Tallinna Ülikool

Digitehnoloogiate instituut

PAINDLIKU TARKVARAARENDUSMETOODIKA JUURUTAMISEST FIRMA X NÄITEL

Bakalaureusetöö

Autor: Liis Vanem

Juhendaja: Inga Petuhhov

Autor: ..... ,, ... ,, 2016

Juhendaja: ..... ,, ... ,, 2016

Instituudi direktor: ..... ,, ... ,, 2016

Tallinn 2016

AUTORIDEKLARATSIOON

Deklareerin, et käesolev bakalaureusetöö 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 _____ (sünnikuupäev: _____)
(autori nimi)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
_____
_____
_____
(lõputöö pealkiri)

mille juhendaja on _____,
(juhendaja nimi)

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, _____ (digitaalne) allkiri ja kuupäev

SISUKORD

MÕISTED

Kuna autoril oli üheste tõlgete leidmisega probleeme, siis on kasutatud enamjaolt samu tõlkeid, kui Jüri Siilivask oma 2014 aastal kaitstud magistritöös.

Arenduse koskmudel *(ingl Waterfall)* - Tarkvaraarenduse protsessi mudel, kus tegevused edenevad ühest etapist järgmisesse järjestikvoona: kontseptsiooni loomine, algatamine, analüüsimine, projekteerimine, ehitamine, testimine ja haldamine. (Siilivask, 2014)

Huvigrupp *(ingl Stakeholder)* – Huvigrupp on igaüks, kes saab mingit kasu toote edu eest, näiteks kasutajad, klienditeenindus, müügiosakond ja arendusosakond ise. (Heusser & Markus, 2015)

Häkk *(ingl Hack)* - Programmi lähtekood, mis enamasti kujutab endast olemasoleva tarkvara juures avastatud probleemi kiiret lahendust või on mõeldud tarkvara parendamiseks. Klassikalisi häkke kirjutatakse enamasti madalkeeltes, kuid nõnda on hakatud nimetama ka skriptikeeltes kirjutatud koodi. (e-teatmik, kuupäev puudub)

Iteratsioon *(ingl Iteration)* - väikseim võimalik töötsükkel, mille jooksul toodetakse töötav tarkvara. Iteratsioonid võivad üldjuhul olla kuni 3-kuu pikkused, kuid tavaliselt jäävad vahemikku 1 kuni 4 nädalat (vt ka Sprint). (Siilivask, 2014)

Kasutuslugu *(ingl User Story)* - kasutuslugusid kasutatakse paindlikes arendusmetoodikates ärinõuete kirjeldamiseks. (Siilivask, 2014)

Klient – ettevõte või isik, kes tellib tarkvaraarendusmeeskonnalt tööd.

Püstijala koosolek *(ingl Stand-up Meeting)* - Igapäevased koosolekud, mille eesmärk on anda kiiresti ja tõhusalt ülevaade tehtust, edasistest tegevustest ning kõrvaldada võimalikud tekkinud takistused. (Siilivask, 2014)

Scrum - Paindlik tarkvaraarenduse raamistik, mis põhineb sprintidel ja tavalisel koosneb järgmistest osapooltest: toote omanik, Scrum'i meister ja meeskond. (Siilivask, 2014)

Scrum'i meister *(ingl Scrum Master)* - Isik, kes on koolitatud igapäevaste Scrum koosolekute läbiviimise, takistuste kõrvaldamise, meeskonna arengu valdamise ja meeskonna uuenduste jälgimiseks. (Siilivask, 2014)

Sprint - Scrum metoodikas kasutatav sõna iteratsiooni kirjeldamiseks. (Siilivask, 2014)

Toote kuhi *(ingl Product Backlog)* - Toote kuhi on tähtsuse järgi järjestatud nimekiri kasutaja lugusid ja vigu, alates süsteemi kõige väärtuslikumast kuni vähemväärtuslikuni. (Siilivask, 2014)

Toote omanik *(ingl Product Owner)* - Scrum metoodikast pärinev roll, mis on tänapäeval leidnud laialdast kasutust ka sõltumata Scrum metoodikast. (Siilivask, 2014)

SISSEJUHATUS

Selleks, et leida ühine mõistmine agiilse tarkvaraarendusmetoodika kohta kogunesid 17 tarkvaraarendajat 2001 aastal. Nad panid paika neli väärtust, millest arendati agiilse tarkvaraarenduse manifest (*ingl „Manifesto for Agile Software Development“).* Scrum on agiilne raamistik, mis toetub tarkvaraarenduse manifesti väärtushinnangutele. (Scrum Alliance, kuupäev puudub) Nendeks on: (Manifesto for Agile Software Development, 2001)

„Me tahame luua uusi paremaid tarkvaraarendusmetoodikaid. Me hindame:

- enam inimesi ja nende vahelisi suhteid kui protsesse ja arendusvahendeid;
- enam töötavat tarkvara kui täielikku dokumentatsiooni;
- enam kliendi osalust arenduses kui lepingute koostamist;
- enam muudatustele reageerimist kui plaani järgimist.“

Käesolev bakalaureusetöö sündis vajadusest muuta ühes tiimis tarkvaraarendusprotsess läbipaistvamaks, efektiivsemaks ja parandada info liikumist. Firmas, kus see tiim asub, töötab mitu meeskonda erinevate projektidega ning esimesed tiimid läksid Scrum’ile üle juba mitu aastat tagasi. Praeguseks kasutavad enamus tiimid Scrum’i.

Kunagi oli bakalaureusetöö raames kirjeldatud meeskond väiksem ning tellimusi kliendilt tuli vähem, seetõttu ei olnud varem vajadust kasutada mingit kindlat tarvaraarendusmetoodikat. Seoses tööde tellimuse suurenemisega on tiim laienenud ning on kerkinud ka vajadus kasutada mingit süsteemi. Valituks sai arendusmetoodika Scrum, kuna samas firmas kasutavad paljud tiimid juba Scrum’i ning see raamistik on ennast õigustanud. Töö autor on ise ka seotud selle meeskonnaga.

Samm-sammult on juba varem tahetud tiimis Scrum’i juurutamisega algust teha, kuid ei saa öelda, et praegu sujuks töö täielikult agiilsena. Arendused jaotatakse küll “sprintideks”, kuid need ei ole kliendi poolt kinnitatud ning tavaliselt eelnevalt hinnatud. Samuti on võimalik sinna iga hetk töid juurde panna ning iteratsioonide pikkus ei ole selgelt kindlaks määratud . Tavaliselt kestavad need niikaua kuni kõik iteratsiooni valitud tööd on tehtud. Valmis tööd pannakse toodangusse ükshaaval mitte kõik korraga nagu agiilses arenduses tavaliselt tehakse. Nii klient kui meeskond vajavad kindlamat süsteemi ning ajaraame tarkvaraarenduste tegemisel.

Bakalaureusetöö on arendusuuring ning eesmärk on tutvustada agiilset tarkvaraarendusmetoodikat Scrum raamistikus ning hakata seda kasutama tiimis, mis ei ole siiani töötanud agiilselt.

Selleks, et saavutada töö eesmärk on autoril plaanis järgnevad tegevused:

1. Tutvuda Scrum'iga kirjandust läbi töötades
2. Väljaselgitada, kuidas teistes ettevõtte meeskondades on Scrum'i juurutatud
3. Teha plaan Scrum'i kasutusele võtmiseks tiimis
4. Järgides koostatud plaani viia läbi üks sprint
5. Analüüsida toimunud sprinti ja anda soovitused uue sprindi jaoks

Käesolev töö koosneb neljast peatükist. Esimeses peatükis on üldine tutvustus Scrum'i ja seal olevate rollide kohta. Lisaks on ülevaade Scrum'i elementidest nagu koosolekud ja Scrum'i tahvel.

Teises peatükis on täpsem ülevaade tarkvaraarenduspraktikatest, mida kasutatakse Scrum'i raames. Nendeks on ühiktestimine, ümberstruktureerimine, testjuhitud arendus ja pidev integreerimine.

Kolmandas peatükis kirjeldatakse, kuidas peaks toimuma paindlikule tarkvaraarendusmetoodikale üleminek ning kuidas teistes meeskondades kasutatakse Scrum'i. Lisaks on tutvustus, milline olukord on meeskonnas tänasel päeval ning millised on need tegevused, mida peaks tegema selleks, et tiimis juurutada Scrum'i raamistikku.

Neljandas peatükis on kirjeldatud pilootprojekti sprinti ja selle edenemist ning millised õppetunnid saadi, mida järgmises sprindis peaks teisiti tegema.

1 SCRUM'i TUTVUSTUS

Kuni 2005. aastani kasutasid enamus tarkvaraettevõtteid arendustes koskmeetodit. Tihtipeale läksid need projektid üle tähtaja, ei pidanud eelarvest kinni ja tulemus ei vastanud klientide ootustele. Seetõttu oli vaja uut raamistikku, kuidas muuta projekte kiiremaks, efektiivsemaks ja usaldusväärsemaks. (Sutherland, 2014)

Kirjanduses kasutatakse Scrum'i kohta kahte erinevat tähendust – osades allikates on ta meetod, kuid teistes raamistik. Kuna Ken Schwaber, kes on üks Scrum'i loojaid, on öelnud, et Scrum on raamistik mitte metoodika, siis autor otsustas edaspidi kasutada väljendit Scrum'i raamistik. See, et Scrum on raamistik, tähendab ühtlasi seda, et ei ole täpselt ette öeldud, kuidas peab kasutama Scrum'i vaid igaüks võib teda kohaldada vastavalt oma situatsioonile. (Kniberg, 2007)

Scrum'i raamistikus ei toodeta enne arendust lõplikke dokumente, kus on kirjas, kuidas projekt peab lahendatud olema. Lahendusi mõtleb välja tiim, kes teab kõige paremini, kuidas probleeme lahendada. Scrum'i tiimis ei ole tavaliselt tiimijuhti, kes otsustab, milline tiimiliige, mida teeb või kuidas mingit probleemi lahendatakse. Need teemad otsustatakse tavaliselt tiimisiseselt. (Mountain Goat Software, kuupäev puudub) Scrum arenduses on kõige olulisem tiimitöö (Sutherland, 2014).

