TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT

Vladislav Šikirjavõi 134659IAPB

SCRUMBANI RAKENDAMINE VÄIKSE TARKVARAARENDUSETTEVÕTTE NÄITEL

bakalaureusetöö

Juhendaja: Deniss Kumlander

PhD (Tehnikateaduste doktor)

TTÜ Vanemteadur

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Vladislav Šikirjavõi

20.05.2016

Annotatsioon

Töö eesmärk on analüüsida ja võrrelda IT-Projekt OÜ arendamisprotsessi, mis on hetkel ülesehitatud agiilse praktika alusel ning selle alusel välja pakkuda ning juurutada uue praktika, mis paremini sobiks ja lahendaks olemasolevad probleemid võimaldadades sujuvama arendamisprotsessi. Töös rakendati kaks tuntud arendusmetoodikat (Kanban, Scrumban).

Töös käsitletavad probleemid on, et kas teatud agiilse tarkvaraarenduse metoodika rakendamine on arendusprotsessi jaoks vajalik ning kas selle kasutamine on kasulik meeskonnaliikmete arvates.

Töö üheks tulemuseks on väike tarkvaraarendusettevõtte arenemise tegurite määratlus ja ülevaade nende mõjust arengule.

Töö põhitulemuseks on väike tarkvaraarendusettevõtte arengu analüüs vastavalt bakalaureusetöö peatükis (vt. 4.4.5 Kanban vs. Scrumban) väljatoodud teguritele.

Kummagi agiilse metoodika rakendamise analüüsimisel tulid välja parenduskohad, mille kasutamisel saaks arendusportsessi efektiivsemaks muuta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 6 peatükki, 4 joonist, 10 tabelit.

Abstract

Implementation of Scrumban in a small software development company

The aim of this thesis is to analyze and compare the software development process of IT-Projekt OÜ, which transitions to the agile development methodology, and to introduce new agile development methodology to fix problems.

The thesis is divided into five chapters. The first chapter is introduction. The second chapter introduces small software development company called IT-Projekt OÜ, and describes the team and the project. In the third chapter author brings out The Agile Manifesto main plan and twelve principles it is based on. The fourth chapter introduces three agile software development methodologies, brings out both advantages and disadvantages, and compares them. Finally, in last chapter author analyses Kanban and Scrumban separately, and proposes improvements that should make the software development process more effective.

The main problems addressed in the thesis is to finding out if applied agile practice is necessary for software development process and if it's useful for team members.

One of the results of the thesis is to define the small software development company development factors and provide an overview of their impact to the company development.

The main result of the thesis is the analysis of a small software development company according to the 4.4.5 chapter factors.

Analysing and comparing agile methodology development processes, the author found out many things that could be done to improve IT-Projekt OÜ software development.

The thesis is in Estonian and contains 54 pages of text, 6 chapters, 4 figures, 10 tables.

Lühendite ja mõistete sõnastik

WIP	*Work in progress*, töös olevad ülesanded
skoop	*Scope*, nõutav töö, mis koosneb eri funktsionaalsustest vajalikud projekti käigus
Responsive disain	*Responsive design*, veebilehe kuvad mitmes eri suuruses, kus vajalik nähtavus püsib
Back-end	*Back-end*, serveripoolne, kuidas veebileht töötab
Front-end	*Front-end*, kasutaja jaoks nähtav osa (disain, HTML, CSS)
Multitegumtöö	*Multitasking,* teha mitu asja korraga
vs	*Versus,* vastu

Sisukord

1 Sissejuhatus 10
2 Väikettevõtte kirjeldus 12
2.1 IT-Projekt OÜ 12
2.1.1 Meeskond 12
2.1.2 Projekt 13
3 Agiilsest tarkvaraarendust üldiselt 14
3.1 Agiilse tarkvaraarenduse ajalugu 14
3.2 Agiilse tarkvaraarenduse manifest 15
3.3 Agiilse tarkvaraarenduse manifesti põhimõtted 15
4 Agiilse tarkvaraarenduse metoodikad 17
4.1 Scrum 17
4.2 Kanban 21
4.3 Scrumban 23
4.4 Võrdlus 28
4.4.1 Scrumi eelised ja puudused 28
4.4.2 Kanbani eelised ja puudused 30
4.4.3 Scrumbani eelised ja puudused 31
4.4.4 Scrum vs. Scrumban 32
4.4.5 Kanban vs. Scrumban 33
5 Analüüs 34
5.1 Kanbani rakendamine 34
5.1.1 Planeerimine 34
5.1.2 Task 35
5.1.3 „Pull“ põhimõte 36
5.1.4 Retrospektiiv 37
5.1.5 Töövoo visualiseerimine 37
5.1.6 Meeskond ja rollid 38
5.2 Scrumbani rakendamine 39
5.2.1 Planeerimine 39

5.2.2 Task 40
5.2.3 „Pull“ põhimõte 41
5.2.4 Retrospektiiv 42
5.2.5 Töövoo visualiseerimine 43
5.2.6 Meeskond ja rollid 44
5.3 Järeldused 44
5.3.1 Kanbani analüüsi kokkuvõte 44
5.3.2 Scrumbani analüüsi kokkuvõte 47
6. Kokkuvõte 51
Kasutatud kirjandus 52
Lisa 1 – Kanban’i töövoo visualiseerimine 54

Jooniste loetelu

Joonis 1 Graafiline kujutis Scrum’i arendusmetoodikast 21
Joonis 2 Graafiline kujutis Kanban tahvlist 23
Joonis 3 Graafiline kujutis Scrumban'i tahvlist 27
Joonis 4 Kanban'i töövoo visualiseerimine 54

Tabelite loetelu

Tabel 1 Scrum'i eelised 28
Tabel 2 Scrum'i puudused 29
Tabel 3 Kanban'i eelised 30
Tabel 4 Kanban'i puudused 30
Tabel 5 Scrumban’i eelised 31
Tabel 6 Scrumban’i puudused 31
Tabel 7 Scrum vs. Scrumban 32
Tabel 8 Kanban vs. Scrumban 33
Tabel 9 Kanban'i analüüsi kokkuvõtte tabel 46
Tabel 10 Scrumban'i analüüsi kokkuvõtte tabel 49

1 Sissejuhatus

Tänapäeva pidevalt arenevas infotehnoloogia maailmas on suur tähtsus tarkvaraarendamisel. Rahulolematus raskekoeliste tarkvaratehnika lähenemisviisidega põhjustas seda, et paljud tarkvaraarendajad tegid 1990ndal aastal ettepaneku uue agiilsete meetodite jaoks. Tänu sellele sai arendusmeeskond keskenduda rohkem tarkvarale kui disainile ja dokumentatsioonile. Agiilsed meetodid sobivad kõige enam tarkvaraarendusele, kus nõuded muutuvad pidevalt arendusprotsessi käigus. Seda hakkasid kasutama mitmed erinevad tarkvaraarendajad, mille tõttu aja jooksul kujunesid välja erinevad agiilse tarkvaraarenduse meetodid, millel kõigil on omad omadused. [4]

Käesolevas bakalaureusetöös keskendub autor agiilsete metoodikate rakendamisele tarkvaraarenduses. Töös toimub üleminek Kanban'ilt Scurmban'ile ning analüüsitakse nende mõju väikse tarkvaraarendusettevõtte näitel.

Töö on suunatud peamiselt väike tarkvaraarendusettevõtetele, kes oleks huvitatud tarkvaraarendust efektiivselt alustama, samuti firmadele, kes tahavad parandada oma töö tegemist, ning ka inimestele, kes on huvitatud antud teemast.

Autori eesmärgiks on analüüsida ja võrrelda IT-Projekt OÜ arendamisprotsessi, mis on hetkel ülesehitatud agiilse praktika alusel ning selle alusel välja pakkuda ning juurutada uue praktika, mis paremini sobiks ja lahendaks olemasolevad probleemid võimaldades sujuvama arendamisprotsessi ning parandaks seejuures töötajate rahulolu ning meeskonna koostöövõimet. Samuti saada teada, kas ja miks on kasutatav agiilne praktika kasulik ja vajalik ettevõttele ning pakkuda välja parendusi ja muudatusi, kuidas muuta IT-Projekt OÜ tarkvaraarendusprotsessi kindla agiilse praktika puhul paremaks. [3]

Esmalt tutvus autor teooriaga ja tõi välja agiilse tarkvaraarenduse manifesti põhipunktid ning kirjeldas ära selle põhimõtted. Siis kirjeldas autor ära kolm erinevat agiilse tarkvaraarenduse metoodikat (Scrum, Kanban, Scrumban) ning samuti tõi välja nende eelised ja puudused, mis nende rakendamisega tekivad. Seejärel võrdles autor neid arendusmetoodikaid omavahel kasutades erinevaid võrdlusmomente. Kaks nendest

arendusmetoodikast rakendati ja analüüsiti töös käsitletud projekti käigus. Nende metoodikate analüüsist püüab autor teada saada, kui kasulikud ja vajalikud on need väike tarkvaraettevõtte jaoks. Seejärel pakub autor välja parendusi, et muuta rakendatud agiilsed praktikad ettevõtte ning meeskonna jaoks efektiivsemaks.

Bakalaureseutöö koosneb viiest osast. Esimene osa koosneb sissejuhatusest, kus täpsustatakse töö taust, probleem ja ülesande püstitust ning kirjeldatakse seejärel, kuidas jõutakse eesmärkide täitmiseni.

Töö teises osas annab autor ülevaate väikettevõttest IT-Projekt OÜ, kus kirjeldatakse selle meeskonda ja töös käsitletud projekti.

Töö kolmandas osas toob autor välja agiilse tarkvaraarenduse manifesti põhipunktid ning kirjeldab ära põhimõtted.

Neljandas osas kirjeldab autor agiilse tarkvaraarenduse metoodikaid Scrum, Kanban ja Scrumban ning toob välja nende eelised ja puudused. Samuti võrdleb autor omavahel Scrum'i ja Scrumban'i ning Kanban'i ja Scrumban'i tuues välja erinevad võrdlusmomendid.

Viiendas osas analüüsib autor Kanban'i ja Scrumban'i kasutamist väikeettevõttes selgitades ära iga võrdlusmomendi põhimõtted. Selle analüüsi alusel teeb autor kokkuvõtte, kus tuuakse välja üldistusi ja järeldusi ning pakutakse parenduskohti.

Bakalaureusetöö kirjutati Tallinna Tehnikaülikoolis, infotehnoloogia teaduskonnas, informaatikainstituudis, tarkvaratehnika õppetoolis, 2016.aasta kevadel.

2 Väikettevõtte kirjeldus

Alljärgnevas peatükis kirjeldab autor ühte ettevõtet, milleks on IT-Projekt OÜ. Firma on suhteliselt noor. Autor analüüsib väikefirmat üldiselt, meeskonda ja selle rolle ning bakalaureusetöös käsitletud projekti.

2.1 IT-Projekt OÜ

Ettevõte alustas oma tegevust 2010.aasta märtsis. Selle eesmärk on luua uus koduleht kindlale platvormile või täiendada olemasolevat. Samuti on aidanud firma leida nii sobivaid arhitektuurilisi lahendusi kui ka süsteemide omavahelisi liidestatusi. Firma on tegelenud nii lihtsamate kui ka keerulisemate lahendustega. IT-Projekt aitab tarkvaralahendusi luua ja neid ka edaspidi hallata ning töös hoida. [3]

2.1.1 Meeskond

Kokku on IT-Projekt OÜ-s 7 töötajat:

- Projektijuht – arenduse esiotsas, suhtleb pidevalt kliendiga ning käib kohtumistel. Vastavalt vajadusele korraldab lisakohtumisi. Märgib kohtumistel ära nõutavad funktsionaalsused ning koostab *task*'e arendajate jaoks. Arutab kliendiga läbi kõikvõimalikud aspektid, mis seondub tarkvaraarendusega. Projektijuht täidab ka analüütiku ja testija rolli, kus kordineerib *task*'i valmimist ning testib selle tulemust vastavalt funktsionaalsusele või disainile.
- Disainer – ei tööta meeskonnaga koos. Projektijuht võtab disaineriga ühendust vastavalt vajadusele, kui on kliendiga kohtumine, kus pannakse paika tarkvara disain. Peab arvestama tarkvaraarenduses kujunevate problemaatiliste kohtadega enne nende tekkimist. Teeb disaini ära täpselt vastavalt kliendi vajadustele esimese korraga, kõik muud muudatused arutatakse läbi ainult meeskonna(sealhulgas projektijuhi) ja kliendi vahel

- Arendajad – teevad ära ülesandeid vastavalt vajadustele. Neile on määratud hulk *task*'e mida tuleb kindla aja jooksul täita. Peavad arvestama skoobi muudatustega. Ettevõttes on nii *back-end* kui ka *front-end* arendajad, kuid on ka neid, kes tegelevad mõlemaga. Enamus arendajaid on veel üliõpilased ning seetõttu töötavad osakoormusega.

