TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Agiilsete praktikate rakendamine IT arenduse idufirmades

Bakalaureusetöö

Üliõpilane: Erkki Muuga
Üliõpilaskood: 112764IABB
Juhendaja: lektor Karin Rava

Tallinn
2015

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.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Töö eesmärk on analüüsida kahe idufirma toote arendusprotsessi ja uurida agiilsete praktikate rakendamist nendes kolme enimtuntud arendusmetoodika (Scrum, ekstreemprogrammeerimine ja Kanban) põhjal.

Töö olulisemad käsitletavad probleemid on, et kas teatud praktika on kasutuses ja kas selle praktika kasutamine on kasulik arvestades ettevõtte suurust, ärilist poolt ja arengufaasi.

Kummagi idufirma toote arendusprotsessi analüüsimisel tulid välja parenduskohad, mille rakendamisel saaks arendusprotsessi efektiivsemaks muuta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 46 leheküljel, 6 peatükki, 6 joonist, 8 tabelit.

Abstract

The aim of the work is to analyse the software development process of two start-up companies and compare it to the agile software development methodology practices.

The main problems of the work is to finding out if each agile practice is being applied in the software development process and if it's justified taking into account the start-up company's size, and business logic.

In both start-up companies' software development process, there were two improvement points forund. Applying these improvement points should make the software development process more effective.

The thesis is in Estonian and contains 46 pages of text, 6 chapters, 6 figures, 8 tables.

Lühendite ja mõistete sõnastik

Idufirma ***start-up***

Ettevõte, mille tegevust iseloomustab määramatus, mis väljendub selles, et ettevõtte toodet või teenust ei ole veel äriliselt tõestatud, ärimudel rajaneb oletustel, on pidevas aktiivses arengus ning võib väga oluliselt muutuda arendusprotsessi käigus. Tootearendusprotsessis otsitakse tootele või teenusele globaalselt turupotentsiaali, mistõttu peab vastava toote või teenuse järele olema suur nõudlus või nähakse ette nõudluse hüppelist kasvu lähiajal. [1]

Praktika ***practice***

(Tarkvaraarenduse) praktika on toote arenduses ühe kindla tegevuse kord või reegel, mida arendusmeeskond järgib. Selle eesmärk on saavutada tarkvara toote arenduses efektiivsem tulemus rahalises või ajalises mõttes.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Idufirmade arv ja agiilsete arendusmetoodikate praktikate rakendamine on kasvavas trendis. Lõputöös uurin, kui palju need omavahel seotud on.

1.1 Taust ja probleem

Idufirmad peavad olema väga suure kohanemisvõimega, seega nende arendusprotsess peab olema samuti väga kohanemisvõimeline.
Töö on vajalik, et leida idufirmade toote arendusprotsessis võimalusi agiilsete arendusmetoodikate praktikate paremaks rakendamiseks.

Töö on vajalik alustavatele idufirmadele, kes tahavad oma toote arendusprotsesse välja töötada.

Lõputöö on koostatud 2014. aasta sügisel. Töö tegemisel on võetud aluseks kaks idufirmat, mille toote arendusprotsesse uuritakse ning võrreldakse agiilsete arendusmetoodikate praktikatega.

1.2 Ülesande püstitus

Lõputöö eesmärgid on:

- Eesmärk 1: Uurida, kas analüüsitavates idufirmades kasutatakse agiilsete arendusmetoodikate praktikaid.
- Eesmärk 2: Uurida, kas ja kuidas on arendusmetoodika praktika kasutamine idufirmas põhjendatud.
- Eesmärk 3: Uurida, millise agiilse arendusmetoodikaga idufirma toote arendusprotsess kõige enam sarnaneb.
- Eesmärk 4: Leida iga idufirma kohta 1-3 parenduskohta, kuidas saaks agiilsete arendusmetoodikate praktikaid rakendada, et idufirma toote arendusprotsess oleks efektiivsem.

1.3 Metoodika

Eesmärkideni 1 ja 2 jõuan intervjuude käigus vastavate idufirmadega.

Eesmärkideni 3 ja 4 jõuan analüüsi tulemusena.

1.4 Ülevaade tööst

Kõigepealt annan ülevaate idufirmadest Fleep ja RentMarket. Seejärel kirjeldan agiilsete arendusmetoodikate Scrum, Kanban ja Ekstreemprogrammeerimise praktikaid. Siis analüüsin iga nende praktikate kasutamist käsitletavates idufirmades ning ka seda, kas iga praktika kasutamine on põhjendatud. Selle alusel teen kokkuvõtte, kus teen üldistusi ja järeldusi.

2. Idufirmade kirjeldus

Alljärgnevalt kirjeldan kahte idufirmat, milleks on RentMarket ja Fleep. Nende mõlema ühine joon on see, et nad on alustavad ettevõtted. Kuigi üks neist on nii rahaliselt kui ka tegutsemise aja suhtes toorem kui teine. Analüüsin idufirma ärilist poolt üldiselt, üldisemalt toote arendusprotsesse ja meeskonda ning meeskonnasiseseid rolle.

2.1 RentMarket

Ettevõte alustas oma tegevust 2014. aasta jaanuaris. Selle eesmärk on propageerida rentimist ostmise asemel ning teha rentimine võimalikult lihtsaks ja kättesaadavaks.
Hetkel on valmis toode, mis koondab kokku rendikontaktid erinevatest valdkondadest. See toode täidab universaalse rendi-telefoniraamatu eesmärki.
Google't kasutades on väga ajamahukas leida enda jaoks vajalikku teavet, kuna info on eri lehekülgedel erinevalt paigutatud ja tihti puudulik.
RentMarket võimaldab võrrelda erinevate rendipunktide infot omavahel standardselt kujul ja väga hõlpsasti.
Rakendusega on võimalik näha, mis on kõige lähem avatud olev asukoht, kust saab mingit eset rentida, sinna helistada või kaardi peal kuvada marsruuti sinna jõudmiseks.

Arenduse poole pealt suurim projekt on Androidi mobiiliaplikatsiooni välja arendamine.
Mobiiliaplikatsioon kujutab endast kaardirakendust, mis koondab kokku rendikontaktid erinevatest rendivaldkondadest.
Suuruselt teine arenduse projekt on *backoffice*'i välja arendamine, mille kaudu käib andmebaasi rendipunktide info sisestamine ja haldamine.
Valmis on arendatud kasutajate lisamise, klientide lisamise ja rendipunktide lisamise funktsioonid.

RentMarketi kasutaja on inimene, kes soovib mingit konkreetset eset rentida. Ta soovib kiirelt leida enda lähedal olevate vastavate rendiettevõtete kontakte.
Kasutajad pole veel ülemäära palju. On olnud 1000 mobiiliaplikatsiooni allalaadimist ning hetkel on nendest umbes 20 päevast kasutajat.

2.1.1 Meeskond

Kokku on RentMarketis 5 töötajat:

- Tegevjuht – Strateegilise suuna paika panija. Strateegilise suuna järgi kujuneb see, millised funktsionaalsused arendusse võetakse ja millal.
- Projektijuht – Arenduse juhtija. Täidab ka analüütiku ja testija rolli. Koostab lähteülesande, koordineerib selle valmimist ning testib tulemuse vastavust lähteülesandele.
- Veebi arendaja – Arendab *backoffice* keskkonda ning annab nõu tehnlise poolte pealt. Esmane tagasiside funktsionaalsuste paika panemisel ning mahuhinnangu määramisel.
- Android arendaja – Vabakutseline mobiiliaplikatsiooni arendaja, kes töötab etapipõhiselt.
- Disainer – Tunnipõhise palgaarvestusega vabakutseline disainer. Suures osas kasutajaliidese välja töötaja, samas tema pädevusse kuulub ka kasutajakogemuse analüüsimine.

2.1.2 Planeerimine

Ettevõttes toimub nii pikaajaline kui ka lühiajaline planeerimine. Analüüsin kumbagi eraldi.

2.1.2.1 Pikaajaline planeerimine

Kuus korra vaatavad tegevjuht, projektijuht ja arendaja üle pikaajalise strateegilise suuna. See toimub umbes 2-tunnisel koosolekul, kus arutatakse suures plaanis edasist suunda ja kas praegune kurss ühtib soovitud suunaga. Koosolekut juhib tegevjuht ning jagab mõtteid ja viimaseid sündmusi müügivallas. Nii tihe kokkusaamine on vajalik kahel põhjusel:

- Ärisuund ei ole veel 100% välja kujunenud
- Arendaja saab rohkem teada väljapoole suunatud suhtlusest

Kui on mõned arendusega seotud murekohad, siis arutatakse ka neid.

2.1.2.2 Lühiajaline planeeriine

Iga nädal saavad kokku tegevjuht ja projektijuht operatiivsel koosolekul, mida juhib projektijuht. Koosolekul arutatakse arenduse käiku, prioritiseeritakse funktsionaalsusi ja võetakse kokku hetkese arenduse seis.

2.1.3 Arendusprotsess

Kirjeldan üldiselt arendusprotsessi nummerdatult tegevuste toimumise järjekorras. Detailsem protsessi kirjeldus on protsessijoonises lisas 1.

