napravi načrt sprinta, projektnemu vodji sporočijo, katere od njihovih nalog bodo realizirane v trenutnem sprintu. Na tedenskem nivoju se nato usklajujejo, ocenijo napredek in ugotovijo, če so se pojavile naloge z višjo prioriteto ali če je prišlo do zapletov, ki bi ogrozili dokončanje nalog. Na usklajevanjih vedno tudi opozorijo na dodatna vprašanja, ki so se pojavila pri izpopolnjevanju seznama zahtev.

Kadar se razvija večja funkcionalnost, ki sega preko več sprintov, produktna ekipa razvite dodelave tedensko predstavi skrbniku področja v projektni ekipi. Le-ta pri tem poda svoje komentarje, ki se upoštevajo pri razvoju, navadno v naslednjem sprintu. Po zaključku razvoja skrbnik funkcionalnost preveri in poda komentarje oz. javi napake, ki se rešijo v naslednjem sprintu.

Končni naročnik med razvojem ne izvaja testov. Validacija in verifikacija s strani končnega naročnika med razvojem potekata v obliki t.i. CRP delavnic (kratica angleškega izraza *conference room pilot*), ki se zgodijo približno enkrat mesečno in trajajo en do dva dni. Na posamezni delavnici svetovalci projektne ekipe prikažejo določen proces ter sklop razvitih dodelav na njem. Udeležijo se je ključni uporabniki področja, ki se predstavlja. Predstavitev poteka v obliki neinteraktivne demonstracije, se pa svetovalci trudijo v popolnosti simulirati proces naročnika, zato se za delavnico pripravijo dejanski podatki in poslovni primeri naročnika. Naročnik torej ve, da delo poteka po scrumu, da tečejo sprinti in da rešitve dobivajo v etapah. V zadnjih fazah projekta se bodo začele delavnice v obliki testiranja. Tam bodo uporabniki prvič lahko dejansko uporabljali rešitev, preverili testne primere in potrdili ustreznost rešitve.

Kljub temu je končni naročnik precej vpleten v projekt. Uporabniki so sodelovali že pri analizi, ko so se tudi prilagajali procesi. Ko je bil spisan dokument FDD, so svetovalci uporabnikom predstavili dizajn rešitve z demonstracijami v AX-u. Tudi med samim razvojem svetovalci redno na sestankih predstavljajo rešitev, preverjajo pravilnost domnev, skupaj z uporabniki dopolnjujejo šifrante itd.

Nastavitve

Veliko nastavitev, predvsem finančnega modula, je definiranih že v fazi analize. Med zajemom zahtev in pogovori s stranko o različnih specifičnih primerih se poskuša kar sproti nastavljati AX, tako da se lažje ugotovi potreba po spremembah. Če ima na primer stranka zelo podroben kontni načrt, se ugotovi, ali stranka res potrebuje tako natančne podatke ali se lahko nastavitev poenostavi. V veliko primerih se izkaže, da stranka lahko prilagodi svoj proces. V fazi analize se poskuša pridobiti tudi osnovne podatke (seznam osnovnih sredstev, produktov itd.) ter testne primere. Na podlagi nastavitev in testnih primerov se v fazi analize dosti lažje ugotovi obseg in vrsto potrebnih dodelav.

Veliko nastavitev in osnovnih podatkov je torej v fazi razvoja že znanih in nastavljenih. Vsak razvijalec ima svoje razvojno okolje, ki si ga prvotno skopira iz testnega, že nastavljenega okolja. Dodatne nastavitve na razvojnih okoljih si razvijalci pripravijo sami oz. jim pomaga svetovalec. Ko razvijalec svoje okolje usklajuje s spremembami, ki so jih naredili ostali razvijalci, se mu občasno zgodi, da potrebuje nastavitev, ki je še nima nastavljene pri sebi. V tem primeru si mora sam narediti ali skopirati nastavitev iz testnega okolja, če je tega preveč, pa lahko svoje okolje zavrže in si spet skopira testno okolje. To je zapleten in zamuden proces, vendar se ne dogaja pogosto. Razvijalci so namreč precej porazdeljeni po področjih, tako da se pri razvoju in razvojnem testiranju navadno ne dotikajo drugih funkcionalnosti in posledično nastavitve zanje niso potrebne.

Testiranje

Ekipa A nima pripravljenih avtomatiziranih regresijskih testov. Pri dizajnu so pozorni, na katere module bi lahko dodelava vplivala. Na splošno pa se trudijo, da bi s svojimi dodelavami čim manj posegali v standardno kodo. Če pa se že poseže v neko standardno funkcionalnost, to naredijo nastavljivo, tako da je možno spremembe izklopiti.

Vloga produktnega vodje

Produktni vodja ekipe A seznam zahtev redno pregleduje in dopolnjuje. Pozornost posveča tudi tistim postavkam, ki so nizko po prioriteti oz. po planu, saj v nasprotnem primeru nikoli ne bi prišle na vrsto. Svojo ekipo redno seznanja z nalogami, ki jih bo še potrebno opraviti. S tem jim poskuša podati širšo sliko in v njih spodbuja zavedanje o pomembnosti dokončevanja nalog.

Med sprintom produktni vodja ekipe A s pomočjo orodja TFS dnevno preverja trend zaključevanja nalog. Če opazi, da se stvari ne odvijajo tako, kot bi se morale, stopi do člana ekipe in preveri, ali mu bo uspelo končati vse naloge. Kadar se na ta način ugotovi, da katere od nalog ne bo možno zaključiti v sprintu, se jo tudi odstrani iz sprinta.

Produktni vodja ekipe A ne spremlja samo napredka na sprintu, ampak tudi napredek celotnega projekta, torej nalog, ki jih je ekipa A dobila v razvoj od projektne ekipe. Pri tem si pomaga s seznamom funkcionalnosti z ocenami, ki je nastal pred začetkom razvoja. Teh ocen ni podala ekipa, vendar vseeno služijo kot okvir pri ugotavljanju napredka v primerjavi s končnim ciljem. Večinoma pa kot merilo napredka služijo CRP delavnice, ki so neke vrste mejniki v projektu. Dogovorjeno je, kdaj bodo potekale te delavnice in katere funkcionalnosti morajo biti razvite, da se jih lahko na delavnicah prikaže. Če že ob načrtovanju sprinta in izpopolnjevanju seznama zahtev postane jasno, da neke dodelave ne bo moč razviti do delavnice, se to javi projektnemu vodji ter se poskuša reorganizirati.

2.3.2 Scrum v ekipi B

O scrumu v ekipi B sem govoril s Petrom, ki je v ekipi dve leti nastopal v vlogi vodje scruma. Peter je po izobrazbi diplomirani inženir računalništva in informatike ter magister poslovne informatike. Ima deset let delovnih izkušenj, od tega je na Adacti zaposlen pet let. Ob začetku uvajanja scruma na oddelek je v ekipi opravljal vlogo vodje razvoja. Prevzel je vlogo vodje scruma in jo opravljal skoraj dve leti. En mesec pred intervjujem je postal regionalni manager za razvoj. V ekipi B še vedno opravlja vlogo arhitekta rešitve, vlogo vodje scruma pa je predal drugemu članu ekipe.

Ekipa B se ukvarja z implementacijo in podporo AX vertikale MECOMS, ki je namenjena obračunu energije. Vertikalo MECOMS razvija belgijsko podjetje Ferranti Computer Systems, Adacta pa je njegov implementacijski partner na področju nekdanje Jugoslavije.

Projekt ABC

Ekipa B je bila sestavljena za potrebe projekta implementacije vertikale MECOMS pri stranki ABC (ime je zaradi zagotavljanja anonimnosti spremenjeno). Projekt je bil implementiran zaporedno modularno, saj je bil proces obračuna porabe plina v produkciji matičnega podjetja uporabljen že v roku enega meseca od začetka projekta, podpora za ostale procese (tudi ERP) pa se je dodajala postopoma, prav tako se je postopoma uvajalo rešitev v hčerinska podjetja. To pomeni, da se ekipa že od svojega nastanka ukvarja hkrati z razvojem novih funkcionalnosti in s podporo uporabnikom pri uporabi rešitve v produkciji.

Projekt XYZ

Skoraj štiri leta je ekipa izključno nudila podporo stranki ABC, v produkcijo inkrementalno vpeljevala nove module, razvijala nove funkcionalnosti in rešitev uvajala v povezana podjetja. Leta 2015 pa je pridobila implementacijski projekt za naročnika XYZ iz iste panoge. Projekt XYZ se je začel septembra 2015. Ideja je bila, da bi šlo za uvedbo obstoječe rešitve. Med analizo pa se je izkazalo, da je produkt, ki ga trži podjetje XY, precej bolj kompleksen, kot ga lahko podpre rešitev, pripravljena za podjetje ABC. Ker je bila ena od zahtev naročnika, da njegove stranke ne bi občutile spremembe programske opreme, je bilo potrebno narediti funkcionalnost skoraj identično po meri narejeni rešitvi, ki jo bo AX nadomestil. To je pomenilo dodaten razvoj in posledično tudi podrobnejšo analizo in načrtovanje. Prišlo je do spremembe načrta uvedbe, tako da se v prvi fazi podpre prodaja plina, ki se jo da podpreti z obstoječo rešitvijo, v drugi fazi pa se podpre še prodaja elektrike.

Ker je stranka XYZ locirana na Hrvaškem, se je za projekt implementacije formirala še ena ekipa v okviru Adacte Zagreb (v nadaljevanju »ekipa ZG«). Rutinske, enostavnejše dodelave na projektu dela manj izkušena zagrebška ekipa, slovenska ekipa pa razvija nove module in kompleksnejše dodelave. Želja je, da ekipa B na ekipo ZG tekom implementacije prenese znanje o rešitvi, tako da bo ekipa ZG po prehodu projekta v produkcijo sposobna nuditi podporo stranki in kasneje izvajati nove implementacijske projekte na Hrvaškem.

Poleg obeh AX ekip je v projekt vpletena še tretja ekipa, ki se ukvarja z implementacijo Microsoft Dynamics CRM produkta. Ta rešitev za upravljanje odnosov s strankami bo v podjetju XYZ uporabljena za celoten proces prodaje. To predstavlja večje odstopanje od projekta ABC, saj se tam večino prodajnega procesa odvije neposredno v sistemu AX. V okviru implementacije je bilo potrebno razviti kompleksno integracijo med AX in CRM sistemom.

S projektom XYZ se je začel oblikovati zaokrožen produkt, torej nek sklop dodelav nad MECOMS vertikalo. Vsaka od strank ima svojo ločeno izpeljanko te rešitve z dodelavami po meri. Med procesom razvoja na projektu XYZ se odločajo, ali dodelave vključiti v produkt, pri čemer pa upoštevajo morebiten vpliv na stranko ABC.

Sestava ekipe

Ekipa B obsega deset članov - štiri svetovalce in šest razvijalcev. Sestava ekipe je stalna, pri čemer imata dva člana ekipe zadolžitve tudi zunaj ekipe. Tako vodja scruma ekipe B opravlja še vlogo regionalnega vodje razvoja in tudi arhitekta rešitve v ekipi ZG, svetovalka, ki opravlja vlogo produktnega vodje ekipe B, pa opravlja isto vlogo tudi v ekipi ZG. Poleg teh dveh delnih članov ZG ekipo sestavljajo še trije svetovalci in trije razvijalci.

Ekipa B je v dveh letih od uvedbe scruma močno napredovala in je na svojem področju (vertikali) in v modulih, s katerimi se srečujejo na projektu, samozadostna. Če bi šli v nov implementacijski projekt standardnega AX, pa verjetno ne bi zmogli sami. Kadar je potrebno osvojiti novo znanje, bodisi z izobraževanjem, delavnico ali s pomočjo nekoga zunaj ekipe, to postane nova naloga v sprintu.

Podporne naloge

Ekipa se obenem ukvarja projektnim razvojem in s podporo strankam. Ker je podporne naloge nemogoče planirati in v sprint vnašajo kaos, so se organizirali tako, da se za vsak sprint določita en razvijalec in en svetovalec, ki sta tisti sprint zadolžena za podporo. Ta dva člana ekipe v tistem sprintu nimata razvojnih nalog. Tudi podporne naloge dobijo neko prvotno oceno. Če se pri reševanju problema doseže ocena, ne da bi se našla rešitev, k reševanju pristopi še nekdo drug, da se problem hitreje reši.

Scrum dogodki

Cikel sprintov je štirinajst dni. Nove različice za stranko ABC se na produkcijo nameščajo enkrat mesečno, projekt XYP pa je ravnokar v fazi prehoda v živo, zato reden cikel prenosa v produkcijo še ni vzpostavljen.

Ekipa izvaja vse scrum sestanke, torej načrtovanje sprinta, izpopolnjevanje seznama zahtev, pregled sprinta, retrospektivo sprinta ter dnevni scrum.

Ekipa B izvaja preglede sprinta skupaj z ekipo ZG. To je smiselno, ker obe ekipi delata na istem produktu. Na pregledu sprinta ena drugi predstavita, kako sta nadgradili produkt. Pregledi sprinta niso namenjeni strankam.

Retrospektiva se izvaja redno. Na retrospektivi vodja scruma in ekipa predlagata malenkostne izboljšave razvojnega procesa. Tako so se na retrospektivah na primer dogovorili za vpeljavo seznama opravil v definicijo zaključenosti, za protokol dela s TFS-jem, identificirali so problem z neupoštevanjem regresijskih testov v ocenah ipd.

Za bolj učinkovito izpopolnjevanje seznama zahtev je ekipa uvedla sestanek predizpopolnjevanje. Ta sestanek se je zgodil neposredno pred izpopolnjevanjem seznama zahtev, na njem pa so trije najizkušenejši člani ekipe dobro specificirali postavke. Kasneje se je koncept predizpopolnjevanja iz skupnega sestanka razvil v aktivnost, ko dva ali več članov ekipe skupaj natančno specificirajo postavke, vključno s testnimi scenariji ter idejami načrta rešitve.

Poseben dogodek, ki ga je ekipa B vnesla v svoj proces, da bi izboljšala transparentnost, so vmesni pregledi kode in postopkov testiranja, ki jih člani ekipe izvedejo pred dnevnim

scrumom. Namen teh pregledov je ugotoviti natančen status nalog, da lahko na dnevnem scrumu realneje ocenijo svoj napredek in preostalo delo.

Scrum izdelki

Za upravljanje seznama zahtev in plana sprinta ekipa B uporablja programsko rešitev JIRA podjetja Atlassian. Kljub temu, da ekipa B in ekipa ZG delata na istem projektu, imata ekipi ločena seznama zahtev in načrta sprinta. Ločitev je potrebna zato, ker imata do svojih scrum izdelkov dostop oba naročnika. Podjetji ABC in XYZ sta konkurenta, zato ni sprejemljivo, da bi imela vpogled v razvojne naloge drugega.

Ekipa je vzpostavila definicijo pripravljenosti – to je dokument, ki predpisuje potrebno obliko postavke seznama zahtev, da jo je možno uvrstiti v sprint. Po tej definiciji mora postavka seznama zahtev vsebovati odgovore na naslednja vprašanja:

- Kdo je sprožil zahtevo?
- Kaj je potrebno narediti?
- Zakaj je to potrebno narediti?
- Kaj je merilo, na podlagi katerega se meri uspešnost končanja naloge?
- Ali gre za možno razširitev standardnega produkta (Init, Lokalizacija)?
- Katere skupine uporabnikov bodo uporabljale razvito funkcionalnost?
- Kakšen je vpliv na integracije?

Poleg tega mora postavka vsebovati tudi model oz. načrt rešitve.

Definicija zaključenosti ekipe B vsebuje seznam korakov, ki morajo biti zaključeni, preden se naloga razume kot zaključena. Ta seznam poleg razvoja in testiranja predvideva tudi pregled kode in pregled testiranja, pripravo dokumentacije, obvestilo stranki o dodelavi in drugo.

