Tallinna Ülikool
Digitehnoloogiate Instituut

Infotehnoloogia juhtimine

IT arendusprojektide metodoloogiate Waterfall ja Scrum kombineerimislähenemine eeskirjalike piirangutega

Magistritöö

Autor: Lavrenti Tšudakov

Juhendaja: Vladimir Tomberg

Autor: …… "____"_____2017.a
Juhendaja: …… "____"_____2017.a
Instituudi direktor: …… "____"_____2017.a

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev magistritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Lavrenti Tšudakov (sünnikuupäev: 06.03.1984)

1. Annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „IT arendusprojektide metodoloogiate Waterfall ja Scrum kombineerimislähenemine eeskirjalike piirangutega“ mille juhendaja on Vladimir Tomberg, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, 23.04.17

(digitaalne) allkiri ja kuupäev

SISUKORD

SISSEJUHATUS

Paljud kaasaegsed ja kindlasti tuleviku ühiskonna aspektid suuremal määral on seotud IT arendustega, mis toob endaga koos suurt vajadust luua süsteeme ning uusi tarkvara lahendusi. Samuti, mainitud vajaduse lisaks, omab mõju see fakt, et tavaline tarkvara eluiga on 5-7 aastat, mis pärast vajab ümbertegemist või nullist loomist, kuna aeg dikteerib uuendatud ja rangemaid nõudeid. Hetkel Eestis on juba käes IT sektori tööjõu puudujääk (Kiisler, 2016) ning ajaga vajaliku spetsialistide hulk ainult kasvab.

Viis, kuidas luuakse tarkvara on sarnane tootmisega. Esialgu uuritakse, mis on äriline hetkevajadus ja kuidas süsteem peab funktsioneerima. Edaspidi luuakse disain, kirjutatakse koodi ning tehakse valmistoode. Pärast seda testitakse ning antakse üle kasutamiseks tellijale ning hoolduseks. Aga nagu tootmises, protsessi efektiivsus on üks primaarsetest mõõdikutest. Tellija investeerib raha, ning selle investeeringu kaudu tahab saada tulu või vähendada kulusid. Eesmärk on leida sobiva tasakaalu kvaliteedi, maksumuse ning ajakulu vahel.

Iga ettevõte ise otsustab milliste metodoloogia abil luua tarkvara, kas see on olemasolev raamistik või enda oma leiutis. Eesmärk on formaliseerida reegleid ning protsesse, et terve meeskonna töö oleks korrastatud, ning tellija saaks tellitud lahendusi sobival viisil. Kuna magistri töö autori kogemus on suured ettevõtted ning tarkvara loomine on suur väljakutse, siis tekkis vajadus ka kaaluda levinud metoodikate tugevad ja nõrgad küljed ning pakkuda oma lahendust. Samuti reaalse projekti näitel tehakse katsetus, ning järeldusi, kas uus lähenemine paranes olukorra või mitte.

Frederik Brooks omas essees kirjutas, et „hõbekuuli ei eksisteeri“ (Brooks ,1986) ning viitas sellele, et ei ole olemas universaalset tarkvara arenduse metodoloogiat, mis rahuldaks kõike vajadusi, lahendaks probleeme ning aitaks tõsta efektiivsust ühel teatud viisil. Iga olukord vajab oma lähenemist ning viis, kuidas töö on korraldatud peab vastama konkreetsele situatsioonile. Arvesse võttes seda, et mainitud esse sai kirjutatud aastal 1986 ning probleem on seni aktuaalne, saab teha järeldust, et tõepoolest, universaalne lahendus

ei eksisteeri. See andis antud töö autorile julgust ning motivatsiooni, et katsetada luua lähenemist, mis sobib samuti konkreetse ettevõte vajadustele ning võiks pakkuda abi teiste arendusmeeskondadele, kuidas saab kohandada raamistiku just oma vajadustele.

Hetkel, kui arendustiim omab eesmärgi hakata kasutama tarkvara arendusi metodoloogiat, siis valik on väga lai: *Waterfall, Scrum, Kanban, XP, TSP, PSP, Dual Vee* model, *DevOps, Lean, Prototyping* ja paljud-paljud teised. Valiku tegemine on väga raske ning esialgelt konkreetne valik peab vastama tarkvara nõuetele, tiimi suurusele, seotud osapoolte huvidele ja eelarvele. Autori kogemus näitab, et suured ja inertsed ettevõtted eelistavad *Waterfall* lähenemist, kuna see annab väga hea projekti läbipaistvust, juhtimist, kulude prognoosi ja annab võimalust planeerida pikalt. Teiselt poolt *Agile* sobib väga hästi kiirematele ja väiksematele ettevõtetele, millised tahavad kiiresti näha esimest tulemust, kiiresti läbi kukkuda valede ideedega ning olla valmis alati teha muudatusi. Siin tõuseb loogiline küsimus, kui mainitud lähenemised on nii erinevad, kuidas võtta mõlematest ainult parimaid tunnuseid ning edukalt neid kasutada igapäevases töös? Antud probleemi laheduse leidmine ning kombineeritud lähenemise proovile panemine on autori magistritöö peamine väljakutse. Pakutud raamistik peab olema mõõdetav ning efektiivne. Antud töös antakse *Waterfall* ja *Scrum* metodoloogiate ülevaade, mis on nende peamised mõõdikud, plussid ja miinused. Selle baasil tehakse otsus, milliste tunnuste abil on mõistlik teha kombineerimist. Kolmandas osas tuuakse mõned näited eksisteerivate hübriidsete lähenemiste kohta, et uurida, kas tõepoolest on olemas vajadus pakkuda uut lähenemist või on võimalik alusena kasutada mingi olemasolevat. Neljandas osas tuuakse loodud hübriidse lähenemise kirjeldust ja kuidas seda mõõdeti, lähtudes uuritud teooriate parimatest praktikatest. Antud osa eesmärk on kirjeldada projekti loomist ja edenemist hübriidse lähenemise raames ning anda ülevaadet, kui edukalt seda õnnestunud rakendada. Ülevaade antakse nii üldisema IT projekti näitel, mis ei ole töö kirjutamise ajal valmis, kui ka alamprojekti kohta, mis on lõpetatud ja järelduste tegemine on võimalik ning loob lisaväärtust.

Töö käigul peavad olema leitud vastused järgmistele küsimustele:

- Kuidas luua efektiivset hübriidset lähenemist?

- Millised on peamised mõõdikud? (nii lähte- , kui ka pakutud raamistikute jaoks)
- Mis on pakutud lähenemise plussid ja miinused lähtuvalt saadud kogemusest?
- Kuidas praktikas rakendada ettevõttes uut lähenemist?

Samuti, et saada ülevaadet, kuidas sai korraldatud uuendatud arenduse protsess ning mis mõju tema omas projektidele said korraldatud mitteformaalsed vestlused. Sihtgrupi määramisel antud töö autor lähenes sellest, et peavad olema kaetud IT projektide valmimise olulisemad aspektid nagu:

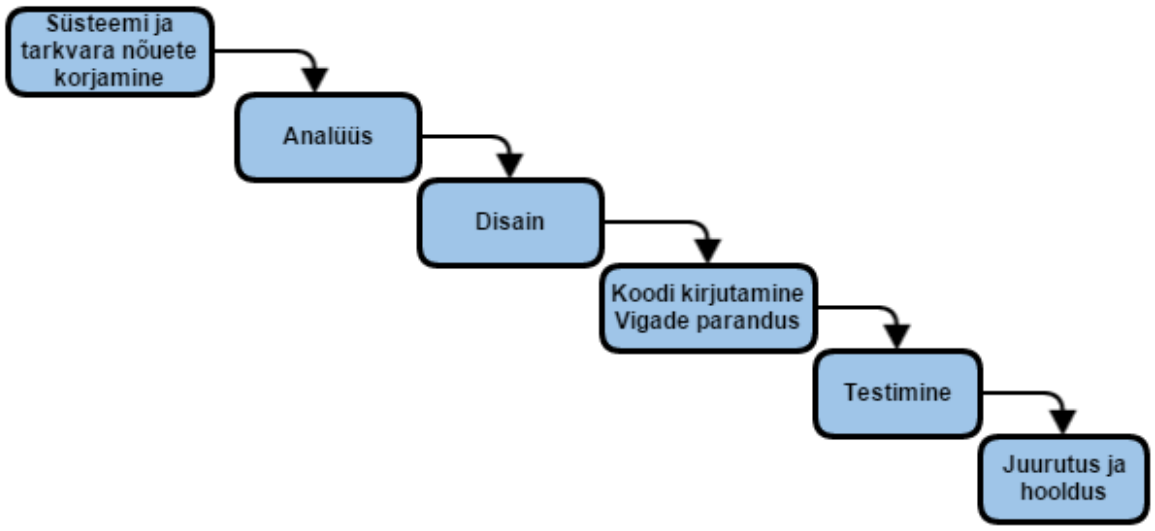
- IT juhtimine
- Projektijuhtimine
- IT arenduse protsessi korraldamine
- Hangete korraldamine
- Arendusmeeskondade juhtimine (siin on mõeldud juhtimine meeskonna sees, mitte see, kuidas grupi väljastpoolt juhitakse)
- Pakutud raamistiku edukus

Antud punktide lähtudes said valitud järgmised rollid vestluste tegemiseks: ettevõte IT juht, IT Arenduse valdkonna juht, projektijuhid, arendajate grupijuht, analüütikud, arendajad ja äripoolse tellijad. Lisaks sellele autor projekti raames on ise töögrupi liige ja oli võimeline mõjutada kuidas korraldatakse projekti tööd, millist raamistiku kasutada ning kuidas. Vestluste eesmärk oli kontrollida, kas projekti ehitus ning tulemused vastavad autori arusaamisele. Seega hübriidse lähenemise tööriistade katsetused, järeldused ning tulemuste ülevaade on suuremas mahus isiklik kogemus. Antud plaani alusel töö on juhtumi uuring, kus tehakse projekti analüüs erinevatest aspektidest: projekti juhtimine, inimeste juhtimine, mõõdikute kasutamine, tööriistade seadistamine.

Selleks, et pakkuda kombineeritud lähenemist ning teha järeldusi, on vaja detailselt vaadata olemasolevate paradigmade aspekte ning kaaluda nende sobilikkuse ettevõte vajaduste rahuldamiseks. Antud ülevaade on lähtepunkt uue lähenemise pakkumiseks.

1. WATERFALL METODOLOOGIA

Antud osa eesmärk on anda ülevaadet *Waterfall* metodoloogiast. Kaalutakse tugevad ning nõrgad rakendamise aspektid ja antud uuring aitab edaspidi pakkuda efektiivsemat kombineeritud lähenemist. Antud meetodil on pikk ajalugu. Esimene formaalne kirjeldus on loodud 1970 aastal Royce artiklis (Royce, 1970, lk 329), kus ta kirjeldas seda metodoloogiat nagu ebakorrektset mudelit. Ametlikult *Waterfall* terminina hakati kasutama Bell ja Thayer poolt ning süsteemi arenduse faasid defineeritud järgmiselt: süsteemi nõuete korjamine, tarkvara nõuete korjamine, detailanalüüs, koodi kirjutamine ning vigade parandused, arenduse testimine, formaalne testimine, juurutamine ning hooldus (Bell & Thayer, kuupäev puudub, lk 62). Reegel on see, et iga etapi algus on võimalik ainult juhul, kui eelmine etapp on edukalt lõppenud, mis tähendab, et ei ole võimalik teostada etappide kattuvusi. See on põhjus, miks metodoloogia sai nimetuseks "juga" või kaskaadmudel. Royce mudeli etapid (Joonis 1) graafiliselt on kaskaad ning liikumine on võimalik ainult vasakult paremale.



Joonis 1. *Waterfall* protsess

Tarkvara loomine protsessina lähtub sellest, et iga samm on täpselt defineeritud ning kasutatud vastavalt reeglitele. Järgmised alampeatükki lühidalt kirjeldavad iga *Waterfall* protsessi sammu sisu.

1.1. Tarkvara ning süsteemi nõuete korjamine

Antud etapi eesmärk on kirjalikult fikseerida kokkulepet äri ja IT vahel, milline süsteem tulemusena peab olema loodud ning millised nõuded peab rahuldama valmistoode. Antud kirjeldus võib sisaldata kasutajalugusid, funktsionaalsed ja mittefunktsionaalsed nõuded.

Funktsionaalne nõue räägib sellest, kuidas käitub programm või tema komponent. Näiteks arvutuse tegemine, dokumendi printimine, andmete manipulatsioon jne. Üldistatud kujul seda saab defineerida nagu: “süsteem peab tegema <nõue>”. Mittefunktsionaalne nõue kirjeldab seda, milline loodav süsteem peab olema ning annab põhjust hindama tema käitumist. Näiteks see võiks olla, süsteemi laiendatavus, jõudlus, hooldatavus jne. Antud vajaduste üldistatud kuju on “süsteem peab olema <nõue>”. (Wiegers, 2003)

Nõuete kirjeldus ning kasutajate lood edaspidi kasutatakse mitte ainult arendustiimi poolt, et jätkata disainiga, kuid ka arendajate poolt, et pärast valideerida funktsionaalsust. Seega etapp on oluline eeldus, et tagada terve projekti edukust. Sarnane lähenemine sai kasutatud antud töös käsitletud IT projekti raames ning tulemused on positiivsed.

Süsteemi nõuete korjamise etapi väljund on rakenduse nõuete dokumentatsioon, mis sisaldab ärilist ning tehnilist vaadet, projekti ulatust. Loodud dokumentatsioon kasutatakse selleks, et jätkata detailanalüüsiga ning luua tarkvara disaini. Peamised osad nõuete dokumentatsioonis on järgmised:

- Eesmärgid, peamised definitsioonid

- Üldine kirjeldus: toode vaade, peamised interfeisid (süsteemsed, kasutajate, riistvaralised, tarkvaralised, kommunikatsiooni), disaini piirangud, toode funktsioonid, kasutajad, eeldused ning sõltuvused
- Spetsiifilised nõuded: väliste interfeiside nõuded, funktsionaalsed nõuded, jõudluse nõuded, loogilise andmebaasi nõuded, süsteemsed atribuudid (Stellman & Greene, 2005, lk 308)

Kui antud etapp on läbitud, siis saab jätkata detailanalüüsiga, mis on oma poolt alusdokument disaini loomiseks.

1.2. Detailanalüüs

Kui nõuete koostamise faas on lõppenud, siis on võimalik täpsemates detailides kirjeldada projekti meeskonna jaoks, millised on ärilised vajadused, seotud ärilised protsessid, tulevased kasutajad ja nende vajadused.

Esialgu identifitseeritakse kes on peamised huvilised ettevõttes (Nuseibeh & Easterbrook, kuupäev puudub, lk 5). Läbi intervjuude defineeritakse kasutajate interaktsioonid süsteemiga, peamised protsessid ettevõttes, millised peavad olema toetatud või seotud uue tarkvaraga.

Peamine eesmärk on süvendada analüüsi ning lahti kirjutada millised omadused peab omama süsteem, et disaini faasis seotud osapooled oskaksid pakkuda oma lahenduse visiooni. Analüüsi faasi dokumentatsioon peab andma detailiseeritud probleemide ülevaadet (Systems Engineering Fundamentals Defense Acquisition University Press, 2001) ning anna sisendit järgmistes kategooriates:

- Ärilised vajadused: kuidas süsteemi tulevikus kasutatakse, milliste eesmärkide saavutamiseks ja kuidas kasutakse erinevad komponente, millised nendest on kriitilised, oodatud eluiga, ekspluatatsiooni keskkond

- Arhitektuurilised nõuded
- Funktsionaalsed ja mitte funktsionaalsed nõuded
- Jõudluse nõuded
- Põhifunktsionaalsus
- Disaini nõuded

Loodud dokumentatsioon omalt poolt on disaini loomiseks sisend.

1.3. Disain

Disaini faasi eesmärk on kirjelduse loomine, mis seletab kuidas ehitada süsteemi ning on peamine sisend arendustiimile. Eelmise etapi raames sai kirjeldatud mis on peamised ärilised probleemid, aga nüüd tekkib visioon kuidas neid vajadused rahuldada. Iga tarkvara ei ole puhas koopia igast suvalisest varem tehtud lahendusest ning seetõttu disaini sisu ei ole universaalne. Iga kord valitakse konkreetne viis ja tööriistad kuidas seda realiseerida. Tulemusena on dokument, mis kirjeldab millistest komponentidest peab olema loodud uus tarkvara, kuidas antud komponendid töötavad koos, mis andmed, kuidas liiguvad ning kuidas neid hoidma, mis annab andmebaaside struktuuri sisendit, arhitektuuriline vaade ja muud teised aspektid.

Disainis faasi lõpus toimub üleminek, kus süsteemi kirjelduse põhjal hakatakse reaalselt koodi kirjutama. Paralleelselt projektijuht igas etapis koordineerib tööd ning vastutab selle eest, et oleks koostatud ühine ja nähtav plaan, mis kirjeldab kes ja millises järjekorras peab tegema oma tööd, et saavutada soovitud tulemust. Autor kogemus näitab, et reeglina projekti meeskond ise otsustab mis on vajalik dokumentide komplekt, mis sobib konkreetse olukorra lahendamiseks. Näiteks, kui projekt on piisavalt väike, siis arenduse vajadusi saab katta lühike dokument. Põhinõue on see, et kirjelduse loomine tehakse sisendi andmiseks, mitte dokumenti tekitamiseks. Kui tarkvara disain teada, siis arendusmeeskond saab alustada luua rakendust.

1.4.Koodi kirjutamine ning vigade parandused

See on protsessi samm, kus programmeerijad peavad nõuete ning disaini kirjelduse kasutades kirjutama käivitava programmi koodi. Arenduse raamistiku ning keele valik saab olla dikteeritud paljude aspektide poolt. Näiteks, meeskonna kogemus, süsteemi nõuded, projekti suurus ja keerukus, kasutatav keskkond ja teised. Oluline osa on algoritmide välja töötamine ning nende realiseerimine. Algoritmidest sõltub süsteemi jõudlus, konkreetsete ülesannete lahendusvõimekus ja stabiilsus.

Tulemusena, kui koodi tükid saavad valmis, neid hakatakse käivitama, et valideerida funktsioneerimist. See on esimene hetk, kus saab hinnata loodava tarkvara kvaliteedi. Vigade hulk samuti sõltub väga paljudes parameetritest. Mõned allikad (McConnel, 2006) pakkuvad, et statistiliselt vigade arv programmeerimises on keskmiselt 15-50 vigu 1000 tarnitud koodirea kohta. Seega tuleb arvestada, et arendaja ei saa ainult kirjutada koodi, kuid peab alati teha parandusi või vajadusel tõsta koodi kvaliteedi, kui mingi parameeter, näiteks, jõudlus ei ole rahuldav. *Bugide* leidmiseks ja kindluse valideerimiseks on vaja ka läbida eraldi testimise faasi.

1.5.Tarkvara testimine

Tavaline praktika kaskaadmudelis on see, kui testimine tehakse iseseisva testijate tiimi poolt pärast, kui vajalik funktsionaalsus on arendatud ning enne, kui tehakse üleandmine ärile (EtestingHub, kuupäev puudub). Testimise eesmärk on valideerida programmi kvaliteedi. Kõikide vigade leidmine tarkvaras praktikas on võimatu (Pan, 1999) ning eesmärk on kontrollida, kas vajalik funktsionaalsus töötab nagu peab ning lahendus vastab ärilistele vajadustele.