1.1 Rollid Scrum'i raamistikus

Scrum'i raamistikus eristatakse tavaliselt kolme rolli – tooteomanik, arendustiim ja Scrum'i meister (Kniberg & Mattias, 2010).

Tooteomaniku rollil on tavaliselt kaks ülesannet arendustöös: edastada tiimile visioon ja paika panna reeglid. Meeskonnad töötavad paremini, kui kõigil liikmetel on silme ees ühtne eesmärk ja teavad, mis on nende arendatava toote eelis, kes on konkurendid, kellele toodet müüakse, kuidas areneb tulevikus jne. See loob pikemaajalise suhte nende vahel, kes toodet arendavad ja kasutavad. Toote omanik võib visiooni rõhutada läbi toote kuhja korrastamise, kuid see ei ole kohustuslik. Ta võib selle delegeerida ka kellelegi teisele. Veel on oluline paika panna tiimi jaoks reeglid, milleks on näiteks minimaalne arendatav funktsionaalsus sprindi lõpuks, suurem jõudlus, ressurside vähendamine ning mingitel juhtudel ka tähtajad. (Cohn, 2012) Lisaks ta peab vastama jooksvatele küsimustele ja suhtlema huvigrupiga (Heusser & Markus, 2015).

Scrum’i meistril ei ole mingit mõjuvõimu tiimi üle vaid tema valitseb protsessi. Ta ei või kedagi tiimist lahti lasta kuid ta võib teha ettepaneku, et muuta sprindid kahenädalasteks senise kolme asemel. Tänu piiratud autoriteedile on Scrum’i meistri töö keerulisem kui tavalise projektijuhi oma, kuna ta ei või teha ettepanekuid, mis puudutab projektimeeskonda. Hea Scrum’i meister on tavaliselt vastutustundlik, tagasihoidlik, koostööaldis, pühendunud, mõjukas ja teadlik. (Cohn, 2012) Ta teenib arendustiimi ja tooteomanikku ning vastutab selle eest, et kõik sprindiga seotud inimesed saavad efektiivselt tööd teha (Heusser & Markus, 2015).

Scrum’i tiim või arendustiim on vastutav töötava produkti tootmise eest sprindi jooksul. (Heusser & Markus, 2015) Arendustiim on iseorganiseeriv ja -juhtiv grupp, kus tavaliselt on kolm kuni üheksa inimest. Tavaliselt koonduvad tiimi erinevate oskustega inimesed, kes varem töötasid erinevates osakondades. Tiimile antakse autonoomia selles, kuidas sprindi eesmärki täita. Tiim on tavaliselt edukas, kui kõik liikmed asuvad samas ruumis. (James, kuupäev puudub)

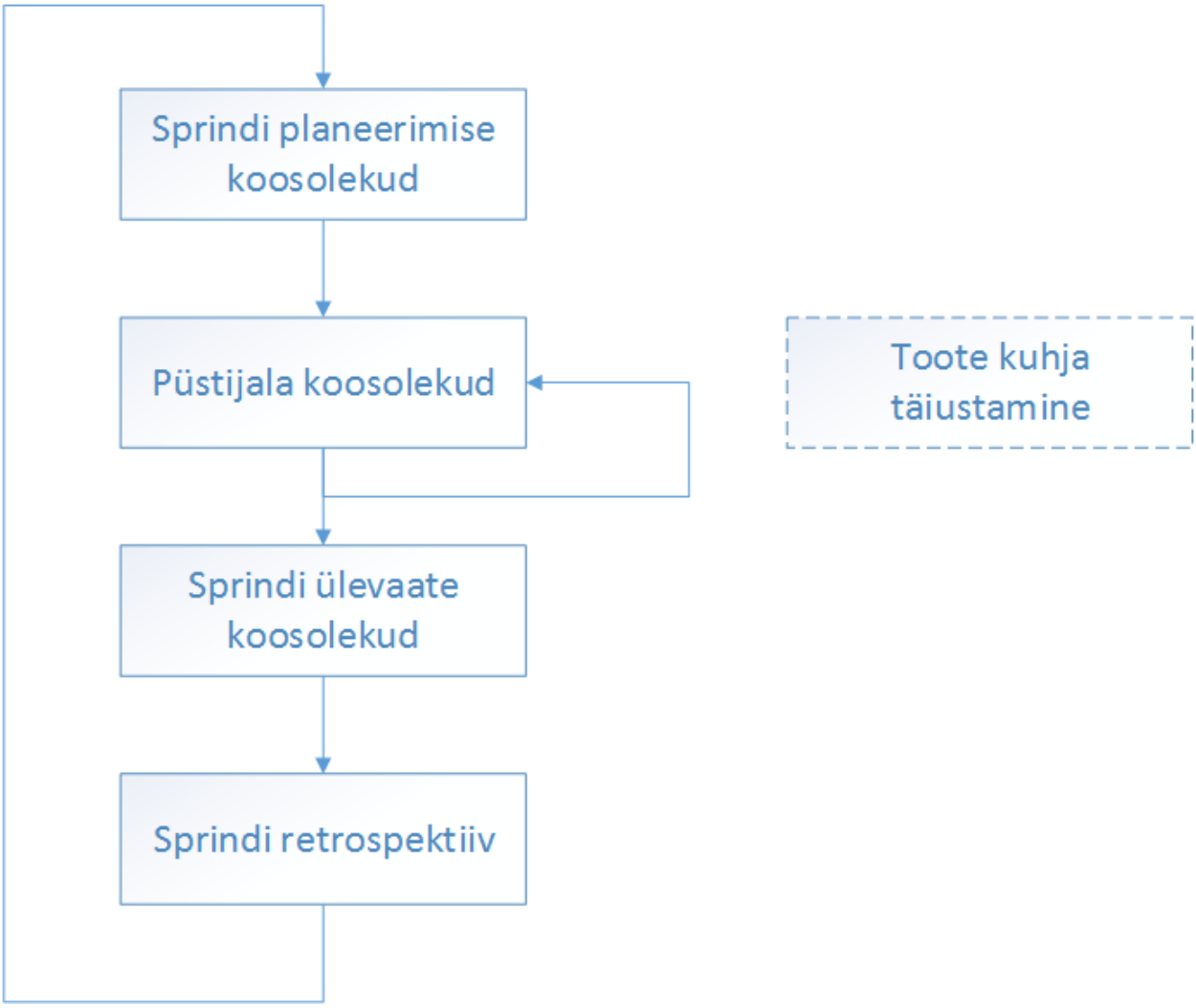
Kuigi huvigrupp ei ole kirjas osana Scrum’i tiimist, siis tavaliselt ei saa Scrum’i tiim ilma huvigrupita hakkama. Huvigrupis on igaüks, kes saab arendustööst kasu, näiteks turundus, kliendid, keskastmejuhid jne. Nad töötavad koos tooteomanikuga ja aitavad tal toote kuhja nimekirja täiendada. Lisaks on nende tagasiside arenduste kohta arendustiimile väga oluline. (Heusser & Markus, 2015)

1.2 Scrum’i elemendid

Selleks, et teha arendusi Scrum’i jälgides, on kokkulepitud koosolekud, mida arendustiim peab läbi viima. Tavaliselt viib neid koosolekuid läbi Scrum’i meister, kuna tema on see, kes peab jälgima, et Scrum toimiks tiimis. Samas ta ise ei pea nendest osa võtma ja tal ei ole õigust otsuseid vastu võtta. (James, kuupäev puudub)

Nendeks koosolekuteks on (vt Joonis 1):

- sprindi planeerimise koosolekud;
- püstijala koosolekud;
- sprindi ülevaate koosolekud;
- sprindi retrospektiiv;
- toote kuhja täiustamine.



Joonis 1. Koosolekud Scrum'i raamistikus (James, kuupäev puudub)

1.2.1 Sprindi planeerimise koosolek

Sprindi planeerimist tehakse iga uue sprindi alguses. Sellel koosolekul peavad kohal olema kõik, kes panustavad käimasolevasse sprinti. (Cohn, 2006) Scrum'i tiim vastab sellel koosolekul kahele põhiküsimusele:

1. Milliseid uusi funktsionaalsusi planeerime arendada järgmises sprindis?

2. Kuidas me planeerime realiseerida uusi funktsionaalsusi?

Vastus esimesele küsimusele asetseb toote kuhjas. Seal on sorteeritud ja tähtsuse järgi pandud funktsionaalsused, mida on vaja tootes realiseerida. (Heusser & Markus, 2015) Toote omanik on vastutav selle eest, et äriliselt kõige kriitilisemad tööd läheks järgmisesse iteratsiooni (James, kuupäev puudub). Toote omanik peab tutvustama Scrum'i tiimile töid, mis on järjekorras prioriteetsed ning seejärel Scrum'i meeskond hindab, kas töö mahub uude sprinti olemasolevate

tööde kõrvale. Sellega seoses lepitakse kokku sprindi eesmärk, mis peab hõlmama neid töid, mis on toote kuhjast sprinti võetud. Võib juhtuda, et tooteomanikul on enne arendustiimiga rääkimist juba sprindi eesmärk välja mõeldud. Või siis arendustiim valib üksikud tööd välja ja nende põhjal pannakse arenduseesmärk kirja.

