TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Informaatikainstituut

Infosüsteemide õppetool

Arendusprotsessi täiustamine ja juurutamine ettevõttes ByteLife Solutions OÜ

Magistritöö

Üliõpilane: Anna-Liisa Roomet

Üliõpilaskood: 132416 IABM

Juhendaja: Gunnar Piho

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 eesmärgiks on analüüsida ettevõtte ByteLife Solutions OÜ tarkvaraarendusprotsessi, leida peamised murekohad protsessis ja täiustada protsessi nii, et loodav tarkvara oleks kvaliteetsem ja usaldusväärsem.

Töös leitakse, et arendusprotsessis on hetkel vähe tagasisidet. Samuti on dokumenteerimist vähe ja puudulik on hilisem kontroll. Seetõttu ei saa protsessi täiendada nii, et see muutuks kvaliteetsemaks.

Töö tulemuseks on täiustatud tarkvaraarendusprotsess ByteLife's, mis on vastavuses CMMI teise taseme nõuetega ja ettevõttes kasutusel olevate arendusmetoodikatega. Uus protsess võimaldab protsessi jälgida ja täiustada.

Lõputöö on kirjutatud eesti keeles ning on kirjutatud 74-l leheküljel, millest 5 on lisad. Töös on koos sissejuhatuse ja kokkuvõttega 9 peatükki, 14 joonist ja 10 tabelit.

Abstract

The aim of the research is to analyze the software development process of ByteLife Solutions OÜ, to find the main problem areas and to improve it to increase the quality and reliability of the product developed.

In the research, it was found that there is not enough feedback in the development process. Too little documentation is done and the control later is inconsistent, the research has shown. The abovementioned problems make the process to lack increase in quality.

The result of the work is the improved software development process in Bytelife, valued against the capability maturity defined level of CMMI: The new process is traceable and improvable, that is quality control.

The thesis is in Estonian and contains 74 pages of text including 5 pages of extras, 9 chapters including introduction and summary, 14 figures, 10 tables.

Lühendite ja mõistete sõnastik

ByteLife	***ByteLife Solutions OÜ*** Ettevõtte nimi
CMMI	**Capability Maturity Model Integration** Küpsuse mudel, kogum parimaid praktikaid, et aidata ettevõtetel täiustada oma protsesse nii, et toote kvaliteet paraneks
CMMI-DEV	**CMMI for development** arendusalane CMMI
SEI	**Software Engineering Institute** Tarkvaratehnika instituut Carnegie Mellon Ülikooli juures
mantisBT	***mantisBT*** Vigade raporteerimiseks mõeldud programm, mida kasutatakse ka ByteLife's.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

ByteLife Solutions OÜ on infotehnoloogia ettevõtte, mis tegeles varem põhiliselt infrastruktuuri loomisega ja seadistamisega, samuti võrguseadmete ja riistvara müügiga. Varasemalt tegeleti ka automatiseerimisega, millest on paari aasta jooksul välja kasvanud tarkvaraarendus. Hetkel on ByteLifes teiste tiimide kõrval ka automatiseerimise tiim, kes tegeleb igapäevaselt tarkvaraarendusega. Selles töös keskendungi automatiseerimise tiimile.

Kuna tarkvaraarendus on ettevõttes suhteliselt uus valdkond, siis alles katsetatakse ja proovitakse, milline meetod on parim. Siit aga tulenebki probleem, et tarkvara pole alati nii kvaliteetne nagu soovitakse. Paljud vead tulevad välja arenduse hilises faasis, mistõttu kulub nende parandamisele rohkem ressurssi ja kui vead on jõudnud juba kliendini, siis kahjustab see ka mainet.

1.2 Eesmärk

Töö eesmärgiks on analüüsida praegust arendusprotsessi ByteLife's ja leida protsessis kohad, mis vajavad täiustamist. Teine eesmärk on välja töötada arendusprotsess ByteLife'i jaoks, mis muudaks tarkvara kvaliteetsemaks ja usaldusväärsemaks. Kolmandaks eesmärgiks on protsessi täiendada testimisega.

1.3 Metoodika

Praeguse protsessi hindamiseks ja analüüsimiseks kasutatakse CMMI mudelit. Testimisprotsess koostatakse uurimuste, artiklite, konspektide ja raamatute põhjal kirjeldatud testimismeetodite analüüsi alusel. Samuti kasutatakse uue protsessi välja töötamisel CMMI standardit ning Scrumi ja Kanbani põhimõtteid.

1.4 Ülevaade tööst

Töö järgmises osas kirjeldatakse arendusprotsessi hetkeolukorda firmas ByteLife Solutions OÜ. Ära on toodud detailne protsessikirjeldus projekti arendusetappide kaupa. Kirjeldatud on kasutatavaid programme ja meetodeid, protsess on ka ära toodud mudelina.

Arendusprotsessi ja selle tulemuse kvaliteedi parandamiseks on protsessi vaja võrrelda usaldusväärse mõõdikuga, et leida võimalikud puudused ja seejärel lahendused. Organisatsioon ei saa liialt ootama jääda vaid tublidele töötajatele, vaja oleks protsessi, mille toimimine tagaks kvaliteedi püsimise ka siis, kui mõni olulise kogemusega inimene kollektiivist lahkub. Töö kolmandas osas antaksegi ülevaade CMMI-DEV meetodist, mis on SEI (Software Engineering Institute) loodud raamistik, mis hõlmab endas parimaid praktikaid ja mis aitab ettevõtetel välja töötada kvaliteetseid tarkvara arendusprotsesse.

ByteLife´i arendusprotsessi hinnatakse neljandas osas just CMMI-DEV meetodile tuginedes ning leitaksegi ByteLife´i küpsuse ja võimekuse taseme parandamiseks vajalikud eesmärgid ja tegevused. CMMI järgi selgus, et vaja oleks rohkem dokumenteerida, jälgida protsessi ja teha kontrolle. Samuti selgus, et firma enda töötajad viitasid eelkõige puudusele testimisel, seetõttu keskendutaksegi töös testimisprotsesside täiustamisele ja parendamisele.

Viiendas osas kirjeldatakse enamkasutatavaid testimise meetodeid, mida oleks võimalik kasutada arendusprotsessi täiustamisel. Kuuendas osas täiendatakse nende abil protsessi. Selles osas on ka ära toodud uus protsessikirjeldus, selle sobivust on võrreldud CMMI-ga ja ByteLife’s kasutatavate arendusmetoodikatega. Protsessi täiendatakse ühiktestide, testikirjelduste ja küsimustike loomisega. Viimast kahte kasutatakse funktsionaalsuse testimiseks. Samuti võetakse kasutusele testimine programmikoodi põhjal ja luuakse automaatsed kasutajaliidese testid.

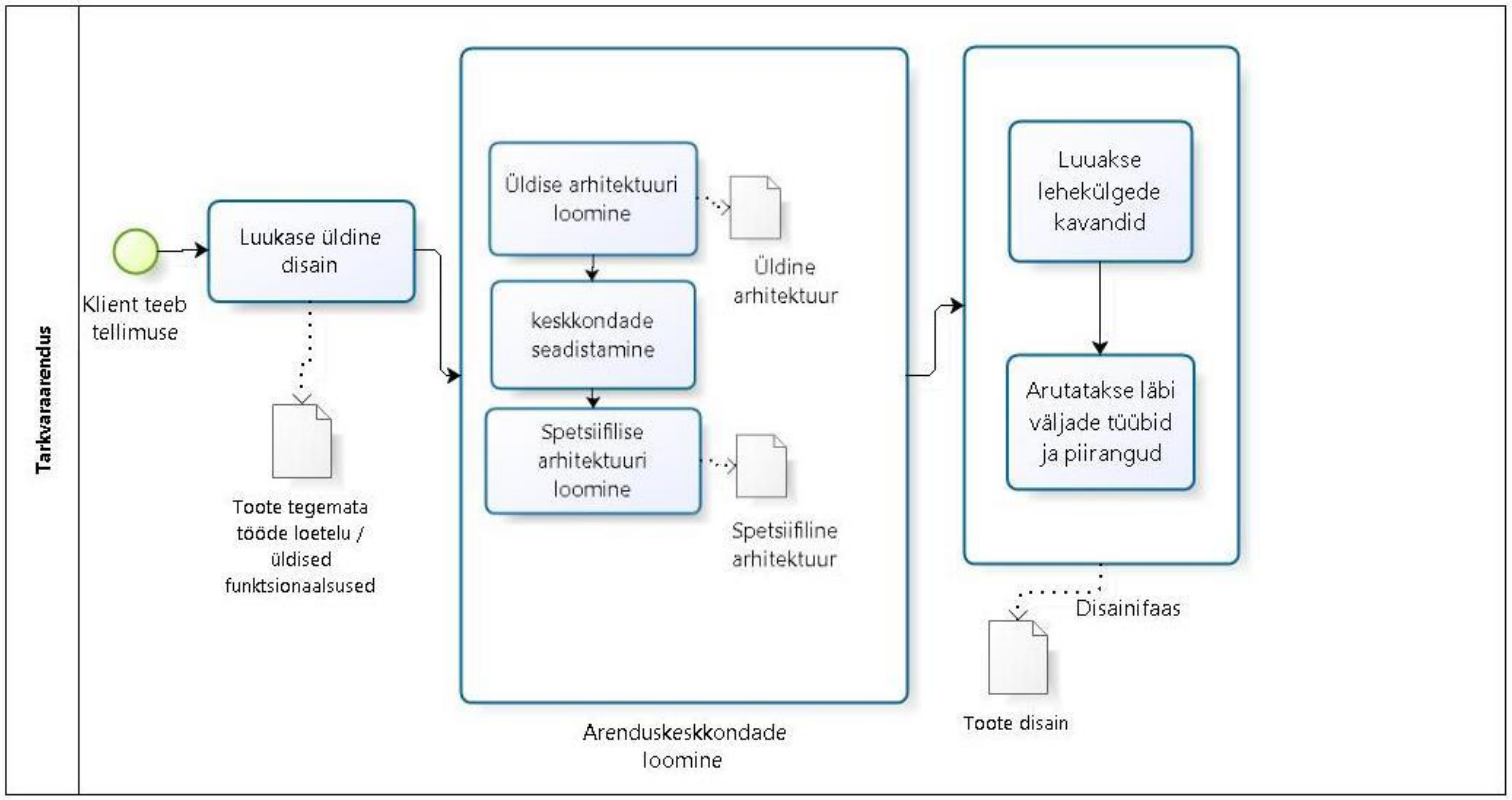
Seitsmendas osas antakse juhiseid muudatuste juurutamiseks ja tuuakse välja punktid, millega tuleb ByteLife’s uue protsessi kasutusele võtmisel arvestada. Selle osa lõpetuseks tuuakse välja, mida on ByteLife juba jõudnud kasutusele võtta ja mis veel tegemist vajab.

Töö kaheksandas osas võrreldakse tehtud tööd teiste analoogsete töödega.

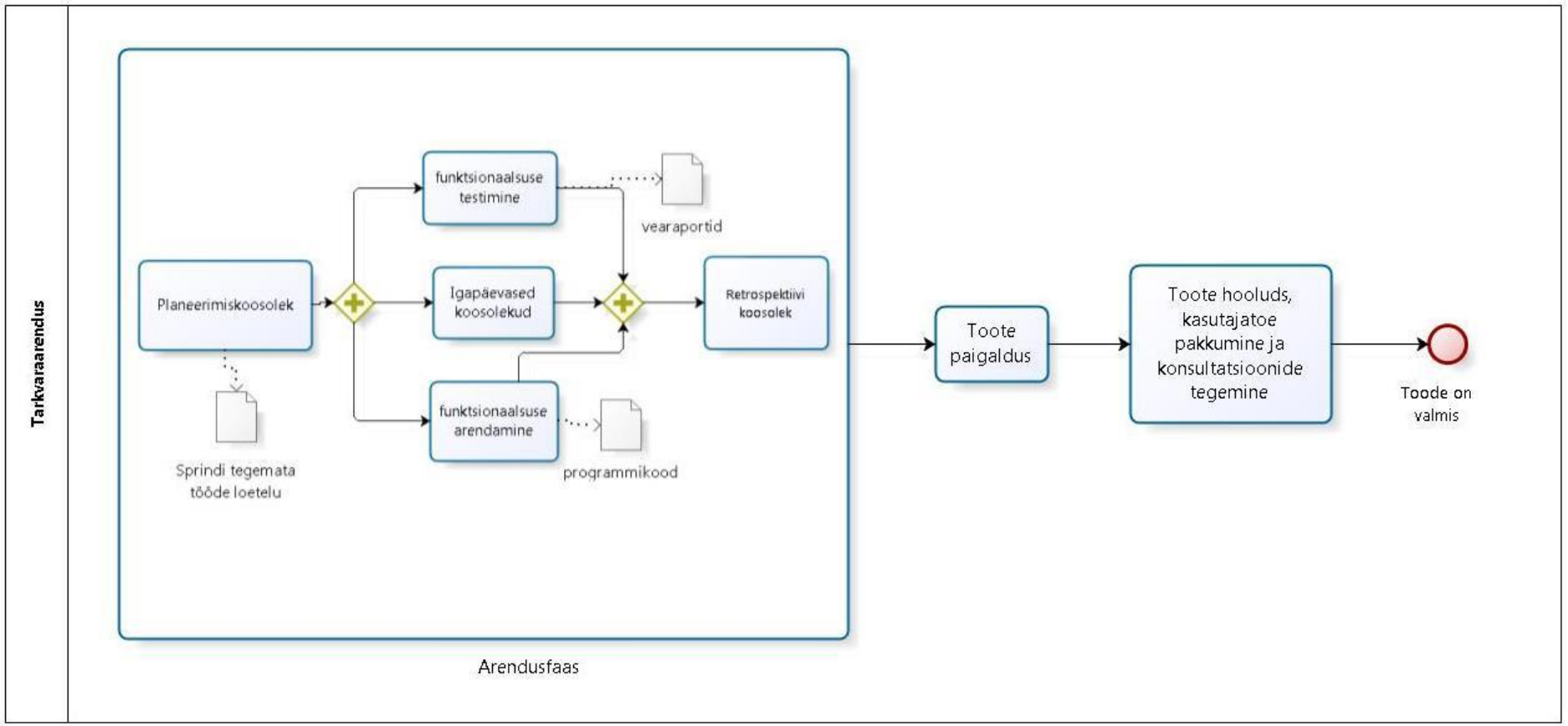
2. Arendusprotsessi kirjeldus

Tarkvaraarendus algab ettevõttes ByteLife tavaliselt sellest, kui keegi ettevõttesiseselt leiab, et oleks vaja luua mingi toode või siis kui klient tellib mingi konkreetse tarkvara. Hetkel on ByteLife´s käsil sellise toote arendus, mis on välja kasvanud meeskonna enda ideest, samas kõik ülejäänud projektid on konkreetsete klientide tellimused. Arendusprotsess on mõlemal juhul sarnane, üldiselt kirjeldatakse töös konkreetsele kliendile loodava tarkvara arendusprotsessi, aga kui see erineb ettevõtte enda initsiatiivil loodava toote arendusprotsessist, siis on see täiendavalt ära toodud.

Järgnevalt on arendusprotsess ära toodud mudelina, millele järgneb detailsem seletus. Joonistel on protsess parema loetavuse saavutamiseks kahte ossa jaotatud. Esimesel joonisel (Joonis 1) on kujutatud protsessi selle algusest kuni disainifaasi lõpuni, teisel joonisel (Joonis 2) on protsess arendusfaasi algusest projekti lõpuni.



Joonis 1 Tarkvaraarendusprotsess enne – osa 1



Joonis 2 Tarkvaraarendusprotsess enne - osa 2

Kõigepealt saab kliendiga kokku müügimeeskond, kes saab kliendiga suheldes teada, mida klient sooviks või millised on nende vajadused, nõudmised ning huvid. Tavaliselt on kaasas ka tehniline konsultant või arhitekt, kes müügimeeskonda aitab. Selles etapis saab selgeks üldine idee ja eesmärk, millega edasi minna. Seejärel arutatakse idee läbi ettevõttesiseselt, kaasates kõiki asjaosalisi. Müügitiim kirjeldab tehnilisematele inimestele, mida klient soovis ja milline eesmärk tuleb saavutada ning koos pannakse paika esialgne tehniline plaan: mõeldakse läbi, kas toodet on võimalik tehniliselt realiseerida, kui kaua see aega võtab, millised on nõuded ja põhifunktsionaalsus. Üldjoontes pannakse kokku üldine disain. Sellega tegelevad arhitekt ja projektijuhid. Disaini eest vastutab toote omanik. Selle etapi lõpuks tuleb saada nimekiri (list) üldistest funktsionaalsustest, mis kirjeldatakse toote tegemata tööde loetelus, samuti peaks olema selge, millal töödega on võimalik alustada ja millised funktsionaalsused on prioriteetsemad.

