Tallinna Ülikool

Informaatika Instituut

Agiilsete tarkvaraarendusmeetodite võrdlus

Seminaritöö

Autor: Kristen Kivimaa

Juhendaja: Inga Petuhhov

Tallinn 2014

Sisukord

Sissejuhatus

Tänapäeva pidevalt arenevas IT-maailmas on suur tähtsus tarkvaraarendamisel. Kui eelmisel sajandil ainuõigeks peetud arendusviisid enam ei sobinud lihtsakoeliste ning pidevalt muutuvate programmide väljatöötamiseks, loodi uudne meetod, mis sai nime agiilne tarkvaraarendus. Seda hakkasid kasutama mitmed erinevad tarkvaraarendajad, kuid nad tegid seda kõik isemoodi, mille tõttu kujunesid välja erinevad agiilse tarkvaraarenduse meetodid, millel kõigil on omad ühised jooned ning samuti ka mitmed erinevused.

Käesoleva seminaritöö teema valimise põhjuseks on töö autori isiklik huvi erinevate agiilsete tarkvaraarendusmeetodite vastu. Kuna autor on külastanud mitmeid erinevaid tarkvaraarendusfirmasid, siis on märgatud, et paljud nendest kas juba arendavad tarkvara agiilselt või üritavad lisada praegustele meetoditele ka agiilseid külgi.

Antud seminaritöö eesmärk on tutvustada agiilset tarkvaraarendust ning võrrelda omavahel mõningaid meetodeid. Samuti on eesmärk tuua välja erinevused traditsioonilise ning agiilse lähenemise vahel.

Käesolev töö on jagatud kaheks suureks peatükiks. Esimese peatüki esimeses pooles tutvustatakse agiilse tarkvaraarenduse ajalugu ning seletatakse, kuidas arendati tarkvara enne 1990ndaid. Peatüki teises pooles võrreldakse neid kahte arendusviisi. Võrdluse aluseks võetakse erinevad skeemid ning tabelid.

Teise peatüki on autor jaganud neljaks peatükiks, kus esimesed kolm tutvustavad kõik erinevat agiilset tarkvaraarendusmeetodit, milleks on *Scrum* meetod, Ekstreemprogrammeerimine ning *Crystal* meetod. Peatüki viimases osas võrreldakse kõiki kolme meetodit. Võrdluse aluseks võetakse Agiilse Tarkvaraarenduse Manifesti põhitõed ning selgitatakse, kuidas iga meetod neid täitab.

Seminaritöö koostamisel suuri probleeme ei tekkinud, küll aga osutus raskeks paljude sõnade tõlkimine eesti keelde. Kuna pea kõik materjalid, kust informatsioon hangiti, olid ingliskeelsed, siis ei leitud tihtipeale kasutatavaid eestikeelseid vasteid, milletõttu võivad osad väljendid olla ebaharilikud.

1 Agiilne tarkvaraarendus

Tarkvaraarenduse meetodeid on võimalik jagada mitmete tunnuste järgi erinevatesse rühmadesse. Tihtipeale jaotatakse need aga kaheks: agiilseteks ning traditsioonilisteks tarkvaraarendusmeetoditeks. Mainitutest esimene on uuema ajastu lähenemisviis, mille väljakujunemisele aitas suuresti kaasa programmeerijate rahulolematus sel ajal kasutuses olnud traditsioonilistele meetoditega.

1.1 Agiilse tarkvaraarendamise ajalugu

1980ndatel ja 1990ndate alguses oli levinud arvamus, et parim viis saavutada hea tarkvara on hoolikas projektiplaneerimine, formaliseeritud kvaliteeditagamine, arvutipõhiste tarkvaratehnikate kasutamine keerukate analüüside ning projekteerimismeetodite puhul ja kontrollitud ning range tarkvara arendamise protsess. Selline arusaam tuli tarkvaratehnika kogukonnalt, mis oli harjunud välja töötama suuri, pika elueaga tarkvarasüsteeme, nagu näiteks kosmose ja valitsuse süsteeme (Sommerville, 2011).

Selline tarkvara oli arendatud suurte meeskondade poolt, mis tihtilugu töötasid erinevate ettevõtete käsutuses. Meeskonnad olid sageli geograafiliselt hajutatud ja tarkvaraarendusprotsess kestis reeglina pikka aega. Näiteks seda tüüpi tarkvara on kasutusel õhusõidukite juhtimissüsteemides, mille arendamine võib kesta kuni 10 aastat alates esmasest spetsifikatsioonist kasutuselevõtuni. Sellised plaani järgivad lähenemisviisid toovad kaasa lisakulutusi süsteemi planeerimisel, projekteerimisel ja dokumenteerimisel. See õigustab ennast ainult siis, kui töö mitme arendusmeeskondade vahel peab olema koordineeritud, kui tegemist on kriitilise süsteemi loomisega ja kui tarkvara eluea jooksul tegeleb selle hooldusega mitmed erinevad inimesed (Sommerville, 2011).

Samas, kui seda raskekoelist plaan - orienteeritud tarkvaraarengu lähenemisviisi rakendatakse väikeste ja keskmise suurusega ettevõtete süsteemidele, võivad üldkulud tõusta nii suureks, et sellise tarkvara väljaarendamine ei tasu ennast ära. Rohkem aega kulub sellele, kuidas süsteemi tuleks arendada, kuigi peaks hoopiski kuluma programmi väljatöötamisele ja katsetamisele. Niipea, kui süsteemi nõuded muutuvad, tuleb hakata juba valmisoleva programmi spetsifikatsiooni ja disaini ümber tegema (Sommerville, 2011).

Rahulolematus nende raskekoeliste lähenemisviisidega tarkvaratehnikas viis selleni, et hulk tarkvaraarendajad tegid 1990ndal aastal ettepaneku luua uus meetod, mille nimetasid kergeks tarkvaraarenduseks. See lubas arendusmeeskondadel keskenduda tarkvarale endale, mitte niiväga enam disainile ja dokumentatsioonile (Sommerville, 2011).

Üldiselt lubasid kerged meetodid arendajatel tegeleda tarkvara spetsifikatsiooni, arenduse ja tarnimisega järkjärguliselt. Need sobivad kõige paremini selliste rakenduste arendamiseks, mille süsteeminõuded kipuvad tavaliselt kiiresti muutuma programmi väljatöötamise protsessi käigus. Nende eesmärk on kiiresti esitada töötav tarkvara klientidele, kes saavad seejärel välja pakkuda uusi ja muudetud nõuded, mis lisatakse programmile edaspidiste muudatuste käigus. Nende eesmärk on vähendada protsessi bürokraatiat vältides tööd, millel on kaheldav pikaajaline väärtus ja kaotades dokumentatsioon, mida ilmselt kunagi ei kasutata (Sommerville, 2011).

