TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Indrek Majas

Tarkvaraarenduse metoodika valiku protsess (CGI Eesti AS näitel)

Bakalaureusetöö (6 EAP)

Juhendaja(d): Raul Raudsepp
Helle Hein

Tartu 2016

Tarkvaraarenduse metoodika valiku protsess (CGI Eesti AS näitel)

Lühikokkuvõte:

Alustades uut tarkvaraarenduse projekti, tekib alati küsimus, milline tarkvaraarenduse metoodika oleks antud projekti läbiviimiseks parim valik. Osad metoodikad toimivad paremini mingit tüüpi projektide puhul, teised mitte nii hästi. Käesoleva töö eesmärgiks on tuua välja erinevad tarkvaraarenduse metoodikad, analüüsida erinevate metoodikate omadusi ning selgitada välja sobivaim metoodika erinevat tüüpi projektide läbiviimiseks.

Võtmesõnad:

Agile, Scrum, Kanban, Waterfall, Agile Software Development

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Software Development Methodology Selection Process (by Example of CGI Estonia AS

Abstract:

Starting a new software development project raises always the question of which software development methodology would be the best option for carrying out the project. Some methodologies are performing better in some types of projects than others. The purpose of this paper is to highlight the different software development methodologies, to analyse the characteristics of the various methodologies and to identify the most suitable methodology for various types of projects.

Keywords:

Agile, Scrum, Kanban, Waterfall, Agile Software Development

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Tänapäeva infotehnoloogilises ühiskonnas toimub pidev areng uute tehnoloogiate ja uute tarkvaralahenduste suunas. Kättesaadava info hulk kasvab ning info tarbimist võimaldavate tehnoloogiliste platvormide pidev areng võimaldab infot tarbida üha väiksema vaevaga. See aga tähendab, et infol ja info kättesaadavusel on tänapäeva infotehnoloogilises ühiskonnas võtmeroll. Info- ja kommunikatsioonitehnoloogia võimaldab tõsta mistahes majandus- ja eluvaldkonna konkurentsivõimet. See omakorda aga tähendab, et konkurentsis püsimiseks vajadus erinevate tarkvaralahenduste järele järjest kasvab.

Aastaid tagasi oli tarkvaraarendus ning samuti vajalik riistvara väga kallis – tarkvara ja IT-süsteeme telliti ja oli jõukohane tellida ainult kitsa ringi üsna sarnase juhtimismudeliga suurettevõtete poolt. Tüüpiline tarkvaraarendus kestis aastaid, tarkvaraarenduse jaoks töötati välja klassikaline tarkvaraarenduse metoodika – koskmudel (ingl. *waterfall model*).

Aastatega on infotehnoloogia levinud laiemalt – tarkvaraarendus on tänu uutele kiiretele ja lihtsamatele arendusvahenditele muutunud odavamaks ja samuti on järjest langenud riistvara hind – arvutid on saanud iga ettevõtte asendamatuks osaks. Seoses infotehnoloogia levikuga on ettevõtte vajadustele vastavad IT-lahendused ja spetsiaaltarkvara ka väiksemate ettevõtete igapäevase töö osaks. Infotehnoloogia järjest laiemalt levides on selge, et erinevate ettevõtete juhtimismudelid on erinevad, kliendi ootused on erinevad, arendatava tarkvara iseloom on erinev ja klassikaline tarkvaraarenduse metoodika ei ole alati piisavalt hea, välja on töötatud mitmeid uusi väledamaid (ingl. *Agile*) arendusmetoodikaid.

Siit tekib aga kohe küsimus – millist arendusmetoodikat millise kliendi, projekti, tarkvaralahenduse puhul eelistada? Kõik arendusmetoodikad töötavad kõige paremini selles situatsioonis, mille jaoks need loodi, kehvemini teises ja võib-olla ei tööta üldse kolmandas. Milline tarkvaraarenduse metoodika valida? Milline tarkvaraarenduse metoodika antud projekti läbiviimiseks kõige paremini sobiks? Selleks, et neile küsimustele vastused saada, toob käesoleva töö autor välja oma väljatöötatud protsesside kasutamise juhtumianalüüsi (ingl. *Case Study*) ning analüüsib lisaks maailma parimaid praktikaid.

Töö esimeses peatükis kirjeldame töö ja töö autori tausta, seejärel töö teises peatükis toome välja ja anname ülevaate peamistest töö autori poolt oma igapäevases töös kasutatavatest tarkvaraarenduse metoodikatest, töö kolmandas peatükis kirjeldame nende tarkvaraarenduse metoodikate iseloomulikud omadused, misjärel töö neljandas peatükis analüüsime maailma parimaid praktikaid ning nende sobivust mingit tüüpi tarkvaraarendusprojektide läbiviimiseks ning töö viimane, viies peatükk, sisaldab endas töö kokkuvõtet.

1. Taustainfo

Bakalaureusetöö autor on olnud aastate 2006 kuni 2015 jooksul algul Tieto Estonia AS tarkvaraarenduse projektijuht, hiljem projektimeeskonna juht. Aastast 2015 on autor CGI Eesti AS tarkvaraarenduse projektimeeskonna juht ning vastutav tarkvaraarendusprojektide eduka läbiviimise eest.

Tieto on Põhja-Euroopa suurim täislahendusi pakkuv IT-ettevõte. Tieto loodi 1968. aastal andmetöötlusettevõttena, kohe varsti hakkas Tieto pakkuma tarkvara ja IT-süsteemide arendust. Tieto Estonia AS põhisuund on tarkvaraarenduse ja IT-lahenduste pakkumine Eestis.

CGI Eesti AS kuulub CGI gruppi, mis on suuruselt viies infotehnoloogia teenuseid pakkuv ettevõte maailmas. Eesti turul on ettevõte tegutsenud üle 20 aasta ning CGI kontorid on Tallinnas ja Tartus. Lõviosa CGI Eesti AS tegevusest moodustab tarkvaraarendus – üle 90% kogukäibest.

Eesti turul on Tieto Estonia AS ja CGI Eesti AS konkurendid. Mõlemad ettevõtted on oma iseloomult üsna sarnased – kokku on aktiivseid kliente umbes 30, korraga töös olevaid tarkvaraarendusprojekte ligikaudu samapalju. Kliendid on nii avalikust kui ka erasektorist, tarkvaraarendusprojektide iseloom varieerub väiksemahulistest hooldusprojektidest kuni mitmeaastaste sadade tuhandete kasutajatega üleriigiliste süsteemide arendamiseni.

Samuti on kasutatavate tarkvaraarendusmetoodikate valik mõlemal tarkvaraarendusettevõttel sarnane – Scrum, Kanban, Waterfall.

Kuna töö algmaterjal ja algupärased mõisted on suures osas inglise keeles, siis arusaadavuse huvides kasutame mõisteid eesti keelde tõlgituna. Tõlkimise aluseks oleme võtnud teadus- ja arendusfirma Cybernetica poolt väljatöötatud standardipõhise tarkvaratehnika sõnastiku.[1]

[1]Standardipõhine tarkvaratehnika sõnastik, Cybernetica AS http://stats.cyber.ee/

2. CGI-s kasutatavad tarkvaraarenduse metoodikad

Kuni aastani 2007 kasutati CGI-s peamiselt seni laialdaselt levinud koskmudelit. Aastal 2007 otsustati tarkvaraarenduse kvaliteedi, efektiivsuse ja klientidele pakutava väärtuse tõstmiseks juurutada paindlikud tarkvaraarenduse metoodikad. Sel hetkel oli populaarsust kogumas Scrum, Eestis kohapeal oli juba saada vajalikke koolitusi ning konsultante, seega valituks osutus Scrum. Otsustati, et Scrum juurutatakse 100-protsendiliselt, mitte ei hakata kasutama ainult mõningaid Scrumi elemente. Otsus põhines veendumusel, et ainult nii on võimalik saavutada juhitav ja efektiivne protsess. Eestis ja ka (tol hetkel) Logica kontsernis oli Logica Eesti AS üks esimesi, kes Scrumi nii suures ulatuses kasutusele võttis. Logica rakendas Scrumi 90% projektides, sh ka väga suurtes projektides.

Aastal 2011 otsustati ühe kliendi tarkvaraarenduse eripäradest lähtudes juurutada ühe meeskonnaga Kanban, ka see osutus toimivaks metoodikaks.

Käesoleval hetkel kasutatakse CGI-s peamiselt kolme tarkvaraarenduse metoodikat:

- klassikaline koskmudel;
- Scrum;
- Kanban.

Neist Scrum ja Kanban on väledad tarkvaraarenduse metoodikad (ingl. *Agile software development*).