Viga leidmisel, seda registreeritakse ning suunakse tagasi arendajatele, et vastav parandus saaks tehtud. See on ajaline lisakulu, millega tuleb arvestada projekti planeerimisel. Viga parandamise hind sõltub etapist (McConnell, 2004, lk 29), millal see sai tuvastatud ja korrigeeritud. Kui varem, siis odavam on paranduse tegemine ning esitamine ärile. Sama seost on näha ka antud töös uuritud projektis.

Samuti. nagu programmeerimises, testimiseks kasutakse palju meetodeid, tööriistasid ning lähenemisi. Näiteks, programmi saab kontrollida "valge kasti" meetodil, kui testijale on näha programmi koodi ja algoritme, ning testide ehitamine korraldatakse antud teadmistega. Või vastupidine lähenemine - "musta kasti" meetod, kui testija ei tea koodi, ning saab kontrollida ainult programmi sisendit ja väljundit. („Differences Between Black Box Testing and White Box Testing", kuupäev puudub)

Kui testijate meeskond lõpeb oma tood, siis äri teeb katsetuse kasutada rakendust. Seda nimetatakse vastuvõtmise testimisena, mille eesmärk on kindlaks teha funktsionaalsuse kvaliteedi. On tark luua vastuvõtmise-üleandmise akti, kus kirjalikult fikseeritakse tulemusi ning äri pool kinnitab, et loodud toode vastab reaalsetele vajadustele. Praktikas, ilma selleta võivad tekkida vasturääkivused. Kinnitamise pärast äri saab alustada juurutamist, et lõplikult võtta toodet kasutusse.

1.6. Juurutamine ning hooldus

Pärast kvaliteedi kontrolli uus tarkvara peab olema paigaldatud vajaliku seadmete peale. Vajadusel, kui näiteks vanas süsteemis eksisteerivad vajalikud andmed, neid migreeritakse ning kontrollitakse ka ülekandmise kvaliteedi. Kui äri pool hakkab igapäevaselt kasutama rakendust, siis peab olema tagatud tugi probleemide tekkimisel ning hooldus, et tagada pideva äriprotsesside funktsioneerimist. Selleks võib olla loodud eraldiseisev kasutajate toe valdkond, mille eesmärk on korjata pöördumisi ning lahendada jooksvaid probleeme.

Kirjeldatud sammud annavad ülevaadet, kuidas luuakse tarkvara *Waterfall* metodoloogia kasutusel. Vastutuste piiritlemiseks on vaja defineerida ka rolle (Pierce, 2016) projektis.

1.7. Rollid

Arendusprotsessi sammud ei ole ainukene oluline aspekt, mis peab olema arendusmeeskonnale teada, et efektiivsemalt korraldada oma tootmist. Et ei tekkiks küsimust: „Mida ma pean projektis tegema ja mille eest vastutan?“, tuleb piiritleda rolle (Berkun, 2008, lk 190). Kaskaadmudeli meeskonna liigete rollid reeglina on defineeritud järgmiselt:

Analüütik – siin mõnikord eristatakse süsteemi ja ärianalüütikud. Peamine ülesanne on teostada analüüsi, et aru saada millised on ärilised vajadused, selle põhjal loob dokumentatsiooni, mis on läheülesanne arendajatele. Tarkvara lähteülesanne on dokument lai ulatusega, mis räägib nii toode visioonist, kuid ka detailselt kirjeldab vajaliku funktsionaalsust. Antud protsessis kaasatakse ka arhitekte, arendajad ning projektijuhi, et korjata võimalikult palju arvamusi

Arendaja – meeskonna liige, kes kirjutab koodi lähteülesanne ja spetsifikatsiooni põhjal

Testija - kui kood on kirjutatud, siis testija kontrollib selle kvaliteedi. On erinevad liike teste, et kontrollida integratsioone, funktsionaalsust, jõudlust jne. Eesmärk on tarnida kvaliteetset toodet, ning antud roll teostab antud validatsiooni enne tarkvara üleandmist ärile. Kui tuvastatakse viga, siis seda registreeritakse ning suunakse tagasi arendajale parandamiseks.

Projektijuht - peamiselt administratiivne roll. Projektijuht koordineerib arendustiimi, koostab projektiplaani ja eelarvet, jagab tööd inimeste vahel ning jälgib progressi. Peamine huvi on see, et valmis toode on üle antud kokkulepitud ajal, rahuldab ärilisi vajadusi ning

jagab arendusmeeskonna koormust nii, et tööd ning vastutus oleksid jagatud proportsionaalselt inimeste vahel.

Autori kõik IT projektid sisaldasid kirjeldatud rolle, kuna peamiselt tegu oli kaskaadmudeliga. Aga isegi nii hästi formaliseeritud protsessis võivad esineda nii tugevad, kui ka nõrgad küljed.

1.8. *Waterfall* metodoloogia tugevad ja nõrgad küljed

Kaskaadmudel on kasutusel paljude ettevõtete poolt ning selle ajalugu on piisavalt pikk, et said tehtud järeldused, millised küljed saab defineerida nagu tugevused. Antud kogemuse kasutamine on eeldus, et hübriidmudel võtab endasse ainult need tunnuseid, millistest on reaalne kasu. Aga reeglina tugevustega koos on koht kriitika jaoks ja eksisteerivad olukorrad, kus *Waterfall* rakendamine on keeruline ja ebamõistlik. Antud alampeatükk on jagatud kaheks osadeks ning annab peamiste tunnuste ülevaadet plusside ning miinuste perspektiividest.

1.8.1. Plussid

Waterfall lähenemisel on palju plusse (Smart, 2006) ning selle kaskaadprotsess annab ettevõttele eeliseid, milliste saavutamine *Agiilsete* praktikatega on keeruline:

- Projekti parem läbipaistvus ja jälgitavus
- Tugev formaliseerimine, mis maandab palju riske
- Täpsem ja kindlam plaan

- Disaini vead on tuvastatud enne tarkvara loomist, mis säästab aega koodi kirjutamise faasis
- Dokumentatsioon on terviklik ning valmis enne koodi kirjutamist, seega uuele arendajale on lihtsam alustada tööd
- Lähenemine on väga struktureeritud ning progressi jälgimine on lihtsam etappide järgi
- Projekti kogumaksumuse piisavalt täpne arvutamine on võimalik, kui kõik nõuded on korjatud (funktsionaalse ning kasutajate interfeisi spetsifikatsioonide kaudu)
- Testimine on lihtsam, kuna stsenaariumid on defineeritud funktsionaalses spetsifikatsioonis

1.8.2. Kriitika

Samuti antud metodoloogia rakendamisel tuleb arvestada sellega, et eksisteerivad selle nõrgemad küljed, millised saavad mõjutada tulemust. Nende tundmine saab aidata vältida negatiivsete efektide tekitamist. Antud ülevaade sai koostatud antud töö autori kogemusel ning allika (Smart, 2006) kombineerimisel:

- Ebapiisav paindlikus
- Formaalne eesmärk on juhtida projekti, isegi maksumuse, kvaliteedi ning eelarve kahjuks
- Äri poolt näeb reaalset tulemust ainult projekti lõppfaasis
- Muudatuste sisseviimine on keeruline
- Projekti skoobimuudatus reeglina tähendab lisaarendusi, eelarve ülekulu
- Tellijale on keeruline formaliseerida nõuded abstraktsel tasemel ning lõplikult saab aru, mis tõepoolest vaja on, ainult kui valmis toode on tarnitud. Selle pärast hilisem ümber tegemine on keeruline ning kallis

- Arendusfaasis ei ole võimalik muuta nõudeid
- Võrreldes *agiilse* metoodikaga projekti kestvus on pikem

Kui protsess on paigas ning vastutused jagatud, siis on võimalik projekti käivitada. Iga projekti valmimise protsess nõuab mõõtmist, et hinnata progressi ja töötajate efektiivsust. Mõõdikute lisaks on vaja defineerida ressursse, ehk sisuliselt mida mõõdetakse. Selleks on kasutatud erinevad meetrikaid ning nende ülevaade on järgmise peatükki eesmärk. Antud kirjelduse andmisel peab olema leitud vastus küsimusele mida ja kuidas mõõdetakse.

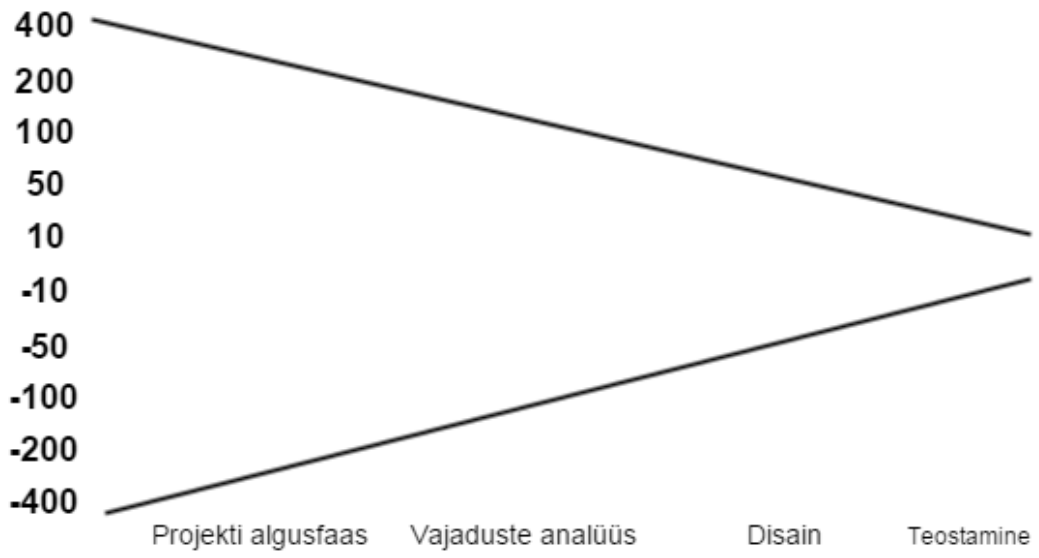
1.9.Meetrikad

Projektijuhi roll on juhtida projekti ressursside kasutamist kõige optimaalsel viisil. Kuna iga IT arenduse projekt on investeering, siis tellija tahab teada kuidas edeneb tarkvara valmistamine ning millal ta hakkab tulu tooma. Peamised projekti ressursid on raha, inimesed ning aeg. Kui tellija saab lahendus lubatud ajal, eelarve piirides ning kokkulepitud funktsionaalsusega, siis saab peeta projekti edukaks. Projektide peamiste läbikukkumise põhjuste hulgas (Charette, 2005) on samuti sama ressursside väär kasutamine või ebapiisav kaasamine.

Projektijuhil on oluline roll, et leppida kokku antud vajaliku ressursside mahu ning korraldada progressi ning kasutuse jälgitavust. Reeglina luuakse projektiplaani, kus on määratud projekti osapooled ning nende vastutused. Projektis osalev liige kannatab kokkulepitud rolli, mis omalt poolt tähendab, et ta on vastutav konkreetse töö liigi ja hulga eest. Iga meeskonna liige annab oma tööde hinnangut ning see on alus projekti plaani koostamiseks. Et tõsta terve projekti läbipaistvust seda jagatakse loogilisteks osadeks - etappideks. Igal etapil võib olla tähtaeg, eelarve ning kaasatud osapooled, konkreetne ülesehitus lepitakse kokku eraldi projekti algfaasis. Kui meeskond loob oma visiooni ja on

võimeline öelda, kui palju aega võtab iga konkreetne ülesanne valmimiseks, seda pannakse plaani. Mis annab esimest olulist parameetrit - **hinnang**.

Hinnang on eeldus ja ennustus, mis annab indikatsiooni vajaliku töömahu kohta ülesanne täitmiseks. Graafik (Joonis 2) on annab ülevaadet, kui täpsed võivad olla hinnangud igas projekti faasis.



Joonis 2. Projekti hinnangu ebamäärasus

Hinnangud peavad olema tihti arutatud ning uuendatud, kuna isegi piisavalt täpne prognoosi viga akumuleerub ajas, ning omab "lumepalli" efekti. Näiteks, kui abstraktse esimese etappi hinnang on täpsusega 90% ja teise etapi ennustuse täpsus on sama - 90%, siis tulemusena kahe etappi hinnangute täpsus on 90%*90% = 81%. (Berkun, 2008, lk 32-39)

Projektijuht paneb antud prognoosi plaani ning hakkab regulaarselt võrrelda tegeliku tööde valmimist ning hinnanguuid. Antud viis annab võimalust korraga jälgida erinevaid parameetreid. Näiteks, hinnangu täpsust, plaani täitmist, kui palju võtab aega vaheetappide valmimine. See annab järgmist parameetrid, mida mõõta - **tegelik kulutatud aeg**. Lisaks sellele, tellija tahab reeglina ka mõõta **rahaliselt** nii hinnangu, kui ka tegeliku kulude perspektiividest.

Iga projekti ülesanne, mis peab olema täidetud meeskonna poolt on jälgitav ühik. Selleks, et piisavalt vara tuvastada ajalist nihkumist riski on vaja mitte ainult jälgida iga etapi edenemist. Putnam ja Lawrence omas raamatus defineerivad viit põhimõõdikuid (Putnam & Mayers, 2003), millised on hädavajalikud projekti progressi ning kvaliteedi mõõtmiseks. Need kasutakse antud töös alusena ning samade meetrikate näitel tehakse ülevaade, kuidas saab mõõta ka Scrum projekte. Antud juhul mõõtmine on võrreldav ning osa mõõdikutest saab olla edaspidi kasutatud hübriidses lähenemises.

- Projekti suurus - funktsioonide arv, on subjektiivne mõõdik, kuna funktsiooni mõiste defineeritakse projekti sees ning ei saa olla universaalne
- Projekti kvaliteet - tuvastatud defektide arv (tuvastatud ning registreeritud vigade arv) projekti etapis või kumulatiivne defektide arv, et hinnastada terve projekti kvaliteedi
- Projekti kestvus - mõõdik, mis annab indikatsiooni kui kaua kestab terve projekt kuudes või päevades
- Meeskonna tootlikus - Tunnid / ärifunktsionaalsuse kohta
- Pingutus - inimkuud on ühik mille arvutuseks võetakse osalevate isikute arvu ning korrutatakse seda kuude arvuga, milliste jooksul nad teevad tööd projekti raames. Näiteks, kui meeskonnas on 4 liiget, ning nemad töötavad 2 kuud, siis antud perioodi jooksul sai kulutatud 4*2=8 inimkuud. Reeglina ühe inimese kohta väärtus on 160 tunde ühe kuu jooksul.

Kirjeldatud mõõdikud on põhilised ning vajadusel hulk võiks olla laiendatud. Antud peatükkis kirjeldatud osad on alus *Waterfall* protsessi käivitamiseks ning juurutamiseks. Järgmisena antakse ülevaade *agiilsetest* praktikatest ning oma magistritöös autor katsetab pidada sama struktuuri, et toetada metodoloogiate võrreldavust.

2. SCRUM METODOLOOGIA

Waterfall metodoloogia vastandlikuna ajaga sai pakutud *agiilne* lähenemine, kuhu kuulub ka *Scrum*. Ajalooliselt see oli vastus aeglasele ning väga rangelt formaliseerinud *Waterfall* lähenemisele („The Scrum History“, kuupäev puudub), mis ei toeta muudatuste tegemist projekti käigul. Antud metodoloogia põhimõte on iteratiivne arendusprotsess, dünaamiline nõuete loomine ning nõuete realiseerimine iseorganiseeruvate rühmade sees. Mõned *agiilsete* metodoloogiate näited on ekstremaalne programmeerimine, *Scrum*, *FDD*. Kõike paremal viisil lähenemist kirjeldab *agiilne* manifest: “Tarkvara luues ning teisi tarkvara loomise juures aidates oleme leidnud selleks tööks paremaid viise. Oleme hakanud hindama: inimesi ja nendevahelist suhtlust rohkem, kui protsesse ja arendusvahendeid töötavat tarkvara rohkem, kui kõikehõlmavat dokumentatsiooni koostööd kliendiga rohkem, kui läbirääkimisi lepingute üle reageerimist muutunud oludele rohkem, kui algse plaani järgimist Ka parempoolsetel teguritel on väärtus, kuid me hindame vasakpoolseid tegureid kõrgemalt“. (Beedle et al., 2001)

Samuti antud manifesti põhjal sai tuletatud 12 printsiipe („12 Principles Behind the Agile Manifesto“, kuupäev puudub), milliste täitmine on oluline täieliku ja õige *agiilse* metodoloogia rakendamisel.

- Meie peamine prioriteet on tellija rahulolu vara ning pideva väärtusliku tarkvara tarnimise kaudu
- Muudatused on teretulnud, isegi arenduse hilja etapis. *Agiilne* protsess kasutab muudatusi kliendi konkurentsi eeliseks
- Tarni töötava tarkvara tihti, paarest nädalast kuni paarini kuuni, eelistades lühikesema perioodi
- Äri ja IT inimesed peavad tegema koostööd projekti raames igapäevaselt
- Ehita projekti motiveeritud indiviidide hulgas. Anna nendele parima keskkonna ja toeta nende vajadusi, usalda neid, et töö saaks tehtud

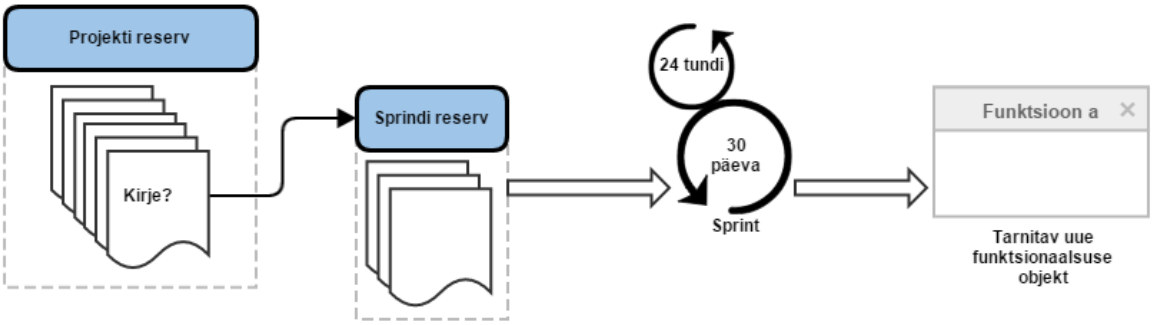
- Kõige efektiivsem ja parim meetod informatsiooni vahetuseks arendustiimi sees on isiklik suhtlemine.
- Töötav tarkvara on primaarne tööprogressi mõõtja
- Agiilne protsess toetab säästliku arendust. Sponsorid, arendajad ja kasutajad peaksid suutma toimetada ühtlases tempos lõputult
- Pidev tähelepanu tehnilise kvaliteedile ja hea disain parandab agiilsust
- Lihtsus, kunst mitte tehtud töö maksimeerimine, on väga oluline.
- Parim arhitektuur, nõuded ja disain on toodud iseorganiseeruva meeskonna poolt
- Regulaarselt meeskond arutab kuidas efektiivsemalt tööd tegema, siis teeb parandusi ja korrigeerib oma käitumist vastavalt sellele

VersionOne uuringu tulemusena selgub („The 10th Annual State of Agile Report“, 2016), et peamised põhjused, miks valitakse just mingi agiilne raamistik on valmistoode tarnimise kiirendamine, võimalus paremalt juhtida prioriteete, produktiivsuse tõus, tarkvara kvaliteedi paranemine, tarkvara tarne ennustatavuse paranemine ning parem äri ja IT koostöö. Need aspektid on head motivaatorid, et katsetada antud lähenemist.