Sellest ajast alates hakkasid välja kujunema erinevad viisid, kuidas arendada tarkvara just seda uut meetodit kasutades. Varajaste meetodite hulka kuulusid *Rational Unified Process*, *Scrum*, *Crystal Clear*, *Extreme Programming*, *Adaptive Software Development*, *Feature Driven Development* ja *Dynamic Systems Development* metoodikad. Sel ajal polnud veel üheselt kirja pandud, mis määrab ära, kas tarkvaraarendamist saab nimetada kergeks või mitte (Larman, 2004).

Kuna erinevate arendusviiside kasutajad tundsid vajadust luua ühtne dokument, mis määraks ära, kuidas peaks tarkvara kergelt arendama, kohtusid 2001. aasta veebruaris erinevate tarkvaraarendusmeetodite juhtivad esindajad Snowbird'i suusakeskuses, Utah'is, et püüda leida selles küsimuses ühine keel. Seal lepiti kokku, et edaspidi hakatakse sellist lähenemisviisi nimetama agiilseks tarkvaraarenduseks ning pandi kirja „Agiilse Tarkvaraarenduse Manifest" (Beck jt, 2001c).

Manifest määratleb lähenemisviisi, kuidas peaks toimima agiilne tarkvaraarendus. Mõned manifesti autorid moodustasid Agiilse Liidu (*Agile Alliance*), mittetulundusliku organisatsiooni, mis edendab tarkvaraarendust vastavalt manifestis määratud väärtustele ja põhimõtetele (Beck jt, 2001c). Kuna tippkohtumisel oli esindatud seitseteist erinevate arvamustega praktikut, siis ei olnud nad paljudes asjades ühel nõul, kuid siiski lepiti kokku üksmeelselt neli põhiväärtust (Ocamb, 2013). Manifest ütleb (Beck jt, 2001a):

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

inimesi ja nendevahelist suhtlust rohkem, kui protsesse ja arendusvahendeid,

töötavat tarkvara rohkem, kui kõikehõlmavat dokumentatsiooni,

koostööd kliendiga rohkem, kui läbirääkimisi lepingute üle,

reageerimist muutunud oludele rohkem, kui algse plaani järgimist.“

Manifestis on veel ära kirjeldatud 12 põhimõtet, mida peaksid järgima kõik arendajad, kes soovivad teha tarkvara agiilselt. Nendeks on (Beck jt, 2001b):

- „Kõige olulisem on tagada kliendi rahulolu, tarnides talle vajalikku tarkvara võimalikult kiiresti ja tihti“.
- „Mõistame muutuvaid olusid, isegi kui need ilmnevad arenduse lõppjärgus. Agiilsed meetodid pööravad sellised muutused meie kliendi konkurentsieeliseks“.
- „Tarnime tarkvara nii tihti kui võimalik, soovitavalt iga paari nädala kuni paari kuu tagant“.
- „Valdkonna spetsialistid ja tarkvaraarendajad peavad töötama igapäevaselt koos kogu projekti vältel“.
- „Projekti edukuse aluseks on motiveeritud inimesed. Loo neile meeldiv ja toetav töökeskkond ning nad saavad iseseisvalt tööga hakkama“.
- „Kõige tõhusam ja tulemuslikum viis info jagamiseks arendusmeeskonnas on näost näkku vestlus“.
- „Edu peamiseks mõõdupuuks on töötav tarkvara“.
- „Agiilse tarkvaraarenduse protsessid soodustavad jätkusuutlikku arendust. See tähendab, et projektiga saab samas tempos jätkata määramata aja jooksul“.

- „Tehnilist täiuslikkust ja head disaini pideva tähelepanu all hoides tagatakse tarkvaraarenduse kiirus ja paindlikkus“.

- „Lihtsus - ebavajaliku töö tegematajätmise kunst - on väga oluline“.

- „Parimad arhitektuurilised lahendused, nõuded ja disain tekivad iseorganiseeruvates meeskondades“.

- „Meeskond otsib regulaarselt võimalusi saamaks veelgi tõhusamaks ja muudab end vastavalt vajadusele“.

Alates sellest ajast võib rääkida agiilsest tarkvaraarendusest kui arendusmetoodikast, millel on omakorda veel mitmeid meetodeid, mis on omavahel küllaltki erinevad, kuid järgivad ikkagi ühiseid printsiipe, mis on kirja pandud Agiilse Tarkvaraarenduse Manifestis.

1.2 Agiilse tarkvaraarendusmeetodite võrdlus traditsiooniliste lähenemistega

Eelnevas peatükis on kirjeldatud põgusalt nii agiilset kui ka traditsioonilist meetodit. Sommerville'i (2011) aurutluste kohaselt iseloomustab traditsioonilise meetodi arenduslaade hoolikas projektiplaneerimine, formaliseeritud kvaliteeditagamine, arvutipõhiste tarkvaratehnikate kasutamine keerukate analüüside ning projekteerimismeetodite puhul ja juhitud ning range tarkvara arendamise protsess.

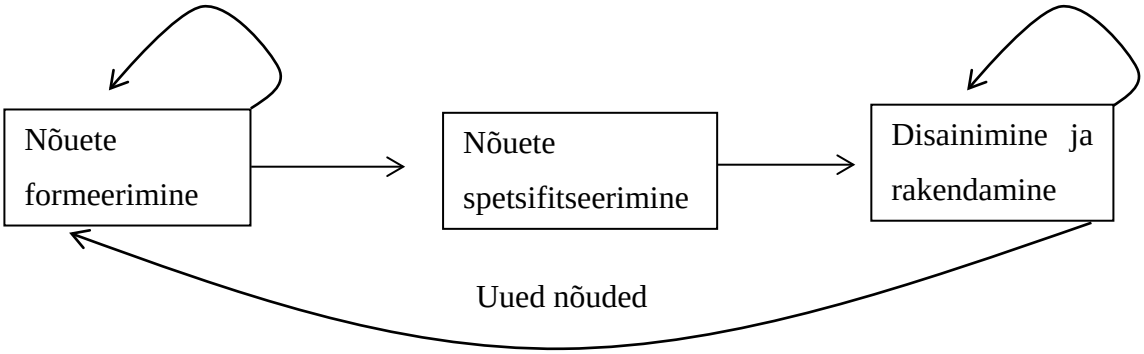
Klient näeb valmis toodet alles lõppfaasis ning seetõttu võib juhtuda, et arendajad on mahukast dokumentatsioonist aru saanud erinevalt kui selle kirjapanijad mõelnud on. Teiselt poolt, kui projekti alguses soovitakse teatud kindlat rakendust, siis aja jooksul võivad soovid muutuda ning traditsiooniliste tarkvaraarendusmeetodite puhul on nende muudatuste rakendamine väga keeruline ja see pikendab oluliselt toote valmimise aega. Kui viiakse läbi uuendus, siis tuleb hakata otsast peale dokumentatsiooni muutma ning enne arendamise juurde asumist eelneb sellele veel selle analüüsimine ja spetsifitseerimine (Ghilic-Micu jt, 2013).