2.1 Klassikaline tarkvaraarendusmetoodika

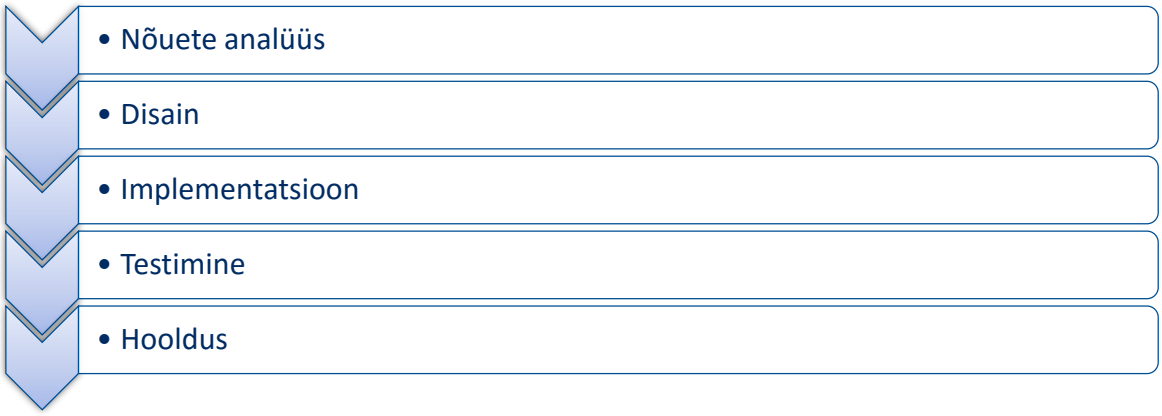
Ajalugu

Klassikaline koskmudel on üks esimesi tarkvaraarenduseks välja töötatud tarkvaraarenduse metoodikaid. Herbert D. Benington esitles 29. juunil 1956 „Täiustatud programmeerimismeetodid digitaalarvutitele“ sümpoosionil esimese tänapäevase koskmudeli sarnast mudelit. Mudelit oli kasutatud tarkvaraarenduseks SAGE (Semi-Automatic Ground Environment) [1] jaoks. Esimese formaalse koskmudeli kirjeldusena on tihti viidatud 1970 aastal Winston W. Royce poolt kirjutatud artikkel „Suurte tarkvarasüsteemide arenduse juhtimine“ [2], kuigi Royce ei kasutanud oma artiklis mõistet „koskmudel“. Esimest korda mainiti mõistet „koskmudel“ aastal 1976 Bell ja Thayer poolt [3]. Aastal 1985 Ameerika Ühendriikide Kaitseministeerium kirjeldas selle lähenemise tarkvaraarenduse partnerite tööstandardina DOD-STD-2167A [4], mis ütles: „Töövõtja peab rakendama tarkvaraarenduse tsüklit, mis koosneb järgnevast kuuest faasist: Eeldisain, detailne disain, kodeerimine, ühiktestimine, integratsioon ja testimine“.

Klassikaline koskmudel on järjestikune tarkvaraarenduse metoodika, välja kasvanud riistvara arendamise protsessist, mis kohandati tarkvara arendamiseks.

Protsess

Miks mudelit nimetatakse koskmudeliks? See tuleneb mudelit kirjeldavast protsessipildist (Joonis 1).



Joonis 1. Koskmudeli diagramm.

Protsess justkui „voolab“ ülevalt alla, iga etappi lõpus tulem kinnitatakse, protsess liigub edasi järgmisse etappi, tagasi eelmise etapi juurde ei pöörduta. Vaadates protsessi tausta (riistvara arendus), on selline mittetagasipöördumine ka täiesti arusaadav, valmis riistvara ei ole võimalik peale selle valmimist muuta, nõuete muutmisel protsess katkestatakse ja protsessi tuleb uuesti algusest alustada. Samuti kasutatakse koskmudelit peamise mudelina ehitussektoris. Koskmudeli kasutamine eeldab väga põhjalikku planeerimist, mis on võimalik ainult sarnaste projektidega varasema suure kogemuse olemasolul.

Praktikad

Projekti planeerimise käigus koostatakse projektiplaan. Projektiplaan kirjeldatakse tavaliselt Gantt'i diagrammina, mis toob välja projekti tööde kriitilise ahela – tööd, millest sõltub projekti tähtaeg. Tööd tükeldatakse töö liigendusstruktuuri (ingl. *Work Breakdown Structure*) ehk (WBS) meetodi abil pisemateks iseseisvateks tööülesanneteks, kõik tööülesanded saavad ajahinnangud. Töö liigendstruktuur järgib „100% printsiipi“ ehk 100% töödest peab olema plaanis kirjeldatud, kaasa arvatud projektijuhtimine ise. Kõik tööd on paigaldatud ajateljele, lisatud puhvrid.

Ilmselt on tajutav, kui suur töö on kogu selle plaani tegemine, jälgimine, et iga töö tegemiseks vajalik ressurss oleks õigel hetkel saadaval, et tööd algaksid ja lõpeksid plaanikohaselt, kui mitte, siis korrigeerida puhvreid.

Kasutusala

Klassikalist koskmudelit kasutatakse väga tihti Eesti riigi avalikus sektoris. Samuti on mõistlik koskmudelit kasutada väga suurte projektide ja meeskondade puhul. Koskmudelit kasutatakse väga tihti juhul kui on vaja realiseerida mitmeid oma olemuselt ja mahult sarnaseid projekte. Sellisel juhul saab suurt osa projektiplaanist kasutada tulevaste projektide puhul mustandina, viies sisse ainult projektipõhised täiendused.

2.2 Väledad metoodikad

Infotehnoloogia levides kasvas tarkvaraarendusprojektide ja IT-lahenduste järele vajadus, neid oli vaja rohkem, kiiremini, lahendused polnud enam ainult korporatiivsed vaid suunatud rohkem ka väikeettevõtete äri toeks.

Oli selge, et klassikaline koskmudel ei ole kõigi projektide jaoks piisavalt paindlik – projekti käigus tekivad uued ideed, turusituatsioon muutub, konkurendid tulevad uute toodete ja teenustega välja ja muudatusi on vaja nüüd ja kohe. Eelkõige tootearenduse puhul on oluline

olla oma tootega turul varem kui konkurendid, reageerida muudatussoovidele kiirelt, toodete/teenuste elutsükkel ei pruugi olla kuigi pikk. Uute metoodikate väljatöötamise vajadus ja põhiline eesmärk on kiire reageerimine muudatustele. Paindlikkuse tagamiseks töötati välja paindlikud ehk väledad metoodikad. Neist tuntumad on Scrum ja Kanban.

Hiljutised uuringud näitavad, et 42% paindlikest projektidest on edukad. See on kolm korda rohkem kui traditsiooniliste projektide puhul. Paindlikest projektidest läheb üle eelarve või üle tähtaja 49% ning ainult 9% projektidest on ebaõnnestunud täielikult. [2]

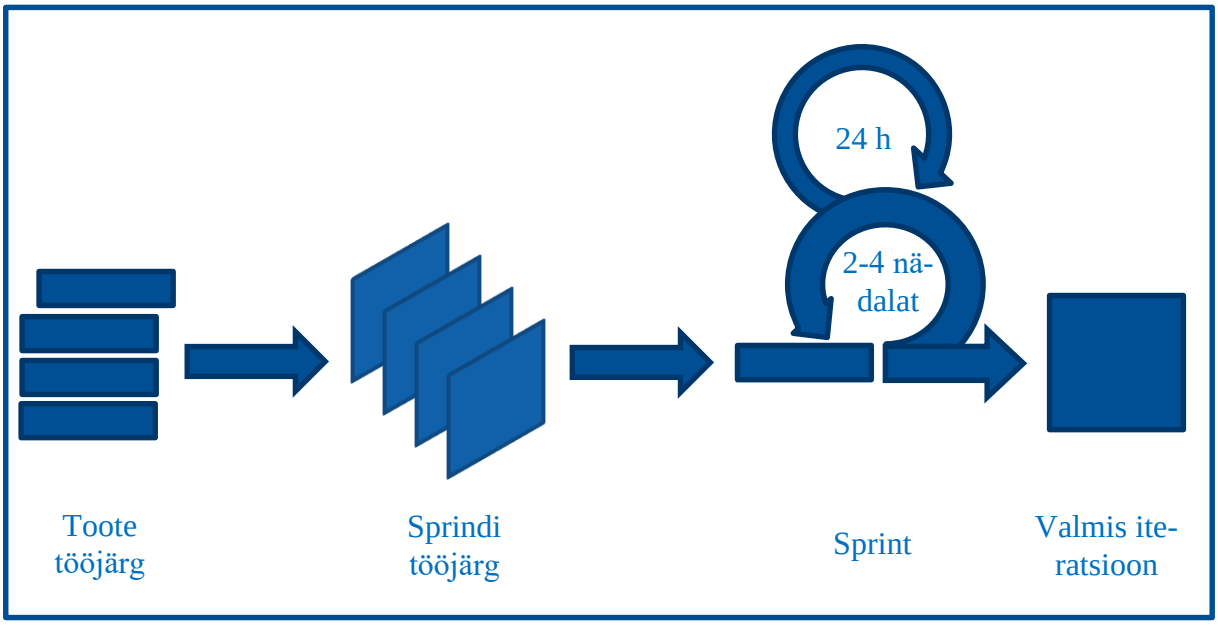
2.2.1 Scrum

Scrumi väljatöötamist alustati 90-ndate alguses ja esimene Scrumi kirjeldav raamat ilmus aastal 1995, autoriteks Jeff Sutherland ja Ken Schwaber[3]. Raamatus esitatakse Scrumi põhiideed. Tegemist on küllaltki noore tarkvaraarenduse metoodikaga. Hetkel on Scrum levinuim iteratiivne ja inkrementaalne väleda tarkvaraarenduse raamistik.

Protsess

Scrumi idee on töötada nn. sprintidena, iga sprindi lõpus tarnida kliendile valmis iteratsioon ning omakorda iga iteratsiooni jooksul ka parendada tööprotsessi.

Scrumi iseloomustab järgnev protsessipilt (Joonis 2Joonis 2. Scrum protsess.):



Joonis 2. Scrum protsess.