Ekipa vzdržuje dokumentacijo modulov, v kateri se dokumentira tudi nastavitve, vključeni pa so tudi testni scenariji.

Komunikacija z naročnikom

Projekt ABC je v fazi podpore. Vse dodelave se naročijo in plačajo. Cena novih dodelav je narejena na podlagi dejanske ocene in dizajna. Ta analiza ni zaračunljiva, če se stranka ne odloči za razvoj. Ker pa večino dodelav, ki jih naroči, dejansko potrebuje, se to zgodi redko. Najpogosteje je umik zahteve posledica protislovij oz. nedefiniranosti zahtev, kar svetovalci v fazi analize s podrobnimi vprašanji hitro odkrijejo. Včasih ponudbo stranka tudi zavrne oz. ugovarja oceni. Kot posledica konfliktov tipa »to bi moralo tako delati«, se je s stranko

uvedel dogovor, da mora pred začetkom razvoja nove funkcionalnosti potrditi seznam testnih scenarijev, kar definira obseg dodelave. Ko je dodelava pripravljena, se predstavi stranki in se ji preda v testiranje.

Na XYZ projektu je verifikacije in validacije s strani stranke malo. En od razlogov za to je dejstvo, da je za en mesec zamujala infrastruktura, zaradi česar ni bilo mogoče vzpostaviti testnega okolja. Ker se rokov kljub temu ni premaknilo, se niso vzpostavile redne predstavitve, ampak se je komunikacija zreducirala na krajše prikaze posameznih funkcionalnosti.

Testiranje

Pri ERP sistemih je največji problem testiranje obsežnih obdelav, kjer je potrebno pripraviti veliko količino testnih podatkov. Vodja scruma ekipe B je dal kot primer testni mesečni obračun porabe energije, kjer je potrebno imeti veliko količino osnovnih podatkov, kot so stranke, odjemna mesta, poraba itd. S testiranjem tega so imeli velike težave, dokler niso pripravili kode za avtomatizirano testiranje. Le-ta je pred zagonom obračuna z uporabo standardnih procesov pripravila celoten set podatkov, po končanem obračunu pa je razveljavila zapis teh podatkov v bazo in ohranila le rezultate obračuna.

Tudi razvijalec vertikale MECOMS predvideva, da za vsako dodelavo pripraviš tudi generator testnih podatkov. V primeru, da neke podatke dobiš preko integracije z zunanjim sistemom, je potrebno veliko usklajevanja z lastnikom zunanjega sistema, da pošlje podatke, ki jih potrebuješ za testiranje nadaljnjih procesov v sistemu. V tem primeru je priročno pripraviti skripto, ki generira tipične primere podatkov.

Stranka ABC je dolgo časa teste izvajala kar v produkciji. Prenos dodelav v produkcijo se je na začetku izvajal vsaka dva tedna neposredno z razvojnega okolja. Če se je odkrila napaka, se je dogovorilo za poluren premor med delovnikom, da se je namestil popravek kode, nato pa se je v nekaj urah izvedla obdelava za popravek podatkov. Z razvojem projekta in uvedbami v hčerinska podjetja pa so se število uporabnikov, količina transakcij ter posledično stopnja tveganja močno povečali. Do tega zavedanja je prišlo šele po večjem zapletu, ki se je zgodil zaradi napake v kodi, odkrite šele ob zagonu obdelave v produkciji. Takrat so uvedli nov protokol izdaje novih verzij, ki predvideva mesečne prenose dodelav na testno okolje. Po potrditvi uporabnikov se po protokolu (pravila in postopki, opisani v dokumentu) nova verzija namesti na produkcijo. Pripravi se tudi dokument s popisom novih funkcionalnosti.

Z uvedbo scruma se je potreba po avtomatiziranih testih povečala, saj zdaj funkcionalnost ni zaključena, dokler testi niso uspešni. Novo zavedanje o pomembnosti testiranja se pozna pri ocenjevanju dodelav. Na začetku so ocene obsegale le razvoj in testiranje dodelave, zdaj pa vedo, da je potrebno narediti še regresijske teste, da zagotoviš inkrement, pripravljen za

izdajo. Stranka je zaradi še sveže bolečine to tudi pripravljena plačati. Tudi pri regresijskih testih poskušajo uvesti avtomatizirano testiranje, vendar so k temu storjeni le prvi koraki. Zaenkrat jim je uspelo na ta način avtomatizirati le testiranje ene integracije.

Vloga vodje scruma

Vodja scruma ekipe B je vestno opravljal naloge vodje scruma. Na začetku je redno izvajal vse scrum sestanke, vendar je sproti ocenjeval, kateri ekipi ustrezajo in kateri ne. Tiste, ki v ekipi zaradi določenih razlogov niso dosegli želenega učinka, je prilagodil. Obenem pa je vpeljal nove dogodke, za katere je ocenil, da bodo ekipi prinesli dodano vrednost in povečali njeno učinkovitost ali zadovoljstvo. Ves čas se je trudil za vključevanje vseh članov ekipe v razvojni proces, s čimer je povečal pretok znanja.

2.3.3 Pogled na metodologijo s perspektive managerja

O vpeljavi agilne metodologije scrum na oddelek sem se pogovarjal z Dejanom, ki v podjetju nastopa v vlogi regionalnega direktorja oddelka Dynamics AX. Direktor AX oddelka je po izobrazbi univerzitetni diplomirani inženir računalništva in informatike, z implementacijo ERP sistemov pa se ukvarja že več kot dvajset let.

Uvedba scruma

Direktor AX oddelka je bil pobudnik in sponzor vpeljave scruma v oddelek. Scrum se je v AX oddelek začel vpeljevati v začetku leta 2014. Povod za uvedbo scruma so bile težave na enem od projektov, kjer je bilo v nekem trenutku v ekipi dvajset ljudi, pa nihče ni imel pregleda nad tem, kaj kdo dela. Takrat se je začelo razmišljati, kako urediti proces. Začeli so z dnevnimi sestanki, kar pa zaradi velikosti ekipe ni funkcioniralo. Zato so ekipo razkosali na tri ekipe. Pri nadaljnji prenovi procesov jim je pomagal zunanji svetovalec, ki je po opravljeni analizi predlagal vpeljavo scruma.

Organizacijska struktura

Oddelek ima matrično organizacijo, in sicer gre za preplet funkcijske in projektne organizacije. Funkcijsko se oddelek deli na vsebinski oz. svetovalski del ter tehnični oz. razvijalski del. Vsaka funkcija ima svojega vodjo, ki ima pod seboj po svoji liniji skupine, te skupine pa imajo svoje vodje. Gre za stalne vodje, ki so odgovorni za ljudi, za njihovo ocenjevanje, nagrajevanje in karierni razvoj.

Projektno pa je oddelek organiziran v scrum ekipe, ki delajo na projektih ali produktih. Scrum ekipe so večfunkcijske, sestavljene iz arhitekta, razvijalcev in svetovalcev. Ekipo vodi produktni vodja, ampak samo za čas trajanja projekta. Produktni vodja članom ekipe ni nadrejeni, ampak je le oseba, ki skrbi za to, da se produkt oz. projekt pravilno razvija.

Funkcijska in projektna organizacija se marsikdaj prepletata. Produktni vodja je lahko nekaterim članom ekipe tudi funkcionalni vodja, ni pa nujno.

Matrična organizacija se je razvila sčasoma. Ob vpeljavi scruma je bila organizacija zgolj projektna - postavljene so bile scrum ekipe s svojimi produktnimi vodjami. Produktni vodja je skrbel za produkt oz. projekt in bil obenem tudi nadrejeni vsem članom ekipe, torej je skrbel za ocenjevanje, nagrajevanje in karierni razvoj. Taka organizacija je delovala, vse dokler se prvi projekti niso zaključili in so se ljudje začeli premikati na nove projekte. Takrat so njihovi nadrejeni popolnoma zgubili stik z njimi. Ugotovili so, da vsak zaposleni potrebuje osebo, ki je z njim ves čas v kontaktu, spremlja njegov razvoj, mu nudi povratno informacijo in mu pomaga. To naredi tako, da gre do produktnega vodje scrum ekipe, katere član je zaposleni trenutno, ter ga vpraša, kako je zadovoljen z njegovim delom. Na takem osebnem sestanku vodja pridobi podlago za povratno informacijo podrejenemu, obenem pa je tudi produktnemu vodji omogočeno podati povratne informacije osebi, kateremu sicer ni vodja.

Funkcionalni vodja z izjemo mesečnih osebnih sestankov in občasnih izobraževalnih delavnic nima nadzora nad časom svojega podrejenega. Članu scrum ekipe lahko naloge daje le produktni vodja. Kdorkoli potrebuje pomoč člana ekipe, mora iti do njegovega produktnega vodje in se z njim dogovoriti, ali in kdaj lahko član pomaga. Če gre za nujno zadevo in produktni vodja oceni, da je to pomembno za Adacto, takoj napoti člana ekipe na to nalogo. Če pa zadeva ni nujna, se uvrsti v seznam zahtev in se na naslednjem načrtovanju sprinta uvrsti v plan.

Ekipe

Vsi ljudje na oddelku, ki niso del ekipe za podporo strankam, so vključeni v eno od scrum ekip, tudi direktor AX oddelka. Ekipe so stalne, premikov ni veliko. Se pa zaradi pomanjkanja ljudi dogaja, da so nekateri člani več ekip hkrati. Gre predvsem za ključne ljudi, ki so v več ekip vključeni zaradi njihovih izkušenj in vsebinskega znanja. Primera sta na primer produktni vodja in arhitekt ekipe B, ki isti vlogi opravljata tudi v ekipi ZG. Obe ekipi namreč delata na isti vertikali, v Adacti Zagreb pa ljudi s poznavanjem te vertikale sploh ni. Posebno vlogo ima tudi najizkušenejši AX arhitekt oddelka, ki je član kar treh scrum ekip, vendar v vseh nastopa bolj v vlogi »misleca«, ki pomaga pri arhitekturi in usmerja razvoj v pravo smer. Od njega se ne pričakuje, da bo kaj razvil ali testiral niti da bo pripravil podroben dizajn. Njegov prispevek je predvsem preučiti določeno problematiko, na primer integracije med moduli, in svetovati najboljšo rešitev.

Projekti

Oddelek ima v času pisanja sedem aktivnih projektov. Vsi so projekti s fiksno ceno, kar pomeni, da je s pogodbo določeno, katere funkcionalnosti se bodo razvile in kdaj bo

implementacija končana. Da bi bili projekti tudi finančno uspešni, oddelek že v predprodajne aktivnosti vključuje svetovalce in arhitekte, ki lahko konkretno ocenijo potreben trud za izvedbo. Na ta način se prepreči podcenitev projekta zaradi nepoznavanja standardnega paketa.

Včasih se analiza in dizajn prodata kot samostojen projekt. Na podlagi dizajna se določita obseg in cena implementacijskega projekta, od katerega pa lahko tako naročnik kot tudi Adacta odstopita, če se jima zdi predrag oz. nedobičkonosen. Rezultat takega pristopa sta seznam funkcionalnih zahtev (FRD) in funkcionalni dizajn (FDD), ki vsebuje natančen opis ujemanja in odstopanja od standardnega produkta skupaj z ocenami potrebnega truda za razvoj in uvedbo. Naročnik tako ve, kaj bo dobil, Adacta pa lahko precej natančno določi, kaj je potrebno narediti in koliko časa bo to trajalo. Na podlagi tega se lahko pripravita natančna ponudba in pogodba. Na ta način je tudi dosti lažje upravljati z zahtevki za spremembo, ki nastanejo med projektom. Bolj kot je načrt natančen in bolj kot so opisi podrobni, lažje je argumentirati, da je neka dodelava zunaj obsega projekta in da jo je potrebno doplačati.

Projektni vodje

Pred uvedbo scruma so projekte vodili projektni vodje. Njihova naloga je bila skrbeti, da se je razvijal izdelek, ki ga je naročnik želel. Tako so bili zadolženi za komunikacijo z naročnikom in opredelitev zahtev, načrtovanje in nadzor projekta, koordinacijo vseh delovnih skupin, nadzor stroškov, kakovosti in tveganja. Pogosto so tudi podajali ocene za dodelave ter obljube naročniku glede rokov.

Z uvedbo scruma je oddelek ohranil vloge projektnih vodij. Njihova vloga je predvsem administracija projekta, skrb za zapisnike, izstavljanje faktur in pogajanje z naročnikom za dodatni proračun zaradi med razvojem odkritih komplikacij. Nimajo pa projektni vodje več vpliva na strokovne odločitve, saj je v scrumu odgovornost za le-te prešla na scrum ekipo. Projektni vodje so lahko zadolženi za enega ali za več projektov hkrati.

Sodelovanje z naročnikom

Sodelovanje z naročnikom je boljše kot pri tradicionalnem slapovnem pristopu, saj poteka ves čas trajanja projekta. Začne se pri analizi in verifikaciji dizajna, kjer uporabniki aktivno sodelujejo. Ker je AX standardni programski paket, se lahko analize izvajajo na dejanskih primerih v postavljenem testnem okolju. Tu že dobijo prve informacije, kako bo videti produkt.

Aktivno sodelovanje naročnika se nadaljuje tudi v fazi razvoja. Kako, je odvisno od scrum ekipe in projekta. Na projektu EDP se recimo enkrat na sprint zgodi sestanek, imenovan »pregled napredka razvojnih aktivnosti«. Na tem sestanku se zberejo produktni vodja,

projektni vodja, direktor AX oddelka in dva arhitekta s strani naročnika. Arhitektoma se prikaže, na katerih nalogah dela ekipa v tekočem sprintu ter katere naloge so načrtovane za naslednje sprinte. Onadva pa lahko spremenita prioritete in s tem vsebino naslednjih sprintov. Razlogi za to so lahko različni: lahko da je tretji partner, s katerim je potrebno narediti integracijo, zdaj pripravljen začeti z delom, ali pa sta arhitekta ocenila, da uporabnike skrbi delovanje neke določene funkcionalnosti in bi to radi videli čim prej. Taka transparentnost in sodelovanje s strani naročnika ima pozitiven vpliv na projekt, ekipa pa ve, kaj vse je potrebno narediti, zato vrstni red nalog ne naredi velike razlike.

3 ANALIZA UPORABLJENE METODOLOGIJE

Na podlagi opravljenih intervjujev sem dobil vpogled v organizacijo oddelka in njegovo delovanje. Kot sem predvideval, je implementacija scruma prilagojena potrebam oddelka in specifikam razvoja ERP. V tem poglavju bom opravil analizo procesa implementacije ERP sistema, analizo implementacije scruma v primerjavi s čistim scrumom ter na koncu še analiziral organizacijo oddelka.

3.1 Water-Scrum-Fall

Številni avtorji (West et al., 2011; Boehm & Turner, 2003) ugotavljajo, da se scrum večinoma uporablja v kombinaciji z bolj tradicionalnim slapovnim pristopom. To se je izkazalo tudi na primeru AX oddelka. Projekti se začnejo s fazama analize in dizajna, ki jim sledi faza razvoja, ki poteka po scrumu. Faza testiranja se začne že tekom razvoja, vendar se večji del testiranja izvede na delavnicah, ki se organizirajo v končnih sprintih razvoja. Na koncu sledita faza vpeljave v produkcijo in prenos projekta v ekipo za podporo strankam.