Teisele küsimusele, kuidas realiseeritakse valitud töid, vastab tavaliselt arendustiim ise. Tiimiliikmed hindavad üldisemal tasemel disaini ja arhitektuuri. Kuna selle käigus võib tiimiliikmetel tulla küsimusi tooteomanikule, siis peaks ka tema olema kättesaadav. Peale seda jätkab tiim ülesannete väiksemateks tükkideks tegemisega.

Sprindi planeerimine ei tohiks võtta aega rohkem kui 5% sprindi ajast. (Heusser & Markus, 2015) Kui sprint on tavaliselt kahenädala pikkune, siis ta kestab 80 tundi. Sellest 5% on neli tundi, mida võiks sprindi planeerimise jaoks kulutada.

Sprindi plaan võib olla väga lihtne – näiteks tavaline tabel või märkmepaberite peale kirjutatud ülesanded. Kuid silmas peab pidama, et väiksemaks tükeldatud ülesanded peab saama kokku panna näiteks kasutuslugudega, mille pärast see töö ette võetakse. (Cohn, 2006)

Sprindi planeerimise tulemusel tekib sprindi kuhi. See koosneb ülesannetest, mis olid varem toote kuhjas. (Heusser & Markus, 2015)

1.2.2 Igapäevased püstijala koosolekud

Püstijala koosolek tähendab seda, et kõik tiimiliikmed seisavad füüsiliselt püsti 15 minutit, kui koosolek kestab. See aitab inimeste tähelepanu hoida. Koosoleku ajal peaksid kõik vastma kolmele küsimusele:

- Mida ma tegin eile, mis aitas arendustiimi jõuda lähemale sprindi eesmärgile?
- Mida ma teen täna selleks, et aidata arendustiimil jõuda lähemale sprindi eesmärgile?
- Kas ma näen mõnda takistust, mis takistab sprindi eesmärgile jõudmast?

Püstijala koosolekud on kohustuslikud Scrum’i meistrile ja Scrum’i tiimile. Seal võivad kuulata ka teised rollid nagu toote omanik ja huvipooled, kuid nad ei saa sellest osa võtta ning see pole neile kohustulik. (Heusser & Markus, 2015)

1.2.3 Sprindi ülevaate koosolekud

Sprindi ülevaate koosolek on suurem koosolek huvipooltega selleks, et demonstreerida, mida tiim on ehitanud selles sprindis ja saada tagasisidet ning otsustada mida arendada järgmiseks (Heusser & Markus, 2015). Veel vaadatakse üle tööd, mida ei jõutud ära lõpetada või alustadagi. Need hinnatakse ümber ja pannakse tagasi toote kuhja, nii et saaks tulevikus tagasi sprinti võtta. (James, kuupäev puudub) On oluline, et arendusmeeskond on avatud ja ei hakka ennast õigustama. (Heusser & Markus, 2015) Võib juhtuda, et neile antud tagasisidest tekivad uued tööd toote kuhja (James, kuupäev puudub).

Sprindi lõpus on väga oluline demonstreerida, mis tiim on valmis saanud, kuna tänu sellele näevad teised inimesed, millega meeskond on tegelenud ning liikmed saavad heakskiitu ja tunnevad ennast hästi. Lisaks on see võimalus huvigruppidel objektiivsemat tagasisidet anda sprindi töödele. See on ka motivatsioon meeskonnale selleks, et nad peavad arendused ära lõpetama ning toodangusse panema (kasvõi testkeskkonda). Kui esitluse käigus selgub, et mõned tööd on tegemata või poolikud, siis on ilmselt kogu tiimil halb tunne ja tänu sellele nad pingutavad järgmises sprindis rohkem (Kniberg, 2007). Demonstreerimise käigus peaks kindlasti keskenduma kasutuslugudele ja mitte näitama ainult uusi funktsionaalsusi vaid esitlema, kuidas need käituvad koos vanadega ja moodustavad terviku.

Sprindi ülevaate koosolek peaks kestma tavaliselt neli tundi, kui sprint on kuuajaline. (Heusser & Markus, 2015) Kahenädalasel sprindil võiks see kesta kaks tundi.

1.2.4 Sprindi retrospektiiv

Retrospektiiv on spetsiaalne koosolek, kus meeskond koguneb peale töö lõpetamist selleks, et üle vaadata oma meetodeid ja tiimitöö (Derby & Diana, 2011). Seal peaksid osalema Scrum'i meister, arendustiim ja tooteomanik. Kõik need rollid saavad kasu, kui nende koostöö sujub. See koosolek on ühtlasi sprindi viimane tegevus. (Heusser & Markus, 2015)

Väga oluline on see, et retrospektiivid toimuksid ja neid ei jäetaks ära. Tavaliselt, kui on kiired ajad tiimides, siis tahetakse neid edasi lükata, kuid õige on just siis korraldada retrospektiiv. (Goldstein, 2014) Väga oluline on meeldiv õhkkond – meeskonnal peab olema mugav ja mõnus olla ning esimeste minutite jooksul peaks kõik tiimiliikmed midagi ütlema. Muidu on üpris suur tõenäosusus, et need, kes on alguses vaiksemad ei ütle midagi kuni koosoleku lõpuni, kuid väga oluline on, et kõik võtaksid aktiivselt osa. (Derby & Diana, 2011) Ideaalne oleks, et retrospektiivile tulevatel osalejatel oleks kaasas nimekiri probleemidest ja soovitustest, nii et kohe saaks teema

juurde asuda. Peamised teemad, millest rääkida, võiksid olla kommunikatsioon, protsess, kvaliteet, keskkond ja oskused.

Kokkuvõtte tegemiseks joonistatakse tahvlile horisontaalne joon, kus kõige vasakul pool on märge „Vajab arendamist“, keskel on „Kõik on korras“ ja paremal pool on „Läheb väga hästi“. Seejärel jaotatakse osalejaile märkmepaberid ja nad saavad asetada oma probleeme ja mõtteid tahvlile sobivasse kohta. Tavaliselt retrospektiivi käigus tehakse uusi kokkuleppeid, millest meeskond peaks kinni pidama hakkama. Soovitatav on need kirja panna tahvlile, mida meeskond kogu aeg saab jälgida. (Goldstein, 2014)

1.2.5 Toote kuhja täiustamine

Selle koosoleku eesmärgiks on ette valmistada toote kuhi järgmiseks sprindiks, see võib võtta aega umbes 5-10% arendusmeeskonna ajast. Toote kuhja täiustamisel on kolm peamist tegevust: (Heusser & Markus, 2015)

- suuremad ülesanded teha väiksemateks;
- hinnata uusi toote kuhja ülesandeid;
- nimetada ülesande vastuvõtmise kriteeriumid.

Need tegevused on omavahel lõimunud. Näiteks mõne ülesande hindamisel võib avastada, et need on vaja sprinti võtmisel teha väiksemateks osadeks. (Heusser & Markus, 2015)

1.2.6 Scrum’i tahvel

Tavaliselt kõige tuntum element Scrum’i projektidest on visuaalne meeskonna ülesannete tahvel. See on tavaliselt kohas, kust ta on kõikidele meeskonnaliikmetele nähtav. Võiks mõelda, et miks tänapäevasel tehnoloogiaajastul peaks kasutama füüsilist tahvlit tööde ülesmärkimiseks. Kuid selle põhjuseks on lihtne inimpsühholoogia. Selles on midagi rahustavat ja meeldivat, kui tõusta oma kohalt üles, jalutada tahvlini, võtta märkmepaber ja panna see „Tehtud“ veergu. (Goldstein, 2014)

Veerud võivad olla erinevad, Ilan Goldstein on soovitanud teha neli veergu: „Alustamata“, „Töös“, „Valmis ülevaatamiseks“ ja „Tehtud“. (vt Tabel 1) Iga märkmepaber esindab kindat ülesannet. Märkmepaberil võiks olla järgmine info: (Goldstein, 2014)

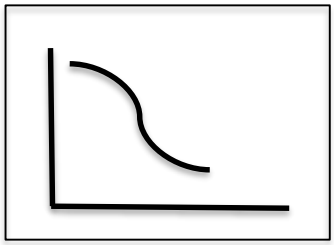
- ülesande number (tavaliselt digitaalsetes keskkondades automaatselt tulev number ülesandele);

- arendaja initsiaalid, kes selle ülesandega tegeleb;
- ülesande kirjeldus;
- töömahu hinnang tundides;
- arendaja lisab jooksvalt juurde, kui kaua aega kulub veel tal selle ülesande lõpetamiseks.

Lisaks tehakse tahvlile tavaliselt ka juurde sprindi lõpuni jäänud aja graafik päevade kaupa *(sprint burndown chart)*. Seda saab hakata koostama pärast sprindi planeerimise koosolekut, kui on igale ülesandele lisatud tunnid, palju see konkreetne ülesanne aega võtab. (Mittal, 2013) Graafikut uuendatakse tavaliselt enne igapäevast püstijala koosolekut, et oleks näha kohe visuaalselt, kui palju on veel jäänud sprindi lõpuni (Goldstein, 2014).

Veel võib lisada tahvlile sprindi ja retrospektiivi eesmärgi, definitsioonid ja põhimõtted, prototüübid, kliendi kommentaarid. (Goldstein, 2014)

Tabel 1. Scrum'i tahvli näidis

Alustamata	**Töös**	**Valmis ülevaatamiseks**	**Tehtud!!!**	**Aeg sprindi lõpuni**
				

2 AGIILSED TARKVARAARENDUSPRAKTIKAD

Agiilne programmeerimine tähendab, et arendajad toodavad tihedalt uut koodi, muudavad seda ja peavad tähtaegadest kinni pidama. Sellistes tingimustes traditsionaalsed programmeerimispraktikad ei ole piisavad. (VersionOne, kuupäev puudub) Agiilsed protsessid peavad olema tagatud usaldusväärsete tarkvaraarenduspraktikatega. Olulisemad neist on: (Rasmusson, 2010)

- automaatne ühiktestimine *(ingl Automated Unit Tests)*;
- ümberstruktureerimine (*ingl Refactoring*);
- testidel põhinev arendus *(ingl Test-Driven Development)*;
- pidev integratsioon *(ingl Continuos integration)*.