Agiilsed lähenemisviisid peavad kavandamist ja realiseerimist peamisteks ning tähtsaimateks tegevusteks tarkvaraarenduse protsessis. Need kaasavad ka muid tegevusi, nagu vajaduste

väljaselgitamine ja testimine toote kavandamisel ja rakendamisel (Sommerville, 2011). Arendatava tarkvara nõuded jagatakse ära väikesteks kasutajalugudeks ning seejärel hakatakse neid ühekaupa realiseerima. See annabki võimaluse uuenduste sissetoomiseks keset arendamist, sest igal osal on omad nõuded ning neid saab eraldi käsitleda (Pammy, 2013).

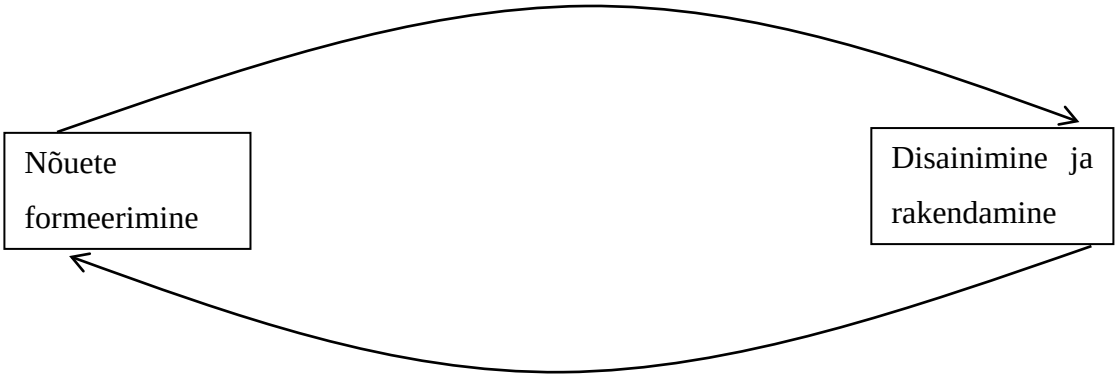
Täpsema ülevaate traditsioonilisest ning agiilsest tarkvaraarendusmeetodist annavad joonis 1 ning joonis 2.

Traditsiooniline lähenemisviis:



Joonis 1. Traditsioonilise tarkvaraarendusmeetodite kirjeldus (Sommerville, 2011)

Agiilne lähenemisviis:



Joonis 2. Agiilse tarkvaraarendusmeetodite kirjeldus (Sommerville, 2011)

Järgnev tabel 1 annab ülevaate agiilsete ning traditsiooniliste meetodite erinevustest.

Tabel 1. Traditsiooniliste ja agiilsete tarkvaraarendusmeetodite võrdlus (Ghilic-Micu jt. 2013; Mahapatra jt. 2005)

	Traditsiooniline tarkvaraarendus	**Agiilne tarkavaraarendus**
Juhtimisstiil	Käskiv ning kontrolliv	Koostööl põhinev
Suhtlus kliendiga	Ametlik	Mitteametlik
Arendusmudel	Elutsükli mudel (tarkvara spetsifitseerimine, arendamine, testimine toimub eraldi tsüklitena)	Evolutsiooniline mudel (arendamist, testimist tehakse kogu aeg, pidevalt)
Organisatsiooniline korraldus	Mehhaaniline, suunatud suurtele projektidele	Orgaaniline, suunatud väikestele ja keskmistele projektidele
Kvaliteedi tagamine	Keerukas planeerimine ja range kontroll	Nõuete, disaini ja lahenduste pidev kontroll
Arenduse suunitlus	Kindlalt määratletud	Kergesti muudetav
Kliendi kaasatus	Madal, klient näeb toodet siis, kui rakendus on valmis	Kõrge, klient on osa arendajate meeskonnast ning näeb tulemust pärast igat muudatust
Testimine	Pärast koodi valmimist	Arenduse käigus, pidevalt
Täiendavad nõudmised arendajatele	Pole	Suhtlemisoskus ja põhiteadmisi kliendi ärist
Projekti sobiv suurus	Suuremad projektid	Väiksemad ja keskmised projektid
Nõuded	Stabiilsed, projekti alguses teada	Pidevalt muutuvad
Esmane eesmärk	Kõrge turvalisus	Kiire tulemus
Meeskonna suurus	Suur	Väike

Tabel 1 näitab, et agiilsed meetodid erinevad traditsioonilistest küllaltki palju. Kuna paindlike meetodite puhul on tarkvara arendav meeskond väike, ei teki seal ka tunnetatavat bürokraatiat, mistõttu juhtimisstiil on peamiselt koostööl põhinev. Klienti peetakse meeskonna liikmeks, kes osaleb koosolekutel ning näeb projekti kulgu selle arenemise käigus. Kuna nõuded võivad olla pidevalt muutuvad, siis see hoiab ära olukorrad, kus projekti lõppfaasis selgub, et valminud tarkvara pole päris see, mida telliti. Agiilsetes tarkvaraarendusmeeskondades peetakse väga oluliseks suhtlemisoskust. Kuna iga inimene ei

ole harjunud töötama grupis, siis selline tihe meeskonnasisene töö ei pruugi sobida igale programmeerijale.

Traditsiooniliste meetodite puhul toimub arendus elutsükli mudelit järgides. See tähendab, et ühe projektiga tegelevad mitmed erinevad meeskonnad. Näiteks koskmudeli korral on nendeks nõuete analüütikud, disainerid, programmeerijad, testijad ja hooldajad. See on ka põhjuseks, miks meeskonnad peavad olema suured ning projekti juhtimisstiil range ning käskiv. Kuna tarkvara nõuded on juba projekti alguses määratletud ning tavaliselt ei muutu, siis pole ka arendajatele täiendavaid nõudmisi, nagu seda oli agiilsete meetodite puhul. Arenduse kvaliteet tagatakse range kontrolliga nõuete järgimise üle ja arenduse keeruka planeerimisega, mis on võimalik just seetõttu, et projekti alguses on dokumentatsioonis projekti nõuded spetsifitseeritud.