Direktor AX oddelka je pojasnil, da je bila odločitev za tak pristop dosežena zaradi tipa projektov. Če bi želeli delati popolnoma agilno, bi potrebovali agilno pogodbo. To pomeni, da je projekt plačan po dejanski porabi časa in materiala ter nima fiksnega obsega, ampak se iz sprinta v sprint dogovarja, kaj se bo razvijalo. Ker obseg ni fiksen, zahtevki po spremembi, ki nastanejo med razvojem, ne predstavljajo problema. Take spremembe se z veseljem implementira, saj se ve, da so koristni za naročnika in obenem ne vplivajo na finančni oz. terminski uspeh projekta. Projekti AX oddelka pa niso taki, saj so vsi prodani za fiksno ceno in imajo v pogodbi točno določene funkcionalnosti, ki jih je potrebno razviti. V takem primeru zahtevek za spremembo navadno ne pomeni, da se naročnik odpoveduje eni od dogovorjenih funkcionalnosti. Gre za dodatno naročilo, za realizacijo katerega so potrebni dodatni viri ali dodaten čas, kar pa je naročniku marsikdaj težko razložiti. Taki zahtevki za spremembo niso planirani in premikajo roke ter povečujejo kompleksnost, zato se jim poskušajo izogniti.

Skladno z ugotovitvami West et al. (2011) je glavni razlog za ločeni fazi analize in dizajna finančne narave, saj je zavezujoča ponudba pogoj za pridobitev projekta. Zaenkrat na oddelku še niso naleteli na naročnika, ki bi bil pripravljen financirati popolnoma agilen projekt, pri katerem ne bi točno vedel, kaj bo dobil do roka oz. kdaj bo dobil želene funkcionalnosti. Po opravljeni analizi in napravljenem dizajnu je Adacta zmožna natančno oceniti, katere dodelave bo potrebno razviti in koliko časa in truda bo potrebno za realizacijo projekta. Tako lahko naročniku odda ponudbo, na podlagi katere se kasneje sklene pogodba. Na ta način ima naročnik neke vrste zagotovilo, da bo na koncu projekta dobil delujočo rešitev, ki bo pokrila njegove potrebe. Ta pogodba pa je koristna tudi za Adacto, in sicer kot vzvod za nadzorovanje in omejevanje dodatnih zahtev, ki bi negativno vplivale na potek projekta. To pa seveda ne pomeni, da spremembe med procesom razvoja niso mogoče. Z redno komunikacijo z naročnikom, bodisi v obliki CRP delavnic bodisi na kak drug način, se izvajata verifikacija in validacija, korekcije pa se izvedejo v naslednjih sprintih.

Na podlagi izkušenj s projektom XYZ bi lahko sklepali, da scruma v fazi vpeljave v produkcijo ni mogoče izvajati v popolnosti. Potrebno je zaključiti vse nedokončane funkcionalnosti, obenem pa se začnejo šolanja in delavnice. Nekatere scrum aktivnosti se opustijo. Tako se ekipa ne ukvarja več s kapaciteto, retrospektivami in podobnim, še vedno pa se ohranijo dnevni sestanki, izpopolnjevanje seznama zahtev, definicija zaključenosti in ostali koncepti, ki zagotavljajo kvalitetno izvedeno delo. Vodja scruma ekipe B kot najpomembnejšo scrum aktivnost, ki se je tudi v končnih fazah projekta ne sme zanemariti, izpostavlja načrtovanje. Dobro se je namreč vsaj enkrat na dva tedna vprašati, katere naloge so se zaključile, katere se niso in katere so se pojavile na novo. Na projektu XYZ do prehoda v živo vse naloge niso bile zaključene, so pa bile zaradi sprotnega načrtovanja končane najpomembnejše naloge. Ker pa je v okviru oddelka projekt XYZ zaenkrat edini, ki se je začel in končal z uporabo metodologije scrum, ni mogoče izpeljati zaključka, da je tako dogajanje običajno in pričakovano.

3.2 Scrum dogodki

Ob uvedbi scruma je zunanji svetovalec predlagal način dela, ki je precej ustrezal osnovnim smernicam scruma. Predlagano trajanje sprinta je bilo štirinajst dni. Izvajala naj bi se večina sestankov - načrtovanje sprinta na ponedeljek prvega tedna, enkrat tedensko ena ura izpopolnjevanja seznama zahtev. Na četrtek zadnjega tedna je bila predlagana retrospektiva sprinta. Pregled sprinta kot poseben sestanek ni bil predviden, pač pa naj bi se izvajal sproti v obliki predaje končanega dela razvijalca svetovalcu, ki narejeno potrdi. Predlagana aktivnost, ki je odstopala od klasičnega scruma, je bil dan za pripravo (angl. *get-ready day*), ki naj bi potekal vsak zadnji dan v sprintu. Na ta dan naj bi se vsak član ekipe poglobil v naloge naslednjega sprinta in opravil potrebne raziskave, kar naj bi botrovalo boljši pripravljenosti na naslednji sprint.

3.2.1 Sprinti

Dvotedenski sprinti, ki so se uveljavili na oddelku, so v skladu s priporočili scrum metodologije. Če se je lahko ob uvedbi scruma vsaka ekipa zase odločila za trajanje sprinta, se je izkazalo, da neusklajenost sprintov močno otežuje komunikacijo in sodelovanje med ekipami. Zato je celoten oddelek prešel na dvotedenske sprinte, vsi so se tudi začeli enako številčiti. Tako je usklajevanje med ekipami in projekti precej lažje, saj ni potrebno več razmišljati, kdaj posamezna ekipa začne s sprintom in kako dolgo ta traja. Usklajenost sprintov med ekipami je pomembna tako ekipi A kot tudi ekipi B, saj z ostalimi ekipami veliko sodelujeta.

Kljub uvodnim priporočilom o dvotedenskem ciklu sprintov so ekipe preizkušale tako daljše kot tudi krajše cikle. Na podlagi teh izkušenj je bila sprejeta odločitev za enotne dvotedenske sprinte na celotnem oddelku. Ekipa A je na začetku imela enomesečne sprinte. Ugotovili so, da je sprint predolg, med trajanjem sprinta se lahko spremeni preveč stvari, ekipa pa je rigidna in ne more pravočasno reagirati. Člani ekipe sicer nad prehodom na dvotedenski sprint niso bili najbolj navdušeni, saj so bili mnenja, da je razmerje med časom, porabljenim za sestanke, in časom, porabljenim za razvoj, preveč nagnjeno v korist sestankov. Produktnemu vodji ekipe A pa dvotedenski sprinti precej bolj ustrezajo, ker so bolj obvladljivi. Če bi ohranili enomesečne sprinte, bi bilo tekom sprinta več sprememb, s čimer bi se zmanjšala preglednost.

Vodja scruma ekipe B ima še en pogled na sprinte, daljše od dveh tednov. Poleg vloge vodje scruma je opravljal tudi delo razvijalca in arhitekta rešitve. Pri tem se mu je zgodilo, da se je zakopal v problem in spregledal, da se je nekomu zataknilo in da porablja preveč časa na napačni stvari. Pri dvotedenskem sprintu je škoda, če nekdo zaide, manjša kot pri npr. enomesečnem sprintu. Poleg tega je dvotedenski sprint dober tudi za vsebinsko retrospektivo. Če ti v časovnem obdobju dveh tednov ni uspelo končati naloge, je to dober signal, da moraš prekiniti razvoj in razmisliti, zakaj ti ni uspelo. Morda je dobro za naslednji sprint nalogo podrobneje definirati.

Vodja scruma ekipe B ima izkušnje tudi s sprinti, krajšimi od dveh tednov. Nekajkrat se je ekipa B zaradi praznikov odločila, da bo sprint trajal samo en teden. Vodja scruma ekipe B je opazil, da so pri enotedenskem sprintu za vse scrum dogodke (načrtovanje sprinta, dnevni scrum, pregled sprinta, izpopolnjevanje seznama zahtev, …) porabili preveč časa v primerjavi s časom, porabljenim za delo na nalogah. Poleg tega je en teden premalo časa za neko konkretno delo. Če želiš imeti vse naloge končane znotraj sprinta, ima v enotedenskem sprintu razvijalec približno dan in pol časa za razvoj in razvojno testiranje, da lahko funkcionalnost preda svetovalcu, ki ima spet približno dan in pol časa za testiranje in potrditev.

3.2.2 Načrtovanje sprinta

Obe ekipi načrtovanje sprinta izvajata prvi ponedeljek v sprintu. Scrum metodologija ta sestanek predvideva na zadnji dan sprinta, vendar je večina ekip ugotovila, da na koncu sprinta vedno zmanjka časa za zaključevanje nalog. Težko si je vzeti skoraj cel dan in osredotočeno načrtovati naslednji sprint, ko še nisi zaključil trenutnega sprinta.

3.2.3 Dnevni scrum

Dnevni scrum izvajajo vse ekipe in vsi se strinjajo, da je to dober način izmenjave informacij znotraj ekipe. Kljub temu pa se dogaja, da se na tem sestanku ne identificirajo problemi. Produktni vodja ekipe A na primer opaža, da razvijalci po celem dnevu nenačrtovane pomoči ekipi za podporo strankam ne preverijo svoje naloge in je ne opozorijo, da je nastal problem. Zato produktni vodja ekipe A pozorno spremlja trend zaključevanja nalog in samoiniciativno stopi do člana ekipe in ga povpraša o statusu njegovih nalog. To ob pravilnem izvajanju scruma ne bi smelo biti potrebno.

3.2.4 Izpopolnjevanje seznama zahtev

Obe ekipi redno izvajata izpopolnjevanje seznama zahtev, in sicer v polni zasedbi ekipe. Se pa obe ekipi srečujeta s težavami, ki jih bolj ali manj uspešno rešujeta.

V ekipi B se je pri izpopolnjevanju seznama zahtev pokazala velika razlika v znanju znotraj ekipe - ocenjevanje nalog je potekalo počasi, saj je veliko časa namenjalo vsebinski razlagi zahtev manj izkušenim članom, pa še potem ti dostikrat niso razumeli nalog in načrtov. Poleg tega se je pogosto dogajalo, da so se podrobnosti zahtev razčiščevale na samem izpopolnjevanju seznama zahtev, kar je zaradi poglobljenosti diskusije povzročilo izključenost manj izkušenih članov ekipe ter posledično izgubo pozornosti. Problem se je rešil z vpeljavo predizpopolnjevanja kot aktivnosti, s katero se za vsako zahtevo natančno definira naloge, vključno s testnimi scenariji ter idejami načrta rešitve. Na ta način se je doseglo, da se na izpopolnjevanju seznama zahtev ne izgublja časa s preverjanjem, ali je zahteva ustrezno definirana, ampak se zgolj ocenjuje potreben trud za njeno izpolnitev. Če se na izpopolnjevanju seznama zahtev ugotovi, da je zahteva pomanjkljivo definirana, se jo zavrne in se je ne ocenjuje.

Sestavni del izpopolnjevanja seznama zahtev je tudi ocenjevanje potrebnega truda za izvedbo naloge. Pri ocenah, ki jih podajo posamezni člani ekipe, se pojavljajo razlike, na katere po mnenju vodje scruma ekipe B vplivata dva faktorja: izkušenost in funkcijsko področje. Manj izkušeni člani ekipe tipično zgrešijo ocene na področjih, ki jih ne poznajo. Z izkušnjami se navadno ocene višajo, saj je višje tudi zavedanje o potrebnih aktivnostih, na katere manj izkušeni pozabijo. Na začetku se je na primer dogajalo, da je bila ocena dana samo za razvoj, sčasoma pa se je vzpostavilo zavedanje, da je razvito potrebno tudi testirati,

zato so bile ocene vedno višje. V določenih primerih bolj izkušeni člani poznajo načine, kako do rezultata priti na enostavnejši način, kar močno zniža oceno. Večje težave imajo pri ocenjevanju svetovalci, še posebej ko je potrebno oceniti kakšno bolj tehnično nalogo. Medtem ko razvijalec z izkušnjami pridobiva tudi vsebinsko znanje, svetovalec navadno tehnično znanje osvoji le do neke mere. Kljub temu je sodelovanje svetovalca pri ocenjevanju tehničnih nalog zaželeno in spodbujano, saj mora tudi svetovalec pridobiti vsaj okviren občutek, koliko časa je potrebno za določene vrste nalog. Svetovalci so namreč več v stiku s stranko in marsikdaj morajo podati hitro oceno oz. obljubo, do kdaj bo nekaj narejeno.

Produktni vodja ekipe A ima z ocenjevanjem še drugačne izkušnje. Ocene postavk seznama zahtev ekipe A niso slabe, bi pa lahko bile boljše. Tukaj je ekipa že močno napredovala. Izkazalo pa se je, da nekateri člani ekipe praviloma podajajo prenizke ocene. Ker je ocenjevanje ekipno, lahko preostanek ekipe tako na podlagi poznavanja področja in izkušenj kot na podlagi preteklih izkušenj z napačnimi ocenami tega člana ekipe njegovo oceno popravi.

Izpopolnjevanje seznama zahtev ima velik pozitiven vpliv na deljenje znanja. Vodji scruma ekipe B se je pogosto pripetilo, da je član ekipe določeno nalogo na podlagi izkušenj iz drugih sistemov dojemal kot zelo zahtevno, dokler mu ni bilo razloženo, da je moč to v AX narediti precej enostavneje. Včasih je kdo tudi pozabil kak pristop k problemu in je potem na izpopolnjevanju seznama zahtev osvežil znanje.

Tudi produktni vodja ekipe A je opazil pozitiven učinek izpopolnjevanja seznama zahtev na znanje v ekipi. Ker morajo vsi člani ekipe sodelovati pri ocenjevanju, se morajo vsi poglobiti v problem, da bi razumeli vsebino naloge. V tem pogledu produktni vodja ekipe A opaža težavo pri enemu od razvijalcev, ki je izrazito osredotočen na eno samo področje. Ko se izpopolnjujejo postavke z njegovega področja, pri ocenjevanju sodelujejo vsi, on pa ne sodeluje vedno pri ocenjevanju ostalih postavk.

3.2.5 Retrospektiva sprinta

Retrospektiva sprinta v scrumu predstavlja ključni dogodek za nadzor in prilagajanja procesa v ekipi. Ker pa za projekt ni ključna, jo marsikatera scrum ekipa izpusti.

Produktni vodja ekipe A priznava, da to ni dobra praksa. V njeni ekipi retrospektive ne izvajajo in zadnjih nekaj sprintov opaža, da se pravil scruma držijo vedno bolj ohlapno. Preveč se tolerira, da se sprint ne zaključi v celoti in da ostanejo stvari nedokončane. To je včasih utemeljeno, ker se kljub planiranju in dizajnu stvar zaplete, včasih pa nalog ne morejo končati zaradi novih, nenačrtovanih zahtev, ki se pojavijo, ko sprint že teče. Takrat niso dovolj dosledni, da bi sprint reorganizirali in iz njega odstranili odvečne naloge, zato ne dosežejo ciljev sprinta. Produktni vodja ekipe A je izpostavil tudi slabo disciplino na

sestankih, saj člani ekipe med sestanki uporabljajo telefone oz. računalnike, zaradi česar niso osredotočeni na sestanek. Vse to so teme, ki bi se jih moralo predelati na retrospektivi sprinta, da bi se scrum ekipa razvijala in vzdrževala optimalno učinkovitost.

Vodja scruma ekipe B po drugi strani razume pomen retrospektive in jo s pridom izkorišča za vpeljavo izboljšav procesa v ekipo. Na prvih retrospektivah ekipa ni aktivno sodelovala, težko je bilo dobiti kako mnenje ali idejo za izboljšavo. Zdaj pa so te retrospektive precej bolj konstruktivne in učinkovite. Izkazalo se je, da ni mogoče uvesti veliko sprememb naenkrat. Ekipa lahko dojame in vpelje eno ali dve manjši prilagoditvi procesa, večje spremembe pa hitro izzvenijo. Retrospektiva vzame veliko energije, zato ni dobro, da se izvaja na isti dan kot načrtovanje in pregled sprinta. Je tudi dobra priložnost, da se pohvali dobro narejeno delo.

3.2.6 Posebnosti

Dan za pripravo, ki ga je zunanji svetovalec predlagal kot aktivnost zadnjega dne sprinta, ni uspel v nobeni ekipi. Vodja scruma ekipe B razlaga, da sta za neuspeh kriva dva glavna razloga. Prvi je ta, da naloge sprinta večinoma še niso bile končane, zato se jih je zaključevalo še ta zadnji dan. Drugi razlog pa je dejstvo, da večina ekip nima načrtovanih sprintov vnaprej, zato se nanje ni mogoče pripraviti.