2.1.3.1 Üldine veebi arendusprotsessi määratlus

1. Operatiivsetel koosolekutel lepitakse kokku, milliseid funktsionaalsuste blokke tahetakse lähiajal arendada.
2. Projektijuht kirjutab nendele funktsionaalsuste blokkidele anaüüsi ning jaotab funktsionaalsused väiksemateks tükkideks (*task*'ideks).
3. Arendaja annab igale *task*'ile mahuhinnangu.
4. Projektijuht ja arendaja saavad iga nädal kokku, et panna *task*'idest kokku nädala arenduse plaan (*sprint backlog*).
5. Projektijuht annab ka disainerile ülesandeid vastavalt arenduse suunale.
6. Disain valmib paralleelselt arendusega.
7. Projektijuht testib valminud funktsionaalsusi.
8. Projektijuht edastab ilmnenud vead arendajale. See protsess kordub kuni *task*'i funktsionaalne osa toimib vigadeta.
9. Kui valmib vigadest vaba avaldamiseks valmis versioon, laeb arendaja selle üles avalikku keskkonda. Sellega on *sprindi* arendus lõppenud.

2.1.3.2 Rollid

Kuna meeskond on väike ja arendajate hulk ei ole väga suur, siis töötab väga hästi hetkel selline korraldus, et suhtlus arendajate ja disaineri vahel käib läbi projektijuhi. Kuna nii disainer kui ka üks arendajatest on vabakutselised, siis aitab projektijuhikeskne kommunikatsioon kuludel silma peal hoida.

Kui arendajaid tuleks juurde, siis peaks hakkama mõtlema ka kommunikatsiooni optimeerimisele.

Samuti on suur pluss, et arenduse seisust on projektijuhil pidevalt ülevaade olemas. Projektijuht hoiab tegevjuhti pidevalt arendusega kursis ning koos mõeldakse edasisele suunale.

2.1.3.3 Rollide koormus

Projektijuht: Koormus kõigub umbes 100% juures. Põhiline projektijuhi töö on analüüsi kirja panemine ja prototüüpimine. Arenduse ajal hoiab ta arendusel silma peal, suhtleb disaineriga ning testib funktsionaalsusi nende valmimisel. Sprindi lõppedes on põhiline tegevus testimine.

Tegevjuht: Koormus ei sõltu arendusest, kuna tema ei puutu otseselt arendusega kokku.

Disainer: Koormus on kuskil 20%, aga väga kõikuv. Kui tehakse analüüsi, siis on ka disanerit rohkem vaja, kuna koostöös projektijuhiga mõtlevad nad välja kasutajaliidest ja –mugavust.

Arendaja: Koormus on alati 100%, kuna arenduse plaanis on väga palju *task*'e. Küll aga ei planeerita üle 35h arendusele, kuna aega peab jääma veel koosolekutele ja äkitselt üles kerkivate probleemide lahendamisele.

2.2 Fleep

Fleep on teinud *chat*-põhise suhtlusplatvormi lihtsamaks ja universaalsemaks tuues ettevõtte suhtluse kokku ühte kohta.

Ettevõte on loodud 2012. aasta novembris nelja endise Skype'i inseneri poolt kui tekkis vajadus funktsionaalsema sõnumisaatmise järele. Näiteks ei saanud Skype'i sõnumites otsida ega lisada vestlusesse inimesi, kel polnud Skype'i kontot. Samuti ei sünkroniseeruinud sõnumid eri seadmete vahel.
Fleep võimaldab kõike eelnimetatut, kui ka sõnumite märgistamist, teemade pealkirjastamist ja failide saatmist. Hiljem on võimalik faile eraldi *tab*'is lehitseda, ega pea otsima kuskilt kirjavahetusest.
Lisaks, hea sõnumirakendus peaks olema ühilduv kõigi seadmete ja platvormidega, et kommunikatsioon ei jääks tehniliste nüansside taha kinni.

Tavaline probleem on see, et kui meeskonnasisene kommunikatsioon peab meeskonna seest välja levima, siis on uue osapoole sisse toomine keeruline – tihti on vaja kontot registreerida, kinnitada ja kogu standardprotsess läbi teha. Fleep'iga on see lihtne, kuna saab kasutada ka tavalist e-maili aadressi, et uut inimest vestlusesse tuua.
Fleep keskkonnas on ka:

- Failihaldussüsteem, kus saab tähtsaid dokumente ühes koos hoida.
- Integratsioonid GitHub'i ja JIRA'ga.

Seega Fleep on sõnumirakendus, mis teeb meeskondade suhtluse paremini organiseeritumaks, kiiremaks ja lihtsamaks.

Fleep on kättesaadav nii veebirakendusena, iOSi kui ka Androidi aplikatsioonina.

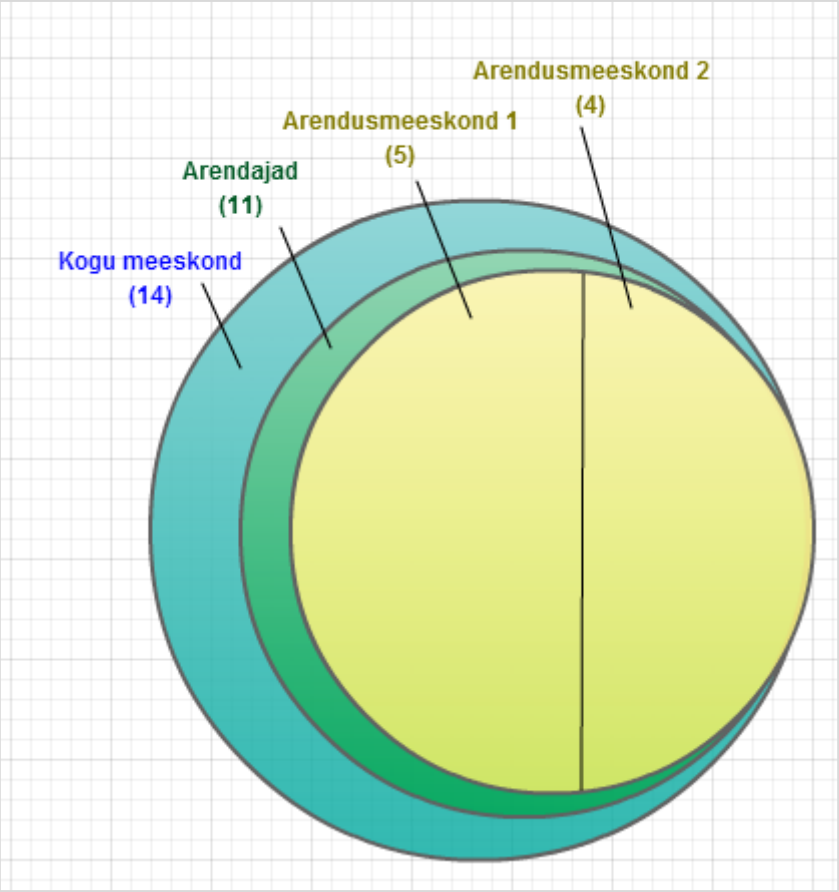
Fleep'il on üle 19 000 kliendi, millest suur osa kasutajatest on idufirmade maastikult. See pole üllatav, kuna suhtlus idufirmades peab olema lihtne, kuid paindlik.

2.2.1 Meeskond

Kokku on Fleep'is 14 töötajat:

- Analüütik – Ühtlasi ka ettevõtte tegevjuht. Kirjutab analüüsi ja kasutajaliidese spetsifikatsiooni. Tehnilistesse detailidesse ja teostusesse ei lasku – see jääb arendajate teha.
- Disainer – Kujundab ja mõtleb kasutaja kogemuse osa vastavalt kogu arenduse vajadusele. Suures osas analüütikuga koos töötav.
- Turundusspetsialist
- 11 arendajat:
    - 1 Android arendaja
    - 1 iOS arendaja
    - 4-liikmeline arendusmeeskond (veeb ja *backend*)
    - 5-liikmeline arendusmeeskond (veeb ja *backend*)

Fleep'i arendusmeeskonda võib kujutada järgneva joonisega



Joonis 1 Graafiline kujutus Fleep'i arendusmeeskonnast

2.2.2 Planeerimine

Ettevõttes toimub nii pikaajaline kui ka lühiajaline planeerimine. Analüüsin kumbagi eraldi.

2.2.2.1 Pikaajaline planeerimine

Iga kvartali alguses on järgmise 3 kuu tegevuse planeerimine.
See on pikk 2-päevane koosolek, kus on kohal analüütik ja kõikide valdkondade juhid (Android, iOS, veeb, disain).

Planeeritavad tegevused võetakse backlogist. Esmajärjekorras on kõik, mis oli planeeritud, aga jäi tegemata eelmises kvartalis. Antakse ka mahuhinnangud funktsionaalsustele, kuigi need on väga üldised – kas millegi arenduse maht on nädalate või kuude piires.