2.1.2 Projekt

Bakalaureusetöös käsitletakse ühe e-poe arendamist, mille nägemus on uuenenud inimesest, kes on teadlik oma tegelikust olemusest, valdab teadlikkust, kasutab realismil põhinevat loogikat, muudab teadlikult tegutsemisviisi või teemat kooskõlaliseks enda või teiste vajadustega, oskab iseseisvalt vabaneda enda üleskerkivast takistusest, on võimeline eristama olulist ebaolulisest.

E-poe tulevaseks kasutajaks on inimene, kes soovib ennast vaimselt arendada või õppida midagi uut. Avatud loengud on hea viis saada aimu teadlikkuse rakenduse õpetusest või ka omandada lisaks juba õpitule täiendavaid teadmisi ja oskuseid. Kuulajate küsitud teemad hõlmavad erinevaid inimeseks olemise, vaimse arengu ja igapäevaeluga seotud küsimusi ja probleeme.

E-poe arendamisega alustati 2015.aasta detsembris ning arendus kestab 2016.aasta maini. Arenduse käigus kasutati kahte erinevat agiilse tarkvaraarenduse metoodikaid. Alustati arendamist Kanban'iga ning poole arenduse pealt mindi üle Scrumban'iga seotud praktikate juurde. Scrumban'i kasutamise mõtteks oli tarkvaraarenduse efektiivsuse tõstmine ning meeskonnasisese suhtlemise parandamine võrreldes Kanban'iga. Scrum'i rakendamine ei tundunud meeskonna jaoks vajalik, kuna selle kasutamine ei ole niivõrd vajalik väikse projekti arendamisel.

Kuna kohtumised kliendi ja projektijuhi vahel toimusid arenduse käigus nädalas vähemalt korra, siis parimaks variandiks oli kasutada agiilse tarkvaraarenduse metoodikat. Koostöö kliendiga oli pidev ning iga nädal tuli kliendile ette näidata tehtud töö, samuti meeskonnas oli arendus tähtsal kohal ning peaaegu välditi dokumenteerimist. Seoses kliendi soovidega pidi meeskond olema harjunud sellega, et e-poe funktsionaalsusi ja disaini tuli tihti muuta.

3 Agiilsest tarkvaraarendust üldiselt

Käesoleva peatüki eesmärgiks on anda ülevaade agiilse tarkvaraarenduse ajaloost ja samuti tuua välja agiilsele lähenemisele iseloomulikud väärtused ja põhiprintsiibid.

Tarkvaraarenduse meetodeid jaotatakse tihtipeale agiilseteks ja traditsioonilisteks tarkvaraarendusmeetoditeks ning neist esimene on tänapäeva lähenemisviis. Agiilsed tarkvaraarendusmeetodid on aidanud suuresti kaasa programmeerijate rahulolule võrreldes enne kasutuses olnud traditsiooniliste meetoditega. [19]

3.1 Agiilse tarkvaraarenduse ajalugu

1980ndatel ja 1990ndate alguses oli laialtlevinud põhimõte, et parim viis kuidas saavutada parem tarkvara, on läbi hoolika projektiplaneerimise, formaliseeritud kvaliteeditagamise, analüüsi ja disaini meetodite kasutamise ning kontrollitud ning range tarkvara arendamise protsessi. Selline vaade tuli tarkvara arendamise kogukonnalt, kes olid seotud suurte tarkvarasüsteemide arendamisega. [4]

See tarkvara oli arendatud eri ettevõtete suurte meeskondade poolt. Meeskonnad olid tihti geograafiliselt hajututad ning töötasid tarkvaraga pikka aega. Näiteks õhusõidukite juhtimissüsteemides kasutati seda tüüpi tarkvara, mille arendamine võib aega võtta kuni kümme aastat alates esimesest spetsifikatsioonist kasutuselevõtuni. Lisakulutused süsteemi planeerimisel, projekteerimisel ja dokumenteerimisel on põhjustatud just sellisest plaanist. Selline olukord õigustab end ainult seljuhul, kui tegemist on kriitilise süstemi loomisega, kus töö peab olema mitme meeskonna vahel kordineeritud ning kus mitmed erinevad inimesed tegelevad hooldusega. [4]

Kui seda raskekoelist plaani järgivat tarkvara arendamise lähenemisviisi kasutatakse väikeste ja keskmise suurusega ettevõtete süsteemidel, siis võib juhtuda, et selline tarkvara arendamine ei tasu end ära, kuna üldkulud ületavad lubatud piire. Rohkem aega kulutatakse ka sellele, kuidas süsteemi tuleks arendada. Kui süsteemi nõuded muutuvad, ja need muutuvad, siis spetsifikatsioon ja disain peab muutuma koos programmiga. [4]

Rahulolematus nende raskekoelist tarkvaratehnika lähenemisviisidega põhjustas seda, et paljud tarkvaraarendajad tegid 1990ndal aastal ettepaneku uue agiilsete meetodite jaoks. Tänu sellele sai arendusmeeskond keskenduda rohkem tarkvarale kui disainile ja dokumentatsioonile. Agiilsed meetodid sobivad kõige enam tarkvaraarendusele, kus nõuded muutuvad pidevalt arendusprotsessi käigus. Nende eesmärgiks on tuua töötav tarkvara kliendini nii kiiresti kui võimalik, kes saavad uusi soove ja muudatusi avaldada hilisemate iteratsioonide ajaks. Samuti on tähtis elimineerida ebavajalik dokumentatsioon, mida arvatavasti kunagi ei kasutata. [4]

Agiilsete meetodite filosoofia avaldub agiilse tarkvaraarenduse manifestis, mis oli kokku lepitud paljude juhtivate arendajatega. [4]

3.2 Agiilse tarkvaraarenduse manifest

“Tarkvara luues ning teisi tarkvara loomise juures aidates oleme leidnud selleks tööks paremaid viise . Oleme hakanud hindama:

- Inimesi ja nendevahelist suhtlust rohkem kui protsesse ja arendusvahendeid
- Töötavat tarkvara rohkem kui kõikehõlmavat dokumentatsiooni
- Koostööd kliendiga rohkem kui läbirääkimisi lepingute üle
- Reageerimise muutunud oludele rohkem kui algse plaani järgimist

Ka parempoolsetel teguritel on väärtus, kuid me hindamine vasakpoolseid tegureid kõrgemalt[1].

3.3 Agiilse tarkvaraarenduse manifesti põhimõtted

Manifestis on ära kirjeldatud 12 põhimõtet, mis avaldati samal ajal manifestiga. Alljärgnevalt on need põhimõtted välja toodud: [2]

1. Kõige olulisem on tagada kliendi rahulolu, tarnides talle vajalikku tarkvara võimalikult kiiresti ja tihti.

2. Mõistame muutuvaid olusid, isegi kui need ilmnevad arenduse lõppjärgus. Agiilsed meetodid pööravad sellised muutused meie kliendi konkurentsieeliseks.

3. Tarnime tarkvara nii tihti kui võimalik, soovitavalt iga paari nädala kuni paari kuu tagant.

4. Valdkonna spetsialistid ja tarkvaraarendajad peavad töötama igapäevaselt koos kogu projekti vältel.

5. Projekti edukuse aluseks on motiveeritud inimesed. Loo neile meeldiv ja toetav töökeskkond ning nad saavad iseseisvalt tööga hakkama.

6. Kõige tõhusam ja tulemuslikum viis info jagamiseks arendusmeeskonnas on näost näkku vestlus.

7. Edu peamiseks mõõdupuuks on töötav tarkvara.

8. Agiilse tarkvaraarenduse protsessid soodustavad jätkusuutlikku arendust. See tähendab, et projektiga saab samas tempos jätkata määramata aja jooksul.

9. Tehnilist täiuslikkust ja head disaini pideva tähelepanu all hoides tagatakse tarkvaraarenduse kiirus ja paindlikkus.

10. Lihtsus - ebavajaliku töö tegematajätmise kunst - on väga oluline.

11. Parimad arhitektuurilised lahendused, nõuded ja disain tekivad iseorganiseeruvates meeskondades.

12. Meeskond otsib regulaarselt võimalusi saamaks veelgi tõhusamaks ja muudab end vastavalt vajadusele.

4 Agiilse tarkvaraarenduse metoodikad

See peatükk analüüsib peamisi võimalusi tarkvaraarenduse protsessides erinevate agiilsete metoodikate järgi. Wikipedia järgi „agiilsed metoodikad on fokuseeritud tarkvaraarenduse elutsükli erinevatele aspektidele. Mõned keskenduvad praktikale (näiteks XP, Agiilne modelleerimine), kuid samal ajal teised aga tarkvaraarenduse projektide haldamisele.“ [19]

Kuigi kõik agiilsed meetodikad sõltuvad tugevalt järgulisest arendamisest ning üleandmisest, siis kasutavad nad erinevaid protsesse selle saavutamiseks. Samas, kasutavad nad neid samu agiilse tarkvaraarenduse manifesti põhiprintsiipe, ning selle tõttu sarnanevad omavahel. Need põhiprintsiibid on kirjeldatud eelnevates peatükkides (vt. 3.2 Agiilse tarkvaraarenduse manfiest ja vt. 3.3 Agiilse tarkvaraarenduse manifesti põhimõtted). [4]

Seoses sellega, et erinevate agiilsete tarkvaraarenduse metoodikate kohta on väga palju informatsiooni, keskendus lõputöö kirjutaja agiilsetele metoodikatele, mis hõlmavad endas pigem tarkvaraarenduse projektide haldamist.

Lõputöö kirjutaja uuris kolme erinevat metoodikat Scrum, Kanban, Scrumban ja see peatükk analüüsib ning võrdleid neid omavahel.

4.1 Scrum

Scrum on agiilne viis projekti haldamiseks ning tavaliselt on see seotud tarkvaraarendusega. Enamasti peetakse Scrumi agiilse tarkvaraarenduse metoodikaks, kuid parem on seda vaadelda kui arendusprotsessi haldamiseks kasutatavat raamistikku. Esialgselt oli Scrum loodud meeskondadele, kus kõik liikmed said osaleda igapäevastel koosolekutel. Samas, kuna tänapäeval tarkvaarendus on tähtis ka meeskondadele, kus liikmed asuvad erinevates maailmakohtades, siis seda metoodikat rakendatakse ka sellise olukorra puhul. [4] [5]

Scrum on sobilik projektidele, mille vajadused muutuvad pidevalt arendusprotsessi käigus. Selle meetodi käigus ei anta täielikult detailne plaan, kuidas projekti arendada,

vaid selle asemel jäetakse palju otsustada Scrumi tarkvaraarendusmeeskonna liikmetele, kuna nemad teavad kõige paremini, kuidas kindlaid probleeme lahendada. Idee Scrumi taga on selline, et kõik meeskonnaliikmed on võimelised otsuseid langetama ise ning sellepärast ei ole Scrumi meeskonnas põhilist juhti, kes vastutab kõigi töö eest ja annab pidevalt juhiseid. [4] [5]

Scrumi tarkvaraarenduse protsess koosneb arendustsüklitest, mida nimetatakse *sprint*'ideks ning mis kestavad 1-4 nädalat. Scrumi mudeli järgi iga *sprint* algab planeerimisega ning lõppeb ülevaatusega.

Järgnevalt on kirjeldatud Scrumi rollid:

- Product owner (tooteomanik)
    - vastutab selle eest, et toode omab maksimaalset väärtust ja samuti vastutab ka Scrum'i meeskonna tehtud töö eest, et see saavutaks õige tulemuse. Vastutab tegemata tööde loetelu eest. Omab visiooni sellest, mis tulemuse peab meeskond arendamise käigus saavutama. [5] [6]

- Scrum master
    - Korraldab igapäevaseid koosolekuid, vastutab Scrum'i arusaadavuse ja kasutatavuse eest, kontrollib *backlog*'is olevaid *task*'e, mõõdab meeskonna progressi. Scrum master suhtleb kliendiga meeskonnavälisel ja vastutab takistuste kõrvaldamise eest, et meeskond suudaks tarnida oodatavaid tulemeid. [5] [7]

- Development team (arendusmeeskond)
    - koosneb spetsialistidest, kes töötavad, et tarnida valminud toote juurdekasvu ehk inkrementi iga sprindi lõpuks [8]

Scrumi praktikad on alljärgnevad: [8]

- Sprint
    - 1-4 nädalat kestev arendustsükkel, mille käigus arendatakse välja kindlad funktsionaalsused

- Task
    - Kindel ülesanne inimesele, mille maht ei tohi ületada 12 tundi (või 2 päeva), kuid on tavaline, et meeskond nõuab selle lahendamist ühe päevaga

- Product backlog (kogu tarkvaratoote tegemata tööde loetelu)
    - Tähtsuse järjekorras olev tegemata tööde loetelu, seotud funktsionaalsuse nõudmistega

- Sprint backlog (Sprindi tegemata tööde loetelu)
    - Tähtsuse järjekorras olev nimekiri *task*'ide lahendamiseks *sprint*'i käigus

- Inkrement
    - *Sprint*'i ja eelmiste *sprint*'ide jooksul valminud inkrementide väärtus tarkvara tööde loetelu osadest

- Sprint review (ülevaatus)
    - Tagasiside tööle, mis oli tehtud ja mis ei olnud; Valmis töö esitatakse kliendile

- Sprint retrospective (retrospektiiv)
    - Arutatakse eelmise sprindi kohta; Räägitakse sellest, mis läks hästi ja mis läks halvasti ja kuidas oleks võimalik seda paremaks muuta

- Stand-up meeting (igapäevane koosolek)
    - Lühike koosolek, kus arutatakse eile tehtud ja täna plaanis olevaid töid ning mis takistab nende edenemist

- Sprint planning (planeerimine)
    - Iga sprindi jaoks tööde valimine ja hindamine

- Burndown chart
    - Progressi jälgimise tööriist, kui palju arendust on veel ees projekti lõpuni

Järgnevalt esitatakse Scrum'i lihtsustatud mudeli kirjeldus: [9]

- Tooteomanik koostab prioriteedi järjekorras paikapandud loetelu tegemata töödest, mida nimetatakse product *backlog*'iks.