V ekipi B se po potrebi vpelje t.i. čas tišine, to je nekaj ur v dnevu, ko člani ekipe v miru delajo na postavkah sprinta, zato motnje s strani članov in nečlanov ekipe niso zaželene. To se je izkazalo za učinkovit ukrep v situacijah, ko določeni člani ekipe zaradi neprestanih motenj niso uspeli dokončati svojih nalog.

3.3 Scrum vloge in izdelki

Na AX oddelku vloga razvojne ekipe ne pride do izraza, saj sta produktni vodja in vodja scruma vedno tudi del razvojne ekipe, torej opravljata tudi razvojne naloge. Tako produktni vodja poleg seznama zahtev administrira tudi načrt sprinta, kar je po scrumu naloga razvojne ekipe. Včasih produktni vodja opravlja tudi nekatere naloge vodje scruma, še posebej kadar le-ta ni dovolj zavzet za svojo vlogo. V takšnih ekipah ima produktni vodja pravzaprav vlogo vodje ekipe iz klasične organizacije. Menim, da to za scrum ekipo ni najboljše, saj se lahko izgubi občutek skupne odgovornosti za rezultate.

Scrum ne pozna vloge projektnega vodje in direktor AX oddelka razlaga, da se je z uvedbo scruma na AX oddelek pojavilo vprašanje, kaj je sploh še vloga projektnega vodje oz. kaj narediti z ljudmi, ki so opravljali to vlogo v prejšnji organizaciji. Cohn (2010) pojasnjuje, da projektni vodje ob uvedbi scruma navadno prevzamejo eno od scrum vlog, ki po scrumu nosijo odgovornost za aktivnosti, za katere so prej skrbeli projektni vodje. Tako lahko

projektni vodja prevzame vlogo produktnega vodje, vodje scruma ali člana razvojne ekipe - odvisno od njihovih izkušenj, sposobnosti, znanja in interesov.

V AX oddelku so ohranili vlogo projektnega vodje, ki je zdaj predvsem administrator projekta, ki skrbi za zapisnike, izstavlja fakture in podobno. Na strokovne odločitve pa projektni vodja nima več vpliva. Kljub temu imajo projektni vodje na projektih pomembno vlogo. Tudi na strani naročnika je nekdo v vlogi projektnega vodje in potrebuje sogovornika na strani Adacte. To ne sme biti produktni vodja, saj je konflikt, ki nastaja med ljudmi, preveč vpletenimi v implementacijo, in projektnim vodjo na strani naročnika preveč čustven. Projektni vodja ni nikoli toliko vpleten v implementacijo, da bi poznal posamezne funkcionalnosti. Zato mu ni težko slišati na primer očitka, da je nekaj slabo narejeno, medtem ko produktni vodja točno ve, koliko truda je bilo vloženo, da se je prišlo do te vsaj približno pravilne rešitve. Zato lahko projektni vodja na projekt in težave na njem gleda objektivno in predlaga rešitve, najboljše za obe strani.

3.3.1 Seznam zahtev in načrt sprinta

Ekipa A ima vzorno urejen seznam zahtev v smislu preglednosti vsem interesnim skupinam. Tako imajo do seznama zahtev dostop vsi tisti, ki imajo pravico dodati novo postavko, in vsi tisti, ki jih lahko zanima status postavk.

V ekipi B vlada situacija, ki je v literaturi nisem zasledil, in sicer da ekipa sočasno dela za dve konkurenčni podjetji. Posledično ekipa vodi dva seznama zahtev, kar otežuje načrtovanje in krni preglednost. Ta problem je bolj natančno opisan v poglavju 3.5.

Tako produktni vodja ekipe A kot tudi vodja scruma ekipe B izpostavljata problem pomanjkljivega nadzora nad postavkami načrta sprinta. Predvsem je sporna nedisciplina pri umikanju nalog iz načrta sprinta, ko postane jasno, da načrtovanih nalog ne bo mogoče opraviti zaradi naknadno dodanih nujnih nalog. Zaradi tega prihaja do lažnih padcev sprintov, saj je videti, kot da ekipa ni uspela zaključiti vseh nalog, čeprav so bile v resnici nekatere naloge zavestno izpuščene. To zmanjšuje preglednost pri ugotavljanju statusa sprinta. Če je naloga izpuščena, pa ni umaknjena iz načrta sprinta, potem produktni vodja vidi slabšo sliko, kot je v resnici. Bolj nevarna pa je situacija, ko je produktni vodja vajen, da izpuščene naloge ostajajo v načrtu sprinta, in zato ne zazna, da neka naloga res ni opravljena.

3.3.2 Definicija zaključenosti in definicija pripravljenosti

Ekipni zavezi - definiciji pripravljenosti in zaključenosti - v katerih je določeno, kaj vse mora vsebovati zahteva, da se lahko začne razvoj, in kaj vse je potrebno narediti, da je razvoj zaključen, vsaka ekipa prilagodi po svoje. Pri tem so nekatere ekipe bolj uspešne od drugih. Vseeno pa so nekatere stvari skupne in se pričakujejo na nivoju oddelka. Izkušnje so

pokazale, da nekatere ekipe ne spoštujejo ekipnih zavez, oz. so te zaveze premalo natančne. Če v ekipi kakovost šepa, to pomeni, da ima slabe vhodne definicije in slabe izhodne kontrole. To nakazuje, da ekipe ne izvajajo dosledno retrospektiv sprinta, na katerih bi take probleme identificirali in pripravili izboljšave.

3.4 Ekipe

Analiza ekip pokaže, da AX oddelek ni povsem agilen. Ekipe so sicer načeloma večfunkcijske in samozadostne, niso pa stalne. Prilagajajo se potrebam projektov, posamezniki pa so lahko člani več ekip naenkrat.

Direktor AX oddelka razlaga, da je na ta način zagotovljena fleksibilnost oddelka pri soočanju z novimi projekti, predvsem pa izkoriščenost ključnih ljudi na oddelku. Le-ti imajo namreč večjo vlogo v ključnih fazah projekta kot načrtovalci, nato pa nastopajo bolj kot usmerjevalci drugim članom ekipe. Taka organiziranost je bolj podobna klasičnemu slapovnemu pristopu, kjer najbolj izkušeni ljudje opravijo analizo in načrtovanje, implementacijo pa prepustijo manj izkušenim članom (Hopkins & Jenkins, 2008). Je pa Adactin pristop tu vendarle bolj agilen, saj ključni člani ekipe niso prisotni zgolj pri analizi in načrtovanju, ampak so še vedno del ekipe in sodelujejo pri vseh scrum dogodkih in tudi pri implementaciji. Na ta način lahko oddelek njihovo znanje izkoristi na več projektih hkrati, namesto da bi bili vezani na eno samo ekipo in projekt v vseh njegovih fazah. S sodelovanjem v več ekipah naenkrat pa je omogočeno večje prehajanje znanja teh oseb na ostale ljudi na oddelku. Predvsem pa taka organizacija olajša proces prodaje in časovnega načrtovanja novih projektov. Redko se zgodi, da se projekt zaključi in se točno takrat začne nov projekt. Navadno se projekti vsaj deloma prekrivajo, kar pomeni potrebo po prerazporejanju ljudi. To je včasih potrebno tudi zaradi specifičnih znanj, ki jih imajo točno določeni zaposleni - ta znanja se poskušajo uporabiti na novih projektih, kjer je le mogoče. Stalne agilne ekipe take fleksibilnosti ne omogočajo.

Poleg zgoraj naštetih prednosti pa ima taka organizacija seveda tudi slabe strani. Predvsem bi izpostavil manjšo povezanost ekip in posledično manjši prirastek učinkovitosti. Kot opozarjajo West et al. (2011), agilne metodologije spodbujajo formiranje ekip, ki skupaj sodelujejo v več projektih. S tem se odpravi potreba po vsakokratnem privajanju članov ekipe na skupno delo. Prav tako se lahko ob reformiranju ekip izgubijo dobre prakse, ki jih je ekipa razvila med projektom. Pozitivne strani stalne ekipe bi izpostavil na primeru ekipe B, ki je že nekaj let dokaj nespremenjena. Od uvedbe scruma se ekipa neprestano izpopolnjuje in išče nove izboljšave procesa. Včasih se izboljšave sicer ne obnesejo, vendar spremembe limitirajo k optimalnemu delovanju. Če bi to ekipo razbili oz. jo koreniteje spremenili, bi bilo potrebno vse te izboljšave procesa ponovno oceniti, saj morda v novi ekipi ne bi več funkcionirale.

Uvedba scruma je po mnenju produktnega vodje ekipe A pozitivno vplivala na ekipo. Prva pozitivna stvar so jasni cilji - znano je, kaj je potrebno doseči v nekem časovnem okviru. Poleg tega so člani ekipe bolj predani, ker so od njih odvisni tudi drugi. Razvijalec mora funkcionalnost razviti, da jo lahko svetovalec testira. Tako je vedno prisoten nek notranji pritisk, kar povečuje učinkovitost. Prisoten je tudi ekipni duh - sprint je en za celo ekipo in če sprint pade, pade za celo ekipo. To vodi v proaktivnost - če nekdo nečesa ne ve, bo vprašal ostale člane ekipe, vse z namenom, da se naloga reši in da sprint ne pade.

3.4.1 Distribuiran scrum

Na oddelku se pojavlja tudi geografska oddaljenost članov ekip, kar označujemo z izrazom **distribuiran scrum** (angl. *distributed scrum*). Tako sta v ekipi A dva člana locirana na Hrvaškem in ena članica v Srbiji, hrvaška ekipa ZG pa ima produktnega vodjo in arhitekta (Petra) iz Slovenije. Literatura (Paasivaara, Durasiewicz, & Lassenius, 2009; Korkala & Abrahamsson, 2007) kot največji problem distribuiranega scruma izpostavlja komunikacijo. To se je izkazalo tudi v ekipi A, predvsem pri razvijalcih. Zaradi oddaljenosti je bilo težko nadzirati njihovo delo, kar se je čutilo na kvaliteti razvitih dodelav. Iz tega razloga so že pred časom iz ekipe umaknili razvijalca iz Adacte Beograd. Tudi svetovalko iz Beograda uporabljajo le kot vir informacij in ne opravlja nalog iz sprinta. Precej boljšo izkušnjo imajo s hrvaškimi sodelavci, predvsem zaradi redne osebne prisotnosti na planiranjih sprinta.

Produktni vodja ekipe A največji doprinos take organizacije in sestave ekipe vidi v dvigovanju ravni znanja in zavedanja o produktih hrvaških sodelavcev. Distribuirana sestava scrum ekipe torej terja prilagoditve scrum dogodkov. To zna biti za ekipo obremenjujoče (celodnevni sestanki enkrat na dva tedna, komunikacija preko telefona oz. telekonference), vendar je širšega pomena za Adacto, saj zagotavlja pretok informacij med podjetji v skupini. Tako hrvaški člani v ekipi A skrbijo za pravočasne informacije o zakonskih spremembah na Hrvaškem in za razvoj dodelav, ki te spremembe pokrivajo, obenem pa služijo kot vir informacij o razvoju produktov Init in Lokalizacija vsem AX ekipam v Adacti Zagreb. Tudi ekipa ZG ima distribuirano sestavo iz enakega razloga - zaradi pomanjkanja izkušenega kadra v Adacti Zagreb in posledične potrebe po prenosu znanja.

Lahko bi rekli, da distribuiran scrum sicer negativno vpliva na učinkovitost ekipe, vendar ima pozitiven vpliv na rast in razvoj AX oddelka.

3.4.2 Kolektivno lastništvo kode

Kolektivno lastništvo kode, kot ga zagovarjajo agilne metodologije, je pri implementacijah ERP sistemov težko doseči. Kadar se agilni pristop uporablja na projektu, kjer se programska oprema razvija povsem na novo, se ukvarjaš predvsem s tehničnimi problemi, vsebinsko znanje pa prihaja od naročnika oz. se pridobiva tekom projekta. V tem primeru ni težko sestaviti ekipe, ki bi imela vsa potrebna tehnična znanja oz. imela sposobnost naučiti se vseh

potrebnih tehnologij za izvedbo projekta. V taki ekipi se lahko katerikoli član loti katerekoli naloge, zato velja tudi skupna odgovornost za kodo. Pri implementaciji ERP sistemov je to bistveno težje, saj je potrebno imeti veliko vsebinskega znanja ter znanja o delovanju standardne rešitve, da bi lahko razvil pravilne in učinkovite dodelave. Ljudi s takim znanjem pa je malo, zato je praktično nemogoče sestaviti ekipo, v kateri bi se lahko katerikoli član lotil katerekoli naloge.

Temu idealu pa se lahko približaš z ekipo, ki je skupaj daljše obdobje, kot je na primer ekipa B. V tej ekipi se je že razvilo neke vrste kolektivno lastništvo kode, kar se kaže v načinu usklajevanja podpornih in razvojnih nalog. Kratkoročno bi bilo bolj učinkovito, da bi podporno nalogo opravil tisti član ekipe, ki je razvil problematično dodelavo, vendar bi s tem znanje o tej dodelavi ostalo le v domeni tega člana ekipe. Rotiranje zadolžitve za podporne naloge je dolgoročno boljše, saj se s tem zagotovi pretok znanja in povečuje univerzalnost članov ekipe. Univerzalnost se v ekipi B spodbuja tudi pri razvoju s tem, da se razvojne naloge občasno dodelijo tudi osebi, ki področja ne pozna dobro, s čimer jo prisilijo, da razširi svoje znanje.

Taka delitev nalog pa ne ustreza vsem. Produktni vodja ekipe A ugotavlja, da nekateri ljudje enostavno ne funkcionirajo dobro v scrumu. Težko se vključujejo v skupne aktivnosti (na primer v izpopolnjevanje seznama zahtev) in raje delajo na ožjem, dobro poznanem področju. Produktni vodja ekipe A to povezuje z naravo posameznika. Take ljudi je potrebno obravnavati posebej, kar pa lahko potencialno v ekipo vnaša neskladja. Postavlja se vprašanje, ali taki posamezniki sploh sodijo v scrum ekipo ali bi bilo bolje, da delujejo kot zunanji sodelavci zunaj nje.

3.5 Sodelovanje z naročnikom

Poduktni vodja ekipe A pravi, da se je bilo potrebno sodelovanja s projektno ekipo kot internim naročnikom naučiti. Na začetku je bila komunikacija slaba. Ko se je pri razvoju pojavilo vprašanje, je bila prva reakcija, da vse piše v dokumentu, da je to s končno stranko dogovorjeno in zakaj bi spreminjali. Sčasoma pa se je le vzpostavilo zavedanje, da gre za partnerski odnos, in da če razvojna ekipa izpostavi problem, to počne v dobrobit stranke. Popolnoma isti proces poteka tudi pri razvoju za končno stranko - tudi z njo je potrebno vzpostaviti partnerski odnos. Brez scruma oz. sprotne komunikacije, ki jo scrum vzpostavlja, bi razvojna ekipa razvila dodelave po svojem razumevanju dokumenta. Ko pa bi dostavili končni produkt, bi nastal velik problem. Tako situacijo so dejansko imeli, ko je bilo moč dogovore v dokumentu razumeti na dva načina, ki sta se med seboj bistveno razlikovala.

Agilne metodologije zagovarjajo transparentnost, zato predvidevajo, da ima naročnik dostop do vseh izdelkov scruma, tudi seznama zahtev in načrta sprinta. To se je izkazalo kot problematično v primeru ekipe B, ki sočasno opravlja naloge za dve konkurenčni podjetji.

Ločena seznama zahtev in načrta sprinta otežujeta načrtovanje na visokem nivoju in usklajevanje med obema ekipama.