Scrum protsessi sammud on järgmised:

1. Jagada kogu projekti funktsionaalsus väiksemateks konkreetseteks iseseisvalt tarnitavateks osadeks ehk tööjärgedeks (ingl. *Backlog*);
2. Optimeerida tööjärg ja määrata prioriteedid;
3. Jagada organisatsioon väiksemateks multifunktsionaalseteks iseorganiseeruvateks meeskondadeks;
4. Jagada kogu projekti aeg lühikesteks fikseeritud pikkusega sprintideks, mis on piisava pikkusega, et iga sprindiga oleks võimalik midagi tarnida;
5. Tarnida kliendile võimalikult tihti;
6. Saada kliendilt kiire tagasiside;

7. Parendada pidevalt tööprotsessi.

Praktikad

Projekti alguses pannakse kokku toote tööjärg – nimekiri funktsionaalsustest (Scrumi mõistes: soovilugudest), mida soovitakse realiseerida. Soovilugu on puhtalt äriline ülesande kirjeldus. Iga sooviloo maht ja keerukus on hinnatud ja soovilood on järjestatud – olulisemad eespool, vähem olulisemad tagapool. Tööjärje algusest valitakse sel hetkel kõige prioriteetsemad soovilood sprindi tööjärge ning realiseeritakse. Realiseeritakse alati seda, mis on hetkel kõige olulisem. Ühelt poolt eeldab selline lähenemine toote tööjärjes olevate funktsionaalsuste tükeldamist piisavalt väikesteks ja tarnitavateks tükkideks, et need oleksid ühe arendussprindiga realiseeritavad. Teiselt poolt, kui tekivad kõrge prioriteediga muudatussoovid, on võimalik need kohe järgmises sprindis töösse võtta ning kliendile realiseeritakse üsna lühikese sammuga ja just kõige olulisemat funktsionaalsust.

Sprindisiseselt on ärilised soovilood tükeldatud väikesteks tehnilisteks tööülesanneteks, tööülesandeid hinnatakse tundides, igapäevaselt on sprindi lõpuni jäänud tundide baasil võimalik välja joonistada sprindi jääktöö graafik (ingl. *Sprint Burndown Chart*), mille baasil on võimalik tuvastada takistusi ning hinnata, kas sprindi eesmärgid saavad täidetud. Iga sprindi lõpus toimub tagasivaatekoosolek (ingl. *Retrospective Meeting*), mille eesmärgiks on välja tuua viimase sprindi arendusprotsessi vajakajäämised ning parendada järgmise sprindi arendusprotsessi.

Scrumi meeskond on iseorganiseeruv. Sellest tulenevalt klassikalist projektijuhi rolli Scrumi metoodika järgi ei ole. Reaalses elus on siiski vajalik ka projektijuht, kelle ülesandeks on hallata kogu projekti administratiivset poolt, finantse, meetrikat, meeskonna ehitamist (ingl. *Teambuilding*).

Rollide jaotus on järgmine:

- Tooteomanik (ingl. *Product Owner*), kes hoolitseb toote tööjärje eest ja peegeldab meeskonnale tagasi toote jääktöö graafikut;
- Rüsinameister (ingl. *Scrum Master*), kes hoolitseb meeskonna fookuse hoidmise eest, hoiab Scrumi formaati, aitab meeskonnal areneda ning peegeldab meeskonnale tagasi sprindi jääktöö graafikut;
- Scrumi meeskonna liige (ingl. *Scrum Team Member*) ehk tavaline meeskonnaliige (arhitektid, analüütikud, arendajad, testijad jt).

Scrumi puhul kasutatakse peamiselt kahte iteratiivset lähenemist:

- Iga iteratsiooniga tarnitakse lõpuni valmis arendatud funktsionaalsus, sama funktsionaalsuse juurde enamasti tagasi ei pöörduta – iga uus iteratsioon lisab uut funktsionaalsust;
- Funktsionaalsust realiseeritakse „Make it Work, Make it Fast, Make it Pretty“ põhimõttel – esimeste iteratsioonide käigus realiseeritakse kogu funktsionaalsus võimalikult lihtsa aga töötavana, järgmiste sprintide käigus optimeeritakse lahendused ning viimaks pööratakse rõhku lahenduse lõplikule viimistlusele.

Esimest lähenemist kasutatakse enamasti tootearenduse puhul, teist lähenemist juhul, kui Scrumiga realiseeritakse fikseeritud hinna, ulatuse ja tähtajaga projekti.

Kasutusala

Scrumi kasutatakse väga tihti lühikese elueaga toodete ja teenuste arendamisel, kus oluline on turule tuleku kiirus ja mida on võimalik iteratiivselt täiendada. Näiteks igasugused iseteeninduse lahendused, portaalid, teenused.

Hetkel on Scrum levinuim CGI-s kasutatav tarkvaraarendusmetoodika. Projektid, mille omadused on sobinud Scrumiga, on osutunud edukaks. Näitena Sotsiaalministeeriumi Terviseameti Laborite infosüsteem TALIS valmis soovitud funktsionaalsuses koguni kaks nädalat enne kokkulepitud tähtaega ning ülejäänud aeg oli võimalik arendada kliendile täiendavat funktsionaalsust.

2.2.2 Kanban

Üldine Kanbani idee on pärit 50-ndate algusest Toyota Motor Corporation inseneri Taiichi Ohno poolt, kes töötas välja metoodika tootmise efektiivsuse tõstmiseks. Siiski kulus aastakümneid, enne kui töötati välja Kanban metoodika tarkvaraarenduse juhtimiseks. Üks esimesi, kes formuleeris Kanbani IT ja tarkvaraarenduse jaoks ning juurutas selle 2004. aastal Microsoft Corporation'is, oli David Anderson, kes andis aastal 2010 välja ka ühe esimese selleteemalise raamatu „Kanban" [4]. Kanban ei ole otseselt tarkvara arenduse metoodika, Kanban on muudatuste haldamise metoodika. See tähendab, et Kanbani abil ei ole efektiivne läbi viia tarkvaraarenduse projekti, küll aga on Kanban lihtne ja efektiivne metoodika juba olemasoleva tarkvara või IT-lahenduse igapäevaseks hoolduseks ja täiendamiseks.

Protsess

Kanbani idee on pärit tootmisest – ühelt poolt ei tohi võtta liiga palju tööd korraga ette, kuna ühe töö läbiviimise aeg kasvab, teisalt peab olema töid piisavalt, et ei tekiks arendustegevuses auke. Protsess koosneb lihtsustatult järgmistest sammudest:

- visualiseerida töövoog;
- limiteerida käsiloleva töö kogus ehk WIP (ingl. *Work-In-Progress*);
- mõõta töövoo läbimise kiirust.

Praktikad

Kanbani kasutusele võtmisel esimesena visualiseeritakse töövoog – tarkvaraarenduse puhul arenduse töövoog sooviloost tarnimiseni. See on oluline, et oleks läbipaistev, millised on tarkvara arenduse töövoo erinevad etapid ja millises etapis mingi soovilugu hetkel on.

Järgmisena limiteeritakse igas töövoo etapis olevate soovilugude arv. Selle tegevuse eesmärgiks on hoida ära tööde kuhjumine ühes töövoo etapis. Selleks limiteeritakse käsiloleva töö kogust ehk määratakse maksimaalne arv ühes tarkvaraarenduse töövoo etapis olevaid töid – kui arenduspuhvris on vähem töid, võetakse uusi töid juurde; kui käsilolevate tööde kogus on täis, uusi töid järjekorda juurde ei võeta. Igal töövoo etapil on määratud selle etapi maksimaalne käsiloleva töö kogus, korraga mahub töösse alati üks ja sama arv töid. Tööde suurus pole oluline.

Kasutusala

Kanbani kasutatakse peamiselt sisearenduste puhul, samuti ka hooldusprojektide puhul.

CGI-s kasutatakse Kanbani peamiselt nende projektide puhul, mille funktsionaalsus on lepingus kokkulepitud mahus realiseeritud ja projekt on üle antud, kuid projekti kohta on sõlmitud hooldusleping ning igapäevaselt tuleb teha väiksemaid täiendusi. Samuti kui tarne

toodangusse mineku järel tuleb olla valmis kiireks vigade paranduseks ning Scrumi kasutamiseks ja mõistliku pikkusega sprintide planeerimiseks ei ole piisavalt tööjärge ees; mõistlik on kasutada lihtsamat metoodikat, milleks on Kanban. Näitena SMIT-i projekt SOS2 arenduses kasutame vaheldumisi Scrumi ja Kanbani –funktsionaalsuse arendamisel kasutame Scrumi, tarne toodangusse paigaldamise järel lülitume ümber Kanbani peale kuni soovitavate pisitäienduste hulk langeb ning meeskonnal on võimalik uuesti uut funktsionaalsust arendama hakata.

2.2.3 Scrumi ja Kanbani võrdlus

Kuigi esmapilgul tundub, et Scrum ja Kanban on täiesti erinevad ning neil ühist väga palju ei ole, on neil siiski ühine baas, mis lubab neid võrrelda.

Mõlema metoodika kohta võib öelda:

- on paindlik;
- on eelduseks, et meeskond „tõmbab“ ise tööülesandeid eest ära;
- limiteerivad käsiloleva töö koguse (Scrum sprindi mahuga, Kanban töövoo etappide mahuga);
- eesmärgiks on leida protsessi kitsaskohad ning need parandada;
- fookus on tarnida võimalikult kiirelt ja tihti;
- kliendi tagasiside tehtud tööle on kiire;
- baseeruvad iseorganiseeruvatel meeskondadel;
- nõuavad suuremate tööde tükeldamist pisikesteks iseseisvalt tarnitavateks tükkideks;
- arendusprotsessi optimeeritakse mõõdikute abil (Scrumi puhul meeskonna võimsus, Kanbani puhul töövoo läbimise kiirus).

Siiski on Scrum orienteeritud selliste tarkvaraarendusprojektide läbiviimiseks, mille puhul on oluline ka hinnata tööde mahtu, et oleks võimalik hinnata kas ja mis valmib etteantud tähtaja või eelarve piirides. Kanbani puhul tööde mahu hindamine ei oma tähtsust.

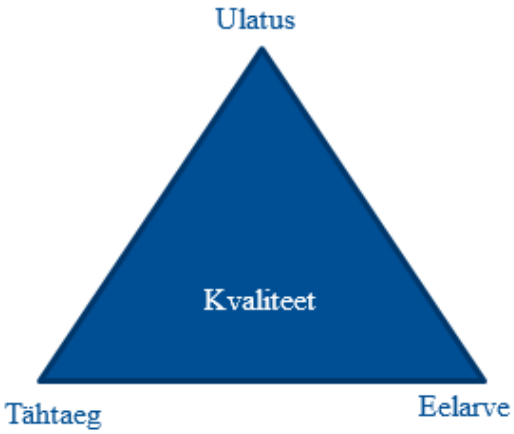
3. Tarkvaraarendusmetoodikate omadused

Käesolevas peatükis kirjeldame projekti iseloomustavad terminid ning kirjutame lahti erinevate tarkvaraarendusmetoodikate neid iseloomustavad omadused. Selleks, et paremini kirjeldada tarkvaraarendusmetoodika omadusi, toome sisse mõned projekti iseloomustavad terminid:

- raudne kolmnurk;
- Stacey maatriks.

Raudne kolmnurk

Iga projekti kolm olulisemat omadust, mille põhjal on võimalik projekti hinnata, on projekti ulatus, eelarve, tähtaeg. On selge, et muutes ühte, peavad muutuma ka teised, vastasel juhul kannatab projekti kvaliteet. Sellist seoste komplekti nimetatakse ka klassikalise projektijuhtimise „raudseks kolmnurgaks“ (ingl. *Iron Triangle*) (Joonis 3).

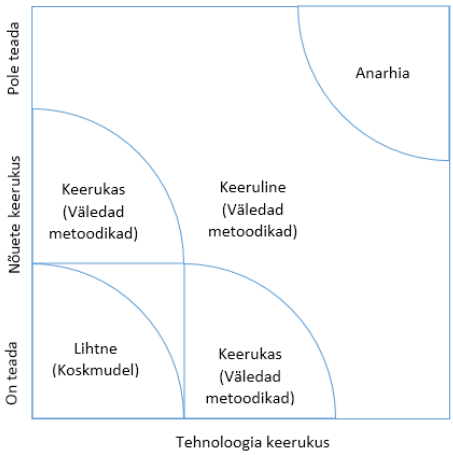


Joonis 3. „Raudne kolmnurk“

Kolmnurka nimetatakse „raudseks“ seetõttu, et kui muuta ükskõik millist kolmest nurgas olevast omadusest, siis peavad muutuma ka teised (või üks teistest) või siis kannatab projekti tulemi kvaliteet. Ei ole võimalik saavutada sama kvaliteeti suurendades projekti ulatust, samas jättes projekti tähtaja ja eelarve samaks.

Stacey maatriks

Projektide iseloomu hindamiseks kasutatakse Stacey maatriksit (Joonis 4). Stacey maatriks aitab hinnata projekti keerukust ning valida sobivaima tarkvaraarenduse metoodika projekti läbiviimiseks. Stacey maatriks kujutab endast kahte mõõdet – vertikaalteljel on projekti nõuete selgus ja horisontaalteljel projekti tehnoloogiline/protsesside keerukus.



Joonis 4. Stacey maatriks.

Projekti nõuete skaala ühes otsas on kõik nõuded täpselt teada, skaala keskel on osa nõudeid teada, osa teadmata ning skaala teises otsas nõuded pole teada. Projekti tehnoloogiate ja protsesside skaala ühes otsas on lihtsad, teada ja tuntud tehnoloogiad, juurdunud ja toimivad tööprotsessid. Skaala keskel on segu tehnoloogiatest, millest osa on teada, osa uued, tööprotsessid osaliselt paigas. Skaala teises otsas kõik tehnoloogiad uued või tundmatud, meeskonnal varasem kokkupuude puudub, tööprotsessid puuduvad või ei ole juurdunud.

3.1 Koskmudeli omadused

Koskmudeli üheks positiivseks omaduseks on, et erinevaid tarkvaraarenduse etappe saab tellida erinevatelt tarkvaraarenduse ettevõtetelt, samuti töö tulemina iga etapi tulem on korralikult dokumenteeritud ja valideeritav/auditeeritav. Siit tuleneb omakorda ka järgmine koskmudeli eelis – kuna iga etapi väljund järgmise etapi sisendina on lõplik ja valideeritud, arendamise hetkeks on arhitektuur ja disain fikseeritud, siis nõuded arendusmeeskonnale ei ole väga kõrged. See on antud mudeli eeliseks, kuna IT-sektor kasvab kiiresti ning nõudlus IT-spetsialistide järele kasvab. Kuigi infotehnoloogia on populaarne ja sellesse valdkonda suundujate ja infotehnoloogiat õppivate inimeste arv on suur, võtab kogemuste omandamine siiski aega ning professionaalseid analüütikuid, arhitekte ja disainereid napib. On positiivne, kui tipptasemel spetsialiste on võimalik kasutada projektis ainult osaliselt ning seejärel vabastada järgmise projekti jaoks ning tarkvaralahendus on võimalik seejärel realiseerida ka vähemkvalifitseeritud meeskonnaga.

Koskmudeli puhul ei pea kogu meeskond olema 100% projektis hõivatud ning üks inimene võib osaleda samaaegselt mitmes projektis. Kuna kõik tööd on koskmudeli puhul ette teada ja planeeritud, siis selline ressursside jagamine probleeme ei tekita. Oluline on, et planeeritud hetkel on vajalik ressurss planeeritud mahus saadaval.

Klassikalise koskmudeli ideaalvariandi korral on kui raudse kolmnurga kõik kolm nurka on fikseeritud. Koskmudeli eesmärgiks on realiseerida projekti ulatus tähtaja ja eelarve piirides, prioriteediks on ulatus.

Siit tuleneb ka üks tingimus selle tarkvaraarenduse protsessi valikuks – nõuded peavad olema väga täpselt kirjeldatud, projekti ulatus, eesmärk, tähtajad ei tohi projekti jooksul muutuda. Koskmudel on mõeldud stabiilsete pikemaajaliste projektide juhtimiseks.

Koskmudeli miinuseks on, et peale analüüsi etapi lõppu sisuliselt puudub võimalus muudatusi teha, ilma, et see mõjutaks teisi projekti iseloomustavaid omadusi (Joonis 3) ning tellija

näeb projekti tulemit projekti lõpus, kui kõik on juba valmis. Negatiivne omadus on see seetõttu, et muudatusi tuleb alati – suur hulk süsteemi nõudeid selgub alles arendustöö käigus. Koskmudeli puhul muudatused on küll võimalikud aga väga ressursikulukad – kogu projekt tuleb ajutiselt seisma panna, nõudeid täiendada, projekt uuesti planeerida ning alustada uuesti algusest.

Koskmudeli positiivse küljena võimaldab mudel realiseerida kuitahes mahukaid projekte kuitahes suure meeskonnaga – kui projekti ulatus on fikseeritud, on see alati võimalik tükeldada erinevateks alamprojektideks, jagada tükid erinevate meeskondade vahel, realiseerida üksteisest sõltumatult.

3.2 Scrumi omadused

Scrumi puhul ei pea projekti kogu ulatus olema ette teada, nõuded selguvad projekti käigus. Toote tööjärg ei pea olema lõpuni ette valmis analüüsitud ning soovilugude detailne kirjeldamine tehakse vahetult enne arendussprindi algust. See eeldab kliendi pidevat valmidust kogu arendustsükli jooksul projektis aktiivselt osaleda.

Siit tuleneb üks Scrumi ja üldse väledate metoodikate miinus – tooteomanik peab ise omama väga kõrgel tasemel valdkonna kompetentsi ning suutma operatiivselt kõigile töö käigus tekkivatele küsimustele vastata, vastasel juhul jääb arendus seisma kuniks ülesannet selgitatakse. Ei ole võimalik tellida väliselt partnerilt ette nõuete analüüsi ja kirjeldust, kogu kompetents peab jooksvalt kogu projekti kestel kättesaadav olema.

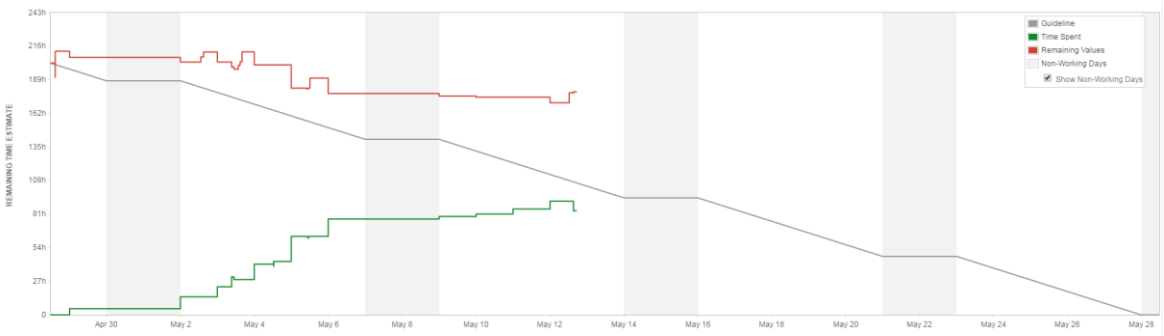
Scrum eeldab pühendunud ja kompetentset meeskonda – Scrumi edu aluseks on väga professionaalsed analüütikud ja arhitektid. Juba esimestest soovilugudest alates peab meeskond suutma valida kogu projekti jaoks õige arhitektuuri ehk kandva skeleti, mille peale kogu järgnev süsteem üles ehitada, vastasel juhul tekib varem või hiljem tehniline võlg ning loodud süsteemi arhitektuuriline lahendus tuleb ümber ehitada, mis on enamasti väga töömahukas ülesanne. Scrum ei toimi kui meeskond ei ole pühendunud, kui projekti liikmed ei osale projektis täiskohaga või on jagatud mitme projekti vahel. Scrum eeldab ühiselt ühele eesmärgile pühendunud meeskonda. Scrumi meeskond peab olema multifunktsionaalne ja kõigi meeskonna liikmete kompetentsid kokku peavad katma kõik projekti jooksul vajalikud kompetentsid. Meeskonda tavaliselt ei kaasata ajutiselt väliseid kompetentse. Efektiivne Scrumi meeskond on 7±2-liikmeline, suurema meeskonnaga muutub arendus ebaefektiivseks kuna suuremale meeskonnale ei jõua klient igapäevaselt tööd ette anda. Kliendi poolt täiendavaid inimesi kaasata ei ole mõistlik – tööjärjel peab olema üks omanik, kes määrab prioriteedid, tellib ja võtab vastu valminud funktsionaalsuse.

On võimalik tekitada mitmeid Scrumi meeskondi ning jagada funktsionaalsus meeskondade vahel (Scrum of Scrums), kuid kuna detailne spetsifikatsioon tekib töö käigus ja kõik lahendused tuleb teiste Scrumi arendusmeeskondadega igapäevaselt kooskõlastada, kasvab ebaefektiivsus.

Scrumi puhul on arendussprint kindla pikkusega (tavaliselt 2-4 nädalat) arendustsükkel, mille jooksul realiseeritakse funktsionaalsus vastavalt sprindi tööjärjele.

Scrumi meetrika on järgmine – kõigi soovilugude keerukust hinnatakse punktides (SP, ingl. *StoryPoint*), iga sprindi jooksul realiseeritakse mingi kogus SP-sid, mistõttu juba paari esimese sprindi jooskul kujuneb välja meeskonna võimsus (ingl. *Velocity*) ehk mitu SP-d suudab meeskond sprindi jooksul realiseerida. Kuna kogu tööjärg on hinnatud, on alati võimalik välja joonistada toote jääktöö graafik (ingl. *Product Burndown Chart*)(Joonis 1), mille baa-

sil omakorda on võimalik öelda, ligikaudu millise funktsionaalsuse jõuab tähtajaks või antud eelarve piirides valmis. Scrumi meetrika ei keskendu projekti kogumahule ja juba kulunud ajale vaid on tulevikku vaatav ehk keskendub sellele kui palju on veel jäänud teha.



Joonis 1. Jääktöö graafik

Raudse kolmnurga (Joonis 3) vaates on Scrumi projekti puhul fikseeritud tavaliselt kas tähtaeg või eelarve ehk siis Scrumi abil realiseeritakse enamasti projekte, mille eesmärgiks on:

- realiseerida tähtajaks maksimaalne võimalik hulk funktsionaalsust prioriteetide järjekorras;
- realiseerida etteantud eelarve piirides maksimaalne võimalik hulk funktsionaalsust, samuti prioriteetide järjekorras.

Scrumi eelis on, et väga lühikese ajaga on võimalik tarnida kliendile juba esimesi tükke tarkvarast, klient saab need toodangusse tarnida ning raha tagasi teenima hakata. Scrumi suurim kasu tuleb välja realiseerides Scrumi abil keskmisest keerulisemaid projekte, mille puhul projekti ulatuse määramatus on suur. Näiteks täiesti uue toote või süsteemi arendus.

3.3 Kanban omadused

Sarnaselt Scrumile on ka Kanbani eelduseks pühendunud ja kompetentne meeskond – väga professionaalsed analüütikud ja arhitektid.

Raudse kolmnurga (Joonis 3) vaates ei ole Kanban projekti puhul fikseeritud tavaliselt ükski nurk, st nii eelarve, ulatus kui ka tähtaeg on fikseerimata. Kanbani kasutatakse juhul kui eesmärgiks on tarnida võimalikult lühikese aja ja kuluga funktsionaalsust, loomulikult arvestades prioriteete.

Kanbani efektiivsuse mõõdikuks on ühe sooviloo kogu tarkvaraarenduse töövoo kõigi etappide läbimise kiirus ehk kui kiiresti jõuab soovilugu järjekorda jõudmisest tarnesse. Vähendades käsiloleva töö kogust enamasti kasvab ühe sooviloo tarnesse jõudmise kiirus aga langeb meeskonna hõive, kuna kõigil pole alati antud sooviloo juures midagi teha. Tõstes käsiloleva töö kogust on efekt vastupidine. Tuleb leida optimaalne käsiloleva töö kogus iga töövoo etapi jaoks.

4. Parimad praktikad ning nende sobivus erinevate tarkvaraarendusprojektide läbiviimiseks

Teades eelpool kirjeldatud kolme tarkvaraarenduse metoodika tugevaid ja nõrku külgi, tekib siiski küsimus, millal millist tarkvaraarenduse metoodikat eelistada. Sellele küsimusele vastuse leidmiseks võtame maailmast mõningad parimad praktikad, toome välja nende põhilised tähelepanekud ning analüüsime neid erinevate projekti iseloomustavate omaduste võtmes.

Võrdluseks ja otsustamiseks piisavate andmete kogumiseks viiakse läbi projekti analüüs. Selleks oleme valinud traditsioonilise koskmudeliga võrdlemiseks ja analüüsiks ühe Scrumi praktika ning ühe Kanbani praktika. Alljärgnevalt toome välja need praktikad ning seejärel analüüsime neid projekti iseloomustavate omaduste abil.

4.1 Meeskonnad, mis lõpetavad töö varem, kiirendavad rohkem: Mustrid kõrgel tasemel Scrumi meeskondadele

Jaanuaris aastal 2014 tegid J. Sutherland, N. Harrison ja J. Riddle 47. Hawaii rahvusvahelisel konverentsil "Systems Sciences (HICSS)" ettekande teemal "Meeskonnad, mis lõpetavad töö varem, kiirendavad rohkem: Mustrid kõrgel tasemel Scrumi meeskondadele" [5]

Ettekandes kirjeldati Scrumi mustreid, ettekande sisu lühidalt oli järgmine.

Scrum põhineb mustritel. Scrumi mustrid on üldised korduvkasutatavad lahendused Scrumi raamistikkus esinevatele sagedamini esinevatele probleemidele. Scrumi struktuur on lihtne ja üles ehitatud aitamaks Scrumi meeskonnal kohaneda muudatustega kuid Scrum ei lahenda kõiki probleeme. Kui Scrum on juurutatud, siis Scrum aja jooksul areneb, et lahendada peamisi tekkivaid probleeme.

Igal aastal esitatakse Scrum PLoP (Pattern Languages of Programs) konverentsil üha uusi Scrumi mustreid, need läbivad läbivaatusprotsessi ning kui muster annab väärtust, siis see lisatakse Scrumi mustrite hulka. Kuna erinevaid mustreid on palju, siis on need jagatud alamhulkadeks.

Peamisi Scrumi mustrite alamhulki on üheksa.

1. **Stabiilsed Scrumi meeskonnad.** Hoides meeskonnad stabiilsena ja vältides inimeste liikumist projektide ja meeskondade vahel on meeskonnal stabiilne suutlikkus, mille teadmine võimaldab äripoolel prognoose teha.
2. **Ennustamine.** Eelmiste sprintide suutlikus on kõige usaldusväärsem mõõdik ennustamaks järgmise sprindi suutlikkust.
3. **Fokuseerimine.** Fokuseerides kogu meeskonnaga ühele kõige olulisemale tulemile sprindi tööjärjes, saab see kõige kiiremini valmis, seejärel fokuseeritakse olulisuselt järgmisele tulemile.
4. **Katkestamine.** Uute tööde ilmnedes, mis ei ole sprinti planeeritud, need tööd katkestatakse ning nende tööde jaoks on välja arendatud uute tööde haldamise protsess.
5. **Korrektne kood.** Eesmärgiks on täiesti korrektne lähtekood iga päeva lõpuks.
6. **Hädaolukorra plaan.** Takistuste ilmnedes, mis ei luba sprinti planeeritult jätkata, käivitatakse hädaolukorra plaan. Sprint ei tohi ohtu sattuda.
7. **Kaizen.** Võetakse kõige olulisem tagasivaatekoosolekul kirjeldatud takistus ja lahendatakse see enne käesoleva sprindi lõppu. Selleks lisatakse Scrumi protsessi parendus sprindi tööjärge nagu tavaline sprindi tulem ja lahendatakse sprindi käigus. Sellist protsessi abil protsessi parandamist nimetatakse Kaizen [6] (jaapani keeles „parendus")

8. **Õnnelikkuse mõõtmine.** Meeskonna liikmete õnnelikkust hinnatakse õnnelikkuse skaalal ning kui meeskonna õnnelikkus muutub järsult, siis on oluline tegeleda meeskonna õnnetunde parandamisega. See ennetab meeskonna motivatsiooni ja suutlikkuse langust.
9. **Meeskond, mis lõpetab sprindi funktsionaalsuse varem, areneb kiiremini.** Scrumi puhul ei ole oluline meeskonna võimsus, vaid meeskonna areng. Lõpetades sprindi ennetähtaegselt, on meeskonnal võimalik ülejäänud aja jooksul mõelda selgemalt, mida ja kuidas nad teevad, eemaldada takistusi, valmistada ette järgmise sprindi tööjärge, arendada „võitmise suhtumist“ ja tõsta ennustuste täpsust.