- Sprindi planeerimise ajal, meeskond võtab ette väiksema osa tegemata tööde loetelust, ning otsustab, kuidas neid osasid implementeerida.

- Meeskond omab kindlalt ajapiirangut, mida nimetatakse sprint'iks ja mis kestab 1-4 nädalat. Selle aja jooksul peab meeskond oma töö lõpetama ning selle abiks kasutatakse ka Scrum'i igapäevaseid koosolekuid.

- Arendusprotsessi käigus hoiab Scrum Master meeskonda pidevalt keskendunud püstitatud eesmärkide saavutamisele.

- Sprint'i lõpus peab töö olema tarnitav ja täitma nõutavaid funktsionaalsusi. Tarkvara peab olema valmis kliendile üleandmiseks, poeriiulile asetamiseks või osanikule näitamiseks.

- Sprint lõpeb ülevaatuse ja retrospektiiviga.

- Samas, kui algab uus sprint, siis meeskond võtab ette väiksema osa tegemata tööde loetelust, ning otsustab, kuidas neid osasid implementeerida

Scrumi arendustsüklit saab kujutada järgneva joonisega: [9]

Joonis 1 Graafiline kujutis Scrum'i arendusmetoodikast

4.2 Kanban

Kanban on järjekordne raamistik, mis rakendab agiilset tarkvaraarendust. Kanban on jaapanikeelne sõna, mis tähendab visuaalset kaarti. Eelmise sajandi keskpaigas kasutasid seda edukalt autotööstuse töötajad, kes lihtsustasid oma tööd sellega, et näitasid, kui kaugel nad oma tööga parasjagu on ning kas on võimalik koormust tõsta. Tänu sellele mudelile vähendati kulu ja tõsteti väärtust töö käigus. [10] [11]

Praegusel ajal on Kanban tähtsal kohal infotehnoloogia valdkonnas, kus seda peamiselt kasutatakse töö visualiseerimiseks Scrumi meeskondades. Kanban aitab meeskondadel muuta planeerimist paindlikumaks, võimaldab kiiremat tulemit, aitab parandada koostööd ning annab parema ülevate tarkvaraarenduse protsessist. [10] [11]

Kanban'i meeskond on keskendunud ülesannetele, mis on parasjagu töös. Kui meeskonnaliige täidab tööülesande, siis võib võtta järgneva *backlog*'ist. Tooteomanikule on antud vaba voli muuta ülesandeid *backlog*'ist, kuna iga muudatus väljaspool töös olevatest ülesannetest, ei sega meeskonna tööd. Nii kaua kuni tooteomanik hoiab kõige olulisemad ülesanded *backlog*'i tipus, teab meeskond, et nad täidavad olulisemad nõuded enne. Selle tõttu ei vaja Kanban paikapandud iteratsiooni pikkust nagu Scrum'is. [10]

Multitegumtöö vähendab kasumtegurit, kuna mida rohkem töös olevaid ülesandeid on, seda vähem saab spetsiifilisele ülesandele keskenduda. See on peamine põhjus, miks Kanban'i puhul piiratakse töös olevaid ülesandeid. [10]

Sooritusaeg on põhimõõdik Kanban'i meeskondade jaoks. Sooritusaeg on aeg, mida arvestatakse alates kliendi tellimusest kuni toote kättetoimetamiseni. Seda aega optimiseerides, saab meeskond ennustada tulevikus tehtud töö kättetoimetamist. [11]

Kanban'i peamiseks edu põhjuseks on selle visuaalne kontseptsioon. Pilt ütleb rohkem kui tuhat sõna, kuna aju töötleb visuaalset informatsiooni 60000 korda kiiremini kui teksti. Visuaalne informatsioon hõlmab 90% andmetest, mis jõuavad ajju, mis omakorda tõesta seda, et aju on piltide suhtes vastuvõtlikum. [11]

Kanbani peamised tunnused on: [11]

- Visualiseeri töövoog
    - Visuaalse mudeli loomisega töö ja voo jaoks, on palju parem ülevaade töövoo üle Kanbani süsteemis. Tänu sellele, et töö visualiseeritakse, muutub kohe ka suhtlemine ja koostöö efektiivsemaks. Iga funktsionaalsuse arendus peab pidevalt tahvlil liikuma järgmisesse faasi sel hetkel kui ta lõplikult valmis saab.

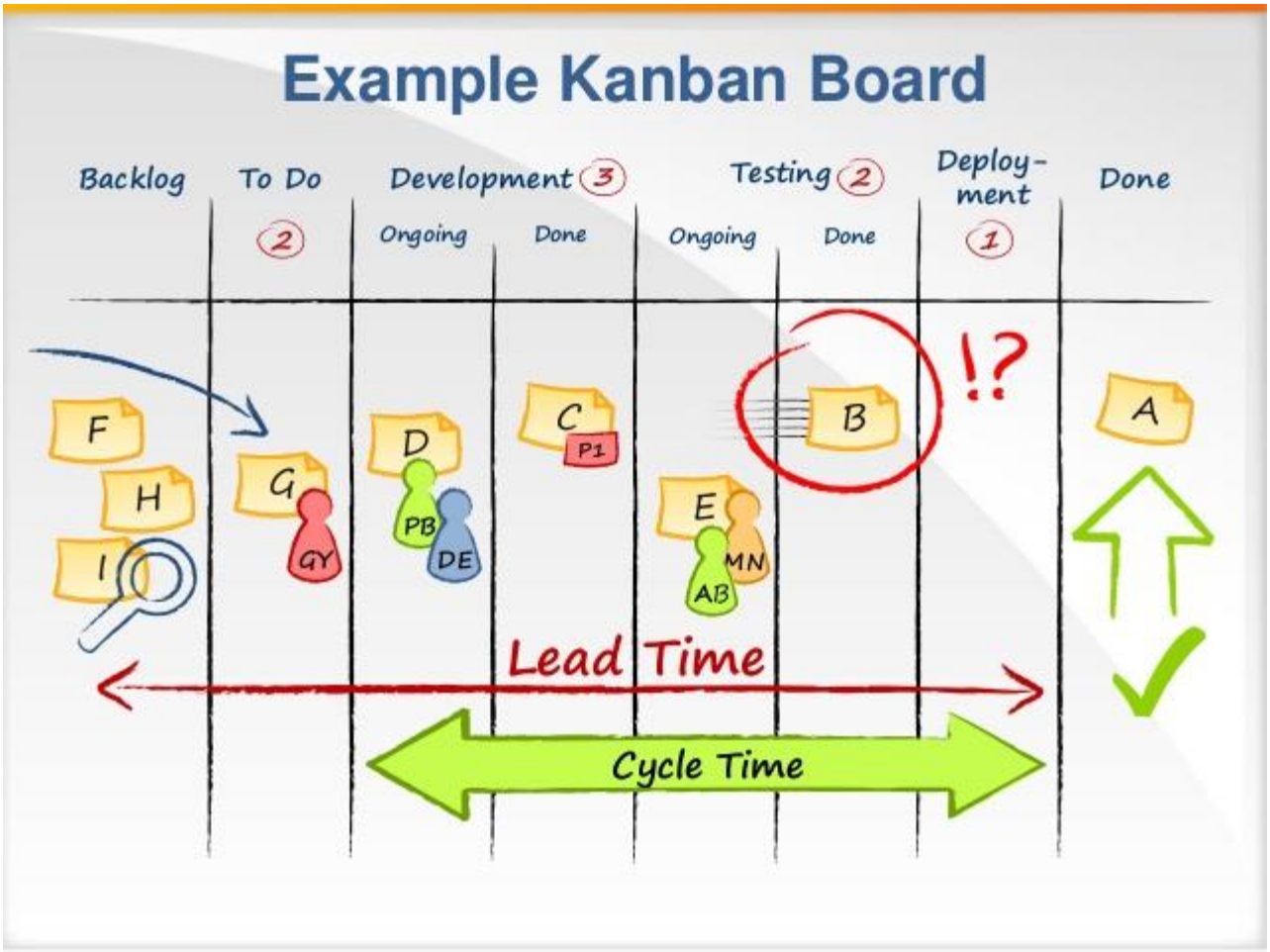
- Piira töös olevaid ülesandeid
    - Töös olevate ülesannete piiramine aitab kaasa aja kokkuhoiule läbi Kanbani süsteemi, samuti hoiab see ära pideva ülesannete vahetamise ning nende tähtsuse muutmise. Samuti tuleb eemaldada liiasust, mis ei anna projektile ega kliendile lisaväärtust.

- Kontrolli voogu
    - WIP kasutamine aitab töövoo efektiivsemaks muutmisele kaasa. Tänu sellele saab töövoogu analüüsida ja tänu sellele seda täiustada ning võimalikke probleeme ennetada.

- Pidev arendamine
    - Meeskonnad näevad pidevalt oma töö voogu, kvaliteeti. Eksperimendid ja analüüs muudavad tarkvaraarendust efektiivsemaks.

- “Pull” põhimõte
    - Pudelikaelade vältimiseks on igas faasis kindel arv lubatud ülesandeid, seda arvu ei ole lubatud ületada. [12]

Kanbani tahvlit saab kujutada järgneva joonisega: [10]



Joonis 2 Graafiline kujutis Kanban tahvlist

4.3 Scrumban

Scrum on agiilne viis projekti haldamiseks ning tavaliselt on see seotud tarkvaraarendusega. Seda võib pidada Scrum’i ja Kanban’i hübriidiks. Enamasti peetakse Scrumi agiilse tarkvaraarenduse metoodikaks, kuid parem on seda vaadelda kui

arendusprotsessi haldamiseks kasutatavat raamistikku. Scrumban sarnaneb suures osas Scrum'ile, kuid samuti laenab palju omadusi Kanban'i metoodikast. Selle tõttu, et tegu on hübriidiga, kasutavad paljud meeskonnad seda arenduses erinevalt. Kasutamisviis sõltub suuresti meeskonna kogemusest, organisatoorsest strukuurist ning kultuurist.[13] [14]

Kanban peab kinni erinevatest uskumustest, nagu näiteks töövoo visualiseerimine, töös olevate ülesannete piiramine ning produktiivsuses mõõtmine. Scrumban pean kinni samuti nendest uskumustest arenduse käigus.[13]

Tihtipeale Scrum kasutab lugu, mis koosneb väiksematest *task*'idest, mis moodustavad endast töö, mis tuleb ära teha. Scrumban'i kasutamise käigus kasutatakse seda lugu, mis liigub arenduse käigus mööda erinevaid etappe. Need lood, millega meeskond tegeleb ning kes vastutavad selle kindlate aspektide eest, on nähtavad kõigile igas etapis.[13]

Scrumban ei kasuta praktika poolest WIP aja limiteerimist, vaid kasutab selle asemel WIP üldist vähendamist iga etapi puhul. Selle eeliseks on see, et kompositsiooniga seotud probleemid muutuvad selgemaks. Lisaks see lubab meeskonnal võtta tööse erineva suurusega lood ning ei pea arvestama reeglite paikapanemisega.[13]

Scrumban, nagu ka Kanban, kasutab produktiivsuse mõõtmiseks sooritusaega. Sooritusaeg on põhimõõdik Scrumban'i meeskondade jaoks. Sooritusaeg on aeg, mida arvestatakse alates kliendi tellimusest kuni toote kättetoimetamiseni. Seda aega optimiseerides, saab meeskond ennustada tulevikus tehtud töö kättetoimetamist. [13][14]

Scrumban'i rakendamise põhikäigud: [15]

1. Esialgselt tuleb paika panna lihtne Kanban'i tahvel, mis koosneb kolmest osast. Esimene osa koosneb ülesannetest, mis teha, teine osa töös olevatest ülesannetest ning kolmas osa ülesannetest, mis on tehtud. Selles etapis ei pea muretsema planeerimise pärast, kuna oluline on anda meeskonnale piisavalt tööd. Üks *task* meeskonnaliikme kohta on alustuseks piisav, kuni hakata lisama juurde uusi ülesandeid.