Stranka ABC ima s scrumom problem predvsem zaradi dvotedenskega cikla, saj le-ta pomeni, da traja vsaj dva tedna od izraza zahteve do uvedbe dodelave v produkcijo. Nove marketinške akcije, ki jih uvajajo, nočejo obelodaniti nikomur, niti ekipi, do nekaj dni pred nastopom akcije . Ko je bil projekt še v implementacijski fazi, je marsikdaj prevladala volja stranke in so bile stvari narejene takoj, mimo pravil scruma. Danes, ko vsaka taka zahteva pomeni zahtevek za spremembo, je tega manj. Izkazalo se je tudi, da je ta dvotedenski zamik včasih celo koristen, saj želene dodelave niso bile vedno do konca domišljene. Ko se neko idejo začne analizirati za potrebo implementacije, se marsikdaj izkaže, da ideja ne sodi v naročnikov proces. Stranka ABC je s scrumom pridobila predvsem redno dostavljene funkcionalnosti. Prej se je dostikrat zgodilo, da je bila dodelava razvita, potem pa se je ni dostavilo v produkcijo.

3.6 Scrum in ERP

Implementacije ERP navadno niso visokotehnološki projekti. Fokus implementacijske ekipe tako ni na tehnoloških, temveč na vsebinskih rešitvah. Pomembno je, da se naročnikov proces podpre pravilno in skladno s standardno ERP rešitvijo. Zato se sam ne strinjam z West et al. (2011), ki kot problematično izpostavljajo dejstvo, da pri fazi analize in dizajna ne sodeluje celotna razvojna ekipa. Sicer je res, da analize in dizajna ne dela celotna ekipa, vendar pa so v to vključeni ključni člani ekipe (navadno arhitekt in produktni vodja), kar pomeni, da znanje izvira iz ekipe in potem tudi ostane v ekipi. Menim, da je za uspeh ERP projekta bolj pomemben dobro narejen dizajn kot vključenost ekipe v vse dele razvojnega procesa.

Ena od principov agilnih metodologij je tudi čim zgodnejša priprava delujočega prototipa, na osnovi katerega naročnik validira svoje nadaljnje zahteve. Menim, da pri ERP-jih to ni tako pomembno, saj se implementacija začne na podlagi že delujočega produkta. Analizo lahko tako izvajaš na obstoječi instalaciji v testnem okolju, dodelave pa so manjše in lažje razumljive. Pri razvoju popolnoma nove programske rešitve je drugače, saj začenjaš »iz nule«.

Uporaba agilnih metodologij se mi zdi še posebno smiselna v primeru, ko imamo že vnaprej dogovorjen (modularno) fazni pristop vpeljave ERP sistema. Namesto podprojektov za razvoj posameznega modula se lahko uporabi natančnejše planiranje po sprintih. V prve sprinte se tako vključi modul, ki ga bomo uvedli najprej. Ker je vsak inkrement pripravljen za vpeljavo, se lahko v trenutku, ko so razvite vse dodelave prvega modula, produktni vodja in naročnik odločita za uvedbo tega modula v produkcijo. To ne ustavlja procesa razvoja, je pa od uvedbe prvega modula naprej potrebno načrtovati določen čas za podporo stranki v produkciji.

3.6.1 Testiranje

Testiranje v ERP sistemu je lahko precej mukotrpen proces, saj mora izvajalec testiranja pogosto opraviti veliko korakov, preden pride do faze procesa, ki jo želi testirati. V tem primeru pridejo prav skripte za generiranje naključnih podatkov v kombinaciji s kodo, ki avtomatsko opravi želene korake. Taki avtomatizirani testi dodatno vrednost dobijo na dolgotrajnih projektih, kjer se pogosto delajo spremembe skozi celoten proces. V ekipi B je priprava takih skript že med samim razvojem postala skoraj pravilo. Vodja scruma ekipe B izpostavlja dodatno prednost takih skript, ki se izkaže šele po daljšem času, ko je potrebno ponovno testirati vse scenarije. Če svetovalec v trenutku implementacije še ve, kakšna so pravila za pripravo podatkov, se čez nekaj mesecev ali let tega ne spomni, kar pomeni, da je ročna priprava testnih podatkov veliko težja. Skripte za pripravo testnih podatkov torej močno olajšajo regresijske teste. Je pa potrebno te skripte tudi vzdrževati, kar je načeloma preprosto, vendar se je potrebno na to spomniti in vedeti, katere skripte je potrebno dodelati. Za to ekipa B še nima sistema.

Pomemben vidik je tudi motivacija izvajalca testiranja. Z besedami vodje scruma ekipe B: »Testirati nekaj novega je zanimivo, če pa stalno testiraš isti proces, je tako, kot da bi neprestano pisal isto vrstico kode.«

Ker so razvijalci porazdeljeni po področjih, je potreba po poznavanju in vzdrževanju pravilnih nastavitev celotnega sistema na razvojnih okoljih manjša.

3.7 Splošno zadovoljstvo s scrumom

Po mnenju direktorja AX oddelka je bil bistven cilj uvedbe scruma - plani in ritem - dosežen. Dosegli so predvidljiv cikel razvoja, predvsem pa vedo daleč vnaprej, kaj vse morajo narediti, da lahko pripravijo načrt, kdaj bo kaj narejeno. Kvaliteta narejenega se je dvignila in uvedba scruma je pomagala uspešno zaključiti projekte, ki jih prej niso mogli. Izboljšala se je preglednost, saj je na dnevnem scrumu enostavno ugotoviti, kaj se dela, kdo kaj dela in kdo ima kje težave.

Direktor AX oddelka je dobil vtis, da so ljudje relativno bolj zadovoljni. Tudi tisti, ki scruma ne vidijo kot bistveno pridobitev, niso proti njemu. Glavni očitek, ki pride na plano, je prevelika pogostost sestankov. Produktni vodja ekipe A se z očitkom ne strinja: “Pri scrumu je malo več birokracije, je več občutka, da veliko časa zabiješ na sestankih, vendar mislim, da imajo sestanki svojo težo. Na teh sestankih analiziraš problem in postavljaš dizajn. Ko nekdo dobi to zahtevo v razvoj, je jasno, kaj mora narediti.” Glavno pridobitev pa v scrumu vidijo tisti, ki so v prejšnji organizaciji vodili ekipe. Produktni vodja ekipe A na primer v scrumu precej lažje opravlja svoje naloge, saj ima boljši pregled nad stanjem zahtev, tako tistih v sprintu kot tudi tistih, ki še čakajo na umestitev v sprint.

Stranke so se takemu načinu dela hitro privadile in prilagodilo se je tudi upravljanje izdaj. Ne dogaja se več, da bi se koda v produkcijsko okolje prenašala vsak dan, ampak se točno ve, kdaj bo koda izdana. Ko je izdaja pripravljena, je predana v testiranje. Če so testi uspešni, se jo prenese v produkcijo, če pa testi niso uspešni ali če ni potrebe po takojšnji izdaji, se nadaljuje z naslednjo iteracijo - nov razvoj, novo testiranje in nov prenos v produkcijo.

Ves čas vpeljave sta vladala močna podpora in pritisk s strani vodstva, ki sta zagotovila, da se scrum izvaja in spoštuje. Ta podpora še vedno ne popušča. V teh dveh letih se je močno spremenila tudi kultura oddelka, scrum je postal del miselnosti. Vsi se o sprintih pogovarjajo kot o nekem naravnem, samoumevnem konceptu. Direktor AX oddelka ocenjuje, da ni več možnosti, da bi scrum zvodenel, ostajajo le še možnosti za izboljšave. Zaposleni verjamejo, da je to prava pot. Dosežena je ena ključnih lastnosti visoko zmogljivih ekip, in sicer opolnomočenost. Ko ekipa dobi nalogo, lahko sama pove, kako bo nalogo opravila. Sami so odgovorni, da se to na koncu res naredi. Za odločitve znotraj seznama zahtev odgovarja produktni vodja in direktor AX oddelka se v to ne vpleta. Če se mu zdi kaka odločitev napačna in škodljiva za Adacto, bo to izpostavil, če pa nima takega razloga, bo ekipo pustil delati v miru.

Direktor AX oddelka poudarja, da dandanes že veliko strank daje prednost agilnemu pristopu k projektu ali ga celo zahteva. Način vodenja projekta in razvoja, ki ga uporablja AX oddelek, je zato konkurenčna prednost in je kot taka posebej izpostavljena v procesu predprodajnih in prodajnih aktivnosti.

3.7.1 Vključevanje novih kadrov

Direktor AX oddelka opaža, da novi kadri z vidika pridobivanja znanja zaradi vključenosti v scrum proces hitreje napredujejo. Scrum sam po sebi spodbuja deljenje znanja, saj ekipa dela skupaj in se dnevno pogovarja o tem, kaj počne. Tudi na izpopolnjevanjih seznama zahtev, načrtovanjih sprintov in retrospektivah sprintov sodeluje cela ekipa. To se razlikuje od klasičnih pristopov, kjer analitik razmisli in naredi načrt, ki ga nato preda razvijalcu v izvedbo. V tem primeru nihče drug ne ve, kaj je analitik pripravil in kaj razvijalec dela, saj ni bil vključen v ta proces. Če je začetnik vključen v proces analize in dizajna, lažje poveže niti in hitreje razume tematiko. Tudi prikaz delovanja dodelav, ki bi moral biti en od elementov pregleda sprinta, dejansko predstavlja deljenje znanja. Začetnike še vedno vključijo v mentorski program, vendar je potreba po individualni predstavitvi določenega področja s strani mentorja manjša, saj se take predstavitve implicitno dogajajo znotraj scrum procesa. Tudi dokumentacija, na primer testni primeri, so po scrum principih vedno na voljo, saj so eden od kriterijev definicije zaključenosti. To v klasičnem procesu ni vedno nujno.

3.7.2 Identificirane težave

Vse še ne teče točno tako, kakor so si zamislili. Problema, ki ju izpostavlja direktor AX oddelka in s katerima se trenutno največ ukvarjajo, sta pomanjkanje predanosti in pomanjkanje kakovosti. Pomanjkanje predanosti se kaže v neuspešnih sprintih, ko ekipe ne dostavijo inkrementa, za katerega so se obvezale, da bo dostavljen. Pomanjkanje kakovosti pa je večinoma posledica slabo testirane kode. Prevečkrat se zgodi, da se že razvite dodelave predelujejo, ker se v produkciji izkažejo napake. Premalo se tudi zahteva testiranje s strani naročnika, zato le-ta ne ve, kaj dobi. Ko se taka dodelava da v produkcijo in pride do problema, je potrebno to popraviti, kar zruši naslednji sprint.

Izboljšanje kontrol kakovosti je tako ena ključnih nalog letošnjega leta. Da bi dvignili nivo upoštevanja scruma in izpolnjevanja zavez, bodo spet uvedli vlogo trenerja scruma. Gre za osebo, ki se udeležuje dnevnega scruma, retrospektive ali drugega scrum dogodka ekipe in svetuje, kako scrum proces izvajati še bolje. To vlogo so ob uvedbi že imeli, vendar je oseba, ki jo je izvajala, prevzela vodstvo ene od ekip v podjetju, nihče pa ni vloge trenerja prevzel od nje. Trener scruma bo imel tudi moč uvajanja sprememb.

Vodja scruma ekipe B izpostavlja še problem, da se kljub trudu vodstva scrum v organizaciji včasih ne upošteva. Dogaja se, da ob odsotnosti vodje scruma oz. produktnega vodje projektni vodja stopi neposredno do razvijalca in mu naloži neko nalogo. Če razvijalec ne zna sam oceniti prioritete naloge in zavrniti tak pristop, se lahko zgodi, da zaradi take ad hoc naloge trpi bistven razvoj. V preteklosti se je že zgodilo, da je bilo zaradi takega ravnanja kljub natančnemu planiranju za tri mesece naprej potrebno delati nadure in celo prestaviti prehod v produkcijo. Je pa zavedanje o načinu dela v organizaciji vedno večje in se podobne stvari dogajajo vedno redkeje. Organizacija poskuša dobre prakse okrepiti tudi z uvedbo kompetenčnega modela, v katerem so definirane ključne vrednote zaposlenih.

3.8 Predlogi za izboljšave

Scrum je v AX oddelku že dobro vpeljan in sprejet. V dveh letih uporabe je že precej dobro znano, kateri pristopi delujejo in kateri ne. Jasno je, da zaradi specifik projektov popolna agilnost ni mogoča. Vseeno pa bi predlagal, da se k njej teži, kolikor je le mogoče. Od uvedbe je scrum veliko doprinesel k boljši organiziranosti in učinkovitosti scrum ekip. Mislim, da bi morala biti naslednja stopnja usvojitve scruma še boljša komunikacija z naročnikom. Kot izpostavljajo Carvalho et al. (2010), je le-ta bistvenega pomena za pravilno identifikacijo potrebnih dodelav. Kot pravi ena od vrednot Manifest agilnega razvoja programske opreme, je bistveno zmanjšati količino nepotrebnega dela. Pri implementaciji ERP sistema je to najlažje doseči tako, da naročnik dovolj zgodaj prepozna, da je njegova zahteva po prilagoditvi nepotrebna - bodisi ni dovolj domišljena, bodisi funkcionalnost oz. njena alternativa že obstaja. Da pa bi to dosegli, je potrebno naročniku čim pogosteje dostavljati delujoče inkremente. Korak naprej v primerjavi s slapovno metodologijo je že narejen z

mesečnimi CPR delavnicami, ki pa so neinteraktivne in od naročnika zahtevajo dobro predstavo o delovanju sistema, ki ga v praksi še nikoli ni uporabljal. Predlagam, da se ekipe in vodstvo na prihodnjih projektih potrudijo naročnika prepričati v pomembnost zgodnjega spoznavanja sistema ter sprotnega testiranja dodelav. Seveda pa to pomeni, da je potrebno dovolj zgodaj zagotoviti infrastrukturo za testno instalacijo.

V primeru dveh seznamov zahtev ekipe B bi predlagal, da se najde tehnična rešitev, ki bi omogočala vodenje obeh projektov znotraj enega seznama zahtev. Ena od možnih rešitev je uporaba ločevalnega atributa na postavkah, na podlagi katere se določi pripadnost produktu. Obenem pa bi bilo potrebno strankama omejiti vpogled v seznam zahtev in načrt sprinta, tako da bi videli samo svoje postavke. Manj elegantna rešitev bi bila tretji seznam zahtev, v katerega bi se zrcalile zahteve iz seznamov zahtev obeh produktov. Obe rešitvi bi olajšali načrtovanje sprintov ter izboljšali preglednost nad seznamom zahtev in načrtom sprinta.

4 ANALIZA PRIMERNOSTI VPELJAVE METODOLOGIJE SCRUM V DYNAMICS NAV ODDELEK

4.1 Celovita informacijska rešitev Microsoft Dynamics NAV

Prvo različico Dynamics NAV (ali krajše samo NAV) je leta 1987 razvilo dansko podjetje PC&C A/S pod imenom Navision. Uspeh rešitve je botroval spremembi imena podjetja v Navision Software A/S. Najprej se je Navision prodajal predvsem v skandinavskih državah, od različice 3.0 naprej pa je bila rešitev na voljo tudi v drugih evropskih državah. Leta 2000 se je podjetje Navision Software A/S združilo s podjetjem Damgaard A/S in se leta 2001 preimenovalo v Navision A/S. Združeno podjetje je bilo leta 2002 prevzeto s strani podjetja Microsoft. Produkt je doživel več sprememb imena - Navision Financials, Navision Attain, Microsoft Business solutions Navision Edition in končno Microsoft Dynamics NAV. Zadnja izdana različica je Microsoft Dynamics NAV 2016 (Lorente & Lorente, 2013).

Do različice Dynamics NAV 2009 je bila arhitektura dvonivojska, nove različice pa so tronivojske. Rešitev je tako sestavljena iz:

- Microsoft SQL Server relacijske podatkovne baze za shranjevanje podatkov,
- aplikacijskega strežnika, ki izvaja poslovno logiko in
- klientskih aplikacij (Lorente & Lorente, 2013).