Järgnevalt tutvustab autor iga praktikat ka lähemalt.

2.1 Automaatne ühiktestimine

Automaatsete ühiktestide käigus testitakse kindlaid funktsioone, klasse või üksuseid koodist (McFarlin, kuupäev puudub). Nad on mahult väga väiksed ning arendajad uuendavad neid igakord, kui teevad tootele uusi funktsionaalsusi juurde. Automaatsed ühiktestid on hindamatud, kuna neid saab käivitada igakord, kui tarkvarasse lisatakse arendusi. Tavaliselt agiilsetes arendustes on sadu, kui mitte tuhandeid ühikteste. Mõned positiivsed aspektid nende testide kasutusele võtmiseks: (Rasmusson, 2010)

- Nad annavad kohese tagasiside – kui teha koodi muudatusi ja test ei toimi, siis on kohe teada, et uus kood on vigane (ei pea ootama kolm nädalat toodanguni).
- Nad kärbivad väga märkimisväärselt manuaalse testimise kulusid – selle asemel, et kõiki pisiasju, mis toodangusse lähevad käsitsi üle kontrollida iga kord, kui mõni muudatus tuleb, saab keskenduda suuremate keerulisemate muudatuste testimisele käsitsi.
- Vähendavad vigade otsimise aega – automaattestid näitavad kohe ära, kus on viga ja enam ei pea käima mitmeid ridu koodi läbi selleks, et seda üles leida.
- Toodangusse saab panna arendusi kindlamalt – kui automaattestid saab tööle panna, siis on kindel, et kiirelt saab tagasisidet toodangu kohta. See ei ole küll sada protsenti

kindlus, kuid parem kui mitte midagi ja jätavad aega selleks, et keerulisemaid teemasid ise testida.

Võib tulla ette juhuseid, kui automaattestide kirjutamine osutub väga keeruliseks ja osadel juhtudel ei saagi neid kirjutada, kuid sellest ei maksa lasta ennast heidutada. Kui automaattestidega jääb osa koodist katmata, tuleb see osa manuaalselt üle testida.

Automaatsed ühiktestid on muutunud populaarseks eriti just viimasel ajal, nii et modernsematel keeltel on isegi raamistikud ja õpetused nende kohta. (Rasmusson, 2010)

2.2 Ümberstruktureerimine

Ümberstruktureerimine on olemasoleva koodi lihtsustamise ja selgemaks tegemise protsess ilma, et muudetakse üldist käitumist (VersionOne, kuupäev puudub). Agiilsetes meeskondades võib tulla ette olukordi, kus tähtaja lähenedes läheb väga kiireks ning mingit koodiosa on vaja mitu korda kopeerida ja kleepida. Sellega seoses kaotab kood oma kvaliteeti ning lõpuks ei saa enam keegi aru sellest. See tähendab, et kui tulevikus teha arendusi, peab muudatusi tegema mitmes erinevas kohas, mis võtab rohkem aega ning suurendab potensiaalsete vigade arvu. Lõpuks võib selguda, et uut funktsionaalsust ei saagi enne lisada, kui vana koodi ei ole ümber kirjutatud. (Rasmusson, 2010) See omakorda võib agiilsetes meeskondades tähendada juba seda, et kas sprint saadakse õigeks ajaks valmis või mitte (VersionOne, kuupäev puudub).

Siia juurde kasutatakse ka mõistet tehniline võlg *(ingl technical debt)*. Selles on kõik otseteed, häkid, kordused ja kõik muu, mis rikub head kodeerimistava. Aja jooksul peaks vähendama oma süsteemis tehnilist võlga. (Rasmusson, 2010)

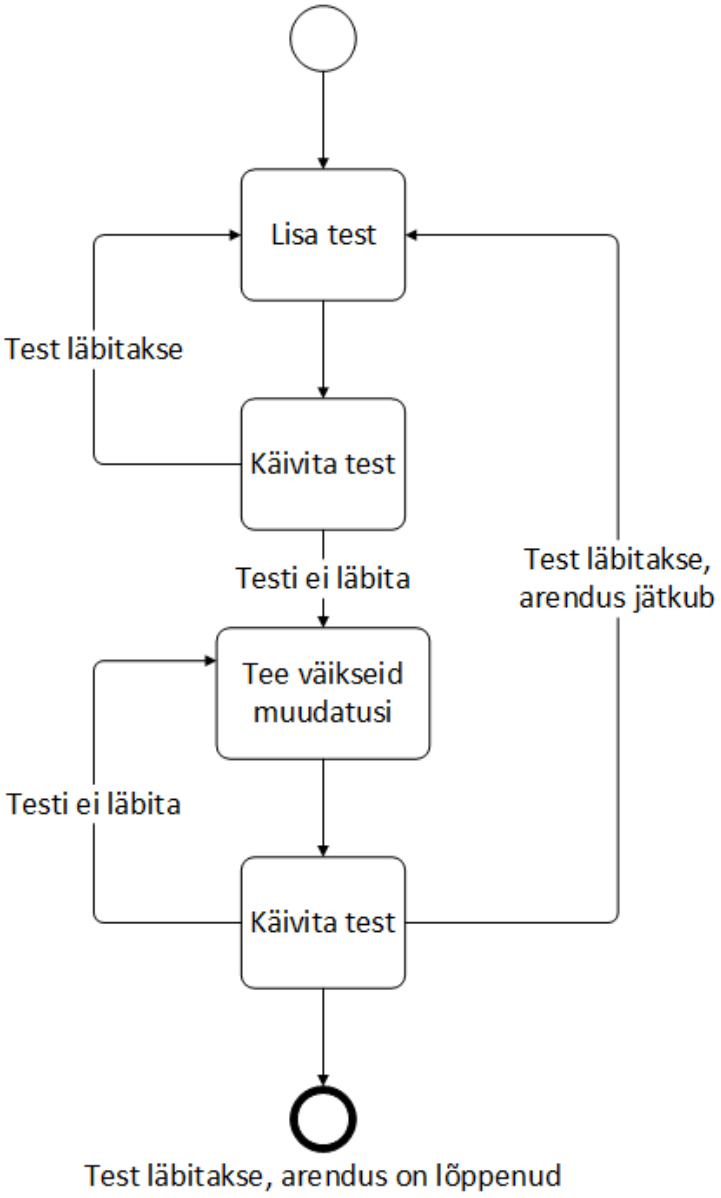
Kui koodi restruktureeritakse, siis ei lisata uut funktsionaalsust või parandata vigu vaid tehakse kood arusaadavamaks ja kergemini muudetavaks. Näiteks, kui muuta halvasti nimetatud meetodit või muutujat, selleks, et kood oleks paremini arusaadav, siis tegeldakse juba ümberstruktureerimisega. Kõige parem oleks seda teha järjepidevalt, et ei tekiks sellist olukorda, kus kogu koodi peaks korraga muutma hakkama. Kõige ideaalsem ümberstruktureerimine toimub nähtamatult. (Rasmusson, 2010)

2.3 Testjuhitud arendus

Testjuhitud arendus on tarkvaraarendustehnika, mis kasutab väga lühikesi arendustsükleid selleks, et järk-järgult disainida süsteemi (Rasmusson, 2010). Testjuhitud arenduses tehakse esimesena valmis automaatsed ühiktestid arendatavale osale enne kui selle kood on valmis kirjutatud (VersionOne, kuupäev puudub). Ehk peab ette kujutama, et uus arendatav funktsionaalsus on juba olemas (Rasmusson, 2010).

Sammud testjuhitud arendusel on järgmised (vt Joonis 2): (Rasmusson, 2010)

1. Enne, kui kirjutada koodi uue süsteemi jaoks peab kirjutama ühiktesti, milles on kirjas, mida uus kood peab tegema. Siin peab mõtlema eelkõige koodi disaini peale. Tavaliselt kukub see test läbi, kuna lisatavat funktsionaalsust ei ole veel.
2. Seejärel peab kirjutama täpselt niipalju koodi kui on vajalik selleks, et test läheks läbi.
3. Koodi peab ära puhastama kõigest, mis sinna võis tekkida selle aja jooksul, kui muudeti ta toimivaks ehk siis ümberstruktureerima.



Joonis 2. Testjuhitud arendus (Ambler, kuupäev puudub)

Testjuhitud arendus on spetsiifiline tegevus selleks, et olla kindel, et kood on ilma vigadeta ja läbinisti testitud (Ambler, kuupäev puudub).

2.4 Pidev integreerimine

Pidev integreerimine on selline arenduspraktika, kus arendajad peavad integreerima koodi jagatud hoidlasse mitu korda päevas. Iga koodi üles laadimine on tuvastatav automaatselt, nii et tiimid saavad probleemidega kiiremini tegeleda. Regulaarselt integreerides saab tuvastada vigu kiiremini. (ThoughtWorks, kuupäev puudub)

Agiilses arendustiimis näeb see protsess välja järgmine: (Rasmusson, 2010)

1. arendaja võtab hoidlast kõige viimase koodi versiooni;
2. arendaja teeb muudatusi koodi;
3. arendaja testib koodi;
4. arendaja kontrollib, kas keegi on teinud uuema versiooni koodist;
5. arendaja testib uuesti koodi;
6. arendaja integreerib koodi.