   Järgmine oluline praktika, mida tuleks kasuta on „Pull põhimõte". Kanban'i meeskondade jaoks ei tule see mõiste üllatusena. „Pull põhimõte" põhiideeks on see, et meeskonnaliikmed saavad ise valida ülesandeid, millega tegelema hakata.

Scrumban'i puhul lisatakse *task*'id tahvlile, kuid neile ei määrata tegijat. Meeskonnaliikmed valivad ise tahvlil olevate *task*'ide seast ülesanded, millega nad soovivad tegeleda ning mis vastab nende tasemele.

Lisaks, võib arendusprotsessi viia järgmisele tasemele ning tegeleda *task*'ide prioritiseerimisega. Selleks kas korraldatakse koosolekuid, kus meeskond paneb paika olulisemad ülesanded, või projektijuht otsustab ise, millised *task*'id peab lähiaja jooksul ära tegema. Samuti koosolekuid on mõistlik korraldada juhul, kui *backlog*'is on mingi kindel arv tegemata jäänud ülesandeid. Koosoleku käigus lisatakse *backlog*'i uusi *task*'e, mis on vaja ära teha arendusprotsessi käigus.

Kanban'i puhul peab olema paika pandud WIP limiit, kuna niimoodi arendajad on oma ülesannetele rohkem keskendunud ning vähendatakse multitegumtöö võimalust. Kõige parem on panna WIP kindel limiit, kus arv tähendab mitme *task*'iga võib meeskond tegeleda. See tähendab, et kui meeskond koosneb neljast liikmest ning WIP limiidiks on samuti neli, siis meeskond võib maksimaalselt tegeleda nelja *task*'iga. Kui *task*'i täitmisel esinevad probleemid ning seda ei ole võimalik ära teha, siis pannakse see tagasi *backlog*'i ning sealt võetakse töösse uus ülesanne. [13]

2. Arendusprotsessi käigus on tähtis pidevalt areneneda. Scrumban'i puhul on oluline selgeks teha, kuidas planeerida *backlog*'i ning kuidas *task*'e prioriteedi järjekorda panna. *Backlog* võib olla samamoodi piiratud nagu WIP, aga see tähendaks, et oleks mingi kindel arv kohti tahvli erinevates osades, mida ei tohi ületada.

   Esialgselt on vaja läbida mitu iteratsiooni, et aru saada, kui palju *task*'e suudab meeskond iteratsiooni käigus käsitleda. Samuti on vaja arvutada *backlog*'i suurus. Kui meeskond suudab nädalas täita kümme *task*'i, siis see tähendab, et meeskonna *task*'ide täitmiskiirus on kaks *task*'i päevas. Tänu sellele kalkulatsioonile saab planeerida järgmist iteratsiooni, kus on meeskonna jaoks võimalik määrata samuti 10 *task*'i.

Tavaliselt on mõistlik ühe iteratsiooni kohta planeerida vähe *task*'e. Kui *task* on sooritatud, siis Kanban'i tahvlil tuleb see viia veergu, mis vastutab tehtud ülesannete eest. Kui on aga olemas ka vähekäigud, siis peab *task* ka need läbi käima. Kogenud meeskonnad planeerivad oma tegevust niiviisi, et *backlog*'is on ainult kaks või kolm *task*'i.

3. Kui tahvlil on WIP limiidiks 5 ja veergude limiidiks on määratud 4, siis tähendab see seda, et meeskonnas on lubatud mõnel määral ka multitegumtöö. Meeskond võib maksimaalselt kasutada oma tööjõudu ühes kindlas veerus, kuid teiste veergude puhul võib põhimõisteks olla multiteugmtöö. Näiteks, kui testimise veerus ei ole võimalik kindla *task*'iga tegeleda, kuna testimise keskkond ei tööta, siis võib selle ülesande sinna jätta kuni keskkond tehakse korda. Sellise viisiga on võimalik vältida võimaliku pudelikaelaefekti tekkimist.

   Joonis 3 näitab lihtsustatud kujul, kuidas Scrumban toimib. [15]

4. Kui on kindel arusaam sellest, kui kaua võtab meeskonnas ühe *task*'i sooritamine aega ning kui mitu *task*'i on meeskond võimeline nädalas tegemas, siis on võimalik kliendiga kindla aja suhtes kokku leppida, millal üks või teine asi valmis saab. Tänu töövoo visualiseerimisele on võimalik ennustada, kui mitu *task*'i on meeskond võimeline tegema antud aja jooksul.

   Scrumban'i kasutamise puhul on planeerimine tähtsamal kohal kui tehtud töö esitamine. Planeerimist korraldatakse ainult vajadusel ning valmis tööd esitatakse siis kui võimalik.

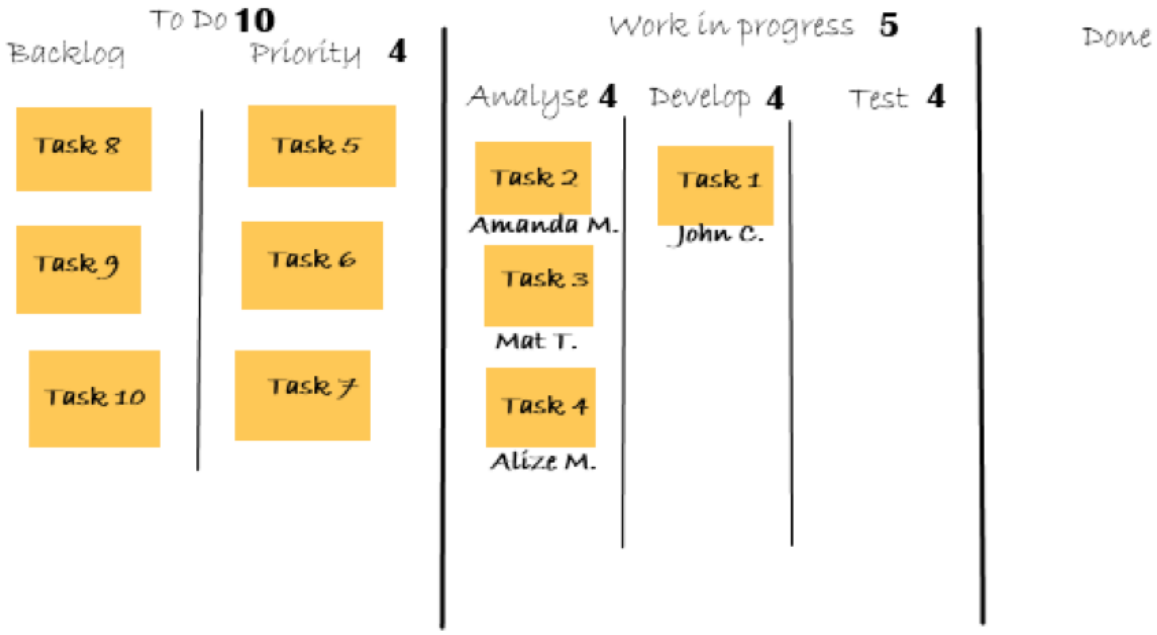
   Scrumban'i puhul järgitakse mitmeid reegleid:

   - *Backlog* peab olema võimalikult väike
   - Planeerimine peab toimuma ainult vajadusel
   - *Backlog* peab olema jagatud mitmesse veergu, et aru saada, millised ülesanded on teistest tähtsamad

- Multitegumtöö on lubatud, kuid seda on vaja kontrolli all hoida

- Töö põhimõõdikuks kasutatakse sooritusaega ning oluline on teada, mitu *task*'i on meeskond võimeline nädala/iteratsiooni jooksul täita

- Meeskonnaliikmed võivad ise valida ülesandeid *backlog*'is vastavalt oma tasemele



Joonis 3 Graafiline kujutis Scrumban'i tahvlist

4.4 Võrdlus

4.4.1 Scrumi eelised ja puudused

Analüüsin Scrum’i eeliseid ja puuduseid ning kirjeldan neid. [16] [17]

Tabel 1 Scrum'i eelised

Eelis	**Kirjeldus**
Iteratiivne meetod	Projekti pidev jälgimine ning koheste tulemuste saavutamine. Jälgimise puhul kontrollitakse pidevalt tarkvaraarenduse töövoogu, mis näitab kui efektiivselt ja kui palju tööd on tehtud
Kohanemisvõime	Scrumi puhul võib nõudmisi pidevalt muuta või samuti neid juurde lisada
Pidevalt kasvav produktiivsus	Tarkvara on võimalik arendada kiiremini, kui on teada selleks kuluv aeg ning meeskonna tööproduktiivsus.
Koostöö	Pidev suhtlus kliendiga
Pidev suhtlus ja täiustatud kommunikatsioon	Kõik Scrumi meeskonna liikmed on arendusprotsessiga pidevalt seotud ja saavad erinevaid otsuseid mõjutada. Samuti iga meeskonnaliige võib oma arvamust avaldada ning tänu sellele hoitakse taktistustest eemale.

Tabel 2 Scrum'i puudused

Puudus	**Kirjeldus**
Ajakulu	Peale iga *sprint*'i lõppu on vaja korraldada uus planeerimine, mis võib võtta palju aega, kui planeeritakse mahukam *sprint*.
Vajab kogenud meeskonda	Scrumi metoodika on mõeldud meeskonnale suurusega 5-8 inimest. Kõik meeskonna liikmed peavad olema pühendatud projekti, kuna Scrum vajab kogenud meeskonda. Kui keegi algaja on meeskonna, siis see võib arenduse kiirust vähendada ning on oht projekt õigeks tähtajaks mitte valmis saada.
Pole lõpptähtaega	Scrum'il pole lõpptähtaega, mistõttu viimati avaldatud tarkvaras võib puududa mitmeid vajalikke funktsionaalsusi, mida klient soovis ning seetõttu on vaja ka neid funktsionaalsusi lisada.
Iteratsioonide selge definitsioon	Scrum'i puhul on üks raskemaid ülesandeid iteratsioonide selge defineerimine. Kui selle defineerimine ei ole täpne, siis võib see mõjutada ajalist ning finantsilist poolt.

4.4.2 Kanbani eelised ja puudused

Analüüsin Kanban’i eeliseid ja puuduseid ning kirjeldan neid. [16] [17]

Tabel 3 Kanban'i eelised

Eelis	Kirjeldus
Liiasuse väike osakaal	Tarkvara arendamisel tehakse ära ainult need osad, mis on vajalikud, mille tõttu üleliigne arendamine on peaaegu võimatu.
Arendamise paindlikkus	Kanban’i puhul on projekti pidev nõutud funktsionaalsuste ülevaatamine tähtis, mistõttu nõudmiste muutuse korral muutuvad ka arendaja ülesanded selle projekti suhtes.
Produktiivsuse ja kasuteguri kasv	Kanban’i üheks suurimaks eeliseks on ajakulu vähendamine, tänu millele saavad arendajad keskenduda oma tööle.