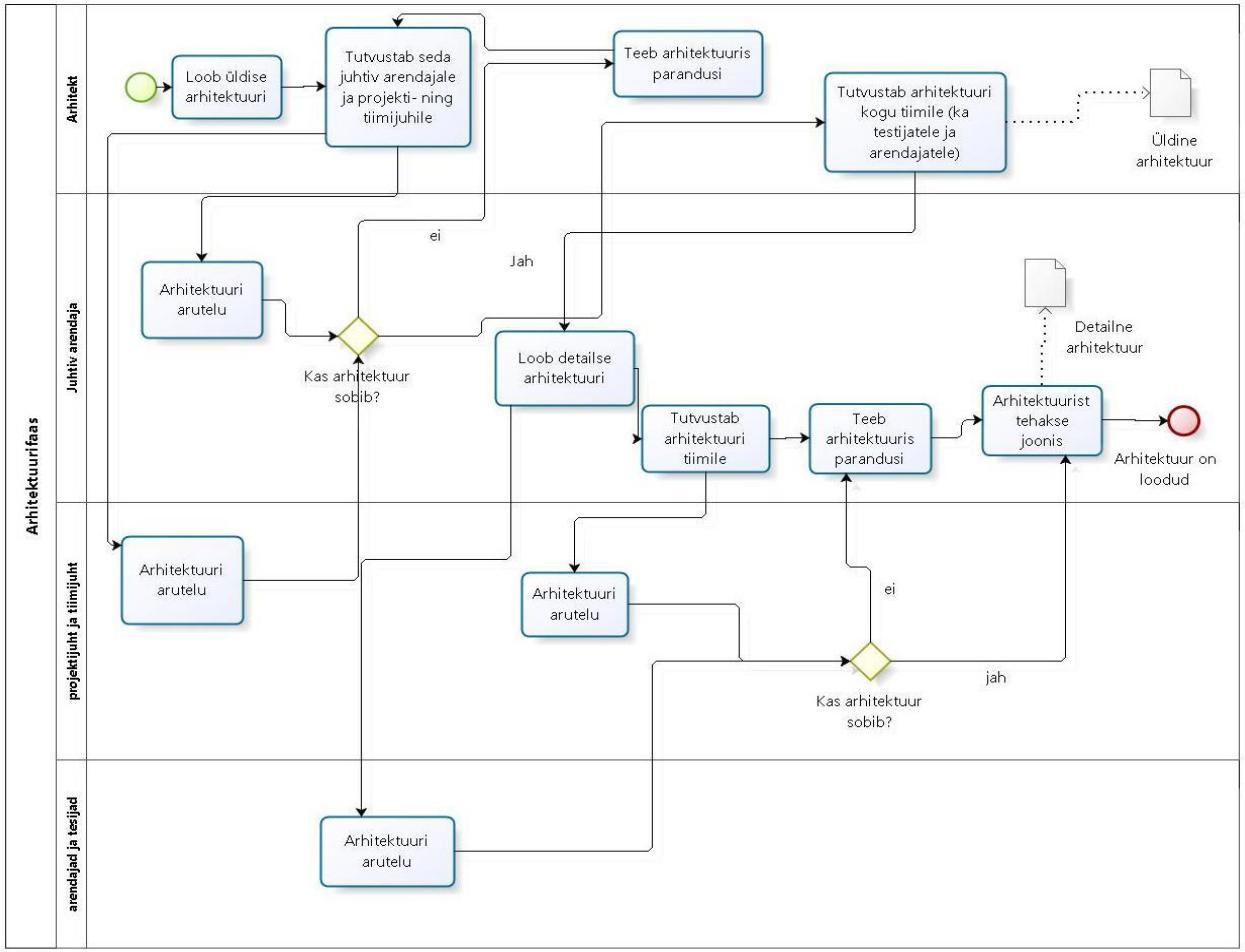
Seejärel esitletakse disaini kliendile, arutatakse veelkord läbi, kas kirjapandud funktsionaalsused on need, mida klient soovis, vajadusel neid kirjeldusi muudetakse, lisatakse või kustutatakse. On võimalik, et tuleb muuta ka prioriteetsust. Kui klient on otsustanud, et ta tahab toodet vastava funktsionaalsusega, siis sõlmitakse leping, kus on täpselt kirjas, mida looma hakatakse. Kliendiga lepitakse tavaliselt ka kokku, kui palju ta tahab toote arenduses osaleda ehk siis see, kui tihti talle tagasisidet antakse ja realiseeritud funktsionaalsusi näidatakse.

2.1 Arenduskeskkondade loomine

Järgmise etapina tuleb konkreetselt läbi mõelda toote arenduskeskkonnad, toote üldine arhitektuur ja tehnoloogiad ning raamistikud, mida kasutama hakatakse. Üldise arhitektuuri loomisega tegeleb tavaliselt tiimijuht, kes on sel hetkel arhitekti rollis. Tavaliselt käib see etapp nii, et arhitekt mõtleb enda jaoks läbi, milline on toode, kellele see on mõeldud ja läbi selle mõtleb välja, millised on nõudmised tootele: kas on vaja andmebaasi, kui on, siis millist kasutada, millised nõuded on serverile ja nii edasi. Lühidalt öeldes, arhitekt koostab esialgse arhitektuurilahenduse, mida seejärel ülejäänud meeskonnale tutvustatakse. Sel hetkel ei kaasata reeglina veel testijaid ega arendajaid, välja arvatud juhtivarendajat, kes järgmises etapis detailse arhitektuuri koostab. Seejärel saab ülejäänud tiim teha ettepanekuid selle kohta, mida peaks muutma ja mis on juba hästi. Arutelu tulemusena peaks valmima toote üldine arhitektuur, mille eest vastutab arhitekt.

Kui arhitektuur on paigas, suurendatakse tiimi nii, et sellega liituvad ka arendajad ja testijad. Neid viiakse just selleks toimuval koosolekul kogu eelnevaga kurssi ja seejärel on juhtivarendaja ülesandeks välja mõelda detailne arhitektuur ning vajadusel ka raamistikud ja tehnoloogiad, mida hakatakse kasutama. See toimub sarnaselt üldise arhitektuuri loomisele: arendaja mõtleb enda jaoks projekti läbi ja teeb vastavalt esialgse plaani, mis kogu tiimiga läbi arutletakse. Sellest valmib tavaliselt ka joonis, mis pannakse siseveebi ülesse. Vajadusel tehakse muudatusi ja parandusi ning tulemuseks on madalama taseme arhitektuur, mida on võimalik hiljem ka muuta ja täiendada.

Järgenvalt on arhitektuurifaasi tegevused ära toodud joonisel.



Joonis 3 Arhitektuurifaas

2.2 Disainifaas

Kui arhitektuur on paigas, siis hakatakse mõtlema tarkvara kasutajaliidese peale. Tehakse koosolekuid, kus mõeldakse välja ja joonistatakse veebilehtede kavandid. Pannakse paika, milliseid lehti on vaja, juhul kui tegemist on veebilehega, ja millised need välja näevad, sealhulgas mõeldakse välja ka väljade nimed ja tüübid ning vajalikud piirangud. Disainikoosolekutel osaleb kas terve tiim või siis osa tiimist, see sõltub projektist.

2.3 Arendusfaas

Pärast disaini loomist algab tavaline arendustsükkel. Hetkel kasutatakse Scrumi [1], millele on lisatud ka Kanbani [2] põhimõtteid. Arendustsükkel on jagatud iteratsioonideks, mis kestavad kaks nädalat. Sprint algab planeerimiskoosolekuga ja lõpeb retrospektiiviga. Kui on vajadus näidata demoversiooni, siis tehakse seda planeerimiskoosoleku alguses. Iteratsiooni jooksul on igal hommikul 15-minutiline koosolek, kus iga tiimiliige ütleb, mis ta eelmisel päeval tegi, mis ta täna teeb ja millised mured tal on tekkinud. Samuti tehakse nendel koosolekutel täpsustusi nõuetes, juhul, kui see on vajalik.

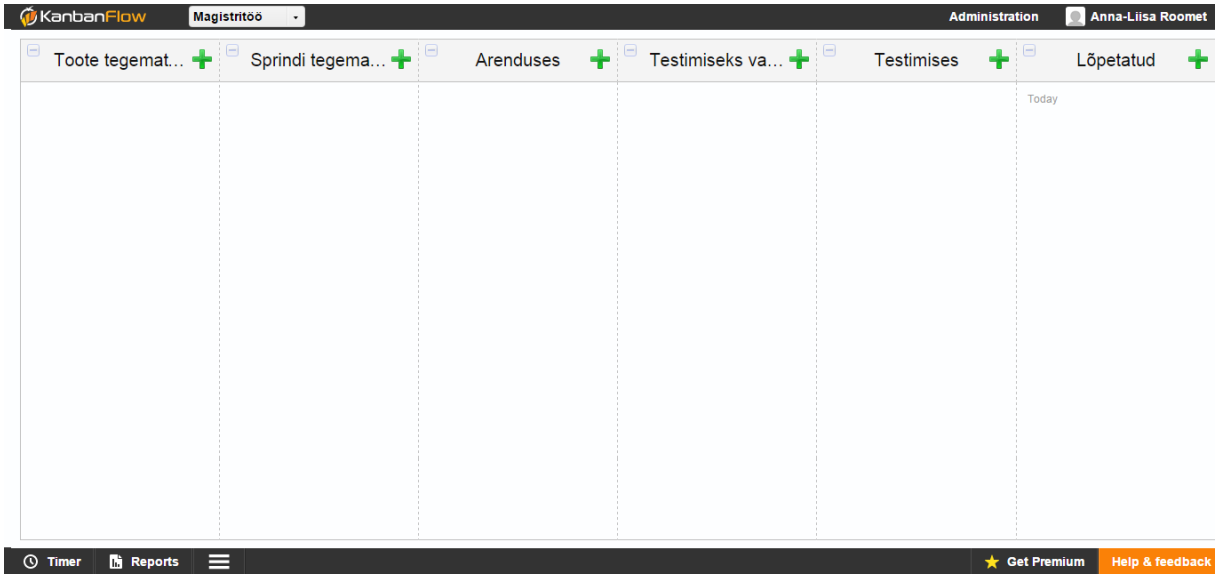
Planeerimiskoosolekul pannakse paika kahe järgneva nädala jooksul tehtavad ülesanded. Kõigepealt võetakse ette varem loodud toote tegemata tööde loetelu, kus keskendutakse kõige prioriteetsematele funktsionaalsustele [3]. Seejärel hakatakse neid väiksemateks ülesanneteks tegema, lähtudes põhimõttest, et iga ülesanne peab olema nii väike kui võimalik, ent siiski peab ülesanne andma lisaväärtust tootele. [4] Samuti tuleb mõelda, kas ülesannet on võimalik testida või mitte. Kui ei ole võimalik testida, siis tuleb talle lisada vastav märge.

Mõnikord, eriti enda algatusel tehtud toodetega, juhtub nii, et enne kui funktsionaalsust saab detailsemalt lahti kirjutada, tuleb teha veel koosolekuid, kus arutatakse, mida konkreetse üldise funktsionaalsuse all mõeldi ja millised on konkreetsed sammud, kuidas seda teha. Tavaliselt tehakse sel juhul nii, et planeerimiskoosolekul pannakse sprindi tegemata tööde loetellu üks üldisem ülesanne, mis pärast koosolekuid detailsemaks tehakse.

Esimese sprindi jaoks võetakse suvaline hulk ülesandeid, vastavalt arendajate arvamusele, mitu nad valmis jõuavad. Edaspidi võetakse aluseks eelmisel iteratsioonil valmis saadud ülesannete hulk ja kui arendus on kestnud kauem kui kolm iteratsiooni, siis võetakse aluseks kolme viimase sprindi tehtud tööde keskmine [3]. Näiteks kui esimeses sprindis tehti valmis 10 ülesannet,

teises 8 ja kolmandas 11, siis võetakse neljandas sprindis (10+8+11)/3 = 9,6 ehk siis 9 ülesannet. See tähendab seda, et kui funktsionaalsus on jagatud väiksemateks ülesanneteks, siis 9 prioriteetsemat võetakse sprindi tegemata tööde loetellu ja ülejäänud jäävad esialgu toote tegemata tööde loetellu.

Selleks, et oleks parem jälgida, kes mida arendab ja millised ülesanded on sprindi tegemata tööde loetelus ja toote tegemata tööde loetelus ning millised ülesanded on juba valmis, kasutatakse programmi nimega KanbanFlow. [3] Tegemist on justkui Kanbani tahvliga, kus konkreetselt meie projektides on 6 lahtrit nimedega „toote tegemata tööde loetelu“, „sprindi tegemata tööde loetelu“, „arenduses“, „testimiseks valmis“, „testimises“ ja „lõpetatud“ ehk „valmis“ ülesanded (vaata Joonis 4).

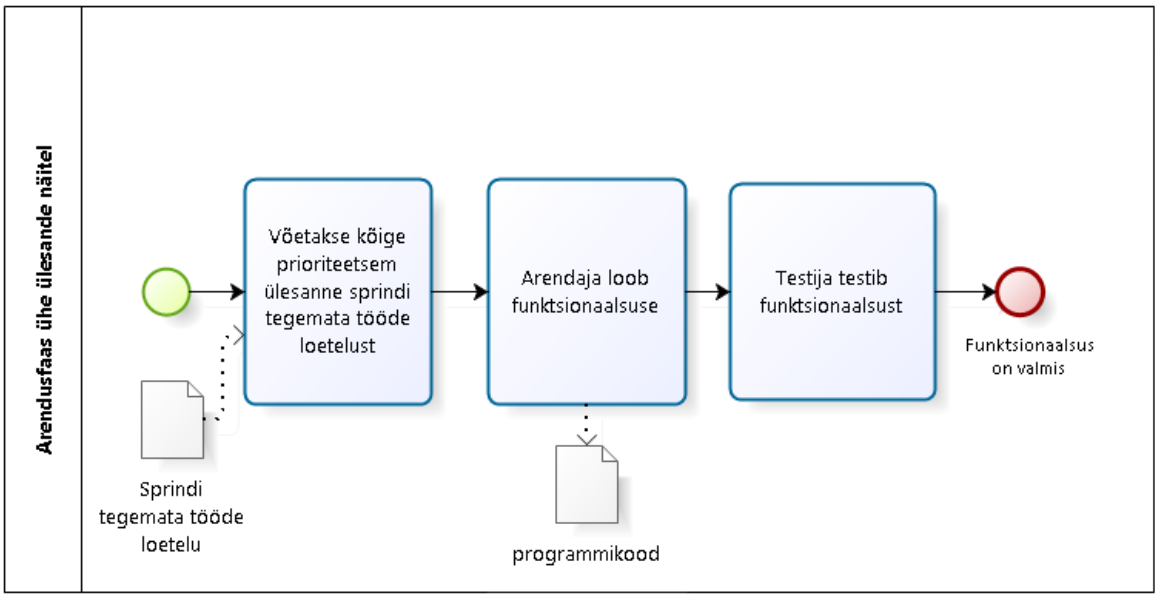


Joonis 4 KanbanFlow tahvel

Süsteem on järgmine: kõigepealt on kõik ülesanded sprindi tegemata tööde loetelus nii, et kõige üleval on prioriteetsemad ülesanded ja allpool on vähem prioriteetsed. Iga arendaja, kellel pole parasjagu midagi teha, võtab kõige ülemise ülesande ja hakkab seda tegema. Sel hetkel tõstab ta ülesande sprindi tegemata tööde loetelust arenduses olevate ülesannete alla ja lisab juurde oma nime. Juhul kui ülesanne on valmis, paneb ta selle testimiseks valmis olevate ülesannete alla. Iga testija, kellel pole parasjagu ülesannet, võtab kõige ülemise ülesande testimiseks valmis olevate ülesannete hulgast ja lisab juurde oma nime nagu arendajadki. Seejärel hakatakse testima.

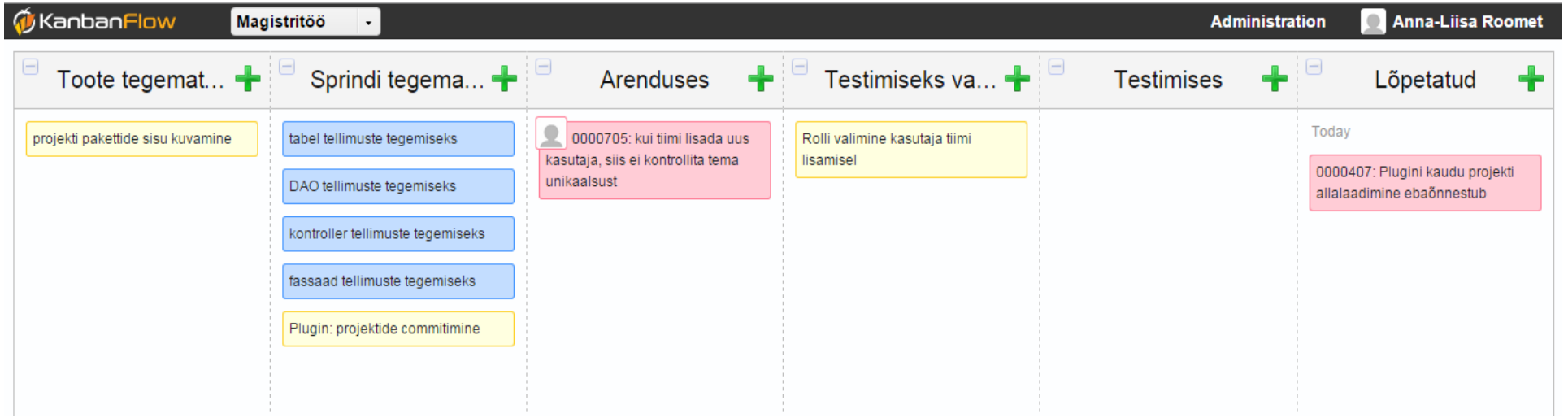
Kui testija veendub, et testitav ülesanne töötab nii nagu ette nähtud, siis tõstab ta selle tehtud tööde listi. See, milline on ülesande õige käitumine, on selgeks tehtud planeerimise koosolekul. Selle eest, et ülesanne oleks tehtud nii nagu peab, vastutab testija. Juhul kui ülesanne ei tee seda, mida vaja, on testija kohustus see tagasi *sprindi* tegemata tööde loetellu tõsta, kust arendajad seda uuesti valida saavad ja seejärel parandada. Juhul kui keegi avastab vea, siis tehakse selle kohta mantisBT-sse vearaport, samuti tuleb viga panna kirja toote tegemata tööde loetellu, kus seda järgmisel planeerimiskoosolekul valida saab.