Populaarseim *agiilsetes* lähenemistest on *Scrum* („The 10th Annual State of Agile Report“, 2016), mis on üks peamistest põhjustest, miks just selle raamistiku näitel katsetakse pakkuda hübriidset lähenemist. Teine väga oluline põhjus on see, et paljude tiimide liikmetel vastav kogemus esines projekti käivitamisel.

Scrum protsess oli loodud Jeff Sutherlandi poolt aastal 1993 („The Scrum History“, kuupäev puudub), ning termin on võetud analoogiast, mis oli kasutatud Takeuchi ja Nonaka uuringus. Mainitud uuring sai kirjutatud 1986 aastal (Takeuchi, H. & Nonaka, I., 1986), kus sai võrreldud kõrge jõudlusega polüfunktsionaalsed tiimid ning *Scrum* formatsioon ragbi mängus.

2.1.*Scrum* protsess



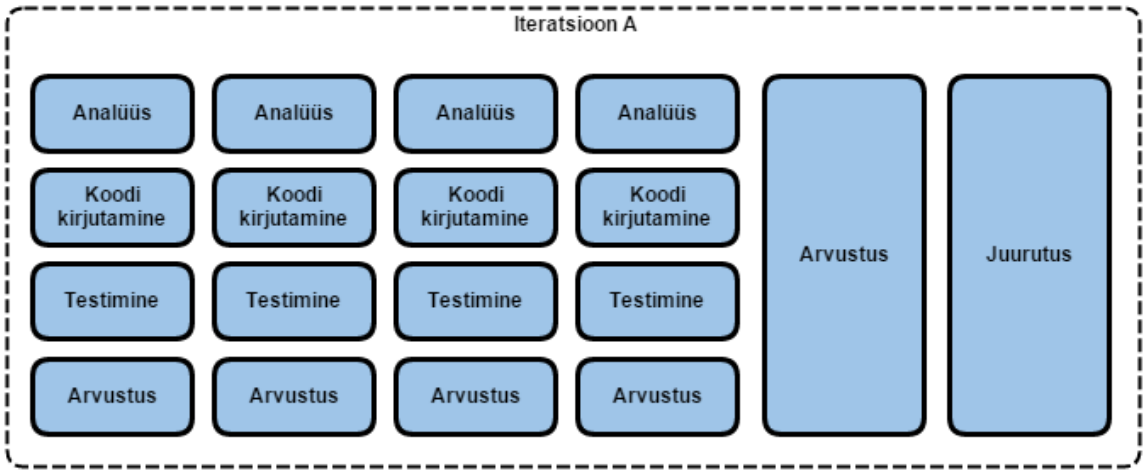
Joonis 3. *Scrum* protsess

Terve projekt *Scrumis* on jagatud *Sprintideks*. *Sprint* on fikseeritud aja aken, mille suurus ei saa olla muudetud projekti käigul ning kestvus on kokkuleppeline. Reeglina pikkus on 2 nädalat, aga mitte rohkem kui 1 kuu, metodoloogia kirjelduse järgi. Kui reaalselt tiimid ise valivad endale sobiliku kestvust ja mõnikord see on kauem, kui 1 kuu. Peamine reegel on see, et *Sprindi* tulemusena peab olema tarnitud kasutatav tarkvara (Joonis 3) osa ja see on iteratsiooni lõpu eesmärk. (Schwaber & Sutherland, 2016)

Sprindil on defineeritud hulk reegleid:

- Ei tohi teha muudatusi millised mõjuvad Sprindi eesmärke
- Ei tohi alandada kvaliteedi
- Projekti ulatus võiks olla täpsustatud ning arutatud PO ja arendusmeeskonna vahel, kui uued teadmised on tekkinud

Sprindi tühistamine on võimalik ainult Toote omaniku poolt. Iteratsioon sisaldab endas väiksemad tegevused, millised on tüüpilised tarkvara arendamise protsessi jaoks. Need alamtegevused (Joonis 4) on analüüs, koodi kirjutamine, testimini, koos äriga funktsioonide arvustus. Sprindi lõpus, kus kokkulepitud funktsionaalsuse osa saab valmis, siis tehakse lõplik vastuvõtmine ning juurutamine, et tellija saaks alustada selle kasutamisega. Ning ongi tagatud *Scrumi* üks peamistest postulaatidest, et iga iteratsioon peab tooma reaalsed tulemused. (Schwaber & Sutherland, 2016)



Joonis 4. Iteratsiooni sisu

Sprint koosneb teistest tegevustest (Schwaber & Sutherland, 2016), milliste eesmärk on tagada agiilse protsessi funktsioneerimist, nad on: *Sprindi* planeerimine, Igapäevane *Scrum*, arenduse töö, *Sprindi* ülevaade ning *Sprindi* Retrospektiiv. Järgmised peatükid annavad antud tegevuste detailsema ülevaadet. Eesmärk on uurida, kas mingi antud protsessidest on sobiv selleks, et sisse ehitada *Waterfall* raamistikku ning edukalt kasutada kombineeritud lähenemises.

2.2.*Sprindi* planeerimine

See on grupitöö, kus osaleb terve meeskond. Antud koosoleku jooksul on vaja leida vastused küsimustele: “mis saab olla tehtud Sprindi jooksul?” ning “kuidas realiseerida valitud töömahu?”. Selleks koos arutakse *Backlogi* kirjeid ning otsustatakse mis osa sellest on teostatav. Et seda valikut teha meeskond vaatab eelmise Sprindi ning prognoositava tootlikkuse peale. Kui kirjed on valitud, siis koostatakse *Sprindi* eesmärk. Kui eesmärk on teada, siis arendusrühm loob vajaliku tarkvara osa arhitektuuri. Planeerimise tulemusena on võimeline seletada *Scrum* meistrile ning tooteomanikule, kuidas kokkulepitud projekti etapi ulatus peab olema realiseeritud.

2.3. Igapäevane *Scrum*

Igapäevaselt arendusmeeskonna sees tööde sünkroniseerimiseks ja järgmise päeva plaani väljatöötlemiseks korraldatakse nii nimetatud "Igapäevane *Scrum*". See on 15 minutiline koosolek, kus iga tiimi liige peab vastama kolmele küsimustele:

- Mis sai tehtud eile, et jõudma iteratsiooni eesmärkideni?
- Mis on minu plaan tänaseks, et jõudma iteratsiooni eesmärkideni?
- Kas on mõned takistused?

Antud lähenemine annab võimalust kiiresti reageerida muutuvale olukorrale ning parandab kommunikatsiooni.

2.4. *Sprindi* ülevaade

Iga *Sprinti* lõpus tehakse mitteformaalne tulemuste ülevaade meeskonna ning huvigrupi vahel. Koos vaadetakse läbi funktsionaalsusele, mis on tehtud ja mis mitte. Valmis tehtud tarkvara kohta tehakse ka demonstratsioon. Arendusmeeskond annab tagasiside mis on läinud hästi ja mis mitte ning kuidas parandada edaspidist tööd. Samuti koostakse plaani järgmise *Sprinti* jaoks ning arutakse ajaplaan ning eelarve.

2.5. *Sprindi* Retrospektiiv

See on rühmatöö, mis reeglina korraldatakse *Sprinti* planeerimise ning *Sprinti* ülevaade vahel. Arendusmeeskond teeb "tervise kontrolli", et aru saada mis läks hästi eelmisel

Sprindil, inimsuhete, protsesside ning vahendite vaates. Samuti koostetakse plaani, kuidas parandada olukorra.

2.6.Scrum väljundid

Scrum protsess kasutab erinevad tööriistad, et tagada läbipaistvust ning koordineerida tööd. Need on äriliste vajaduste kirjeldus (Schwaber & Sutherland, 2016) kokku lepitud detailsusega ning luuakse tabelikujul. Ilma antud väljundite kasutamiseta ei ole võimalik eeldada, et kogenud *Scrum* meeskond saab aru kuidas ta peab toimetada ning mis on arendus tööde alus.

2.6.1. Product Backlog

Esimene ning peamine artefaktidest on *Toode Backlog*. See on list, mis kirjeldab kõik tööd, millised on vajalikud toode valmistamiseks. List on dünaamiline ning sisaldab rakenduse funktsioone, nõudeid, detaile ja parandusi. Tooteomanik on vastutav selle eest, et *Backlog* oleks täidetud vajaliku informatsiooniga ning omaks järjekorra. Kirjed, mis omavad kõrgema prioriteedi ning asuvad list eespool omavad ka rohkem detaile. Arendusmeeskond kavatseb panna hinnanguid nende jaoks, mis aitab koostada *Sprindi* plaani. Teiselt poolt kirjed, millised ei oma nii kõrge prioriteedi ei vaja nii palju detaile, mis tähendab, et nende põhjalikuma analüüsiga tegeletakse ainult vajadusel. (Certified ScrumMaster, 2015, lk 23)

2.6.2. Sprint Backlog

See on plaan järgmiseks *Sprindiks*, mis ütleb milline toode funktsionaal peab olema tarnitud iteratsiooni tulemusena ning kui suur on toode "järelkasv". Teoreetiliselt iga kirje *Sprindi* lõpus peab olema märgistatud nagu "tehtud". Kui osa kirjetest saab tehtud, siis uuendatakse hinnanguline jäänud tööde maht, mis on progressi indikatsioon. Oluline märkus on see, et võrreldes kaskaadmudeliga, maht antakse mitte ajaliselt, aga hinnanguliselt, et kirjeldada kui suur/keeruline kirje on. Ainult arendusmeeskond omab õigust teha muudatusi ja uuendab pildi igapäevaselt, et tagada operatiivsust. Oluline aspekt on kokku leppida ning ühiselt kasutada "tehtud" definitsiooni. Iga meeskonna liige peab aru saama mida täpselt see tähendab, ning teadmine peab olema jagatud seotud tiimide vahel ka. Autori kogemus näitab, et oluline erinevus võib esineda, kui, näiteks „"tehtud" tähendab ainult koodi kirjutamist, ilma äritestimist. (Certified ScrumMaster, 2015, lk 29)

2.7.Rollid

Võrreldes *Waterfalliga*, *Scrum* protsessis on defineeritud vähem rolle. Mõned nendest eeldavad seda, et vastutav osapool on võimeline teha antud situatsioonis vajalikud tööd. *Scrum* metodoloogia raames („Scrum Roles Demystified", kuupäev puudub) on sõnastatud järgmised rollid:

- **Toote omanik** (*product owner - PO*) - on peamine kontakt äri ja IT vahel. Koostöös aitab koostada toote visiooni ning seletada seda arendus meeskonnale. Loob toode *Backlogi* ning prioritiseerib arendusülesandeid.

- **Scrum meister** (*Scrum Master - SM*) - peamine vastutus on suurendada meeskonna tootlikust. Kõrvaldab takistusi tööprotsessis, toetab metodoloogia parema kasutamist igapäevaselt, motiveerib, koolitab ning aitab.
- **Arendustiim** (*Development Team*) - protsessi poolt eeldatakse, et meeskond sisaldab kõikvajalikud kompetentse, et tarnida funktsioneeriva toodet. On iseorganiseeruv ning tervikuna vastutab tulemuse eest. Mitte keegi ei ütle tiimile, kuidas realiseerida funktsionaalsust. Vaatamata sellele, et iga meeskonna liige võiks täita mingi klassikalist rolli, näiteks analüütiku, ikkagi kasutakse ainult „arendaja“ definitsiooni.

Scrum meeskonna suurus peab olema nii väike, et olla krapsakas ning piisavalt suur, et realiseerida olulist tööd *Sprindi* raames. Arendajate hulk, vähem kui kolm liiget on potentsiaalne risk mitte jõuda tarnida realiseeritava toode osa. Tiim suurusega rohkem, kui üheksa inimest tekitab liiga palju koordineerimise ülekulusid. (Schwaber & Sutherland, 2016)

2.8.*Scrum* metodoloogia tugevad ja nõrgad küljed

Sarnaselt *Waterfall* protsessiga ei ole võimalik olukord, kui protsessis esinevad ainult tugevad küljed. Nagu eelnevalt oli juba mainitud, antud osa on sisend hübriidse mudeli loomiseks ja tehakse katsetus kasutada ainult positiivsed jooned, kuid on arusaadav, et ideaalne tulemus ei ole saavutatav. Erinevate allikate uurides ning *Scrum* metodoloogia koolituse materjalide („Certified ScrumMaster“, 2015) vaadates antud töö autor tuli järeldusele, et väga hea plusside ja miinuste ülevaade (Gilley, 2015) sai tehtud Cliff Gilley poolt, kelle rohkem kui 10 aastane tooteomaniku kogemus võimaldas teha häid järeldusi ning peegeldada peamised plussid ja miinused.

2.8.1. Plussid

Nagu kaskaadmudeli kirjelduses esimesena tehakse plusside ülevaade.

- Äri pool saab esimesed tulemused pidevalt ning vara
- Skoobi muudatuste tegemine on võimalik ja lihtsam, võrreldes *Waterfalliga*
- Tellija tagasiside antakse pidevalt ning sellega on võimalik arvestada projekti suvalisel etapil
- Ei ole vaja teha detailsemad ning põhjaliku analüüsi, et alustada tööd

2.8.2. Miinused

Samuti sarnaselt *Waterfall* metodoloogiaga *Scrum* toob kaasa mõned riskid ning negatiivsed küljed. VersionOne („The 10th Annual State of Agile Report“, 2016) ülevaates tuuakse, et peamine takistus antud raamistiku rakenduses on ettevõte kultuur ning vähene kogemus ning juhtkonna tugi. Aga kuna kogemus on eeldus eduka rakendamiseks, siis on tark õppida varasemate katsetuste näitel:

- Arendusmeeskond ning äri peavad olema teadlikud ja kogenud *Scrum* metodoloogia kasutusel. Vastasel juhul kaose tekkimise oht on kõrge
- *Scrum* sobib väikese ning keskmise suurusega projektidele. Suuremate projektide koordineerimine on väga keeruline
- Halb prognoositavus ning terve projekti ülevaatlikus
- Ettevõte kultuur ei ole sobi antud lähenemisele, kuna *Scrum* toob kaasa palju määramatust

Kokkuvõttes ei saa öelda, et *Scrum* ise on ideaalne ning lahendab ettevõte kõike probleeme, tasub ainult seda rakendada. Nagu iga algatamine see peab olema kaalutud ning analüüsitud samm. Ettevalmistused ning ärilised ja IT inimeste koolitused on eeldus eduka kasutusele

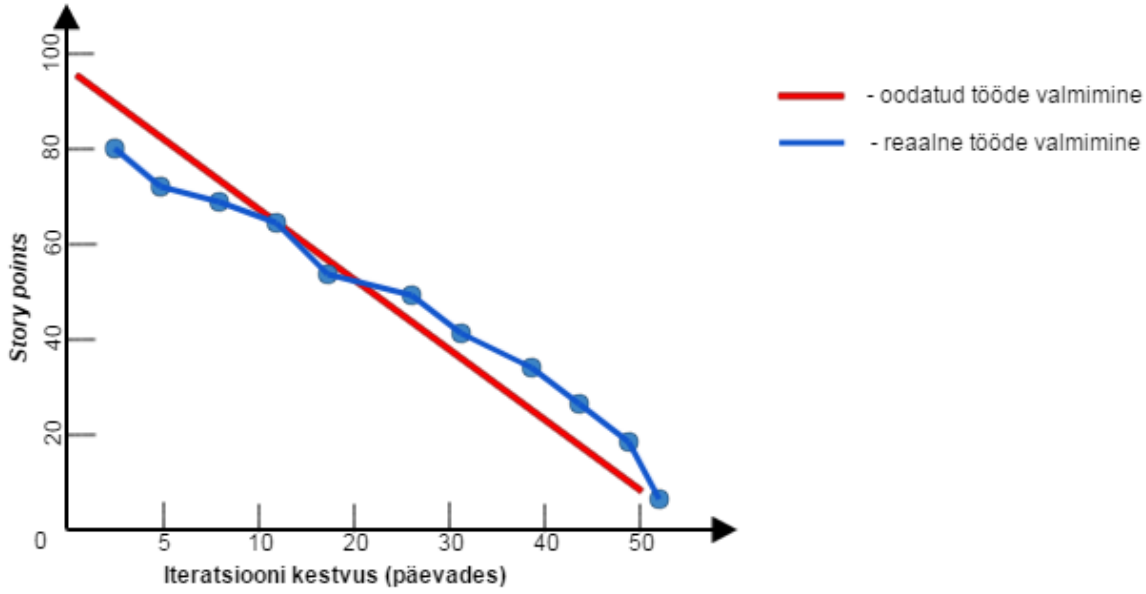
võtmiseks. Kui rakendamine õnnestub, siis samuti, nagu *Waterfall* puhul, on vaja mõõta projekti näidikuid. Selle kaudu on võimalik aru saada, kas edenemine on ootusepärane ja kas mingid muudatused jooksvas protsessis on valikud.

2.9.Mõõdikud

Scrum protsessis meetrikate fookus on selles, et kontsentreerida reliisi peale, tellitud funktsioonide tarnimisel ning hinnangu tegemisel, kuidas arendustiim saab hakkama oma tööga. Teine oluline aspekt on see, et võrreldes kaskaadmudeliga mõõdikud ei saa olla kasutatud meeskondade omavahel võrdlemiseks, kuna oma sisu poolt on subjektiivsed. Peamiste meetrikate list (Putnam & Mayers, 2003), nagu juba mainitud, on ehitatud nii, et ta oleks võrreldav *Waterfall* protsessiga:

- Projekti suurus - vajaliku funktsionaalsuse osade hulk, millised peavad realiseeritud projekti raames
- Projekti kvaliteet - tuvastatud defektide arv ühes sprindis või kumulatiivne defektide arv, et hinnastada terve projekti kvaliteedi
- Projekti kestvus - sarnaselt *Waterfalliga* mõõdetakse kuudes või päevades
- Meeskonna kiirus - kui palju õnnestub funktsionaalsust luua iteratsiooni sees
- Pingutus - *Story pointide (SP)* arv, samuti subjektiivne mõõdik, kuna tehase valik kuidas seda mõõta projekti sees ning ei saa olla kasutatud, et teha võrdlust teiste projektidega. Määrab funktsionaalsuse keerukust

Statistika ka näitab seda, et ettevõtted eelistavad mõõdikud, millised on suunatud arusaamisele, kuidas õnnestub teha kokkulepitud tööd. Näiteks, kõige populaarsemad („The 10th Annual State of Agile Report“, 2016) on kiirus, tööde valmimise tempo ning planeeritud tööde mahv reaalse tempo vastu. Antud näidikute visualiseerimiseks *Scrum* kasutab nii nimetatud “maha põlemise skeemi” (Joonis 5).