Tabel 4 Kanban'i puudused

Puudus	Kirjeldus
Protsessi voog	Kanban’i puhul on vajalik iganädalane planeerimine igapäevase paindlikkusega, mistõttu selle kasutamine ei sobi mitme või lühiprojektide arendamisel

4.4.3 Scrumbani eelised ja puudused

Analüüsin Scrumban'i eeliseid ja puuduseid ning kirjeldan neid. [14] [16]

Tabel 5 Scrumban'i eelised

Eelis	**Kirjeldus**
Ajakulu vähendamine	Scrumban'i puhul kasutatakse sellist planeerimise tehnikat, mille puhul ei ole vajalik aja ning üldist planeerimist korraldada.
Kvaliteet	Tänu Scrumban'i ajakulu vähendamisele on võimalik keskenduda kvaliteedi kontrollile. Varuaeg aitab arendamise käigus leida tarkvara vead ning nende leidmisel elimineeritakse need koheselt.
Liiasuse vähendamine	Scrumban'i puhul kasutakse diagramme, mis aitavad välja selgitada, millised osad ei lisa väärtust tarkvarale ja kliendile

Tabel 6 Scrumban'i puudused

Puudus	**Kirjeldus**
Harjumine	Seoses Scrumban'ile üleminekuga on vaja harjuda Scrumban'iga seotud praktikatega ning aktiivselt kasutada arenduse käigus

4.4.4 Scrum vs. Scrumban

Võrdlen Scrum'i ning Scrumban'i omavahel väljatoodud tunnuste abil.[18]

Tabel 7 Scrum vs. Scrumban

	Scrum	**Scrumban**
Tahvel/Artifaktid	Tahvel, tegemata tööde loetelu, progressi vaatlemine	Tahvel
Tseremoniaalsused	Kõik Scrum'iga seotud tseremoniaalsused	Vajadusel korraldatakse Scrum'iga seotud tseremoniaalsusi
Iteratsioonid	Jah	Ei
Hinnang	Jah	Ei
Meeskonnad	Spetsialiseeritud	Võivad olla spetsialiseeritud
Rollid	Product Owner, Scrum Master, meeskond	Meeskond + vajaminevad rollid
Meeskonnatöö	Sõltub *task*'i vajadusest	Pidev koostöö
Muutused	Peab ootama kuni järgmise *sprint*'ini	Lisatakse tahvlile nõutud muutused
Kogu tarkvara tegemata tööde loetelu	Koosneb erinevatest lugudest	*Task*'id *backlog*'i veerus
Takistused	Tegeletakse koheselt	Välditakse

4.4.5 Kanban vs. Scrumban

Võrdlen Kanban'i ning Scrumban'i omavahel väljatoodud tunnuste abil.[18]

Tabel 8 Kanban vs. Scrumban

	Kanban	**Scrumban**
Rollid, meeskond	Ettemääratud rollid puuduvad	Meeskond + vajaminevad rollid, speatsialiseerumine *task*'ide järgi
Planeerimine ja igapäevane koosolek	Planeerimine toimub, igapäevane koosolek puudub	Toimub, et saada pidev tagasiside töövoo kohta
Tagasiside ja retrospektiiv	Retrospektiive ei korraldata	Vajadusel korraldatakse retrospektiive
Töövoog	Pidev	Pidev
Task	Ükskõik mis suuruses, kuid ei tohiks ületada 2 päeva	Ükskõik mis suuruses, kuid ei tohiks ületada 2 päeva
Taski määramine	Tavaliselt projektijuhi poolt vastavalt oskusele ja tasemele	*Task'*i on võimalik ise valida nimekirjast
Taskide prioritiseerimine	Puudub	*Task'*idele määratakse nendele vastav prioriteet
Töövoo visualiseerimine	Kasutatakse tahvlit	Kasutatakse tahvlit
„Pull" põhimõte	*task*'ide arvu pole paika pandud, kuid arendusmaht on määratud	*task*'ide arvu pole paika pandud, kuid arendusmaht on määratud

5 Analüüs

Peatükis vaadeldakse ja analüüsitakse Kanban'i arendusmeetodi kasutumisega seotud tseremoniaalsuste mõju meeskonnale 3 kuud enne Scrumban'i rakendamist.

Peatükis vaadeldakse ja analüüsitakse Scrumban'i arendusmeetodi kasutamisega seotud tseremoniaalsuste mõju meeskonnale 3 kuu jooksul.

Tarkvaraarendusettevõttes ei rakendatud Scrum'i arendusmeetodit, kuna see koosneb mitmest erinevast osast, millest tuleb kinni pidada. Antud ettevõttes ei olnud võimalik hakata aktiivselt tegelema Scrum'i kõigi tseremoniaalsustega.

5.1 Kanbani rakendamine

5.1.1 Planeerimine

Kanban'i kasutamise ajal oli tähtsal kohal arendamise tarkvaraarendusega kaasas käiv planeerimine. E-poe arendamise käigus planeeriti iteratsioone nii tihti kui võimalik. Kui *task*'e ei saanud enam *backlog*'ist normaalsel kiirusel võtta, siis planeerimine oli prioriteedi järjekorras esimesel kohal.

Iteratsioonide planeerimine toimus koos projektijuhiga, kes andis ülevaate uutest *task*'idest ning näitas kuhu ja miks oleks vaja muudatus teostada. Eelnevalt oli projektijuht suhelnud kliendiga, kes avaldas oma soove nõutavate funktsionaalsuste kohta, mida tuleks skoobi jooksul ära teha või muuta. Projektijuht kohtus kliendiga vähemalt korra nädalas, et läbi arutada kõik e-poega seonduvad probleemid. Seoses kliendi pideva soovide muutusega, tuli planeerimist korraldada nii tihti kui võimalik, et tarkvaraarendus oleks efektiivne.

Esialgu oli iteratsioonide planeerimine meeskonna jaoks keeruline, kuna liikmete jaoks oli problemaatiline *task*'ide ära paigutamine iteratsiooni. Samuti ei osatud arvestada ajakuluga, mis võib kaasneda nõutavate funktsionaalsuste teostamisega.

Küsitluse alusel arvas üks arendajatest, et tänu sellele, et projektijuht kohtus kliendiga võimalikult tihti, siis oli võimalik pidevalt uusi *task*'e *backlog*'i lisada. Vastaja jaoks oli tähtis, et kõige olulisemad nõutavad funktsionaalsused jõuavad arendajateni nii kiiresti kui võimalik, kuid mainis, et arenduse algstaadiumis oli raske planeerida, millised funktsionaalsused tuleb varem ära teha. Seetõttu oli projekti arendamise kiirus algstaadiumis madal. Teine meeskonnaliige arvas, et pidevad projektijuhi kohtumised kliendiga mitte ainult ei kiirenda tarkvaraarenduse protsessi, vaid vähendab ka üleliigse töö tegemise. Samuti oli teise vastaja arvates kasulik planeerida iteratsiooni koos projektijuhiga, kes sai selle käigus seletada lahti kõik mõttekohad.

Töö autori arvamusel tänu planeerimisele said meeskonnaliikmed paremini aru, mida klient e-poe arendamise käigus vajas ning said tegeleda esmatähtsate ülesannetega. Tänu sellele oli klient alati rahul hetkeseisuga, sest projektijuht sai talle kohtumistel demonstreerida neid funktsionaalsusi, mida ta kõige enam vajas.

5.1.2 Task

Seoses Kanban'i kasutamisega jagati ülesanded, mille tulemus täidab nõutavat funktsionaalsust, *task*'ideks. Kanban'i kasutamise ajal *task*'i suurus ei ületanud 2 päeva ning selle mahuks oli maksimaalselt 12 tundi, nagu mainitud ka eelnevas peatükis (vt. 4.4.5 Kanban vs. Scrumban). Kuna Kanban'i metoodika puhul ei ole määratud *task*'i maksimaalset suurust, siis vajadusel oli seda võimalik ka igal hetkel muuta.

Seoses Kanban'i kasutamisega oli *task*'ide puhul kindlaks määratud selle täitja ehk internetis olemasoleva tahvli peal oli märgitud tegija nimi. Selle määramisega tegeles projektijuht, kes pidi arvestama meeskonnaliikmete oskusi iga *task*'i täitmise puhul.

Task'ide prioriteeti Kanban'i metoodika puhul ei määratud ning selle tõttu tekkis ka e-poe arendamise käigus segadus. Kuna *task*'idele polnud määratud nende prioriteeti, siis arendajad tihtipeale hakkasid tegelema selliste ülesannetega, mis ei olnud tegelikult esmatähtsad. See oli üks peamisi põhjusi, miks arendamise lõpupoole tekkis ajapuudus, kuna tasemepoolest raskemad *task*'id võtsid tegemiseks rohkem aega.

Meeskonnas läbiviidud küsitluse käigus arvasid mõlemad vastajad, et *task*'idele tegija nime määramine oli arenduse käigus kasulik. Võimalik oli kohe tabelist järgi vaadata,

mis ülesanded kuuluvad kindlale isikule. Samuti olid mõlemad arendajad rahul, et *task*'ide kirjeldamisega tegeles projektijuht, ning tänu sellele said arendaja keskenduda rohkem arendusele. Üks arendajatest märkis, et *task*'id peaks olema määratud prioriteedi järgi, et esialgu tegeleda tähtsamate ülesannetega. Samuti arvas ta, et *task*'ide maht on olnud sobilik. Teine vastaja arvas, et *task*'ide mahtu polnud täpselt määratud ning mille tõttu olid ülesanded erineva mahuga, kuid ei ületanud lubatud ülempiiri. Seetõttu jäi tihti iteratsiooni lõpufaasi mitu tähtsat *task*'i teha, mis tegelikult nõudsid rohkem aega.

Autori arvates on selline *task*'i suurus kasulik tarkvaraarenduse käigus, kuna niimoodi on võimalik säilitada pidev töövoog ning efektiivne arendamine. Tänu sellele, et projektijuht jagas *task*'id tegijate vahel ise laiali, oli igal arendajal kindel kohustus ka neid täita. Seega võib väita, et meeskonnas oli igale liikmele kindlalt määratud ülesanded, mis tema peab arenduse käigus ära tegema. Samas, oleks võinud *task*'id määratud prioriteedi järgi, et muuta tarkvaraarendust efektiivsemaks.

5.1.3 „Pull" põhimõte

Kanban'i kasutamise ajal üheks põhiliseks põhimõtteks oli „Pull" põhimõte, mille peamisteks ülesanneteks oli raiskamiste vältimine ja töös olevate ülesannete hulga piiramine.

„Pull" põhimõte oli meeskonnas tähtsal kohal, kuid seda täideti muudetud kujul. Arendaja jaoks määratud *task*'ide arv polnud paika pandud, kuid tähtis oli määramisel jälgida seda, kui suur maht seoses sellega tekib. Samuti oli oluline arendajale määrata õige arv *task*'e, et teda mitte üle koormata. Meeskonna jaoks oli tähtis vanad ülesanded enne ära teha, kui uued töösse võtta, kuna selline viis aitas suurendada meeskonna keskendumist. Samuti sellega vähendati poolenisti tehtud ülesannete olemasolu.

Meeskonnas läbiviidud küsitluse alusel arvas üks vastajatest, et temale määratud *task*'ide arv ning maht oli tema võimetele vastav. Ta ei tundnud, et teda oleks kuidagi moodi tarkvaraarenduse käigus üle koormatud. Teisele meeskonnaliikmele tundus aga, et *task*'ide mahtu pole kusagil täpselt piiratud või väljatoodud ning sellest tulenevalt olid *task*'id tema arvates erineva mahuga. Mõlema meeskonnaliikme arvates oli kasulik ära teha olemasolevad ülesanded enne, kui võtta uued töösse.

Töö koostaja arvamusel aitas „Pull“ põhimõtte kasutamine kaasa sellele, et ei tekkinud arenduse käigus sellist olukorda, kus tarkvaraarendajat koormati üleliigsete või ebavajalikke ülesannetega. Sellised mõtted vastavad ka Agiilse tarkvaraarenduse manifesti põhimõtetes(vt. 3.3 Agiilse tarkvaraarenduse manifesti põhimõtted).

5.1.4 Retrospektiiv

Seoses Kanban’i kasutamisega ei peetud vajalikuks retrospektiivide korraldamist, kuigi samas kontoris töötades tundus see raisatud võimalusena.

Retrospektiiv oleks aidanud meeskonnaliikmetel saada tagasiside selle kohta, kuidas neil parasjagu läheb ja mida võiks paremini teha. Samuti võib väita, et retrospektiivi kasutamine oleks aidanud probleeme varem märgata ja lahendada, sest tagasiside oleks tulnud otse meeskonnaliikmetelt.

Küsitluse käigus arvas üks meeskonnaliikmetest, et retrospektiivide korraldamine oleks väga oluline. Tema arvates aitab pidev üksteise ligidal töötamine kaasa sellele, et on igal hetkel võimalik arutada probleemi üle ja vajadusel nõu anda või saada abi seoses tarkvaraarendusega. Teise arendaja arvates konkreetselt selles projektis retrospektiivide pidamine ei oleks toonud tuntavat erinevust, kuid märkis, et koosolekute pidamine on siiski vajalik. Tema jaoks meeskonna koosolekud aitaksid tööd kiirendada.

Töö autori arvates oleks aidanud retrospektiivi kasutamine meeskonnale kaasa, tänu millele oleks töökeskkond palju meeldivam kui enne, mida mainib küsitluse käigus ka üks meeskonnaliikmetest. Samuti mainitakse seda eelnevates peatükkides (vt. 3.2 Agiilse tarkvaraarenduse manfiest ja 3.3 Agiilse tarkvaraarenduse manifesti põhimõtted, kus peetakse inimesi ja nendevahelist suhtlust rohkem oluliseks kui protsesse ja arendusvahendeid.

5.1.5 Töövoo visualiseerimine

Kanban’i kasutamisega ei visualiseeritud *task*’e küll füüsikalisel tahvlil, kuid selle asemel kasutati internetis olemasolevat tarkvara, mis täitis Kanban’i tahvli funktsionaalsusi. Nagu mainitud ka peatükis 4.2.1 visuaalse mudeliga kaasneb töövoo üle parem ülevaade, samuti aitab see muuta suhtlemist ja koostööd efektiivsemaks.