Različice Dynamics NAV 2013 do 2016 so osnovni Windows klientski aplikaciji dodale še:

- spletno aplikacijo,
- Microsoft SharePoint aplikacijo,
- tablično aplikacijo in

- telefonsko aplikacijo (Microsoft, 2015).

Dynamics NAV je ERP, fokusiran na vloge. Sistem je osnovan okoli posameznikov v organizaciji, njihovih vlog in opravil, ki jih izvajajo. Uporabniki, ki prevzemajo različne vloge, bodo imeli drugačen pogled na sistem. Vsak bo videl funkcionalnosti, ki jih potrebuje za izvajanje svojih dnevnih nalog (Lorente & Lorente, 2013).

Dynamics NAV pokriva naslednja funkcionalna področja v organizaciji:

- vodenje financ – knjigovodstvo,
- prodajo in trženje,
- nabavo,
- skladiščne procese,
- proizvodnjo,
- projekte,
- načrtovanje virov,
- servis in
- upravljanje s človeškimi viri (Lorente & Lorente, 2013).

Dynamics NAV je mogoče s pomočjo integriranega razvojnega okolja prilagoditi oziroma mu dodati nove funkcionalnosti.

4.2 Predstavitev Dynamics NAV oddelka

O Dynamics NAV oddelku sem se pogovarjal z direktorjem oddelka. Direktor NAV oddelka je diplomirani ekonomist. Z ERP sistemi ima deset let delovnih izkušenj. Začel je kot projektni vodja, po nekaj uspešnih projektih je prevzel vlogo vodje oddelka za podporo strankam in to vlogo opravljal štiri leta. Trenutno nastopa v vlogi direktorja Dynamics NAV oddelka, obenem pa je aktivno vključen v implementacijske projekte.

Oddelek NAV sestavlja osem ekip s povprečno sedmimi zaposlenimi. Ekipe so načeloma namenjene podpori obstoječim strankam, ki jih je v času intervjuja že preko sto. Razmerje med opravljenimi urami za podporo strankam in opravljenimi urami za implementacije je na nivoju oddelka 60:40. Pri tem nekateri ljudje izvajajo zgolj podporo in se nikoli ne vključujejo v implementacije, na primer vodja ekipe ter navadno še en svetovalec in en razvijalec. Približno petnajst od petinšestdesetih ljudi pa se ukvarja samo z implementacijami. Gre za najbolj izkušene ljudi, ki dobro poznajo tako procese kot standardno rešitev, obenem pa so sposobni inovirati.

Ko se začne nov projekt, se sestavi implementacijska ekipa. Osnova te ekipe so ljudje, ki delajo samo na implementacijah, v ekipo pa se po potrebi vključi tudi posameznike iz

različnih podpornih ekip. Profil vključenih zaposlenih je odvisen od potreb. Izobraževanje uporabnikov lahko na primer izvaja manj izkušen svetovalec, za zahteven razvoj pa vključijo osebo, ki ima potrebno znanje in izkušnje. Po zaključku projekta se projektna ekipa razpusti in ljudje se vrnejo v svoje ekipe. Ko so ljudje vključeni v projektno ekipo, navadno še vedno 20 do 30 odstotkov svojega časa opravljajo delo za matično ekipo.

4.2.1 Projekti

V tem trenutku sočasno poteka šest različno velikih implementacijskih projektov. Na enem od projektov dela samo ena oseba, v nekaj projektov je vključenih dva do štiri ljudi, v implementacijski ekipi največjega projekta pa je deset ljudi. Preprostih projektov je malo, večinoma so taki, na katerih dela več kot pet ljudi in trajajo približno eno leto. To se je spremenilo v zadnjih nekaj letih. Adacta je namreč dobila sloves implementatorja bolj kompleksnih projektov, medtem ko konkurenca pridobiva enostavnejše projekte.

Projekti so prodani za fiksno ceno na podlagi grobe ocene, pridobljene s površno diagnostiko pred analizo. Podrobna analiza se opravi šele po začetku projekta.

4.2.2 Uporabljena metodologija razvoja

NAV oddelek uporablja Microsoft Sure Step metodologijo, ki temelji na slapovni metodologiji (Shankar & Bellefroid, 2011). Implementacija se začne z analizo, pri kateri sodelujejo arhitekt in svetovalci. Nato se napravi dizajn in se začne razvijati. Razvoju sledi testiranje in prehod v produkcijo.

Projekti so organizirani tako, da so razdeljeni na področja, ki imajo svoje nosilce. To so svetovalci, ki morajo v popolnosti razumeti procese tega področja in so zanj odgovorni. Pri tem ne gre za funkcionalna področja, ampak na primer za:

- proces od nabave blaga do prodaje,
- področje reklamacij in
- tehnično področje integracije z določenim zunanjim sistemom.

Nosilci področij so tisti, ki opravijo analizo ter nato z arhitektom načrtujejo rešitev, ki gre potem v razvoj. Nosilci so tisti, ki poganjajo projekt. Sodelovanje med nosilci nastopi samo tam, kjer se procesi prepletajo. Usklajevanje znotraj ekipe je naloga arhitekta. On je tisti, ki skrbi, da se funkcionalnosti med seboj skladajo, da se koda ne podvaja itd.

Prehod v fazo testiranja se začne pred zaključkom razvoja, vendar pa redko preden je razvitih vsaj 70 odstotkov dodelav. Zahtevki za spremembo, identificirani med testiranjem, se razvijajo vzporedno z razvojem preostalih dodelav.

4.2.3 Sodelovanje z naročnikom

Sodelovanja z naročnikom je veliko v fazi analize, ko se analizirajo procesi v podjetju in zajemajo zahteve. Naročnik tudi potrdi pripravljen dizajn. Verifikacija in validacija napravljenega s strani naročnika se med razvojem izvaja v majhni meri. Večina testiranja je izvedena v testni fazi na koncu projekta. Če vzamemo za primer Projekt S, ki je trenutno z desetimi člani implementacijske ekipe največji projekt oddelka, se je sodelovanje z naročnikom začelo šele, ko je bilo razvitih 80 odstotkov dodelav. Takrat se je ekipa preselila na lokacijo naročnika in začela z validacijo rešitve. Izkazalo se je, da so procesi kompleksnejši, kot je pokazala analiza, zato se zdaj ta validacija izvaja že tri mesece. Obenem se razvija še nerazvite dodelave ter spremembe, vezane na med implementacijo odkrite probleme. Teh je bilo kar nekaj predvsem na ključnih točkah v procesih, na primer pri integraciji z aplikacijo za upravljanje skladišča. Izkazalo se je tudi, da naročnik šele ob validaciji narejenega dejansko dobi idejo, kaj resnično želi.

4.2.4 Management sprememb

V kolikor vodja projekta oceni, da so na novo identificirane spremembe pomembne za poslovanje naročnika, se mu poskuša ustreči in spremembe implementirati, marsikdaj tudi znotraj obstoječega proračuna.

4.3 Analiza obstoječe metodologije

Direktor NAV oddelka ugotavlja problem, da se projekti ne vodijo dovolj dobro oz. v skladu z slapovno metodologijo. Prevečkrat se zgodi, da pride tekom projekta do presenečenj, torej do nekih novih zahtev, ki niso bile identificirane v fazi analize. Preveč se tudi dela ad hoc, torej brez podrobnih dizajnov.

Pri analizi nikoli ni mogoče popisati zahtev v popolnosti, saj implementator ne pozna dobro problemskega področja, naročnik pa ne ve točno, kaj želi oz. potrebuje. Ocena direktorja NAV oddelka je, da je le 60 odstotkov zahtev, podanih v fazi analize na začetku projekta, dejansko pravih. Ta delež pa je še manjši, če gre za dolgotrajen projekt. Iz analize lahko torej nastane polovična rešitev, zato je potrebno dizajn prilagajati tekom projekta.

Problem kompleksnih in posledično dolgih projektov je tudi ta, da tekom projekta pri naročniku pogosto pride do spremembe procesov in menjave kadrov. Bolj kot je organizacija pri naročniku nestabilna, slabše je to za projekt. Naročnik implementacijskega projekta S na primer med projektom izvaja reorganizacijo in kadrovske menjave, že v osnovi pa ni dobro organiziran. To se pozna na projektu, ki se izvaja podobno kaotično. Velikost projekta in urejenost izvedbe sta zelo povezana. Pri manjših projektih je hitreje in lažje narediti analizo,

saj je procesov manj oz. so le-ti preprostejši. Kljub temu pa tudi pri teh manjših strankah tekom projekta pride do novih zahtev oz. želja, ki v analizi niso bile identificirane.

Po mnenju direktorja NAV oddelka je slapovna metodologija pravi pristop pri manjših projektih oz. projektih z malo neznankami. Pri takih projektih se opredeli, kaj je standardni paket, nato pa se z analizo ugotovi odstopanja procesov podjetja od obstoječe rešitve. Z razvojem dodelav za ta odstopanja si praktično opravil 90 do 95 odstotkov dela. Če pa je projekt kompleksen oziroma je vanj vključenih veliko ljudi, je potrebno imeti bolj natančen nadzor nad tem, kaj se dogaja na projektu. In v tem pogledu imajo z obstoječo metodologijo probleme. Pri določenih projektih se recimo dogaja, da projektni vodje ne znajo povedati, kaj je potrebno ta teden narediti. Problem torej ni samo v tem, da projektni plani ne odražajo dejanskega stanja, ampak da je dejansko stanje težko ugotoviti. Tako se dogaja, da je v zadnjih 20 odstotkih obdobja trajanja projekta potrebno razviti 80 odstotkov funkcionalnosti.

4.3.1 Management sprememb

Slapovni pristop ni učinkovit, če se na začetku jasno ne opredeli, kaj je obseg projekta - katere funkcionalnosti pridejo z osnovnim produktom in kaj je potrebno dodelati. Direktor NAV oddelka meni, da bi se moral projekt začeti s predstavitvijo produkta in njegovih zmogljivosti. Vse spremembe bi se morale obravnavati kot zahtevki za spremembo. V praksi pa se to ne dogaja. Eden od razlogov za to je slovenski trg, ki ga direktor NAV oddelka ocenjuje kot zelo specifičnega. Prevladuje namreč praksa, da stranke pričakujejo ponudbo, preden sploh specificirajo predmet implementacije. Situacija na trgu je taka, da je marsikateri implementator pripravljen na tak pristop, da le dobi projekt, zato se mora prilagoditi tudi Adacta. Podobno je bilo na projektu S, pri čemer se je po oddaji ponudbe napravila še revizija projekta. Le-ta je pokazala, da za tako plačilo projekta vseeno ne bo moč izpeljati, tako da je bilo naročniku razloženo, da lahko odstopi od projekta ali pa pristane na višji znesek. Kljub naknadnemu dogovoru je še vedno ostalo veliko neznank, tako da se je v projekt šlo z zavedanjem, da je Adacta na spodnji meji dobičkonosnosti projekta.

Naročniki v Sloveniji pogosto ne sprejmejo dodatnih stroškov za zahteve po spremembah, identificirane tekom razvoja. Izkušnje direktorja NAV oddelka z naročniki iz tujine so drugačne - tam se dodatne stroške razume kot normalne. Naročnik sam omejuje svoje ljudi pri postavljanju dodatnih zahtev, če pa se le-te ocenijo kot potrebne, so jih pripravljeni tudi plačati.

Direktor NAV oddelka pravi, da je pri velikih projektih lažje priti do dodatnega proračuna. Razlogi za to so v večji pomembnosti delujoče programske opreme in manjšem vložku v implementacijo v primerjavi s prometom podjetja. Eden od razlogov je tudi mednarodno udejstvovanje teh podjetij, zaradi katerega imajo drugačen pogled na take projekte. Še vedno pa obstajajo tudi naročniki, ki niso tako razumevajoči, pač pa zaradi svoje velikosti čutijo,

da so pomembne stranke, zato bolj pritiskajo na Adacto in izsilijo boljše pogoje (brezplačne dodelave).

4.3.2 Zamude na projektih

Manjši projekti načeloma ne zamujajo. Večji projekti pa zamujajo takrat, ko je tematsko področje slabo poznano. Če je projekt po procesih in področju podoben kakšnemu od že končanih projektov, je namreč moč potreben trud in čas precej dobro oceniti, ker veš, kaj te čaka. Če pa je področje projekta povsem novo, je tveganje za napačno oceno bistveno večje. Časovne zamude so navadno do 20 odstotkov prvotno ocenjenega časa. Kljub temu so naročniki večinoma zadovoljni, saj razumejo, da do zamude pride tudi zaradi naknadnih zahtevkov po spremembah.

Načelo Adacte je, da je zadovoljstvo stranke na prvem mestu, kar je tudi povzeto v sloganu »*Consider IT done*«. Ker polovične rešitve niso sprejemljive, se sicer vzpostavlja pritisk na stranko, naj omeji svoje dodatne zahteve, vendar pa se na koncu vedno poskrbi, da je stranka zadovoljna. Zamude, do katerih včasih pride, gredo pogosto tudi na račun dobičkonosnosti projekta. Adacta namreč razmišlja dolgoročno. Ko vstopi v neko panogo, ne pričakuje, da bodo naredili dobiček s prvim projektom, pač pa da bodo dobičkonosne naslednje implementacije panožne rešitve v Sloveniji ali v tujini.

4.4 Scrum

Z direktorjem NAV oddelka sem govoril o metodologiji scrum in možnostih o vpeljavi le-te v NAV oddelek. Direktor NAV oddelka s scrumom nima praktičnih izkušenj, ve pa, kako se scrum uporablja v AX oddelku. Ena od pozitivnih posledic, ki jih opaža, je definitivno to, da te scrum prisili v razmišljanje, kaj boš počel v naslednjem sprintu, torej v vzpostavljanje prioritet. Ker se točno ve, kaj so prioritete in kaj mora v nekem obdobju posameznik narediti, so ljudje bolj pod nadzorom v smislu učinkovitosti. Le-to lahko na koncu sprinta tudi oceniš. Ljudem tudi ne zmanjka inercije, saj je iz seznama zahtev transparentno, kaj vse je še potrebno narediti.

Ni pa povsem prepričan, da je moč s scrumom priti do zaključka znotraj omejitev projekta. Ne glede na uporabljeno metodologijo ima projekt enake cilje. Rešitev je potrebno razviti, jo testirati, dati v validacijo stranki, stranko naučiti to rešitev uporabljati in rešitev uvesti v produkcijo. Metodologija ti določa, na kakšen način kontroliraš vmesne točke, torej kako dobro veš, kaj ljudje delajo, in kako si sposoben postavljati prave prioritete na mikro nivoju. Direktor NAV oddelka domneva, da je scrum pristop v tem pogledu boljši kot slapovni. Ni pa uporaba scruma sama po sebi zagotovilo, da bo projekt uspešen, če si zgrešil bistvo problemske domene. Direktor NAV oddelka je podal naslednjo analogijo: “Če tečeš v napačno smer, ne boš prišel do cilja tako ali drugače. Scrum ti zagotavlja ograjo, ob kateri tečeš, slapovni pristop pa te ograje ne zagotavlja in marsikdo skrene s poti.” “Ograjo” pri

scrumu predstavljata sprint in načrt sprinta. Če se dobro dogovoriš, kaj bo kdo delal, če postaviš prave prioritete, je učinek ekipe večji.