Joonis 5. "Põletamise graafik"

Antud skeemi (Dinwiddie, 2009) kasutusel antakse võrdlus kahe parameetri vahel. Esimest saab arvutada projekti alguses ja see on oodatud SP valmimine iteratsiooni käigul. Teist arvutakse jooksvalt, ning see on reaalselt õnnestunud valmistada SP arv. Nende võrdlusel on võimalik ehitada projektisooni ning aru saada, kas jooksev kiirus on piisav või vastav muudatus peab olema tehtud. Antud viis on mugav, kui on soov saada ülevaadet terve projekti kohta. Kahjuks, antud töös käsitletud hübriidse lähenemise sees antud viis ei ole rakendatud, kuna projekti algfaasis sai loobunud alusmeetrikast. Esialgne plaan nägi selle kasutamist. Selle asemel sai korraldatud ajaline võrdlus valmis ning planeeritud tööde vahel funktsionaalsuse kaupa, mis oli loodud antud mõõdiku põhimõtte kasutusel.

Kaks ülaltoodud peatükke on andnud ülevaadet olemasolevate metodoloogiate kohta. Aga maailmas said juba tehtud palju katsetused *Waterfalli* ning *Scrumi* kokku panna. Järgmine alamosa annab vastavate lähenemiste kiirülevaadet ning autor põhjendab, miks just need mudelid ei olnud sobinud püstitatud projekti jaoks.

3. OLEMASOLEVAD HÜBRIIDSED MUDELID

Küsimus, kuidas kokku panna erinevate metodoloogiate plusse, vältides miinuste kaasa võtmist on ka teiste meeskondade probleem. On tehtud mõned pakkumised ning ettevõtted jagavad oma kogemust, kuidas seda õnnestud. Allikate uurides, ei õnnestunud leida ühtegi domineerivat, kuid nende uurimine võib olla kasulik, kui tekkib vajadust sarnast lähenemist pakkuda. Tehtud vead või läbikukkumisi kogemus on ka vajalik sisend.

3.1. Scrummerfall

See on katsetus panna kaskaadlähenemist *Scrumi* sisse. Brad Wilson defineeris seda nagu: "See on *Scrum* ja *Waterfall* kombineerimispraktika mis annab võimalust läbi kukkuda tunduvalt kiiremini, võrreldes *Waterfalli* kasutamisega" (Erickson, 2009). Antud raamistikus arendust jagatakse järgmisel viisil: üks nädal disaini jaoks, 2 nädalat realiseerimiseks ning viimane iteratsiooni nädal testimiseks ning juurutamiseks. Aaron Erickson (Erickson, 2009) nimetas seda maailma halvima lähenemisena. Antud viis sisuliselt jagab kaskaadmudelit väiksemateks tükkideks ning ei muuda protsessi ise. Antud viis ei anna oodatud projekti ülevaadet. Samuti see viis iseennast ei toeta kiirt muudatust projekti läbiviimisel, mis ei toeta püstitatud eesmärke.

3.2. Waterscrum

Kevin Neher defineeris seda nagu: "Kaassõltuvad *Scrum* ning *Waterfall* projektid. Katse mängida Ameerika ning Euroopa jälgpalli samal mänguplatsil samal ajal, mis vaja

lisakoordineerimist ning ajastamist" (Neher, 2009). Sisuliselt seda tähendab, et kasutatakse kaskaadmudeli faaside kontrollpunkte ja nende vastu tehakse katsetus joondama arendust. Arendust aga tehakse iteratiivselt. See lähenemine tundub, nagu sobilik antud töös vaadeldud projekti realiseerimiseks. Suurim miinus on see, et eksisteerib vajadus teha valmis funktsionaalsust iteratiivselt, ning katuseprojekt ei saa seda dikteerida alamprojektide suuna, nagu eeldab *Waterscrum*. Hübriidse raamistiku põhimõtte on selles, et toetada progressi jälgitavust, mitte dikteerida suuna.

3.3. Water-scrum-fall

Antud lähenemises projekti valmimist jagatakse kolmeteks suuremateks faasideks (Mehan, 2012): *water* - projekti plaani valmistatakse kõige alguses ja see on projekti algfaas, *scrum* - iteratiivne plaani realiseerimine, mis sai valmis esimeses faasis, mis sisuliselt tähendab koodi kirjutamist. Ning viimane on *fall* - valmistoode juurutamisefaas, kusjuures seda tehakse organisatsiini eeskirjade kooskõlastusega ning infrastruktuuri piirangutega. Antud viis ei saa ka sobida, kuna organisatsioon näeb oma eesmärgina iteratiivselt ning pidevalt tarnida äripoolele valmis funktsionaalsust, mitte projekti lõpus. Antud viis seda ei võimalda, kuna iteratiivselt luuakse ainult kood, aga juurutamine on ühe tervikuna tehakse lõpus.

Sarnaseid lähenemisi on veel olemas ning esialgne kiiranalüüs näitas, et hetkeolukorra paranemiseks ning püstitatud eesmärkide saavutamiseks ühtegi ei leia, mis võiks 100% sobida ilma muudatuste tegemiseta. Samuti paljude lähenemiste proovile panemine on mahukas algatus. Seega sai tehtud otsus proovida välja töötada oma raamistiku ning asendada selle osasid, kui selgub, et nad ei täida oma rolli. Loodava raamistiku peamised tunnused peavad lähtuma organisatsiooni eeskirjalikke piirangutest. Lähtudes *Waterfall* ja *Scrum* metodoloogiate kirjeldusest, kus on tehtud peamiste plusside ning miinuste ülevaade, hübriidmetodoloogia peab endasse võtta järgmised tunnused:

- Projekt peab olema läbipaistev ning prognoositav
- Projektiplaan peab olema lõplik ning sisaldama kõikide etappide väljundite kirjeldusi
- Projekt on jagatud etappideks, ning äripool peab olema teadlik, millal jõuab valmis tellitud lõplik tarkvara
- Projekti kogumaksumus peab olema teada algfaasis
- Esimesed valmis funktsionaalsuse osad peavad olema üle antud ärile võimalikult vara
- Iga iteratsiooni tulemusena peab olema valmis järgmine potentsiaalselt tarnitav tarkvara osa
- Väikeste skoobimuudatuste sisseviimine peab olema tagatud suvalises projekti faasis
- Detailne ja lõplik analüüs ei ole eeldus, et alustada arendustöödega. Aga piisavalt hästi tehtud, et hinnata keerukust ja maksumust

On selge see, ei ole võimalik võtta mõlematest maailmadest kasutades ainult parimaid tunnuseid. Kõrval efektina tuleb investeerida lisaraha või aega projekti koordineerimiseks. Kuna sarnase lähenemise kogemuse eeldamine arendustiimide poolt ei ole võimalik, siis projekti juhtrühm oli valmis selleks, et mõnede viiside proovi panemisel tuleb muuta protsesse või otsida uut lähenemist. Samuti negatiivse efektina mõned valmis tehtud tööd kuuluvad ümbertegemiseks või üldse ei lähe kasutusse. Siin sai kasutatud printsiip „ebaõnnestu kiiresti“, et teha vajalikud muudatused ka piisavalt kiiresti ning viia projekti stabiilsesse seisundisse nii vara, kui võimalik.

Samuti, et paremini aru saada kuidas luua oma raamistiku on vaja aru saada millised on püstitatud eesmärgid projekti raames. Antud lähtepunktide analüüs on viimane eeldus, et hakata defineerima hübriidlähenemist. Järgmine osa annab ülevaadet nii eesmärkidest, kui ka olukorrast, millises asub ettevõte ning miks tekkis vajadus käivitada antud tarkvara loomist.

4. ÄRILISED EESMÄRGID JA HETKEOLUKORD

Äri strateegilised eesmärgid on defineeritud lähimate aastate jaoks. Organisatsiooni peamine fookus on ettevõte edaspidine kasv. Sellest lähtuvalt arenduse suund on defineeritud ettevõte strateegia dokumendis ning põhilised fookuspunktid on sõnastatud järgmiselt:

- Äri peab kasvama (teenuste punktide hulk kasvab)
- Tootmismahtude kasvu toetamine, mis tõstab vajadust efektiivsemates protsessides
- Lihtsustada teenuste mudelit
- Lihtsustada hinnastamist lõppkliendile
- Tagada teenuste paindlikust

Antud kõrgetasemelised eesmärkide saavutamiseks on vaja ühelt poolt ärilisi muudatusi mudelites ning protsessides, kui ka tugisüsteemide kohandamist või edasiarendamist. Sellest lähtus IT valdkond, kui hakkas looma oma strateegiat. Aga ei saa vaadata ainult tuleviku ning on vaja aru saada mis on praegune seis ja olukord. Selle kaudu on võimalik teostada analüüsi ning aru saada mis on kõige mõistlikum viis edasi arendada ja mis peab olema parandatud. Et defineerida efektiivset IT strateegiat on vaja mainida peamisi punkte, millised kirjeldavad IT praegust seisukorra:

- On antud suur hulk jääksüsteeme, kus funktsionaalsus ei ole alati loogiliselt piiratud. Näiteks, aadresside teenuste süsteemis eksisteerib mitte sellega seotud teenuste tellimusvõimalused
- Olemasolevate süsteemide edasiarendamine on keeruline ning kallis
- Iga uus arendus tõstab olemasolevate süsteemide keerukust nii, et nende haldus on raskendatud hooldusemeeskonna poolt
- Süsteemide koormus ning andmemahud kasvavad nii kiirest, et tipp perioodidel olemasolev infrastruktuur ei ole võimeline tagada teenuste kvaliteedi

Sellest lähtudes sai tehtud mõned arhitektuurilised otsused. Projekti esimene faas näeb ette, et tehakse ringi ainult põhifunktsioonid ning loodud alus on sisend edaspidiste

arenduste ellu viimiseks. Antud visioon on koostatud nii praeguse olukorra parandamiseks, kuid ka näeb oma eesmärgina toetada tuleviku kasvu. IT arhitektuuriline nägemus on defineeritud järgmiselt:

- Osa uuetest tagatoa süsteemidest kirjutatakse nullist, kuna vanade süsteemide edasiarendamine on ebamõistlik
- Loodava süsteemi funktsionaalsus on piiratud moodulite kaupa
- Iga moodul on loogiline tervik ning vastutab ainult oma teenuste eest
- Iga moodul ei tea mitte midagi teistest, mis annab võimalust mitte luua sõltuvusi
- Kuna mainitud sõltuvused puudu, siis tulevikus on võimalik asendada iga moodulit uuega ilma selleta, et teha muudatusi teistes
- Moodulid omavahel suhtlevad rangelt kokkulepitud protokolli kaudu
- Funktsionaalsus on tagatud mikroteenuste kaudu
- Jõudluse nõuded on defineeritud samuti moodulite põhiselt, et tagada terve süsteemi funktsioneerimist
- Kasutajatega interaktsiooni eest vastutab ühine graafiline liides ning äriline kasutaja ei pea teadma millise mooduliga hetkel tehakse tööd

Ülalmainitud punktide silmas pidades sai välja töötatud oma hübriidne projektijuhtimise raamistik. Paljudes ettevõttes eksisteerib hulk reegleid IT arenduse projektide algatamiseks, mis loob piiranguid ning nõudeid tellijale ning teostajale. Ka sellest peab lähtuma pakutud lähenemine. Vastasel juhul käitumine ja tööde teostamine on reeglite rikkumine. Selle vältimiseks tuuakse ka ülevaade selle kohta, kuidas tohib ettevõttes läbi viia IT projekte.

5. IT SÜSTEEMIDE ARENDAMISE KORD

Antud peatükis tuuakse projektide algatamise korra lühikokkuvõtet ning peamised nõuded, millised peavad uued arendussoovid rahuldama. Reeglistiku teadmine ning täitmine ühelt poolt tagab tarkvara loomise kvaliteedi ning teiselt poolt kontrollib investeeringu mõislikust. Kahjuks, antud reeglistiku täies mahus toomine antud töö kaasa ei ole võimalik, kuna dokument on klassifitseeritud nagu „konfidentsiaalne“.

Üks nõuetest on IT arenduste kava väljatöötamine. Arengukava koostakse järgmiseks aastaks ning peab sisaldama iga projekti kestvust ja umbkaudset maksumust. Antud nõue loob piirangut, mis saab rahuldatud ainult kaskaadmudeli kasutamisel. *Agiilsete* projektide juhul algfaasis on keeruline defineerida täpsema funktsionaalsuse komplekti, projekti kogumaksumust ning kestvust. *Scrumi* puhul kestvust saab fikseerida, aga kui selgub, et ei õnnestu luua kõike funktsiooni, siis kokkulepitud osa jääb tegemata, mis ettevõte seisukohalt ei ole aktsepteeritav. Samuti projekti juhtrühma ülesannete hulgas on konkreetse projekti tegevuskava kinnitamine, kuhu kuuluvad eesmärgid, skoop, üldine projektiplaan ja eelarve. Lisaks sellele on defineeritud protsess, millist tuleb kasutada arendusvajaduste teostamisel. Peamised faasid on eelanalüüs, algatamine, arendus, testimine ja juurutamine. Nagu on näha, siis iga projekti katusena ametlikult saab olla ainult *Waterfalli* sarnane arendusprotsess. Aga eeskiri ei loo piiranguid projekti etappide jaoks. Näiteks arenduse sammu saab korraldada paindlikumal viisil. Siin saab tuua analoogiat arenduspartnerite kasutamisega. Ettevõttel on mõned koostööpartnerid, kelle poolt tellitakse ka projektid tervikuna. Selleks, et algatada arendust, on vaja läbi viia esimesed faasid ja pärast partner tegeleb otseselt koodi kirjutamisega. Tellijale ei ole teada, kuidas on teostatud arendusprotsess partnertiimi sees ja see ei ole oluline. Peamine on see, et tellitud tarkvara on valmis õigel ajal ja kokkulepitud eelarve ja funktsionaalsuse sees.

Nõutud dokumentide hulgas on toodud ka tasuvusarvutus. Antud analüütiline arvutus annab ülevaadet sellest, kuidas tehtud kulud projekti algusfaasis ajaga hakkavad tooma tulusid tulevikus. Kui projekti eluiga saab määratud, kus vaikimisi väärtus on 5 aastat, siis saab teha tuleviku prognoosi. Antud väärtuste kasutusel arvutatakse näidikuid nagu IRR

(sisemine tasuvus) ja NPV (praegune puhasväärtus). NPV on tulevaste rahavoogude nuudis väärtuste summa, millest on maha lahutatud esialgne investeering . Seega kui antud väärtus on rohkem kui 0, siis investeering toob ettevõttele rahalist väärtust. IRR on projekti tegelik kasuminorm ehk projekti sisemine tulumäär ja see näitab, millise rentaabluse projekt tegelikult annab. Sisemine tulumäär näitab, mitme protsendi võrra aastas kasvab keskmiselt projekti paigutatud kapital. Sisemise tulumäära näitaja väljendatakse protsentides (Zeiger, 2013).

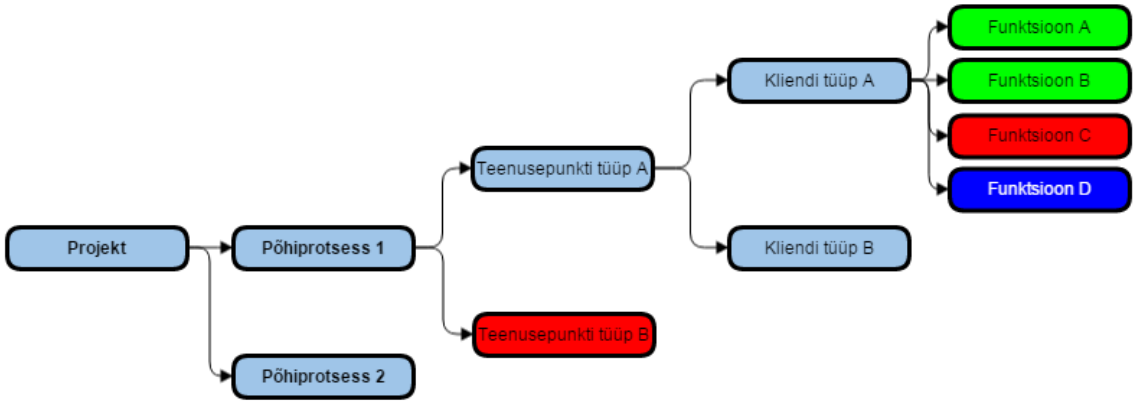
Antud peatüki kokkuvõttes saab öelda, et antud ranged piirangud on õigustatud, kui ettevõte tunneb vastutust tehtavate investeeringute eest. Juhtkond otsuste tegemisel tahab teha seda paljude sisendite kaaludes ning riskide hindades. Vastasel juhul saab rakendada *start-up* mudelit, kus proovitakse teha arendust ja vaadata selle mõju turule ning klientidele. Aga kui tegu on, näiteks, riigi ettevõttega, siis antud lähenemine on kehtestatud reeglite vastu ning peab olema kasutatud formaalne ja prognoosi võimeline arendusprotsess, mida pakub *Waterfall*. Antud töö eelmised osad andsid ülevaadet millised lähenemised on võtnud alusena ja millistes situatsioonides nad on head. Pärast sai tehtud ülevaadet millistest printsiipidest lähtub ettevõte arendusprojektide teostamisel ning millises seisus on praegused IT süsteemid. Lähtudes sellest on võimalik defineerida hübriidset lähenemist, mis konsolideerib endas nii *Waterfall*, kui ka *Scrum* tunnuseid.

6. HÜBRIIDNE RAAMISTIK

Ettevõte vajaduste rahuldamiseks sai loodud oma lähenemine, mis peab endasse võtma hulk tunnuseid, millised said kirjeldatud peatükkis „OLEMASOLEVAD HÜBRIIDSED MUDELID“. Loodud projektijuhtimise raamistik peab defineerima kuidas seadistada ning mõõta arendustiimide vahelist tööd. Iga alampeatükk annab ülevaadet arendustööde korraldamisest ning lähtudes kogemusest tehakse järeldusi, kas see oli edukas katsetus või tuli teha vastavaid jooksvaid parandusi. Pakutud lähenemine ei ole ideaalne ning mõned aspektid nõudsid printsipiaalset muutmist. Aga kirjeldatud ebaõnnestusid on oluline sisend teistele ettevõtetele, mis näeb vajadust rakendada sarnast protsessi.