Analüüsi ja funktsionaalsuste valiku teeb analüütik. Tema kirjutab ka spetsifikatsiooni.
Tehnilistesse detailidesse analüütik ei lasku, see on puhtalt arendajate läbimõtlemise osa.
Funktsionaalsuste valik (mida edasi arendada) tuleb:

- Kvartali planeerimisel paika pandud suunast
- Tagasisidest kasutajatelt
- Nö „kõhutundest“

2.2.2.2 Lühiajaline planeerimine

Iga nädala alguses on nädala planeerimine, kus on kohal scrum masterid, disainer, analüütik ja turundusspetsialist. Sellel koosolekul annavad scrum masterid ülevaate oma meeskondade olukorrast. Räägitakse põhiliselt, mida käesoleval nädalal on plaanis teha.

2.2.3 Arendusprotsess

Esitan üldiselt arendusprotsessi nummerdatult tegevuste toimumise järjekorras. Detailsem protsessi kirjeldus on protsessijoonises lisas 2.

2.2.3.1 Üldine veebi arendusprotsessi kirjeldus

1. Kvartali planeerimine. Kõikide valdkondade juhid saavad kokku ning arutavad järgmine kvartali suunda.
2. Analüütik formuleerib järgmise kvartali suuna ning oskab juba funktsionaalsusi prioritiseerida.

3. Analüütik paneb paika järgmise nädala *backlog*'i ja kirjutab esialgse funktsionaalsuste spetsifikatsiooni. Pärast seda uusi mõtteid sellesse nädalasse vastu ei võeta, vaid pannakse need tuleviku *backlog*'i.
4. Uuel nädalal on nädala planeerimine, kus osalevad scrum masterid, disainer ja analüütik. Vaadatakse koos üle spetsifikatsioon. Kui erinevate valdkondade inimesed vaatavad koos otsa samale probleemile, tekivad erinevad mõtted, lahendused ja tagasiside.
5. Analüütik muudab spetsifikatsioonidokumendi vastavalt tagasisidele. Mõnikord on muudatused väiksed, mõnikord kuni 1\3 ulatuses.
6. Algab töö täpsustatud veebi funktsionaalsuste välja arendamiseks.
7. Disain valmib paralleelselt arendusega.
8. Disaini tehakse vastavalt vajadusele ümber.
9. Arenduse käigus tekivad vigade olukorrad, mida arendajad lahendavad vastavalt esinemisele. Vajadusel tõmmatakse skoopi koomale kui tehniline teostus osutub väga palju keerulisemaks.
10. Kui tekib uusi mõtteid, siis need kantakse tuleviku *backlog*'i.
11. Igas meeskonnas testivad arendajad valminud funktsionaalsusi nende valmimisel.
12. Kui valmib viimane vigadeta versioon, laetakse see üles avalikku keskkonda.

2.2.3.2 Rollid

Tööjaotus on umbkaudse läbirääkimise tulemus. Positiivne on see, et ei kulutata palju aega läbirääkimistele, vaid pigem koodi kirjutamisele. Negatiivne on see, et üllatusi tuleb liiga palju. Olukordi, kus arendaja ei saa edasi tegutseda, kuna vajab midagi (nt mõni koodi osa on veel kirjutamata või analüüs mittetäielik). Sellised olukorrad lahendatakse nende esinemisel, reageeritakse olukorrale.

Scrum master – Meeskonna poolt valitud esindaja, ei ole otseselt juht, vaid samuti arendaja. Tema suhtleb *product owner*'iga, võtab oma meeskonna tegemised kokku ja seisab meeskonna huvide eest. Teadlikult ei tembeldata teda „juhiks", kuna sel juhul hakkaks ülejäänud meeskond kohtlema teda kui juhti ning see rikuks meeskonna ühtsust.

Testimine – Toimub meeskonna siseselt ja tihti. Kui võrrelda Skype'ga, siis seal olid selged kihid – arendus, testimine, analüüs. Fleep'is on tiim ise organiseeruv ja kollektiivselt vastutav.

2.2.3.3 Rollide koormus

Analüütik: Koormus kõigub umbes 100% juures. Arenduse järjekorras olevate funktsionaalsuste sisu ja prioriteedid muutuvad tihti. Lisaks on nende funktsionaalsuste arv alati suurem kui arendaja reaalselt valmis jõuab arendada. Iga nädal kantakse mingid funktsionaalsused üle järgmise etapi arendusse. Positiivne on see, et igal ajahetkel on olemas mingi analüüsi spetsifikatsioon, mida pole veel realiseeritud. Seega, ei jää arendus töö puudumise tõttu kunagi seisma.
Kuna analüütik peab tegelema paljude operatiivsete teemadega, siis tihti on nädala lõpus palju planeeritud asju veel tegemata ja koormus kasvab järsult just nädala lõpus.

Disainer: Samuti pidevalt täiskoormus. Mingite ajahetkede tagant tehaks suurem kasutajaliidese parendamise projekt, kus refaktoreeritakse kogu disaini ning viiakse sisse suuremad disainimuudatused. Nendel aegadel on disaineri koormus ka suurem.

Arendajad: Arendajate koormus ka ei sõltu kvartali planeerimistest. Arendus toimub pidevalt ja seisakuid tekib vähe. Kui plaanis on võtta ette mõni arendus, mis mõjutab kogu koodi struktuuri, siis tehakse ka kogu koodi refaktoreerimisi. Neid planeeritakse rohkem kui disini refaktoreerimisi, seega ei kasva sellega arendajate koormus.

3. Agiilsed arendusmetoodikad ja nende praktikad

Agiilne arendusmetoodika sai alguse 2001. aastal, kui sai selgeks, et IT arendus jookseb liiga tihti üle eelarve, üle aja ning selle tulemusena pole arenduse tellija rahul. [2] Grupp inimesi pani aluse uuele arendumetoodika suunale, mis väärtustab:

- Inimesi ja suhtlust rohkem kui protsesse
- Töötavat tarkvara rohkem kui dokumentatsiooni
- Kliendi koostööd rohkem kui lepingu läbirääkimisi
- Muutustega kohanemist rohkem kui plaani järgimist

Lähtun kolmest agiilsest arendusmetoodikast, milleks on Scrum, Kanban ja ekstreemprogrammeerimine. Kirjeldan neid metoodikaid ning toon välja nende praktikad.

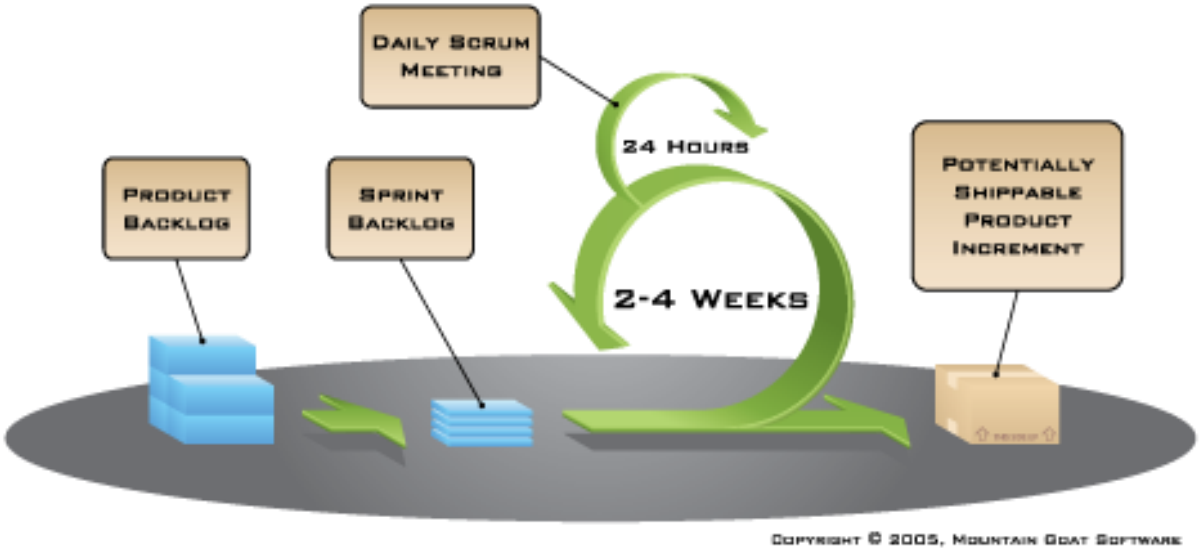
3.1 Scrum

Scrum on paindlik tarkvara arenduse raamistik, mille töö põhineb tsüklilisel arendusel. Neid arendusetsükleid nimetatakse *sprint*'ideks. Arenduses lähtutakse kliendi vajaduste prioriteetidest. [3]
Scrum praktikad on alljärgnevad:

- Task
    - Konkreetne ülesanne ühele inimesele. Ajaliselt kuni 2 päeva.
- Stand-up meeting
    - Igapäevane koosolek kiire ülevaate saamiseks. Maksimaalselt 15 minutit kestev. Arendajad räägivad, mida on teinud, mida plaanivad teha ja mis neid takistab.
- Product backlog
    - Projekti reserv, mis koosneb nõudmistest funktsionaalsusele ja on järjestatud tähtsuse järgi.
- Sprint
    - *Task*'idest koosnev 2-4 nädalat kestev arendustsükkel, mille käigus arendatakse välja kindlaks määratud funktsionaalsus või funktsionaalsused, mis on kliendile sel hetkel kõige prioriteetsemad.
- Sprint planning
    - Iga sprindi jaoks tööde valimine ja hindamine

- Sprint review meeting
    - Demo, mille käigus liikmed demonstreerivad tellijale, mida nad on sprindi jooksul teinud. Põhiline nõue, et pärast igat sprinti peab programm töötama.
- Sprint retrospective meeting
    - Tiimi liikmed jagavad mõtteid eelmise sprindi kohta. Teevad kokkuvõtte, mis läks hästi, mis läks halvasti ja mida tuleks parandada.
- Burndown chart
    - Progressi jälgimise tööriist, mis näitab, kui palju arendust on veel ees projekti lõpuni. Tavaliselt on see graafik jaotatud *sprint*'ide kauap.

Scrumi arendustsüklit võib kujutada järgneva joonisega



Joonis 2 Graafiline kujutis Scrum arendusmetoodikast [4]

3.2 Kanban

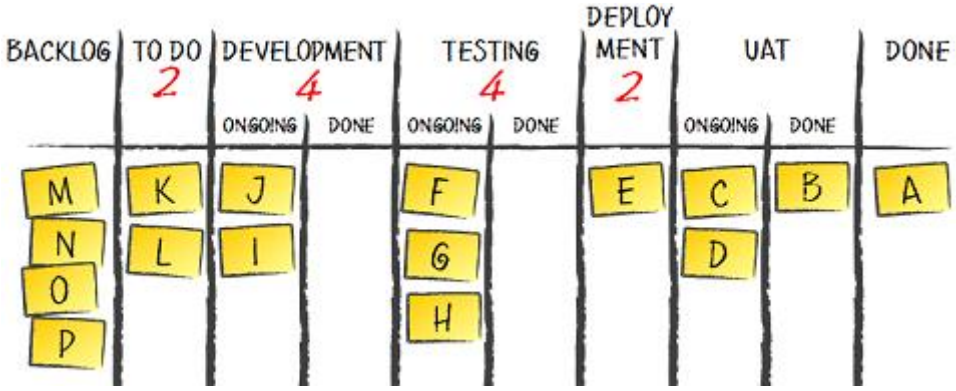
Kanban on tarkvaraarenduse metoodika, mille põhilised komponendid on tööde visuaalne juhtimine, käimasoleva töö hulga piiramine ning põhimõte, et tööd „tõmmatakse" juurde, mitte ei suruta edasi järgmisse faasi. [5]

Kanban praktikad on alljärgnevad:

- Protsessi läbipaistvus
    - Kogu arendusprotsess peab olema nähtav. Selleks on kanban tahvel, kus liigutatakse taske ringi kohe, kui nad valmivad. See tahvel omab keskset rolli.

- Liikuvus
    - Iga funktsionaalsuse arendus peab liikuma pidevalt tahvlil. Niipea, kui ta ühes faasis valmis saab, peab liikuma juba järgmisse faasi. Ei mingeid vaheladusid.
- „Pull“ põhimõte
    - Pudelikaelade vältimiseks on igas faasis maksimaalne lubatud arv *task*'e. Kui faasis on juba maksimaalne arv *task*'e, siis ei saa sinna *task*''e juurde lisada.
- Liiasuse eemaldamine
    - Liiasus (*waste*) on kõik, mis kasutab ressursse, aga ei anna lisaväärtust. Igasugune liiasus tuleb eemaldada.
- Pidev töövoog
    - *Sprint*'e ei ole, töö peaks pidevalt käima. See saavutatakse pudelikaelade ja liiasuse eemaldamisega.
- Meetingud
    - Kõik võivad rääkida, mitte ainult arendajad. Põhiline teema on tahvli peal toimunud muudatused.

Kanban tahvlit, tööde limiite ja tegemises olevaid töid võib kujutada järgneva joonisega



Joonis 3 Graafiline kujutus Kanban tahvlist [6]

3.3 Ekstreemprogrammeerimine

Ekstreemprogrammeerimine on tarkvara arenduse metood, mis põhineb lihtsusel, kommunikatsioonil, tagasisidel, julgusel ja austusel. Terve meeskond tuleb kokku kliendi juurde, kes ongi äripoole esindaja. [7]

Ekstreemprogrammeerimise praktikad on alljärgnevad:

- Paarisprogrammeerimine
    - Kaks osavat programmeerijat suudavad kirjutada koodi rohkem kui kahekordse efektiivsusega. 1+1=3 idee. Üks arendaja kirjutab koodi, teine kontrollib ja keskendub koodile laias plaanis.
- Planeerimismäng
    - Koosolek, mis toimub tavaliselt korra sprindi jooksul. Selgitatakse välja lähiajalised prioriteedid ja millal neid peaks realiseerima. Siin uurivad analüütikud ja arendajad funktsionaalsusi ja valmistatavad ette ideed arenduseks.
- Test-driven development
    - Keskendub lühikestesse ja korduvatesse arendustsüklitesse. Arendaja kirjutab alguses funktsionaalsusele testi. Seejärel kirjutab minimaalse hulga koodi, et see test annaks positiivse tulemuse. Seejärel korrastab koodi vastavalt standarditele.
- Meeskond
    - Klient peaks pidevalt kättesaadav olema, et vastata küsimustele, mis on seotud funktsionaalsuste ja äriloogikaga. Arendajad peavad laiemalt aru saama ka ärist, mille jaoks seda arenduste tehakse.
- Pidev arendus
    - Arendusmeeskond peaks töötama ainult viimase koodi kallal. Iga paari tunni tagant peavad kõik arendajad oma koodi üles laadima, et keegi ei arendaks funktsionaalsust vana koodi peale.
- Refaktoreerimised
    - Arendada ainult seda, mida on hetkel vaja ja seda nii lihtsalt kui võimalik. See võib mõnikord luua olukorra, kus kood on raskesti skaleeritav. Sellistel hetkedel refaktoreeritakse koodi pidades silmas tuleviku vajadusi.
- Lühikesed iteratsioonid
    - Tarkvara tootmine järgib põhimõtet, et väljalasked peaksid toimuma tihedalt. See annab tellijale kindlust, et projekt areneb. Nii saab ka kiiret tagasisidet ning hoiab ära liikumist vales suunas.
- Koodi standard
    - Arendajad lepivad kokku ühtses stiilis ja kindlates koodimustrites. Kõik peaksid üht moodi mõistma, milliseid praktikaid kasutada ja millistest hoiduda antud projekti raames.

- Kood on ühisomand
    - Iga arendaja on koodi eest vastutav täies ulatuses. See tähendab ka seda, et igaüks võib mistahes osa koodist muuta. See kiirendab ka arendust, sest iga arendaja võib mistahes viga parandada.
- Koodi lihtsustamine
    - Arendajad peatuvad tihti ja küsivad endalt, et kas on olemas sama funktsionaalsuse teostamiseks lihtsam viis. Kui jah, siis kirjutatakse see osa koodist ümber lihtsamale kujule.
- Ühtlane tempo
    - Arendajad tegelevad arendusega mitte rohkem kui 40 tundi nädalas. See on vältimaks kvaliteedi langust.
- User story
    - Kogu arendus on jaotatud kasutuslugudeks, mis kirjeldab mittetehnilises keeles kasutaja tegevust süsteemiga. Põhimärksõna on kasutuslugude lihtsusel ja arusaadavusel.

Ekstreemprogrammeerimise praktikate tsüklilisust võib illustreerida järgneva joonisega

Joonis 4 Graafiline kujutus Ekstreemprogrammeerimise tsüklitest [8]

4. Agiilsete praktikate rakendamise analüüs

Analüüsin igat idufirmat eraldi ning käsitlen neid arendusmetoodikate alusel. Uurin, kas metoodika praktikaid rakendatakse ning kas praktikate rakendamine või mitterakendamine on põhjendatud või mitte.

Kirjeldan tabeli strukuuri:

- Tabeli bloki alampealkiri nimetab metoodikas kasutatava praktika.
- Tabeli vasak veerg esitab, kas seda praktikat rakendatakse või mitte. Kui ei rakendata, siis kirjeldatan, kuidas see protsess alternatiivina toimub.
- Tabeli parem veerg esitab, kas selle praktika kasutamine on põhjendatud või mitte. Kui on põhjendatud, siis täpsustan, kuidas see on põhjendatud. Kui ei ole põhjendatud, siis täpsustan, miks praktika kasutamine ei ole põhjendatud.

4.1 RentMarket

Analüüsin igat agiilse arendusmetoodika praktikat eraldi. Samuti analüüsin, kas konkreetsete praktikate rakendamine või mitterakendmine ettevõttes RentMarket on põhjendatud või mitte.

4.1.1 Scrum

Analüüsin Scrum arendusmetoodikat ettevõttes RentMarket praktikate kaupa.

Tabel 1 Scrum metoodika rakendamine ettevõttes RentMarket

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Task	
Osaliselt	**Jah**
Töö on *task*'ideks jaotatud, kuigi mõni *task* võib olla pikem, kui 2 päeva.	Töö jaotamine funktsionaalseteks osadeks on hea meetod arenduse haldamiseks. Üle 2 päeva pikad taskid ei ole põhjustanud segadust.
Stand-up meeting	
Osaliselt	**Ei**
Veebi arendaja ja projektijuht teevad iga päev *stand-up* koosolekuid.	Veebi arendaja ja projektijuht ei peaks nii tihti stand-up koosolekuid tegema, kuna nad on üksteise tegemisega piisavalt kursis.

Product backlog	
Jah	**Jah**
Iga *sprindi* jaoks on backlog, mille funktsionaalsuste valmimine on üldjuhul võrdse tähtsusega, kuna versioon avalikustatakse alles nende kõigi valmimisel. Arendaja valib ise arendamise järjekorra.	Planeerimine on väga sujuv selle tõttu, et on tehtud etapi vajalike funktsionaalsuste nimekiri.
Sprint	
Osaliselt	**Jah**
See on 1 nädala pikkune ning sisaldab teatud hulka funktsionaalsusi. Soovitatud on minimaalselt 2 nädalane sprint.	Selline arenduse pikkus kindlustab pidevad ja regulaarsed rakenduse väljalasked. See hoiab lõppkasutajad huvitatuna.
Sprint planning	
Jah	**Jah**
Iga nädala lõpus tehakse järgmiseks nädalaks arendatavate funktsionaalsuste valikut.	Siiamaani pole arendatud veel ebavajalikku funktsionaalsust ega arendajal pole olnud olekut, kus ta ei tea, mida edasi teha.
Sprint review meeting	
Ei	**Jah**
Kuna testimine toimub vastavalt vajadusele ja meeskond on väike, siis ollakse pidevalt arenduse käekäiguga kursis.	Teadaoleva funktsionaalsuse demomine võtab aega ja pole vajalik.
Sprint retrospective meeting	
Ei	**Jah**
Selleteemast eraldi koosolekut ei toimu. Kui projektijuht või arendaja paneb tähele midagi, mis on hästi, mis on halvasti, mida võiks paremini, siis ta jagab seda meeskonnaga kohe.	Nendel koosolekutel poleks midagi rääkida, kuna kõikide murekohtadega tegeletakse vastavalt vajadusele esinemise hetkel.
Burndown chart	
Ei	**Jah**
Kõikidele *task*'idele on antud mahuhinnang ning kokku annavad nad teatud tundide arvu.	Kuna arendajaid on vähe, siis ei näe vajadust eraldi *chart*'i jaoks.

4.1.2 Kanban

Analüüsin Kanban arendusmetoodikat ettevõttes RentMarket praktikate kaupa.

Tabel 2 Kanban metoodika rakendamine ettevõttes RentMarket

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Protsessi läbipaistvus	
Osaliselt	**Jah**
Protsessi ei visualiseerita füüsilisel kanban tahvlil, kuid *task*'ide haldamiseks kasutatakse tarkvara, mis meenutab kanban tahvlit.	Füüsilisele tahvlile task'ide kandmine on ajamahukas. See on samuti ebapraktiline, kuna praktiseeritakse palju kodukontorit ning siis sobibki paremini kõigile kättesaadav *online* tarkvara.
Liikuvus	
Ei	**Ei**
Analüüs on pikemalt ette ära tehtud ja arendajal on alati backlogis suurel hulgal ülesandeid juba ees.	Kuna vastavalt prioriteetide muutumisele, kasutajate tagasisidele ja ärisuuna muutustele muutuvad arenduste funktsionaalsed nõuded, siis on pikalt ette analüüsimine analüütiku ajaressursi ebapraktiline kasutamine.
„Pull" põhimõte	
Osaliselt	**Jah**
Ei limiteerita taskide arvu arenduses. Küll aga pannakse tundidega paika sprindi arenduse maht. Nädala arenduse maht on 35h.	Arendajat ei taheta üle koormata ning seetõttu planeeritakse arendust.
Liiasuse eemaldamine	
Jah	**Osaliselt**
Projektijuht üritab kõik administratiivsed ülesanded arendajalt enda peale võtta, et viimane ei peaks tüütuid ülesandeid arenduse ajast tegema.	Arendajale võiks jääda mõned loovust vajavad ülesanded, mis pakuvad vaheldust. Projektijuht peaks ära taipama, millised need konkreetsed ülesanded on ja edastama need arendajale.
Pidev töövoog	

Ei	**Jah**
Kuna toimuvad väljalasked, siis on arendus jaotunud *sprint'*ideks. Küll aga s*print*'ide vahel ei ole pause, kuna analüüsitud funktsionaalsusi on alati *backlog*'is olemas.	*Sprindid* on hea vahend planeerimiseks, kui tahetakse kontrollide ka uute versioonide väljalasete aega.
Meetingud	
Jah	**Jah**
Strateegia koosolekutel osaleb ka veebi arendaja.	Arendajapoolne tehniline vaade asjadele on tihti aitanud suuna paika panemisel.

4.1.3 Ekstreemprogrammeerimine

Analüüsin Ekstreemprogrammeerimise arendusmetoodikat ettevõttes RentMarket praktikate kaupa.

Tabel 3 Ekstreemprogrammeerimise metoodika rakendamine ettevõttes RentMarket

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Paarisprogrammeerimine	
Ei	**Jah**
Ei leia rakendust vähese arendusressursi tõttu.	Palgal olevaid arendajaid on üks ja tellitud arendust teeb samuti ainult üks arendaja.
Planeerimismäng	
Ei	**Jah**
Toimuvad järjepidevalt strateegia ja operatiivsed koosolekud, kus seatakse prioriteedid. Ei ole seotud sellega, et arendajad uurivad funktsionaalsusi.	Planeerimine eksisteerib küll, aga funktsionaalsuste tehnilist külge uurib arendaja iseseisvalt. Koosolekutel liiga tehnilistesse detailidesse laskumine ei ole ajaliselt efektiivne.
Test-driven development	
Ei	**Jah**
Kui on mõni keerukas funktsionaalsus, siis kirjutatakse sellele hiljem test. Enne funktsionaalsuse välja arendust testi ei kirjutata.	Vigade parandamine nende esinemisel on kokkuvõttes kiirem, kui iga funktsionaalsuse kohta testi kirjutamine. Arvestades hetkest

	koodi keerukust, muudaks TDD arendusprotsessi aeglasemaks.
Meeskond	
Jah	**Jah**
Arendajatele selgitatakse ka äriloogikat ja põhjusi, miks teatud arendusi tehakse. Projektijuht on isegi nädalavahetustel kättesaadav, et arendajaga suhelda.	Arenduste järjekord ja funktsionaalsuste sõltuvused on ärilises mõttes olulised. Sisendit nendele otsustele oskab hästi anda just arendaja.
Pidev arendus	
Jah	**Jah**
Kuigi Android rakenduse ja veebi kallal töötab ainult kummaski üks arendaja, on neil hea harjumus tihedalt koodi *committ*'ida.	Versioonihalduse jaoks on see vajalik.
Refaktoreerimised	
Jah	**Jah**
Kui arendatakse välja mingit funktsionaalsust, siis arendaja ei mõtle liiga palju selle peale, mida tulevikus juurde arendatase. Refaktoreerimisi tehakse vastavalt vajadusele.	Liiga tulevikku mõtlemine aeglustab liialt arenduse tempot. Kuna arendus peab olema võimalikult adaptiivne, siis on see õige praktika.
Lühikesed iteratsioonid	
Jah	**Jah**
Veebi arendaja töötab täiskohaga ning veebi uute funktsionaalsuste väljalasked toimuvad mitu korda nädalas.	Nii saadakse kasutajatelt kohe tagasisidet uue funktsionaalsuse kohta.
Koodi standard	
Osaliselt	**Jah**
Veebi ja android arendaja ühtne stiil seisneb ainult API ühilduvuses mõlema keskkonnaga. Küll aga seda ei saa lugeda ühtseteks koodimustriteks, kuna tegu on ikkagi erinevate keskkondadega.	Kuna väga palju ühtseid stiile ei saa eri keskkondadel seada, siis on praegune praktika põhjendatud.
Kood on ühisomand	
Ei	**Jah**

Kuna mõlema platvormi kallal töötab ainult üks arendaja, siis tema vastutab täielikult koodi eest.	Ühe inimese vastutus ei ole ühisomand.
Koodi lihtsustamine	
Ei	**Jah**
Mõne funktsionaalsuse lihtsustamine tuleb küsimuseks alles siis, kui seda funktsionaalsust on vaja edasi arendada.	Liiga kauaks mõne funktsionaalsuse juurde jäämine pärsib edasiminekut.
Ühtlane tempo	
Jah	**Jah**
Arenduse planeeriminel määratakse veebi arendajale 35 tunni ulatuses *task*'e.	Koodi kvaliteedi hoidmiseks on see vajalik.
User story	
Ei	**Jah**
Analüüsi saab hetkel ka arenduse *task*'ideks piisavalt lihtsasti jaotada.	Ärianalüüs ja süsteemianalüüs toimuvad sama inimese poolt.

4.2 Fleep

Analüüsin igat agiilse arendusmetoodika praktikat eraldi. Samuti analüüsin, kas konkreetsete praktikate rakendamine või mitterakendmine ettevõttes Fleep on põhjendatud või mitte.

4.2.1 Scrum

Analüüsin Scrum arendusmetoodikat ettevõttes Fleep praktikate kaupa.

Tabel 4 Scrum metoodika rakendamine ettevõttes Fleep

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Task	
Jah	**Jah**
Töö on *task*'ideks jaotatud, kuid mõni *task* võib olla pikem, kui 2 päeva.	Töö jaotamine funktsionaalseteks osadeks on hea meetod arenduse haldamiseks. Üle 2 päeva pikad *task*'id ei ole põhjustanud segadust.
Stand-up meeting	

Ei	**Ei**
Hetkel asendab seda grupipõhine *chat*.	Nähakse aina rohkem vajadust selle järele, et planeerida arenduste järjekorda paremini.
Product backlog	
Jah	**Jah**
Sprindi task'ide arendamise järjekord tuleb meeskonna enda poolt.	Meeskond saab ise koordineerida ja paika panna kes mida teeb ja mis järjekorras tehakse.
Sprint	
Jah	**Jah**
Kogu arendus on jaotatud *sprint*'ideks.	Tagab regulaarsed väljalasked.
Sprint planning	
Osaliselt	**Jah**
Seda tehakse lühiajalistel koosolekutel koos *scrum master*'itega. Ülejäänud arendajad seal ei osale.	*Scrum master* on kogu meeskonna arendustega kursis ning suudab tervet meeskonda esindada. See on ressursisäästlikum, kui see et kogu meeskond oleks kohal.
Sprint review meeting	
Ei	**Jah**
Ollakse arendusega kursis ning eraldi *demo*'mise vajadust ei ole.	Kõigile teadaoleva funktsionaalsuse demomine võtab aega ja pole vajalik.
Sprint retrospective meeting	
Osaliselt	**Jah**
Seda ei praktiseerida pärast igat sprinti. Selline arutelu tuleb kvartali planeerimise koosolekul.	Arendus on suhteliselt välja kujunenud juba ning kord kvartalis tegevuste analüüsimisest praegu piisab.
Burndown chart	
Ei	**Jah**
Siiamaani pole vajadust olnud. Seda ei saaks kohe implementeerida ka, kuna ei kasutata isegi *estimation* punkte.	Seda tuleks hakata siis kasutama, kui projektijuht ei suuda enam arenduse pikkust planeerida.

4.2.2 Kanban

Analüüsin Kanban arendusmetoodikat ettevõttes Fleep praktikate kaupa.

Tabel 5 Kanban metoodika rakendamine ettevõttes Fleep

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Protsessi läbipaistvus	
Osaliselt	**Jah**
Füüsilise tahvli peal ei visualiseeria task'e, kuid kasutatakse taskihaldustarkvara.	Füüsiline tahvel ei ole vajalik, kuna *stand-up* meetingud hoiavad inimesi samuti üksteise tegemistega kursis.
Liikuvus	
Jah	**Jah**
Kuna ettevõttes on juba üle kümne arendaja, siis peab rohkem arvestama, kas kellegi töö sõltub kellegi teise töö valmimisest. Selliste seisakuid üritatakse meeskondade siseselt aktiivselt ära hoida.	Kui ühte kohta jääb mõne funktsionaalsuse arendus kinni, siis viibib kogu arenduse valmimine.
„Pull" põhimõte	
Osaliselt	**Jah**
Üritatakse vanad ülesanded ära lahendada enne kui uusi võetakse. Kui tekib pudelikaela efekt, siis kogu meeskond püüab lahendada koos seda *task*'i. See vähendab seisakut.	Kui võtta uusi taske peale enne, kui vanad on tehtud, siis meeskonna fookus hajub ja taskid jäävad poolikult tehtuks.
Liiasuse eemaldamine	
Ei	**Jah**
Lähtutakse põhimõttest, et tarkvaraarendus on loominguline protsess ja mõningane muude asjadega tegelemine on vajalik.	Arendaja vajab natuke mõttevaheldust ning muude asjadega tegelemist, et mõtlemine püsiks värskena.
Pidev töövoog	
Ei	**Jah**
Kuna toimuvad väljalasked, siis on arendus jaotunud s*print*'ideks. Küll aga s*print*'ide vahel ei ole pause, kuna analüüsitud funktsionaalsusi on alati *backlog*'is olemas.	Planeerimiseks kasutatakse *sprint*'e, kuna see aitab paremini väljalaskeid planeerida.
Meetingud	

Jah	**Jah**
Nii lühiajalistel kui ka kvartali koosolekutel osalevad *scrum master*'id.	Arendajad peaksid ärilise poole ja üldise suunaga kursis olema. Neile vajaliku info edastab *scrum master*.

4.2.3 Ekstreemprogrammeerimine

Analüüsin Ekstreemprogrammeerimise arendusmetoodikat ettevõttes Fleep praktikate kaupa.

Tabel 6 Ekstreemprogrammeerimise metoodika rakendamine ettevõttes Fleep

Kas praktikat rakendatakse?	**Kas rakendamine/mitterakendamine on põhjendatud?**
Paarisprogrammeerimine	
Ei	**Jah**
Paarisprogrammeerimist kasutatakse vaid siis, kui ettevõttesse tuleb uus arendaja ning teda peab juhendama. Seega selle eesmärk ei ühti ekstreemprogrammeerimise metoodikat defineeritud eesmärgiga.	*Task*'id ei ole nii keerukad ja laiahaardelised, et oleks vaja juurde teist arendajat, kes hoiaks silma peal struktuuril ja jälgiks funktsionaalsuse mõju ülejäänud süsteemile.
Planeerimismäng	
Ei	**Ei**
Sprindipõhist planeerimist pole. On kvartali- ja nädalaplaneerimine.	Paljud planeerimata tööd tekivad *sprindi* ajal ning tihti on nädala lõpus töökoorem kasvanud liiga suureks. Planeerimata *task*'id tuleks juba alguses arendusse sisse planeerida.
Test-driven development	
Osaliselt	**Ei**
Kui arendaja mõistab kohe alguses, et antud funktsionaalsus on piisavalt keeruline, et mõistlik on enne kirjutada testid. Sellisel juhul on funktsionaalsuse reeglite hulk on päris suur.	On olnud juhuseid, kus rakendus laetakse avalikku keskkonda ning selgub, et rakendus on vigane. Selle parandamiseks testitakse, parandatakse ning laetakse uus versioon. Kogu see protsess võib võtta mitu nädalat. Testimist peaks rakendama selleks, et katta suurem osa koodist testidega.

Meeskond	
Jah	**Jah**
Scrum masterid osalevad nii lühiajalistel kui ka pikaajalistel planeerimiste koosolekutel. Meeskond on kättesaadav.	Kui arendajad mõistavad ärilist poolt, suudavad nad arenduses teha ka kaalutletud otsuseid. See hoiab ära loogikavigu ja lisab loomingulist mõtlemist.
Pidev arendus	
Jah	**Jah**
Tehakse väga tihti *commit*'e, et vältida igasuguseid võimalikke konflikte koodis.	Koodikonfliktide lahendamine võtab palju rohkem aega, kui koodi tihe *commit*'imine.
Refaktoreerimised	
Jah	**Jah**
Refaktoreerimine käib teatud aja tagant, stabiilsete sammudena. Jõutakse arendusega kuhugi maale, kus saadakse aru, et koodi kvaliteeti tuleb parandada. See korraks destabiliseerib kogu arenduse.	Kuna arendus peab olema võimalikult adaptiivne, siis on see õige praktika. Kaugete tuleviku arenduste peale mõtlemine aeglustab arenduse tempot.
Lühikesed iteratsioonid	
Jah	**Jah**
Iteratsioone hoitakse lühikestena, see võimaldab väljalaskeid teha tihedalt.	Tihedad väljalasked uute funktsionaalsuste ja parandustega hoiavad kliente õnnelikena.
Koodi standard	
Ei	**Jah**
API osas lepitakse kokku. Mõned üldised koodimise kokku lepped on ka, aga neid on vähe. Platvormisisestes koodimustrites kokku ei lepita.	Kuna siiamaani ei ole see probleeme tekitanud, siis ei näe vajadust seda praktikat ka sisse tuua.
Kood on ühisomand	
Jah	**Jah**
Androidi ja iOSi arendust teeb kummalgi ainult üks arendaja, siis tema vastutab täielikult koodi eest. Veebi platvormil ja *backend*'is tehakse pidevalt muudatusi ja seega tunnevad kõik vastutust.	Kui arendajal on vastutus, siis ta on motiveeritud kirjutama kvaliteetsemat koodi.