Esimesed kaks mustrit aitavad meeskonnal valmistuda edukaks arendussprindiks. Mustrid kolmandast kuuendani aitavad meeskonnal tegeleda peamiste sprindi jooksul tekkida võivate probleemidega. Seitsmes ja kaheksas muster võimaldavad meeskonnal tõusta üli-tootlikkuse tasemele ning üheksas muster on eelnevate mustrite rakendamise tulemusel tekkiv positiivne järelmõju.

4.2 Kanbani tõmme ja voolamine – Läbipaistev töövoog kõrgema kvaliteedi ja suurema tootlikkuse saavutamiseks tarkvaraarenduses

Septembris 2013 tegid H. K. Raju ja Y. T. Krishnegowda Bangalore 5. rahvusvahelisel konverentsil „Communication and Computing (ARTCom 2013)“ ettekande teemal „Kanbani tõmme ja voolamine – Läbipaistev töövoog kõrgema kvaliteedi ja suurema tootlikkuse saavutamiseks tarkvaraarenduses“ [7].

Ettekandes räägiti erinevatest tarkvaraarendusmetoodikatest (koskmudel, Scrum, Kanban), põhifookus oli Kanbani juurutamise praktikatel.

Kanban toob inkrementaalsed täiendused protsessi, mille peal see juurutatakse. See elimineerib sprintide iteratiivsuse. Läbi pideva tarnimise fokuseerib Kanban ülikiirelt reageerivale meeskonnale, mitte jäikadele ajalise pikkusega fikseeritud iteratsioonidele. Kanbani eesmärgiks tarkvaraarenduses on arendada tarkvara, mis sisaldab ainult kliendi poolt soovitud funktsionaalsust.

Ettekande autorid toovad oma kogemustele ja kirjandusele tuginedes välja kaheksa soovituslikku sammu Kanbani juurutamiseks tarkvaraarenduses.

1. **Luua läbipaistvus tehtavatesse töödesse ja viisidesse, kuidas seda tööd tehakse.** Kanbani tahvlil tarkvaraarenduse tööprotsessi erinevate sammude ja iga protsessi läbiva tööülesande oleku visualiseerimine toob meeskonnale rohkem läbipaistvust. On näha, mis staatuses iga tööülesanne on ja kes parajasti millise tööülesande täitmisega tegeleb. On võimalik tuvastada protsessi pudelikaelad ning astuda samme pudelikaelte eemaldamiseks.
2. **Vähendada ooteaegu protsessis piirates WIP-i vastavalt meeskonna võimekusele.** Kanban ei sea ajalist piirangut töö kestusele vaid sunnib piirama hetkel korraga töös olevate tööülesannete arvu. Kanban kasutab WIP-i limiite et muuta tavalist projektijuhtimise paradigmat – selle asemel, et lükata tööülesandeid töösse, tõmmatakse tööülesandeid töösse. Kui töösolevaid tööülesandeid on alla limiidi, võtab meeskonna liige uue tööülesande töösse, kui limiit on täis, siis ootab kuni järgmine töövoosamm töö ära tõmbab.

3. **Vähendada tööülesande tööprotsessi läbimise aega vähendades järjekordi.** Tarkvaraarenduse käigus veedavad tööülesanded palju aega lihtsalt oodates järjekorras – oodates ülevaatust, dokumentatsiooni uuendamist, üleandmist, järgmist etappi jne. See on pudelikael, mis mõjutab tööülesande kogu töövoo läbimise kiirust. Neid pudelikaelu tuleb analüüsida ning leida võimalused kas järjekorra vähendamiseks või seada WIP limiit, et meeskond ei tegeleks uute tööülesannete järjekorda tootmisega vaid järgmises etapis eest ära tõmbamisega.
4. **Võta töösse ainult väärtust loov funktsionaalsus.** Lähtudes statistikast 45% tarkvaraarenduse käigus loodud funktsionaalsusest ei leia kunagi kasutust. See ei loo väärtust. See funktsionaalsus on ebavajalik – lisab süsteemi keerukust, tõstab võimalike vigade riski ja on lihtsalt kulu. Seega tuleks mitte arendada funktsionaalsust, mida keegi just hetkel ei vaja, mitte kirjutada rohkem lähteülesannet kui meeskond suudab arendada, mitte kirjutada rohkem koodi kui meeskond jõuab testida ja mitte testida rohkem kui jõuab paigaldada.
5. **Mõõda ja juhi töövoogu, et kiirendada töötsüklit.** Töötsükkel on aeg tööülesande realiseerimise alustamisest tööülesande valmis olekuni. Tööülesande tööprotsessi läbimise aeg on aeg tööülesande nõude tekkimisest tööülesande tarnimiseni, töötsükli kestus on tööülesande reaalselt töös oldud aeg.
6. **Suurenda tootlikust.** Tootlikkus on väärtust omavate tööülesannete hulk, mis on tarnitud kliendile mingi ajalise iteratsiooni jooksul. Iteratsioonide keskmine tootlikkus võimaldab prognoosida protsessi edaspidist iteratsioonide tootlikkust.
7. **Kasuta parandamisvõimaluste tuvastamiseks mudeleid.** Kanban julgustab kasutama protsessi optimeerimiseks teaduslikku lähenemist. Vältida tuleks katse-eksituse meetodit ja kasutada pigem struktuurset lähenemist (Lean, Six Sigma, Piirangute teooria jpt).
8. **Kirjelda kõik protsessi läbimise eeskirjad selgelt.** Kui tööprotsess ei ole meeskonnale arusaadav ja selge, on väga keeruline rääkida protsessi parendamisest. Ilma selgete eeskirjade ja arusaamiseta on arutelud subjektiivsed ja emotsionaalsed. Tehes protsessi eeskirjad selgeks, on võimalik tuvastada tööprotsessi parendusvõimalusi ning lahendada probleeme objektiivselt ja ratsionaalselt.

Kanbani kasutuselevõtt on esimene samm arendusprotsessi parendamise programmist. On tavaline, et inimesed ei soovi muudatusi oma igapäevases töökorralduses. Kasutades lähenemiseks Kanbani ja kirjeldatud kaheksat sammu, saame aidata kaasa Kanbani kasutuselevõtul ja Kanbani kasutamisest saadava kasu suurendamisel.

4.3 Projekti omaduste analüüs

Järgnevalt kirjeldame, millele iga omaduse puhul tähelepanu tuleks pöörata. Kirjeldatud mõõdikute kaudu kaardistame projekti erinevad omadused ning analüüsides projekti erinevaid omadusi toome välja metoodika sobivuse projekti iseloomuga.

Iga omaduse kaardistamise käigus hinnatakse selle omaduse sobivust erinevate tarkvaraarendusmetoodikatega ning kujuneb välja trend mingi tarkvaraarendusmetoodika sobivuse või mittesobivuse kohta hinnatava projekti läbiviimiseks.

Meeskonna hõive

Väledate tarkvaraarendusmetoodikate puhul on oluline, et meeskond töötaks pühendunult koos ühise eesmärgi nimel – seda on oluliselt lihtsam saavutada kui kogu meeskond on

samaaegselt hõivatud ühe projektiga. Jagades meeskonda samaaegselt mitme projekti vahel, on meeskonna liikmete töö hakitud ning kuna igasugune ümberfokuseerimine võtab aega, siis töö efektiivsus langeb ning on keeruline hoida meeskonda motiveerituna.

Klassikalise koskmudeli puhul ei ole oluline, et meeskond tegeleks korraga ainult ühe projektiga. Oluline on, et vajalik ressurss oleks saadaval planeeritud ajal ja planeeritud mahus. Millega see ressurss enne või pärast hõivatud on, ei ole oluline.

Meeskonna kompetents

Väledate tarkvaraarendusmetoodikate puhul on oluline, et tarkvaraarenduse sprindi jooksul realiseeritavad soovilood ja soovilugude raames realiseeritavad tööd ei oleks personaalsed, st. igaüks meeskonnast võiks võtta ühe töö valmides kohe ette järgmise. See aga eeldab, et meeskonna keskmine kompetents on kõrge. Scrumi õige rakendamise käigus meeskonna tehniline kompetents ühtlustub kiiresti – loodavad tehnilised lahendused arutatakse ühiselt läbi, kokku on lepitud koodi vorming ning seda jälgitakse, arendussprintide tagasivaatekoosolekutel kohendatakse arendusprotsesse vastavalt meeskonna efektiivsusele.

Klassikalise koskmudeli puhul arendusmeeskonna kompetents ei pea olema kõrge – kogu töö on ette planeeritud, tehniline lahendus kirjeldatud, arendamine toimub spetsifikatsiooni järgi. Küll aga eeldab klassikalise koskmudeli kasutamine väga kompetentseid analüütikuid, arhitekte, disainereid, projektijuhti, kes suudavad projekti nõuded, nõuetele vastava tehnilise lahenduse, tehnilise lahenduse realiseerimiseks vajalikud tööd ja tööde tegemiseks vajaliku ressursi kirjeldada projekti alguses. Klassikalise koskmudeli miinuseks on, et töö käigus meeskond igapäevaselt omavahel kogemusi ei jaga, meeskonna kompetents ei ühtlustu ning iga meeskonna liige ei ole võimeline tegema kõiki projekti käigus vajalikke töid.

Leping

Tarkvaraarendustööde hankimiseks kasutatakse Eestis tavaliselt kahte tüüpi lepinguid:

- fikseeritud hinna, ulatuse ja tähtajaga leping;
- kulupõhine T&M (Time & Material) leping.

Fikseeritud hinna, ulatuse ja tähtajaga lepinguid kasutatakse enamasti avalikus sektoris avalikel hangetel tarkvaraarendusprojekti hankimisel. Projekti ulatus ja tähtaeg on enamasti hankija poolt kirjeldatud, hankeleping sõlmitakse soodsaima pakkujaga. Ostetakse valmislahendust soodsaima hinnaga.

Kulupõhist T&M lepingut kasutatakse Eestis enamasti erasektoris kui ostetakse pikaajalist tarkvara arenduspartnerit. Viimasel ajal kasutatakse kulupõhist T&M lepingut ka avalikus sektoris tarkvaraarendusteenuse ostmisel.

Kõrvutades neid kahte lepingutüüpi tarkvaraarenduse metoodikatega on selge, et kui projektil on „raudse kolmnurga“ järgi lepingus fikseeritud nii tähtaeg, ulatus kui ka eelarve, on mõistlik selle projekti tööde teostamiseks kasutada klassikalist koskmudelit. Seda seetõttu, et sellisel juhul kohe projekti algul planeeritakse projekti tehnilised lahendused, ajakava ja ressursid selliselt, et need mahuksid projektis etteantud piiridesse. Iga projekti ulatuse muudatus tähendab lepingu muudatust.

Kuna väledat arendusmetoodikat kasutava projekti peamised eesmärgid on:

- realiseerida tähtajaks maksimaalne võimalik hulk funktsionaalsust prioriteetide järjekorras;
- realiseerida etteantud eelarve piirides maksimaalne võimalik hulk funktsionaalsust, samuti prioriteetide järjekorras,

siis on väledatele arendusmetoodikatele sobivam kulupõhine T&M leping. Esimesel juhul ei ole fikseeritud projekti ulatus ega ka eelarve, aga on fikseeritud tähtaeg; teisel juhul on aga fikseeritud eelarve, kuid ei ole fikseeritud projekti ulatus ega tähtaeg. Mõlemal juhul oleks sellist projekti väga keeruline realiseerida fikseeritud hinna, ulatuse ja tähtajaga lepingu raames.

Siiski ei määra lepingutüüp üheselt soovitatavat kasutatavat tarkvaraarenduse metoodikat – oluline on meeskonna kogemus erinevat tüüpi tarkvaraarenduse metoodikate kasutamisel. Kui on kogenud Scrumi meeskond, siis on mõistlik mitte hakata muutma sisseharjunud ja juurdunud tarkvaraarenduse protsesse, vaid kasutada fikseeritud hinna, ulatuse ja tähtajaga lepingu täitmiseks Scrum tarkvaraarenduse metoodikat. Sellisel juhul kasutatakse enamasti „Make it Work, Make it Fast, Make it Pretty“ iteratiivset lähenemist. Selline lähenemine lisab küll teatava keerukuse projekti hinna, ulatuse ja tähtaja hoidmisel aga tagab projekti ulatuse mahtumise projekti eelarve ja tähtaja piiridesse.

Kliendi ressurss

Kliendi ressursi hindamisel on oluline pöörata tähelepanu ressursi saadavusele. Oluline on aru saada, kas kliendi poolt on ressurss saadaval kogu projekti vältel või on võimalik kliendi inimesi kaasata perioodiliti.

Väledate arendusmetoodikate puhul on oluline, et kliendi ressurss oleks kaasatud kogu projekti vältel. Kogu projekti vältel on oluline kaasata kliendi esindaja arendusprotsessi – kliendi esindaja peab tükeldama tööd, seadma prioriteedid, täpsustama lähteülesande, võtma vastu realiseeritud funktsionaalsuse ning andma tagasisidet.

Klassikalise koskmudeli puhul kliendi ressursi saadavus kogu projekti vältel oluline ei ole. On oluline, et kliendi esindaja oleks saadav analüüsi ja planeerimise faasis, kinnitaks lähteülesande ja projektiplaani. Projekti käigus kliendi esindaja projekti arendusprotsessis igapäevaselt osalema ei pea.

Kliendi kompetents

Kliendi kompetentsi hindamisel on oluline teada, kas klient omab põhjalikku kompetentsi ja teadmist tarkvara tulevastest omadustest ning on võimeline kohe projekti algul koostama detailse ja lõpliku lähteülesande või selline kompetents puudub.

Klassikalise koskmudeli puhul on projekti edukuse vaatepunktist kriitiline, et kliendil on põhjalik valdkonna kompetents, kõik protsessid on detailideni teada ja kohe projekti algusfaasis on võimalik kirjeldada ärireeglid ning seosed teiste süsteemidega ja koostada detailne lähteülesanne, mille baasil tarkvara realiseerima hakata.

Scrumi puhul on nõutav, et kliendil oleks visioon, üldisem nägemus, millist ärilist eesmärki loodav tarkvara toetama hakkab. Detailne valdkonna kompetents ei ole kohe projekti algul vajalik, lähteülesanne täpsustatakse jooksvalt arendusprotsessi käigus.

Projekti tüüp

Stacey maatriksilt (Joonis 4) on näha, et klassikaline koskmudel sobib projektidele, mille puhul nõuded on üldiselt teada ning tehnoloogiad samuti teada ning nendega on varasem kokkupuude olemas. Tänapäevaste tarkvaraarendusprojektide puhul satub siia kategooriasse väga väike osa tarkvaraarendusprojektidest.

Maatriksi vastandnurgas on projektid, mille puhul nõuded ei ole teada ning ka projekt baseerub tehnoloogiatel, mille osas meeskonnal kompetents puudub. Selliste projektide läbi-

viimiseks pole ükski tarkvaraarenduse metoodika hea ning selliste projektidega oleks mõistlik mitte alustada – otstarbekas oleks uuesti hinnata, mis on projekti eesmärk ja mida üritatakse projektiga saavutada.

Enamus tänapäevastest tarkvaraarendusprojektidest on Stacey maatriksi järgi klassifitseeritavad kui keerukad või keerulised, selliste projektide läbiviimiseks sobivad väledad tarkvaraarendusmetoodikad suurepäraselt. Maatriksil on keerukad projektid eraldi välja toodud põhjusel, et kui kas projekti nõuded või tehnoloogiad ja protsessid on väga hästi teada, siis võib nende projektide läbiviimisel kasutada ka klassikalisel koskmudelil baseeruvaid metoodikaid kuid väledate metoodikate tõeline eelis tuleb välja keeruliste tarkvaraarendusprojektide realiseerimisel.

Skoobi selgus

Selleks, et otsustada, milline tarkvaraarenduse metoodika on projekti läbiviimiseks sobivam, tuleb analüüsida ja hinnata projekti skoobi selgus. Klassikalise koskmudeli puhul on oluline, et projekti skoop on täpselt teada ning on võimalik kogu projekti tööd täpselt kirjeldada ning hinnata. Juhul, kui projekti üldine skoop on küll teada kuid konkreetsed nõuded vajavad täpsustamist või võivad muutuda, on otstarbekas kasutada väledaid tarkvaraarendusmetoodikaid (nt. Scrum). Projektide puhul, kus ka projekti üldine ulatus ei ole teada, on mõistlik valida projekti läbiviimiseks muudatuste haldusel põhinev metoodika (nt. Kanban) või siis projektiga mitte alustada ning uuesti hinnata, milleks projekt ellu kutsutakse ning mida üritatakse projektiga saavutada.

Skoobi muudatuste tõenäosus

Klassikalise koskmudeli puhul on skoobi muudatuste sisseviimine väga töömahukas ja aeganõudev – kogu projekt tuleb seisata, analüüsida muudatuste mõju projektile, vajadusel projekt uuesti planeerida ning uuesti töödega uue plaani järgi alustada.

Kui projekti skoop ei ole selge või on projektis eeldada suuremat hulka muudatusi, siis on mõistlik kaaluda projekti läbiviimiseks väledaid metoodikaid, mis on välja töötatud just muudatustele kiireks reageerimiseks.

Kliendi rahulolu

Reeglina mõjutab kliendi rahulolu arendusprotsessi läbipaistvus, tihe kommunikatsioon, sagedased tarned ning kliendi soovidega arvestamine. Seda kõike toetavad väledad tarkvaraarendusmetoodikad oluliselt rohkem kui traditsioonilisel klassikalisel koskmudelil põhinevad tarkvaraarendusmetoodikad.

Väledad tarkvaraarendusmetoodikad on fokuseeritud kliendi kaasamisele tarkvaraarendusprotsessi, tihedale kommunikatsioonile, sagedasele tarnimisele ja kliendi poolt tulnud muudatusettepanekutele kiirele reageerimisele.

Meeskonna rahulolu

Väledate metoodikate puhul kaasatakse meeskond arendusprotsessi aktiivselt, see tähendab seda, et meeskond saab ise otsustada mida protsessis parandada, kuidas efektiivsemalt arendada, samuti võtab meeskond vastutuse töö tulemi eest. Selline lähenemine tõstab meeskonna rahulolu ning motivatsiooni.

Traditsioonilise koskmudeli puhul meeskonna rahulolule eraldi rõhku ei pöörata, oluline on saada töö tehtud etteantud tähtajaks etteantud mahus. Selline lähenemine pikemas perspektiivis mõjub meeskonna rahulolu ja motivatsiooni langetavalt.

4.4 Järeldused

Analüüsides parimaid praktikaid, võime järeldusena tuua välja erinevate tarkvaraarendusmetoodikate ja projekti iseloomustavate omaduste vaheliste seoste kohta alljärgneva tabeli (Tabel 1).

Tabel 1. Projekti iseloomustavad omadused.

Projekti iseloomustav omadus	**Koskmudel**	**Scrum**	**Kanban**
Meeskonna hõive	Meeskond jagatud projektide vahel	Üks projekt täiskohaga	Üks projekt täiskohaga
Meeskonna kompetents	Kompetents ebaühtlane, planeerimisel vaja spetsialiste	Ühtlane, kompetents kasvab ja ühtlustub kiiresti	Ühtlane, kompetents kasvab ja ühtlustub kiiresti
Lepingu tüüp	Fikseeritud / T&M	T&M / fikseeritud	T&M
Kliendi ressurss kogu projekti vältel	Pole vajalik	Vajalik	Vajalik
Kliendi kompetents	Peab omama kogu valdkonna kompetentsi detailideni projekti alguses, võib tellida töö väljast	Projekti algul piisab visioonist, kompetents peab kogu projekti algul olemas olema	Kompetents kaasatakse jooksvalt vastavalt vajadusele
Projekti tüüp	Lihtne	Keerukas, keeruline	Keerukas, keeruline
Skoobi selgus	Väga oluline	Oluline	Pole oluline
Skoobi muudatuste tõenäosus	Väike	Keskmine	Suur
Kliendi rahulolu	Langeb	Tõuseb	Tõuseb
Meeskonna rahulolu	Langeb	Tõuseb	Tõuseb

Tabelist 1 on näha, et iga tarkvaraarendusmetoodika sobib kõige paremini teatud kindlate omadustega projekti läbiviimiseks. Kaardistades projekti omadused, analüüsides ning võrreldes neid toodud tabelis välja toodud tarkvaraarendusmetoodikate omadustega, on võimalik tuvastada ebakõlad.

5. Kokkuvõte

Käesolevas bakalaureusetöös analüüsiti tarkvaraarenduse metoodikaid kasutades juhtumianalüüsi. Anti ülevaade ja hinnang tarkvaraarenduse metoodikate kasutamisele Põhja-Euroopa suurimas täislahendusi pakkuvas IT-ettevõttes. Otsiti lahendust küsimusele, millist tarkvaraarenduse metoodikat konkreetse projekti korral kasutada.

Hoolimata käesolevas bakalaureusetöös kajastatud tarkvaraarenduse metoodikate välja toodud omadustele ja valikukriteeriumitele, ei ole valik tavaliselt üks-ühene ja valitav tarkvaraarenduse metoodika ei sobi enamasti kõigi valikukriteeriumitega üks-üheselt kokku.

Peamine, mis käesolevas bakalaureusetöös välja toodi, on see, et valik, mis tehakse, peab olema läbimõeldud, ümbritsev keskkond ja selles valitsevad tingimused kaardistatud – sellisel juhul on võimalik teades valitud tarkvaraarenduse metoodika ja ümbritseva keskkonna ebakõlasid ning sellest tulenevaid riske, koostada plaan, mis neist riskidest tulenevad ohud kas elimineerib või viib miinimumini. Enamasti ei sobi ükski tarkvaraarenduse metoodika täpselt olukorraga kuid neid kõiki on võimalik kohendada, tuues juurde teiste tarkvaraarenduse metoodikate elemente, võttes kasutusele ennetavad abinõud ning seeläbi jõuda lahenduseni, mis sobib. Samuti on võimalik teatud piirides muuta ümbritsevat keskkonda mingi tarkvaraarenduse metoodikaga paremini sobivaks.

Teades ja tundes iga tarkvaraarenduse metoodika häid ja halbu külgi, sobivust ja sobimatust ümbritseva keskkonnaga, on võimalik valida projekti läbiviimiseks parim tarkvaraarendusmetoodika ning projekti iseloomu ja metoodika ebakõlad likvideerida või minimeerida. See aitab kaasa projekti õnnestumisele.

6. Kasutatud kirjandus

[1] "Wikipedia: The Free Encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/Semi-Automatic_Ground_Environment.

[2] W. W. Royce, "Managing The Development Of Large Software Systems," 1970. [Online]. Available: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf.

[3] T. E. Bell and T. A. Thayer, "Software Requirements: Are They Really A Problem?," 1976. [Online]. Available: https://static.aminer.org/pdf/PDF/000/361/405/software_requirements_are_they_really_a_problem.pdf.

[4] United States Department Of Defence, "Military Standard," 1985. [Online]. Available: http://www.product-lifecycle-management.com/download/DOD-STD-2167A.pdf.

[5] J. Sutherland and K. Schwaber, Software in 30 days: how agile managers beat the odds, delight their customers, and leave competitors in the dust., John Wiley & Sons, Inc., April 2012, p. 216.

[6] J. Sutherland and K. Schwaber, "http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf," Juuli 2013. [Online]. Available: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf.

[7] D. Anderson, Kanban, Washington: Blue Hole Press, 2010.

[8] J. Sutherland, N. Harrison and J. Riddle, "Teams That Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams," in *Teams That Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams*, Waikoloa, HI, USA, 2014.

[9] M. Imai, "Definition of Kaizen," Kaizen Institute Consulting Group, 1986. [Online]. Available: https://www.kaizen.com/about-us/definition-of-kaizen.html.

[10] H. K. Raju and Y. T. Krishnegowda, "Kanban Pull and Flow — A transparent workflow for improved quality and productivity in software developmet," in *Kanban Pull and Flow — A transparent workflow for improved quality and productivity in software developmet*, Bangalore, 2013.

Lisad

I. Terminid

Toote tööjärg Toote tööjärg on järjestatud nimekiri kõigist toote võimalikest omadustest ja on ainus toote muudatussoovide allikas. Toote tööjärg sisaldab kõiki omadusi, funktsioone, nõudeid, täiendusi ja parandusi, mis moodustavad muudatused, mis tootesse tulevikus sisse viia. Toote tööjärje üksustel on kirjeldus, prioriteet (järjekord), töömahuhinnang ja väärtus.	Product Backlog The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate and value.
Sprindi tööjärg Sprindi tööjärg on komplekt toote tööjärje üksuseid, mis on välja valitud sprindi koosseisu. Lisaks kirjeldab sprindi tööjärg endas plaani toote iteratsiooni ja sprindi eesmärgi saavutamiseks. Sprindi tööjärg teeb nähtavaks kogu töö, mida arendusmeeskond näeb olevat vajalik sprindi eesmärkide saavutamiseks.	Sprint Backlog The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog makes visible all of the work that the Development Team identifies as necessary to meet the Sprint Goal.
Sprint Sprint on kindla ajalise pikkusega etapp, mille jooksul realiseeritakse kasutatav ja potentsiaalselt tarnitav toote täiendus. Sprindid sisaldavad ja koosnevad sprindi planeerimisest, igapäevastest Scrum koosolekutest, arendustööst, sprindi tulemuste ülevaatusest ja sprindi tagasivaatest.	Sprint Sprint is a time-box of one month or less during which a useable, and potentially releasable product increment is created. Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.
Tooteomanik Tooteomanik on vastutav toote maksimaalse väärtuse saavutamise ja arendusmeeskonna töö eest.	Product Owner The Product Owner is responsible for maximizing the value of the product and the work of the Development Team.

Scrum meeskond Arendusmeeskond koosneb professionaalidest, kes töötavad toote potentsiaalselt tarnitava valmis täienduse tarnimise nimel.	Scrum Team The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.
Rüsinameister Rüsinameister on vastutav selle eest, et Scrumi metoodika on arusaadav ja juurutatud. Rüsinameister tagab, et arendusmeeskond järgib Scrumi teooriat, praktikaid ja reegleid. Rüsinameister on Scrumi meeskonna aitav juht. Rüsinameister aitab väljaspool Scrumi meeskonda olijatel mõista millised nende tegevused on aitavad ja millised mitte. Rüsinameister aitab kõigil tegutseda selles suunas, et maksimeerida Scrumi meeskonna poolt toodetavat väärtust.	Scrum Master The Scrum Master is responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules. The Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.
WIP Hetkel korraga töös olevate tööde hulk.	WIP WIP (Work-In-Progress) is work that is currently in progress.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Indrek Majas**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Tarkvaraarenduse metoodika valiku protsess (CGI Eesti AS näitel),
(*lõputöö pealkiri*)

mille juhendaja on Raul Raudsepp ja Helle Hein,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
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.

Tartus, **12.05.2016**