Oluline on kindlasti koodi integreerida nii tihti kui võimalik, näiteks kümne kuni viietest minuti tagant. (Rasmusson, 2010)

3 PAINDLIKU TARKVARAARENDUSMETOODIKA JUURUTAMINE

Käesolevas peatükis kirjeldatakse soovitusi, kuidas peaks agiilsele tarkvaraarendusmeetodikale üle minema. Lisaks antakse ülevaade, milline on olukord meeskonnas, kus peaks agiilset raamistikku Scrum'i kasutama hakkama. Tutvustatakse, kuidas ettevõtte teised tiimid praktiseerivad agiilset metoodikat. Kuna aeg-ajalt renditakse arendajaid ka teistesse tiimidesse, siis on kasulik, kui kõik oskavad sama metoodikat.

3.1 Üleminek paindlikule tarkvaraarendusmetoodikale

Paljud tarkvaraarendusettevõtted lähevad agiilsele arendusele üle, kuna agiilsetes tiimides arendatud tarkvara on parema kvaliteediga, vastab kasutaja vajadustele ning tootmine on madalamate kuludega. Paljud ettevõtted, mis on Scrum'ile üle läinud näevad kasu juba üsna kiirelt – nad suudavad tooteid tuua turule palju kiiremini ning see tõstab klientide rahulolu olulisel määral. Kuid üleminek Scrum'ile ja teistele agiilsetele metoodikatele on kindlasti raske. (Cohn, 2012) Protsessimuudatused ei ole kunagi kerged, eriti kui muudetakse seda, kuidas tiimiliikmed on harjunud koostööd tegema (Adzic, 2011).

Järgnevalt on refereeritud M.Cohn'i, kes on oma raamatus *„Succeeding with Agile"* toonud välja kaks varianti, kuidas Scrum'ile üle minna (Cohn, 2012).

Üks võimalus, kuidas juurutada paindlikku tarkvaraarendusmetoodikat, on ette valmistada pilootprojekt, õppida sellest ja siis juba kasutada terves ettevõttes ehk alustada aeglaselt. Nii saavad ettevõtte teised tiimid, õppida esimeste meeskondade vigadest ja teha asju kohe õigesti. Mõned selle viisi plussid:

- Maksab vähem – kuna tavaliselt ettevõttes ei ole inimesi, kes teaksid, kuidas Scrum'i raamistiku kohaselt arendada, siis on vaja väljast palgata Scrum'i meister, kellele on vaja maksta. Kui teha aeglasem üleminek, siis ei ole vaja väljastpoolt spetsialisti palgata vaid firmas olevad inimesed jõuavad selle ära õppida.
- Varajane edu on garanteeritud – tavaliselt valitakse pilootprojekti läbiviimise meeskond, kes suure tõenäosusega saavutab edu.

- Väiksem risk – kui kohe terve organisatsioon üle viia ja midagi läheb valesti, siis ilmselt meeskonna liikmed ei anna enam kergelt uut võimalust järgmiseks katsetuseks.
- Väiksem stress – organisatsioonides on niigi palju stressi ja keegi ei taha seda juurde. Kuid selline uudis, et terve ettevõte peab oma tööviisi muutma järgmisest päevast, ei mõju töötajatele motiveerivalt. Kui on olemas eeskujusid, kes on juba edukalt üle läinud, siis see tõstab töötajate enesekindlust.
- Saab teha ilma ümberorganiseerimata – kohe ei pea inimesi hakkama üle viima uutesse rollidesse või tiimidesse.

Teine võimalus on võtta Scrum kasutusele üleöö, mis võib osadele ettevõtetele paremini sobida. Näiteks sellised firmad, mis on agressiivsemad ja neil on peamiselt tulemustele orienteeritud kultuur. Tavaliselt on usk Scrum'i väga suur ning arvatakse, et milleks viia üks-kaks tiimi üle, kui on võimalus kõik korraga Scrum'i kasutama panna ja kohe hakkavad meeskonnad efektiivsemalt töötama. Mõned plussid selle viisi kasuks otsustamisel:

- Vähendab vastupanu – ikka leidub skeptikuid, kes arvavad, et see ei ole hea viis ja loodavad, et peale esimest projekti lõpetatakse Scrum'i kasutamine ära. Kui nad näevad, et terve ettevõte võtab selle kohe kasutusse, siis see tähendab seda, et sellessse usutakse ja kõhklemiseks ei jää ruumi.
- See vähendab probleeme, mis tekivad kui Scrum'i tiimid ja mitte Scrum'i tiimid töötavad koos. Kui samas ettevõttes töötavad meeskonnad Scrum'iga ja mõned mitte, siis see võib tekitada probleeme nende omavahelises koostöös.
- Üleminek toimub kiiremini – kuigi agiilsele arendusele üleminek ei toimu kunagi nii, et saab öelda, et nüüd on kõik „tehtud" ja alati saab teha parendusi, siis kiirelt üleminevad tiimid teevad ikkagi suurema osa tööst kiiremini ära.

Variant on kasutada neid mõlemaid viise korraga. See tähendab seda, et paar tiimi hakkab Scrum'i kasutama ja seejärel kõik teised meeskonnad tulevad järgi. Vajalik on edukaid tiime esimesena üle viia, kuna siis on olemas head näited, et Scrum töötab ja teised peavad sama kaugele jõudma oma arendustega. (Cohn, 2012)

3.2 Agiilne arendus teistes meeskondades

Vaadeldava ettevõtte teistes tiimides praktiseeritakse agiilset arendust igapäevaselt. Töö autor oli ühe nädala teises meeskonnas töövari ning jälgis sealset töökorraldust.

Vaadeldavas tiimis on arhitekt, analüütik, tiimijuht ja kolm arendajat. Meeskonnas on kolm korda nädalas (esmaspäev, kolmapäev ja reede) hommikuti kell 10.00 püstijalakoosolekud. Need kestavad tavaliselt 10-15 minutit ning nende käigus jutustavad tiimiliikmed, millega nad tegelesid eelmisel päeval, mida planeerivad ette võtta samal päeval ning kas nende töös on takistusi. Kui kellelgi on tekkinud töö käigus mingeid küsimusi teistele tiimiliikmetele, siis need saavad vastuse kiirelt koosoleku käigus. Kui on näha, et mingi probleem vajab pikemalt tegelemist, siis lepitakse kokku eraldi koosoleku aeg selleks.

Sprindid kestavad kaks nädalat ning enne uut sprinti toimub sprindi planeerimine. Selles tiimis tehakse seda nii, et tiimi siseselt võetakse ette toote kuhi ja hakatakse sealt töid sprinti võtma. Toote kuhja haldab tavaliselt tiimijuht ja analüütik. Analüütik lisab sinna ülesanded, mis on välja tulnud analüüsi käigus. Klient ise ei saa panna uusi ülesandeid otse arendajatele vaid kõik käib läbi analüütiku, kes haldab töödevoogu.

Tavaliselt on ühes sprindis ligikaudu 20 tööd. Aga see võib erineda sõltuvalt tööde suurusest. Hinnangud pannakse samuti paika sama koosoleku käigus. Suurematele töödele on juba eelnevalt hinnangud antud. Kui sprint on tiimi sees ära kinnitatud, siis saadetakse see kliendile kinnitamiseks. Kui klient on ära kinnitanud sprindi, siis võetakse sprint töösse järgmiseks kaheks nädalaks. Selle kahe nädala jooksul toimuvad analüütikul koosolekud kliendiga, kus arutatakse tuleviku arendusi. Poole sprindi pealt tehakse kliendile esitlus arendustest, mis on tehtud selleks ajaks. Kui arendajad saavad mõne arenduse valmis, siis analüütik testib neid ning kinnitab, kas on kõik tehtud nii nagu nõuetes kirjas. Sprint lõpeb sellega, et viimase arenduspäeva hommikul tehakse kliendile uuesti ettekanne arendustest, mis on ära tehtud. Selle lõpuks peab klient ütlema, kas ta võtab vastu selle arenduste paki või mitte. Lisaks annab klient tagasiside, kas on veel midagi muuta enne toodangusse panekut.

Õhtul pannakse terve sprint toodangusse. Sellest järgmisel päeval parandatakse kiirelt vigu (kui neid on) ja tehakse uus sprindi planeerimine.

3.3 Praegune olukord meeskonnas

Meeskonnas on nooremarendaja, vanemarendaja, tiimijuht ja analüütik. Meeskonnas on ka üks eripära, nimelt vanemarendaja ja tiimijuht ei istu samas kontoris, kus ülejäänud meeskond. See loob natuke teistsuguse olukorra, kui on ühes ruumis töötavatel tiimidel.

Nooremarendaja on siiani teinud pisemaid arendusi, mis vanemarendaja on talle andnud teha. Tavaliselt tema tehtud koodi vaatab vanemarendaja üle.

Vanemarendaja on suure kogemusega programmeerija ning oskab välja õpetada teisi. Annab tehnilistes küsimustes nõu ka analüütikule.

Tiimijuhi ülesandeks on projektide juhtimine, suhtlemine kliendiga ning teha kindlaks, et kõigil liikmetel oleks tööd. Samuti on ülesandeks lahendada töö käigus tekkinud probleeme.

Analüütik suhtleb kliendiga ning teeb selgeks, millised on kliendi vajadused ja soovid tulevasteks arendusteks. Paneb need kirja dokumenti, mis aitab arendajaid programmeerimisel.

Praegu tehakse kord nädalas koosolek kliendi ja kõikide tiimiliikmetega, kus vaadatakse üle, mis staatuses on tööd, mis on iteratsioonis. Seejärel võetakse sealt töid välja ja ka lisatakse juurde (näiteks jooksvalt tekkinud vigade parandamine). Samuti otsustakse, kas on vaja mingeid töid toodangusse panna järgmise nädala jooksul.