Järgneval joonisel on ära toodud protsess, mille läbib iga ülesanne.



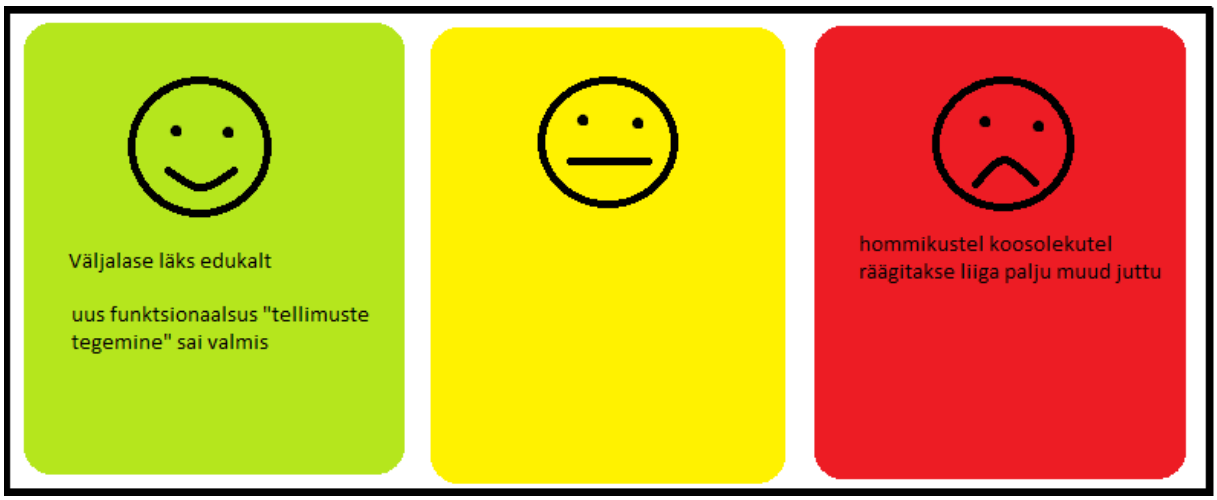
Joonis 5 Arendusfaas ühe ülesande näitel

Järgevalt on näitena ära toodud täidetud KanbanFlow tahvel. Kasutatud on hetkel käimas oleva projekti funktsionaalsusi ja ülesandeid. Magistritöö jaoks on need tõlgitud eesti keelde. Punastes kastides on parandamist vajavad vead, sinised on sellised ülesanded, mida eraldi ei testita ja kollased on tavalised arendusülesanded.



Joonis 6 Täidetud KanbanFlow tahvel

Sprint lõpeb retrospektiiviga. See on koosolek, kus meeskonnaliikmed saavad avaldada arvamust selle kohta, mis läks hästi ja mida tuleb protsessis parandada. Kõigepealt kirjutab iga tiimiliige enda paberile, mis oli tema meelest hästi ja mis halvasti. Sildid pannakse vastavalt lahtritesse: kõik, mis läks hästi, pannakse rohelisse lahtrisse, mis läks halvasti, pannakse punasesse ja mis on vahepealne, läheb kollasesse lahtrisse (Joonis 7) .Hästi läinud tegevused ja omadused pannakse kõrvale nii, et alles jäävad keskmised ja halvad sündmused. Seejärel grupeeritakse sarnased märkused.



Joonis 7 Retrospektiivi tahvel

Gruppidele hakatakse panema punkte, see on vajalik seetõttu, et hiljem võetakse kolm kõige rohkem punkte saanud gruppi ja neile tehakse analüüs, kuidas nendes gruppides olukorda parandada. Iga tiimi liige saab kokku anda kolm punkti, need võib jagada välja ühe kaupa, kaks punkti ühele grupile ja 1 teisele või siis kõik kolm punkti ühele grupile. Kolm punkti antakse ühele grupile juhul, kui tiimiliige arvab, et selles grupis on olukord väga tõsine ja sellega tuleb kohe tegeleda. Kolmes kõige enam punkte saanud grupis tehakse muudatusettepanekuid, et olukorda parandada. Näiteks kui ülal joonisel on probleem, et koosolekutel räägitakse muud juttu ja kui on tekkinud grupp „koosolekud“, siis võib muudatusena välja tuua, et scrumi meister kontrolliks, et tiimiliikmed teemast kõrvale ei kalduks.

Järgneval retrospektiivi koosolekul vaadatakse kõigepealt üle eelmisel koosolekul kirja pandud muudatused ja kontrollitakse, kas neid on rakendatud. Kirjapaneku eest vastutab toote omanik, aga iga konkreetse ülesande juurde pannakse samuti konkreetne vastutaja.

Mõnikord selgub hilisemas faasis, et esialgu taheti funktsionaalsusi, millest hiljem loobutakse või mida tuleks muuta. Kui leping on varasemalt sõlmitud, siis on kaks varianti, kas

funktsionaalsus ikkagi luuakse või siis räägitakse kliendiga läbi, kas on võimalik teha üks funktsionaalsus teise asemel. Sel juhul peavad need olema enam vähem sama suure mahuga. Kui sellises asjas jõutakse kokkuleppele, siis arutletakse planeerimiskoosolekul uuesti üle prioriteedid ja lisatakse uus funktsionaalsus oma õigele kohale.

Rohkem muudatusi tuleb ette juhul, kui toode on enda ettevõtte initsiatiivil loodud ja otsest klienti veel ei ole. Iseenesest on protsess aga mõlemal juhul sarnane. Planeerimiskoosolekul arutatakse läbi, kui prioriteetne on uus funktsionaalsus ja lisatakse ta toote tegemata tööde loetelus õigele kohale. Edasi toimib arendus samamoodi nagu varem seletatud.

2.4 Paigaldus

Kliendile loodud tooted on valmis juhul, kui kõik lepingus olevad funktsionaalsused on valmis. Kui toode on valmis, tuleb tegeleda selle paigaldamisega ja hiljem kasutajatoe pakkumisega. Paigaldust saab reeglina teha kahel viisil: kaugelt ehk üle võrgu või siis kliendi juurde kohale minnes. Mõlemal viisil pakutakse tavaliselt ka konsultatsiooni. Kohale minnes tehakse ka koolitus. Paigalduse, koolituse ja konsultatsiooniga tegeleb ettevõttes üks ja sama inimene, kes on administraatori ja koolitaja rollis.

Pärast paigaldust saab klient toodet proovida ja testida. Juhul, kui kliendi keskkonnas tekivad vead, saadab klient vearaporti, millele järgneb veaparandus meie poolt. Parandus toimub samamoodi sprindina nagu tarkvaraarendus. Kui vead on parandatud, paigaldatakse uus versioon kliendile, kes seda uuesti testib. Nii jätkub protsess, kuni vead on parandatud. Kui hilisemas faasis mõni viga tekib, siis on olemas kasutajatugi, kes sellega tegeleb. Ettevõttes on kaks või kolm inimest, kellele kliendid saavad e-kirju saata. Juhul, kui kirja saaja ise ei suuda veale lahendust leida, siis saadab ta probleemi kirjelduse tervele ettevõtte arendustiimile Skype'i teel. Kui kohe vea põhjust ei teata, siis tavaliselt on keegi vastutajaks, kes veaga tegeleb. Olenevalt vea suurusest ja olulisusest otsustatakse, mida sellega teha. Kui tegu on kriitilise veaga, siis läheb see kohe parandusse, vähem kriitilised vead pannakse aga kirja mantisBT-sse.

Hetkel on toodete nimekirjas ka veebileht, mille puhul otseselt paigaldust ei ole. Selliste toodete puhul toimuvad vastavalt vajadusele veebilehe uuendused. Selle eest, et toode toimiks enne üles panemist nii, nagu hetkel vaja, vastutab juhtivarendaja ja selle eest, et uus versioon õigesti üles pannakse, vastutab tiimijuht.

2.5 Hooldus ja konsultatsioon

Ka veebilehe puhul pakutakse kasutajatoe teenust. Kõik, kellel tekib probleeme või küsimusi, saavad ühendust võtta tiimiga. Kolmel inimesel tiimist on ligipääs e-kirjadele ja protsess toimib samamoodi nagu kliendipõhise rakenduse kasutajatoe puhul. Edaspidist hooldust pakutakse tasulistele *enterprise* versioonidele, muul juhul pakutakse ainult koolitusi, konsultatsioone ja kasutajatuge.

2.6 Rollid ja faasid

Tabelis 1 on ära toodud, missuguses faasis missugune roll tegev on.

Tabel 1 Faasid ja rollid esialgses protsessis

Rollid/faasid	**Projekti algus**	**Arhitektuurifaas**	**Disainifaas**	**Arendusfaas**	**Paigaldus**	**Hooldus**
Müügitiim	X					
Projektijuht	X	X	X	X	X	X
Tiimijuht	X	X	X	X	X	X
Keskkondade seadistaja		X				
Juhtiv arendaja		X	X	X		
Arendaja			X	X		
Testija			X	X		
Paigalduse ja hooldusega tegelev isik					X	X

3. CMMI

Ettevõttel on soov muuta oma loodud tooted kvaliteetsemaks. Selleks tuleb aga teada, mis on hetkel valesti ja kus protsessis on vaja teha täiendusi. Selleks, et hinnata, millistes etappides ja missuguseid muudatusi teha, uuritakse töös hetke arendusprotsessi vastavust CMMI tasemetele.

CMMI on SEI (Software Engineering Institute) loodud raamistik, mis aitab ettevõtetel protsesse küpsemaks muuta. CMMI mudelid ise ei ole protsessid ega ka konkreetsed protsesside kirjeldused, need on lihtsalt juhtnöörid. Konkreetne protsess sõltub siiski konkreetsest ettevõttest. On olemas kolm mudelit: CMMI tarkvaraarendusele (CMMI for Development), CMMI teenustele (CMMI for Services) ja CMMI sisseostmisele (CMMI for Acquisition). Selle töö raames vaadatakse edaspidi CMMI mudelit, mis on mõeldud tarkvaraarenduse protsesside jaoks ehk CMMI-DEV'i. [5]

3.1 CMMI-DEV

CMMI-DEV on konkreetselt tarkvaraarendusprotsesside parendamiseks mõeldud mudel. See on kohandatud arendamisele ja katab projektijuhtimise, protsessijuhtimise, süsteemiprojekteerimise ja tarkvaraarenduse. Mudel võimaldab rakendada CMMI parimaid praktikaid, mis aitavad luua kvaliteetseid tooteid ja teenuseid, mis katavad klientide ja kasutajate vajadusi. [5]

CMMI-DEV mudel koosnev 22-st komponendist ehk protsessi valdkonnast.

- Põhjuslik analüüs ja lahendus (CAR)
- Konfiguratsiooni juhtimine (CM)
- Otsuste analüüs ja lahendus (DAR)
- Integreeritud projekti juhtimine (IPM)
- Mõõtmine ja analüüs (MA)
- Organisatsiooniline protsessi definitsioon (OPD)
- Organisatsiooniline protsessi fookus (OPF)
- Organisatsiooniline tootlus/jõudluse juhtimine (OPM)
- Organisatsiooni protsessi käitumine (OPP)
- Organisatsiooniline õppus (OT)
- Toote integreerimine (PI)

- Projekti jälgimine ja kontroll (PMC)
- Projekti kavandamine (PP)
- Protsessi ja toote kvaliteedi kindlustamine (PPQA)
- Kvantitatiivne projektijuhtimine (QPM)
- Nõuete arendamine (RD)
- Nõuete haldamine (REQM)
- Riskijuhtimine (RSKM)
- Tarnelepingu juhtimine (SAM)
- Tehnilised lahendused (TS)
- Valideerimine (VAL)
- Verifitseerimine (VER)

Igal valdkonnal on oma spetsiifiline eesmärk (SG), mis aitab konkreetset protsessi parendada. Spetsiifilist eesmärki aitavad saavutada spetsiifilised tegevused (SP). Kui mitmel valdkonnal on ühine eesmärk, siis nimetatakse seda üldiseks eesmärgiks (GG) ja seda aitavad saavutada üldised tegevused (GP). [5]

Organisatsiooni arenguetapi mõõtmiseks kasutatakse tasandeid. Neid on kahte liiki: üks, mida kasutatakse konkreetse protsessi hindamiseks ja teine, mis on mõeldud protsesside kogumi hindamiseks. Pideva lähenemise protsessi hinnatakse võimekuse tasandiga ja on mõeldud konkreetse protsessi hindamiseks. Protsesside kogumi hindamiseks kasutatakse astmelist lähenemist ja hinnatakse küpsuse tasandiga. Hindamise eesmärk on aru saada, kui hästi organisatsioon või selle protsess toimib. [5]

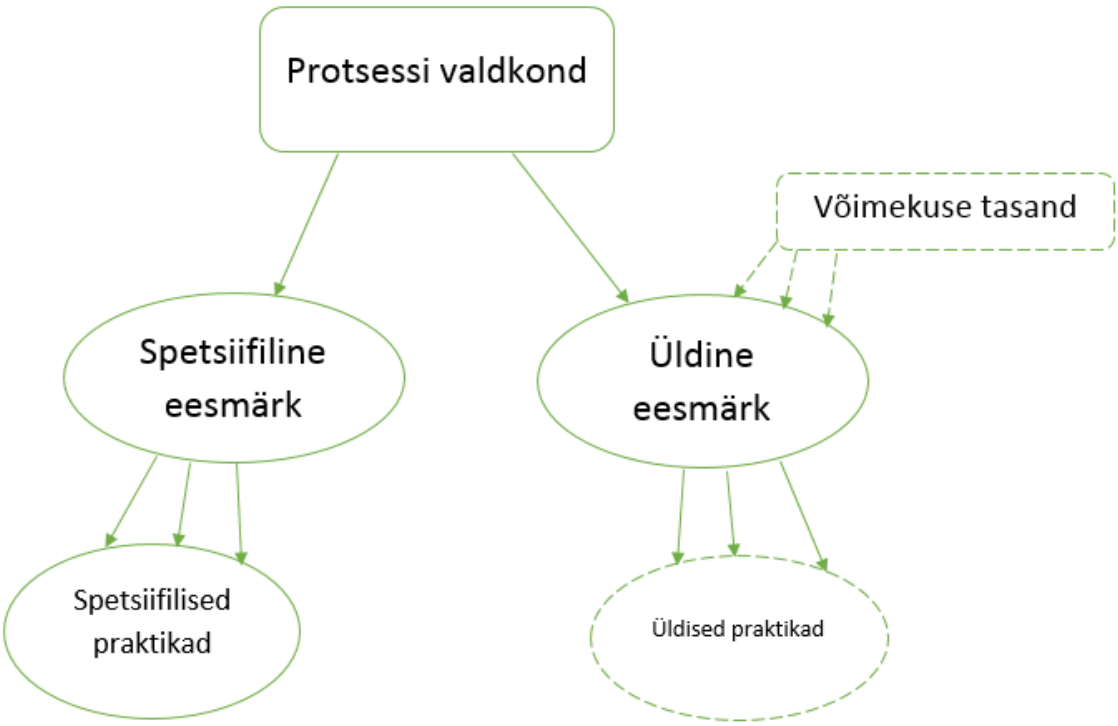
Tabel 2 Võimekuse ja küpsuse tasandite võrdlus [5]

Tasand	Pideva protsessi võimekuse tasand	Astmelise protsessi küpsuse tasand
0	Pooleli	X
1	Tehtud	Algne
2	Juhitud	Juhitud

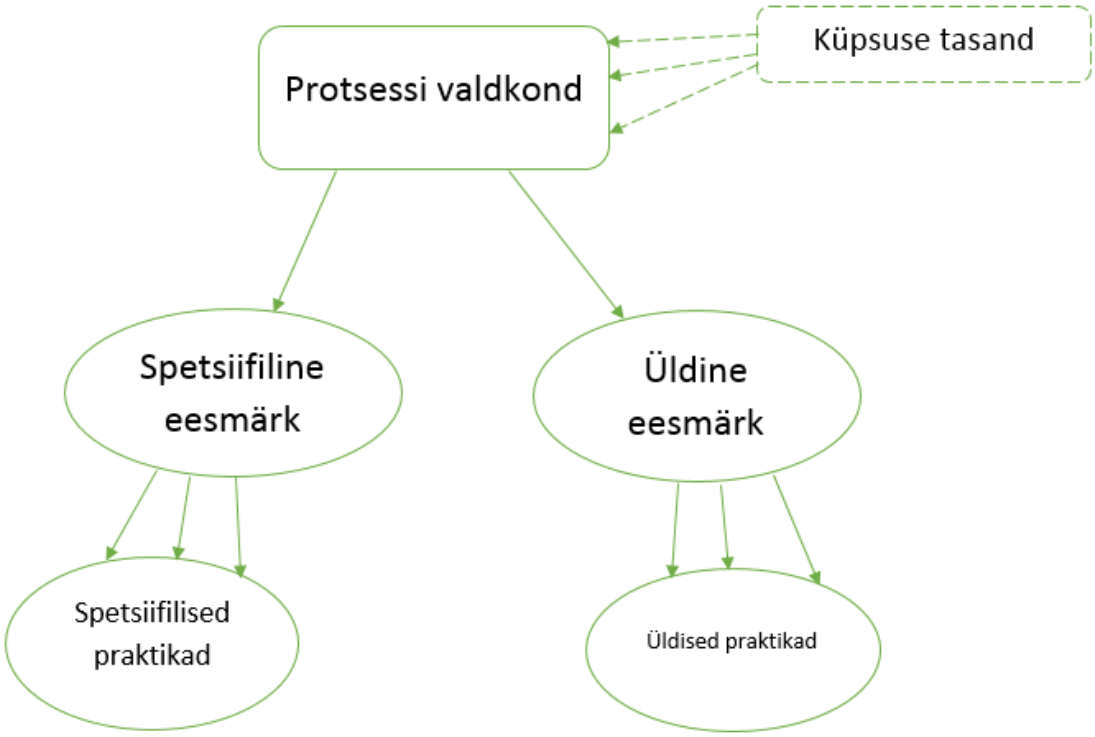
Tasand	Pideva protsessi võimekuse tasand	Astmelise protsessi küpsuse tasand
3	Defineeritud	Defineeritud
4	X	Kvantitatiivselt juhitud
5	X	Optimeeriv

Selleks, et mingit tasandit saavutada, tuleb kõik selle tasandi eesmärgid täita, hoolimata sellest, kas tegemist on võimekuse või küpsuse tasandiga. Mõlemad on oma olemuselt samasugused.