Agiilsed tarkvaraarendusmeetodid on eelistatud väiksemate ning keskmise suurusega projektide korral, kus on tähtis, et toode valmiks kiiresti ning odavamalt. Traditsioonilist viisi aga kasutatakse suuremamahuliste arendamiste puhul, kus seatakse eesmärgiks kõrge turvalisus.

Tänapäeval on ühe enam hakanud levima agiilsed meetodid, sest tarkvaraarenduses on nüüdisajal väga raske ette ennustada, millised probleemid võivad arenduse käigus ette tulla. Muudatust vajavate tõrgete korral on neid lihtsam parandada väledaid meetodeid kasutades, kuna probleemidele saab kohe lahendust otsima hakata sama väike meeskond, mis projektiga tegeleb. Traditsiooniliste meetodite puhul on enne muutuste läbiviimist vaja neid erinevate meeskondade poolt analüüsida ja disainida, enne kui programmeerijad neid täide saavad hakata viima.

2 Agiilsed tarkvaraarendusmeetodid

Järgnevas peatükis tutvustab autor erinevaid agiilseid tarkvaraarendusmetoodikaid ning võrdleb neid. Kuna erinevaid variante, kuidas arendada paindlikult on palju, valiti uuritavateks meetoditeks enim kasutatavad väledad arendusmeetodid maailmas Azizyan jt. (2011) poolt läbi viidud küsitluse järgi. Antud uuringust võtsid osa 120 ettevõtet 35-st erinevast riigist. Uuringust selgub, et üle poolte vastanutest, 54 protsenti, kasutavad tänapäeval *Scrum* meetodit. Kolmkümmend kaks protsenti vastanutest arendavad tarkvara nii, et kastutakse *Scrum*'i koos Ekstreemprogrammeerimisega. Puhtast XP-d kasutavad juba märgatavalt vähem ettevõtteid ning uuringust selgub et vastanutest kõigest 11 protsenti kasutavad seda meetodit. Ülejäänud agiilsed tarkvaraarendusmeetodeid kasutavad juba alla kümne protsendi vastanutest. Nimetatud meetodid olid *Kanban*, *Crystal*, Dünaamiline tarkvaraarendus, Testidel põhinev arendus ning Omaduspõhine tarkvaraarendus.

2.1 *Scrum* meetod

Scrum on üks populaarsemaid agiilseid tarkvaraarendusmeetodeid, mida tänapäeval kasutatakse. Azizyan jt. (2011) uuringu järgi kasutavad just seda meetodit üle poolte ettevõtetest, mis tegelevad agiilsete meetodite rakendamisega.

Scrum on meetod, mille eesmärk on aidata väikestel, tihedalt seotud meeskondadel arendada keerulisi tooteid. (Louise & Sims 2012:). Selle meetodi eesmärgiks on hoida komplitseeritud ärikeskkonda lihtsana. Termin pärineb ragbist, kus see on strateegia, mille eesmärk on vastastelt saada tagasi kaotatud pall kasutades selle jaoks meeskonnatööd. *Scrum* meetodiga arendades ei ole paika pandud kindlat eelarvet enne projekti. See kui palju antud programmi arendamine maksma läheb, selgub arenduse lõppjärgus. Rohkem keskendutakse hoopis sellele, kuidas peaksid arendusmeeskonnad vastastikku toimima, et luua paindlikku, adaptiivset ja produktiivset süsteemi pidevalt muutuvas keskkonnas. Meetodi on välja töötanud ja arendanud Ken Schwaber ja Mike Beedle (Ghilic-Micu jt, 2013).

Scrum meetodit viljelevates meeskondades on tegevus ära jaotatud kolme erineva rolli vahel, milleks on toote omanik, *Scrum* meister ning meeskond.

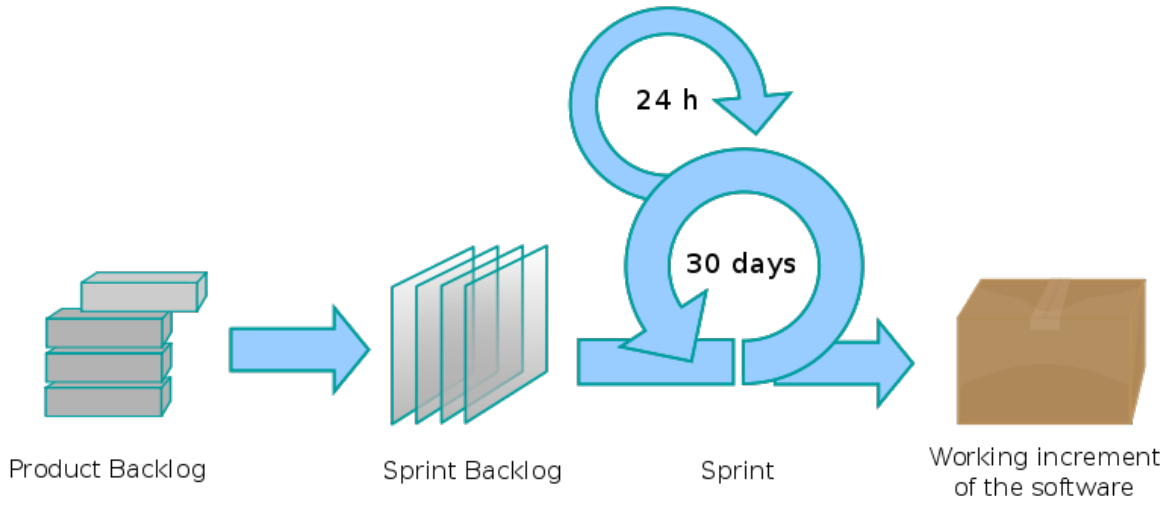
Toote omanik on inimene, kes vastutab nende ees, kes on toote või süsteemi tellinud. Tema ülesandeks on olla vahelüli kliendi ja meeskonna vahel. Ta peab tagama esialgse projekti rahastamise ning veenduma, et seda jätkuks terveks tootearendustsükliks. Toote omanik esindab kõiki, kes on panustanud projekti ja selle tulemusse. Projekti alguses kogub ta esialgsed nõuded ning paneb need tööde nimekirja. Tema otsustab, millised nõuded on tähtsamad ning millises sprindis peaks neid arendama hakkama. Projekti käigus toote omanik kontrollib projekti edenemist ning kliendi vajaduste täitmist (Rossberg 2008).