Tööde haldamiseks kasutatakse meeskonnas projektihaldustarkvara Jira. Sinna sisestatakse uusi töid ning neid saab suunata kindlale inimesele, kes selle tööga peaks tegelema hakkama. Iga projektis olev inimene saab ülesande juurde lisada ka aja, mis on tal selle tegemise peale kulunud. Kõik tööd, mis on sprindis, on näha Jira’s ühel tahvlil üldvaates. Seda tahvlit jälgitakse ka koosolekut tehes, kui vaadatakse üle tööde staatusi.

3.4 Rollide jagamine meeskonnas ja Scrum’i juurutamise plaan

Tavaliselt Scrum’i meeskondades saab tiimijuhist Scrum’i meister, toote omanik või tiimiliige, sõltuvalt sellest, millised on ta kogemused, oskused ja huvid. Kuna analüütikutel on head kommunikatsiooni oskused ja põhjalikud teadmised tootest, siis nad võivad samuti üle võtta tooteomaniku rolli. Kuid on meeskondi, kes endiselt arvavad, neil on ka Scrum’i tiimis analüütikut vaja, kuigi analüütiku roll kindlasti muutub. Scrum’i projektis saab eesmärgiks käimasoleva

projekti analüüs, enam ei pea analüütik olema analüüsiga mitu kuud ees. Analüütik peaks olema natuke ees oma tööga, kuid samas peab ta olema võimeline andma informatsiooni hetkel käimasolevas sprindis olevate tööde funktsionaalsuste kohta. Rohkem infot jagatakse suuliselt mitte mahukate dokumentide kaudu. Scrum'i tiimi analüütik aitab testimisega, vastab küsimustele arendatavate funktsionaalsuste kohta ning osaleb püstijalakoosolekutel. Natuke on muutunud ka programmeerijate roll – nad programmeerivad, testivad, analüüsivad, disainivad. Nad teevad kõike selleks, et sprint saaks valmis. Scrum'i tiimis nad ei saa ainult oodata, millal neile tööd antakse vaid nad peavad ise olema aktiivsemad ja aru saama tootele esitatud nõuetest. (Cohn, 2012)

Hetkel jäid meeskonnas kõikide ametinimetused samaks, kuid muutusid mõnevõrra ülesanded. Analüütik võttis üle Scrum'i meistri ülesanded ning tiimijuht võttis tooteomaniku rolli.

Selleks, et hakata kasutama Scrum'i raamistikku, on oluline koostada selle juurutamise plaan. Kui vaadata M.Cohn'i soovitusi Scrum'i kasutuselevõtuks, siis pigem saab kasutada sellisel juhul Scrum'ile üleminekuks esimest varianti, mis tähendab seda, et ettevõttes on juba edukaid meeskondi, keda saab võtta eeskujuks,. M.Cohn soovitab panna kirja kõik tegevused Scrum'i kasutuselevõtuks ja võtta seda kui toote kuhja – tegevused võtta järjest töösse.

Autor koostas Scrum'i juurutamise tegevused tuginedes kirjandusele ning jälgides ettevõttes olevaid teisi meeskondi ning nende töökorraldust. Kõigepealt on kirjas tegevused, mida tehakse meeskonna siseselt (vt Tabel 2).

Tabel 2. Tegevused, mida teeb meeskond

Tegevused	**Vastutaja**
1. Tutvustada meeskonnale uut töökorraldust	Tiimijuht
2. Anda hinnangud tootekuhjas olevatele töödele	Analüütik, vanemarendaja, Tiimijuht
3. Koostada konkreetse sprindi analüüsidokument	Analüütik
4. Jaotada omavahel tööd	Vanemarendaja

5. Peale sprinti teha retrospektiiv	Analüütik, tiimijuht

1. Tutvustada meeskonnale uut töökorraldust

Enamus meeskonnast küll teadis varem Scrum'ist, kuid kuna igas tiimis võib seda erinevalt praktiseerida, siis tutvustatakse põhimõtteid, mida just selles projektis hakatakse kasutama.

2. Anda hinnangud tootekuhjas olevatele töödele

Meeskond paneb igale tööle tundides hinnangu juurde. Selle järgi on kergem vaadata, milliseid töid sprinti võtta.

3. Koostada konkreetse sprindi analüüsidokument

Kui on kokku lepitud, millised tööd võetakse sprinti, siis koostatakse vastavate tööde täpsemad kirjeldused ning analüüsitakse läbi.

4. Jaotada tööd meeskonnas omavahel

Kuna meeskonnas on arendajaid vähe, siis vanemarendaja, kellel on suur kogemus, jaotab arendustööd ära enda ja nooremarendaja vahel.

5. Peale sprinti teha retrospektiiv

Peale esimest sprinti on oluline teha retrospektiiv meeskonnas, et aru saada, mida peaks parandama tulevikus ning mida teha teisiti. Retrospektiive tehakse ka kliendiga koos, kuid meeskond otsustas teha ilma kliendita esimene kord.

Järgmisena pani autor kirja tegevused, millesse peab ka klienti kaasama (vt Tabel 3).

Tabel 3. Tegevused, mida tehakse koos kliendiga

Tegevused	**Vastutaja**
1. Tutvustada kliendile uut töökorraldust	Tiimijuht
2. Esimeseks sprindiks valida välja pilootprojekt	Klient, tiimijuht

3. Moodustada kogu arenduse tootekuhi	Klient, tiimijuht
4. Planeerida sprint	Klient, Scrum’i meeskond
5. Kinnitada sprint	Klient
6. Leppida kokku sprindi eesmärk	Klient, tiimijuht
7. Otsustada sprindi kestvus	Klient, Scrum’i meeskond
8. Planeerida järgmist sprinti käimasoleva sprindi ajal	Klient, analüütik
9. Leppida kliendiga kokku pideva integreerimise reeglid	Klient, tiimijuht
10. Esitleda kliendile tulemust sprindi lõpus	Klient, analüütik

1. Tutvustada kliendile uut töökorraldust

Selleks, et klient saaks aru, mida tähendab paindlik tarkvaraarendusmetoodika peab talle selgitama seda täpsemalt ning klient peab mõistma oma rolli. Selles projektis tehti enne arenduste algust töötuba, kus tutvustati kliendile, mis selle tähendus on.

2. Esimeseks sprindiks valida välja pilootprojekt

Tavaliselt pole mõtet vanade tööde põhjal teha pilootprojekti vaid mõttekas näib võtta uus ülesanne, mida hakata tegema uues tarkvaraarendusmetoodikas. Kuna hetkel olemasoleval tarkvaral on tekkinud suur tehniline võlg, otsustati seda koodi mitte parandama hakata vaid teha täiesti uus kood. Neid arendusi hakatakse tegema kasutades paindlikku tarkvaraarendusmetoodikat. Samal ajal lisatakse uut funktsionaalsust.

3. Moodustada kogu arenduse tootekuhi

Enne sprindi alustamist, tuleb teha kõikide arenduste tootekuhi. See aitab mõista, milliseid töid on üldse vaja teha. Käesolevas projektis moodustas selle klient.

4. Planeerida sprint

Koos kliendiga vaadata üle toote kuhja tööd ning tõsta sprinti need, mis on kriitilisemad ja vajalik esimese prioriteedina ära teha.

5. **Kinnitada sprint**

Kui tööd on sprinti tõstetud ja neile on antud hinnangud, siis on oluline see ära kinnitada kliendi poolt.

6. **Leppida kokku sprindi eesmärk**

Kõikidele sprintidele tuleb määrata oma eesmärk. Pilootprojekti eesmärgiks saab sprindi lõpetamine, nii et kõik tööd saavad tehtud.

7. **Otsustada sprindi kestvus**

Sprindi kestvuseks otsustati kaks nädalat, kuna seda praktiseeritakse kõige rohkem ning tundub kõige optimaalsem sprindi kestvusaeg.

8. **Planeerida järgmist sprinti käimasoleva sprindi ajal**

Enne sprindi lõppemiston oluline ettevalmistada juba järgmist sprinti, kuna see on vaja koheselt töösse võtta peale seda, kui käimasolev sprint saab läbi.

9. **Leppida kliendiga kokku pideva integreerimise reeglid**

Kliendi sooviks oli võtta kasutusele pidev integreerimine. Sellega seoses toimusid sprindiga paralleelselt koosolekud, et saavutada kokkulepped, kuidas seda tegema hakatakse.

10. **Esitleda kliendile tulemust sprindi lõpus**

Kui sprint on lõppenud, siis on väga oluline kliendile esitleda selle sprindi tulemust. Nii otsustati praktiseerida ka antud projektis.

Lisaks koostas autor nimekirja vahenditest ja võtetest, mida on plaanis kasutada pilootprojekti sprindis (vt Tabel 4).

Tabel 4. Nimekiri vahenditest ja võtetest

Tegevused	**Vastutaja**
1. Teha uue projekti jaoks eraldi Jira tööde tahvel	Analüütik, tiimijuht

2. Kasutada tiimi toas Scrum’i tahvlit	Analüütik, tiimijuht
3. Hakata tegema igapäevaseid püstijalakoosolekuid	Analüütik
4. Kasutada ühiktestimist	Vanemarendaja

1. Teha uue projekti jaoks eraldi Jira tööde tahvel

Kuna pilootprojekti tulevad tööd, mis on teistest ülesannetest täiesti eraldi seisnevad, siis on parem kasutada eraldi tahvlit nende tööde jaoks. Sel juhul on fookus selgelt paigas, mis just selle iteratsiooni raames toimub.

2. Kasutada tiimi toas Scrum’i tahvlit

Teha Scrum’i tahvel meeskonna kontoriruumi. Nii on parem jälgida, mis tööd kellelgi on ja mis tööd on lõpetatud. Meeskond otsustas, et tööde jälgimise tabelisse tuleb neli veergu: „Alustamata“, „Töös“, „Ootab ülevaatamist“ ja „Tehtud“.