6.1. Projekti skoop

Projekti algusfaasis oli vaja defineerida minimaalset elujõulisi skoobi (edaspidi MVP), et ootused äri poolt saaksid defineeritud, loodav funktsionaalsus oleks kirjeldatud kõrgemal tasemel ning algne sisend analüüsi jaoks omaks oma piire. Vastasel juhul eksisteerib oht, et nõuete korjamisel tekkib liiga palju sisendit ilma täpsemate piiranguteta ning projekti skoop on liiga suur realiseerimiseks mõistliku aja jooksul. Äriga sai tehtud kokkulepe, et esimesena realiseeritakse ainult äriliselt kriitilised funktsioonid ning teised peavad tulema hiljem. Et tekitada konstruktiivset arutelu, sai korraldatud lühianalüüs, et uurida välja millised peamised protsessid eksisteerivad ettevõttes. Kui protsessisammud on teada, siis uuritakse põhifunktsioonid, millised peavad olema toetatud loodava süsteemi poolt. Kuna teemade arv on päris suur, siis konstruktiivse arutelu tekitamiseks sai valitud graafiline formaat. Antud formaadi kasutusel tekkis koondülevaade ning osapooled saaksid lihtsasti viidata mingi alamosa peale ja teha vastava ulatuse kokkulepet. IT analüütikute poolt sai loodud funktsioonide kaart (Joonis 6).



Joonis 6. Funktsioonide kaart

Antud viis näitas enda väga hästi ning saadud tulemus in seni (aprill, 2017) kasutusel. Värviliste kodeeringu abil sai näidatud, mis on IT valdkonna visioon ja millised funktsioonid või äri teenused võiksid olla MVP sees või väljas. Värvid omavad järgmist tähendust:

- Roheline - näitab seda, et funktsioon või kaardi osa kuulub MVPsse
- Punane - näitab seda, et funktsioon või kaardi osa ei kuulu MVPsse
- Tumesinine - funktsioon või kaardi osa nõuab lisaanalüüsi, hetke teadmiste tase on liiga madal, et lõpliku otsuse tegemiseks

Arutelu on kestnud 4 päeva, kus osalesid ettevõte juhtkond, iga ärivaldkonna võtmeesindajad, IT Arenduse, Hoolduse ja Arhitektuuri valdkondade analüütikud ning äri võtmeesindajad. Iga samm oli esialgu tutvustatud ärianalüütiku poolt, et maandada riski, et vale arusaamise tõttu tehakse vale otsus. Tutvustuse pärast tekkis arutelu ning sai tehtud lõplik otsus, kas antud arendusvajadus kuulub MVPsse või mitte. Samuti oli kehtestatud reegel, et kui praegu antud funktsionaalsust ei eksisteeri, siis seda tehakse hilisemates etappides, mitte esimeses. Kuna mõned teemad olid liiga keerulised, et neid arutada funktsioonide tasemel, siis sai loodud hulk tugimaterjale. Antud materjalide peamine fookus oli teenuste mudelitel ning hinnastamisel. Materjalide eesmärk oli näidata millised muutused toimuvad teenustes ning sai tehtud võrdlus tänapäevase olukorraga.

Kokkulepe tegemisel seda pandud protokolli, mis edaspidi kasutati üldvisioonina. Kuna toimunud arutelus osales ainult piiratud hulk inimesi, siis tuli teha laiema tutvustust, et terve ettevõte oleks ühel informatsioonilisel lainel. Järgmise etapina sai loodud detailsem projekti visiooni dokument. Kuna dokument oli ametlik väljaanne, siis tuli katta ka laiema ringi teemasid. Dokumendil oli järgmine struktuur ja alamosad:

- Projekti etappide ülevaade - milline on projekti plaan, mida tähendab MVP ettevõte jaoks, põhilised eesmärgid
- Tulevased muudatused ettevõttes - ülevaade, kuidas teenuste ning hinnastamise uuendatud mudelid mõjutavad terve ettevõtet ning protsesse
- Projekti skoop - millised peamised teemad on MVPs ja millised mitte
- Põhielementide ülevaade - funktsioonide grupid ja nende eesmärgid, mis andis alust luua modulaarset struktuuri
- Riskid - ülevaade nii ärilistest, kui ka IT riskidest. Samuti oli toonud meetmed, kuidas riske maandada

Riskide osast tasub eraldi tuua ühte nendest - ärianalüüs ei ole võimeline kujutama kõik ärilised vajadused või vajadused muutuvad koodi kirjutamise faasil. Nagu meede sai pakutud kasutada *agiilset* arendusmeetodit. See pakkumine võimaldas juurutada *Scrum* protsessi ametlikult esimene kord ettevõttes.

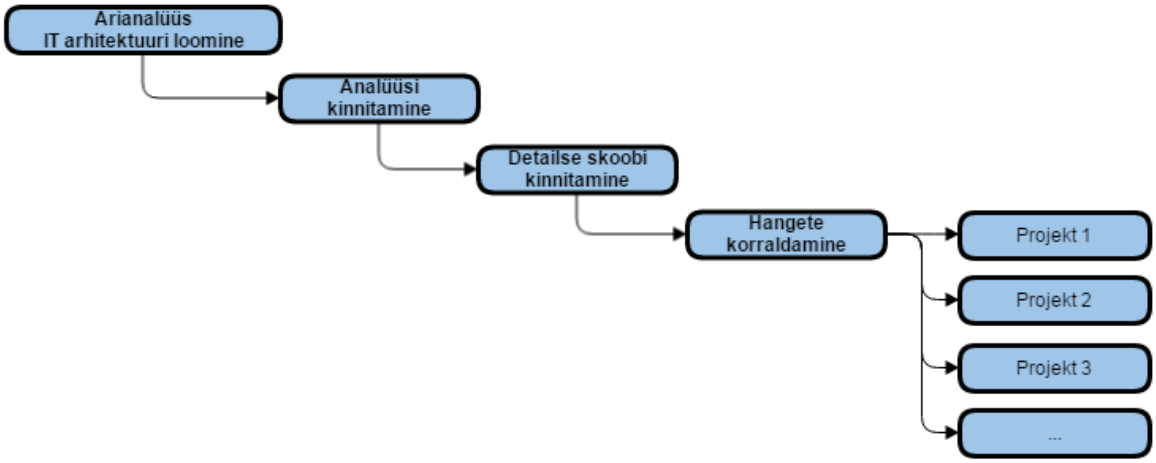
Oluline aspekt on ka see, et kuna sai tehtud eeldus, et tuleb korjata detailset ärilist sisendit terve projekti jooksul, siin on vaja pidevalt kaasata äri esindajaid. Et äri pool saaks ka oma tööd organiseerida, siis iga teema või funktsioon said oma ärilist vastutajat. Mainitud esindaja peamine eesmärk on organiseerida sisendi korjamist, tulemuste kinnitamist ning vastuvõtu testide läbiviimist. Iga esindaja jaoks sai määratud projektis osalemise FTE (täistööajale taandatud), et tema ajaline ressurss oleks tagatud ning IT analüütik saaks teda kaasata õigel ajal ja mahus. Siin saab tuua ka tulemust, et äriline pool oli väga motiveeritud ning hea meelega andis nii palju sisendit, kui oli vaja. Paljud ärilised inimesed said osaluse 0.4 FTE, mis tähendab, et igapäevase töö jaoks jääb vähem aega ning tuleb väga hoolikalt juhtida oma plaani. Rohkem probleemi tekkis, kui tuli kaasata korraga mitu esindajaid, et teha vajaliku kooskõlastusi. Aga isegi siin projekti aasta jooksul ei tekkinud ühtegi suuri

konflikti. VersionOne uuringus („The 10th Annual State of Agile Report“, 2016) on toodud, et *agiilsete* raamiskitu kasutusele võtmise barjäär 28% juhtudel ongi äriesindajate halb kättesaadavus. Antud kokkulepe projekti alguses peaaegu täielikult elimineeris antud riski.

Kokkuvõttes, välja arvatud mainitud riski, antud lähenemine ja esimene projekti faas on kaskaadmudeli rakendamine. Tekkib fikseeritud ning pikk plaan ja skoop, mille muutmine suuremate funktsioonide tasemel ei ole lubatud. Järgmise osana sai otsustatud, et kuna funktsioonide grupid on teada ja neid saab jagada arhitektuuriliste moodulite vahel, siis on vaja detailiseerida analüüsi, detailiseerida plaani ning juurutada sobiliku arenduse raamistiku.

6.2. Projektiplaan

Kuna skoop ning tähtajad olid fikseeritud, siis MVP detailse projektiplaani tegemine on oluline, et kontrollida progressi ning vajadusel teha muudatusi - kas ressurssides või skoobis. Selleks oli määratud eraldi projektijuht, kes konsolideeris ärilist sisendit, väga kõrgetasemelised arendusmeeskondade hinnanguid ning koostas plaani. Mainitud dokumendi struktuur oli *Waterfalli*’ põhine ning nägi seda, et iga suur etapp peab olema lõpetatud enne, kui saab jätkata järgmisega. Peamised sammud olid defineeritud tee kaardil (Joonis 7).



Joonis 7. Projektiplaani struktuur

Nagu kaardilt on näha, siis visiooniloomise pärast korraldatakse ärianalüüs. Analüüsi eesmärk ei olnud uurida kõige detailsemal tasemel ärilisi vajadusi, kuid pidi olema nii täpne, et võimaldada luua hanke dokumentatsiooni. Selle loomine abstraktsel tasemel toob riski, et pakkumiste hinnad on kõrgendatud või liiga ebatäpsed. Antud tegevusega paralleelis oli korraldatud ka IT detailiseeritud arhitektuuri loomine, et modulaarne struktuur oleks täpsemalt paigas.

Analüüsi valmimisel oluline faas on veel kord üle vaadata projekti skoobi ning kinnitada saadud tulemusi äri poolega. Selle mittetegemise juhul võib tekkida risk, et ärilised ootused ja reaalne plaan ei ole kooskõlas, mis on alus konfliktide tekitamiseks. Antud analüüsi käigul suuri muudatusi ei tulnud, kuid aitas tuua palju vajaliku detaile.

Kui tekkis arusamm, et mingi konkreetse teema teadmiste hulk on piisav, et luua hankedokumentatsiooni, siis koheselt alustati vajaliku dokumendi loomisega. Eesmärgiks oli iga mooduli jaoks leida sobiliku arendustiimi nii kogemuste, kui ka võimekuse mõttes. Hange dokumendid ei olnud piisavalt detailselt selleks, et koheselt luua vajaliku süsteemi ja see oli oodatud tulemus. Seega iga hange esimene faas pidi sisaldama ainult süvendatud ja fokuseeritud analüüsi, et tekkiks lõplik teadmine, kuidas moodulit luua.

Hangetulemusena käivitatakse eraldi alamprojekt, mis sisuliselt tähendas mooduli ehitamist. Nagu on näha, enne antud hetkeni projekt on *Watefall* põhine. Aga ootus on see, et edaspidi iga mooduli arendus on iseseisev projekt ning nad kõik liiguvad paralleelselt.

Antud etapi koordineerimine oli edukas ning kõik hanged said tehtud õigel ajal ning vajaliku sisendiga. Samuti õigel ajal oli valmis ka IT arhitektuur, mis toetas korrektse projekti häälestuse tegemist tehniliselt poolt. Selle kaudu iga uus meeskond koheselt sai kätte vajaliku tehnilist sisendit kuidas tööd korraldada ja millistest paradigmadest lähtuda. Projektijuht hakkas siin luua mõõdikuid, et jälgida edaspidist progressi. Mõõdikute komplekt oli määratud järgmiselt:

- projekti suurus - funktsioonide arv
- projekti kestvus - iga etapp eraldi, päevades. Siin sai jälgida oodatut ning võrrelda tegeliku kestvusega
- Iga mooduli ja etapi maksumus - selle abil saaks hinnata projekti kogumaksumust ning võrrelda määratud eelarvega. Projekti terve maksumuse jaoks sai tehtud üldine kood ning iga alamprojekt sai oma koodi ja amortisatsiooni perioodi

Antud faasi raames reaalset progressi valmistoode mõttes veel ei olnud, õnnestunud ainult saada tulemust kuidas liigub analüüsi faas.

Et siduda projektiplaani ettevõte teiste projektidega, siis sai tehtud „masterplan“. Ta kuvas endas juba nimetatud *roadmapi* ning lisaks sellele sisaldas iga suure projekti kõrgetasemelised etapid. See võimaldas mitte minna konflikti teiste ettevõte algatustega, näiteks paremini ühtima strateegia projektiga.

6.3. Hangete korraldamine

Kui plaan on paigas siis järgmine loogiline etapp selle ellu viimine teostamine. Reaalsus on see, et tööde maht ning ajaline piirang nõuavad ettevõte sisenemise arendustiimi

laiendamist kordades. Kuna oli selge, et organisatsioonis on vähe inimressursse ning vajalikud kompetentsid olid puudu, siis sai tehtud otsust korraldada riigihankeid. Hangete süsteem on mitmetasemeline. Esimesena oli korraldatud avalik hange, et otsida ainult vajalikud kompetentsid. Selle tulemusena tulid teenusepakkujad ning tekkis valik arendajatest, testijatest ning analüütikutest. Hindamiskriteeriumid olid vajalikute tehnoloogiate kompetentsid ja kogemus.

Järgmise etapina said korraldatud minihanked iga alamprojekti jaoks eraldi. Alati esimene minihange oli mooduli analüüsi teostamine ning tulemusena pidi sündima analüüsidokument, mis räägib sellest kuidas on vaja ehitada antud moodulit ja mis on antud alamprojekti ulatus. Iga raamlepingus esinev partner oli võimeline teha oma pakkumisi analüüside teostamiseks. Valiku kriteeriumid olid samuti konkreetse vajaduste rahuldav kompetents, kogemus ja lisaks ka hind. Kui see sai tehtud ning analüüs valmis, siis tehakse otsus, kas valitud tiimi tulemused rahuldavad tellijat. Positiivse tulemuse korral sai sõlmitud juba moodulipõhine raamleping ühe *Sprinti* jaoks, mis kestab 2 nädalat. Kui antud iteratsiooni tulemused on samuti rahuldavad, siis vastav leping pikendatakse ja meeskond on võimeline jätkata. Samuti leping näeb ette võimalust asendada tiimi spetsialiste ning partner pidi pakkuma järgmiseks *Sprintiks* asendajat. Antud konstruktsioon oli suur koormus ja väljakutse hankeosakonna jaoks. Aga tulemusena sündis nii paindlik protsess, et said valitud just vajalikud ning parimad kompetentsid. Probleemide puhul said tehtud kiired asendused ilma selleta, et teha muudatusi lepingutes või peatada neid.

Võimalike partnerite tööülesannete andmiseks said tehtud hankedokumendid. Antud dokumentatsioon kuvas endast kõrgetasemelist analüüsi ning nägi oma eesmärgina suunata uut arendusmeeskonna teha süvendatud analüüsi. Dokument pidi olema lühike ja selge, et võimaldada kiiresti reageerida hankele. Struktuur sai kokkulepitud alguses ning iga ettevõte sisene analüütik kasutada antud mustrit, et maandada riski, et oluline osa jääb mainimata. Samuti oli tehtud esimene näidis, et kergendada edaspidist tööd. Hankedokumendi struktuur koosneb järgmistest peatükkidest:

- **Ülevaade** - antud peatükk pidi vastama küsimusele miks antud moodul on äriliselt vajalik, milliste äriliste andmetega ning üksustega hakkab opereerima ja mis on

oodatud kasutuse kogemus. Viimane osa reeglina rääkis sellest, kui suur päringute ja andmete töötluse koormus on ettenähtud, et tagada õige struktuuri loomist. Oluline osa on ka defineerida täpsemalt ärilised eesmärgid

- **Kasutamise juhtumid** - siin sai kasutatud standardne *use case* metoodika, kus näited illustreeritakse ärilise rolliga ning soovitud tegevustega, millised peavad olema tagatud tellitava tarkvara poolt
- **Omadused** - mis on peamine mooduli funktsionaalsus
- **Andmed** - kuidas täpsemalt andmed liiguvad mooduli sees ja moodulite vahel. Millised andmed peavad olema töödeldud ärilisest vaatenurgast ja kuidas. Andmebaasi struktuuri nägemus
- **Nõuded** - peamiselt mittefunktsionaalses nõuded, jõudluse ja koormuse prognoosid, versioonide loomine ja muud sarnased teemad

Iga dokument, valmimise pärast, oli üle vaadatud tooteomanikute, IT Arenduse valdkonna juhi ning IT arhitekti poolt, et saada lõpliku kinnitust. Tulemusena peaaegu kõik partnerid said valitud kvaliteetselt. Tööd valmivad lubatud ajal ning lubatud mahtudes. Juhtudel, kui lubadusi pidamine sai probleemiks, siis asendus ei tekitanud ajalist viivitamist. Antud sanktsiooni tuli rakendada 2 partneritele. Koordineerimise poolt projekti alguses antud teemaga tegelesid mitu inimest täisajal. Seda tähendab, et antud lähenemine oma paindlikkuse poolt nõuab palju koordineerimist. Edaspidi antud koormus läheb alla, aga saadud paindlikus on hea investeering tulevaste probleemide lahendamiseks.

6.4. Projekti *Backlog*

Esimeste hangete korraldamise ajal tootejuhtide grupp paralleelselt tegeles sellega, et koostada üldisema projekti *Backlogi*. Antud tegevus oli oluline sellepärast, et peab olema tsentraalselt hallatav koht, kus ühelt poolt on terve projekti ülevaade ning ka seosed moodulite vahel pidid olema ka tuvastatud. Moodulite analüüsid ei olnud võimelised seda

katta. Samuti seoses vanade süsteemide asendamisega tekkis vajadus teha muudatused ka olemasolevas tarkvaras, mis jääb töös MVP faasi pärast. Siia kuuluvad suured teemad nagu integratsioonid ja andmete migratsioon. *Backlog* lähtus äriliste protsesside kirjeldusest, mis tulevikus on toonud ka vajadust drastiliselt süvendada protsesside analüüsi. Selgus, et olemasolev teadmine ei ole piisav lõpliku ulatuse defineerimiseks. *Backlog* sai loodud nagu Exceli tabel, mis sisaldas endas:

- *Topic* - kunstlik jaotus, loodud ainult selle jaoks, et loogiliselt grupeerida teemasid, näiteks "kliendihaldus"
- *Backlog item* - äriline vajadus, eksisteerib peatükki tasemel, näiteks "kasutajate õiguste haldus"
- *Description* - annab sisendit igale *itemile*. Peamine reegel on see, et peab olema kirjutatud ärilises keeles ja peab sisaldama nii palju teksti, ei vältida mitmetähenduslikust
- Priority - sai kasutatud lihtne süsteem, kus prioriteedid MVP-1, MVP-2 ja MVP-3 määrasid realiseerimie järjekorra. Väärtus 4 tähendas, et antud vajadus peab olema realiseeritud koheselt MVP pärast ja on kriitiline funktsionaalsus, 5 - madalam kriitilisusega nõue, 6 - madalaim prioriteet, vajab üle vaatamist ning eraldi kinnitamist
- Complexity - *story pointid* saadud arutelu käigul, mis oli korraldatud arendusmeeskonna sees. Nagu on juba mainitud antud töös, see on subjektiivne mõõdik. Skaalaks sai valitud Fibonacci arvude vahemik, kus 1 tähendas "väga lihtne asi, tehtav päevadega", kuni 21, mis tähendas "liiga suur tükk, et ole võimalik teha ühe *Sprindi* raames, vajab tükeldamist"
- Owner - IT poolne vastutav isik, kelle ülesanne anda lisasisendit antud teemal või koordineerida äriga kooskõlas lisateadmiste korjamist
- Status - kas antud Backlog item on analüüsifaasis, töös, testimisel või valmis
- Main module - mis moodulis peab olema realiseeritud antud funktsionaalsus

- Related modules - seosed teiste moodulitega, peamiselt sai kasutatud selleks, et defineerida millised teenused ja andmed on vaja tagada, et vajalik funktsiooni töö oleks võimalik

Ajaga tekkis arusamm, et paljudel tiimidel oli tõsine kahtlus, kuidas iga teema on seotud teistega ja kas kindlasti kõik vajadused on kaardistatud. Realiseerimise algfaasis samuti esimene kogemus näitas, et toodud vajadused saab teha isoleerides, aga pärast selleks, et neid kokku panna ja korraldada tööd ühe tervikuna, on vaja teha lisaarendusi teenuste tasemel. Ärile samuti oli raske hinnata, kas kõik nende vajadused on kaetud ja kuidas sobivad kokku kaardistatud teemad. Antud samm ja viis näitas, et nii suure ulatusega projekti raames *Scrum Backlogi* kasutamine on keeruline ning mõnikord tekitab rohkem segadust, kui toob kasu.