Meeskond vastutab toote arendamise eest. Seal pole igal arendajal kindlat rolli, vaid kõik meeskonnaliikmed peavad saama hakkama kõikide vajalike töödega, mida tarkvaraarenduses vaja on. Kuna meeskond on polüfunktsionaalne ja iseorganiseeruv, peavad nad ise tagama selle, et neil oleksid vajalikud oskused toote valmistamiseks. Meeskonnas on keskmiselt seitse inimest. Tavaliselt on seal kaks programmeerijat, kaks testijat, üks analüütik-disainer ning üks dokumenteerija. Meeskond määrab ära, milliste nõuete realiseerimisega tegeletakse iga arendustsükli jooksul (Rossberg 2008).

Scrum meister on vastutav *Scrum* skeleti järgi arendamise eest ning peab tagama, et kõik meeskonnaliikmed, toote omanik ning teised projekti kaasatud inimesed teavad ning mõistavad projekti. Ta jälgib, et kõik järgiksid reegleid, mille seab ette *Scrum* meetod. Ta ei korralda meeskonnatööd. Kui tekivad konfliktid, siis *Scrum* meister peaks need lahendama ning tagama meeskonna rahuliku ning vajaliku töökeskkonna (Rossberg 2008).

Scrum põhineb kahel teguril: meeskonna suhteline iseseisvus ja kohanemisvõime. Meeskonna suhteline iseseisvus tähendab seda, et projekti omanik loob ülesanded, mida meeskond peab tegema, kuid igas tarkvaraarendustsüklis on meeskonnal õigus otsustada, kuidas töötada, et suurendada meeskonna tootlikkust. *Scrum* ei nõua spetsiifilist tarkvaraarengu tehnikat, kuid meeskonnad peavad järgima meetodi põhiskeemi erinevate arendusfaaside jaoks, et vältida segaduse tekkimist, mis võivad tuleneda projekti keerukusest ja ettearvamatusest (Ghilic-Micu jt, 2013).

Seda skeemi, mida kutsutakse ka *Scrum*'i skeletiks, kujutab joonis 3.



Joonis 3. *Scrum*'i skelett (Software Tools, 2013)

Iga sprindi (*sprint*) ajal, mis on tavaliselt 2 – 4 nädalat pikad, loob meeskond potentsiaalselt valmis tarkvara tüki, mis töötab ning mida on testitud. Nõuded (*sprint backlog*), mida soovitakse realiseerida ühe sprindi jooksul on kirja pandud toote tööde nimekirjas (*product backlog*), kus on kõik programmi nõuded järjestatud nende tähtsuse järgi. Seda, milliseid nõudeid sprindi jooksul arendama hakatakse, otsustatakse sprindi planeerimise koosolekul. Selle koosoleku ajal teavitab toote omanik meeskonda, milliseid toote osi peaks meeskond arendama hakkama sprindi jooksul ning seejärel määrab meeskond ära, kui palju saaksid nad soovitud nõuetest täita. Sprindi ajal ei ole kellelgi lubatud muuta sprindi tööde nimekirja. Iga päev algab meeskonnakoosolekuga, kus arutatakse, milliste nõuetega tegeleti eelmisel päeval ning millega tegeletakse eelseisval päeval (Software Tools, 2013).

Sprindi kestvus on kindlalt määratletud, mis tähendab, et sprint peab lõppema kindlal ajal. Kui mistahes põhjustel tingimusi ei täideta antud tähtajaks, jäetakse need nõuded realiseerimata ning minnakse tagasi toote tööde nimekirja juurde ning peetakse uus sprindiplaneerimiskoosolek (Software Tools, 2013).

Meeskond on isemajandav ja delegeerib ülesandeid iseseisvalt ning on pidevas koostöös toote omanikuga. Sprindi lõpus vaadatakse koos projekti tellijaga üle valminud programmi osa ning ühiselt määratletakse muutused ning lisafunktsioonid, mis järgmiseks korraks esitleda tuleb (Rossberg, 2008).

2.2 Ekstreemprogrammeerimine

Ekstreemprogrammeerimine (XP) töötati välja, et tegeleda tarkvaraarendusega väikestes rühmades ning toime tulla ebamääraste ja muutuvate nõudmistega. See agiilne meetod vastandub paljuski traditsioonilistele tarkvaraarenduse eeldustele, nagu näiteks, et valmis tarkvara ühe osa muutmine on kallis ning keeruline protsess (Beck, 1999).

XP algatajate eesmärk oli luua arendusmeetod, mis oleks sobilik objektorienteeritud projektide arendamiseks, mille kallal töötavad kuni kümneliikmelised meeskonnad (Williams, 2007). Meetod põhineb neljal põhiväärtusel: suhtlus, lihtsus, tagasiside ning julgus (Hunt, 2006).

XP eesmärk on hoida inimestevahelist suhtlust, kasutades erinevaid praktikaid, mida ei saa läbi viia suhtlemata. Need tegevused, nagu näiteks ühiktestimine, paarisprogrammeerimine ning ülesande hindamine, on mõistlikud isegi lühiajalisel kasutamisel. Nende praktikate mõju on see, et programmeerijad, kliendid ning juhid peavad omavahel suhtlema (Beck, 1999). Kommunikatsiooni väärtus põhineb tähelepanekul, et enamus projekti probleemidest tulenevad sellest, et keegi oleks pidanud rääkima kellegagi, et selgitada küsimusi või saada abi (Williams, 2007).

Lihtsuse eesmärk on luua lihtsaim toode, mis vastab tellija nõuetele. Selle märksõna oluline aspekt on, et disainitakse ning kodeeritakse ainult seda, mida on klient nõudnud, mitte ennetada ja planeerida määratlemata vajadusi (Williams, 2007). Mida lihtsam on toote kood, seda kergem on leida sealt vigu ning neid parandada. See ei tähenda, et lahendus peab olema tingimata lihtne või tühine, vaid peaks olema lihtsaim lahendus probleemi jaoks (Hunt, 2006).

Tagasiside toimib erinevatel tasemetel. Programmeerija kirjutab ühikteste iga programmi osa kohta, mis võib katki minna. Kui tarkvarasse lisatakse mõni uus tükk, jooksutatakse kõiki teste, et näha, kas erinevad programmi osad ka üheskoos töötavad. Samuti saavad programmeerijad tagasisidet inimeselt, kes jälgib töö progressi. Ta vaatab, et kõik eesmärgid, mis on endile teatud ajaks määranud, saadakse täidetud (Beck, 1999). Veel saab arendusmeeskond kliendilt tagasisidet pärast iga iteratsiooni ja väljalaset, kus tellijal on võimalus teatada, mida peaks antud programmitüki juures muutma (Williams, 2007).