Järgenvalt on ära toodud pideva ja astmelise protsessi struktuurid.



Joonis 8 Pidev protsess [5]



Joonis 9 Astmeline protsess [5]

3.1.1 Tasandid

Võimekuse tasandid

Võimekuse tasand on saavutatud, kui kõik selle tasandi üldeesmärgid on saavutatud. Bytelife eesmärk on organisatsiooni võimekust tõsta, mida tuleb saavutada protsesside küpsuse saavutamisega.

CMMI järgi ollakse võimekuse tasandil 0 juhul kui protsessi ei ole veel üldse loodud või kui seda on tehtud osaliselt. Tasemele 1 jõutakse siis, kui protsess on tehtud ja konkreetse taseme eesmärgid on saavutatud. Sellel tasemel ei ole veel parendamist. Tasemel 2 võib öelda, et protsess on juhitud. On loodud protsess ja selle täitmist kontrollitakse. Vastavalt vajadusele tehakse parandusi. [5]

Kolmanda taseme oluline erinevus tasandiga 2 on, et on olemas projektiülesed organisatsiooni standardid, protsessikirjeldused ja protseduurid, mis on ühised kõigile protsessidele olenemata konkreetsest projektist. Samuti on protsessid kirjeldatud rangemalt kui tasandil 2: selgelt on

kirjas eesmärk, sisend, tegevused, rollid, mõõdikud, kontrollisammud, väljundid, valmimise kriteeriumid. [5]

Võimekuse tasand näitab seda, mis tasemel ollakse konkreetses protsessivaldkonnas. Konkreetses töös vaadatakse valdkonda „protsessi ja toote kvaliteedi kindlustamine“. ByteLife’s on küll loodud protsess tarkvaraarenduseks, ent mitte kvaliteedi kindlustamiseks. Kõik spetsiifilised eesmärgid ei ole täidetud, seega võib väita, et selles protsessis ollakse CMMI järgi võimekuse tasemel 0.

Küpsuse tasandid

Küpsuse tasandid koosnevad seotud konkreetsetest ja üldistest tavadest (praktikatest) eeldefineeritud protsessivaldkondades, mis parandavad organisatsiooni kogu toimimist. [5]

Küpsuse tasand 1. Esialgne

Protsessid on kaootilised, organisatsioon ei paku teotust protsesside stabiliseerimiseks. Ohuks on kulude ja ajakava ületamine ning inimeste kangelaslikkuse ärakasutamine.

Küpsuse tasand 2. Juhitud

Protsesse kavandatakse ja viiakse ellu vastavalt eeskirjale, kasutatakse oskustega inimesi, kellel on piisavalt vahendeid, et luua kontrollitud väljundid, asjaomased osapooled on kaasatud; protsessid on jälgitud, kontrollitud ja üle vaadatud; ja hinnatud vastavus protsessi kirjeldusele. Projekte täidetakse ja juhitakse vastavalt dokumenteeritud kavale.

Asjaomased osapooled on delegeeritud, töö tulemust on sobivalt kontrollitud. Töö tulemus vastab konkreetsele protsessi kirjeldusele, standardile ja protseduurile. [5]

CMMI järgi on veel olemas küpsuse tasandid 3, 4 ja 5, ent antud töös neid ei vaadelda. Kuna on teada, et ByteLife’s puudub protsessi kontroll ja seega on juba teada, et kõik teise taseme eesmärgid ei ole täidetud, siis võib öelda, et ByteLife on esimesel tasemel. Selleks, et saavutada küpsustaset 2, peavad kõik teise tasemega seotud protsessi valdkondade tasemed vastama võimekuse tasandile 2 või 3. Järgnevas tabelis on välja toodud teise taseme protsessi valdkonnad koos kategooriatega.

Tabel 3 Protsessi valdkonnad, kategooriad ja küpsuse tase [5]

Protsessi valdkond	Kategooria	Küpsuse tase
Configuration Management (CM) Konfiguratsiooni haldamine	Tugi	2
Measurement and Analysis (MA) Mõõtmine ja analüüs	Tugi	2
Project Monitoring and Control (PMC) Projekti jälgmine ja kontroll	Projekti juhtimine	2
Project Planning (PP) Projekti kavandamine	Projekti juhtimine	2
Process and Product Quality Assurance (PPQA) Protsessi ja Toote kvaliteedi kindlustamine	Tugi	2
Requirements Management (REQM) Nõuete haldamine	Projekti juhtimine	2
Supplier Agreement Management (SAM) Tarnelepingute haldamine	Projekti juhtimine	2

Kuna kõigis seitsmes valdkonnas vajaliku taseme saavutamine on mahukas töö, siis keskendutakse konkreetses töös protsessi ja toote kvaliteedi kindlustamisele. Alustatakse sellest, sest see on kõige otsesemalt seotud kvaliteedi tagamisega, mis on hetkel ettevõtte suurim murekoht. Seega konkreetse töö eesmärgiks on parendada arendusprotsessi vastavalt

protsessi ja toote kvaliteedi kindlustamise valdkonnale. Selleks, et teada, mida parandada, on vaja teha analüüs, mis on juba olemas ja kus ollakse praegu.

4. Protsessi analüüs CMMI teise taseme põhjal

Järgnevas peatükis analüüsitakse arendusprotsessi vastavust CMMI teisele tasemele.

4.1 Üldised eesmärgid ja tegevused

Üldiseid eesmärke on kolm, kusjuures numbrid vastavad tasemele, nt GG1 on esimese taseme saavutamise eesmärk. CMMI järgi, kui tahetakse saavutada teist taset nagu antud juhul, siis peavad olema täidetud eesmärgid GG 1 ja GG 2. Mõlemal eesmärgil on ka tegevused, mis aitavad eesmärke saavutada.

Järgnevalt on ära toodud eesmärgid ja tegevused ja antud hinnang, kas antud eesmärk on täidetud. Kõik tegevused on võetud CMMI tehnilisest raportist (kasutatud kirjandus [5]).

GG 1 - Protsessivaldkonna spetsiifilisi eesmärke toetab protsess, mis muudab tuvastatud sisendi tuvastatud töötulemiks.

Tabel 4 CMMI esimese taseme üldeesmärgi tegevused

	Tegevus	Olukord ettevõttes
GP 1.1	Luuakse tooteid või teenuseid (vajalik ei ole dokumenteeritud protsessi ega plaani järgimine)	ByteLife tegeleb tarkvara arendusega, mille käigus luuakse uusi tooteid.

GG 2 – Protsess on juhitav

Võib öelda, et kui ollakse saavutanud teine tase, siis on loodud poliitika, mis aitab protsessi järgida ja kontrollib, et seda tehtaks. On olemas plaan, et protsessi järgida, selleks on eraldatud ressursid, paika on pandud vastutusalad ja läbi on viidud vajalikud koolitused. Lühidalt; protsess on kavandatud ja jälgitud.

Tabel 5 CMMI teise taseme üldeesmärgi tegevused

	Tegevus	Olukord ettevõttes
GP 2.1	Luua ja säilitada organisatsiooni poliitika protsessi planeerimiseks ja teostamiseks (printsiibid, direktiivid)	Organisatsioonil on poliitika protsesside planeerimiseks.
GP 2.2	Teha plaan protsessi täitmiseks	Plaan protsessi täitmiseks luuakse. Pannakse paika ressursside jaotumine ja vastutusalad. Protsess vastab nendele ettevõtte poliitikatele, mis on olemas, samuti Eesti Vabariigi seadustele. Kokku on lepitud, et kasutatakse Scrumi ja Kanbani põhimõtteid.
GP 2.3	Varusta vahenditega (raha, oskustega inimesed, tööriistad, programmid)	Projekti planeerimise käigus tehakse vajalik analüüs, kas on piisavalt ressurssi projektiga tegelemiseks. Selle käigus määratakse ka ära, kui palju aega, raha ja tööjõudu projekti peale kulub.
GP 2.4	Määrata vastutaja	Mõned vastutusalad on määratud töölepinguga ja teised vastutusalad pannakse paika vastavalt projektile. Vastavalt Scrumile on projekti vastutaja Toote omanik, tiimi eest vastutab Scrumi meister. Kuigi Scrumi järgi vastutab kõige eest ühiselt tiim, on siiski määratud vastutaja testimise, arendamise ja keskkondade seadistamise jaoks.
GP 2.5	Koolitada inimesi	Kuigi valdavalt tegelevad inimesed sellega, milles nad on tugevad, on siiski sama tööga tegelevate inimeste tase hetkel erinev. Ettevõttes on võimalik tellida raamatuid, otsida enda jaoks koolitusi ja teha sertifikaate, aga ühiseid, ettevõtte poolt pakutud koolitusi ei ole.

	Tegevus	Olukord ettevõttes
GP 2.6	Kontrollida töö tulemust	Hetkel on kontroll töö tulemuse üle nõrk, uue funktsionaalsuse puhul testitakse seda manuaalselt, kirja pannakse ainult vead. Enne väljalaskeid kontrollitakse üle kogu funktsionaalsus. Puuduvad automaatsed testid ja testide raportid.
GP 2.7	Määrata huvigrupid	Huvigrupid on määratud. Klientidega lepitakse projekti planeerimise ajal kokku, kui tihedalt nad projektiga seotud tahavad olla, kui tihti tahetakse uusi funktsionaalsusi näha jne. Kuna igal reedel on üldine koosolek, kus osalevad kõik töötajad ja kus viiakse üksteist kurssi nädala jooksul tehtud töödega, siis on kõik teadlikud üldiselt ettevõttes toimuvaga. Juhtkond arutab projekte ka kinnisemas seltskonnas.
GP 2.8	Kontrollida ja jälgida protsessi	Konkreetse projekti puhul kontrollib toote omanik, et jälgitaks Scrumi ja Kanbani metoodikaid viisil nagu need on protsessis kirjas. Kuid üldisem protsessi kontroll puudub ja standardite mitte-jälgimist ei dokumenteerita.
GP 2.9	Hinnata objektiivselt vastavust standarditele, protseduuridele ja protsessile	
GP 2.10	Tippjuhid vaatavad üle protsessi tulemuse	Kord kuus toimuvad koosolekud, kus juhtkond vaatab läbi kõik käimas olevad projektid.

4.2 Spetsiifilised eesmärgid ja tegevused

SG 1 – hinda objektiivselt tööprotsessi ja töö tulemust

Tabel 6 Protsessi ja toote kvaliteedi tagamise spetsiifilise eesmärgi nr 1 tegevused

	Tegevus	Alamtegevused	Olukord ettevõttes
SP 1.1	Hinda objektiivselt töö protsessi, et see vastaks	On ära määratud, mida ja millal hinnatakse, kes on	Toote omanik kontrollib, et jälgitaks Scrumi ja Kanbani

	Tegevus	Alamtegevused	Olukord ettevõttes
	protsessikirjeldusele, standarditele ja protseduuridele.	hindamisega seotud ja kuidas see läbi viiakse.	metoodikaid vastavalt sellele, kuidas on varasemalt kokku lepitud.
		Määratleta kõik nõuetele mittevastavused	Kui esineb mitte-vastavusi, siis need arutletakse läbi ja parandatakse, aga ei dokumenteerita
		Määratle, mida õpiti	Kirja õpitut ei panda.
SP 1.2	Hinda objektiivselt töö tulemit	On ära määratud, mida ja millal hinnatakse, kes on hindamisega seotud ja kuidas see läbi viiakse.	Tulemust kontrollitakse uue funktsionaalsuse loomisel, enne kliendile üle andmist või enne väljalaset.
		Määratleta kõik nõuetele mittevastavused	Kõik nõuetele mittevastavused pannakse kirja vearaportite kujul. Iga funktsionaalsus kontrollitakse käsitsi läbi, selle eest vastutavad testijad. Vigade parandamise eest vastutavad arendajad.
		Määratle, mida õpiti	Ei dokumenteerita, peetakse lihtsalt meeles

SG 2 – Mittevastavused on objektiivselt hinnatud ja kommunikeeritud, lahendused on leitud

Tabel 7 Protsessi ja toote kvaliteedi tagamise spetsiifilise eesmärgi nr 2 tegevused

	Tegevus	Alamtegevused	Olukord ettevõttes
SP 2.1	Nõuetele mittevastavused on kirja pandud ja meeskonnaga läbi arutatud	Lahendada mittevastavused	Kõik mittevastavused, mis avastatakse, ka lahendatakse
		Dokumenteerida mittevastavused (panna kirja, mis ei ole vastav ja kuidas seda parandada, vajadusel muuta protsessi)	Kui mitte vastavused on konkreetselt toote funktsionaalsetes nõuetes, nii et süsteem ei tee seda, mida nõutakse, siis see dokumenteeritakse vearaportitena. Kui mitte: vastavused on programmide kasutamises, iganenud tehnoloogias ja muudes kohtades, siis neid ei dokumenteerita hetkel.
		Kui mittevastavus on seotud mõne muu osakonnaga, siis delegeerida see neile	Kui mittevastavus on seotud teise osakonnaga, siis see delegeeritakse.
		Teata huvigruppidele	Olenevalt huvigrupist ja mittevastavusest teavitatakse ka huvigruppe.
SP 2.2	Tehakse märkmeid	Kirjutada ja üle vaadata kvaliteedi tagamise tegevused	Hetkel ei tehta

	Tegevus	Alamtegevused	Olukord ettevõttes
		Koostatakse logid ja kvaliteedi raport	Logisid koostatakse, kvaliteediraporteid mitte

4.3 Analüüs

Ülaltoodud tabelitest tuleb välja, et kõige suuremad puudujäägid on dokumenteerimises ja kontrollimistes. Kvaliteedi tagamiseks on vaja, et olemasolevaid protsesse vastavalt vajadusele parandataks. CMMI järgi saab toodet paremaks teha ainult juhul, kui on teada, mis on valesti ja miks on valesti. Hetkel on probleem selles, et ei teata, mis on projektis valesti ja seda kahel põhjusel: testitakse juhuslikult mitte süstemaatiliselt, mistõttu puudub järjepidev kontroll. Nõuete dokumenteerimise puuduse tõttu ei saada teada, mis on valesti ja nõuete ebatäpsustest jäävad testimisel sisse teadmatusest tulenevad vead.

Lisaks ülaltoodule küsiti ka ByteLife'i töötajatelt endilt, mis on nende jaoks kõige olulisem asi, mis aitaks tõsta toodete kvaliteeti. Seda tehti, et arvestada muudatuste tegemisel kõigi osapoolte, ka kaastöötajate arvamusi. Töötajad pakkusid välja järgnevaid lahendusi:

- Testimist tuleks muuta järjekindlamaks ja mitmekesisemaks: hetkel on testimine liiga kaootiline ja suvaline. Tuleks teha rohkem automaatseid teste ja erinevamaid teste, et leida nõuetele mittevastavusi.
- Tuleks kirja panna, mida testiti ja milline oli tulemus, et saaks analüüsida
- Inimesi tuleks koolitada, et tase oleks ühtlasem
- Tuleks õppida raamistikke kasutama ja neid ka järgida
- Rohkem väliseid arvamusi
- Pidada protsessist kinni

Kuna kõiki CMMI küpsuse ja võimekuse teise tasandi saavutamiseks vajalikke kvaliteedi tõstmisega seotud tegevusi ja asjaolusid selles töös uurida ei jõuta, siis esmajärjekorras keskendutakse sellele, mida tõid välja töötajad ja jälgitakse, et see oleks vastavuses CMMI eesmärkidega. Olemasolevat protsessi täiendatakse testidega, dokumenteerimisega ja analüüsiga.