Esialgu töövoo visualiseerimise suurimaks probleemiks oli tahvlil olevate *task*'ide ümberpaigutamine järgnevatesse faasidesse, sest nagu mainitud peatükis 4.2.1 on Kanban'i tahvli üheks tähtsaks omaduseks see, et *task*'id ei jää ühele kohale kauaks seisma, vaid pidevalt liiguvad edasi. Meeskonnaliikmetel oli raskusi *task*'ide grupeerimisega ning sellega harjumine võttis aega.

Meeskonnas läbiviidud küsitluse käigus arvas üks arendajatest, et Kanban'i tahvli kasutamine arenduse käigus aitab tööle kaasa. See muudab kogu töö ülevaatlikumaks ning kuna tegemist ei olnud väga suure projektiga, siis tänu sellele jaksab ka kõiki *task*'e meeles pidada. Teise arendaja arvates aitas Kanban'i tahvli kasutamine samuti arendusele kaasa, kuid seoses sellega, et tegemist ei olnud väga suure projektiga, oleks võinud kasutada Kanban'i tahvli puhul lihtsamat struktuuri, mida oleks kergem arendajate jaoks jälgida.

Töö autori arvamusel aitas töövoo visualiseerimine kaasa sellele, et meeskonnaliikmed said parema ülevaate *task*'idest, millega tuleb tarkvaraarenduses tegeleda. Kuid kindlasti oleks võinud seoses antud projektiga kasutada Kanban'i tahvli puhul lihtsustatud struktuuri. Kuna antud projekti maht polnud väga suur, siis oleks võinud Kanban'i tahvli puhul kasutada neli veergu: *backlog*, tegemisel, testimine ja tehtud.

5.1.6 Meeskond ja rollid

Seoses Kanban'i kasutamisega ei olnud meeskonnas paika pandud rollid ehk kõik liikmed täitsid kõikvõimalikke erinevaid ülesandeid, mida tuli skoobi käigus ära teha. Nagu mainitud ka eelnevas peatükis (vt. 4.4.5 Kanban vs. Scrumban), siis Kanban'i puhul ettemääratud rollid puuduvad.

Üheks probleemiks, mis tekkis meeskonnas, oli see, et mõned *task*'id olid arendajate vahel valesti jagatud. Näiteks arendaja, kellel oli ainult kogemust *front-end*'iga, pidi täitma mahukat *back-end* ülesannet. See muutis tarkvaraarendust aeglasemaks ning arendajale tekitas ebamugavust seoses sellega, et talle oli määratud tema tasemele mitte vastav ülesanne.

Küsitluse käigus arvas üks arendajatest, et *task*'e oleks mõistlik jagada taseme järgi, mis aitaks tarkvaraarendusprotsessi kiirendada. Kuid teise vastaja arvates on *task*'ide jagamine suhteline. Teine arendaja arvas, et kui projektiga on väga kiire ja asjad tuleb

kohe ära teha, siis *task*'ide jagamine taseme järgi on õigustatud, kuid samas selline viis *task*'ide jagamiseks aeglustab meeskonna ning selle liikmete arengut.

Töö autori arvamusel peab meeskonnas olema mingisugune spetsialiseerimine, mille puhul saab spetsiifilisi ülesandeid määrata kindlatele arendajatele, kellel on sellega kogemust. Selline viis aitaks tarkvaraarendusprotsessi kiirendada juba enne probleemide tekkimist. Samas ei tohi arendajatele anda alati ühte tüüpi ülesandeid, kuna selline viis peataks nende arengu.

5.2 Scrumbani rakendamine

5.2.1 Planeerimine

Seoses Scrumban'i kasutamisega oli samuti väga tähtsal kohal tarkvaraarendusega kaasas käiv plaanerimine. E-poe arendamise käigus planeeriti iteratsiooni nii tihti kui võimalik. Kui *task*'e ei saanud enam *backlog*'ist normaalsel kiirusel võtta, siis planeerimine oli prioriteedi järjekorras esimesel kohal.

Nagu ka Kanban'i puhul toimus iteratsioonide planeerimine koos projektijuhiga, kes andis ülevaate uutest *task*'idest ning näitas kuhu ja miks oleks vaja muudatus teostada. Eelnevalt oli projektijuht suhelnud kliendiga, kes avaldas oma soove nõutavate funktsioonalsuste kohta, mida tuleks skoobi jooksul ära teha või muuta. Scrumban'i puhul kohtus projektijuht kliendiga vähemalt korra nädalas. Uueks ideeks oli kogu meeskonna ja kliendi kohtumine, kus ka arendajad said kuulata, mida klient e-poe arendamise kohta arvab. Tänu sellele said kõik meeskonnaliikmed kuulata kliendi soove ning vajadusel kohe läbi arutada, kui oli võimalik neid paremat moodi teostada. Kui Kanban'i ajal muutusid kliendi soovid pidevalt, mille tõttu pidid arendajad e-poe funktsionaalsusi ja disaini tihti muuta, siis Scrumban'i puhul minimeeriti sellega kaasnev risk, et arendajad teevad ära üleliigset tööd.

Scrumban'i ajaks olid meeskonnaliikmed harjunud iteratsioonide planeerimisega ning see ei võtnud enam palju aega.

Läbiviidud küsitluse alusel arvas üks arendajatest, et projektijuhi pidevad kohtumised kliendiga aitasid arendusele kaasa. Tänu sellele sai samuti uusi *task*'e *backlog*'i lisada

ning tema arvates toimis see süsteem samamoodi nagu Kanban'i puhul. Kuid arendaja arvas, et kohtumine kliendiga ei sobinud talle, kuna talle ei tundunud, et see kuulub tema ülesannete alla. Tema arvates on keeruline kliendile tarkvaraarendusega seotud asju selgitada ning sellega peaks tegelema pigem projektijuht, kes seda tööd oskab teha. Teisele meeskonnaliikmele sobis samuti see variant rohkem, mille korral ainult projekijuht kohtus kliendiga. Arendaja arvates ei olnud kohtumine kliendiga tema jaoks kasulik, kuna tema ei oska kliendile asju piisavalt selgeks teha. Samuti arvas ta, et kuna klient ei oska arendajale asju selgitada ning vastupidi, siis jäid paljud asjad pigem arusaamatuks. Kokkuvõtvalt arvas ta, et kliendiga suhtlemine sobib rohkem projektijuhile, kes oskab seda teha.

Töö autori arvamusel aitas planeerimine teha selgemaks, mida klient e-poe arendamise käigus vajas. Samuti oli võimalik paika panna esmatähtsad ülesanded ning nendega esimesel võimalusel tegeleda. Kuna pidevalt tehti ära need *task*'id, mida klient vajas kõige rohkem, siis kohtumised kliendiga olid edukad ning klient oli rahul hetkeseisuga. Meeskonna liikmetele ei sobinud kohtumised kliendiga, kuna nende jaoks oli raske kliendile asju selgeks teha oma vaatepunktist ning seetõttu ei olnud see meeskonnale kasulik.

5.2.2 Task

Scrumban'i kasutamisega ei hakatud *task*'i suuruse määramise põhimõtet muutma võrreldes Kanban'i agiilse tarkvaraarenduse metoodikaga. *Task*'i suurus ei ületanud 2 päeva ning tavaliselt oli selle mahuks maksimaalselt 12 tundi. Kuna Scrumban'i metoodika puhul ei ole määratud *task*'i maksimaalset suurust, siis vajadusel on seda võimalik ka muuta.

Samuti Scrumban'i kasutamisega ei hakatud *task*'i puhul määrama kindlat inimest, kes peaks sellega tegelema hakkama. Scrumban'i metoodika puhul ei määra projektijuht või meeskonnajuht ülesandeid kindlale isikule, vaid iga meeskonnaliige valib ise, mis *task*'iga ta hakkab parasjagu tegelema.

Tänu Scrumban'i metoodikale hakati *task*'idele määrama nende prioriteeti. Kui enne Scrumban'i määrati *task*'idele prioriteet haruharva, siis nüüdsest määrati seda igale

ülesandele. *Task*'ide prioriteeti määrati vahemikus 1-5, kus esimene tähendas kõige olulisemat ülesannet, mida tuleks arendamise käigus täita.

Läbiviidud küsitluse käigus arvasid mõlemad arendajad, et *task*'idele prioriteedi määramine oli arendusele vajalik, kuid üks arendajatest mainis, et prioriteedi määramist ei ole vaja viie tasemel vahel jagada. Mõlema arendaja jaoks oli huvitav kogemus valida ise nimekirjast *task*'e, millega nad soovivad tegeleda. Üks arendajatest arvas, et selline viis sobis talle rohkem kui Kanban'i puhul tegija nime määramine. Peamiseks põhjuseks on see, et arendaja teab oma võimeid kõige paremini ning seoses sellega saab valida nimekirjast antud olukorrale õiged *task*'id.

Töö autori arvates on selline *task*'i suurus kasulik tarkvaraarenduse käigus, kuna niimoodi on võimalik säilitada pidev töövoog ning efektiivne arendamine. Samuti oli meeskonna jaoks kasulik see, et igaüks sai ise valida tööülesandeid, millega järgnevalt tegeleda. See andis kõigile meeskonnaliikmetele valikuvõimalust, mis *task*'iga hakata edaspidi tegelema. *Task*'idele prioriteedi määramine oli kasulik, kuid seda oleks võimalik teha veelgi rohkem lihtsustatud viisil.

5.2.3 „Pull" põhimõte

Nii nagu ka Kanban'i kasutamise ajal oli ka Scrumban'iga seoses üheks põhiliseks põhimõtteks, mida kasutati, „Pull" põhimõte. Nagu mainitud eelnevates peatükis (vt. 4.2 Kanban ja 4.3 Scrumban) on selle peamisteks ülesanneteks raiskamiste vältimine ja töös olevate ülesannete hulga piiramine.

Ka Scrumban'i puhul oli „Pull" põhimõte meeskonna jaoks tähtis. Scrumban'i kasutamise ajal ei jälgitud küll seda, kui palju *task*'e oli igale arendajale määratud, kuid tähtis oli see, et igal arendajal oleks võrdne arv prioriteedi järjekorras kõige tähtsamaid ülesandeid. Tähtis oli vanad ülesanded enne ära lahendada, kui uued töösse võtta. See vähendab poolenisti tehtud *task*'ide olemasolu ning suurendab meeskonna keskendumist. Samuti oli tähtis, et arvestades kõikide arendajate *task*'ide arvu summat, ei ületaks see ülempiiri, milleks oli 12 ehk kolm *task*'i arendaja kohta. Tänu sellele ei koormatud arendajaid üle ning alati toimis meeskonnas võrdne tööjaotus.

Läbiviidud küsitluse arvasid mõlemad arendajad, et *task*'ide jaotus meeskonna vahel oli jaotunud võrdselt. Esimesele vastajale meeldis, et üks arendaja võis tegeleda korraga maksimaalselt 3 *task*'iga, mis aitas tal keskenduda antud ülesannete lahendamisele. Samuti oli sobilik kogu meeskonna vahel määrata maksimaalne arendusmaht, mida ei tohtinud tarkvaraarenduse käigus ületada. Teine vastaja arvas aga, et kuna tegemist ei olnud suure projektiga, siis oleks võinud arendusmahtu isegi suurendada. Arendaja arvates ei olnud projekti *task*'id niivõrd mahukad, et panna tarkvaraarendusele suhteliselt madal arendusmaht. Mõlema meeskonnaliikme arvates oli kasulik ära teha olemasolevad ülesanded enne, kui võtta uued töösse.

Töö koostaja arvamusel aitas „Pull" põhimõtte kasutamine kaasa sellele, et ei tekkinud arenduse käigus sellist olukorda, kus tarkvaraarendajat koormati üleliigsete või ebavajalikke ülesannetega. Sellised mõtted vastavad ka Agiilse tarkvaraarenduse manifesti põhimõtetes(vt. 3.3 Agiilse tarkvaraarenduse manifesti põhimõtted). Kuid meeskonnas oli liikmeid, kes arvasid, et arendusmahtu oleks võinud suurendada, kuna kohati oli koormus liiga väike.

5.2.4 Retrospektiiv

Seoses Scrumban'i kasutuselevõtuga hakati kasutama retrospektiivi teostamist. Kõik retrospektiivid korraldati Tallinna kontoris viie liikmega, kes osalesid e-poe arenduses. Retrospektiivide korraldamine toimus nädalas korra, et meeskonnaliikmed saaksid omavahel tihedamalt suhelda ning mõtteid jagada. Retrospektiivide korraldamisaeg tuli eelnevalt palju varem planeerida, kuna arenduses osalesid üliõpilased, kes õppisid erinevate õppekavade järgi. Seetõttu oli tähtis pidada retrospektiiv just siis, kui kogu meeskond asus kontoris. Retrospektiivide planeerimiseks kasutati internetis saadaval tarkvara kalendriga, kus sai kõik vajalikud ajad kirja panna ning kus kõikidel meeskonnaliikmetel oli ligipääs olemas.