4.4.1 Sodelovanje s strankami

Scrum predvideva, da pri postavljanju v poglavju 4.4 omenjene »ograje« sodeluje tudi naročnik v obliki sprotne validacije in verifikacije tekom razvoja. Direktor NAV oddelka se strinja s koristmi takega pristopa, če je naročnik na pravem nivoju. Če nosilci procesov na strani naročnika razumejo, kako potekajo implementacije, in so sposobni razumeti, kako deluje standardna rešitev in kako naj bi se jo prilagodilo, potem lahko že zgodaj v procesu podajo konstruktivne pripombe in predloge. Če pa nosilci procesov razumejo in so sposobni komentirati le rešitev, ki že odraža njihov proces v kombinaciji z njihovimi podatki, potem jih je nesmiselno vključevati v fazi razvoja, saj takrat še niso sposobni dati uporabne povratne informacije. Tudi če naročnik izvaja teste, jih izvaja površno in se na povratne informacije ne moreš zanesti. V takem primeru pogosto pride do zahtev po prilagoditvi novega sistema v taki meri, da bo deloval popolnoma enako kot prejšnji, saj le tak način delovanja poznajo in razumejo. Strank s ključnimi uporabniki na tako visokem nivoju je malo, po oceni direktorja NAV oddelka le 5 odstotkov. Najpogosteje je arhitekt na strani naročnika vodja IT oddelka, kar je sicer dovolj dobro, da projekt pripelješ do 60 ali 70 odstotkov zaključenosti. Kasneje pa se navadno izkaže, da ta oseba ni poznala detajlov, zato je razvoj ponekod že skrenil v napačno smer.

Izkušnje direktorja NAV oddelka kažejo, da sta obseg in kakovost sodelovanja s stranko zelo odvisna tudi od stranke same. Pomembno je, kako vodstvo naročnika dojema projekt. Če je vodstvo zelo vključeno in angažirano v projekt, se le-ta odvija bolje in ljudje so bolj zavzeti, če pa vodstvo ni vključeno, se projekt dojema kot projekt IT oddelka, tako da je težko priti do ključnih ljudi.

Šok, ki ga stranke doživijo, ko prvič začnejo uporabljati novo rešitev, je odvisen od tega, kaj so uporabljali prej. NAV je strankam načeloma zelo všeč. Če pa je imel naročnik pred tem po meri narejeno rešitev, uporabniki pogosto izpostavljajo, da so imeli prej nekatere stvari bolje urejene. Pri tem ne razumejo, da je njihova rešitev rezultat desetih let razvoja, medtem ko je nova rešitev rezultat nekajmesečnega projekta. Ta pričakovanja morajo usklajevati vodstvo, IT oddelek in ključni uporabniki naročnika.

4.5 Načrti za prihodnost

Direktor NAV oddelka ocenjuje, da je bila obstoječa organizacija v fazi rasti smiselna in morda je do neke mere še vedno. Pri takem deležu podpore strankam je potrebno imeti stabilno ekipo, ki se s tem ukvarja. Opaža pa, da se ljudje nasičijo dela v podpori, zaidejo v pasiven način dela, ne pridobivajo novih znanj in posledično nazadujejo. Projekti bistveno vplivajo na razvoj posameznikov. Bolj izkušeni člani ekipe nastopajo kot mentorji in ljudje

se veliko naučijo. Pride do dviga vsebinskega in tehničnega znanja ter do spoznavanja metodologije, kar vse vpliva na dvig samozavesti in osebnostno rast. Zato je potrebno zaposlene na projektih rotirati. Nekako na tri leta se na oddelku menja organizacija, vzpostavijo se nove ekipe, katerih sestava se bo spremenila. Ravno zdaj poteka taka reorganizacija, ki bo vzpostavila dve ekipi, ki se bosta ukvarjali pretežno s podporo, ostale ekipe pa bodo implementacijske. Obenem se bo vzpostavil tako imenovani kompetenčni center, to je ekipa, sestavljena iz ključnih zaposlenih oddelka, ki se bodo vključevali v prodajne aktivnosti, vodenje projektov, ukvarjanje s strankami, itd.

Dobro je, če so projektne ekipe dokaj stalne, ker se zaposleni spoznajo in vedo, kako funkcionirajo, zato je lahko projekt bolj učinkovit. Do sedaj so bile projektne ekipe vedno znova sestavljene iz drugih članov, vedno znova so se vzpostavljali odnosi, zato niso bili tako učinkoviti. Zato bi direktor NAV oddelka rad imel več manjših ekip, ki v celoti izpeljejo nek projekt. To pa se lahko zagotovi le tako, da člani implementacijskih ekip niso obremenjeni s podporo strankam. Dejstvo pa je, da se bodo skozi čas nabrali končani projekti, ki jih ne bo moč predati ekipam za podporo strankam in jih bodo morale podpirati implementacijske ekipe. Čez tri leta bo verjetno spet potrebna reorganizacija, da se bo postavitev ekip prilagodila tem novim projektom.

Direktor NAV oddelka je o scrumu že razmišljal, in sicer v kombinaciji s slapovnim pristopom. Še vedno si je treba vzeti čas za grobi dizajn. Če gre za projekt za fiksno ceno, analizo izvajaš le na večjih projektih. Na manjših projektih analiza odpre poplavo zahtev po prilagoditvah, česar nikakor ne moreš pokriti v okviru proračuna. Na projektih, plačanih po dejanski porabi časa in materiala, ti analiza natančneje definira potrebe. Scrum bi, tako kot AX oddelek, vpeljal v fazo razvoja, da bi povečal učinkovitost in nadzor. Ko bi razvoj prišel do točke testiranja, bi spet v okviru scruma določil prioritete zahtevkov, ki nastanejo tekom testiranja. Paziti je potrebno, da je v implementacijski ekipi dovolj veliko število ljudi, da ti zahtevki ne motijo razvoja še nerazvitih osnovnih funkcionalnosti. Če se to zgodi, je lahko namreč ogrožen celoten projekt. Razvoj osnovnih funkcionalnosti mora biti v večji meri zaključen, preden se lotiš dodatnih zahtev. Dobro je, da so razvijalci in morda celo arhitekti ločeni po področjih, sicer lahko pride do zmede.

Scrum bi uvedli v implementacijske ekipe. Želja je, da bi bile ekipe čim bolj stalne, in tako se tudi gradijo - poskuša se zajeti vsa znanja, ki so potrebna za nek projekt. Tako se vključi nekoga, ki pozna skladiščne procese, drugega, ki pozna distribucijo, tretjega, ki pozna proizvodnjo, in tako naprej. Popolnoma stalno ekipo je težko sestaviti, vendar je cilj imeti vsaj 80 odstotkov ekipe konstantne. Tem stalnim članom pa se potem po potrebi priključijo bodisi manj izkušeni kadri, ki opravijo dobro specificirane ali bolj tehnične naloge (na primer integracijo z zunanjim sistemom), ali pa ključne osebe na oddelku, ki delujejo kot pospeševalci razvoja (kompetenčni center). Gre za vodje projekta ali arhitekte. Svetovalcev ne moreš deliti med projekti, ker se morajo posvetiti enemu projektu in ga zaključiti. Teh t.i.

arhitektov rešitev (angl. *solution architect*) je malo, zato jih ne moreš vezati na eno ekipo, saj jih potrebujejo vse ekipe.

Za stalnost ekip je potrebno imeti tudi močno prodajo in dovolj projektov, ki čakajo na izvedbo, da se ekipi zagotovi nov projekt takoj po zaključku prejšnjega. Pri tem pa je dodatna komplikacija tudi različna velikost projektov. Če obstaja več manjših ekip, odpre pa se velik projekt, je te ekipe nujno združiti oz. začasno priključiti nove člane. Čeprav bi bil isti projekt lahko opravljen z manjšo ekipo v dvakrat daljšem času, stranke navadno niso pripravljene čakati tako dolgo. Če želiš dobiti projekt, moraš z razvojem začeti čim prej in ga končati v doglednem času.

Direktor NAV oddelka trenutno od scruma pričakuje izključno transparentnost in boljšo kontrolo. Rast posameznikov bo prišla od samega dela na projektu. Ljudje napredujejo, ker imajo okoli sebe bolj izkušene ljudi, ki so zaradi projekta prisiljeni svoje znanje širiti. Rast poteka na področju, na katerem imaš problem, saj si tam odprt za nove informacije, za novo znanje. Izkazalo se je, da razna izobraževanja in mentorstva »na zalogo« ne obrodijo sadov. Če pa imaš problem, ki ti ga nekdo pomaga rešiti, si rešitev zapomniš in se iz nje nekaj naučiš.

5 PRIMERJAVA ODDELKOV NAV IN AX

5.1 Podobnosti in razlike med oddelkoma

V tretjem poglavju sem opravil analizo vpeljave scruma v AX oddelek in ugotovil, da z nastalo hibridno metodologijo ekipe uspešno obvladujejo posebnosti implementacij ERP sistemov. Z vpeljavo scruma so uredili proces razvoja in ga naredili bolj preglednega. Komunikacija z naročnikom sicer ne poteka tako pogosto, kot priporočajo agilne metodologije, vendar je še vedno redna in omogoča odkrivanje in odpravo napak v dizajnu že med samim razvojem.

Analiza NAV oddelka, ki sem jo opravil v četrtem poglavju, nakazuje, da se NAV oddelek srečuje s podobnimi težavami, kot se je srečeval tudi AX oddelek leta 2013. Stanje največjega in najpomembnejšega projekta oddelka ni pregledno, prišlo je do zamude in za ureditev stanja bo potrebno veliko truda. »Bolečine«, ki jih doživljata implementacijska ekipa in vodstvo oddelka, so odlična iztočnica za premislek o učinkovitosti uporabljene metodologije in možnostih vpeljave druge, boljše.

Preden lahko predlagam NAV oddelku, naj vpelje enako metodologijo kot AX oddelek, je potrebno oddelka primerjati in ugotoviti odstopanja, ki bi lahko vplivala na uspeh predlagane metodologije.

Razlike med oddelkoma izhajata iz razlik med ERP sistemoma, katerih implementacije izvajata. Dynamics AX je namenjen srednje velikim in velikim, Dynamics NAV pa majhnim in srednje velikim organizacijam. Temu primerno je AX vsebinsko in tehnično kompleksnejši, implementacije pa so dražje in potekajo dlje časa. Tako ne preseneča, da se za implementacijo AX odločajo večja podjetja kot za implementacijo NAV. Taka podjetja imajo navadno lasten IT oddelek z zaposlenimi arhitekti, kar botruje drugačnim implementacijam, kot če bi bili na strani naročnika le uporabniki. Kot je izpostavil direktor NAV oddelka, je od naročnika med razvojem težko dobiti uporabne povratne informacije, če ključni uporabniki niso sposobni abstraktnega razmišljanja, da bi razumeli, kako bo sistem videti po končanem razvoju. Dejstvo, da ima NAV oddelek več manjših naročnikov, povečuje tveganje, da scrum ne bo prinesel želenih koristi, ki izhajajo iz redne komunikacije z naročnikom.

Kompleksnost ERP vpliva tudi na čas, ki je potreben, da programer ali svetovalec začetnik pridobi dovolj izkušenj, da lahko samostojno opravlja naloge. Izkušnje direktorja AX oddelka kažejo, da NAV razvijalec postane samostojen z enim letom delovnih izkušenj, medtem ko postane AX razvijalec samostojen šele po treh letih delovnih izkušenj. NAV razvijalec potrebuje tudi manj časa, da pridobi dovolj izkušenj in znanja za vlogo arhitekta rešitve. To lahko olajša sestavo večfunkcijske in samozadostne scrum ekipe.

Pomemben vpliv na kadre imajo tudi razlike v uporabljeni tehnologiji. AX razvojno okolje je tehnično zahtevnejše, zato kot razvijalce običajno zahteva izučene programerje. Po drugi strani ima NAV enostavnejše tehnologije in preprostejše razvojno okolje, zato se ga pogosto priučijo tudi svetovalci brez programerske izobrazbe. Temu primerno je razpoložljivih NAV razvijalcev precej, vendar njihova kvaliteta ni sama po sebi umevna. Sklepam, da je iz takih zaposlenih težje sestaviti visoko sposobno in motivirano scrum ekipo.

Na delavnicah, na katerih so ključne osebe obeh oddelkov z zunanjim svetovalcem razvijale kompetenčni model, se je izkazalo, da so problemi, ki jih ima NAV ekipa, v nekaterih pogledih precej različni od problemov AX ekipe. En od takih problemov je predanost, s katero v AX oddelku ni večjih problemov, v NAV oddelku pa se občuti premalo ekipnega duha in zavedanja, da gre za skupinsko prizadevanje. Razlog za to je verjetno v dejstvu, da so bili projekti na NAV oddelku prej manjši, zato jih je bilo mogoče delati ad hoc. Zdaj pa so projekti vedno kompleksnejši, funkcionalnosti so vedno bolj integrirane, zato na primer svetovalci ne morejo delati vsak zase.

Nenazadnje je razlika med oddelkoma tudi ta, da je vodja AX oddelka po izobrazbi programer, vodja NAV oddelka pa ekonomist. To lahko botruje bistveno drugačnemu pogledu na vodenje projektov in metodologijo razvoja.

5.2 Priporočila NAV oddelku

Na podlagi opravljene analize vpeljave scruma v AX oddelek, analize NAV oddelka ter primerjave med obema oddelkoma sem prišel do sklepa, da je vpeljava scruma v NAV oddelek izvedljiva in primerna, seveda s prilagoditvami. Smiselna je vpeljava podobne različice hibridne metodologije, kot jo uporablja AX oddelek, saj je že prilagojena specifikam implementacij ERP sistemov. Je pa potrebno posamezne vidike metodologije sproti ocenjevati in jih po potrebi spreminjati.

Predlagam, naj bo vpeljava sočasna z organizacijskimi spremembami, ki jih načrtuje direktor NAV oddelka. Njegovi načrti za bolj stalne ekipe se skladajo z idejo agilnih metodologij. Ne strinjam pa se popolnoma z njegovim razmišljanjem o rasti posameznikov. Njegova opazka, da ljudje napredujejo že zgolj s sodelovanjem pri projektu, do neke mere drži v primeru, ko v projekt vključiš člana ekipe za podporo strankam. Ta posameznik je iztrgan iz rutine in prisiljen usvojiti nova znanja, da lahko opravi svojo nalogo. Vseeno pa je njegova rast in pravzaprav rast vseh članov ekipe omejena na področje naloge, ki jo dobi v izvedbo. Če je na primer razvijalec odgovoren za dodelave na področju skladiščnega poslovanja, o tej tematiki zve veliko novega in njegovo poznavanje tega področja se poveča. Nič pa ne napreduje na področju finančnega poslovanja, saj v razvoj tega področja ni bil vključen. Scrum v tem pogledu članom ekipe ponuja veliko več, saj z vključenostjo v izpopolnjevanje seznama zahtev, preglede in načrtovanje sprintov, preglede kode in druge scrum dogodke rastejo na vseh področjih projekta.

Zavedam se, da je neizogibno, da se implementacijske ekipe prilagajajo projektom, vendar bi moral oddelek težiti k čim večji stalnosti ekip. Predlagam vzpostavitev vsaj ene konstantne scrum ekipe, ki bi bila specializirana za izvajanje najtežjih projektov. Najbolj smotrno bi bilo, da ta glavna scrum ekipa postane ekipa, ki trenutno dela na implementacijskem projektu S. Gre za ekipo, ki skupaj dela že eno leto in ima že vzpostavljene interakcije med člani, tako da ji bo prihranjena uvodna faza spoznavanja. Smiselnost vpeljave scruma v druge ekipe je potrebno oceniti glede na število članov ekipe ter tip in trajanje projekta. Predlagam, da se scrum vpelje v vse tiste ekipe, ki:

- imajo vsaj štiri člane ter delajo na implementacijskem projektu, ki bo trajal vsaj tri mesece ali
- imajo tri ali manj članov, a jih vodstvo želi razširiti in nadgraditi v stalno scrum ekipo.