3. Hakata tegema igapäevaseid püstijalakoosolekuid

Scrum näeb ette igapäevaseid püstijalakoosolekuid, mis kestavad kuni 15 minutit. Kuna antud meeskonnas asub üks tiimiliige teistest eemal, siis hakatakse neid koosolekuid pidama Skype’i vahendisel.

4. Kasutada automaatset ühiktestimist

Paindlikku tarkvaraarendusmetoodikasse kuulub samuti automaatne ühiktestimine ning meeskond otsustas seda kasutama hakata kohe esimesest sprindist peale. Nii ei pea pärast kulutama sellele aega, et automaatteste kirjutada.

4 TULEMUSED

Vaadeldav tarkvaraarendusmeeskond tegi ühe sprindi, järgides koostatud Scrum'i juurutamise plaani. Esimeseks iteratsiooni eesmärgiks sai: lõpetada sprint, nii et tööd, mis olid sprinti planeeritud, saaksid tehtud. Pilootprojektiks sai meeskonna poolt pikka aega arendatud veebirakenduse ümberkirjutamine, kuna tehniline võlg eelmisel koodil oli niivõrd suureks paisunud, et vana polnud enam mõtet parandama hakata. Järgnevas peatükis kirjeldatakse toimunud sprinti, analüüsitakse õnnestumisi ja ebaõnnestumisi ning tehakse ettepanekuid järgmise sprindi läbiviimiseks.

4.1 Pilootprojekti sprindi kirjeldus

Sprint algas kliendi poolt välja toodud ülesannete prioriseerimisega. Klient kaardistas kasutuslood, mida tavaliselt veebirakenduses tehakse kõige rohkem. Peale seda tegi arendusmeeskond analüüsi ning pakkus välja omalt poolt lahendused, kuidas arendada kliendi poolt kirjeldatud funktsionaalsusi, mis olid kõige olulisemad. Tehti kaks sprindi planeerimise koosolekut, mis toimusid Skype'i vahendusel. Kaks koosolekut oli seetõttu, kuna peale esimest koosolekut muutis klient oma esialgseid otsuseid. Pärast seda oli vaja uuesti hinnangud anda. Üks töö oli selline, kus arutelud läksid pikemaks ning seetõttu otsustati see iteratsioonist välja tõsta, kuna sprindi planeerimine läks pikemaks kui algselt ettenähtud. Kokku läks iteratsiooni planeerimise peale umbes neli tundi.

Püstijalakoosolekud toimusid iga päev ning kestsid kuni 15 minutit. Kuigi koosolekud toimusid Skype's, siis sellegipoolest püüti kinni pidada püsti seismise tavast. Jutud ei läinud sellest hoolimata liiga pikaks ning need koosolekud olid konstruktiivsed. Esimesel püstijalakoosolekul jaotas vanemarendaja tööd enda ja nooremarendaja vahel ära. Peamiselt rääkisid püstijalakoosolekutel arendajad, millega nad tegelesid kuid ka analüütik andis teada, kuidas järgmise sprindi planeerimine läheb. Samuti sai vanemarendaja kohe nooremarendajale päevaks tööd ette anda. Lisaks jälgis analüütik samal ajal Jira tahvlit, kuna arendajad ei olnud selle uuendamisega harjunud, siis oli vaja neile aeg-ajalt meelde tuletada, et nad annaksid töödele õiged staatused.

Antud projekti jaoks tehti eraldi Jira tahvel. Niimoodi oli näha kõik ülesanded, millega tegeldakse ja millele keskendutakse. Selleks, et eraldi tahvlit teha oli vaja Jira administraatori abi, kes nõustas,

milline lahendus oleks sellele projektile kõige õigem valik. Otsustati teha teistsugune tahvel, kui siiani oli kasutatud. Nimelt hakati kasutama lugusid (*epic),* kuna siis saab igale tööle määratleda suurema teema, kuhu alla ta kuulub ning on parem laiemat pilti näha. Kuna Scrum'is tehakse ülesanded väiksemateks tükkideks, siis on oluline näha, millise suurema ülesande alla kuulub väiksem ülesanne, mis on võetud sprinti. Kui see oli otsustatud, siis pidi sisestama kõik sprinti võetud tööd koos hinnangutega Jira'sse. Tööde jaoks tehti viis staatust, mis olid sarnased Scrum'i tahvlile – „Vaja teha" *(To Do)*, „Töös" *(In Progress),* „Testimisel" *(Testing),* „Testitud" *(Tested)* ja „Tehtud" (*Done).* „Vaja teha" töö lahtrisse läksid kõigepealt kõik tööd, mis olid hinnangu külge saanud ja mis pidid sprinti minema. Need tööd olid seal prioriteetsuse järjekorras. Kõige olulisem töö oli kõige esimene ning vähemoluline oli viimane. Sinna lisati ka sellised tööd nagu projektijuhtimine või analüüsidokumendi koostamine. Tänu sellele olid kõigil sprinti valitud ülesanded kogu aeg silme ees. Kui prioriteetsemad ülesanded võeti töösse, siis need said staatuseks „Töös". Peale seda, kui arendaja lõpetas selle töö ja hakkas testima, siis sai see staatuseks „Testimisel". Kui töö oli testitud arendaja poolt, siis sai ta staatuseks „Testitud", mis tähendas ühtlasi seda, et see ülesanne on valmis kliendile üle andmiseks. Kui sprint läheb toodangusse, siis saavad kõik tööd külge tunnuse „Tehtud".

Meeskonna tuppa joonistati ka Scrum'i tahvel, kuid peab tunnistama, et seal ei olnud alati ajakohast infot. Kuna püstijalakoosolekud toimusid Skype's, siis ei jälgitud seda tahvlit nende koosolekute ajal.

Üsna pea pärast sprindi alustamist tekkisid esimesed kitsaskohad, mis olid seotud eelkõige tiimi koosseisuga. Nimelt nooremarendaja vajas juhendamist, kuid vanemarendajal ei olnud alati selleks aega. Seetõttu tekkis sprindi alguses väike tööde seisak, kuid õnneks see lahenes umbes paari päeva pärast ära. Ülejäänud sprindi jooksul enam selliseid seisakuid töös ei esinenud.

Poole sprindi peal tehti sprindi ülevaatuse koosolek, kus selgus, et ollakse enamvähem ajakavas, kuid võib juhtuda, et kõik tööd ei saa tehtud, mis sprinti olid võetud. Sel hetkel sai selgeks, et järgmist sprinti ei tule kohe, kuna klient vajab natuke rohkem aega, et enda poolt järgmised funktsionaalsused ära kaardistada ning läbi mõelda edasised arendused. See tähendas, et analüütik ei saanud enam järgmiseks sprindiks töid ettevalmistada.

Kui sprint lõppes, siis sama päeva püstijalakoosolekul sai selgeks, kui palju funktsionaalsusi saab kliendile näidata sprindi esitluse koosolekul. Oli selge, et üks töö jäi sprindi raames tegemata, kuid ülejäänud ülesanded said siiski tehtud. Põhjus, miks üks ülesanne jäi tegemata, oli selles, et selle

funktsionaalsus osutus suuremaks ülesandeks kui alguses oli hinnatud ning seda ei osatud seda ette näha.

Sprindi esitluskoosolekul osalesid kõik meeskonnaliikmed ning kliendi poolt samad inimesed, kes osalesid analüüsidokumendi kokku panemisel. Kuigi ühelgi arendataval funktsionaalsusel ei olnud korralikku kasutajaliidest *(ingl user interface)* vaid olid pigem programmikoodid, siis sellegipoolest sai neid kliendile näidata ja selgitada, mida täpsemalt iga töö raames tehti. Esitluskoosolekul tuli välja, et üks funktsionaalsus ei olnud nii arendatud nagu klient oli seda mõelnud. Kuid vaadates kokku lepitud ja kliendi poolt kinnitatud analüüsidokumenti, siis kõik oli selle raames tehtud õigesti ning nüüd tuleb anda uus hinnang ja teha järgmine arendus selle kohta. Muud ülesanded olid lahendatud ootuspäraselt.

Peale sprindi esitlust toimus retrospektiiv, mille käigus selgus, et tiimiliikmed olid väga rahul sellega, kui on kindlad ülesanded paigas ning kaks nädalat tegeldakse ainult nende teemadega. Samuti meeldis see, et peeti püstijalakoosolekud ning otsustati, et neid hakatakse kindlasti edaspidi pidama teiste projektide raames. Probleemina toodi välja suhtlus kliendiga. Nimelt klient ei kasuta alati Skype'i, kuid meeskond eelistab just seda vahendit ning info liikus liiga aeglaselt.

4.2 Õppetunnid järgmistesse sprintidesse

Järgmine sprint planeeritakse kahenädalane. Meeskonna liikmete arv jääb tõenäoliselt samaks. Kuna esimesel sprindil tekkis probleeme sellega, et nooremarendaja pidi ootama vanemarendaja järgi, siis järgmisel sprindil tuleb eelnevalt selgeks teha, kas vanemarendajal on piisavalt aega, et juhendada, nii et töö ei jääks seisma.

Väga oluline on anda töödele õiged hinnangud ning see oli kindlasti pilootsprindis kitsaskoht, et ühe töö hinnang ei olnud õige ning seetõttu ei saadud seda valmis. Kindlasti peaks tulevikus uurima meetodeid, kuidas saaks hinnanguid anda täpsemini.

Püstijalakoosolekud võetakse kindlasti üle ning praktiseeritakse edasi neid, kuna tänu nendele liigub info tiimis palju kiiremini. Ka meeskonnaliikmed tunnetasid seda kõige enam, et kommunikatsioon on väga oluline.