5. Testimine

Selleks, et toimuks mitmekülgsem ja suuremamahulisem testimine, tuuakse järgnevas peatükis välja mõned enamkasutatavad testimise liigid ja tüübid.

5.1 Funktsionaalne testimine

Funktsionaalse testimise korral vaadatakse programmi kui musta kasti. Oluline ei ole, missugune on programmi kood, vaid tema sisend-väljund käitumine. Konkreetsete sisendite puhul eeldatakse konkreetseid väljundeid. Need väljundid tulenevad spetsifikatsioonist. Kuna võimalikke sisendeid on tavaliselt väga palju, siis oleks hea need kuidagi süsteemselt ära jaotada. Üks võimalus on kasutada ekvivalentsiklasse. [6]

5.1.1 Ekvivalentsklassid

Ekvivalentsklasside moodustamise idee on selles, et sisendandmed jaotuvad töötluse suhtes enamasti rühmadesse nii, et need andmed, mis on ühes rühmas, neid töödeldakse ühtemoodi. Eeldatakse, et kõik andmed, mis on ühes klassis, käituvad ühtemoodi. Kui esimesest klassist leitakse viga, siis leitakse see viga kõikide andmetega samast klassist ja kui mingi konkreetse andmega ei leita viga, siis ei leita seda ka ükskõik, millise teise andmega samast klassist. Kui klassid on moodustatud, siis luuakse nende põhjal testolukorrad. Selleks võib kasutada järgmiseid põhimõtteid. Kui ekvivalentsklass on

- järjestatud tõkestatud vahemik, siis võtta testid vahemiku seest, enne vahemikku ja pärast vahemikku
- järjestatud tõkestamata vahemik, siis võtta testid vahemiku seest ja väljast
- hulk, siis võtta testid hulga seest ja väljast
- tingimus, siis proovida mõlemaid variante (tingimus täidetud/täitmata)

Iga testolukorra jaoks koostatakse test. Selleks, et testidelt kokku hoida, siis võib teha nii, et õigeid klasse, mis ei tekita viga, üritatakse testolukorda panna maksimaalselt. Vigu tuleks aga testida ühekaupa.

Samuti tuleks vaadata sisendandmete omavahelisi sõltuvusi, sest on olukordi, kus ainult mingite andmete kooskasutamine tekitab vea. [6]

Näiteks FlowGrabis on palju välju, mille korral saab luua ekvivalentsklasse. Näiteks tiimi loomisel on nime väljal võimalik luua järgnevad ekvivalentsklassid: ainult ladina tähed, ainult erisümbolid, ainult numbrid. Samuti on võimalik luua erinevaid kombinatsioone, näiteks ladina tähed ja numbrid, ladina tähed ja erisümbolid või erisümbolid ja numbrid. Kuna on teada, et mõned erisümbolid on lubatud (näiteks punktid ja kriipsud) ja ülejäänud ei ole lubatud, siis saab erisümbolitest luua kaks või rohkem klassi. Ühes klassis on sümbolid, mida kasutades antakse veateade. Lubatud sümbolid saab teha kõik eraldi klassideks, kuna on võimalik, et programm käitub punkti puhul teistmoodi kui sidekriipsu puhul.

5.1.2 Piirolukorrad

Kuna on leitud, et paljud vead esinevad ekvivalentsklasside piiridel, siis tuleks teha ka piirolukordade teste. Kui piiriks on näiteks reaalarv, siis testitakse väärtusi R, R+e ja R-e, kus e on vähim väärtus, mis on rakenduse või arvuti seisukohast eristatav. Kui piiriks on täisarv N, siis testitakse väärtustega N, N-1 ja N+1. Kui ülemist või alumist piiri pole, siis testitakse suvalise väga suure positiivse või negatiivse arvuga. Piirjuhte ei teki kui sisendandmeteks on järjestamata hulgad. [6]

Eelmises peatükis 5.1.1 toodud näite põhjal ei saa koostada piirjuhtusid, kuna tegemist on järjestamata hulkadega. Küll aga saab piirjuhte sisse tuua juhul, kui on teada ka konkreetne välja pikkus. Kui tiimi nimi peab olema vähemalt kolm tähemärki pikk ja maksimaalselt 32 tähemärki pikk, siis saab luua kõigepealt ekvivalentsklassid: vähem kui 3 tähemärki, 3 kuni 32 tähemärki ja pikem kui 32 tähemärki. Sel juhul on piirolukorraks 3 ja 32 ja testida tuleks väärtustega 2,3,4,31,32 ja 33.

5.1.3 Veaotsing

Kogenenum arendaja ja testija oskavad vea-kohti ette aimata. Osad vead sõltuvad konkreetsest valdkonnast või rakendusest ja neid on tavaliselt raske formaliseerida. Veaotsingut võib teha süsteemselt või intuitiivselt. Esimesel juhul kasutatakse mingeid küsimustike konkreetse valdkonna kohta, mida süsteemselt läbi vaadatakse. See on abiks testijatele, kellel on vähem teadmisi ja kogemusi. Kogenud testijad võivad pigem testida juhuslikult, sest nad oskavad veakohti paremini ette näha ja hoiavad sealjuures aega kokku. Tõenäoliste vigade leidmine sõltub mitut sorti eelteadmistest, milleks on üldised teadmised, teadmised konkreetse rakendusvaldkonna kohta, teadmised riist- ja tarkvarakeskkonna kohta, teadmised arendusmetoodika ja konkreetse arendaja või tellija kohta. [6]

5.1.4 Automaatsed kasutajaliidese testid

Kasutajaliidese teste luuakse selleks, et kontrollida, kas kasutajaliides vastab spetsifikatsioonile ja nõuetele. Teste on võimalik kirjutada funktsionaalsuse kohta ja lehe enda kohta. Lehetestid testivad lehe terviklikust ja õiget viitamist. Kui lehel on mõni nupp, siis sellele vajutades peab avanema mingi kindel leht. See on lehetestide ülesanne kontrollida, et see nii oleks. Samuti tuleb kontrollida, kas väljad on vajadusel õigesti eeltäidetud või tühjad. [7]

Teste, mis kontrollivad funktsionaalsust, on võimalik luua sama testikirjelduse järgi nagu käsitsi funktsionaalsust testides. Tarkvara uuendatakse koguaeg ja funktsionaalsuse kasvades peaks tegelikult testima uuesti üle ka kogu varem loodud funktsionaalsuse. Seetõttu on hea luua kohe automaatne kasutajaliidese test kui uus funktsionaalsus on loodud. Sellisel juhul ei pea seda enam edaspidi käsitsi kontrollima, piisab vaid loodud testide käivitamisest.

Näiteks FlowGrabis on olemas leht nimega „indeks“ ja sellel lehel on e-posti sisestamise väli ja nupp nimega „registreeri“. Kui e-posti lahter ära täita ja nuppu vajutada, siis kuvatakse registreerimisleht, millel on eeltäidetud meili väli. Ülejäänud lahtrid peavad olema tühjad. Lehetest peab seega kontrollima kõiki lahtreid.

5.1.5 Funktsionaalse testimise kokkuvõtteks

Kõige efektiivsem (maksumus on väiksem) on tavaliselt vea-otsing, siis piirjuhud ja ekvivalentsiklassid. Seetõttu tasuks alustada esimesest. Funktsionaalse testimise võib jagada järgevalt etappideks:

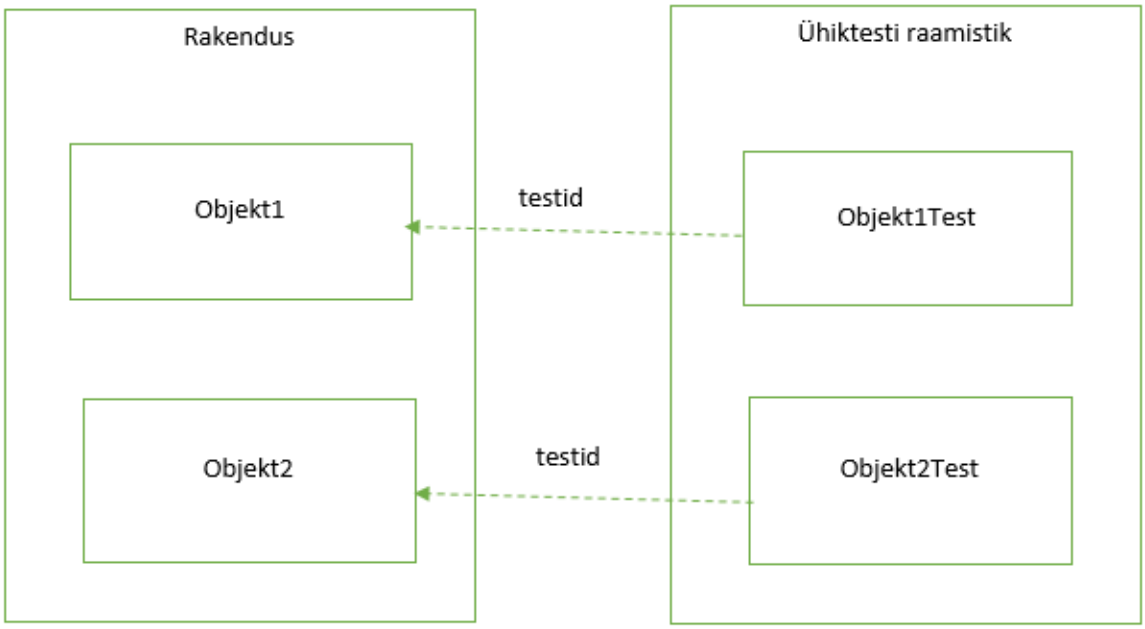
1. eristada sisend- ja väljundandmete ekvivalentsiklassid, sealhulgas erinevate sisendite/väljundite kombinatsioonid
2. leida igas klassis, nende piiridel ja vajadusel kombinatsioonidel esinevad testolukorrad
3. leida igale testolukorrale vastavad sisend-väljundandmed
4. identifitseerida testid, koostada testimise plaan
5. testida, võrrelda tulemusi, hinnata

Funktsionaalne testimine eeldab põhjalikku spetsifikatsiooni. Kõige kasulikum on testid koostadagi samal ajal kui spetsifikatsioon. Sel juhul saab ühtlasi ka spetsifikatsiooni täiendada. Samas tuleb arvestada, et selline testimine võib olla väga mahukas, eriti kui ekvivalentsklassi

on palju ja nad on omavahel sõltuvuses. Kui funktsionaalseid teste korralikult teha, on tulemuseks testikomplekt, mis katab kogu funktsionaalsuse. [6]

5.2 Ühiktestid

Ühikteste kirjutatakse programmikoodi kohta. Iga programmiühiku nt meetodi kohta tuleks teha testid. Testid peavad kontrollima konkreetse ühiku käitumist, vajalik on teha üks test õige käitumise kohta ja üks või mitu testi vale käitumise kohta. Testide arv sõltub meetodist. Järgnev joonis näitab koodi ja ühiktestide suhet.



Joonis 10 Rakenduse koodi ja ühiktestide suhe [8]

Oluline on, et ühikteste saaks käivitada eraldi ülejäänud programmikoodist, nii et testimine toimub isoleeritult. Ühiktestid peavad olema iseseisvad ja nad ei tohi sõltuda teistest komponentidest. Hästi kirjutatud ühiktestid on lühikesed ja konkreetsed, testides ainult konkreetset ühikut. Ühikteste on võimalik luua enne ja pärast programmikoodi kirjutamist.

Ühiktestid koosnevad kolmest osast: esimene on muutujate initsialiseerimine, teine on testitava meetodi käivitamine ja kolmas on kontroll. Selline ülesehitus teeb testid kergesti loetavaks ja on kõigile arusaadav. [8]

5.3 Testimine programmi teksti põhjal

Tekstipõhisel testimisel võetakse ette programmikood ja püütakse selle põhjal teste luua. Kui teha ainult funktsionaalset testimist, siis võivad osad asjad testimata jääda nagu näiteks harud, mida tavalisel täitmisel ei läbita. Tekstipõhisel testimisel võib lähtuda juhtimisest või andmetest. Juhtimisest lähtuv tekstipõhine testimine püüab süstemaatiliselt läbida programmi osasid nt harusid või lauseid. Andmete põhisel testimisel kasutatakse ekvivalentsklasse, piirjuhte ja veaotsingut nagu ka funktsionaalne testimise korral. Sisendandmed valitakse sel juhul aga programmi tekstis antud andmestruktuuride alusel. Erinevus funktsionaalse ja andmepõhisel testimisel on selles, et piirolukorrad võivad tekkida programmis antud kitsendustest nagu näiteks massiivide pikkus, reaalarvu võimalik väärtus vms. Programmiteksti põhine testimine eeldab, et on olemas programmitekst. [6]

5.4 Staatiline testimine

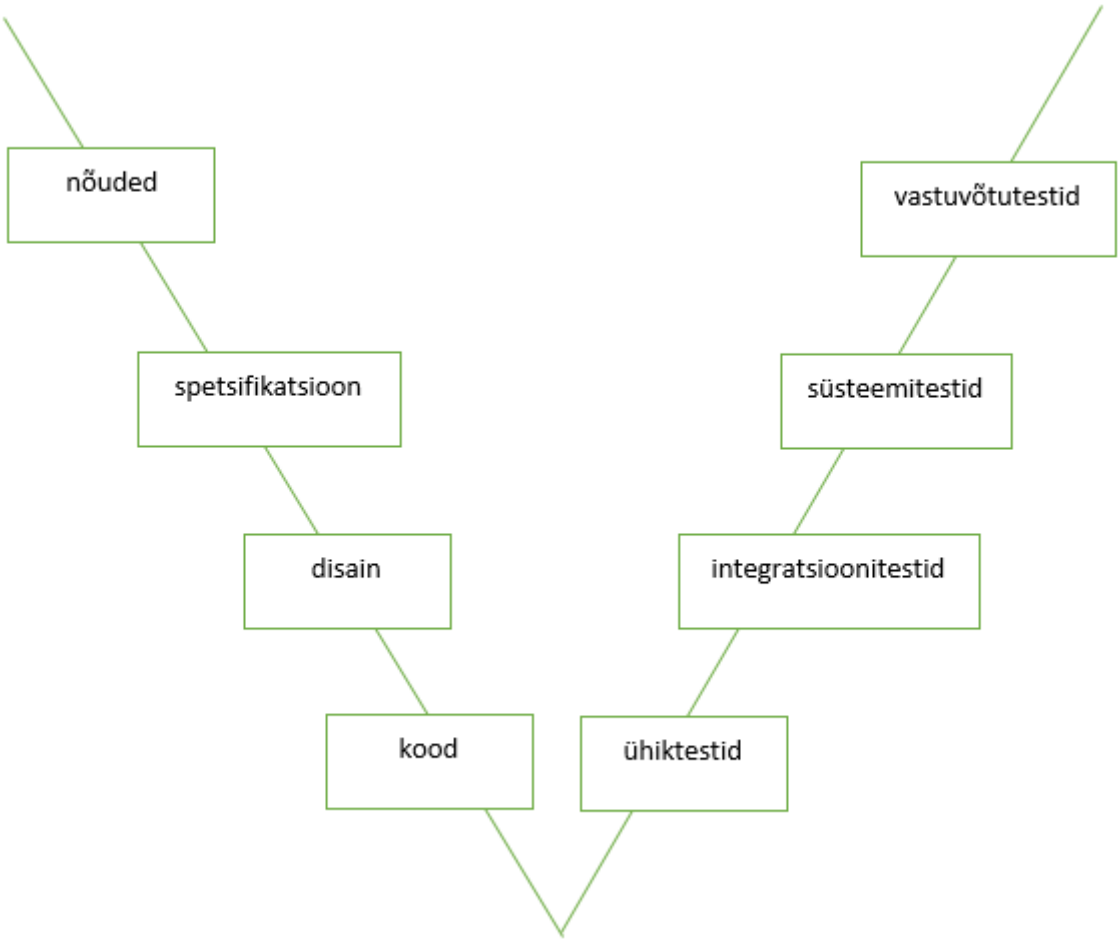
Staatiliste meetodite puhul ei käivitata tarkvara nagu dünaamiliste meetodite puhul. Staatiliste meetodite puhul vaadeldakse tarkvara ja tehakse sellele analüüs. Üks viis seda teha on luua küsimustikud ja hiljem programmi luues neile vastata. [9]

5.4.1 Küsimustikud

Küsimustikud aitavad testijatel laiemalt mõelda ja kasutada ekspertteadmisi. Küsimustikud võivad olla valdkonna, programmeerimiskeele või programmi kohta ja on seetõttu väga erineva mahuga. Küsimustikud sisaldavad momente, millele muidu ei osata tähelepanu pöörata. Küsimustiku näide on lisas 1.

5.5 Elutsükli V-mudel

V-mudel toimib põhimõttel, et iga arendusetapi jaoks luuakse oma testid. Kuna etapid on erinevad, võivad ka testid ja mudelid olla erinevad. Alloleval joonisel on näidatud tüüpiline v-mudel.