Eelnevad kolm väärtust lubavad arendusmeeskonnal olla julged. Seda on vaja, et muuta olemasolevat koodi nii, et see oleks parem kui enne, kuid selliselt, et see ei tooks kaasa muudatusi koodi funktsionaalsuses. Näiteks on vaja julgust, et ära visata koodi osa, mis on vananenud, ükskõik kui palju vaeva on nähtud selle loomisel (Hunt, 2006).

Arvestades nende nelja väärtusega on välja töötatud kaksteist tava, mis aitavad neid põhiväärtusi täita (Hunt, 2006):

1. Plaanimismäng – see keskendub järgmise väljalaske planeerimisele.
2. Väikese väljalasked – tarkvarasüsteemi arendatakse väikeste väljalasetena, mis lisavad iga korraga süsteemi funktsioone.
3. Lihtne disain – koodi hoitakse nii lihtsana kui võimalik.
4. Testimine – ühikteste tuleb pidevalt täiendada ning kood peab läbima testi, et arendamisega jätkata.
5. Ümberstruktureerimine – süsteemi täiustamine selle funktsionaalsust muutmata.
6. Paarisprogrammeerimine – kogu kood arendatakse programmeerijate poolt, kes töötavad paaris sama arvuti taga.
7. Kollektiivne omand – kogu kood kuulub kõigile ning igaühel on õigus seda parendada.
8. Pidev integreerimine – uus kood on integreeritud ning süsteem ehitatakse uuesti üles iga kord kui mõni ülesanne on täidetud.
9. Meeskonnatöö – klient on osa meeskonnast ning on alati valmis vastama küsimustele.
10. Ühtne kodeerimisstandart – kommentaare, meetodeid, muutujaid tähistatakse samas stiilis.
11. 40-tunnine töönädal – arendajad on alati värsked ning valmis väljakutsetele.
12. Süsteemi metafoor – süsteemi ehitust kirjeldatakse lihtsa metafooriga, millest kõik aru saavad.

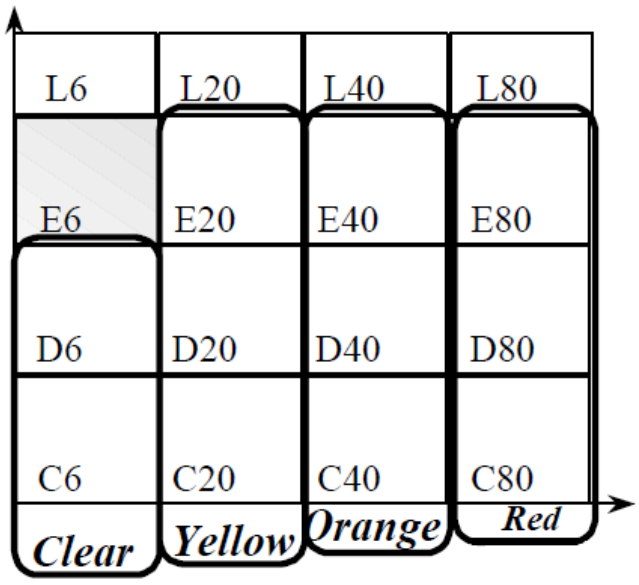
2.3 *Crystal* meetod

Crystal on agiilsete tarkvaraarendusmeetodite perekond, mille puhul arendatakse tarkvara ühiste põhimõtete järgi. Oluliseks peetaks seda, et tarkvara ei arendata korraga ühes tükis, vaid väikeste osade kaupa, neid pidevalt täiustades ning testides. Veel pannakse rõhku inimeste omavahelisele tihedale suhtlusele. Puudub üks kindel *Crystal* meetod. On erinevaid *Crystal* metoodikad erinevate projektide jaoks (Cockburn, 2004).

Nagu geoloogiliste kristallide puhul on igaühel neist erinev värv, on just selle järgi jagatud ka arendusmeetodid vastavalt selge, kollane, oranž ja punane. Igaüks neist põhineb samadel põhimõtetel, aga nende kasutatavus sõltub projekti suurusest ja raskusest (Cockburn, 2006).

Sellisena on erinevad meetodid määratud värvide järgi. Kõige agiilsem versioon on *Crystal Clear* (läbipaistev kristall), millele järgneb *Crystal Yellow* (kollane kristall), *Crystal Orange* (oranž kristall) ning *Crystal Red* (punane kristall) (Williams, 2007).

Erinevaid *Crystal* meetodeid iseloomustab joonis 4.



Joonis 4. Värvide järgi järjestatud *Crystal* meetodid (Cockburn 2006).

X-telg määrab ära meeskonna suuruse. Mida suuremaks meeskond muutub, seda raskem on juhtida protsesse näost-näkku suhtlemisega ning seetõttu on vajalik ka kooskõlastatud dokumentatsioon. Et juhtida suuremat meeskonda, on omakorda vaja sobilikku meetodit. Y-

telg iseloomustab projekti keerukust. Mida üles poole, seda keerukamaks ning tähtsamaks muutub tegevus. Skeemil on tähistatud mugavus (C), sellele järgneb vaba raha (D), hädavajalik raha (E) ning elukriitilisus (L). Kui projekt pole kriitilise tähtsusega ning selle täitmiseks pole vaja suurt meeskonda, siis on seda mõttekas arendada selge meetodiga, mis on *Crystal* metoodikatest kõige paindlikum. Kui projektid muutuvad keerukamateks ning kallimateks, siis tuleks neid realiseerida suuremate meeskondadega ning kasutada traditsioonilisemat meetodit, näiteks *Crystal Red*'i (Cockburn, 2006).

Kõik *Crystal* meetodid rõhutavad inimese tähtsust arendusmeeskondades. See keskendub inimestele, interaktsioonile, töökeskkonnale, oskustele, annetele ja suhtlusele, mis mõjutavad kõige rohkem tulemuslikkust. Arendusprotsess on endiselt oluline, kuid sekundaarne. (Williams, 2007).

Crystal metoodikad põhinevad seitsmele põhiväärtusele, milleks on (Cockburn, 2004):

1. Pidevad väljalasked – testitud kood esitletakse kliendile pärast iga programmiosa valmimist.

2. Tagasisidestatud areng – meeskond peab koosolekuid, kus tutvustatakse mis töötab, millega on probleeme ning üritatakse koos lahendusteni jõuda.

3. Osmootne suhtlemine – inimesed paigutatakse ühte tööruumi nii, et probleemide korral kuuleksid sellest kõik ning võimalusel saaksid aidata.

4. Isiklik turvalisus – võimalus rääkida enda muredest ning probleemidest ilma, et peaks kartma teiste pahameelt. Selle läbi saab meeskond oma nõrkusi parendada.

5. Fookus – selle jaoks, et meeskonnaliikmed saaksid keskendunult projektiga tegeleda, peavad nad selgeks tegema, millega klient tegeleb ja miks on kliendil arendatavat programmi vaja.