V nasprotnih primerih je potrebno projekte in ekipe obravnavati individualno in se odločiti za primerno metodologijo. Tako se mi uvedba polnega scruma ne zdi smiselna v primeru, da gre za:

- ekipe za podporo strankam (zaradi nezmožnosti načrtovanja scrum v njih ne prinaša koristi);
- začasne ekipe, ki bodo delale na projektu, krajšem od treh mesecev (prirastek učinkovitosti ekipe zaradi scruma je nizek, na koncu projekta pa bo zgubljen);
- projekte tehnične nadgradnje brez vsebinskih sprememb (malo usklajevanja);
- projekte, na katerih delata le dve osebi (komunikacijo je nesmiselno formalizirati);
- projekte, na katerih delajo osebe, ki po svojih lastnostih v scrumu ne bi dobro funkcionirale.

Vseeno pa lahko take ekipe vzpostavijo proces, ki iz scruma črpa le koncepte načrtovanja in zagotavljanja kvalitete. Tako se lahko ne glede na velikost ekipe in projekta vpelje koncepte seznama zahtev, izpopolnjevanja seznama zahtev, sprinta in načrtovanja sprinta. Ni pa recimo potrebe po dnevnih scrumih, saj komunikacija iz oči v oči poteka ves čas. Za zagotavljanje kakovosti sta uporabna tudi koncepta definicije zaključenosti ter pregleda kode in testov.

Pred vpeljavo scruma je potrebno člane scrum ekipe usposobiti. Za to bi bil verjetno najbolj primeren scrum trener, ki ga bo v kratkem vpeljala AX ekipa, kasneje pa lahko NAV ekipa vpelje tudi lastnega scrum trenerja. Predlagam tudi, da se produktnim vodjem in vodjem scruma dodeli mentorje, ki v AX oddelku nosijo iste vloge, saj bo tako usvojitev procesov metodologije potekala hitreje.

Najpomembneje pa je, da ima vpeljava agilne metodologije podporo pri vodstvu. Raziskave (van Waardenburg & van Vliet, 2013; Nerur et al., 2005) so pokazale, da agilnost, vpeljana v razvojne ekipe brez podpore vodstva, ne dosega želenih rezultatov in je pogosto obsojena na propad. Vpeljava scruma v AX oddelek je dober primer tega, kako hitro in učinkovito je možno vpeljati agilno metodologijo in spremeniti kulturo oddelka, če je vodstvo pobudnik in sponzor vpeljave. Zato direktorju NAV oddelka predlagam, da se v primeru odločitve o vpeljavi scruma v razvojni proces oddelka temu posveti v popolnosti in z močno voljo ter stalnim pritiskom vsili spremembo kulture na NAV oddelku. Pri tem direktorja NAV oddelka čaka še težja naloga, kot jo je imel pri vpeljavi direktor AX oddelka, saj bo v NAV oddelku v scrum vključen manjši delež zaposlenih kot na AX oddelku. Zaradi tega bo prišlo do večjega trenja med obema konceptoma, med njima pa bo moral pri vodenju oddelka preklapljati tudi direktor NAV oddelka.

SKLEP

Metodologije razvoja programske opreme se delijo na linearne, iterativne in agilne. Med seboj se najbolj razlikujejo po načinu, na katerega upravljajo s programskimi zahtevami. Linearne in iterativne metodologije navadno v fazi analize in fazi načrtovanja razvijejo kar najbolj popoln nabor programskih zahtev, ki je potem striktno nadzorovan. Spremembe

programskih zahtev temeljijo na zahtevkih za spremembo, ki jih odobri ali zavrne odbor za nadzor sprememb. Agilne metodologije pa sicer na začetku določijo okvir produkta in visoko nivojske funkcionalnosti, vendar so oblikovane tako, da omogočajo evolucijo programskih zahtev tekom projekta.

Scrum je trenutno najbolj razširjena agilna metodologija. Uporablja iterativen, inkrementalen pristop za optimizacijo predvidljivosti in nadzor tveganja. Temelji na preglednosti, nadzoru in prilagoditvi, kar pomeni, da imajo vsi udeleženci procesa isto razumevanje dejstev, da se proces redno nadzoruje in se ob zaznanih odstopanjih čim prej prilagodi. Inkremente pripravlja večfunkcijska, samozadostna in opolnomočena ekipa, ki se sama odloča, kako bo inkrement pripravila. Scrum predpisuje vloge, dogodke, izdelke in pravila. Vloge opredeljujejo odgovornosti, dogodki predstavljajo formalno priložnost za nadzor in prilagoditev, izdelki pa so oblikovani za kar največjo preglednost informacij.

V literaturi se pojavljajo nasprotujoča si mnenja glede uporabe agilnih metodologij pri implementacijah ERP sistemov. Nekateri avtorji menijo, da so agilne metodologije boljše za enostavnejše projekte, drugi pa, da so primerne za spopadanje s kompleksnejšimi projekti, kamor sodijo tudi implementacije ERP sistemov. Študije kažejo, da večina razvojnih organizacij uporablja hibridne metodologije, ki uravnotežijo agilnost in disciplino.

AX oddelek podjetja Adacta d.o.o. je pred dvema letoma uvedel agilno metodologijo scrum in jo prilagodil svojim potrebam. Analiza je potrdila domnevo, da je metodologija, ki jo danes uporablja oddelek, hibridna. Scrum se namreč uporablja v kombinaciji z bolj tradicionalnim slapovnim pristopom. Projekti se začnejo s fazama analize in dizajna, ki jima sledi faza razvoja, ki poteka po scrumu. Faza testiranja se začne že tekom razvoja, vendar se večji del testiranja izvede na delavnicah, ki se organizirajo v končnih sprintih razvoja. Na koncu sledita faza vpeljave v produkcijo in prenos projekta v ekipo za podporo strankam.

Na podlagi intervjujev s ključnimi člani AX oddelka sem prišel do spoznanja, da hibriden pristop ni posledica neuspešne uvedbe agilne metodologije, ki je zaradi različnih faktorjev ni uspelo uvesti v celoti, temveč gre za premišljeno odločitev. Popolnoma agilna metodologija na implementacijskih projektih, ki jih izvajajo v AX oddelku, ne pride v poštev predvsem zaradi tipa projektov. Vsi projekti so namreč prodani za fiksno ceno, zato je potrebno pred začetkom razvoja opraviti temeljito analizo in napraviti podroben dizajn, na podlagi katerega se lahko sklene pogodba.

Moja domneva, da so specifike implementacij ERP sistemov glavni razlog za uporabo hibridne metodologije, torej ne drži. Se pa vpliv teh specifik pozna na uporabljeni metodologiji. Najbolj očiten je vpliv na organiziranost scrum ekip. Zaradi vsebinske kompleksnosti ERP-jev je na voljo manjše število izkušenih arhitektov rešitev, zato je potrebno njihovo znanje deliti preko vseh ekip. Ključne osebe oddelka so tako člani več

scrum ekip hkrati, kar lahko negativno vpliva na prirastek učinkovitosti zaradi stalnega izpopolnjevanja ekipe.

Izkušnje intervjuvanih ključnih oseb oddelka kažejo, da je bil bistven cilj uvedbe scruma dosežen. Dosegli so preglednost in predvidljiv cikel razvoja. Kvaliteta narejenega se je dvignila in uvedba scruma je pomagala uspešno zaključiti projekte, ki jih prej niso mogli. Boljša organiziranost je pozitivno vplivala tudi na zadovoljstvo zaposlenih. Izkušnje z uvedbo scruma v AX oddelek torej potrjujejo ugotovitve tistih avtorjev, ki agilne metodologije vidijo kot dober način obvladovanja kompleksnosti projektov implementacije ERP sistemov.

S primerjalno analizo sem ugotovil, da se NAV oddelek od AX oddelka razlikuje predvsem v precej večjem številu projektov v fazi vzdrževanja. Še do pred nekaj leti je veljalo, da ima NAV oddelek manjše projekte od AX oddelka, vendar se zadnje čase tudi NAV oddelek sooča z velikimi projekti. Take projekte z obstoječo metodologijo težje obvladujejo, zato se pogosto srečujejo s pomanjkanjem preglednosti in neučinkovitim manegementu zahtevkov za spremembo. Projekti posledično zamujajo in so manj dobičkonosni, kot bi lahko bili.

Na podlagi opravljene analize vpeljave scruma v AX oddelek, analize NAV oddelka ter primerjave med obema oddelkoma sem prišel do sklepa, da je vpeljava scruma v NAV oddelek izvedljiva in primerna. Moj predlog je, naj se uvede enaka hibridna metodologija, kot jo je že vpeljal AX oddelek, in sicer v implementacijske ekipe, kjer je to smiselno. Manjše ekipe oz. projekti in ekipe za podporo strankam naj ohranijo dosedanjo metodologijo razvoja.

Vpeljava scruma v ERP oddelke podjetja Adacta d.o.o. je lahko primerna tema nadaljnjih raziskav. Področje, ki ga na primer v tej magistrski nalogi nisem mogel zadovoljivo pokriti, je zaključevanje projektov z uporabljeno metodologijo. V času pisanja naloge je bil v okviru scruma začet in zaključen samo en projekt, pa še ta ne v popolnosti. V roku dveh let bo število zaključenih projektov večje, kar bo primeren vzorec za analizo vpliva hibridne metodologije na uspešnost zaključevanja implementacijskih projektov. Večje število zaključenih projektov bo omogočilo tudi raziskavo zadovoljstva strank z implementacijami ERP sistemov z uporabo hibridne metodologije. Morebitna vpeljava scruma v NAV oddelek pa bi bila primerna tema raziskave, ki bi analizirala razlike med metodologijama AX in NAV oddelkov ter razloge zanje.

LITERATURA IN VIRI

1. Adacta d.o.o. (2016). *Predstavitev podjetja Adacta d.o.o.* (interno gradivo). Ljubljana: Adacta d.o.o.
2. Agile Alliance. (2001). *Manifesto for Agile Software Development*. Najdeno 20. marca 2016 na spletnem naslovu http://www.agilemanifesto.org/
3. Bajec, M., & Krisper, M. (2003). Agilne metodologije razvoja informacijskih sistemov. *Uporabna informatika, 11*(2), 68-76.
4. Beck, K. (2000). *Extreme programming explained: embrace change.* Boston: Addison-Wesley Professional.
5. Boehm, B., & Turner, R. (2003). Observations on balancing discipline and agility. *Proceedings of the Agile Development Conference* (str. 32-39). Salt Lake City: IEEE Computer Society.
6. Bourque, P., & Feirley, R. E. (Ured.). (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* Los Alamitos: IEEE Computer Society.
7. Brehm, L., Heinzl, A., & Markus, M. L. (2001). Tailoring ERP systems: a spectrum of choices and their implications. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.* Maui: IEEE Computer Society.
8. Carvalho, R. A., Johansson, B., & Manhães, R. S. (2009). Mapping Agile Methods to ERP: Directions and Limitations. *CONFENIS 2009: The IFIP International Conference on Research and Practical Issues of Entrerprise Information Systems.* Gyor: IFIP Series.
9. Carvalho, R. A., Johansson, B., & Manhães, R. S. (2010). Agile Software Development for Customizing ERPs. V S. Parthasarathy, *Enterprise Information Systems and Implementing IT Infrastructures: Challenges and Issues* (str. 20-39). Hershey: IGI Global.
10. Cohn, M. (2010). *Succeeding with agile: software development using Scrum.* Upper Saddle River: Addison-Wesley Professional.
11. Coram, M., & Bohner, S. (2005). The impact of agile methods on software project management. *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (str. 363-370). Los Alamos: IEEE Computer Society.
12. Hopkins, R., & Jenkins, K. (2008). *Eating the IT elephant: moving from greenfield development to brownfield.* Upper Saddle River: Addison-Wesley Professional.
13. *Islovar*. (b.l.). Najdeno 11. aprila 2016 na spletni strani Islovar: http://www.islovar.org/
14. Johansson, B., & Carvalho, R. A. (2009). Management of Requirements in ERP Development: A Comparison Between Proprietary and Open Source ERP. *Proceedings of the 2009 ACM Symposium on Applied Computing* (str. 1605-1609). New York: ACM.
15. Khanna, K., & Arneja, G. P. (2012). Choosing an appropriate ERP implementation strategy. *IOSR Journal of Engineering*, 478-483.
16. Korkala, M., & Abrahamsson, P. (2007). Communication in Distributed Agile Development: A Case Study. *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)* (str. 203-210). Lübeck: IEEE Computer Society.

17. Lorente, L. N., & Lorente, C. N. (2013). *Implementing Microsoft Dynamics NAV 2013*. Birmingham: Packt Publishing Ltd.
18. Microsoft. (2015). *Microsoft Dynamics NAV 2016 Product Overview and Capability Guide*. Najdeno 22. junija 2016 na spletnem naslovu https://www.microsoft.com/en-us/download/details.aspx?id=49484
19. Nerur, S., Cannon, A., Balijepally, V., & Bond, P. (2010). Towards an Understanding of the Conceptual Underpinnings of Agile Development Methodologies. V *Agile Software Development* (str. 15-29). Berlin: Springer Berlin Heidelberg.
20. Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 72-78.
21. Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using Scrum in Distributed Agile Development: A Multiple Case Study. *2009 Fourth IEEE International Conference on Global Software Engineering* (str. 195-204). Limerick: IEEE Computer Society.
22. Plavše, D. (2015). *Pridobitve vpeljave metode Scrum.* (diplomsko delo), Maribor.
23. Ray, R. (2011). *Enterprise Resource Planning*. New Delhi: Tata McGraw-Hill Education.
24. Rojko, M. (2011). *Uporaba agilne metodologije "Scrum" pri razvoju državnega portala za poslovne subjekte*. (magistrsko delo), Ljubljana.
25. Schmidt, C. (2016). *Agile Software Development Teams*. München: Springer International Publishing.
26. Schwaber, K., Sutherland, J., & Beedle, M. (2013). *The definitive guide to Scrum: The rules of the game*. Najdeno 20. marca 2016 na spletni strani Scrum Guides: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf
27. Shankar, C., & Bellefroid, V. (2011). *Microsoft Dynamics Sure Step 2010*. Birmingham: Packt Publishing Ltd.
28. Sherrell, L. (2013). Waterfall Model. V *Encyclopedia of Sciences and Religions* (str. 2343-2344). Dordrecht: Springer Netherlands.
29. Sutherland, J. (1995). Business object design and implementation workshop. *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum) - OOPSLA '95* (str. 170-175). New York: ACM Press.
30. Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Cutter Agile Project Management Advisory Service: Executive Update*, 1-4.
31. Sutherland, J., & Schwaber, K. (2010). *The scrum papers: nut, bolts, and origins of an Agile framework*. Najdeno 22. marca 2016 na spletni strani Scrum Inc.: http://www.scruminc.com/scrumpapers.pdf
32. Sycor Group. (brez datuma). *From Axapta 1.0 to Microsoft Dynamics AX 2012*. Najdeno 29. aprila 2016 na spletni strani Dynamics AX by Sycor: http://en.sycor-group.com/dynamics-ax/solutions/dynamics-ax/history/
33. The Microsoft Dynamics AX Team. (2014). *Inside Microsoft Dynamics AX 2012 R3*. Redmond: Microsoft Press.
34. Theocharis, G., Kuhrmann, M., Münch, J., & Diebold, P. (2015). Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. *Product-Focused*

Software Process Improvement: 16th International Conference, PROFES 2015 (str. 149-166). Cham: Springer International Publishing.

35. Trąbka, J., & Soja, P. (2014). Agile versus Design-based Approach to ERP System Implementation: A Cross-case Study. *Finanse Journal of Management and Finance, 12*, 305-323.
36. Van Der Hoeven, H. (2009). *Erp and Business Processes.* Coral Springs: Llumina Press.
37. van Waardenburg, G., & van Vliet, H. (2013). When agile meets the enterprise. *Information and Software Technology*, 2154-2171.
38. West, D., Gilpin, M., Grant, T., & Anderson, A. (2011). Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research*.
39. Williams, L. (2012). What agile teams think of agile principles. *Communications of the ACM*, *55*(4), 71-76.
40. Womack, J. P., Jones, D. T., & Roos, D. (1990). *Machine that changed the world.* New York: Simon and Schuster.