Joonis 11 v-mudel [10]

Kui kirjutatakse koodi, siis tuleks kirjutada koodile vastavaid ühikteste, seda kirjeldati ülal. Disainietapis tuleks luua integratsioonitestid, spetsifikatsiooni kohta süsteemitestid ja nõuetele vastavad vastuvõtutestid.

Integratsioonitestid testivad seda, kuidas üksikud moodulid koos töötavad. Kui ollakse veendunud, et ühiktestid üksinda töötavad, luuaks integratsioonitestid. Ülaltoodud kasutajaliidese testid aitavad samuti testida seda, kas kõik komponendid koos töötavad.

Süsteemitestid testivad süsteemi kui tervikut. Testimise käigus testitakse ära kõik funktsionaalsed ja mitte-funktsionaalsed nõuded. Nende hulgas kuuluvad süsteemitestide alla ka koormustestid, taastamis-, taluvus- ja turbetestid. Süsteemitest tehakse siis kui toode on valmis. [11]

Vastuvõtutestid kontrollivad, kas programm teeb seda, mida talt nõutakse. Neid teste teeb tavaliselt ka kasutaja ise. Testid tehakse tavaliselt enne toote kliendile üleandmist. [6] [10]

5.6 Testide sobivus ettevõttele

Eelpool mainitud testidest võetakse antud töös kindlasti kasutusele ühiktestid. See on vajalik selleks, et arendajad ise hakkaksid looma kvaliteetsemat koodi ja et juba nende kirjutatud kood oleks üle testitud. Teiseks võetakse kasutusele funktsionaalne testimine, leitakse ekvivalentsklassid, piirjuhud, mille põhjal luuakse testikirjeldused. Samuti võetakse kasutusele veaotsing ja kõik testid, mille kohta on võimalik, tehakse automaatsed kasutajaliidese testid.

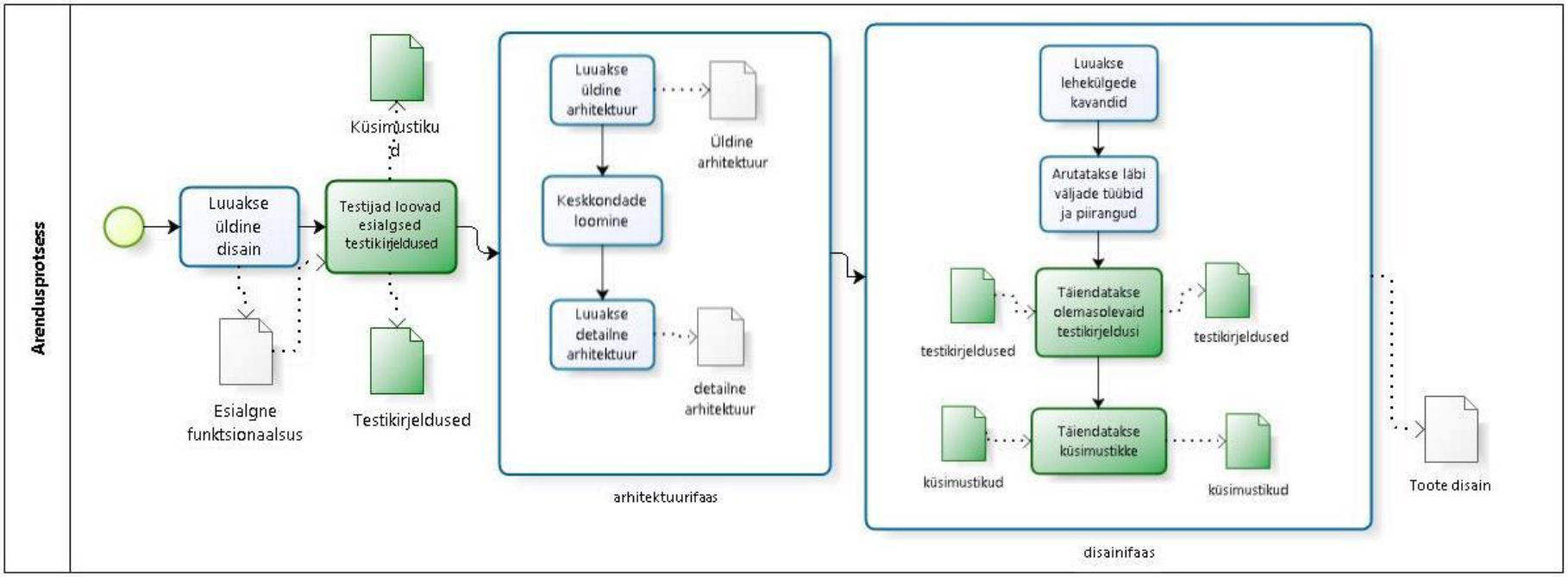
Kuna ByteLife'i testijad on alles kogenematud, siis on hea kasutusele võtta ka küsimustikud, mis aitaks luua erinevaid testikirjeldusi ja leida vigu kohtades, mille peale ise poleks tulnud. Natuke hiljem on plaanis kasutusele võtta ka programmiteksti põhine testimine, et testimine oleks veelgi mitmekülgsem. V-mudelit täies mahus järgima ei hakata, aga see aitab paika panna, millal mis testimist teha.

6. Protsessi täiendamine

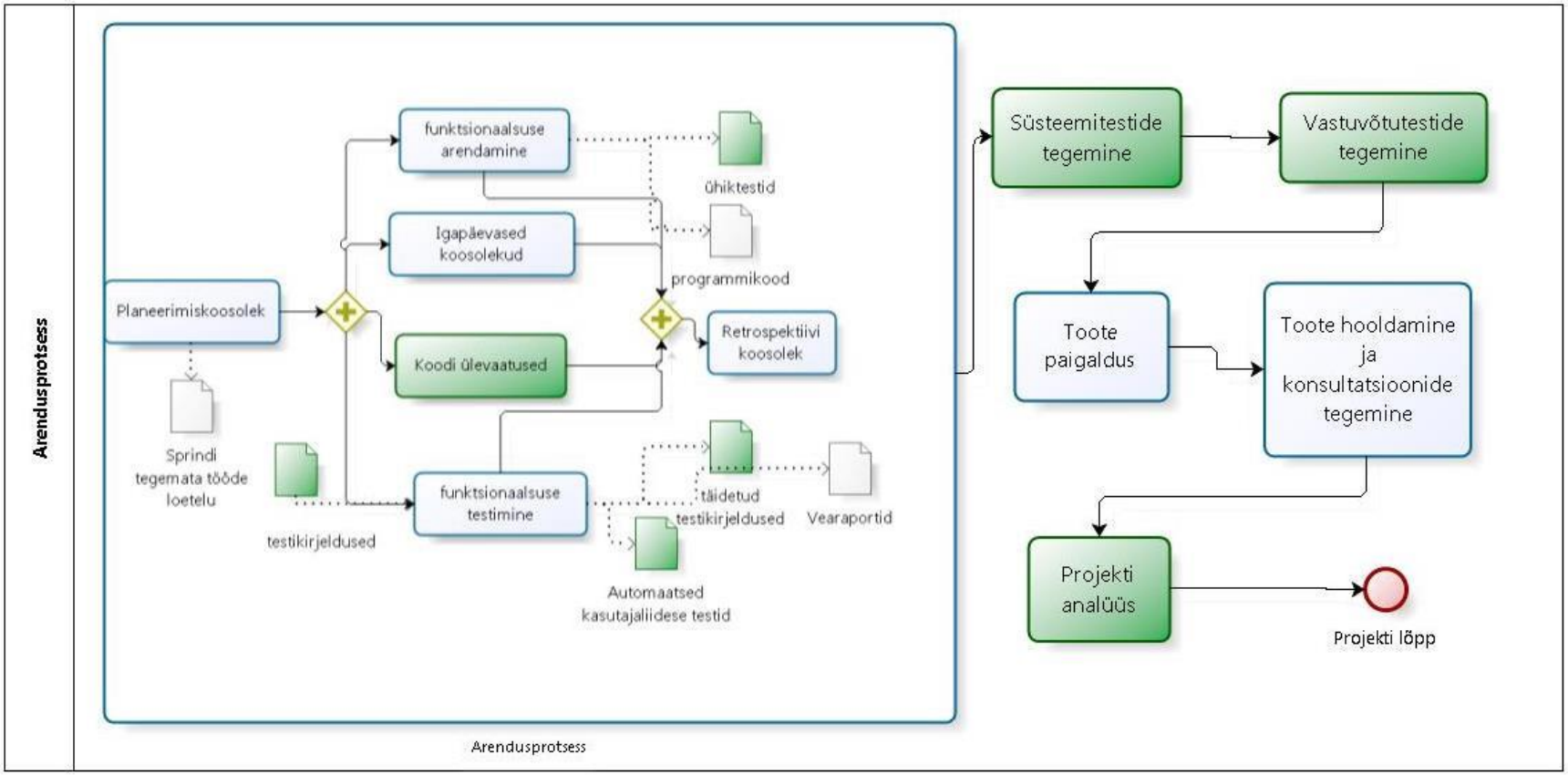
Kõige alustuseks tuleks arutada kõikide arendustiimi liikmetega, mis on kvaliteetne toode ja mida teha, et selleni jõuda. Kirja tuleks panna konkreetsed tegevused, mis tagavad kvaliteedi. Selle töö raames arutleti juba selle üle, mida tuleb teha, et toode oleks kvaliteetne. Tuleb luua protsess, mis vastaks CMMI-le ja seda järgida. Kui protsessi ei järgita, tuleb aru saada, miks seda ei tehta ja kui selgub, et mõni tegevus protsessis ei ole efektiivne, tuleb seda muuta. Järgnevas peatükis pannaksegi paika arendusprotsessi muudatused. Edaspidi tuleks seda protsessi järgida, seejärel uuesti analüüsida ja teha vajalikke muudatusi.

6.1 Arendusprotsess

Järgnevalt on ära toodud tehtud muudatuste mudel. Loetavuse tõttu on protsess jagatud kaheks, esimeses osas on ära toodud muudatused kuni disainifaasi lõpuni (Joonis 12). Teises osas on muudatused arendusfaasist projekti lõpuni (Joonis 13). Olgu ära mainitud, et detailsemad muudatused arendusfaasi kohta leiab punktist 6.1.3. Rohelisega on märgitud tegevused ja dokumendid, mis tulid juurde protsessi täiendamise käigus.



Joonis 12 Muudetud arendusprotsessi mudel osa 1



Joonis 13 Muudetud arendusprotsessi mudel osa 2

6.1.1 Projekti alustamine ja arhitektuurifaas

Tarkvara arenduse esimene osa ehk esmane kliendiga kohtumine ja esialgne kõrgtaseme disaini loomine jäävad samasuguseks. Etteruttavalt võib ka öelda, et muudatusi ei tule paigalduse ja hoolduse etappides. Kui kliendiga on kokku lepitud, milline on sobiv funktsionaalsus ja nõuded, siis arhitektid hakkavad looma arenduskeskkondi ja arhitektuuri nagu varem. Samal ajal tuleks aga kirja hakata panema testistsenaariumeid. Esialgu on võimalik võtta funktsionaalsus ja panna kirja kõik korrektsed käitumisviisid ning kui on juba teada, milline on vale käitumine, siis panna kirja ka need testid. Kui funktsionaalsus on üldiselt sõnastatud, siis võivad olla ka testi kirjeldused üldised. Testandmete valikul tuleks kasutada ülalmainitud ekvivalentsklasse ja piirjuhte. Kui on võimalik, siis võiks ka luua küsimustikke, mis aitaksid hiljem vähemkogenud testijatel veaotsingut teha.

Näide (näited on võetud FlowGrabi projektist)

Funktsionaalsus: kasutaja peab saama luua projekti oma tiimi

Nende teadmiste põhjal saab luua üldise testi. Teststsenaariumi täidetud vorm on olemas lisas 2. Testikirjelduste loomisel lähtuti materjalist [12].

6.1.2 Arenduskeskkondade loomine ja disainifaas

Kui valmivad tarkvara arhitektuur ja disain, siis tuleks juba olemasolevaid testikirjeldusi täiendada ja võimalusel detailsemaks muuta. Loodavates testides saab juba arvestada arhitektuurist tulenevate kitsendustega ja samuti väljade tüüpide ja pikkustega. Esialgu saab teha eelmises faasis mainitud funktsionaalsusele detailsemad testid, kuna on selgunud, milliseid välju ja mis piirangutega konkreetse projekti loomiseks vaja on. Nagu ka testikirjelduste puhul, nii ka küsimustike puhul saab siin faasis teha täiendusi.

Näide

Projekti loomisel tuleb kohustuslikus korras valida projektile nimi, tiim, kategooria ja kaasa panna pakett. Valikuliselt võib lisada kirjelduse, veebilehe, litsentsi ja märked otsingu jaoks (tag).

Järgenvalt vaatame nime välja. Disainifaasis tuli välja, et nimi peab olema vähemalt kolm tähemärki pikk ja maksimaalselt 32 tähemärki pikk. Nimi võib sisaldada ainult ladina tähti,

punkte, sidekriipse, alakriipse ja numbreid. Pole vahet, kas kasutaja sisestab tähed suurena või väiksena, need tehakse kõik väikesteks tähtedeks.

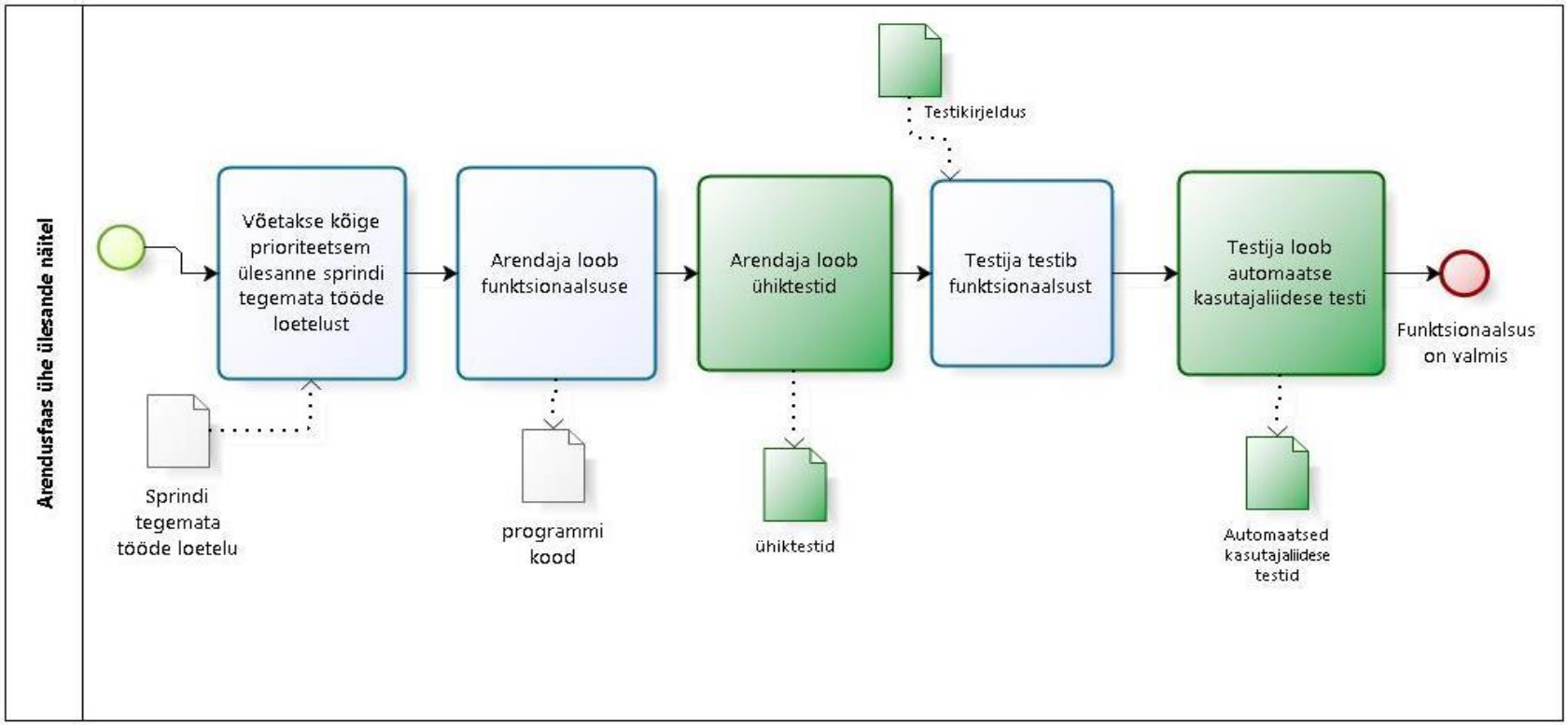
Eeltoodud info põhjal saab luua ekvivalentsklassid välja pikkusele: väiksem kui 3 tähemärkki on esimene klass, teine klass on 3 kuni 32 tähemärki ja kolmas klass on pikem kui 32 tähemärkki. Piirjuhtudeks on 3 ja 32. Nende põhjal saab luua 5 testi.

Teise grupi ekvivalentsklasse saab luua sümbolite kasutamise kohta. Esimene klass on ainult ladina tähed, teine klass on numbrid, kolmas klass on erimärgid välja arvatud punkt, sidekriips ja alakriips. Viimase kolme kohta võiks igaühe kohta luua testi. Seega siit saab teha 6 testi.

Niimoodi tuleb läbi mõelda kõikide väljade testid. Analoogselt tuleb läbi mõelda kõik funktsionaalsusega seotud testid ja need kirja panna. Juhul kui selgub, et midagi on puudu piirangutest või funktsionaalsuses, siis saab kohe teha parandusi, enne kui arendamine üldse alanud on. Disainifaasi lõpuks peaks valmima kogu funktsionaalsust hõlmav testide kogu. Siiski võivad testid olla kohati üldised, kuna mõned funktsionaalsused tehakse detailseks alles planeerimiskoosolekutel. Koos sellega tuleb teha detailsemaks ka testide kirjeldused.

6.1.3 Arendusfaas

Järgnevalt on ära toodud muudetud arendusprotsess (Joonis 14).



Joonis 14 Muudetud aredusfaas: ühe ülesande protsess

Kui lõpuks on selge, mida tuleb teha, siis hakatakse looma funktsionaalsusi prioriteetsuse järgi nagu ka varasemalt. Kasutatakse endiselt KanbanFlow'd ja protsess ise näeb välja samamoodi: toimuvad planeerimiskoosolekud, igapäevased 15-minutilised koosolekud ja retrospektiivid. Erinevus on selles, et kui arendaja võtab KanbanFlow's omale ülesande ja asub seda programmeerima, hakkab arendaja nüüdsest kirjutama ka ühikteste. Neid kirjutatakse iga meetodi kohta välja arvatud *getter'id* ja *setter'id*. Nagu ka ülalpool mainitud, tuleb jälgida, et testid oleksid lihtsad ja kontrolliksid ainult konkreetset meetodit. Kui meetod kasutab andmebaasi või mõnda muud komponenti, siis tuleb neid *mockida,* et tagada testide iseseisvus ja isoleeritus.

Kui arendaja on ülesande lõpetanud, paneb ta selle KanbanFlow's järgmisesse veergu „testimiseks valmis". Üldjuhul peaks testija võtma endale ülesande ja seda testikirjelduse järgi testima. Kuna aga varasemates faasides olevad kirjeldused on üldisemad kui ülesanded, mis on kirjas KanbanFlows, siis tuleks testikirjeldused teha kas detailsemaks või siis alustuseks testida kirjelduse ühte osa. See, kumb lahendus on sobivam, oleneb konkreetsest ülesandest.

Kui konkreetne ülesanne on testikirjelduse järgi testitud, siis kirjutab testija selle kohta ka automaatse kasutajaliidese testi. Seda juhul, kui see on võimalik. Pärast automaatse testi kirjutamist tõstab testija ülesande „lõpetatud" veergu. Oluline on, et siiski saaks testitud ka kogu funktsionaalsus, mitte ainult konkreetne ülesanne. Seetõttu on mõistlik lisada toote tegemata tööde loetellu ka sellised ülesanded, mis on mõeldud konkreetselt mingi funktsionaalsuse testimiseks.

Näide

Ülaltoodud näite põhjal saab öelda, et funktsionaalsus on projekti loomine. Selle jaoks luuakse väiksemad ülesanded, näiteks, „„loo projekt" nupu loomine", „nupu vajutamisel avaneb projekti loomise leht", „nime välja kontrollide loomine", „andmete salvestamine andmebaasi" ja nii edasi, väiksemaid ülesandeid tuleb luua väga palju. Igat väikest osakest tuleb testida kui see jõuab „testimise" lahtrisse. Kuid selleks, et läbi proovida terve funktsionaalsus, võiks tegemata tööde listis olla ka eraldi ülesanne märkega test, mis nõuaks kogu funktsionaalsuse testimist. See on justkui integratsioonitest. Näiteks võiks ülesanne kirja pandud olla nii: „test: projekti loomine".

Lisaks funktsionaalsuse testimisele testikirjelduste põhjal, tuleks pärast koodi kirjutamist luua ka testid koodiprogrammi põhjal. Üle tuleks vaadata, et kõik harud ja laused oleks testitud. Seda testimist võiks teha ainult juhul kui on piisavalt aega.

Kuna varasemalt sai mainitud, et arendajate ja testijate tase on ebaühtlane, siis tuleks kasutusele võtta koodi ülevaatus. Seda tuleks teha kogu arendusprotsessi vältel. Teiste koodi üle vaatamine aitab paremini mõista, mida teised teevad ja niimoodi on võimalik oma teadmisi täiendada.

6.1.4 Süsteemitestid ja vastuvõtutestid

Kui kogu funktsionaalsus on valmis ja seda soovitakse kliendile üle anda, siis tuleks veelkord kogu toode läbi testida. Ühest küljest aitavad seda teha automaatselt loodud testid. Lisaks nende käivitamisele tuleks teha juhuslikku testimist ja testimist küsimustike põhjal. Küsimustele vastamine aitab aru saada, kas kõik on testitud ja läbi mõeldud. Kui kõigile küsimustele on vastatud ja selgub, et kõik töötab nii nagu peaks, siis võib toote üle anda. Hiljem avastatud vead parandatakse nii nagu varasemalt.

6.1.5 Projekti analüüs

CMMI järgi on oluline tehtud vigadest õppida ja neid parandada. Seetõttu võiks pärast igat projekti teha analüüsi, mis läks hästi ja mis halvasti. Samuti võiks kirja panna, mida õpiti ja milliseid tehnoloogiaid/raamistikke/meetodeid tasub ja milliseid ei tasu kasutada. Sellist analüüsi on lihtsam teha kui retrospektiive dokumenteerida. Hetkel kirjutatakse juba ülesse halvasti olevate asjade lahendused, ent juurde võiks ka märkida, kuidas lahenduse kasutusele võtmine olukorda muutis. Sellised dokumendid aitaksid hiljem kogu projekti analüüsida.

6.1.6 Rollid ja faasid

Kuna testikirjelduste loomisega alustatakse kohe pärast funktsionaalsete nõuete kokkuleppimisel kliendiga, siis on uue protsess puhul vajalik, et nad liituksid projektiga juba varasemas faasis ja oleksid kaasatud vähemalt arhitektuurifaasist paigalduse lõpuni. Paigalduse puhul on vajalik teha vastuvõtuteste. Muudatused on märgitud paksema kirjaga.

Tabel 8 Faasid ja rollid muudetud protsessis

Rollid/faasid	Projekti algus	Arhitektuurifaas	Disainifaas	Arendusfaas	Süsteemitestide tegemine	Paigaldus	Hooldus
Müügitiim	X						
Projektijuht	X	X	X	X		X	X
Tiimijuht	X	X	X	X		X	X
Keskkondade seadistaja		X					
Juhtiv arendaja		X	X	X			
Arendaja			X	X			
Testija		**X**	X	X	**X**	**X**	
Paigalduse ja hooldusega tegelev isik						X	X

6.2 Uue protsessi vastavus arendusmetoodikale

CMMI protsessi ja toote kvaliteedi kindlustamise juures on oluline see, et muudatuste tegemisel jälgitaks kirja pandud protsesse, standardeid ja meetodeid. Seetõttu on oluline kontrollida, et töös pakutud muudatused vastaksid Kanbani ja Scrumi põhimõtetele.

Scrumi üldised põhimõtted jäävad kehtima ka protsessi muudatuste ajal. Rollideks on endiselt toote omanik, tiim ja scrumi meister. Samuti tehakse planeerimiskoosolekuid, retrospektiive ja päevaseid püstijala koosolekuid. Olemas on tegemata tööde loetelu, tehtud tööde loetelu ja

sprindi tegemata tööde loetelu. Alljärgnevalt on ära toodud kaks Scrumi põhimõtet, mis on seotud muudatuste tegemisega.

Scrumi üks põhimõte on, et sprindi jooksul ülesandeid ei lisata ega muudeta, see aga eeldab, et sprindi planeerimise koosolekul on funktsionaalsus läbi mõeldud. Ülaltoodud täiendused, et funktsionaalsuse detailsemaks muutmise ajal tuleks luua ka detailne teststsenaarium, aitab seda saavutada, kuna funktsionaalsus on rohkem läbi mõeldud.

Teine oluline Scrumi põhimõte on see, et sprindi alguses saavad kõik tiimiliikmed aru, mida tuleb teha. Testide kirjutamine funktsionaalsusega samal ajal aitab samuti seda saavutada, sest ka testijad peavad kohe kaasa mõtlema, mida funktsionaalsus teeb ja kuidas seda testida. [1]

Kanbani puhul ei ole väga palju reegleid, mida tuleb teha ja mida ei tohi teha, seetõttu ei lähe ülalmainitud muudatused sellega ka vastuollu. Kanbani peamised põhimõtted on, et ülesanded peavad olema tehtud võimalikult väikesteks tükkideks ja et enne ei alustata uue ülesandega kui vana on valmis. Hetkel juba kasutatakse samu põhimõtteid ja neid kasutatakse ka uue protsessi puhul. [13]

6.3 Muudatuste vastavus CMMI-le

Järgenvalt on tehtud muudatused kujutatud tabelina. Esimeses reas on CMMI taseme tegevused lühendina. Esimene tabel on Üldistes eesmärkides tehtud muudatuste kohta.

Tabel 9 CMMI üldistes eesmärkides tehtud muudatused

Tegevus	Enne	Muudatus
GP 2.1	Organisatsioonil on poliitika protsesside planeerimiseks.	Muudatusi ei tehtud
GP 2.2	Mingisugune plaan protsessi täitmiseks luuakse. Pannakse paika ressursside jaotumine ja vastutusalad. Protsess vastab nendele ettevõtte poliitikatele, mis on olemas, samuti Eesti Vabariigi seadustele. Kokku on lepitud, et kasutatakse Scrumi ja Kanbani põhimõtteid.	Muudatusi ei tehtud

Tegevus	Enne	Muudatus
GP 2.3	Projekti planeerimise käigus tehakse vajalik analüüs, kas on piisavalt ressurssi projektiga tegelemiseks. Selle käigus määratakse ka ära, kui palju aega, raha ja tööjõudu projekti peale kulub.	Muudatusi ei tehtud
GP 2.4	Mõned vastutusalad on määratud töölepinguga ja teised vastutusalad pannakse paika vastavalt projektile. Vastavalt Scrumile on projekti vastutaja Toote omanik, tiimi eest vastutab Scrumi meister. Kuigi Scrumi järgi vastutab kõige eest ühiselt tiim, on siiski määratud vastutaja testimise, arendamise ja keskkondade seadistamise jaoks.	Muudatusi ei tehtud
GP 2.5	Kuigi valdavalt tegelevad inimesed sellega, milles nad on tugevad, on siiski sama tööga tegelevate inimeste tase hetkel erinev. Ettevõttes on võimalik tellida raamatuid, otsida enda jaoks koolitusi ja teha sertifikaate, aga ühiseid, ettevõtte poolt pakutud koolitusi ei ole.	Hakatakse tegema koodi ülevaatusi, et õppida teistelt arendajatelt ja testijatelt. Kui leitakse mõni koolitus, siis võetakse sellest osa.
GP 2.6	Hetkel on kontroll töö tulemuse üle nõrk, uue funktsionaalsuse puhul testitakse seda manuaalselt, kirja pannakse ainult vead. Enne väljalaskeid kontrollitakse üle kogu funktsionaalsus. Puuduvad automaatsed testid ja testide raportid.	Kogu loodav funktsionaalsus on esialgu dokumenteeritud, selle põhjal koostatakse funktsionaalsete testide kirjeldused. Kõik avastatud vead on kirjas vearaportitena ja testikirjeldustes. Luuakse automaattestid ja ühiktestid.
GP 2.7	Huvigrupid on määratud. Klientidega lepitakse projekti planeerimise ajal kokku, kui tihedalt nad projektiga seotud tahavad olla, kui tihti tahetakse uusi funktsionaalsusi näha jne. Kuna	Muudatusi ei tehtud

Tegevus	Enne	Muudatus
	igal reedel on üldine koosolek, kus osalevad kõik töötajad ja kus viiakse üksteist kurssi nädala jooksul tehtud töödega, siis on kõik kursis ettevõttes toimuvaga. Juhtkond arutab projekte ka kinnisemas seltskonnas.	
GP 2.8 ja GP 2.9	Konkreetse projekti puhul kontrollib toote omanik, et jälgitaks Scrumi ja Kanbani metoodikaid viisil nagu need on protsessis kirjas.	Muudatusi ei tehtud
GP 2.10	Toimuvad koosolekud, kus juhtkond vaatab läbi kõik käimas olevad projektid.	Muudatusi ei tehtud

Järgevalt on ära toodud tabel muudatustega konkreetsete tegevuste kohta.

Tabel 10 CMMI konkreetsetes eesmärkides tehtud muudatused

Tegevus	Enne	Muudatus
SP 1.1	Toote omanik kontrollib, et jälgitaks Scrumi ja Kanbani metoodikaid vastavalt sellele, kuidas on varasemalt kokku lepitud.	Muudatusi ei tehtud
	Kui esineb mitte-vastavusi, siis need arutletakse läbi ja parandatakse, aga ei dokumenteerita	Lisaks parandamisele pannaks kirja kõik mittevastavused testikirjeldustesse. Uue tulemuse saamisel lisatakse testikirjeldusse uus tulemus, nii et ka vana tulemus jääb alles.
	Kirja seda ei panda, mida õpiti, seda inimesed peavad lihtsalt meeles	Retrospektiividel dokumenteeritakse, mis on halvasti ja millised on lahendused, et protsessi paremaks muuta. Nende põhjal

Tegevus	Enne	Muudatus
		koostatakse pärast analüüs ka kogu projekti kohta.
SP 1.2	Tulemust kontrollitakse uue funktsionaalsuse loomisel, enne kliendile üle andmist või enne väljalaset. Kuna ühikteste ja integratsiooniteste ei tehta, siis sel ajal ei kontrollita.	Tulemust kontrollitakse uue funktsionaalsuse loomisel testikirjelduste põhjal käsitsi testimisega. Edaspidi luuakse kasutajaliidese test, mis kontrollib funktsionaalsust igal käivitamisel ehk siis iga kord kui keegi täiendab koodi ja paneb selle arenduskeskkonda. Enne kliendile üleandmist või enne uut väljalaset kontrollitakse tähtsam funktsionaalsus üle testikirjelduste põhjal. Tehakse ka ühikteste, mis käivitatakse samuti igakord kui keegi lisab oma koodi arenduskeskkonda.
	Kõik nõuetele mittevastavused pannakse kirja vearaportite kujul. Iga funktsionaalsus kontrollitakse käsitsi läbi, selle eest vastutavad testijad. Vigade parandamise eest vastutavad arendajad.	Nõuetele mittevastavused on lisaks vearaportitele dokumenteeritud ka testikirjeldustes
	Seda, mida õpiti, ei dokumenteerita, vaid peetakse lihtsalt meeles	Retrospektiividel dokumenteeritakse, mis on halvasti ja millised on lahendused, et protsessi paremaks muuta. Nende põhjal koostatakse pärast analüüs ka kogu projekti kohta.
SP 2.1	Kõik mittevastavused, mis avastatakse, ka lahendatakse	Muudatusi ei tehtud

Tegevus	Enne	Muudatus
	Kui mitte-vastavused on konkreetselt toote funktsionaalsetes nõuetes, nii et süsteem ei tee seda, mida nõutakse, siis see dokumenteeritakse vearaportitena. Kui mitte vastavused on programmide kasutamises, iganenud tehnoloogias ja muudes kohtades, siis neid ei dokumenteerita hetkel.	Mittevastavused tehnoloogiates ja mujal pannakse kirja projektianalüüsi.
	Kui mittevastavus on seotud teise osakonnaga, siis see delegeeritakse.	Muudatusi ei tehtud
	Olenevalt huvigrupist ja mittevastavusest teavitatakse ka huvigruppe.	Muudatusi ei tehtud
SP 2.2	Hetkel ei kirjutata üles, millised on konkreetsed kvaliteedi tagamise tegevused	Kvaliteedi tagamise tegevused mõeldakse läbi ja edaspidi parandatakse neid vastavalt vajadusele
	Logisid koostatakse, kvaliteediraporteid mitte	Edaspidi koostatakse ka kvaliteediraporteid

7. Juurutamine

Muudatused tuleb läbi viia nii, et need ei segaks hetkel kasutusel olevat töökorraldust. Hetkel on arenduses üks projekt, ent varsti on algamas järgmine. Kõigepealt teen ettepanekud juba käimas oleva projekti tarbeks.

7.1 Juurutamine projektis FlowGrab

Kõigepealt tuleb kasutusele võtta ühiktestid. Kuna aga suurem osa koodist on juba olemas, siis alustatakse sellest, et ühiktestid luuakse järgnevalt iga uue funktsionaalsusega või vigade parandamise käigus. Samamoodi toimitakse ka funktsionaalsete testide korral. Kui uus funktsionaalsus lisatakse, kirjutatakse selle kohta testikirjeldused. Seejärel tehakse manuaalsed testid ja kui võimalik, siis ka automaatsed kasutajaliidese testid. Samamoodi toimitakse ka vigade leidmisel. Kui arendaja parandab ära vea, siis kirjutab ta veaga seotud funktsionaalsuse kohta testikirjeldused ja UI-testid.