6. Kogenenud kasutajate kättesaadavus – kogenud kasutajad annavad kvaliteetset tagasisidet puuduste ja soovide kohta.

7. Tehniline keskkond - nõuab, et tagatud oleks automatiseeritud testimine, konfiguratsioonihaldus ning pidev integratsioon.

2.4 Agiilsete tarkvaraarendusmeetodite võrdlus

Eelnevalt tutvustatud meetodid on populaarseimad, mida kasutatakse, kui soovitakse arendada tarkvara agiilselt. Järgnevalt vaadeldakse nende erinevusi ja sarnasusi kokkuvõtvalt. Autor võrdleb kolme agiilset tarkvaraarendusmeetodid keskendudes nende sarnasustele ning erinevustele. Võrreldakse meetodite ülesehitust, detailsust, arendusmeeskondi ning projekti nõuete täitmise viisi.

Võrdluse aluseks kasutatakse, Agiilse Tarkvaraarenduse Manifestis kirja pandud põhimõtteid ning selgitatakse, kuidas üks või teine meetod antud punkti täidavad.

„Kõige olulisem on tagada kliendi rahulolu, tarnides talle vajalikku tarkvara võimalikult kiiresti ja tihti“. *Scrum* meetodi puhul täidab selle ülesande toote omanik, kelle ülesanne ongi pidevalt suhelda kliendiga ning pärast igat sprinti valminud tarkvaratükk talle esitleda. Pidevad väljalasked on üks *Crystal* meetodi seitsmest põhiväärtusest, kus testitud kood esitletakse kliendile pärast iga programmiosa valmimist. Ekstreemprogrammeerimise täidab selle punkti samuti väikeste väljalasetega.

„Mõistame muutuvaid olusid, isegi kui need ilmnevad arenduse lõppjärgus. Agiilsed meetodid pööravad sellised muutused meie kliendi konkurentsieeliseks“. Kõik nimetatud tarkvaraarendusmeetodid täidavad seda punkti just väikeste ning lühikeste väljalasetega. Kuna arendatav süsteem pannakse kokku väikeste tükkide kaupa, siis on lihtne lisada programmi uusi funktsionaalsusi nii, et midagi muud eriti muutma ei pea

„Tarnime tarkvara nii tihti kui võimalik, soovitavalt iga paari nädala kuni paari kuu tagant“. Ekstreemprogrammeerimise puhul on iteratsioonid väga lühikesed (päevadest kuni nädalateni) ning *Crystal Clear* ja *Scrum* kasutavad pikemaid iteratsioone (kuudes).

„Valdkonna spetsialistid ja tarkvaraarendajad peavad töötama igapäevaselt koos kogu projekti vältel“. Kõigi kolme meetodi puhul on oluline, et klient oleks vajadusel kättesaadav. *Crystal* meetodi puhul täidab selle nõude punkt seitsmest põhiväärtusest, mis nõuab kogenenud kasutajate kättesaadavust. XP seevastu määrab kliendi isegi meeskonnaliikmete kooseisu ning *Scrum'*i puhul on selle eest, et klient oleks alati saadaval, vastutav *Scrum* meister.

„Projekti edukuse aluseks on motiveeritud inimesed. Loo neile meeldiv ja toetav töökeskkond ning nad saavad iseseisvalt tööga hakkama“. *Crystal* rõhub isiklikule turvalisusele, mis tähendab vaba töökeskkonda, kus töötajad saaksid rääkida enda probleemidest ning usaldaksid üksteist. XP aitab motivatsiooni hoida 40 tunniste töönädalatega, mis tähendab, et arendajad on alati värsked ja valmis tööd tegema. *Scrum* meetod otsest tähelepanu antud punktile ei pööra.

„Kõige tõhusam ja tulemuslikum viis info jagamiseks arendusmeeskonnas on näost näkku vestlus“. Siinkohal loob selle võimaluse igapäevased meeskondade koosolekud, mida nõuavad kõik kolm arendusmeetodit. Veel aitab seda punkti täita meeskonnaliikmete vähesus, mis võimaldab kõik selle liikmed paigutada samasse ruumi nagu seda on *Crystal* arendustehnika puhul, kus on üheks seitsmest põhiväärtusest osmootne suhtlemine. See määrab ära, et arendajad peaksid töötama kindlasti samas ruumis ning tihti määratakse ära isegi töölaudade paigutus. Samuti on ka Ekstreemprogrammeerimise meetodit kasutades, sest see nõuab paarisprogrammeerist, kus kaks inimest peavad töötama ühe ja sama arvuti taga. Sellisel juhul ilma näost näkku suhtluseta töötada pole võimalik.

„Edu peamiseks mõõdupuuks on töötav tarkvara“. Projektide kallal töötades keskendutakse peamiselt tarkvara arendamisele ning muu jäätakse tagaplaanile.

„Agiilse tarkvaraarenduse protsessid soodustavad jätkusuutlikku arendust. See tähendab, et projektiga saab samas tempos jätkata määramata aja jooksul“. Kuna kõik kolm meetodit arendavad tarkvara väikeste tükkidena, siis ongi nende meeskondadel võimalik samas tempos, järk-järgult töötada, kuni projekti täieliku valmimiseni. Suure eelise selle punkti järgimisel loob Ekstreemprogrammeerimine, määrab arendajale 40-tunnise töönädala. Sellel juhul ei kurna töötajad end liialt ära, vaid puhkavad piisavalt samas tempos jätkamiseks.

„Lihtsus - ebavajaliku töö tegematajätmise kunst - on väga oluline“. Kõikide nende kolme meetodi puhul ei keskenduta eriti dokumentatsioonile ning toote nõuded pannakse kirja tükkidena, mida hakatakse järjest täitma. Rõhku ei panda ka mudelite koostamisele, sest see kulutab liigseid ressursse ning pole vajalik. Samuti tehakse ainult seda, mida klient soovib ning jäetakse ebavajalikud nõuded realiseerimata. Arendusmeeskond ise valib, milliseid tükke järjest realiseerima hakatakse. Selle juures saab ainsa erinevusena välja tuua *Scrum* meetodi, kus toote omanik määrab ära, millised omadused tuleks täita sprindi jooksul.

„Parimad arhitektuurilised lahendused, nõuded ja disain tekivad iseorganiseeruvates meeskondades“. Kõikide nende kolme meetodi puhul on meeskonnad isemajandavad ning kogu juhtimine ning tähtsate otsuste langetamine käib meeskonnasiseselt. Väljaspool meeskonda figureerivad inimesed projekti töös sõnaõigust ei oma.