Koodi lihtsustamine	
Ei	**Jah**
Kui seda esineb, siis on see arendaja enda isikupära, aga sellist ühtset praktikat ei järgita.	Vajalik on arendada ka uut funktsionaalsust ja ühe funktsionaalsuse juures ei saa liiga pikalt peatuda.
Ühtlane tempo	
Jah	**Jah**
Aktiivsust hoitakse, aga ületöötamist ei teki.	Ületöötamine tekitab pikas plaanis läbipõlemist, mis viib lõpuks palju suurema kahjuni.
User story	
Jah	**Jah**
Analüütik teeb suures osas ärianalüüsi ja jätab süsteemianalüüsi ja tehnilise teostuse arendajate teha.	Analüütik teeb analüüsi nii iOSi, Androidi kui ka veebi jaoks. Seega ei ole tal aega tehnilistesse detailidesse laskuda.

5. Analüüsi kokkuvõte

Võtan kokku tabelite 1-6 tulemused ja teen nende alusel statistika ja järeldused iga käsitletud idufirma kohta eraldi.

5.1 RentMarket

Analüüsin kui palju kasutatakse ettevõttes RentMarket praktikaid käsitletavatest agiilsetest arendusmetoodikate praktikatest. Samuti analüüsin, kui suur protsent praktikate kasutustest on põhjendatult kasutuses.

Tabel 7 Agiilsete praktikate kasutamise analüüs ettevõttes RentMarket

Scrum		**Kanban**		**Ekstreem-programmeerimine**	
Kas praktikat rakendatakse?	Kas rak. on põhjendatud?	Kas praktikat rakendatakse?	Kas rak. on põhjendatud?	Kas praktikat rakendatakse?	Kas rak. on põhjendatud?
Osaliselt	Jah	Osaliselt	Jah	Ei	Jah
Osaliselt	Ei	Ei	Ei	Ei	Jah
Jah	Jah	Osaliselt	Jah	Ei	Jah
Osaliselt	Jah	Jah	Osaliselt	Jah	Jah
Jah	Jah	Ei	Jah	Jah	Jah
Ei	Jah	Jah	Jah	Jah	Jah
Ei	Jah			Jah	Jah
Ei	Jah			Osaliselt	Jah
				Ei	Jah
				Ei	Jah
				Jah	Jah
				Ei	Jah
Praktika kasutus: 62,50%		**Praktika kasutus: 66,67%**		**Praktika kasutus: 50,00%**	

5.1.1 Scrum metoodika analüüs

Võib öelda, et toote arendusprotsess sarnaneb palju Scrum'ile, kuna kasutatakse väga palju ainult Scrum'ile iseloomulikke praktikaid.

Küll aga väikese arendusressursi tõttu ei kasutata neid Scrum praktikaid:

- Sprint review meetinguid
- Sprint retrospective meetinguid
- Burndown chart

Üks võimalik protsessi parendusekoht:
Stand-up koosolekute ära jätmine. Hetkel toimuvad need koosolekud suures osas lihtsalt sellepärast, et tahetakse seda praktikat rakendada praktika pärast. Küll aga ei ole see vajalik just vähese arendusressursi tõttu.

5.1.2 Kanban metoodika analüüs

Protsentuaalselt kõige rohkem praktikaid kasutuses Kanban arendusmetoodikast (66,67%). Kuigi võib öelda, et arendusprotsess ei ole ikkagi Kanban, kuna mõnda Kanbani fundamentaalpraktikat ei kasutata (näiteks Kanban boardi ega liikuvuse põhimõtet)

Üks võimalik protsessi parendusekoht:
rakendada rohkem liikuvuse põhimõtet, mille kohaselt analüüsidokument ei tohiks liiga kauaks seisma ja arenduse ootele jääda. Seda peaks vältima seetõttu, et äriline suund võib pika aja peale muutuda.

5.1.3 Ekstreemprogrammeerimise metoodika analüüs

Kuna ekstreemprogrammeerimise metoodika järgimine eeldab kõigist 12’st praktikast kinni pidamist, siis ei saa kindlasti öelda, et RentMarketi arendusprotsess seda järgib. Küll aga järgib ta umbes pooli häid praktikaid.

Enamuste praktikate mitterakendamise põhjus on väike arendusressurss. Samuti ei leia selle tõttu ka mõned planeerimise praktikad rakendust.

5.2 Fleep

Analüüsin, kui palju kasutatakse ettevõttes Fleep praktikaid käsitletavatest agiilsetest arendusmetoodikatest. Samuti analüüsin, mitu protsenti nendest praktikate kasutustest on põhjendatult kasutuses.

Tabel 8 Agiilsete praktikate kasutamise analüüs ettevõttes Fleep

Scrum		**Kanban**		**Ekstreem-programmeerimine**	
Kas praktikat rakendatakse?	Kas rak. on põhjendatud?	Kas praktikat rakendatakse?	Kas rak. on põhjendatud?	Kas praktikat rakendatakse?	Kas rak. on põhjendatud?
Jah	Jah	Osaliselt	Jah	Ei	Jah
Ei	Ei	Jah	Jah	Ei	Ei
Jah	Jah	Osaliselt	Jah	Osaliselt	Ei
Jah	Jah	Ei	Jah	Jah	Jah
Osaliselt	Jah	Ei	Jah	Jah	Jah
Ei	Jah	Jah	Jah	Jah	Jah
Osaliselt	Jah			Jah	Jah
Ei	Jah			Ei	Jah
				Jah	Jah
				Ei	Jah
				Jah	Jah
				Jah	Jah
Praktika kasutus: 62,50%		**Praktika kasutus: 66,67%**		**Praktika kasutus: 66,67%**	

5.2.1 Scrum metoodika analüüs

Protsentuaalselt kõige vähem sarnaneb arendusprotsess Scrum'ile. Mõni näide, milliseid Scrum praktikaid ei kasutata:

- Sprint planning
- Burndown chart

Põhjus on see, et ei nähta nii suurt planeerimise vajadust.

Üks võimalik protsessi parendusekoht:

Stand-up koosolekute rakendamine. Probleem seisneb väheses kommunikatsiooni arendajate vahel. Seetõttu näeb tegevjuht, et on kasvav vajadus lühiajaliselt planeerida arenduste järjekorda.

5.2.2 Kanban metoodika analüüs

Põhjus, miks mõndasid praktikaid ei kasutata on see, et rakendatakse sprindipõhist arendust ja ei hoita nii rangelt kinni kanban tahvli jälgimisest ega liiasuse eemaldamisest.

5.2.3 Ekstreemprogrammeerimise metoodika analüüs

Protsentuaalselt kõige rohkem praktikaid kasutuses Ekstreemprogrammeerimise arendusmetoodikast (66,67%). Kuigi võib öelda, et arendusprotsess ei ole ikkagi ekstreemprogrammeerimine, kuna see nõuab kõigi 12 praktika ranget järgimist. Kui vähemalt üks nendest praktikatest ei leia kasututst, siis ei saa arendusprotsessi nimetada ekstreemprogrammeerimiseks.

Võimalikud protsessi parendusekoht:

- Test-driven development'i rakendamine. Oleks mõistlik proovida sellist praktikat, et kirjutatakse funktsionaalsuste testid enne, kuna on tekkinud vigu avalikus versioonis ning parandatud versiooni üleslaadimine võib võtta nädalaid.
- Planeerimismängu rakendamine. Proovida rakendada seda planeerimise praktikat, et paremini valmistuda ootamatusteks.

6. Kokkuvõte

Lõputöö eesmärk oli uurida agiilsete arendusmetoodikate praktikaid, nende rakendamist idufirmades ja seda, kas nende rakendamine on põhjendatud. Edasine eesmärk oli teha praktikate kasutamise statistikat, järeldusi ning leida nende idufirmade toote arendusprotsessis parenduskohti.

Olulisemad tulemused oli need, et mõlema idufirma arendusprotsessi analüüsimises tuli välja võimalikke parenduskohti.

Järeldus RentMarketi toote arendusprotsessi kohta olid järgmised:

- Stand-up koosolekuid rakendatakse vaid formaasuse pärast. Need võib edaspidi efektiivsuse huvides ära jätta.
- Rohkem oleks vajalik liikuvuse põhimõtet järgida, et vältida analüüsi kirjutamist funktsionaalsustele, mida ei võetagi kasutusele.

Järeldus Fleepi toote arendusprotsessi kohta olid järgmised:

- Stand-up koosolekut võiks rakendada, et arendaja vaheline kommunikatsioon oleks suurem. Nii on võimalik arenduste järjekorda paremini planeerida.
- Proovida rakendada *Test-Driven Development*'i, et vältida vigu avalikus rakenduse versioonis. Küll aga kui selgub, et selle rakendamine aeglustab liialt arenduse protsessi, siis ehk on mõistlik leppida mõningaste vigadega, kui hoida toote arendusprotsess kiirena.
- Proovida rakendada planeerimismängu praktikat, et vältida olukordi, kus sprindi ajal võetakse liialt palju sprindiväliseid *task*'e arendusse. Nii saaks planeerida sisse ka ootamatud olukorrad, mis alati tekivad.

Kõik lõputööle püstitatud eesmärgid saavutati täies ulatuses. Eesmärgid 1 ja 2 saavutati sellega, et uuriti idufirmade toote arendusprotsessi. Eesmärgid 3 ja 4 saavutati sellega, et tehti üldistusi ja leiti kummagi idufirmade toote arendusprotsessi kohta parenduskohti.