Retrospektiivi peamiseks ülesandeks oli saada vastust kolmele põhilisele küsimusele: mis läks hästi, mida saaks paremini teha ning mida oleks vaja muuta. Juba esimeste retrospektiivide puhul tuli välja mitu probleemi, millele kohe üritati leida mingi kindel lahendus. Projektijuht proovis leida meeskonnaliikmetega lahendusi kõikidele probleemidele. Iga retrospektiiviga tuli aina uusi ettepanekuid, mis aitasid tarkvaraarendust muuta veelgi efektiivsemaks.

Läbiviidud küsitluse käigus arvas üks meeskonnaliikmetest, et retrospektiivide korraldamine oli meeskonnale kasulik. Tema arvates aitas see leida lahendusi erinevatele probleemidele ning muuta suhtlus meeskonnasiseselt aktiivsemaks. Samuti arvas arendaja, et retrospektiivide korraldamine aitas kaudselt kaasa meeskonna koostööle. Teine arendaja arvas, et retrospektiivid olid küll kasulikud, kuid mitte vajalikud antud projekti käigus.

Töö autori arvates aitas retrospektiivi kasutuselevõtt meeskonnale palju kaasa ning muutis probleemide lahendamise palju efektiivsemaks. Samuti aitas see luua meeldivat keskkonda meeskonnasiseselt, mida peetakse ka tähtsaks eelnevates peatükkides (vt.3.2 Agiilse tarkvaraarenduse manifest ja vt. 3.3 Agiilse tarkvaraarenduse manifesti põhimõtted), kus peetakse inimesi ja nendevahelist suhtlust rohkem oluliseks kui protsesse ja arendusvahendeid. Töö autori hinnagul aitas retrospektiivide kasutamine viia tarkvaraarenduse uuele tasemele, kus probleeme jagati omavahel tihedamini, mis aitas leida kindlaid lahendusi. Kindlasti võib öelda, et seoses projekti mahuga polnud retrospektiivide korraldamine otseselt vajalik, kuid sellegipoolest oli see väga kasulik.

5.2.5 Töövoo visualiseerimine

Scrumbani kasutamisega ei visualiseeritud *task*'e küll füüsilisel tahvlil, kuid selle asemel kasutati internetis olemasolevat tarkvara, mis täitis Kanban'i tahvli funktsionaalsusi. Nagu mainitud ka eelnevas peatükis (vt 4.2 Kanban) visuaalse mudeliga kaasneb töövoo üle parem ülevaade, samuti aitab see muuta suhtlemist ja koostööd efektiivsemaks.

Scrumban'i kasutamise ajal kasutati sama tarkvara, mis Kanban'i rakendamisel, ning meeskond oli harjunud Kanban'i tahvliga ning sellega pidev tegelemine ei häirinud meeskonnaliikmete tööd ega keskendumist.

Nagu ka Kanban'iga seoud küsitluse käigus arvas üks arendajatest, et ka Scrumban'i puhul oleks võinud kasutada tabeli lihtsamat struktuuri, mis oleks muutnud arendajate töö lihtsamaks. Samuti oleks arendajal alati parem ülevaade tarkvaraarenduse protsessist. Mõlemad vastajad ei muutnud oma meelt ka selle suhtes, et tahvli kasutamine aitab muuta arendust efektiivsemaks.

Töö autori arvamusel aitas töövoo visualiseerimine kaasa sellele, et meeskonnaliikmed said parema ülevaate *task*'idest, millega tuleb tarkvaraarenduses tegeleda. Samas, antud projekti käigus oleks kasulikum olnud kasutada lihtsustatud strukuuri tahvli puhul.

5.2.6 Meeskond ja rollid

Seoses Scrumbani'i kasutamisega jagati meeskonnas liikmed *task*'ide eelistuse järgi. Ühed liikmed tegelesid *front-end* ülesannetega, samal ajal kui teised *back-end'iga*. Projektijuht kirjutas ülesse erinevad *task*'id, mis jagunesid kahte valdkonda: *front-end* ja *back-end*. Iga meeskonnaliige teadis oma eelistust ning valis oma valdkonnast vastavad ülesanded, mida töösse parasjagu võtab.

Meeskonnas oli kaks arendajat, kes enamasti eelistasid *back-end* ülesandeid, ning kaks arendajat, kes pigem tegelesid *front-end task*'idega. Arendajad võtsid ülesandeid sealt valdkonnast, mida nad eelistasid. Kuid kindla reeglina see ei kujunenud ning vahepeal valisid arendajad ülesandeid ka teisest valdkonnast, et muuta töö mitmekesisemaks.

Läbiviidud küsitluse käigus mainisid mõlemad arendaja, et ülesannete jagamine kaheks valdkonnaks oli arendusele vajalik.

Töö autori arvates oli meeskonnale kasulik selline spetsialiseerumine, kus ülesanded jagati kahte valdkonda ning arendajad olid jaotatud nende valdkondade vahel. Samuti oli kasulik see, et meeskonnas oli lubatud erandid, kus *back-end*'iga tegelev arendaja võttis töösse vahepeal *front-end*'iga seotuid ülesandeid.
Järgnevas peatükis esitatakse meeskonna arendamise analüüsi tulem ja soovitused, mida jälgida meeskonna arendamisel.

5.3 Järeldused

Järgnevas peatükis esitatakse meeskonna arendamise analüüsi tulem vastavalt kindla praktika kasutamisele.

5.3.1 Kanbani analüüsi kokkuvõte

Analüüsin Kanbani'i praktikate kasutamist kokkuvõtvalt ning toon välja võimalikke parenduskohti.

Planeerimine:

- Oli Kanban'i rakendamise ajal tähtsal kohal (pidev iteratsioonide planeerimine, projektijuhi kohtumised kliendiga).
- Esimese vastaja arvates oli planeerimine meeskonnale nii **vajalik** kui ka **kasulik**.
- Teise vastaja arvates oli planeerimine samuti meeskonnale nii **vajalik** kui ka **kasulik**.

Task:

- Nõutavad funktsionaalsused jagati *task*'ideks ning neile oli määratud kindel maht. Samuti määrati iga *task*'i puhul selle tegija nime.
- Esimese vastaja jaoks oli arenduse jagamine *task*'ideks **vajalik**. Kuid mainis, et *task*'e oleks vajalik jagada prioriteedi järgi, et tegeleda algul esmatähtsate ülesannetega.
- Teise vastajate arvates oli samuti arenduse jagamine *task*'ideks **vajalik**, kuid arvas, et *task*'ide maht erines ning kõikus suures vahemikus.

„Pull" põhimõte:

- Oli Kanban'i rakendamise ajal põhiliseks põhimõtteks tänu millele välditi raiskamisi ja piirati töös olevate ülesannete hulka.
- Esimese vastaja arvates oli „Pull" põhimõtte kasutamine nii **vajalik** kui ka **kasulik**. Arendaja pidas kasulikuks ära teha olemasolevad ülesanded enne, kui võtta töösse uued.
- Teise vastaja arvates oli „Pull" põhimõtte kasutamine samuti meeskonnale **vajalik**, kuid. Arendaja pidas kasulikuks ära teha olemasolevad ülesanded enne, kui võtta töösse uued.

Retrospektiiv:

- Ei rakendatud Kanban'i kasutamise ajal.

- Esimese vastaja jaoks oleks retrospektiivide korraldamine olnud meeskonnale nii vajalik kui ka kasulik.

- Teine vastaja arvas, et seoses antud projekti mahuga poleks retrospektiivide korraldamine vajalik. Kuid mainis, et retrospektiivide asemel koosolekute korraldamine meeskonnasiseselt oleks ikkagi kasulik.

Töövoo visualiseerinine:

- Kasutati internetis olemasolevat tarkvara, kus visualiseeriti *task*'e.

- Esimese vastaja arvates oli Kanban'i tahvli kasutamine nii **vajalik** kui ka **kasulik**, kuna see muutis kogu töö ülevaatlikumaks.

- Teise vastaja arvates oli Kanban'i tahvli kasutamine arenduse käigus **vajalik**, kuid kasulikum oleks kasutada Kanban'i tahvli lihtsustatud struktuuri.

Meeskond ja rollid:

- Meeskonnas ei olnud rolle, samuti puudus spetsialiseerumine.

- Esimese vastaja arvates oleks olnud vajalik *task*'e jagada taseme järgi, et kiirendada arendusprotsessi.

- Teise vastaja arvates ei oleks olnud vajalik *task*'ide jagamine taseme järgi, kuna see aeglustab meeskonna ning selle liikmete arengut. Kuid mainis, et *task*'ide jagamine taseme järgi on kasulik seljuhul, kui tarkvaraarendusega on kiire.

Tabel 9 Kanban'i analüüsi kokkuvõtte tabel

	Vastaja 1		**Vastaja 2**	
	Vajalikkus	Kasulikkus	Vajalikkus	Kasulikkus
Planeerimine	Jah	Jah	Jah	Jah
Task	Jah	Ei	Jah	Ei

„Pull“ põhimõte	Jah	Jah	Jah	Ei
Retrospektiiv	-	-	-	-
Töövoo visualiseerimine	Jah	Jah	Jah	Ei
Meeskond ja rollid	-	-	-	-

Nagu näha tabelis, pidasid mõlemad vastajad kõik Kanban’i puhul rakendatud praktikad vajalikuks. Kasulikuks pidasid mõlemad vastajad ainult ühe praktika rakendamist. Peamiseks põhjuseks oli see, et vastajate arvates oli praktikaid võimalik muuta veelgi kasulikumaks antud projekti puhul. Kanban’iga seotud probleemide lahendamiseks juurutati välja uus praktika, Scrumban, mille eesmärgiks oli töötajate rahulolu ja meeskonna koostöövõime parandamine.

Võimalikud parenduskohad:

- Retrospektiivide korraldamine aitaks parandada kesist suhtlemist ja koostööd meeskonnaliikmete vahel. Selleks oleks kasulik rakendada Scrumban’i.
- Tuleks lisada *task*’idele prioriteet, et näha, millised on tähtsamad ülesanded arenduses. Selleks oleks kasulik rakendada Scrumban’i.
- Kasutada lihtsustatud Kanban’i tahvli varianti, et oleks asju lihtsam jälgida.
- Meeskonnasiseselt tuleks paika panna spetsialiseerumine, et kiirendada arendusprotsessi. Selleks oleks kasulik rakendada Scrumban’i.

5.3.2 Scrumbani analüüsi kokkuvõte

Analüüsin Scrumban’i praktikate kasutamist kokkuvõtvalt ning toon välja võimalikke parenduskohti.

Planeerimine:

- Oli Scrumban’i rakendamise ajal tähtsal kohal. Pidev iteratsioonide planeerimine, projektijuhi ja meeskonna kohtumised kliendiga.

- Esimese vastaja arvates oli planeerimine meeskonnale **vajalik**. Kasulikuks pidas ta projektijuhi kohtumisi kliendiga, kuid arvas, et meeskonna kohtumine kliendiga ei andnud midagi juurde.

- Teise vastaja arvates oli planeerimine samuti meeskonnale **vajalik**, kuid arvas, et arendajate kohtumine kliendiga ei olnud kasulik, kuna klient ei oska arendajale asku piisavalt selgeks teha ning vastupidi.

Task:

- Nõutavad funktsionaalsused jagati *task*'ideks ning neile oli määratud kindel maht. *Task*'ide puhul määrati neile prioriteet ning ei määratud tegijat.

- Esimese vastaja jaoks oli arenduse jagamine *task*'ideks **vajalik**. Esimene meeskonnaliige arvas, et *task*'e ei ole mõistlik jagada viie taseme vahel.

- Teise vastajate arvates oli samuti arenduse jagamine *task*'ideks **vajalik** kui **kasulik**.

„Pull" põhimõte:

- Scrumban'i rakendamise ajal üheks põhiliseks põhimõtteks oli „Pull" põhimõte. Sellega välditi raiskamisi ja piirati töös olevate ülesannete hulka.

- Esimese vastaja arvates oli „Pull" põhimõtte kasutamine meeskonnale nii **vajalik** kui ka **kasulik**.

- Teise vastaja arvates oli „Pull" põhimõtte kasutamine samuti meeskonnale **vajalik**, kuid arvas, et arendusmahtu oleks võinud isegi suurendada.

Retrospektiiv:

- Scrumban'i kasutamise ajal teostati retrospektiive.

- Esimese vastaja jaoks oli retrospektiivide korraldamine nii **vajalik** kui ka **kasulik**.

- Teise vastajate arvates oli retrospektiivide korraldamine meeskonna jaoks **kasulik**, kuid mitte vajalik antud projekti käigus.