Loodud *Backlog* ei olnud ainult staatiline vajaduste hoidla, selle lisaks said mõõtmised rakendatud. Esiteks sai hinnatud iga mooduli keerukus. Selleks sai arvutatud iga mooduli *Story pointide* summa. Lisaks olid arvutatud järgmised parameetrid:

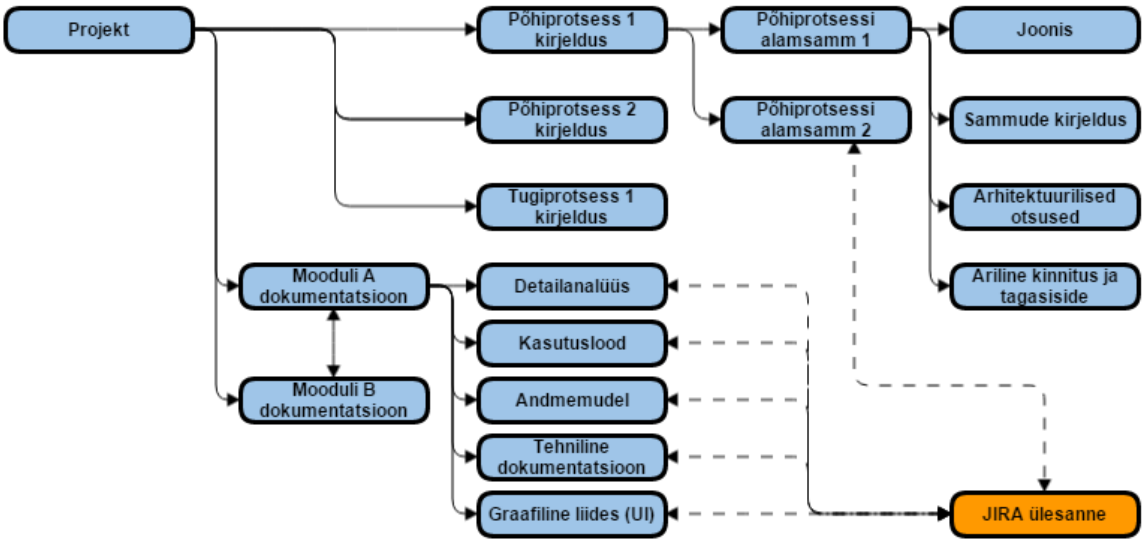
- Punktide kogusumma, välja arvatud *itemid* väärtusega 21
- Punktide kogusumma, kaasa arvatud *itemid* väärtusega 21
- Mooduli skoor - Punktide kogusumma, kaasa arvatud keerukust 21 (MPV1 + MVP2 + MVP3)
- *Backlog itemite* arv, kus väärtus oli võrdne 21-ga
- Mooduli määramatus – kui palju moodul sisaldab teemasid, kus keerukuse väärtus on võrdne 21, väljendatud protsentides. Kui tulemus oli rohkem kui 0%, siis sai rakendatud samuti punane värviline indikatsioon, mis näitab, et tuleb teostada lisaanalüüsi
- Kiirus - kui palju tööpäeva on vaja kulutada ühe mooduli SP valmimise jaoks. Selleks sai arvutatud jäänud tööpäevade arv ning jagatud moodulite maksimaalse skooriga. See on näidik, mis näitas, kas reaalselt arendusmeeskonnad saavad hakkama punktidega plaanijärgselt või on vaja suurendada võimekust

Tabeli sees sai kasutatud ka värviline indikatsioon, mis andis esimest hinnangut, kui keeruline on mooduli arendus tervikuna. Kahjuks, mainitud kahtluste tõttu, tuli loobuda antud dokumendist ning mõõdikud jäid kasutamata.

Kokkuvõttes, vaatamata sellele, et palju tööd oli tehtud antud faili loomiseks, sai tehtud otsus, et tehakse sügavam detailne protsesside analüüs. Esimeste „teemade“ valmimine näitas, et nad on väga fragmenteeritud ning mõnikord ei sobi kokku. Ajaga antud risk ainult suureneks ning ei olnud võimalik vältida võimalust, et mingi oluline teema jääb katmata. Selle vältimiseks, et midagi jääb kaardistamata, sai korraldatud ettevõte detailne äriprotsesside analüüs. Oluline aspekt on saadud teadmisi säilitamine ja arendus ülesannete valmimiseks kasutamine. Selleks sai juurutatud oma struktuur ning protsess, millest lähimalt räägib järgmine peatükk.

6.5. *Jira* and *Confluence* seadistus

Analüüsi käigul sai korjatud palju kirjaliku materjali. See vajab struktureeritud hoidmist. Kuna ettevõttes olid juba kasutusel *Atlassian* tooted, siis otsustatid jätkata *Confluenciga* kirjaliku materjalide haldamiseks ning *Jiraga* arendusülesannete juhtimiseks ning progressi jälgimiseks. Samuti kuna *Jira* kirjeldused reeglina on lakoonilised ja hea praktika on kasutada *Confluence* viitamist, siis tekkis probleem kuidas efektiivselt siduda neid omavahel (Joonis 8). Teine aspekt, kus oli vaja tegeleda keerukusega on suur ülesannete hulk. Kui neid panna ühte listi, siis jälgida progressi on keeruline.



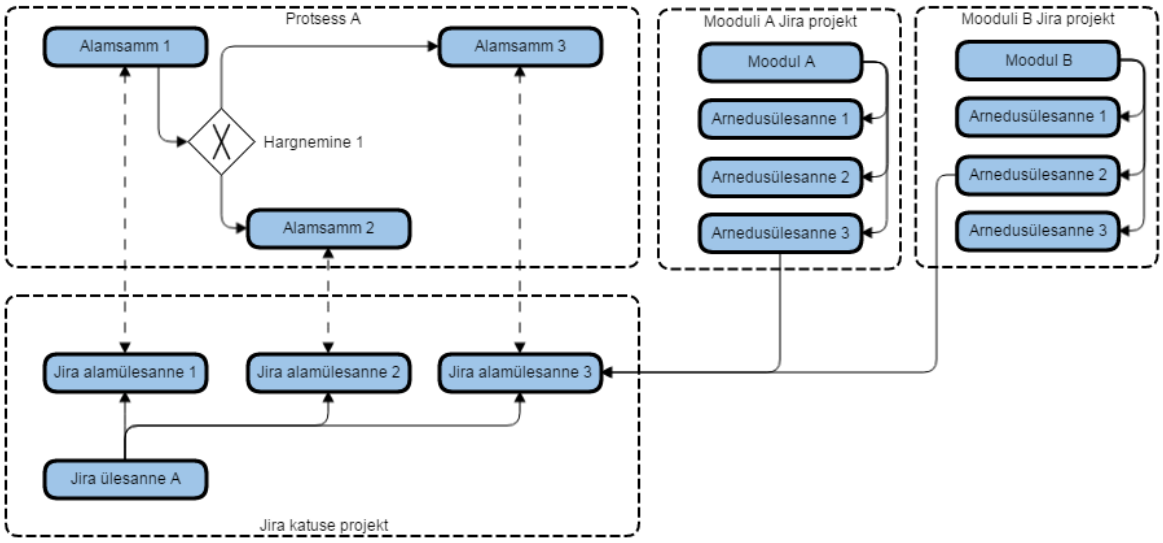
Joonis 8. Jira ja Confluence seosed

Confluencis sai tehtud uus "ruum", et mitte segada uuesti tekkinud dokumentatsiooni vanemate projektidega. Äriprotsesside kirjelduse jaoks sai tehtud jaotus, et iga protsessi alamsamm sai eraldi alamlehte. Navigeerimine oli organiseeritud samas järjekorras, nagu funktsioneerib äriline protsess, mis lihtsustab materjalide kasutamist. Iga protsessi alamsammu lehel tehakse detailne kirjeldust pildiga, mis annab graafilist sisendit. Selleks sai kasutatud BPMN notatsioon („Business Process Model and Notation", 2011). Samuti iga alamprotsessi sammude kirjelduse andmiseks sai koostatud tabel, kus oli toodud antud sammu sisend, väljund, äriline tähendus, seosed moodulitega ja milline funktsionaalsus peab olema tagatud. Kokku tuli rohkem kui 700 mainitud samme. Lisaks oli korraldatud arhitektuuriline ümarlaud, kus sai protokolli ka IT arhitektide arvamus, kuidas realiseerida antud protsesse tarkvara vaatenurgast.

Iga mooduli omanik oli vastutav selle eest, et vedada oma mooduli dokumentatsiooni. Kasutuslugude abil pidi olema defineeritud mooduli *Backlog*, mis edaspidi jagatakse osadeks ning iga iteratsiooni sees peab olema tehtud mingi kokkulepitud osa eeldusega, et mooduli terve funktsionaalsus peab olema valmis põhiprojekti lõpuks. Sai tehtud kokkulepe, et mooduli *Backlogi* loomisel alguspunktina võetakse äriliste protsesside kirjeldus, mis garanteerib seda, et ärivajadused on kaetud mooduli poolt.

Jiras sai tehtud katuse projekt, mille eesmärk on üldisema progressi jälgimine. Selleks oli esitatud struktuur, kus protsessi jaoks sai tehtud eraldi *Jira* ülesanne. Iga protsessi alamsammu jaoks sai tehtud ka *Jira* alamülesannete hulk. Tulemusena äriprotsesside struktuur sai ülekantud *Jirasse* üks ühele (Joonis 9). Iga alamülesanne sisaldas kirjeldust, prioriteedi ning staatust. Staatused said defineeritud nagu: kuulub teostamiseks, töös, testitud, valmis. Arvesse võttes, et antud alamülesannete arv samuti ületas 700, tuli esitada kasutamiseks lihtsat struktuuri.

Kui vastava mooduli analüüs sai valmis, siis selle omanik korraldab oma meeskonna tööd ja loob arendusülesanded samuti *Jiras*. Pärast tuli luua seosed katuseprojekti ning mooduliprojektide ülesannete vahel. Seega alati oli näha, millises valmimise staadiumis asub iga protsesside alamsamm mooduli funktsionaalsuse kaudu.



Joonis 9. Jira ja protsesside seosed

Et anda paremini ülevaadet loodud struktuuri kohta tuuakse ka abstraktne näide. Näiteks, äriprotsessi A alamsammu 3 vastab *Jira* alamülesanne 3. *Confluencis* protsessi kirjeldus ja *Jira* ülesanne kirjeldused täiendavad üks teist, seega äriliselt sisend on täielik. Mooduli A omanik teab, et antud alamsammu funktsionaalsuse tagamiseks tuleb valmis arendada tehnilist ülesannet 3. Mooduli B omanik teab, et tema poolt on vaja valmis teha arendusülesannet 2. Kui nüüd mõlemad moodulid saavad valmis kõik seotud ülesandeid

ning äri kinnitab, et on tagatud vajalik funktsionaalsust, siis katuseprojekti alamülesanne 3 saab staatust “valmis”. Mis omalt poolt tähendab, et äriprotsessi samm on kaetud IT süsteemi funktsionaalsusega.

Vaatamata sellele, et *Jira* peamiselt kasutatakse *agiilsete* projektide jaoks, antud viisil õnnestunud luua seost äriliste protsesside vahel, toetada neid vajaliku kirjeldusega ning jälgida progressi moodulite põhiselt, kui ka üldisemal tasemel, protsessi põhiselt. Peamine keerukus tekkis momendil, kui tuli siduda moodulite ülesanded vastavate protsesside sammudega. Kui antud seadistus sai tehtud, tekkis järgmine vajadus - korraldada tööprotsessi nii, et oleks teada mis milline peab olema arenduse valmimise järjekord ning kuidas ühed moodulid mõjutavad teiste moodulite valmimist. Selle parema ülevaade andmiseks järgmisena tuuakse moodulite sisene töökorralduse kirjeldus.

6.6. Moodulite meeskonnad

Kuna iga moodul on keeruliste funktsioonide hulk ning nõuab spetsiifilisi kompetentsi, siis sai tehtud otsus, et nende eest vastutavad alati eraldi tiimid. Tiimide komplekteerimisest rääkis hangete korraldamise peatükk. Iga meeskond teostab oma tööd *Sprinti* põhiselt ning iga iteratsiooni tulemusena peab olema valmis osa funktsionaalsusest. See on *Scrumi* lähenemise tunnus. Hübriidne lähenemine meeskondade tööde korraldamiseks seisnes selles, et iga esimene hanke nõuab analüüsi, mis on kindel *Waterfall* protsessi faas, aga jätkata oli võimalik *agiilselt*.

Arendusmeeskonna juhtimiseks sai kasutatud roll nagu - tooteomanik, mis on pärit *Scrum* maailmast. Selle kaudu tagatakse äriliste vajaduste peegeldamine iga konkreetse mooduli peale. Samuti tooteomanik on vastutav selle eest, et kokkulepitud tööd on õigeks ajaks valmis, koostab mooduli projektiplaani ning jälgib progressi, mis on sarnane *Waterfall* projektijuhiga. Antud roll oskuste mõttes on nõudlik, kuna korraga peab olema teostatud

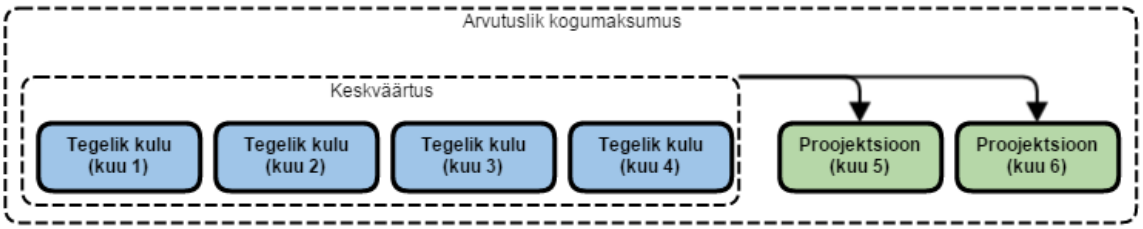
nii detailne analüüs, kui ka meeskonna tööde korraldamine. Aga vastasel juhul eksisteeris oht, et ettevõte arendusvajaduste huvid ei ole piisavalt peegeldatud mooduli peale.

Kui esimene mooduli *Sprint* on valmis ning *Backlog* defineeritud, siis mooduli meeskond hakkab teha tööd, kuhu kuulub ka koodi kirjutamine. Esimesena võetakse tööd, millised peavad tagama mooduli funktsioneerimist: üldine seadistus, andmebaaside loomine, IT arhitektuuri detailne loomine ning infrastruktuuri välja töötamine. Kui antud etapp on valmis, siis töö jätkub nagu seda näeb tavaline *Scrum* protsess. Kooskõlas äriga võetakse kõige prioriteetsemad teemad ning tähtsuse järjekorras hakatakse neid tegema. Selleks, et aru saada kui palju funktsionaalsust saab olla tehtud ühe iteratsiooni valmis tehakse *Sprint* planeerimine. Arendustiim annab oma hinnanguid ning koos tooteomanikuga lepitakse kokku ajakava. Kui ajaline vaade on tehtud, siis vastav informatsioon tehakse nähtav teistele tiimidele, et nemad saaksid sellest lähtudes korrigeerida või luua oma plaane.

Kui *Sprint* sai läbi, siis iga meeskond peab esitama IT arenduse valdkonnajuhile iteratsiooni raportit, mis sisaldab tööde listi. Antud list annab ülevaadet millised tööd said õigeks ajaks valmis, millised jäid tegemata. Raport on *Jira* tööde väljavõte ning sisaldab ülesannete viiteid, et vajadusel saaks uurida täpsemalt, kuidas antud töö valmimine edenes. Iga töö taga määratakse kulunud tundide arvu. Kõik tööd summaarselt näitavad ka seda, kui palju reaalselt oli kulutatud töötunde terve iteratsiooni sees. Antud mõõdik sai kasutatud selleks, et koostada iga konkreetse tiimi võimekust ja iteratsiooni maksumust. Kõikide tööde hinnastades ning olemasoleva statistika kombineerides sai arvutatud ka iga mooduli umbkaudne maksumus. Kõikide moodulite maksumused samuti andsid sisendit kas projekti koondmaksumus on kooskõlas väärtusega, mis oli sõnastatud projekti alguses. Siin tuleb mainida, et antud projekti maksumus algusfaasis oli väga rakse määrata, kuna tuli vahetada palju põhisüsteeme ning sügava detailanalüüsita ei olnud teada täpsema skoobi. Seega sai kasutatud hinnang nagu "hetke parim teadmine" ning tehtud kokkulepe, et projekti finaalne maksumus arvutatakse statistiliste andmete olemasolul. See on suur risk, aga arvesse võttes, et vanade süsteemide asendamine on paratamatu, tuli seda võtta. Praegune projekti edenemine näitab, et esialgne hinnang ei erine drastiliselt tekelikust maksumusest, kuna hinnangu andmisel oli kasutatud erinevate arhitektide ning ettevõtete

kogemust, kus sarnased projektid said teostatud. Tuleb arvestada, et see on tuletatud näidik ja ootamatu kulu alati võib tekkida.

Kui arendustiimid hakkasid tööd tegema, siis reaalne iteratsioonide kulud said tekkima kuupõhiselt. Kui oli kogutud mitu kuu statistikat, sai arvutatud kuu keskmine väärtus. Hetkel, kui on möödunud mitu iteratsioone, on näha, et igakuiselt koondmaksumus hajuvus ei ületa 7%, mis võib nimetada nagu stabiilne näidik. Antud väärtuste kasutusel on teada kui suur kulu toovad järgmised etapid ning on võimalik luua kogumaksumuse projektsiooni (Joonis 10).