Üldine järeldus ja nõu alustavale idufirmale oleks see, et teadvustada endale, kui palju pööratakse tähelepanu kommunikatsioonile ning vajadusel seda suurendada ja vähendada.

Lisaks pöörata tähelepanu operatiivsele planeerimisele ja arvestada rohkem ootamatute ja kiireloomuliste ülesannete tekkimistega.
Kolmas nõu puudutaks funktsionaalsuste elutsüklit ja iga funktsionaalsuse liikuvust. Idufirma äriline osa on liialt liikuv, et mingit funktsionaalsust liiga pikalt ette ära analüüsida.

Paremini oleks saanud analüüsida idufirmade toote arendusprotsesse nii, et oleksin olnud ühe täistöönädala arenduse juures ning näinud arendust, kommunikatsiooni ja koosolekuid oma silmaga ning sealjuures teinud märkmeid. Küll aga võib väita, et antud töö raames saavutati eesmärgid piisava detailsusastmega.

Summary

The aim of the work was to investigate on the agile software development practices, their applications in Start-up companies and if the applications were justified or not. The next aim was to draw conclusions and find potential points to better the software development process.

The main issue of the work was to analyse if the software development process of a start-up follows the practices of a agile software development methodology. When a practice is analysed in the context of a start-up, then the next step was to find out if using or not using a specific practice is reasonable or not. In some cases, a practice not used, but upon analysing the use of it, it would be reasonable to take it into use. Although, in some cases, a practice is used just because it is a habit, but would prove irrational upon analysis.

For the practices, which use were not reasonable, were analysed further and suggestions for bettering the software development process were made.

The outcome of the work was that both start-up companies had improvement points in their software development process. Some of the improvement points were about taking some practices to use, whereas some improvement points suggested to not use a certain practice when considering the nummber of developers and how big the company is. Altogether, five improvement points were brought out. Listing them and suggesting to implement them was the biggest outcome of this work.

In general, for a start-up, which is just starting out, the advice would be to acknowledge the level of communication. Try to analyse if it's too much or too little, because in both cases it's a waste of resource. Also, when planning, take into account that there are always unexpected and urgent tasks that emerge. The third advice would be to keep every functionality block in constant movement and not write the analysis for a functionality too much before it actually reaches development.

Kasutatud kirjandus

[1] Riigiteataja, „seletuskiri äriseadustiku muutmise seaduse eelnõu juurde,“ 2014. [Võrgumaterjal]. Available: http://www.riigikogu.ee/?op=emsplain&page=pub_file&file_id=4b1938bc-234f-43d8-aa47-6ca0cbf64d37&.

[2] A. Manifesto, „Agiilse tarkvaraarenduse manifest,“ 2001. [Võrgumaterjal]. Available: http://agilemanifesto.org/iso/et/principles.html.

[3] „Scrum.ee koolituskeskkond,“ [Võrgumaterjal]. Available: http://www.scrum.ee/miks-scrum.

[4] M. Cohn, „Mountain Goat Software,“ 2014. [Võrgumaterjal]. Available: http://www.mountaingoatsoftware.com/agile/scrum/overview.

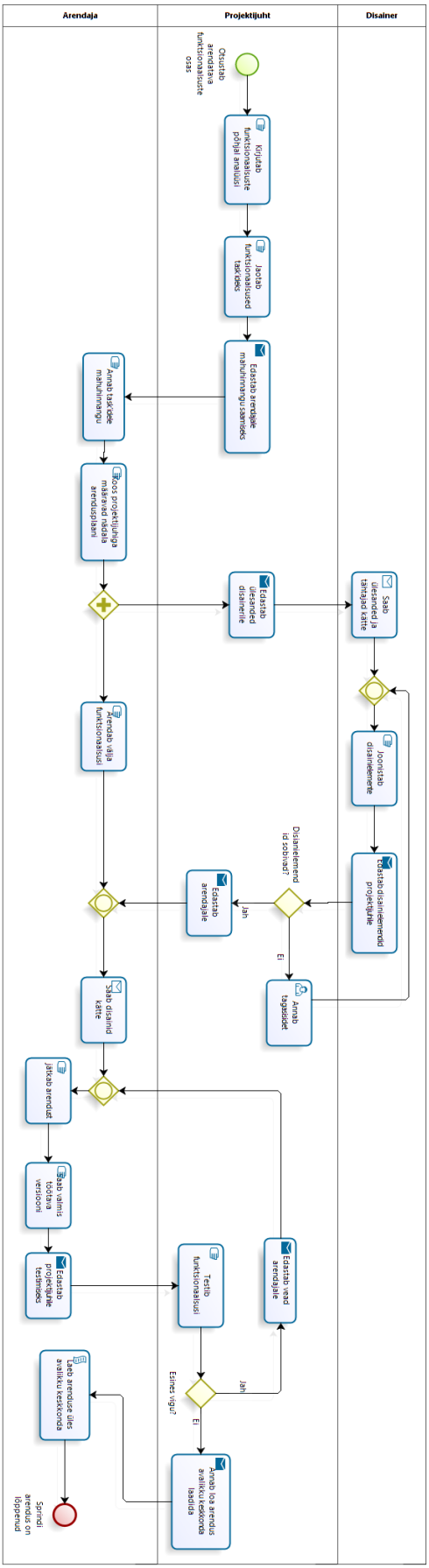
[5] J. Siilivask, „Agiilsete metoodikate rakendamine IT töörkooraldussüsteemi loomisel,“ www.cs.tlu.ee/teemad/get_file.php?id=281, 2014.

[6] P. Brodzinski, „Software Project Management,“ 2009. [Võrgumaterjal]. Available: http://brodzinski.com/2011/08/map-process-to-kanban-board.html.

[7] R. Jeffries, „Agile Software Development Resource,“ [Võrgumaterjal]. Available: http://xprogramming.com/what-is-extreme-programming/.

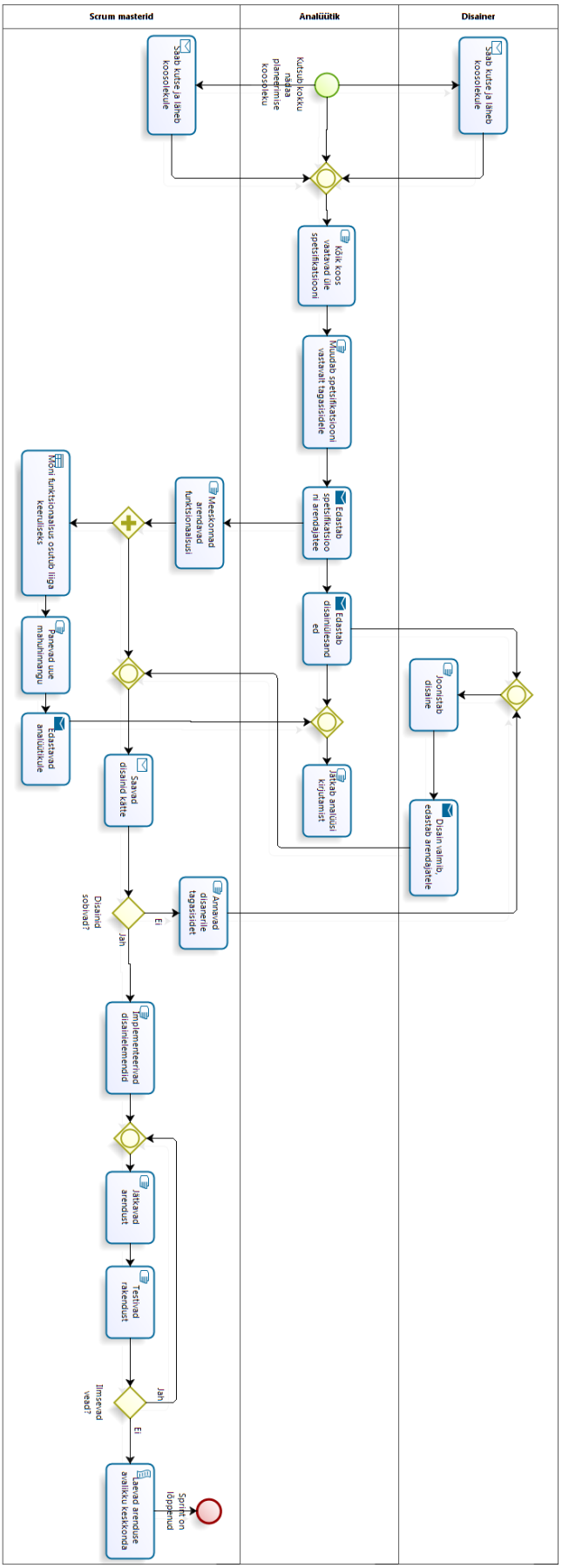
[8] „Software Systems and Applications,“ [Võrgumaterjal]. Available: http://www.ssa-outsourcing.com/services/project-management/.

Lisa 1



Joonis 5 RentMarket arendusprotsessi kirjeldus

Lisa 2



Joonis 6 Fleep arendusprotsessi kirjeldus