Töövoo visualiseerinine:

- Kasutati internetis olemasolevat tarkvara, kus visualiseeriti *task*'e.
- Esimese vastaja arvates oli Kanban'i tahvli kasutamine nii **vajalik** kui ka **kasulik**, kuna see muutis kogu töö ülevaatlikumaks.
- Teise vastaja arvates oli Kanban'i tahvli kasutamine arenduse käigus **vajalik**, kuid kasulikum oleks kasutada Kanban'i tahvli lihtsustatud struktuuri.

Meeskond ja rollid:

- Meeskonnaliikmed olid jagatud kahte gruppi *task*'ide eelistuse järgi (*front-end* ja *back-end*).
- Esimese vastaja jaoks oli *task*'ide jagamine eelistuse järgi nii **vajalik** kui **kasulik**, kus ühed arendajad tegelesid *front-end* ja teised *back-end* ülesannetega.
- Teise vastaja jaoks oli *task*'ide jagamine eelistuse järgi samuti nii **vajalik** kui **kasulik**, kus ühed arendajad tegelesid *front-end* ja teised *back-end* ülesannetega.

Tabel 10 Scrumban'i analüüsi kokkuvõtte tabel

	Vastaja 1		**Vastaja 2**	
	Vajalikkus	Kasulikkus	Vajalikkus	Kasulikkus
Planeerimine	Jah	Ei	Jah	Ei
Task	Jah	Ei	Jah	Jah
„Pull" põhimõte	Jah	Jah	Jah	Ei
Retrospektiiv	Jah	Jah	Ei	Jah
Töövoo visualiseerimine	Jah	Jah	Jah	Ei
Meeskond ja rollid	Jah	Jah	Jah	Jah

Nagu näha tabelis, pidasid mõlemad vastajad peaaegu kõik Scrumban'i puhul rakendatud praktikad vajalikuks. Teine vastaja arvas, et retrospektiivide korraldamine polnud arenduse käigus vajalik. Kasulikuks pidasid mõlemad vastajad kahe praktika rakendamist. Kokkuvõtvalt pidasid vastajad Scrumban'i puhul rakendatud praktikaid kasulikumaks kui Kanban'i puhul ning parenduste abil on seda võimalik muuta veelgi paremaks.

Võimalikud parenduskohad:

- Lõpetada kohtumised arendajate ja kliendi vahel ning korraldada neid ainult vajadusel. Need kohtumised ei tundunud arendajate jaoks kasulikud, kuna ei osatud kliendiga suhelda.
- Kasutada lihtsustatud Kanban'i tahvli varianti, et oleks asju lihtsam jälgida.
- Retrospektiivi asemel lühikese koosoleku korraldamine iga päev, et vahetada aktiivselt informaatsioni meeskonnaliikmete vahel.

6. Kokkuvõte

Lõputöö eesmärgiks oli analüüsida ja võrrelda IT-Projekt OÜ arendamisprotsessi, mis oli ülesehitatud agiilse praktika alusel ning selle alusel välja pakkuda ning juurutada uue praktika, mis paremini sobiks ja lahendaks olemasolevad probleemid võimaldades sujuvama arendamisprotsessi ning parandaks töötajate rahulolu ning meeskonna koostöövõimet. Edasine eesmärk oli analüüsida praktikate kasutamist ning teha nende kohta järeldusi, samuti pakkuda välja parenduskohti. Kõik lõputööle püstitatud eesmärgid saavutati täies ulatuses.

Analüüsides IT-Projekt OÜ arendamisprotsessi Kanban'i rakendamise puhul, leidis autor mitmeid puudujääke, mida võiks teise agiilse tarkvaraarendus metoodika abil parendada, mille abil saaks muuta töötajate rahulolu ja meeeskonna koostöövõimet paremaks. Samuti leidis autor analüüsi käigus mitmeid häid lahendusi ning ideid meeskonnaliikmetelt, kuidas võiks parendustega arendamisprotsessi efektiivsemaks muuta, tänu millele tundus kasulik edasi projekti arendamise käigus rakendada Scrumban'i metoodikat. Scrumban'i rakendamise käigus kasutati ära neid samu häid lahendusi ja ideid arenduses, et muuta tarkvaraarendusprotsessi veelgi efektiivsemaks.

Mõlema agiilse arendusmeetodi puhul leiti mitmeid võimalikke parenduskohti. Kanban'i rakendamise käigus järeldati, et retrospektiivide korraldamine aitaks parandada kesist suhtlemist ja koostööd meeskonnaliikmete vahele, samuti tuleks *task*'idele lisada prioriteet, et näha, millised on tähtsamad ülesanded arenduses, ning meeskonnasiseselt tuleks paika panna spetsialiseerumine, et kiirendada arendusprotsessi. Nende parenduskohtade elluviimiseks rakendati Scrumban'i metoodikat antud projekti käigus.

Kokkuvõtvalt töös, mille eesmärgiks oli analüüsida ja võrrelda ettevõtte arendamisprotsessi, mis oli ülesehitatud agiilse praktika alusel ning selle alusel välja pakkuda ning juurutada uue praktika, mis paremini sobiks ja lahendaks olemasolevad probleemid võimaldades sujuvama arendamisprotsessi, järeldati, et üleminek Kanban'ilr Scrumban'ile vaadeldaval perioodil suurendas töötajate rahulolu ning meeskonna koostöövõimet.

Kasutatud kirjandus

[1] A. Manifesto, “Agiilse tarkvaraarenduse manifest”, [WWW]. Available: http://agilemanifesto.org/iso/et/ (25.04.2016)

[2] A. Manifesto, „Agiilse tarkvaraarenduse manifesti põhimõtted“, [WWW]. Available: http://agilemanifesto.org/iso/et/principles.html (25.04.2016)

[3] “IT-Projekt OÜ“ [WWW]. Available: http://itprojekt.ee (14.05.2016)

[4] I. Sommerville, „Agile Software Development“, [WWW]. Available: http://www.uio.no/studier/emner/matnat/ifi/INF1050/v14/timeplan/sommerville.ch_03_agile_sw_dev.pdf (14.05.2016)

[5] M. Cohn, „Scrum“, [WWW]. Available: http://www.mountaingoatsoftware.com/agile/scrum (14.05.2016)

[6] M. Cohn, „Product Owner“, [WWW]. Available: https://www.mountaingoatsoftware.com/agile/scrum/product-owner (14.05.2016)

[7] M. Cohn, „Scrum Master“, [WWW]. Available: https://www.mountaingoatsoftware.com/agile/scrum/scrummaster (14.05.2016)

[8] “Scrum (software development)“, [WWW]. Available: https://en.wikipedia.org/wiki/Scrum_(software_development) (14.05.2016)

[9] “Why Scrum“, [WWW]. Available: https://www.scrumalliance.org/why-scrum (14.05.2016)

[10] D. Radigan, „A Brief introduction to kanban“, [WWW]. Available: https://www.atlassian.com/agile/kanban (14.05.2016)

[11] “What is Kanban?“, [WWW]. Available: http://leankit.com/learn/kanban/what-is-kanban (14.05.2016)

[12] J. Siilivask, „Agiilsete metoodikate rakendamine IT töörkooraldussüsteemi loomisel“, [WWW]. Available: www.cs.tlu.ee/teemad/get_file.php?id=281 (14.05.2016)

[13] “Scrumban: A Hybrid Framework to Get More Out of Your Agile SDLC“, [WWW]. Available: https://www.veracode.com/blog/2015/01/scrumban-hybrid-framework-get-more-out-your-agile-sdlc (14.05.2016)

[14] N. Nikitina, M. Kajko-Mattsson, „From Scrum to Scrumban: A Case Study of a Process Transition“, [WWW]. Available: https://www.researchgate.net/publication/254041684_From_Scrum_to_Scrumban_a_case_study_of_a_process_transition (19.05.2016)

[15] “Getting Started With Scrumban“, [WWW]. Available: http://www.aboutscrumban.com/how-to-start-using-scrumban/ (14.05.2016)

[16] “Scrum vs. Kanban vs. Scrumban”, [WWW]. Available: http://www.eylean.com/Publications/DownloadPublication/4e93cecc-4cb2-4e3f-849d-810d7aea33a5?name=Whitepaper---Scrum-vs-Kanban-vs-Scrumban (14.05.2016)

[17] “Whitepaper: Kanban vs Scrum“, [WWW]. Available:

http://www.belatrixsf.com/whitepapers/kanban-vs-scrum/ (14.05.2016)

[18] “A.Thangaraj, Scrumban – Being Differently Agile“, [WWW]. Available:

http://www.agilerecord.com/scrumban-%E2%80%92-differently-agile/ (14.05.2016)

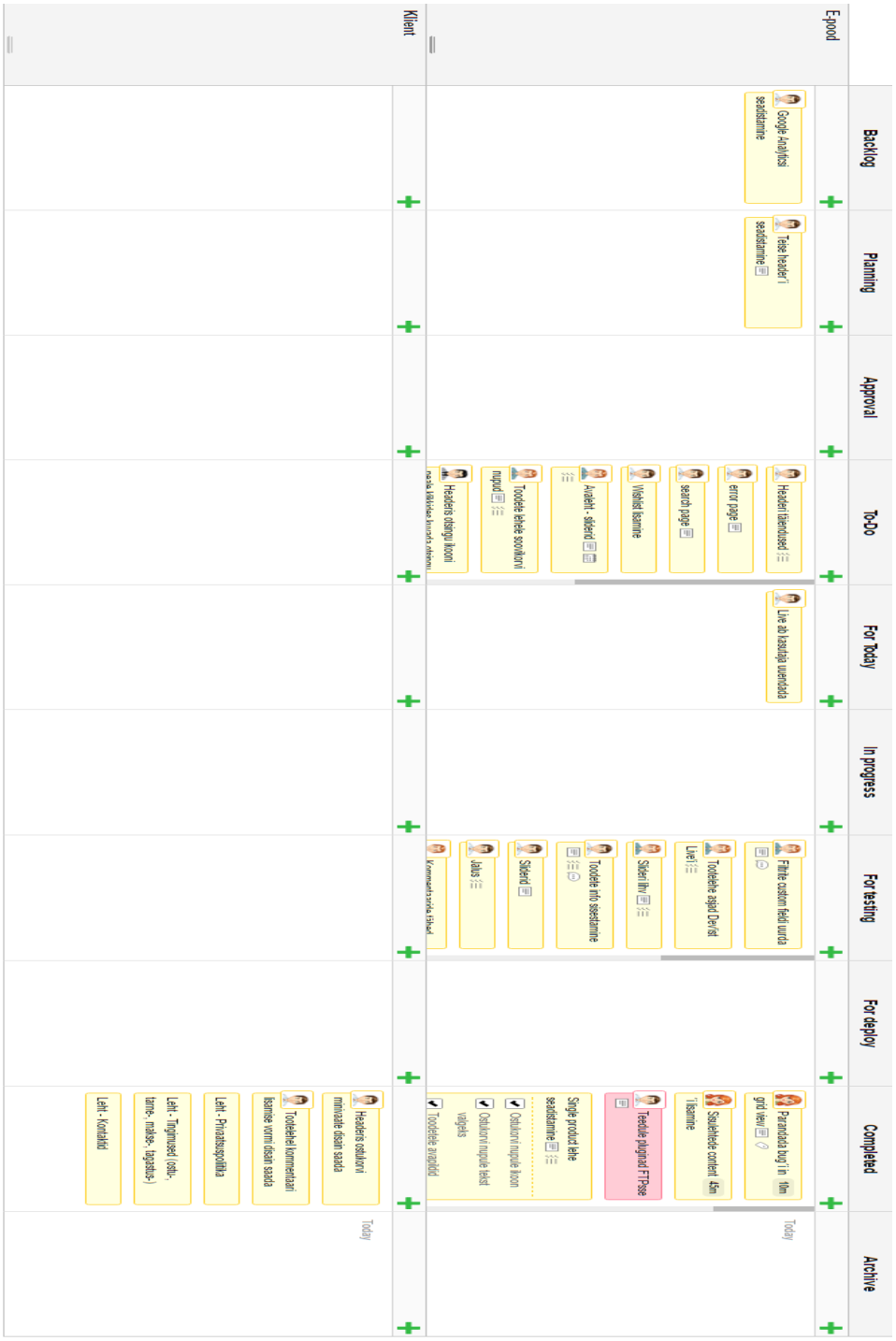
[19] Agile software development [WWW]. Available:

https://en.wikipedia.org/wiki/Agile_software_development (14.05.2016)

[20] E.Muuga, “Agiilsete praktikate rakendamine IT arenduse idufirmades”, [WWW].

Available: http://digi.lib.ttu.ee/i/?2235 (14.05.2016)

Lisa 1 – Kanban’i töövoo visualiseerimine



Joonis 4 Kanban'i töövoo visualiseerimine