Jira'sse tuleb lisada veerg „Koodi ülevaatamine" *(Code Review)*, kuna nooremarendaja kood oli vaja enne testimisse panekut üle vaadata vanemarendajal, siis praegu ei olnud selliste staatuste

jaoks kohta Jira’s. Paljud tööd olid testimise staatuses, kuid tegelikult vanemarendaja vaatas koodi üle. Uues veerus on kohe näha, millised tööd on valmis, aga läbivad alles koodi ülevaatamise etappi.

Otstarbekaks ei osutunud tavalise Scrum’i tahvli kasutamine kuna meeskonnaliikmed ei asunud ühes toas. Lisaks tundus see Jira tahvli kopeerimisena. Ainus motivatsioon seisnes selles, et kui keegi tuppa astus, siis oli näha kohe, et kes millega tegeleb ja mis ülesanded on praegu töös. Pigem peaks tulevikus kirjutama sprindi eesmärgi ning sprindi kestvuse graafiku tahvlile, nii et see oleks meeskonnaliikmetel silme ees.

KOKKUVÕTE

Käesoleva bakalaureusetöö eesmärk oli tutvustada paindlikku tarkvararendusmetoodikat Scrum ning hakata seda kasutama tiimis, mis ei ole siiani töötanud agiilselt. Selleks, et eesmärki täita oli autor nädala teises meeskonnas, kus jälgis, kuidas seal kasutatakse. Sealt sai autor palju ideid ja mõtteid, kuidas oma meeskonnas juurutada Scrum'i. Sellest sündis tegevuste nimekiri, mis on vaja teha selleks, et Scrum'i kasutusele võtta. Kui tegevused olid kirja pandud, tehti kahenädalane sprint pilootprojektiga selleks, et neid tegevusi juurutada. Samal ajal tegi autor märkmeid ja jälgis, kuidas see edeneb.

Kõige olulisem oligi sellest pilootprojektist võimalikult palju õppida ning saada aru, mis järgmine kord teisiti teha. Tähtsamad järeldused sprindist olid järgmised:

- Oluline on teha sprindi planeerimine, kuna siis on kõikidel selge, mis on selle sprindi eesmärk ja mis ülesanded võetakse töösse.
- Oluline on anda töödele hinnangud, see annab selgust nii meeskonnasiseselt kui ka kliendile, kui kaua mingi töö peale võib minna.
- Alguses on keeruline hinnanguid anda, kuna keegi pole varem mõõtnud, kui kaua mingi ülesande peale kulub. Seetõttu ongi oluline alustada hinnangute andmist, sest tänu sellele saab paremini selgeks, kui kaua ülesanded aega võtavad ning tulevikus saab samalaadseid ülesandeid kiirelt ära hinnata.
- Püstijalakoosolekud on kõige kasulikumad koosolekud meeskonnas. Vähemalt nii see selgus selles meeskonnas, kus Scrum'i juurutati.
- Poole sprindi pealt on vajalik ülevaadata, kas on reaalne lõpetada kõiki sprindis olevaid töid või midagi jääb pooleli.
- Sprindi esitluskoosolekud on väga vajalikud, kuna need toimuvad reaalelus mitte Skype's ning need koosolekud on alati konstruktiivsemad, kus inimesed istuvad ühe laua taga.
- Scrum'i tahvel ei osutunud väga oluliseks, selgus, et ka teistes tiimides seda väga ei kasutata. Pigem vaadatakse tööde staatusi Jira's.
- Oluline on anda ausat tagasisidet meeskonnaliikmetele selleks, et vigadest õppida ning nendest aru saada ja mõista. Keegi ei tohiks võtta midagi isiklikult või solvuda.

Kokkuvõttes võib öelda, et bakalaureusetöö raames läbiviidud sprint osutus selle meeskonna jaoks väga oluliseks, kuna tänu sellele praktiseeriti Scrum'i ning saadi rohkem aimu, mida selle

järgimine endast kujutab. Kindlasti tehakse edasi järgmisi ümberstruktureerimisega seotud ülesandeid Scrum'is. Tarkvaraarendusmeeskond sõltub ka kliendist ning tema otsustest, millist metoodikat kasutatakse. Hetkel on klient olnud sedameelt, et agiilne arendusmetoodika on hea viis arenduste tegemiseks, kuid veel ei ole osatud seda sammu teha, kuidas kõiki arendusi viia sellele metoodikale üle.

Järgmiseks sammuks võiks olla uue sprindi katsetamine, kus on arvestatud eelmises sprindis tehtud vigadega. Kuna inimestel on vaja uue töökorraldusega harjumiseks aega, siis ilmselt tuleb ka järgmine sprint pigem katsetus.

SUMMARY

Title: Applying Agile Software Development Methodology. The Case of Company X

This Bachelor's Thesis focused on a problem where work process had to change in one development team. It should be more transparent, effective and information should move faster. Once the team was smaller and client didn't make much job orders, so there was no need to use some development methodology. As there was coming more jobs and the team was growing it was clear that is needed to use some system. The team chose Scrum because in this company many teams are using it already and it has proved itself. The author of this theses is also involved with the development team.

The purpose of this Bachelor's Thesis was to introduce agile development methodology in Scrum framework and start to use it in a team which hasn't worked like agile team before.

For getting the purpose author did the next activities:

1. Learned about Scrum from books
2. Found out how other teams are using Scrum
3. Made proposals how to start using Scrum in a team
4. Made pilotproject where software was developed after Scrum
5. Made analysis of the first sprint and gave suggestions for the new sprint

After author made all those activities the most important conclusions of the first pilotproject were:

- It is important to make sprint planning, then everybody understands what is the purpose of this sprint and what tasks will be taken into work.
- It is important to evaluate every task because then team and client will know how long the job will take.
- At the beginning it is difficult to give any evaluations because nobody hasn't measured how long some task will take. So it is important to start giving evaluations and then it is clear how long tasks will take time and in the future it is easier to evaluate them.

- Stand-up meeting were the most useful meetings in the team. At least in this team where Scrum was applyed.
- In the half sprint it is necessary to look is it realistic to end all jobs which are in a sprint or something will not be done.
- Sprint demo meetings are valuable because sometimes it is good that whole team is gathering around the table and sees each other.
- Scrum board didn't become very useful. It turned out that also other teams are not using it very much. Rather people are looking the statuses in Jira.
- Important is to give honest feedback to the team members for learning from the mistakes and understanding them. Nobody shouldn't take anything personally or be offended.

In conclusion it was seen that this first sprint became very important for this team because it gave them a chance to understand more about Scrum and they saw how projects in Scrum are developed. This certainly encouraged them to practise Scrum also in next sprints. When next sprint is coming then is taken into account the mistakes which were done in the previous sprint. Probably this will be also more of a trial because people need time for getting used to the new system.

KASUTATUD KIRJANDUS

Adzic, G. (2011). *Specification by Example.* New York: Manning Publications Co. .

Ambler, S. W. (kuupäev puudub). *Agile Data*. Allikas: Introduction to Test Driven Development (TDD): http://www.agiledata.org/essays/tdd.html

Cohn, M. (2006). *Agile Estimating and Planning.* Upper Saddle River: Pearson Education, Inc.

Cohn, M. (2012). *Succeeding with Agile.* Boston: Pearson Education, Inc.

Derby, E., & Diana, L. (2011). *Agile Retrospectives.* Dallas: The Pragmatic Bookshelf.

e-teatmik. (kuupäev puudub). Allikas: e-teatmik: http://www.vallaste.ee/

Goldstein, I. (2014). *Scrum Shortcuts without Cutting Corners.* New Jersey: Pearson Education, Inc.

Heusser, M., & Markus, G. (2015). *Save Our Scrum.* Lean Publishing.

James, M. (kuupäev puudub). *DZone*. Allikas: The Ultimate Scrum Reference Card: https://dzone.com/refcardz/scrum?utm_medium=feed&utm_source=feedpress.me&utm_campaign=Feed:%20dzone%2Fpublications&ref=dzone

Kniberg, H. (2007). *Scrum and XP from the Trenches.* C4Media, Inc.

Kniberg, H., & Mattias, S. (2010). *Kanban and Scrum - making the most of both.* C4Media, Inc.

Manifesto for Agile Software Development. (2001). Allikas: Manifesto for Agile Software Development: http://agilemanifesto.org/

McFarlin, T. (kuupäev puudub). *Code Tutorials*. Allikas: The Beginner's Guide to Unit Testing: What Is Unit Testing?: http://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728

Mittal, N. (7. August 2013. a.). *Scrum Alliance*. Allikas: The Burn-Down Chart: An Effective Planning and Tracking Tool:

https://www.scrumalliance.org/community/articles/2013/august/burn-down-chart-%E2%80%93-an-effective-planning-and-tracki

Mountain Goat Software. (kuupäev puudub). Allikas: Scrum: http://www.mountaingoatsoftware.com/agile/scrum

Rasmusson, J. (2010). *The Agile Samurai.* Pragmatic Bookshelf.

Scrum Alliance. (kuupäev puudub). Allikas: Learn About Scrum: https://www.scrumalliance.org/why-scrum

Siilivask, J. (2014). *Agiilsete metoodikate rakendamine IT töökorraldussüsteemi loomisel SMIT näitel.* Tallinn.

Sutherland, J. (2014). *Scrum - The Art of Doing Twice the Work in Half the Time.* New York: Crown Business.

ThoughtWorks. (kuupäev puudub). Allikas: Coninuous Integration: https://www.thoughtworks.com/continuous-integration

VersionOne. (kuupäev puudub). Allikas: Refactoring: https://www.versionone.com/agile-101/agile-software-programming-best-practices/refactoring/