Esialgu tuleb arvestada sellega, et siiani ei ole arendajad kirjutanud ühikteste ja testijad ei ole kirjutanud automaatseid teste. Esialgu tuleb leida aega, et õppida teste kirjutama ja leida töövahendid ja raamistikud. Kui esialgne otsus on tehtud, siis tuleb jälgida selle kasutamist ja kirja panna, kuidas ja mida kasutati. Tulevikus tuleb teha selle kohta analüüs, kas tehnoloogiad ja raamistikud olid sobivad või tuleks otsida mõni uus raamistik. Esialgu võib juhtuda, et arendajad ja testijad ei ole nii efektiivsed, kuna kogemus ja harjumus uutmoodi tööd teha puudub. Seetõttu tuleks iga muudatuse käiku võtmisel varuda aega ja kannatust.

Samuti tuleb meeskonnal arvestada, et kuna tööd on rohkem: varasemaga võrreldes tuleb rohkem dokumenteerida ja luua teste, siis võtab uue funktsionaalsuse arendamine esialgu kauem aega. Kuid hilisemates faasides peaks olukord muutuma vastupidiseks: kuna vead on varasemates faasides avastatud, siis ei pea arendusfaasi lõpus olulisi ja suuri muudatusi tegema ning toote kvaliteedile vastavus on paremini kontrollitud.

Projekti lõppedes tuleb kohe teha sellele analüüs, kuidas läks ja mida saaks paremini teha. Uued mõtted tuleks kirja panna ja järgmises projektis proovida neid rakendada.

7.2 Juurutamine uue projekti puhul

Kui arendusse tuleb uus projekt, tuleks kasutusele võtta kõik muudatused. Seda ei pea kohe tegema täies mahus. Ka siin tuleb arvestada, et kuna testijatel puudub kogemus testikirjelduste koostamisel, ei pruugi nende tegemine olla kohe efektiivne. Ilmselt selgub esimestes projektides, et osadele funktsionaalsustele pole piisavalt teste, sellisel juhul tuleb lihtsalt testikirjeldusi hiljem juurde kirjutada. Oluline aga on, et iga projektiga suudetakse neid paremini teha. Esialgu tuleks rõhku panna funktsionaalsetele testidele ja ühiktestidele. Ajapikku võib juurde võtta ka teisi varem mainitud teste nagu näiteks teste, mis on loodud programmiteksti põhjal.

Siiski tuleb arvesse võtta, et mõned vead on kriitilisemad kui teised. Oluline on jälgida, et testikirjelduste loomist alustataks nendest funktsionaalsustest, mis on kõige prioriteetsemad. Kui on teada, millised on suurimad riskid, siis nende kohta tuleb ka kõige varem testid teha. Ei ole mõtet testida, kas fondisuurus lehel on liiga väike või suur, kui põhifunktsionaalsuses on vead sees. [14]

7.3 Kaugel ollakse ja mida teha edasi

Mai alguseks ollakse jõutud etappi, kus kirjutatakse ühikteste ja automaatseid kasutajaliidese teste. Ühiktestide kasutusele võtmine on läinud kergelt. Problemaatilisem on olukord testikirjelduste loomisel. Kuna nende loomine võtab aega, siis neid kirjutatakse vähem kui peaks. Siiski algus sellega on tehtud. Järgmises projektis, mis algab neljandal mail, tahetakse ka neid rohkem ja detailsemalt koostada. Teine murekoht on automaatsed kasutajaliidese testid. Neid küll luuakse igale uuele funktsionaalsusele ja lehele, ent tundub, et valitud raamistik ei ole kõige parem. Järgmises projektis mõeldakse ka uuesti läbi, kas kasutatakse sama raamistikku, aga teistmoodi või siis võetakse kasutusele mõni muu raamistik.

8. Tehtud töö analüüs ja arutelu

Üldiselt on sarnaseid töid tehtud vähe ja enamik töid, mis leiti, on juba mitu aastat vanad. 2007 aastal tehti Rootsis Blekinge'i Tehnoloogia Instituudis töö testimise planeerimise ja testi protsessi disaini teemal [15].

Selle töö eesmärgiks, mis ei ole tehtud kindla ettevõtte andmete alusel, vaid teooriale tuginedes, ei olnud küll luua uut arendusprotsessi, ent seal vaadeldi, kuidas mõõta testimise planeerimist ja disainimist ja sellest lähtuvalt loodi testimise protsess. Antud töö on konkreetselt mõeldud ByteLife'is kasutamiseks. Wasifi töös leiti pigem üldisi mõtteid ja ideid, mida ettevõtetes kasutada, ent konkreetset protsessi seal paika ei panda.

2009 aastal ilmunud artiklis [16] uuriti, kas korraga on mõttekas samal ajal kasutusele võtta CMMI teine tase ja samal ajal verifitseerimise ja valideerimise protsess. [16] toodi välja, et ettevõtted kipuvad teist taset alaväärtustama ja tihti tahetakse kohe kasutusele võtta kolmanda taseme protsesse, kuna need on selgemad. Jõuti järeldusele, et siiski ei tasuks teist taset vahele jätta, kuna see tagab toote parema kvaliteedi.

2010 aastal Horvaatias tehtud uuringus arendati tarkvaraarendusprotsessi six sigma meetodiga [17] kuluefektiivsuse saavutamiseks. Püüti protsessi arendada nii, et see oleks kvaliteetne ja kuluefektiivne. Ka see uuring on üldine ega keskendu konkreetsele ettevõttele.

Antud tööd tehes võeti aluseks plaan täita kõigepealt teise taseme nõuded. Uuringus ei ole niivõrd lähtutud finantsilisest küljest, vaid pigem ikkagi kvaliteedi saavutamisest ja konkreetse protsessi loomisest.

8.1 Töö võimalikud edasiarendused

Kuna antud töös vaadeldi ainult CMMI teist taset ja sellest ainult ühte protsessi valdkonda, siis on võimalik teha sarnane töö ka teiste protsessivaldkondade kohta, et ettevõttel tervikuna oleks võimalik jõuda CMMI teisele tasemele. Samuti on kasulik juurutada mõned valdkonnad kolmandast tasemest näiteks verifitseerimine ja valideerimine. Sarnaseid töid võiks teha ka teistes ettevõtetes. Sellisel juhul on võimalik teha üldisem analüüs, milline on olukord Eestis tegutsevates ettevõtetes.

Antud töö edasiarendusena võib aasta pärast teha ByteLife’s analüüsi, kuidas protsessi muutmine on neid mõjutanud ja kas see on ka loodetud tulemusi andnud. Samuti võib uurida, kas protsessi on veel uuendatud ja kui palju protsessi järgitud on.

9. Kokkuvõte

ByteLife Solutions OÜ on tarkvara arendusprotsessis suhteliselt uus ja värske tegija. Iga alguse puhul on loomulik, et ei ole kogemuste puudumisest tingituna arendusprotsessi kvaliteedis taset, mida on saavutanud aastakümneid tegutsenud kolleegid. Tööga uuriti ByteLife tarkvara arendusprotsessi kvaliteedi parandamiseks ja tagamiseks vajalikke tegevusi. Selleks hinnati arendusprotsessi hetkeolukord ByteLife´s CMMI võimekuse ja küpsuse tasanditest lähtuvalt.

Nimetatud analüüsi tulemustest ning ByteLife töötajate arvamusest lähtuvalt jõuti järeldusele, et arendusprotsessi tuleb täiustada põhjalikuma dokumenteerimisega ja testimisprotsessidega. Jõuti järeldusele, et kui protsess on dokumenteeritud, siis on võimalik seda jälgida, muuta ja parandada. Testimine aitab aga tarkvaras esinevaid vigu varem ja suuremas mahus avastada, mistõttu tarkvara on töökindlam ja usaldusväärsem.

Töö neljandas osas leitakse enam levinud testid, mille hulgast leiti sobivamad testimismeetodid ByteLife'i jaoks. Töös loodi täiustatud arendusprotsess, mis vastab ByteLifes kasutatud arendusmetoodikatele ja samuti CMMI teisele tasemele. Jõuti järeldusele, et kui projekti käigus tuleb funktsionaalsetes nõuetes ette muudatusi, siis tuleb need lisada ka dokumentatsiooni, samuti tuleb luua testikirjeldused, mille alusel testitakse funktsionaalsusi. Testimisprotsessi lisati ka ühiktestide kirjutamine, automaatsete kasutajaliidese testide loomine ja testimine programmiteksti põhjal.

Töö üheks eesmärgiks oli leida praeguse protsessi nõrgad kohad ja protsessi nii täiendada, et loodav toode oleks kvaliteetne. Vahetulemusena leiti esialgse protsessi puudused. Töö lõpp-tulemuseks oli uus täiustatud arendusprotsess, mis aitab luua kvaliteetsemaid tooteid. Samuti on protsess jälgitav ja kontrollitud. Protsessi juurutamisega on ettevõttes juba ka algust tehtud, ent kasulikkus tuleb välja hiljem.

Summary

ByteLife Solutions OÜ is a quite new company in software development business. It is natural there is no such quality of process because of the lack of experience as to compare with colleagues who have been in the IT area for years. The incentive to the research was statement that the quality of the software development process is not satisfactory. Team members of Bytelife wanted changes in the development process, in order to reach in a higher level of product quality.

In the research, the qualitative methods were used. The main method was to apply CMMI – a model providing guidance for applying CMMI best practices in a development organization. Best practices in the CMMI model focus on activities for developing quality products and services to meet the needs of customers and end users.

In the research, the software development process in ByteLife Solutions OÜ was measured against the CMMI capability maturity model and the level was specified. In order to reach defined capability and maturity levels, general and specific goals and practicies were found. Main reasons of the inefficiency in the process of software development process in ByteLife Solutions OÜ were found to be in the lack of control of the product development in the early stage of the process and insufficient written documentation of amendments and agreements during the development procedure.

According to the research, alternative solutions to increase the quality of the process were suggested: to introduce the process of defined testing to the living model effective in other areas of the company and to satisfy preferences of the team of the company, to apply more testing to the procedure and to consistently document the amendments and agreements during the procedure. The last activity helps to avoid mistakes done by misunderstanding. The specific testing activities for the following project of the company were suggested to apply the new method of the process. According to the research, the unit tests, functional tests and testing according to the text of the program would be appropriate to increase the quality of the product.

All this should make the development process of the company more predictable and the quality of the process should increase. Costs of the new method were not measured, but it is expected to have to consider with the cost of additional time for added testing activities. After the enough experience is gathered, the defined capability and maturity levels are reached by the company. The well-functioning teamwork and cooperation should be continued, as well as the staff

motivation program, including providing training opportunities to achieve the similar knowledge level.

10. Kasutatud kirjandus

[1] K. Schwaber ja J. Sutherland, „The Definitive Guide to Scrum: The Rules of the Game,“ 2013. [Võrgumaterjal]. Saadaval: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100. [Kasutatud 26 aprill 2015].

[2] „Atlassian Documentation,“ Atlassian, [Võrgumaterjal]. Saadaval: https://confluence.atlassian.com/display/AGILE/Kanban. [Kasutatud 2015 mai 4].

[3] K. Beck, Extreme Programming Explained Second Edition, Massachusetts: John Wait, 2012.

[4] A. Sharma, „Scrum Alliance,“ 20 detsember 2013. [Võrgumaterjal]. Saadaval: https://www.scrumalliance.org/community/articles/2013/december/essential-valuable-timely-documentation. [Kasutatud 4 mai 2015].

[5] „CMMI technical report,“ [Võrgumaterjal]. Saadaval: http://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf . [Kasutatud 28 märts 2015].

[6] J. Tepandi, „Tarkvara kvaliteet ja standardid-loengukonspekt,“ 2007. [Võrgumaterjal]. Saadaval: http://www.lap.ttu.ee/erki/failid/konspekt/tarkvara_kvaliteet_ja_standardid_idx5721/idx5721_konspekt.pdf. [Kasutatud 8 märts 2015].

[7] M. Khalili, „Maintainable Automated UI Tests,“ 9 Oktoober 2013. [Võrgumaterjal]. Saadaval: http://code.tutsplus.com/articles/maintainable-automated-ui-tests--net-35089. [Kasutatud 29 Aprill 2015].

[8] P. Hamill, „Unit test frameworks,“ Ameerika ühendriigid, 2004.

[9] "Certified Tester - Foundation Level Syllabus", International Software Testing Qualifications Board, 2011.

[10] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons ja S. V, „Using formal specifications to support testing," Ameerika Ühendriigid, 2009.

[11] „ISTQB Exam certification," [Võrgumaterjal]. Saadaval: http://istqbexamcertification.com/what-is-system-testing/. [Kasutatud 29 aprill 2015].

[12] „How to write effective Test cases, procedures and definitions," 12 veebruar 2015. [Võrgumaterjal]. Saadaval: http://www.softwaretestinghelp.com/how-to-write-effective-test-cases-test-cases-procedures-and-definitions/. [Kasutatud 2 mai 2015].

[13] H. Kniberg, „Kanban vs Scrum – How to make the most of both," 2009. [Võrgumaterjal]. Saadaval: https://www.crisp.se/file-uploads/Kanban-vs-Scrum.pdf. [Kasutatud 26 aprill 2015].

[14] Atlassian, „Atlassian," [Võrgumaterjal]. Saadaval: https://www.atlassian.com/software-testing/?tab=exploratory-software-testing. [Kasutatud 2 mai 2015].

[15] W. Afzal, *Metrics in Software Test Planning and Test,* Rootsi, 2007.

[16] P. Monteiro, R. J. Machado ja R. Kazman, *Inception of Software Validation and Verification Practices within CMMI Level 2,* 2009.

[17] I. Sinovčić ja L. Hribar, *How to improve software development process using mathematical models for quality prediction and elements of Six Sigma methodology*, Horvaatia, 2010.

[18] planIt, „planit,“ [Võrgumaterjal]. Saadaval: http://www.planit.net.au/testing-services/system-integration-uat/#user-acceptance-testing. [Kasutatud 26 aprill 2015].

Lisa 1

Küsimustik

Küsimustik on näitlik ja selle loomisel on kasutatud FlowGrabi projekti. Küsimustik ei ole lõplik, tegelik küsimustik oleks mahukam.

1. Andmete salvestamise kohta andmebaasis
    - Kas väljapikkused on määratud?
    - Kas kõik kohustuslikud väljad on määratud?
    - Kas kõik suured tähed muudetakse salvestamisel väikesteks (kasutajanime puhul)?
    - Kas paroolid on krüpteeritud?
2. Andmete kohta
    - Kas vaikimisi atribuudid on antud õigest?
    - Kas andmed on õigesti initsialiseeritud?
    - Kas muutujad on sarnaste nimedega?
    - Kas muutujad on arusaadavate nimedega?
3. Pöördumine andmete poole
    - Kas pöördumisel on muutujal väärtus olemas?
    - Kas indeks võib minna väljapoole lubatud piire?
    - Kas indeksile omistatakse täisarv?
4. Juhtimise vead
    - Kas tsüklid lõpetavad oma töö?
    - Mis juhtub, kui tsükli tingimus on kohe vale?
    - Kas on vaadatud kõrvalekaldumisi (nt suur kordsute arv)?
5. FlowGrab: projekti loomise õigused
    - Kas kasutaja saab projekti luua oma tiimi?

- Kas ainult lugemisõigusega kasutaja näeb projekti?
- Kas ainult lugemisõigusega kasutaja saab muuta projekti?
- Kas arendaja saab projekti puhul teha *commiti* ja *pushi*?
- Kas arendaja saab luua ise projekti?
- Kas arendaja saab muuta projekti?

6. FlowGrab: administraatori õigused FlowGrabis

    - Kas administraator saab kasutada *dashboardi*, sealhulgas luua projekte?
    - Kas administraator saab luua tiime?
    - Kas administraator saab lisada kasutajaid ja kustutada olemasolevaid kasutajaid?
    - Kas administraator saab ennast ära kustutada?

Lisa 2

Teststsenaariumi loomisel on kasutatud projekti FlowGrab andmeid. Testikirjeldus on näitlik, konkreetsed testid võivad olla nii üldisemad kui ka konkreetsemad.

Test nr 1	
Kokkuvõttev kirjeldus	Testitakse projekti loomist korrektsete andmetega
Eeltingimused	Olemas on kasutaja, kes saab luua projekti. Eelnevalt on vaja luua pakett
Testandmed	Projekti nimi: nimi Kirjeldus: test projekt Tiim: kasutaja vaiketiim Veebileht: www.test.ee Märked (*tagid*): test Litsents: valida valikust “Apache License” Kategooria: valida valikust “OS Deployment” Pakett: kasuta loodud paketti

Testi sammud

#	Tegevus	Oodatav tulemus	Tegelik tulemus
1.	Vajuta nuppu “loo projekt”	Avaneb projekti loomise vorm	10.04.2015 – OK
2.	Täida väljad andmetega	Ühegi välja juures ei kuvata veateateid	10.04.2015 – OK

3.	Vajuta „Salvesta“ nuppu	Ilmub teade „Projekti loomine õnnestus“	10.04.2015 – teadet ei ilmunud 12.04.2015 - OK
		Projekt on projektide listis	10.04.2015 – OK
4.	Vajuta nuppu „vaata projekti“ andmete kontrollimiseks	Avaneb projekti vaatamise vorm	10.04.2015 – OK
		Lehel on kuvatud kõik andmed, mis varem sisestati	10.04.2015 - Veebileht on puudu 12.04.2015 - Veebileht on puudu
Kas kasutajaliidese test on tehtud?	**Ei**		