Joonis 10. Projekti maksumuse projektsioon

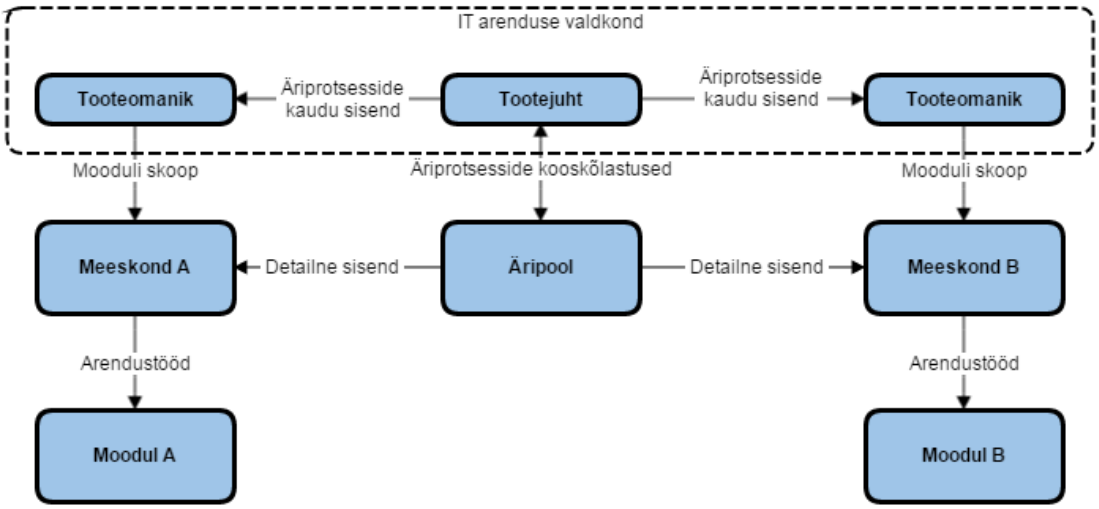
Antud projektsiooni ning edenemist kontrollitakse iga 2 nädalat projekti juhtorgani poolt ning kinnitatakse kõik muudatused, kui tekkib vastav vajadus.

Kokkuvõttes saab öelda, et iga mooduli meeskond toimetab *Scrum* raamistiku põhiselt. On võimalik kokkuleppel muuta *Sprinti* skoobi, moodulites kasutakse *Backlogi*. Samuti meeskond hinnastab iga iteratsiooni kohta kas "tempo" mõõdik on ootusepärane. Juhul kui mitte, antakse tagasisidet kuidas seda võiks olla parandatud. Mõned reaalsed juhtumid näitasid, et meeskonna võimekus või mooduli keerukus olid alahinnatud. Omalt poolt seda tähendas, et tuleb suurendada meeskonna liikmete arvu, otsida parema lahendust või vähendada eelnevalt kokkulepitud ulatust. Skoobi muutmine on võimalik ainult äripoole nõusoleku kudu ning kogemus näitas, et see oli ka aktsepteeritud, kui antakse põhjendatud sisend, miks juba defineeritud ulatus peab olema muudetud. Kõige rohkem antud nüansse on tulnud olukorrast, kui tuli luua seoseid olemasolevate süsteemidega. Oli saadud kinnitus, et antud muudatused on keerulised ning kallid ja pikemas perspektiivis ei too

palju kasu, kuna vanade süsteemide eluiga on läbimas. Moodulite meeskonnad tegelevad iseseisvalt, mis on riskikoht, et lõpptulemus ei ole ühtlane. Selle maandamiseks meeskonnad on kohustatud oma töös lähtuda äriprotsessidest.

6.7. Arendusprotsess

Modulaarne struktuur on mugav arhitektuuriliselt, kuid on raskesti projitseeritav äriliste protsesside peale. Seega tekkis oht, et iga mooduli valmimine liigub oma suunas, kuna omanik defineerib skoobi ise ning tegeleb ainult selle juhtimisega. Teiselt poolt eksisteerib äriline protsess, mis on tegevuste hulk oma sisenditega ja väljunditega. Reeglina ühe protsessi sammu sees eksisteerivad mitu funktsioone. Äriliselt need funktsioonid paistavad graafilise interfeisi kaudu, kus ühe kuva peal on nuppude ja väljade hulk. Seda tähendab, et korraga on vaja mitu moodulite funktsionaalsust ühel kuval. Kuna peamiste protsesside funktsioneerimine peab olema tagatud IT süsteemide poolt, siis on vaja rolli, mis vastutab selle eest, et äriliselt saab valmis õige toode ning erinevad moodulid projekti lõpus tagavad vajaliku funktsionaalsust. Selleks sai pakutud IT tootejuhi roll. Tootejuhid on peamised kontaktid tellija ja arendusmeeskondade vahel, seega see roll sai kasutatud ärilise sisendi korjamiseks ja protsesside eest vastutamiseks (Joonis 11). Mooduliomanikud kasutavad saadud protsesside kirjeldamist, et korraldada oma ulatust. Samuti tootejuhi vastutus on korraldada ärilist testimist ning vastuvõtmist lõpliku ekspluatatsioonile.



Joonis 11. Ärisisendi korjamine ning kooskõlastamine

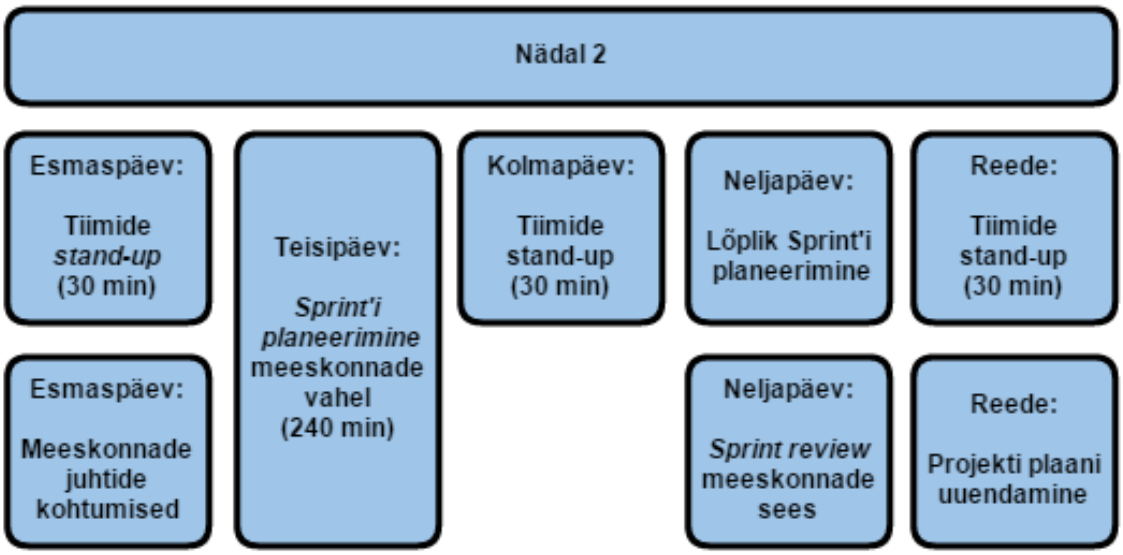
Selleks, et toetada parema läbipaistvust ning kuvada seosed moodulite ning äriliste protsesside vahel sai pakutud lisaks *Jirale* suhteliselt lihtsama struktuuriga tabel. Tabelis määratakse veerudes moodulid, ridades protsesside alamsammud ning lahtrites vajaliku funktsionaalsus. Iga funktsiooni koha määratakse kas selle funktsioneerimiseks on vaja teise mooduli teenus ja millal ta on valmis. Selle kaudu õnnestunud joondama erinevate moodulite arendussuuna.

Kui vajalik mooduli teenus saab valmis, siis vastav märkus tehakse nii vastavas tabelis, kui ka mooduli *Confluence* alamlehel. Lisaks sellele projekti kesk faasis sai kasutatud teenuste versioonide haldus. Ajaga sai selge, et moodulid ei ole võimelised alati hoida loodud teenust samal kujul ja mõnikord tekkib vajadus midagi muuta. Kui teised omanikud sellest ei tea või seda juba kasutab graafilise interfeis, siis ajaga enne töötav funktsionaalsus läheb katki. Selle riski maandamiseks oli tehtud kokkulepe, et iga uus versioon peab olema tagasiulatuvalt kokku sobiv. Kui seda ei ole võimalik tagada, siis kasutakse vana versioon ja vastavad muudatused peavad olema tehtud iga seotud osapoole poolt pärast. Teine väljakutse oli seotud sellega, et reeglina moodul ei saa odata teise mooduli teenust, et jätkata oma arendustööd, tööd tuleb teha paralleelselt. See sai lahendatud nii, et vajadusel

koostakse teenus õige struktuuriga, kuid andmed ja funktsionaalsus on võltsitud. See meetod aitas korraldada kiirt testimist ja parandada kommunikatsiooni äriga.

Protsessi ajaliseks korraldamiseks sai kasutatud kahe nädalane *Sprint*. Inimeste tegevuste sünkroniseerimiseks sai valmis igapäevane graafiline kalender. See võimaldas kiiresti panna paika protsessi ning tutvuda seda arendusmeeskondadele. Selle tulemusena iga projekti osaleja olu kursis mida ja millal tema poolt oodetakse.

Sprint algab esimesel nädalal, kuid selleks sisend luuakse teisel (Joonis 12), sellepärast ülevaadet on mõistlikum alustada teisest nädalast.

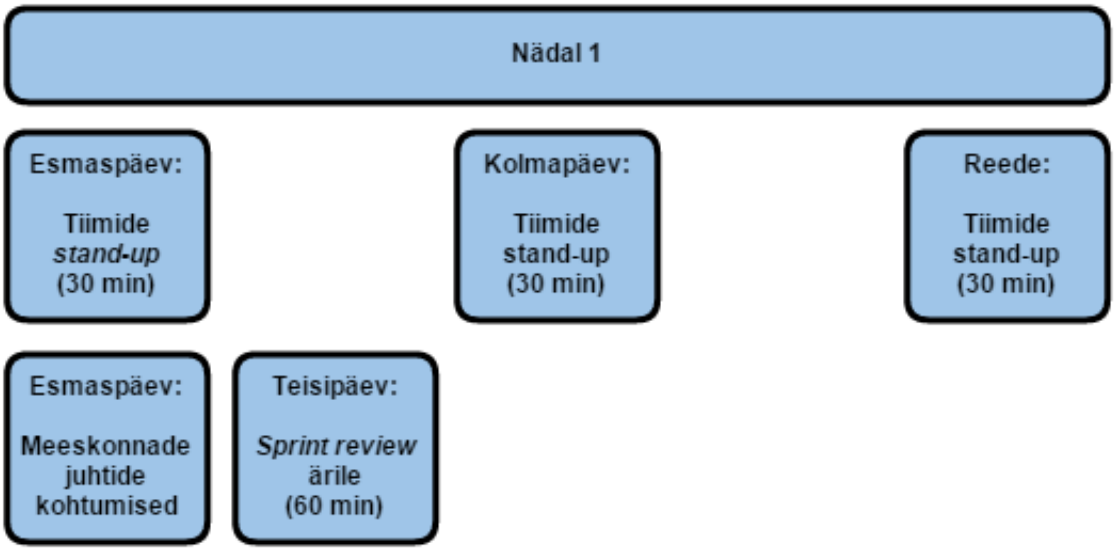


Joonis 12. Sprindi teise nädala kava

Esmaspäeval iga moodulimeeskond arutab lühikese *stand-up* jooksul millised on jooksvad probleemid, et kõrvaldada probleeme operatiivsel tasemel. Kui see saab tehtud, siis tehakse ülevaade nii tiimide juhtidele, kui ka valdkonnajuhtidele. Teisipäeval toimub pikk arutelu, mille eesmärk on arutada eelmise iteratsiooni tulemusi, mis sai tehtud, ja mis jäi tegemata. See on oluline sisend, et hakata planeerima järgmist *Sprinti*. Planeerimise ajal mooduli omanik räägib millised konkreetsed ülesanded on planeeritud ja grupis tehakse otsus, kas antud järjekord ning ulatus on loogiline ja kõigile sobiv. Samuti kui planeeritud tegevused omavad mõju teistele gruppidele, siis tehakse vastav plaani korrektsioon või lepitakse kokku, et teine grupp valmistab ette „võlts teenust“ sujuva progressi toetamiseks. Antud

tegevuse tulemusena Neljapäeval mooduli omanik korraldab sarnast üritust oma meeskonna sees. Seal kaalutakse, kas saadud plaan on realistlik, arutatakse iga ülesanne keerukust ning luuakse lõplik mooduli plaan. Samal päeval vaadetakse eelmise iteratsiooni tulemusi ja vajadusel tehakse korrektsioone. Reedeks iga meeskond esitab oma kinnitatud plaani ning seda konsolideeritakse koondplaanina. Selle kaudu nädala lõpuks peavad olema valmis nii moodulipõhised *Backlogid*, kui ka üldine projektiplaan, et hinnata katuseprojekti edenemist. See on veel üks näide, kuidas sobivad kokku nii *Scrum*, kui ka *Waterfall* raamistikud.

Esimene nädala eesmärk on genereerida koodi, jälgida progressi ning mitte koormata inimesi koosolekutega (Joonis 13). *Sprintid*e tulemusena järk-järgult sai valmis osa funktsionaalsust. Selle demonstreerimiseks sai korraldatud Teisipäeval *Sprint review*, kus saaksid osaleda kõik huvilised ja seotud osapooled. Iga mooduli omanik iteratsiooni raporti esitamisel märgistab, millised tööd on tehtud. Selle materjali abil koostakse koondraport, kus ärile lihtsustatud kujul tehakse kokkuvõtte sellest, kuidas läks eelmine iteratsioon ja mis tööd said tehtud.



Joonis 13. Sprindi esimese nädala kava

On rakendatud vaikimisi reegel, kui osa funktsionaalsusest jääb tegemata, siis ta peab olema tehtud järgmises *Sprindis* pluss planeeritud tööd. Samuti tehakse ülevaade sellest, mis peab olema valmis järgmise etapi raames, mille kaudu äri on võimeline planeerida oma ressursse testimiseks. Selgus, et kui jutt läheb moodulite teenustest või analüüside ülesannetest, siis reaalset väärtust ärile seda ei too. Selle parandamiseks antud ülevaade sai jagatud 2 etappideks. Esimene osa katab juba mainitud raportit, et ametlikult kinnitada progressi. Teises etapis tooteomanikud teevad valmisfunktsionaalsuse demonstratsiooni. Antud lähenemine tekitas, rohkem diskussioone ning aitas visuaalselt näha progressi.

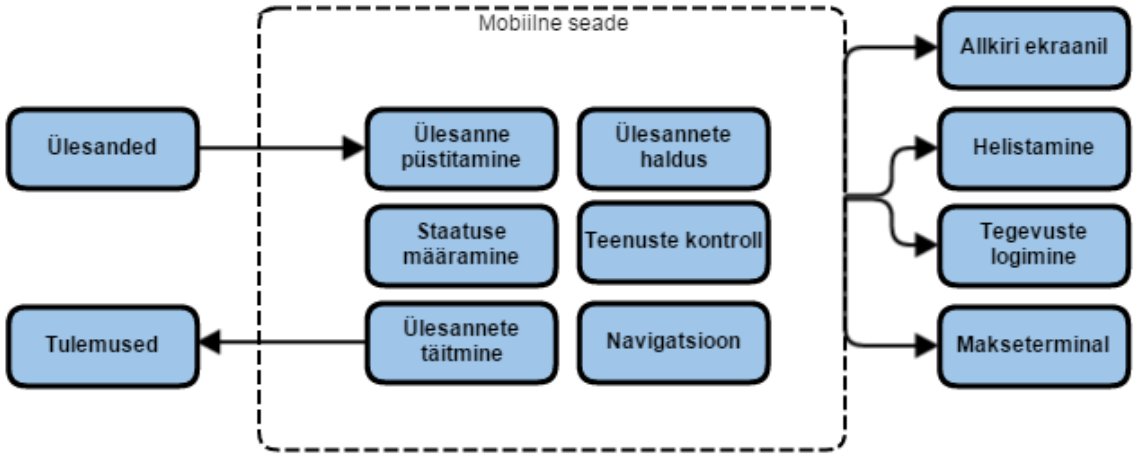
Nagu on näha, siis nii tööde planeerimise protsess, kui ka funktsionaalsuse loomine on ehitatud *Scrumi* protsessi järgi, nagu seda eeldasid hangete reeglid. Siin teooria läks hästi koos praktikaga. Peamine eesmärk sai täidetud - tagada edenemise läbipaistvust ning iga iteratsiooni tulemusena tarnida kasutatava funktsionaalsust. Olid iteratsioonid, kus ei õnnestunud tarnida midagi graafilise interfeisi poolt, ainult mooduli põhiseid teenuseid. Äriliselt antud tulemus ei saa olla aktsepteeritud ning testitud. Ajaga sai selge, et antud probleem on ajutine ning on tüüpiline ainult esimeste iteratsioonide jaoks ja ei vaja eraldi parandamist. Kirjeldatud tööde korraldamise protsess on üldine ning saab olla kasutatud suurema keerukusega projektide vedamiseks. Katuseprojekt ei ole veel valmis ning lõpliku järelduste tegemiseks ei ole kõlblik. Praktilise tausta andmiseks järgmisena tuuakse valmis alamprojekti kirjeldus, mille näitel saab aru saada, kas pakutud lähenemine on toonud oodatud tulemust ja kuidas seda mõõdeti.

6.8. Alamprojekti näide

Alamprojekt on töötajate käskude juhtimise tarkvara. Kuna ta ei erine teistest moodulitest, siis oma lähteülesannet ning analüüsi ta sai katuse projekti alguses, mille kaudu õnnestunud leppida kokku lõpuliku tähtaja ning eelarvet. See võimaldas juhtkonnale saada täieliku ülevaadet investeeringust ning teha vastava otsust, et kinnitada arendustööde algatamist.

Funktsionaalsed nõuded ning seotud protsesside kirjeldused samuti said ärilist aktsepteerimist õigel ajal. Hanke raames sai leitud sobiv meeskond, kes korraldas detailsemate nõuete korjamist, vajaliku seadmete analüüsi ning detailsema skoobi defineerimist. Kõik mainitud tegevused samuti said täidetud lubatud ajal.

Antud tarkvara (Joonis 14) on mõeldud kasutamiseks kullerite poolt. Tsentraalselt, enne tarnete füüsilist teostamist otsustakse, millised saadetised milliste kullerite poolt võetakse kaasa. Kui antud otsus tehtud, siis tööülesanded jagatakse töötajate vahel. Antud tarkvara paigaldatakse mobiilse seadmesse ning nii tarne, kui ka korje ülesanded jõuavad siia. Seega töötajal tekkib ülevaade, millised tööd peavad olema täidetud päeva jooksul. Listis tööd on võimalik ümber paigaldada soovitud järjekorras: aadresside, tähestiku või laadimise järgi. Iga ülesanne sees on saadetiste komplekt ning märgistatud vastavad teenuste tingimused. Näiteks, kui tuleb ID kaardi olemasolu kontrollida, siis tarkvara annab sellest teada ja ootab lisategevusi kulleri poolt. Samuti on võimalik kliendile helistamine ning vastava tulemuse märgistamine, näiteks kas õnnestunud uut kokkulepet teha või klient ei ole üldse kättesaadav. Kliendi juures kuller väljastatud või kätte saadud pakkide märgistamiseks saab skaneerida aadresskaarte ning rakenduses pannakse õige staatus. Vajadusel klient on võimeline jätta allkirja ekraani peal ning seda seostatakse süsteemis väljastatud saadetistega. Tehingu lõppfaasis on võimalik sooritada makset panga poolt väljastatud terminali kasutusel.



Joonis 14. Rakenduse üldehitus

Kui ülesanne sai täidetud, siis vastav uuendus jõuab tsentraalse süsteemi ning seda koheselt näeb seotud osapool. Saadud makse informatsioon suunakse raamatupidamistarkvarasse. Samuti on võimalik tsentraalselt suunata või eemaldada kulleri listist ülesandeid. See on eriti kasulik, kui päeva jooksul tekkib uus soov kliendilt ja töötaja peab sellest operatiivselt olla teavitatud. Juhul, kui selgub, et kuller ei olnud jõudnud ülesannet täita määratud ajaakna raames, siis tarkvara teostab vastava kontrolli, annab teadet ning küsib põhjuse, ilma selleta ei ole võimalik ülesannet kinni panna.

Antud alamprojekti käigul selgus, et mõned teemad vajavad detailsema analüüsi. Samuti sai tuvastatud realisatsiooni faasis, et loogika funktsioonide taga on keerulisem võrreldes plaanidega, kuna tuleb tagada kommunikatsiooni vanade süsteemidega. See põhines seda, et ajaliselt projekt on kestnud 3 nädalat võrra kauem, kui oli planeeritud. See on indikaator sellest, et isegi detailsem, võrreldes *Scrumiga* esialgne analüüs, ei saa tagada õiget ja täieliku teadmist. Miinusena oli ka see, et ärile õnnestunud testimiseks üles anda prototüübi projekti teises etapis ning tagasiside korjamine hilines. Seda põhines mõned ümbertegevusi lõppfaasis. Selleks oli loodud tagaplaan ja lisaarendajad olid ette valmistatud, seega suur mõju antud aspekt ei avaldanud.

Eelarveliselt õnnestunud hankida umbes 5% odavamaid seadmeid, võrreldes plaaniga, tänudes kompetentsi tiimile. Kui rääkida antud projekti maksumusest, siis lõplik hind on plaani järgne ja ei ületanud lubatud puhvrit. Väike ülekulu põhjus ongi sama ootamatu keerukus ning partnerile oli vaja teha rohkem arendustööd, et tagada kommunikatsiooni vanade süsteemidega. Plussina on see, et loodud kommunikatsioone mehhanisme on võimalik kasutada ka teiste moodulite jaoks ning antud ühekordne arendus säästab raha ning aega pikemas perspektiivis.

Lisaks selgus, et protsessidest lähtumine ei saa tagada lõpliku läbipaistvust, kuna arendusmeeskonnale oli raske realiseerida funktsionaalsust ainult protsessisammude kaupa. Tagasisidena selgus, et on vaja detailsema sisendit funktsioonide ning *use caseide* tasemel. Mõned sprindid lõppesid sellega, et sai valmis osa protsessist A, ning mingi osa protsessist B, kuid ühtegi valmis funktsiooni ei saa üle anda ärile testimiseks. Seda

tähendab, et suuremate projektide juhul *Scrumi* lähenemine on võimalik, aga katuseprojekt peab omama teatud mõju tiimide *Backlogidele*, täieliku autonoomsust ei saa lubada.

Antud alamprojekti näitel saab öelda, et erinevate lähenemiste kombineerimine saab olla edukas, kuid tuleb lähtuda projekti keerukusest ning suurusest. Äri pool otsustas esialgu piloteerida valmis tehtud tarkvara. On kokkulepitud protsess, kuidas toimub vigade raporteerimine ning uute ideede kaardistamine. Piloodi esimesel nädalal tekkis 2 probleeme, milliste staatus oli määratud nagu "blokeerija". Vigade parandus õnnestunud kahe päeva sees. Samuti esialgsel süsteemide paigaldamisel, kuhu kuulub ka kõige mahukam ja keerulisem "taga-osa" tuli teha paigaldatud süsteemi tagasi liikumist. Põhjus oli selles, et vanad süsteemid ei olnud valmis selleks, et toetada nii suurt andmete mahu. Arendajad said vastava parandust valmis ning edaspidise töö korraldamiseks sai tehtud kokkulepe, et koormustestid on alati kohustuslikud. Samuti antud katsetus andis hea sisendit kuidas tulevikus on vaja korraldada paigaldamist ja reliisimise protsesse. Oli ka korjatud eesliini töötajate tagasiside. Selgus, et tarkvara on lihtne ja mugav, ning lõpliku vigade eemaldamisega sobib paremini, kui vana. Projekti analüüsietapis, said tehtud mõned eeldused, millised teoreetiliselt pidid muutma tööprotsessi efektiivsemaks ning esimene tagasiside näitas, et eeldused olid õiged. Konkreetne näide võib olla helistamise funktsionaalsus. Vana seade ei olnud seda toetanud, mille tulemusena kuller pidi kasutama teist telefoni. Nüüd aga iga ülesanne taga on vajalik telefoni number koheselt olemas ning seda kasutada nupu "Helista" aktiveerimisel.

Antud alamprojekti puhul said kasutatud mõõdikud nagu projekti suurus, funktsioonide hulga hindamiseks, tiimi tempo, tegelik ja plaanijärgne kulu, tegelik ja plaanijärgne ajakava. *Waterfall* protsessi kaudu vajalikud teenused said valmis teiste moodulite poolt õigel ajal ning tiim ise oli võimeline tegutseda *Scrumi* põhiselt.

7. KOKKUVÕTE

Kui rääkida terve projekti edenemisest, siis kõik moodulid said omas ulatust defineeritud, funktsionaalsuse kirjeldust, andmete mudelid ning analüüsi õigel ajal, mis oli fikseeritud projekti plaanis. Samuti kasutatud modulaarne lähenemine võimaldas leida just õigema kompetentsi konkreetse loogilise tüki valmimiseks. Mõnedel juhtudel selgus, et valitud meeskond ei ole parim konkreetsete tööde tegemiseks. Kasutatud hangete printsiip võimaldas teha kiirt asendust, mis on loonud vajaliku paindlikust IT valdkonnale. Antud hangete süsteemi rajamine ajaliselt ja töömahu poolt ise oli suur investeering. Antud tegevuse alguse ja esimeste hangete ilmumise vahel on umbes üks kuu ja 6 inimeste tööd. See tähendab, et sarnase keerukuse projektisse toomine peab olema kaalutud otsus.

Hetkel iga meeskond on võimeline korraldada oma tööd suuremas mahus iseseisvalt, aga kui on oodatud funktsionaalsus teiste tiimidepoolt, siis abiks on ühine plaan. Antud viis annab tiimile vajaliku paindlikust, et teha oma plaanides vajadusel kiired muudatused. Aga teiselt poolt ülevaade projekti üldskoobist ei võimalda tekitada anarhiat, kus iga meeskond liigub ainult omas suunas. Antud lähenemine on loonud keerukust katuseprojekti jaoks, kuna tuleb siduda iga projekti plaani nii, et tulemusena tekkis ühtlane ning konsolideeritud toode.

Koordineerimise poolt lähenemine, kus iga alamprojekt on *Scrumi* ning katuseprojekt on *Waterfalli* põhine, vajab eraldatud juhtimise ressursse. Hangete poolt eraldi inimene tegeleb sellega, et dokumendid on väljas ning kättesaadavad õigel ajal. Lisaks ta tegeleb kulude konsolideerimisega, et kogumaksumus oleks teada ning vajalikud otsused juhtkonna poolt said tehtud. Protsesside kirjeldamiseks, skoobi juhtimiseks ning side toetamiseks äri ja IT vahel samuti sai kasutatud eraldi roll, ning sellega tegelevad kolm inimest. See on ainukene viis, et äriline ja IT visioonid oleksid kooskõlastatud, arendussidend liiguks õiges suunas ja tempos. Siin tuleb ka mainida ettevõte IT juhi rolli. Ta võttis endale vastutust ka täita projektijuhi ja peamise koordinaatori rolli. Nii suure projekti korraldamine ja liikumine on väga raskendatud, kui keegi tsentraalselt ei võta vastu kiired ja suured otsused, et toetada progressi.

Tervikuna hetkel saab öelda, et hübriidne metodoloogia on edukas. Vaatamata koordineerimiskoormusele, tulemused jõuavad õigel ajal ning väga suure funktsionaalse ulatuse liikumine on sobilikus tempos. Projekti jooksul kaks korda tuli teha suured muudatused – *Backlog itemite* asendamine protsesside analüüsiga ja selle tulemusena tuli täielikult uuendada *Jira* struktuuri, mis sisuliselt tähendas nullist tegemist. Vaatamata sellele, et *Backlog*, mis esialgu koosnes *Itemitest*, sai tervikuna asendatud, see oli oluline kogemust, et parandada ärilise sisendi kvaliteedi. Negatiivne külg on ka see, et suhteliselt keerulised mõõdikud nagu Mooduli skoor, Mooduli määramatus ning Kiirus said asendatud lihtsamatega. Seega eksisteerib oht, et mõõtmata aspekt selgub projekti lõpus ning tuleb loobuda osa funktsionaalsusest. Siin konkreetse näitena võivad olla Kiiruse ning Keerukuse parameetrid. Ilma nendeta on keeruline hinnata, kas tööde valmimine on õiges tempos või tuleb suurendada kaasatud ressursse. Pidevalt jälgitakse tegelikud ja prognoositud tähtajad ning eelarved. Seega mõõtmise poolt sai rakendatud *Waterfall* vaade. *Scrumi* põhine mõõdikute rakendamine jäi moodulite tiimide sees, ning leidnud rakendust üldprojektis. Näidikute arvutamiseks kasutakse statistika, et luua projektsioone tuleviku peale.

Kui jätkusuutlik antud projektijuhtimise metodoloogia on näitab aeg ning lõplik projekti valmimine. Alamprojekti näide on positiivne. Äripool hindab kogemust nagu hea, vaatamata sellele, et pidevalt tuleb oma sisendit andma ja olla kaasatud projekti igapäevaselt. Antud lähenemine aitab ühtlaselt jagada koormust terve tarkvara valmimise ajal. Algusetappidel antul lähenemise rakendamiselt vastupanu oli märgatav, mis on loomulik, luna ettevõtte jaoks see on uus lähenemine.

Antud töö loogiliseks jätkuseks võib olla veel üks metodoloogia katsetus reaalse projekti näitel. Kui olulised õpptunnid on läbitud ja mittesobivad printsiibid kõrvaldatud, siis saadud tulemuse näitel võiks tekkida järgmine mudeli iteratsioon. Nagu tehnoloogiad ja projektid, on vaja aega esimese katsetuse pärast, et tagada parema kvaliteedi ja stabiilsust. Kui suured ja põhimõttelised printsiibid on paigas, siis võiks detailselt uurida, kuidas antud lähenemine mõjutab ettevõtet ning koostööd IT ja äri vahel.

Uut meetodite proovile panemine on eeldus selleks, et otsitakse parem ja efektiivsem viis, et korraldada tootmist. Kindlasti kunagi ei tekki ideaalset mudelit, mis lahendab kõik probleemid korraga ja edaspidine paranemine on mõttetu. Hetkel on tunda, et maailma arenemiskiirus ainult kasvab. Omalt poolt seda tähendab, et jäävad ellu ainult ettevõtted, millised on võimelised kiiresti adopteerida. See on põhjus, miks tekkisid uued *agiilsed* viisid, mis on loomulik reaktsioon kiiremate muutuste tegemiseks. Ajaga tulevad uued ideed ja raamistikute range rakendamine veel ei garanteeri midagi. Kui kõik organisatsioonid teeksid omad tööd ühtemoodi, siis on keeruline liidrina olla, tuleb midagi leiutama, mis aitab saada parema tulemust. Peamine on proovida midagi uut, teistmoodi teha ning õppida saadud kogemusel.

SUMMARY

Given thesis target is to provide hybrid solution for IT project management methodologies. As it is stated in thesis name: “Combining Waterfall and Scrum Methodology Approach in IT Development Projects with Policy Limitations”. Reason of that is constant need in development processes improvement. Using best sides of both frameworks, new approach should be giving great overview of a project from a budget, timeline and management perspective and from the other side ability to make changes in project fast and receive constant feedback from a business side. Main challenge for providing new methodology, based on used frameworks it fact, that both approaches are diametrically different.

Cascade, also known as Waterfall model during its long history has proven itself like a solid framework, which gives opportunity to see project as a whole in a time perspective. Every step should be finished before development team can move forward. When project has in-depth analysis done, great documentation, then many risks mitigated thru it. Waterfall process is simple to use, straightforward but its drawback sides are also have huge impact on a projects. It is not flexible, when changes needed and author experience showed that when business sees result on project final stages for a first time, then there is a high probability, that costly and complex changed should be done, to meet final business requirements.

As Waterfall opposite, Scrum approach emerged during last decades. It had to solve Waterfall problems, giving possibility to get feedback constantly and use renewed business needs to change current situation and plan. Such flexibility is a great for a both sides, but framework has negative sides as well. First, it is hard to manage big projects using Scrum. It is tend to lean whole process in uncertainty and anarchy way of doing things, when used in a wrong way. Analysis itself is detailed for functions, which are being developed first, what means, that complexity and duration of a whole project are unknown parameters. Problem is that big companies are not allowing such approach, when dealing with investments and full transparency of a project should be provided before launching it.

Proposed hybrid model uses Waterfall structure for a roof-project, which consolidates small subprojects under it. Methodology application is not theoretical in frames of given thesis and every proposed tool and method is used in a real project, which lasts until June 2017. Every small sub-project sees its target to create module-based software. Modules are communicating via agreed protocol and using micro services architecture. Procurements built in a way, that every module is developed by a separate team, should deliver new functionality every Sprint and after acceptance of it, agreement can be prolonged.

Additionally, thesis consists of finished sub-project detailed overview. Its target is give understanding what was done using proposed approach. In total, project has a slight delay in deployment – 3 weeks, but budget and functional needs are met, as planned. Such result can be named as a success, taking in account complexity and size of a whole project, where are integrated many old systems and created a lot from a scratch.

As a result, proposed framework has potential and fulfills its targets. From a roof-project perspective there is a fixed and known budget, deadline and progress overview. Every module drives its progress in a Scrum way, so fast changes and constant feedback provided. As a minus side, there is a big load of managing such structure. There is a separate employee, who deals with partners and coordinates them, employee who is tracking progress, budget and real spends and, additionally, 3 employees who consolidating business input and constantly synchronize two worlds – Business and IT. Flexibility of procurements and agreements gave possibility already to change some partners in a fast manner, without losing a tempo. It means, that practically methodologies can be mixed and provide needed benefits to a Business. However, as opposite side company should be ready to invest additional money and time for coordination and correction of failed methods.

It is clear, that proposed methodology cannot serve any random company needs and should be adopted properly. However, it shows, that trying to invent something new, based on existing frameworks, for accelerating development process is a viable approach. A company should consider it, if time and market pressure are forcing to move and change faster than before.

TSITEERITUD TEOSED

1. Kiisler, V. (2016). *Eesti IT vajab 1000 inimest*. Loetud aadressil: http://www.ituudised.ee/uudised/2016/08/15/eesti-it-vajab-1000-inimest
2. Brooks, F. (1986). *No Silver Bullet – Essence and accident in software engineering*. Univercity of North Carolina at Chapel Hill
3. Royce, W. (1970). *Managing development of Large Software Systems*.
4. Bell, T ja Thayer, T. (1976). *Software Requirements: Are They Really a Problem?*. ICSE '76 Proceedings of the 2nd international conference on Software engineering
5. Wiegers, K. (2003). *Software Requirements (2nd ed.).* Redmond: Microsoft Press.
6. Stellman, A. ja Greene, J. (2005). *Applied software project management*. O'Reilly Media, Inc.
7. *Systems Engineering Fundamentals* (2001). Defense Acquisition University Press
8. McConnel, S. (2006). *Software Estimation: Demystifying the Black Art*. Redmond, Wa.: Microsoft Press
9. EtestingHub-Online Free Software Testing Tutorial. *Testing Phase in Software Testing*. Loetud aadressil: http://www.etestinghub.com/testing_lifecycles.php#2
10. Pan, J. (1999). *Software testing.* Loetud aadressil: https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/
11. McConnell, S. (2004). *Code Complete (2nd ed.).* Microsoft Press.
12. Smart, M. (2006). *Waterfall Development Methodology*. The Smart Method Limited.
13. Charette, R. (2005). *Why Software Fails*. IEEE Spectrum. Loetud aadressil: http://spectrum.ieee.org/computing/software/why-software-fails/3
14. Berkun, S. (2008). *Making Things Happen. Mastering Project Management*. O'Reilly.
15. Putnam, L. ja Myers, W. (2003). *Five core metrics : the intelligence behind successful software management*. Dorset House Publishing.

16. Beedle et al. (2001). *Manifesto for Agile Software Development*. Loetud aadressil: http://agilemanifesto.org/
17. 12 Principles Behind the Agile Manifesto. (kuupäev puudub). Loetud aadressil: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/
18. The 10th Annual State of Agile Report. (2016). VersionOne Inc. Laaditud ressursist: https://www.versionone.com/
19. Takeuchi, H. ja Nonaka, I. (1986). *The new product development game*. Harward Business Review
20. Schwaber, K. ja Sutherland, J. (2016). *The Scrum Guide*.
21. Gilley, C. (2015). *The Pros and Cons of Agile Product Development*. Loetud aadressil: http://community.uservoice.com/blog/the-pros-and-cons-of-agile-product-development/
22. Dinwiddie, G. (2009). *Feel the Burn. Getting the Most out of Burn Charts*. Better Software
23. Erickson, A. (2009). *Scrummerfall: World's Worst Software Development Methodology*. Loetud aadressil: http://www.informit.com/articles/article.aspx?p=1392832
24. Neher, K (2009). *WaterScrum: Co-dependent Waterfall and Scrum projects*. Loetud aadressil: http://scrumcommunity.pbworks.com/w/page/10149050/WaterScrum
25. Mehan, M. (2012). *'Water-Scrum-Fall' – Agile Reality for Large Organisations*. Agile Business Conference, 2012. http://www.agileconference.org/wp-content/uploads/2012/10/Water-Scrum-Fall-Manav-Mehan.pdf
26. Zeiger, P. (2013). *Vajalikke teadmisi ettevõtlusest*. Loetud aadressil: http://ettevotlusope.weebly.com/822-investeeringute-planeerimine-ja-investeerimisprojektide-efektiivsuse-hindamine.html
27. Business Process Model and Notation. (2011). Version 2.0. Loetud aadressil: http://www.omg.org/spec/BPMN/2.0/
28. Scrum Roles Demystified. (kuupäev puudub). Loetud aadressil: https://www.scrumalliance.org/agile-resources/scrum-roles-demystified

29. Nuseibeh, B. ja Easterbrook, S. (kuupäev puudub). *Requirements Engineering: A Roadmap*.
30. Differences Between Black Box Testing and White Box Testing. (kuupäev puudub). Loetud aadressil: http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/
31. Pierce, W. (2016). *Team Roles in Waterfall Methodology*. Loetud aadressil: https://atlaz.io/blog/team-roles-in-waterfall-methodology/
32. The Scrum History. (kuupäev puudub). Loetud aadressil: http://www.scrumguides.org/history.html
33. Certified ScrumMaster. Version 3.0. (2015). Management Institute of Finland