Nendest kolmest agiilsest arendusmeetodist on kõige enam omavahel sarnased Ekstreemprogrammeerimine ning *Crystal* meetod. Kõige suurem erinevus eelnevate meetodite ning *Scrum*'i puhul on see, et viimane ei määra ära, kuidas programmeerida. Kui XP nõuab paarisprogrammeerimist ning *Crystal* on paika pannud nõuded töökeskkonna kohta, siis *Scrum* paneb paika arendusprotsessi, aga mitte arendustehnika.

Meetodeid saab omavahel võrrelda ka detailsuse poolest. Kui Ekstreemprogrammeerimisel on kindlalt paika pandud nõuded, milliseid praktikaid tuleb kasutada, siis *Scrum*'i ning *Crystal Clear* puhul on suhteliselt vabad käed, mis moodi tarkvara arendatakse.

Kõigi kolme meetodi puhul on suhteliselt täpselt paika pandud, kui suured meeskonnad peaksid olema. Erandina võib välja tuua *Crystal* meetodite perekonna, mis annab võimaluse arendada tarkvara ka suuremate meeskondadega, kuid selle juures muutub ka arendus sarnasemaks traditsioonilise lähenemisega ning kaugeneb agiilsetest printsiipidest.

Ühte õiget meetodit kõikidele arendajatele pole olemas. Igal meetodil on omad plussid ja miinused, mis ühtedele arendajatele sobivad ning teistele mitte. Tihti kombineeritakse erinevad arendusmeetodid ning kasutatakse hoopis neid. Näiteks on üsnagi levinud meetod, mis jälgib *Scrum* skeletti, aga arendamine toimub Ekstreemprogrammeerimise kaheteistkümne tava järgi. Selline meetodite omavahel segamine võimaldab igaühel tegeleda arendamisega neile sobival viisil.

Kokkuvõte

Käesoleva töö eesmärgiks oli tutvustada agiilset tarkvaraarendust ning võrrelda omavahel mõningaid meetodeid. Samuti oli eesmärk tuua välja erinevused traditsioonilise ning agiilse lähenemise vahel.

Kuigi kõik võrdluses olnud meetodid on liigitatud agiilsete tarkvaraarendus metoodikate alla, on igaühel neist omad eripärad ning põhimõtted, mida nad jälgivad. Kuna Agiilse Tarkvaraarenduse Manifestis on kirja pandud kindlad punktid, mis iseloomustab agiilset arendamist, siis kõik võrdluses olnud meetodid peaksid neid ka järgima. Seminaritööst tulebki välja, et kõik võrdluses olnud meetodid seda ka teevad, kuigi igaüks natukene isemoodi.

Kuigi *Scrum* meetod ei sea nii rangeid piire arendustehnika valimisel, kui seda teeb näiteks Ekstreemprogrammeerimine, siis enda paika pandud reeglite ning ka teatud vabadusega on just see muutunud enimkasutatavaks agiilseks tarkvaraarenduse meetodiks. *Crystal* meetodi võib aga enda karmuse ning reeglite rohkuse järgi paigutada nende kahe eelnimetatud meetodi vahele, mille suureks plussiks on võimalus arendada tarkvara ka suuremamahuliste projektide kallal, ise samal ajal enda töös järgida ka mõningaid agiilse tarkvaraarenduse põhimõtteid.

Tööd võib edasi arendada sellisel suunal, et kaasata võrdlusesse rohkem erinevaid agillseid tarkvaraarendusmeetodeid ning neid samamoodi võrrelda. Veel oleks huvitav olla kaasatud erinevaid arendusmeetodeid viljelevate meeskondade koosseisu ja võrrelda reaalselt nende projektide arendust kohapeal ning ise kogeda, milline meetoditest on kõige tulemusrikkam ja sobilikum.

Kasutatud kirjandus

Azizyan, G., Magarian, M. K., Kajko-Mattson, M. (2011). Survey of Agile Tool Usage and Needs külastatud Külastatud 18. oktoobril, 2014, aadressil http://www.agilealliance.org/files/7313/2435/0836/Survey%20of%20Agile%20Tool%20Usage%20and%20Needs.pdf.

Beck, K. (2000). Extreme Programming Explained: Embrace Change. Addisson-Wesley.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Flower, M., Grenning, J., Highsmith, J., Hund, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001a). Agiilse tarkvaraarenduse manifest. Külastatud 28. septembril, 2014, aadressil http://agilemanifesto.org/iso/et/.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Flower, M., Grenning, J., Highsmith, J., Hund, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001b). Agiilse tarkvaraarenduse manifesti põhimõtted. Külastatud 28. septembril, 2014, aadressil http://agilemanifesto.org/iso/et/principles.html.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Flower, M., Grenning, J., Highsmith, J., Hund, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001c). History: The Agile Manifesto. Külastatud 28. septembril, 2014, aadressil http://agilemanifesto.org/history.html.

Cockburn, A. (2004). Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams. Boston: Addison-Wesley Professional.

Cockburn, A. (2006). Agile Software Development: The Cooperative Game. Boston: Addison-Wesley Professional.

Ghilic-Micu, B., Mircea, M., Stoica, M.(2013). Software Development: Agile vs. Traditional. *Informatica Economică*, 17 (4), 65-77.

Hunt, J. (2006). Wiltshire: Agile Software Construction. Wiltshire: Experis Ltd.

Johnson, L. H., Sims, C. (2012). Scrum: a Breathtakingly Brief and Agile Introduction. Külastatud 3. oktoobril, 2014, aadressil http://www.agilelearninglabs.com/resources/scrum-introduction.

Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Addison-Wesley. ISBN 9780131111554.

Mahapatra, R., Mangalaraj, G., Nerur, S. (2005) Challenges of migrating to agile methodologies. *Communications of the ACM*, 48 (5), 72– 78.

Ocamb, S. (2013). What Does the Agile Manifesto Mean?. Külastatud 28. septembril, 2014, aadressil https://www.scrumalliance.org/community/articles/2013/2013-april/what-does-the-agile-manifesto-mean.

Rossberg, J. (2008). Pro Visual Studio Team System Application Lifecycle Management. NewYork: Springer-Verlag New York.

Sanjiv, A. (2008). Managing agile projects. Upper Saddle River: Prentice Hall Professional Technical Reference..

Sommerville, I. (2011). Software engineering. Boston: Pearson.

Software Tools. (2013). Scrum. Külastatud 3. oktoobril, 2014, aadressil http://www.agilelearninglabs.com/resources/scrum-introduction.

Williams, L. (2007). A Survey of Agile Development Methodologies. Külastatud 11. oktoobril, 2014, aadressil http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf.