TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut

IDK40LT

Ann-Claire Utt 134207IAPB

TARKVARAARENDUSE KULU HINDAMISE METOODIKAD EESTIS: TEOORIA JA TEGELIKKUS

Bakalaureusetöö

Juhendaja: Kadri-Liis Kusmin

Ärijuhtimise magistri kraad
Tarkvaraarendaja

Tallinn 2016

Autorideklaratsioon

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

Autor: Ann-Claire Utt

23.05.2016

Annotatsioon

Antud lõputöö eesmärk on võrrelda tarkvaraarenduse kulu hindamise metoodikaid ning anda soovitusi, et aidata kaasa hinnangute täpsustumisele Eestis. Praeguse olukorra hindamiseks viis autor läbi küsitluse tarkvaraarenduse kulu hindamisega tegelevate inimeste seas, mille abil koguti informatsiooni Eestis tarkvaraarenduse kulude hindamise metoodikate praktiseerimise kohta.

Töö esimeses osas antakse ülevaade erinevatest tarkvaraarenduse kulu hindamise metoodikatest ning koostatakse neid võrdlev tabel. Töö teises osas autor analüüsib küsitlusest saadud vastuseid. Küsimustiku vastustest sai järeldada, et erinevate meetodite teadmine ja kasutamine aitab kaasa täpsemalt tarkvaraarenduse kulu hindamisele. Töö lõppeb soovitustega, mida järgides, on võimalik tarkvaraarenduse kulu hindamist täpsustada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 2 peatükki, 14 joonist, 7 tabelit.

Abstract

Software Development Cost Estimation Methodologies in Estonia Theory and Reality

One of the most complex aspects of software development is to determine how long and how much it will take to deliver a software product. Often these estimations are done without any proper use of cost estimation methods.

The objective of this thesis is to compare different cost estimation methods and to give suggestions to become better in software cost estimation. The author completed this objective with these four steps:

- Investigate different cost estimation methods
- Present and compare chosen methods briefly
- Compose a questionnaire and reach to a big number of people
- Analyze results from questionnaire.

In the first part of the thesis a brief summary of five different cost estimation methods is given: planning poker, use case points, COCOMO and COCOMO II, evidence based scheduling and MoSCoW analysis. The second half concentrates on current software cost estimation culture in Estonia. To find it out a questionnaire was created and 50 people participated in it. From the results it can be said that good knowledge about and using cost estimation methods helps to give more precise cost estimations. The second part ends with recommendations to improve software cost estimations.

The thesis fulfills its objective to compare different cost estimation methods and to give recommendations to help improve software cost estimation in Estonia

The further development of this thesis could be to give a precise overview of software cost estimation methods and their suitability for different types of software development companies.

The thesis is in Estonian and contains 36 pages of text, 2 chapters, 14 figures, 7 tables.

Lühendite ja mõistete sõnastik

API	Application Programming Interface, rakendusliides, reeglistikud, protokollid ja tööriistad olevasoleva tarkvaraga suhtlemiseks
COCOMO	Constructive Cost Model, tarkvara arendamise kulu hindamise mudel
EAF	Effort adjustment factor, pingutuse muutuse faktor
ECF	Environmental Complexity Factor, aitab määrata projekti suurust võttes arvesse ümbritseva keskkonna
Ekstreemprogrammeerimine	Agiilne tarkvara arendamise metoodika
FTP	File Transfer Protocol, failide edastus protokoll
HTTP	Hypertext Transfer Protocol, Hüperteksti edastusprotokoll, protokoll andmete edastamiseks arvutivõrkudes
MoSCoW	Prioritiseerimise meetod, lühend sõnadest *Must have, Should have, Could have, and Would like but won't get*
NASA	National Aeronautics and Space Administration, USA riiklik Aeronautika- ja Kosmosevalitsus
RUP	Rational Unified Process, objektorienteeritud ja veebisõbralik arendussüsteem
Scrum	Iteratiivne ja agiilne tarkvara arendamise raamistik
SEER-SEM	Software Evaluation and Estimation of Resources - Software Estimating Model, projektide haldamise ja monitoorimise tarkvara
TCP/IP	Transmission Control Protocol/Internet Protocol, edastusohje protokollistik internetiprotokolli peal, internetiprotokollistik

TCF	Technical Complexity Factor, aitab määrata projekti suurust võttes arvesse tehnilisi raskusi
TF	Technical factor, tarkvaraarenduse tehiline faktor
UAW	Unadjusted Actor Weight, mõõdab tegijate arvu ja raskusastet
UCP	Use Case points, kasutusjuhu punktid
UML	Unified Modeling Language, unifitseeritud modelleerimiskeel
UUCW	Unadjusted Use Case Weight, kasutusjuhtude arv ja raskusaste

Sisukord

Sissejuhatus 10
1 Tarkvaraarenduse kulu hindamise metoodikad 11
1.1 Planeerimise pokker 11
1.2 Kasutusjuhu punktid ehk UCP 12
1.3 COCOMO ja COCOMO II 15
1.4 Tõestusel põhinev planeerimine ehk *Evidence-based Scheduling* 19
1.5 MoSCoW analüüs 21
1.6 Tutvustatud meetodite võrdlustabel 22
2 Tarkvaraarenduse kulu hindamise metoodikate tegelikkus Eestis 24
2.1 Küsimustiku küsimused, vastused ja statistika 24
2.2 Küsimustiku analüüs 32
2.2.1 Planeerimise pokkeri statistika 32
2.2.2 MoSCoW analüüsi statistika 33
2.2.3 Kokkuvõttev analüüs ja soovitused 34
Kokkuvõte 35
Kasutatud kirjandus 36

Jooniste loetelu

Joonis 1. Tarkvara arendamisega seonduvate protsessidega tegelemise kogemus. 25
Joonis 2. Agiilse ja kose meetodi suhe vastajate seas. 25
Joonis 3. Täpsem arendamismeetod. 26
Joonis 4. Ajahindamise määramisega tegelemise kestvus. 26
Joonis 5. *Planning poker*'i ehk planeerimise pokkeri teadmine. 27
Joonis 6. MoSCoW analüüsi meetodi teadmine. 28
Joonis 7. Ajahinnangu määramise meetodid. 28
Joonis 8. Meetodi COCOMO II teadmine. 29
Joonis 9. Ajahinnangu andmine kliendile võrreldes meetodiga. 29
Joonis 10. Kliendile antud ajahinnangu vastavus tegelikkusele. 30
Joonis 11. Kliendile antava ajahinnangu vastavus tegelikkusele: varem üle andmine. . 31
Joonis 12. Kliendile antava ajahinnangu vastavus tegelikkusele: õigeaegselt üle andmine. 31
Joonis 13. Kliendile antava ajahinnangu vastavus tegelikkusele: hiljaks jäämine. 32
Joonis 14. MoSCoW analüüsi ja projekti enne tähtaega üleandmise suhe. 33

Tabelite loetelu

Tabel 1. Tehnilise keerukuse faktorid. 14
Tabel 2. Keskkonna keerukuse faktorid. 15
Tabel 3. Tarkvaraarenduse projektide koefitsiendid. 16
Tabel 4. COCOMO toote ja riistvara atribuudid. 17
Tabel 5. COCOMO personali ja projekti atribuudid. 18
Tabel 6. COCOMO tarkvara projekti koefitsiendid. 19
Tabel 7. Tutvustatud meetodite võrdlustabel. 22

Sissejuhatus

Tihti arvavad infotehnoloogiast kauged inimesed, et tarkvaraarendus koosneb üksnes koodi kirjutamisest. Siiski ei ole see nii: tarkvaraarenduse üks osa on hinnata palju aega ja raha kulub sellise toote arendamiseks, mida klient tegelikult soovib. Võib ju arvata, et klient tuleb oma nõuetega ja kui nende järgi tarkvara valmis teha, siis on ta rahul. Kohe alguses antakse täpne hinnang, kui kiiresti kliendile tema soovitud funktsionaalsus valmis saab ja hind arvutatakse kulunud aeg korrutades tunnihinnaga. Kahjuks ei ole ka see nii lihtne – selleks, et anda võimalikult täpset ajahinnangut kliendile, tuleb kõik nõuded korralikult läbi mõelda. Isegi juhul, kui kõik on läbi mõeldud, ei pruugi ajahinnang lõpuks sobiv olla, sest tihtipeale tekib uusi nõudeid töö käigus juurde.

Tarkvaraarendamise kulu hindamise teema valiti selle tõttu, et aja- ning kuluhindamine on alati aktuaalne – teema, millest ei ole kunagi liiga palju räägitud. On kuulda, kuidas tarkvaraarenduse projektid lähevad üle tähtaja ning kallimaks kui alguses arvati. Seda olukorda saab parandada võttes kasutusele hindamise metoodikad. Antud teema kohta on väga vähe eestikeelset materjali ning töö annab lisaväärtust suurendamaks eestikeelsete materjalide hulka. Teema sobib autorile seetõttu, et loodab tulevikus ennast siduda aja- ning kuluhindamisega.

Käesoleva lõputöö eesmärk on võrrelda tarkvaraarenduse kulu hindamise metoodikaid ning anda soovitusi, et aidata kaasa hinnangute täpsustumisele Eestis. Autor täitis eesmärki nelja sammuga:

- Uurima erinevaid aja- ja kuluhindamise metoodikaid
- Tutvustama ja võrdlema valitud metoodikaid lühidalt
- Koostama küsimustiku ning saavutama suure esindusliku valimi
- Analüüsima küsimustikust saadud tulemusi.

Töö struktuur on üles ehitatud järgmiselt: kõigepealt antakse lühidalt ülevaade mitmetest kuluhindamise metoodikatest. Peale meetodite kirjeldusi on lühidalt tutvustatud kõiki läbi viidud küsimustiku küsimusi ning nende tagamõtteid. Seejärel tulevad küsimustiku tulemused ning nende analüüs.

1 Tarkvaraarenduse kulu hindamise metoodikad

Erinevaid tarkvaraarenduse kulu hindamise meetodeid on väga mitmeid. Selles peatükis tutvustatakse lähemalt järgnevaid meetodeid:

- Planeerimise pokkerit, sest seda kasutakse tihti populaarsust koguvate agiilsete meetodite puhul,
- Kasutusjuhu punktid, sest see arvestab laialt levinud UML ja RUP'i kasutajatega,
- COCOMO ja COCOMO II, sest need on maailmas tuntud ning pika ajalooga,
- Tõestusel põhinev planeerimine, sest sellel on teistest veidi erinev vaatenurk,
- MoSCoW analüüs, sest see on väga lihtne meetod mida järgida, et takistada üleliigset arendamist.

Esmalt tooks välja ka paar meetodit, mille uurimine lõputöö mahu tõttu kõrvale jäi. Funktsiooni punktide meetod, mille puhul ühe osa kulu, kas tundides või rahas, arvutatakse varasemate projektide pealt ning mille plussiks on fakt, et funktsiooni punkte saab arvutada nõuete pealt – ehk tuleb abiks varajasel hindamisel. Mainimist väärt on ka PRICE süsteemi, mis oli esimene üldsusele avalik kulu hindamise tarkvara ning seda kasutatakse näiteks NASA's. SEER-SEM, millest on välja arenenud mitmeid erinevatele aladele mõeldud programme. Kõik siin lõigus nimetatud meetodid ja mitmed veel jäid välja seetõttu, et käesolev töö oleks vastasel juhul liialt pikaks veninud.

1.1 Planeerimise pokker

Planeerimise pokker, teise nimega Scrum pokker, on mänguline konsensusel põhinev ajakulu hindamise meetod, mis on kõige rohkem kasutuses agiilsel arendamisel ning eriti Scrum'i ja Ekstreem Programmeerimise puhul [1]. Planeerimise pokkerit kasutades, panevad kõik grupist oma arvamusel ülesandele kuluvat ressurssi näitava numbrilise väärtusega kaardi lauale, nii, et numbrit esialgu teistele näha ei ole. Kui kõik on oma hinnangu kaardi näol lauale pannud, keeratakse kaardid ümber ning seejärel arutatakse läbi, miks keegi just sellise numbri antud ülesande raskuseks määras. Meetodi leiutas

James Grenning 2002.aastal ning hiljem sai tuntuks Mike Cohn'i raamatus „Agile Estimating and Planning".

Scrum pokkeri põhimõte on julgustada inimesi mõtlema ning seetõttu annavad inimesed just selle vastuse, mida nad ise arvavad, mitte ei jää lootma kellegi teise arvamuse kopeerimise peale. Selle meetodi kasutamiseks läheb vaja nii mitut pakki planeerimise pokkeri kaarte, kui on inimesi. Nimetatud kaartidel on peal Fibonacci jada numbrid, kuid lisaks sellele on veel küsimärgiga kaart, mis tähendab, et ei oska hinnata, ning kaart kohviga, mis tähendab, et inimene vajab pausi.

Hinnangute andmise koosolekul on üks inimene moderaator, kes juhib koosolekut ja ise ei mängi. Tootejuht annab lühikese ülevaate ülesandest ning kõik võivad küsida küsimusi, et saada paremat ülevaadet. Peale seda panevad kõik oma valitud kaardi lauale ning seejärel keeravad kõik korraga oma kaardi ümber ja samal ajal öeldakse, mis kaardi nad valisid. Neile kes valisid kõrged või madalad kaardid, saavad võimaluse oma valikut õigustada ja seejärel diskussioon jätkub. Hindamise osa korratakse kuni jõutakse konsensuseni.

Planeerimise pokkeri üks tugevamaid külgi on see, et selle hinnangu täpsus võib olla väga hea. Samuti on seda meetodit väga kerge õppida ning tihti saavutatakse selle abil ka parem ettekujutus töösse tulevast funktsioonist.

1.2 Kasutusjuhu punktid ehk UCP

Kasutusjuhu punktid on meetod tarkvara projekti suuruse hindamiseks [2]. Seda kasutatakse siis, kui projektis on kasutuses UML ja RUPi metoodikad. UCP kontseptsioon põhineb sellel, et süsteem kirjutatakse üles kasutades kasutusjuhte.

Gustav Karner lõi UCP meetodi 1993.aastal selleks, et lahendada tarkvara suuruse hindamine objektorienteeritud süsteemidele.

Meetodi kasutamiseks tuleb kõigepealt arvutada välja neli elementi:

- korrigeerimata kasutusjuhu osakaal (UUCW) – mõõdab kasutusjuhtude arvu ja raskusastet
- korrigeerimata tegija osakaal (UAW) – mõõdab tegijate arvu ja raskusastet

- tehnilise keerukuse faktor (TCF) – aitab määrata projekti suurust võttes arvesse tehnilisi raskusi

- keskkonna keerukuse faktor (ECF) – aitab määrata projekti suurust võttes arvesse ümbritseva keskkonna.

Kui kõik eelnevad osad on leitud, saab arvutada lõpliku arvatava projekti suuruse *Use Case Points* ehk UCP kasutades valemit (1).

$$UCP = (UUCW + UAW) * TCF * ECF \quad (1)$$

Korrigeerimata kasutusjuhu osakaal ehk *Unadjusted Use Case Weight* ehk UUCW hindab kasutusjuhtude raskust liigitades need kolme gruppi transaktsioonide arvu alusel: 1-3 transaktsiooni on lihtsad, 4-7 transaktsiooni on keskmised ja 8 või rohkem transaktsioone on rasked. UUCW saab arvutada kui korrutada kasutusjuhtude ühes grupis oleva arvu sellele grupile vastava kaaluga ning seejärel saadud arvud kokku liites. Lihtsate, keskmiste ja raskete kaalud on vastavalt 5, 10 ja 15, seega valemiks saame valemi (2).

$$UUCW = (kergete\ kasutusjuhtude\ arv * 5) + (keskmiste\ kasutusjuhtude\ arv * 10) + (raskete\ kasutusjuhtude\ arv * 15) \quad (2)$$

Korrigeerimata tegija osakaal ehk *Unadjusted Actor Weight* ehk UAW jagab sarnaselt UUCWga tegijad lihtsateks, keskmisteks ning rasketeks, kaaludega vastavalt 1,2 ja 3. Lihtne on tegija, kes on väline süsteem, mis peab suhtlema süsteemiga kasutades hästi defineeritud API't. Keskmised on välised süsteemid, mis peavad suhtlema läbi standartsete protokollide nagu TCP/IP, FTP, HTTP või andmebaas. UAW arvutamiseks peame sarnaselt UUCW'le korrutama tegijate grupi arvu nendele vastava kaaluga ning seejärel saadud arvud omavahel liitma. Seega saame valemiks (3).

$$UAW = (lihtsate\ tegijate\ arv * 1) + (keskmiste\ tegijate\ arv * 2) + (raskete\ tegijate\ arv * 3) \quad (3)$$

Tehnilise keerukuse faktor ehk *Technical Complexity Factor* ehk TCF hindab 13 erineva faktori relevantsust skaalal nullist, mis tähendab, et faktor on irrelevantne, viieni, mis näitab, et see on hädavajalik. Relevantsuse arv korrutatakse etteantud kaaluga (Tabel 1) kõikide 13 faktori puhul ja saadud arvud liidetakse kokku - seda nimetatakse tehniliseks faktoriks ehk TF'iks. TF'i kasutatakse, et arvutada TCF järgneva valemiga (4).

$$TCF = 0{,}6 + (TF/100) \quad (4)$$

Tabel 1. Tehnilise keerukuse faktorid.

Faktor	**Kirjeldus**	**Kaal**
T1	Süsteemi hajusus	2,0
T2	Reaktsiooniaeg	1,0
T3	Lõpp-kasutaja efektiivsus	1,0
T4	Sisemine töötlemise keerukus	1,0
T5	Koodi taaskasutatavus	1,0
T6	Installeerimine on kerge	0,5
T7	Kasutamine on kerge	0,5
T8	Teisele platvormile viimise võimalus	2,0
T9	Süsteemi hooldamine	1,0
T10	Paralleeltöötlus	1,0
T11	Turvalisuse omadused	1,0
T12	Kolmandate osapoolte juurdepääsevus	1,0
T13	Lõpp-kasutaja koolitamine	1,0

Keskkonna keerukuse faktor ehk *Environmental Complexity Factor* ehk ECF hindab 8 erineva faktori (Tabel 2) relevantsust skaalal nullist viieni, kus null tähendab, et ei ole üldse kogemust ja viis, et ollakse selles ekspert. Iga number korrutatakse talle vastava faktori kaaluga. Kõikide saadud arvude summa on keskkonna faktor ehk EF (*environment factor*). Viimast kasutatakse ECF'i arvutamiseks järgmiselt valemiga (5).

$$ECF = 1{,}4 + (-0{,}03 * EF) \quad (5)$$

Tabel 2. Keskkonna keerukuse faktorid.

Faktor	**Kirjeldus**	**Kaal**
E1	Tuttav kasutatava arendusprotsessiga	1,5
E2	Rakenduse tegemise kogemus	0,5
E3	Meeskonna kogemus objektorienteerituse vallas	1,0
E4	Peamise analüütiku oskused	0,5
E5	Meeskonna motiveerimine	1,0
E6	Nõuete stabiilsus	2,0
E7	Osalise tööajaga töötajad	-1,0
E8	Keeruline programmeerimiskeel	-1,0

Kasutusjuhu punktide plussiks on see, et meetodit on lihtne õppida ning seda saab juba väga varases staadiumis kasutada funktsionaalse suuruse hindamiseks. Meetodi miinuseks on aga fakt, et seda ei ole kalibreeritud kasutades regressioonanalüüsi.

1.3 COCOMO ja COCOMO II

COCOMO ehk *Constructive Cost Model* on tarkvara arendamise kulu hindamise mudel, mis hindab programmi suurust, mis on antud tuhandetes koodiridades [3]. COCOMO saab kasutada kolmel juhul, kui on tegemist:

- orgaanilise projektiga – väike meeskond, kellel on hea kogemus, töötab projekti kallal, mille nõuded ei ole ranged
- keskmise projektiga – keskmise suurusega meeskond, kelle töökogemus on erineva suurusega ning projekti nõudeid on nii rangeid kui ka veidi painduvamad
- segu projektiga – arendatud rangete piirangutega või segu esimesest kahest.

Põhilise COCOMO valemid on järgnevad:

- Aja kulu inim-kuudes: $E = a_b * (KLOC)^{b_b}$
- Arendamise aeg kuudes : $D = c_b * (E)^{d_b}$
- Inimeste hulk: $P = \frac{E}{D}$

KLOC on hinnanguline koodiridade arv tuhandetes ning a_b, b_b, c_b ja d_b vastavalt projekti tüübile (Tabel 3).

Tabel 3. Tarkvaraarenduse projektide koefitsiendid.

Tarkvaraarenduse projekt	**a_b**	**b_b**	**c_b**	**d_b**
Orgaaniline	2,4	1,05	2,5	0,38
Keskmine	3,0	1,12	2,5	0,35
Segu	3,6	1,20	2,5	0,32

Põhiline COCOMO on hea kiireks ja ligikaudseks tarkvaraarenduse hinna arvutamiseks. Mudeli puuduseks on asjaolu, et see ei arvesta, milliseid tehnikaid kasutatakse ning kui palju kasutatakse uudseid vahendeid.

Keskmine COCOMO võtab arvesse nii programmi suuruse kui ka osaliselt teisi kuluallikaid nagu riistvara ja personal. Iga 15 atribuuti (Tabel 4, Tabel 5) hinnatakse skaalal „väga madal“ kuni „kõrgeim“. Skaala hinnangu järgi saadud koefitsiendid korrutatakse ning saadakse kokku EAF. Keskmise COCOMO valem (6) kasutades varem arvutatud EAF'i ning a_i ja b_i on sobivad koefitsiendid (Tabel 6).

$$E = a_i * (KLOC)^{b_i} * (EAF) \quad (6)$$

Arendamise aja kulu D saame kasutades sama valemit mis põhilisel COCOMO'l oli, kui kasutame just arvutatud E väärtust.

Tabel 4. COCOMO toote ja riistvara atribuudid.

Kuluallikad	**Väga madal**	**Madal**	**Normaalne**	**Kõrge**	**Väga kõrge**	**Kõrgeim**
Toote atribuudid						
Nõutud tarkvara töökindlus	0,75	0,88	1,00	1,15	1,40	
Rakenduse andmebaasi suurus		0,94	1,00	1,08	1,16	
Toote keerulisus	0,70	0,85	1,00	1,15	1,30	1,65
Riistvara atribuudid						
Käitusaegsed toimimispiirangud			1,00	1,11	1,30	1,66
Mälu piirangud			1,00	1,06	1,21	1,56
Virtuaalse masina keskkonna ebapüsivus		0,87	1,00	1,15	1,30	
Vajalik ümberlülituse aeg		0,87	1,00	1,07	1,15	

Tabel 5. COCOMO personali ja projekti atribuudid.

Kuluallikad	**Väga madal**	**Madal**	**Normaalne**	**Kõrge**	**Väga kõrge**
Personali atribuudid					
Analüütiku võimekus	1,46	1,19	1,00	0,86	0,71
Rakenduse tegemise kogemus	4,29	1,13	1,00	0,91	0,82
Arendaja võimekus	1,42	1,17	1,00	0,86	0,70
Virtuaalse masinaga töötamise kogemus	1,21	1,10	1,00	0,90	
Programmeerimiskeele kogemus	1,14	1,07	1,00	0,95	
Projekti atribuudid					
Tarkvaraarenduse meetodite kasutamine	1,24	1,10	1,00	0,91	0,82
Tarkvaravahendite kasutus	1,24	1,10	1,00	0,91	0,83
Nõutud arendamise ajakava	1,23	1,08	1,00	1,04	1,10

Tabel 6. COCOMO tarkvara projekti koefitsiendid.

Tarkvara projekt	$\mathbf{A_i}$	$\mathbf{B_i}$
Orgaaniline	3,2	1,05
Keskmine	3,0	1,12
Segu	2,8	1,20

Detailne COCOMO erineb eelnevatest selle poolest, et sellega arvutatakse iga astme ajakulu eraldi ja seejärel need liites saadakse kogu kestvus. Detailses COCOMO meetodis on kuus faasi: planeerimine ja nõuded, süsteemi disain, detailne disain, mooduli kood ja testimine, integratsioon ja testimine, kulude konstruktiivne mudel.

COCOMO II on hindamise mudel, mis võtab arvesse projekti, toodet, riistvara ja personali [4]. COCOMO II arvestab võimalusega uuesti kasutada juba valmis kirjutatud koodi või arendada tarkvara juba valmis tarkvara komponentidega ning suuremat ajahulka nõudvat disaini.

Kokkuvõttes on COCOMO ja COCOMO II väheselt kasutatavad, sest ainult suurfirmad saavad lubada endale eksperti, kes tegeleks pidevalt nende meetodite kasutamiseks informatsiooni kogumisega ning aitaks selle kasutamisega kohaneda.

1.4 Tõestusel põhinev planeerimine ehk *Evidence-based Scheduling*

Tõestusel põhineva planeerimise lõi Joel Spolsky ning sellel on kaks põhilist mõtet: alati tuleb hinnata kogu kulutatud aega ja Monte Carlo meetodi, mis on mitmete juhuslike võimalike arvudega arvutuse tegemine, kasutamine kõige tõenäolisema tähtaja leidmiseks [5]. Selleks, et see meetod saaks toimida, peab järgima nelja printsiipi:

- ülesannete suurused on väiksemad kui 16 tundi
- aja kulu jälgimine ja kirja panek
- simuleerida tulevikku kasutades Monte Carlo meetodit

- halda projekti pidevalt.

Esimene printsiip: tuleb hoida ülesannete suurused väiksemad kui 16 tundi. Tänu sellele, et ülesanded on väikese mahuga, on võimalik läbi mõelda, mida täpselt peab tegema, et ülesannet lõpetada. Suure tõenäosusega ollakse väiksemaid osasid sellest ülesandest varem tehtud, seega nendele kuluvat aega osatakse paremini hinnata. Kui liita kokku kõik väga kergete ülesannete aja kulu saadakse teada ka selle alla 16 tunnise ülesande summaarne ajakulu.

Teine printsiip: jälgida aja kulu. Selle mõte on koguda informatsiooni. Enne ülesandega tegelema asumist tuleb kirja panna, mis on selle ülesande hinnanguline ajakulu ja selle sama ülesande lõppedes tuleb üles märkida, kaua selle ülesande jaoks tegelikult aega kulus. Kõik tehtud ülesanded saab panna graafikule, kus ühel teljel on ajahinnang ja teisel on tegelik aeg. Kui jagada ajahinnang tegeliku ajaga, siis saadakse kiirus – ülesande täitmise kiirus võrreldes ajahinnanguga. Aja jooksul koguneb kõikidele arendajatele selliseid kiiruse punkte palju. Tänu kogemusele hakatakse järjest täpsemini hinnanguid andma, seega väga vanad hinnangud on mõistlik unustada.

Kolmas printsiip: simuleerida tulevikku. Kasutades Monte Carlo meetodit tuleb luua 100 võimalikku stsenaariumi, mille igaühe juhtumise tõenäosuseks on 1%. Arvutades arendajale ajakulu tuleb iga ülesande ajahinnang jagada ühe tema eelmiste kiiruste seast suvaliselt võetud kiirusega. Iga Monte Carlo meetodiga saadud tulemus tuleb ümber arvutada tundidest graafikuks ja selle tegemisel peab arvestama arendaja puhkuste ja töögraafikuga. Pärast kõigile arendajatele ajahinnangu loomist tuleb vaadata, kes lõpetab neist kõige hiljem, sest see on aeg, mil kogu meeskond saab valmis. Samuti on väga oluline olla oma aja arvestamisel järjepidev. Selle all mõeldakse seda, et on kaks varianti, kas kirjutada koosolekud ja muud otseselt tööga mitte seotud faktorid plaani ning neid eraldi märkida ka aja arvestamise lehtedele või arvestada see aeg tollel hetkel pooleli oleva ülesande valmimise aja sisse. Võiks arvata, et kui arvestada mitte tööülesannete täitmine ülesande tegemise aja sisse, siis see meetod annab vale hinnangu, kuid olukord on vastupidine: aja jooksul on tööd takistavate koosolekute tihedus ilmselt enam-vähem samas mahus, seega saab endiselt seda meetodit kasutades hea ajalise hinnangu.

Neljas printsiip: projekti pidev haldamine. Kui jagada ülesanded tähtsuse järgi ära, siis saab hinnata, kui kasulik oleks teatud ajani arendada. Samuti võib vaadata kõikide

arendajate võimalikke ülesannetega lõpetamise kuupäevi. Need arendajad, kelle kuupäeva vahemik on väga suur peavad õppima paremini hindama. Need arendajad, kelle tähtaja vahemik ei ole küll väga suur, aga lähevad suures osas üle soovitud kuupäeva, kannatavad liiga suure töökoormuse all ning neilt tuleb ülesandeid vähemaks võtta. Selleks, et vältida ajalist üleminekut projekti jooksul tekkinud lisa nõuete tõttu, soovitatakse kirja panna varu aega uutele funktsioonide ideedele, integratsioonidele, kasutajamugavuse testimisele ja parandustele. Sel juhul saab vajadusel juba vastava eesmärgi jaoks planeeritud aega kasutada.

Meetodi arendaja peab tähtsaks veel mõningaid aspekte. Esiteks ainult programmeerija ise saab anda hinnangut. Teiseks vead tuleks parandada kohe pärast leidmist ning sellele kuluv aeg lisada algsele ülesande lahendamise ajale. Kolmandaks projektijuhid ei tohi mõelda, et nad suudavad motiveerida arendajaid kiiremini töötama, kui see tegelikult võimalik on – see ei anna soovitud tulemust, sest arendaja, kes teab, et tal on liiga palju tööd tunneb end motivatsioonitult. Neljandaks kui aega projekti lõpuni on üks ühik aga tööd, mida teha on kaks ühikut, siin on kaks valikut: kas lükata valmis saamise aega edasi või kustutada mõned ülesannetest – nii, et jõuaks õigeks ajaks valmis.

Kokkuvõtteks on selle meetodi positiivseks osaks see, et see on väga kerge – iga iteratsiooni algul kulub umber kaks päeva, et kirja panna kõik hinnangud ning igapäevaselt mõned minutid, et hoida oma ajaarvestust korras. Negatiivseks osaks on see, et kui arendaja ise on oma ajakulu hindamise poolest väga ebapädev, siis see meetod ei anna ka projektijuhile väga häid tulemusi.

1.5 MoSCoW analüüs

MoSCoW on ühend sõnadest *must* ehk peab, *should* ehk peaks, *could* ehk võiks ja *won't* ehk ei tee[6]. See tehnika aitab mõista, mida tegelikult on vaja, et väärtust luua ja mis on kena, kui oleks. „Peab olema“ ehk *must* - hinnanguga peavad olema featuurid, mis julgustavad kasutajat seda tarkvara kasutama. „Peaks“ ehk *should* - hinnanguga peavad olema featuurid, mis on positiivsed üllatused, mis teevad tarkvara kasutamist meeldivamaks, kuid neid saab lisada ka hiljem. „Võiks“ ehk *could* - hinnanguga peavad olema featuurid, mis ei anna märkimisväärset ärilist väärtust juurde. „Ei tee“ ehk *won't* - hinnanguga featuurid võivad olla näiteks kunagi tulevikus olulised, kuid hetkel on iteratsiooni mahust väljas.

1.6 Tutvustatud meetodite võrdlustabel

Tabel 7. Tutvustatud meetodite võrdlustabel.

Hindamise metoodika	Plussid	Miinused
Planeerimise pokker	Odav kasutusele võtta, lihtne, ei võta kaua aega, saadakse rohkem teada tulevaste ülesannete kohta, täpsus võib olla sama hea kui eksperdi oma	Mõeldud ainult agiilsele arendusele, meeskond peab olema samal ajal kättesaadav, et seda koos kasutada
Kasutusjuhu punktid	Arvestab UML'i ja RUP'i kasutust, ei ole väga keeruline, kasutatavad valemid ei ole keerulised, ei ole raske õppida, saab kasutada projekti varajases staadiumis funktsionaalse suuruse hindamiseks	Ei ole väga kerge, peab hindama mitmeid faktoreid, ei ole kalibreeritud regressioonanalüüsi kasutades, ei arvesta ökonoomsust
COCOMO ja COCOMO II	On mitut varianti, lihtne COCOMO on hea kiireks ja ligikaudseks hinnanguks, keskmine COCOMO võtab arvesse ka teisi kulu allikaid nagu riistvara ja personal, COCOMO II arvestab projekti, toodet, riistvara ja personali	Palju keerulisi valemeid, lihtne COCOMO ei arvesta, milliseid tehnikaid kasutatakse ja uudseid vahendeid ka mitte, keskmine ja detailne COCOMO vajavad palju ressurssi, et neid kasutada, kallis kasutusele võtta
Tõestusel põhinev planeerimine	Arendajale lihtne, projektijuhil pidev võimalus jälgida, võtab vähe aega, lihtsad põhimõtted, iga iteratsiooni juures kulub ligi 2 päeva	Läheb aega, et arendajad oskaksid oma töö ajakulu hästi hinnata, peast oleks võimatu seda metoodika abil ajalist hinnangut anda

Hindamise metoodika	Plussid	Miinused
MoSCoW analüüs	Lihtne, aitab planeerida iteratsiooni mahtu paremini	Tihtipeale on raske osa õigeid asju tõsta ei tee lahtrisse, otsest ajahinnangut ei anna

2 Tarkvaraarenduse kulu hindamise metoodikate tegelikkus Eestis

Uurimus viidi läbi küsimustiku abil, sest uurimistöö eesmärgiks oli välja selgitada erinevate Eesti tarkvaraarendus ettevõtete aja- ja hinnaarvestamise metoodikad. Küsimustik koostati 2016 aprillis ning täideti mai 2016 alguses. Küsitlus koostati kasutades küsitlustarkvara „Google Forms“ ning levitati Internetis. Küsimustiku sihtgrupp oli tarkvara arendamise kulu hindamisega tegelevad inimesed, mistõttu levitati seda mitmetes Eesti tarkvara arendusega tegelevates firmades. Kokku osales küsitluses 50 inimest. Küsimustikus oli 13 küsimust ning küsimused on kvantitatiivsed.

Tarkvara kulu hindamisega tegelevatele inimestele mõeldud küsimustikus oli küsimusi selle kohta kui kaua on tegeletud tarkvara arendamisega ja kas kasutatakse agiilset või kose meetodit. Samuti oli küsimusi muude tausta-infoks vajalike faktorite kohta ning nendele järgnesid küsimused osade meetodite kohta ning kui täpselt nende meetodiga on võimalik hinnata.

2.1 Küsimustiku küsimused, vastused ja statistika

Järgnevalt on loetletud antud uurimistöö raames kasutatud küsitluse küsimused, lühidalt kirjeldatud nende eesmärk ning diagrammiga on ära näidatud, mis olid küsimustikule vastajate vastused protsentides.

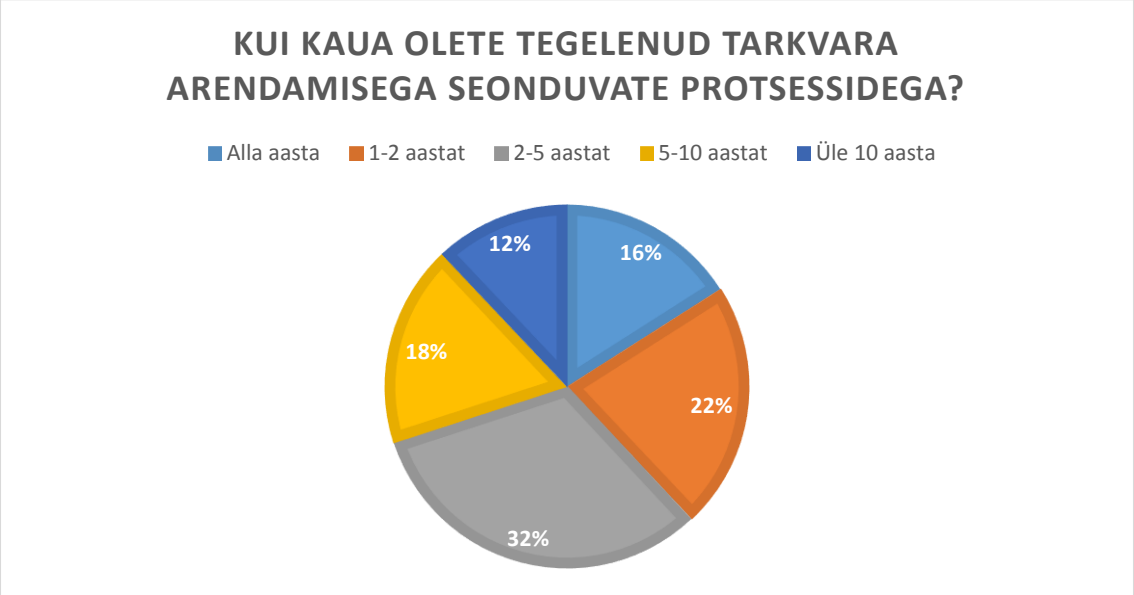
Küsimused jaotusid kolme rühma:

- esimeses rühmas olid küsimused, mis uurisid vastaja tausta kohta,
- teises rühmas olid küsimused meetodite kohta, mida antud töös tutvustati,
- kolmandas rühmas olid küsimused määramaks, kui hästi vastab kliendile antud hinnang tegelikkusele.

Esimesse rühma kuuluvad küsitluse neli esimest küsimust. Esimese küsimuse (Joonis 1) eesmärk oli teada saada küsitluses osalenute osalemise kogemus tarkvara arendamise protsessides, sest isegi siis kui otseselt tarkvara ajahindamisega ei tegeleta, nähakse kui kaua erinevad osad võivad ega võtta. Teise küsimuse (Joonis 2) eesmärgiks on näha, kas

arendamise meetod mõjutab teadmisi erinevatest ajahinnangu meetoditest. Kolmas küsimus (Joonis 3) oli vabas tekstis küsimus teise küsimuse laiendamiseks, et vastajatel oleks võimalus välja tuua täpne arendamise meetod. Neljanda küsimuse (Joonis 4) eesmärgiks oli teada saada,kui kaua on vastaja tegelenud ajahinnangute määramisega.

1. **Kui kaua olete tegelenud tarkvara arendamisega seonduvate protsessidega?**

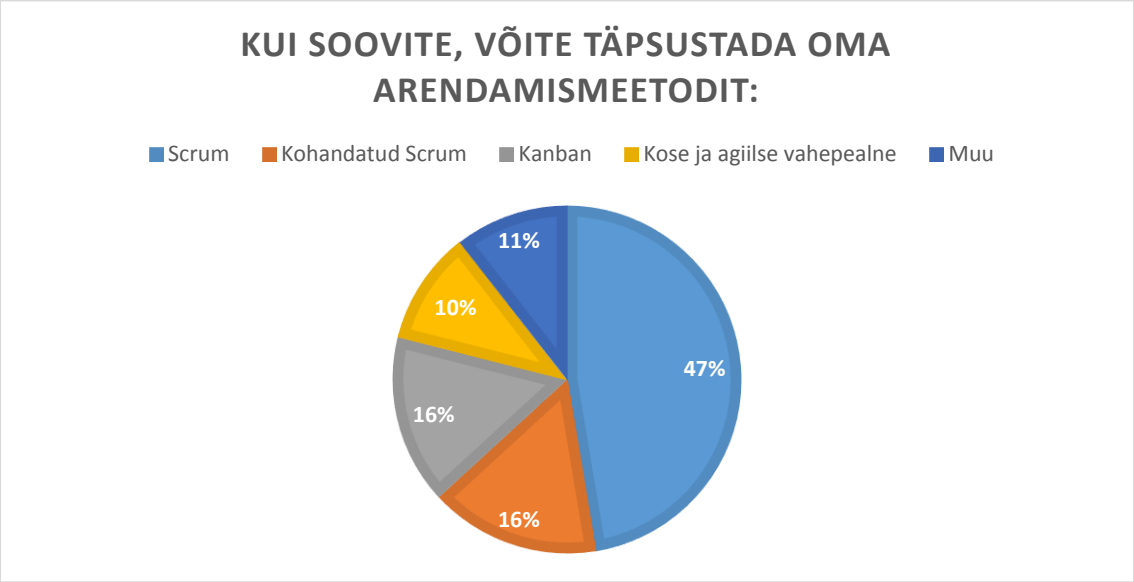


Joonis 1. Tarkvara arendamisega seonduvate protsessidega tegelemise kogemus.

2. **Kas kasutate tarkvara arendamiseks agiilset või kose meetodit?**

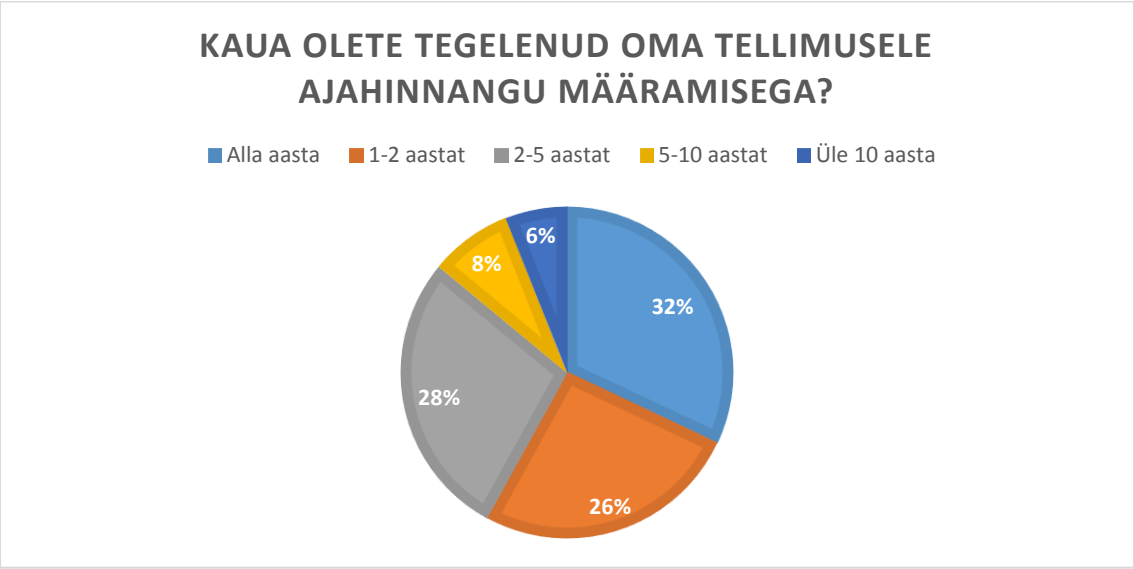
Joonis 2. Agiilse ja kose meetodi suhe vastajate seas.

3. Kui soovite, võite täpsustada oma arendamismeetodit:



Joonis 3. Täpsem arendamismeetod.

4. Kaua olete tegelenud oma tellimusele ajahinnangu määramisega?

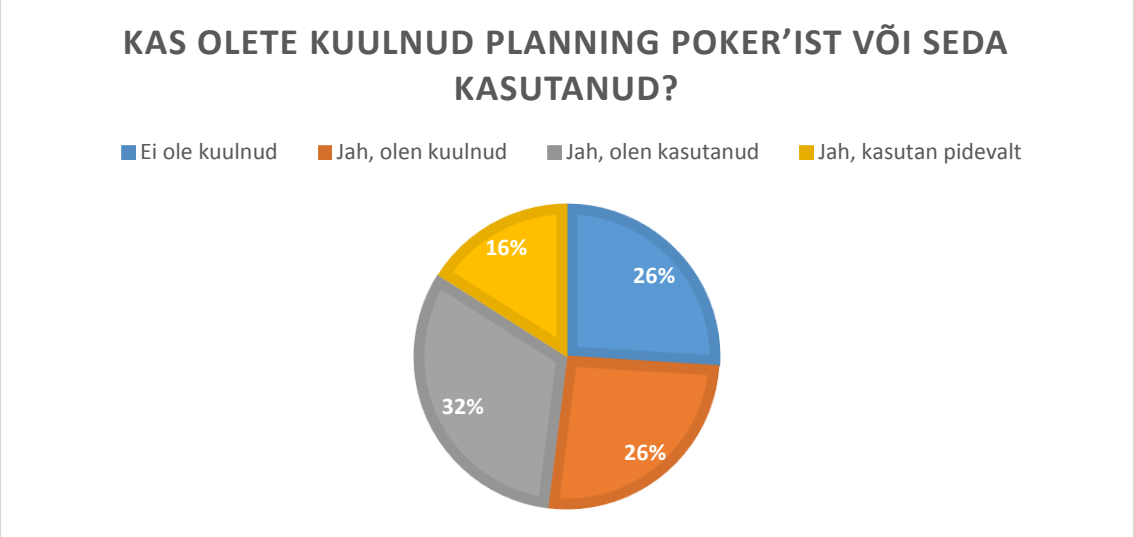


Joonis 4. Ajahindamise määramisega tegelemise kestvus.

Teise küsimuste rühma kuulub viis küsimust ning need on ajahindamise meetodite kohta. Küsimustiku viies küsimus (Joonis 5) oli planeerimise pokkeri kohta, mis on üks nendest meetoditest, mida töös lähemalt tutvustati ning selle eesmärgiks oli näha, kui paljudel vastajatel on teadmisi selle meetodi kohta ning kui paljud seda kasutavad. Kuues küsimus (Joonis 6) oli tegelikult küsimus MoSCoW analüüsi kohta, aga küsimustikus endas oli esitatud ilma nimeta. MoSCoW analüüs on samuti üks meetodeid, mida tutvustatakse töös

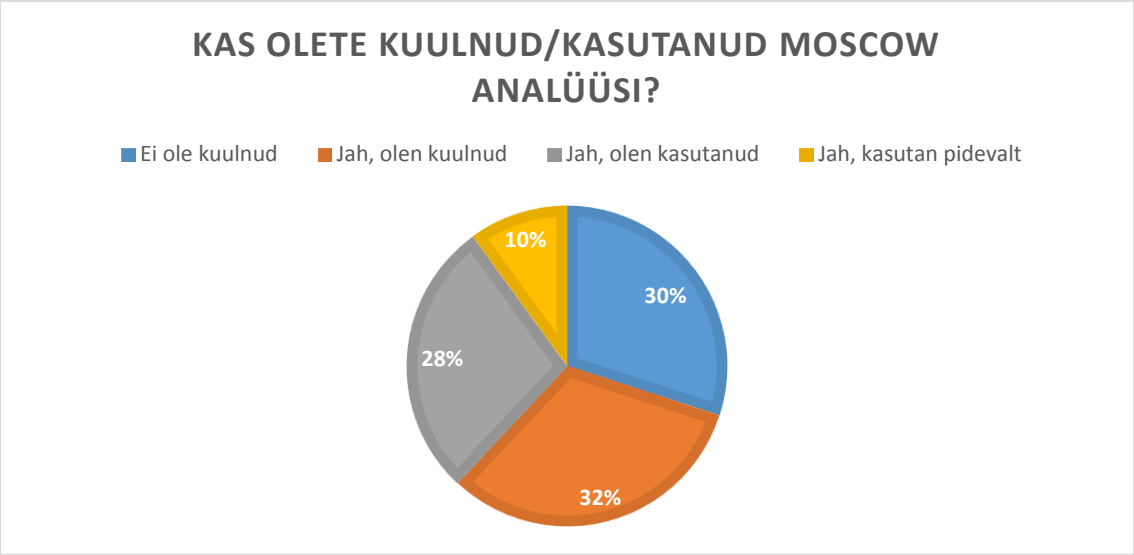
ning eesmärk eelmise küsimusega oli sama. Seitsmes küsimus (Joonis 7) oli vaba tekstiga küsimus, et vastajad saaksid kirja panna, millist meetodit nad kasutavad ajahinnangu andmiseks. Selle küsimuse eesmärk oli teada saada, kas mõni tutvustatud meetoditest on vastanute ajahinnangute meetodite sees. Kaheksas küsimus (Joonis 8) oli COCOMO II kohta, mis on sarnaselt planeerimise pokkeriga ja MoSCoW analüüsiga tutvustatud töös ning eesmärk oli teada saada, kui paljud vastajatest sellest teadlikud on. Üheksas ja selle grupi viimane küsimus (Joonis 9) on selle kohta, mida öeldakse kliendile ajahinnanguks võrreldes oma meetodi antud hinnanguga. Samuti võib sellest küsimusest välja lugeda, kas hindamismeetod on piisavalt usaldusväärne ning arvestab kõikide faktoritega, et selle tulemust ka kliendile öelda.

5. Kas olete kuulnud *planning poker*'ist või seda kasutanud?



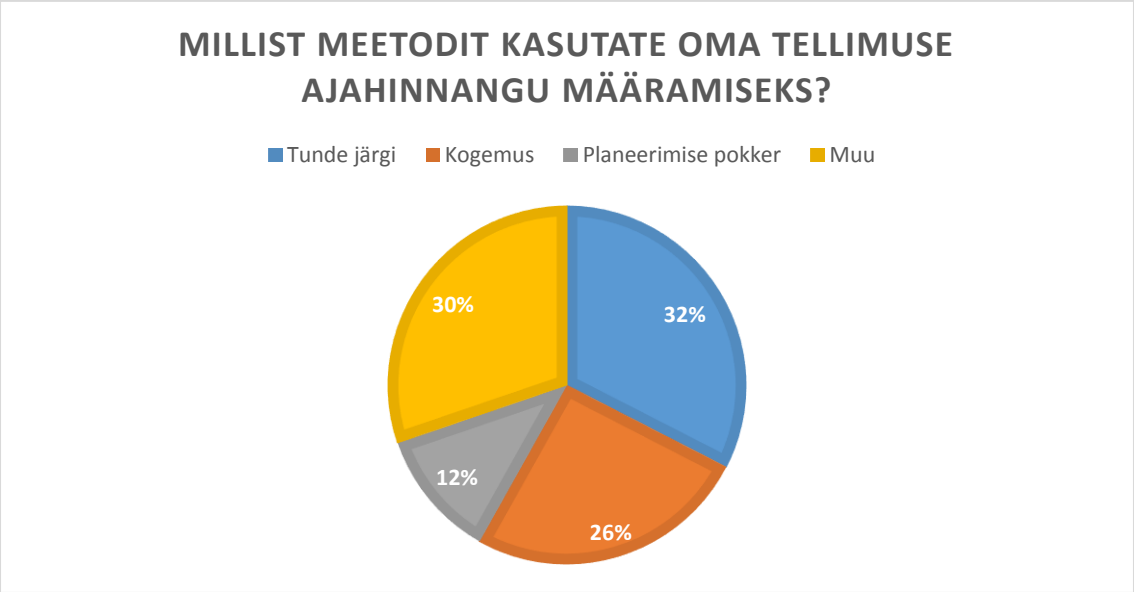
Joonis 5. *Planning poker*'i ehk planeerimise pokkeri teadmine.

6. **Kas olete kuulnud/kasutanud meetodit, kus jaotatakse kõik toote featuurid nelja kategooriasse: peab olema, peaks olema, võiks olla, ei tee (ehk MoSCoW analüüs)?**



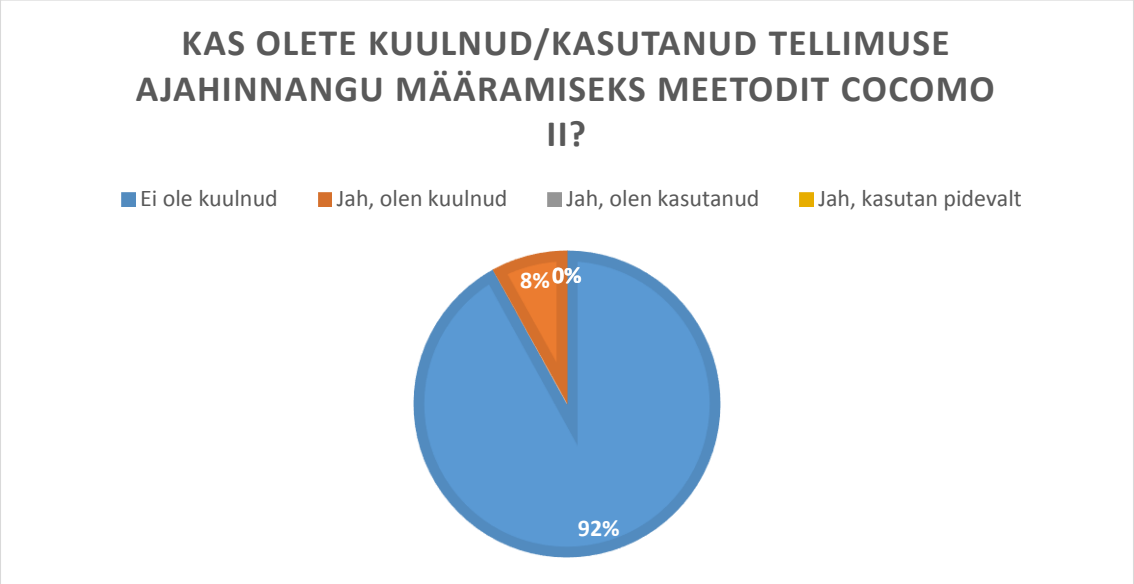
Joonis 6. MoSCoW analüüsi meetodi teadmine.

7. **Millist meetodit kasutate oma tellimuse ajahinnangu määramiseks?**



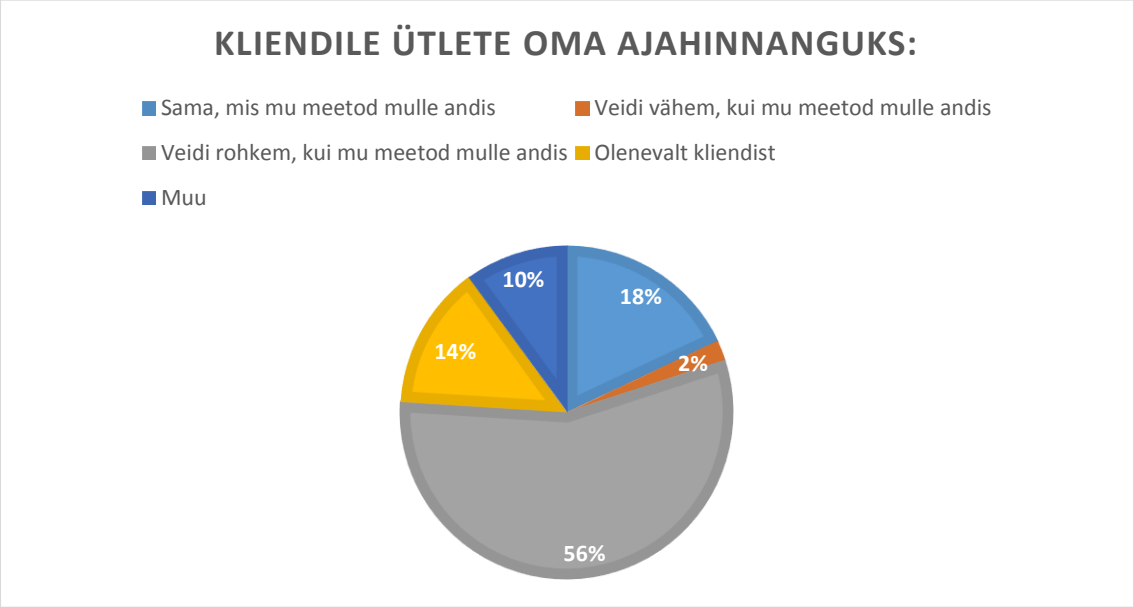
Joonis 7. Ajahinnangu määramise meetodid.

8. **Kas olete kuulnud/kasutanud tellimuse ajahinnangu määramiseks meetodit COCOMO II?**



Joonis 8. Meetodi COCOMO II teadmine.

9. **Kliendile ütlete oma ajahinnanguks:**

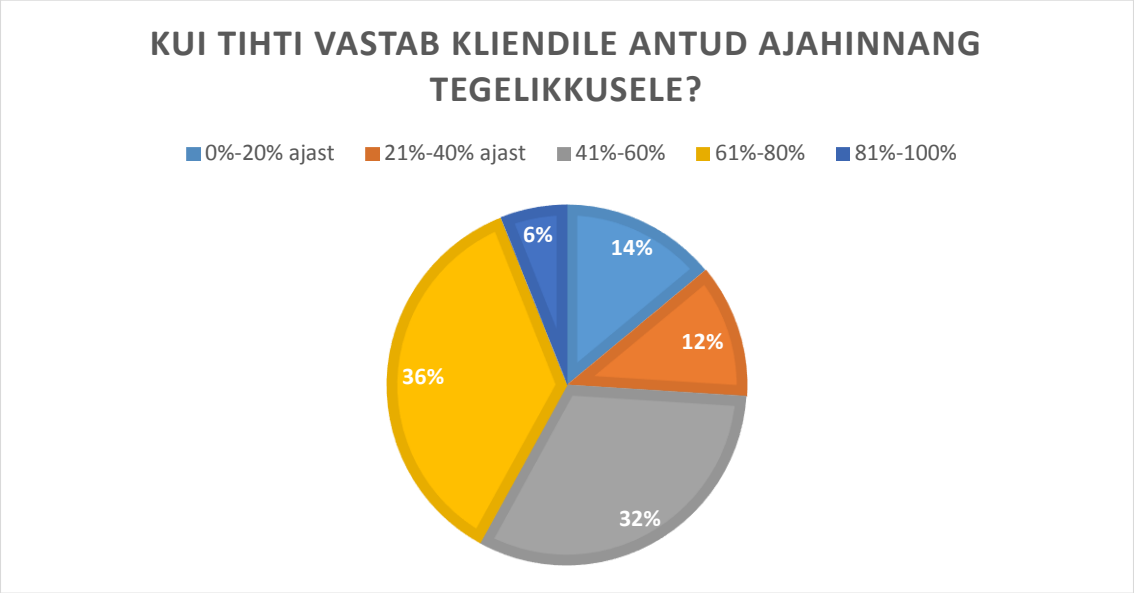


Joonis 9. Ajahinnangu andmine kliendile võrreldes meetodiga.

Kolmandasse ehk viimasesse küsimuste gruppi kuuluvad küsimustiku viimased neli küsimust, mis keskenduvad sellele, et teada saada, kui hästi peavad paika vastajate ajahinnangud tegelikkusele. Kümnes küsimus (Joonis 10) uurib vastajatelt üldiselt, kui tihti vastab ajahinnang tegelikkusele. Üheteistkümnenda küsimuse (Joonis 11) eesmärk

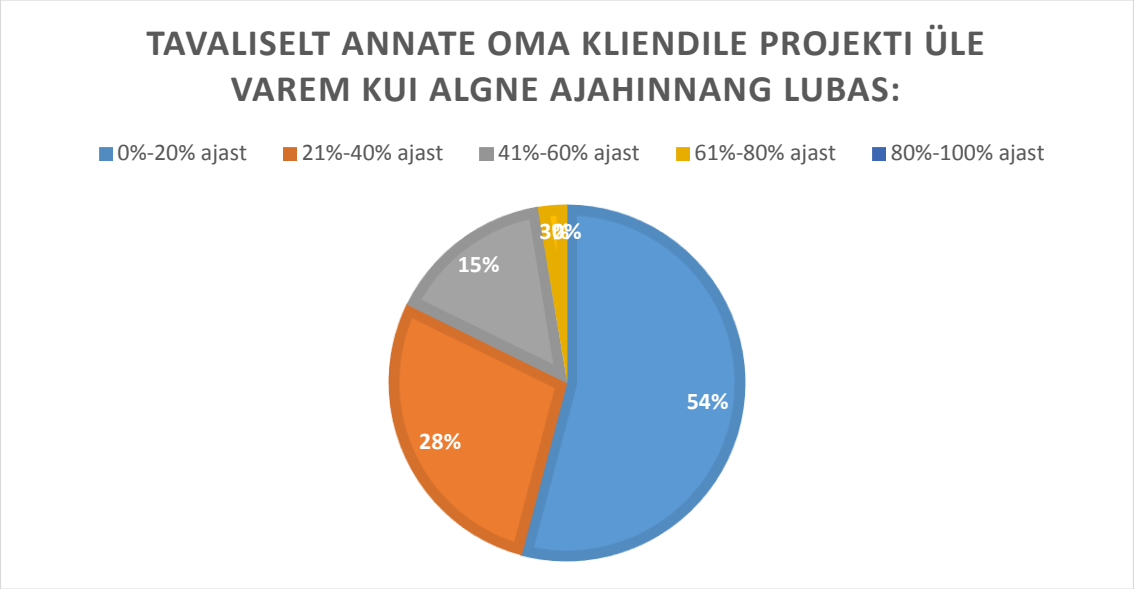
on teada saada, kui palju vastajatest annab projekti kliendile üle enne algselt hinnanguks pandud aja möödumist. Kaheteistkümnes küsimus (Joonis 12) näib küll sarnane kümnendaga, kuid selle all on mõeldud, et kui tihti antakse projekt üle kas siis enne tähtaega või täpselt sellel ajal, kui lubatud – ehk ilma hilinemiseta. Kolmeteistkümnes ja viimane küsimus küsimustikus (Joonis 13) oli mõeldud selleks, et teada saada, kui tihti minnakse oma projekti algsest ajahinnangust üle.

10. Kui tihti vastab kliendile antud ajahinnang tegelikkusele?



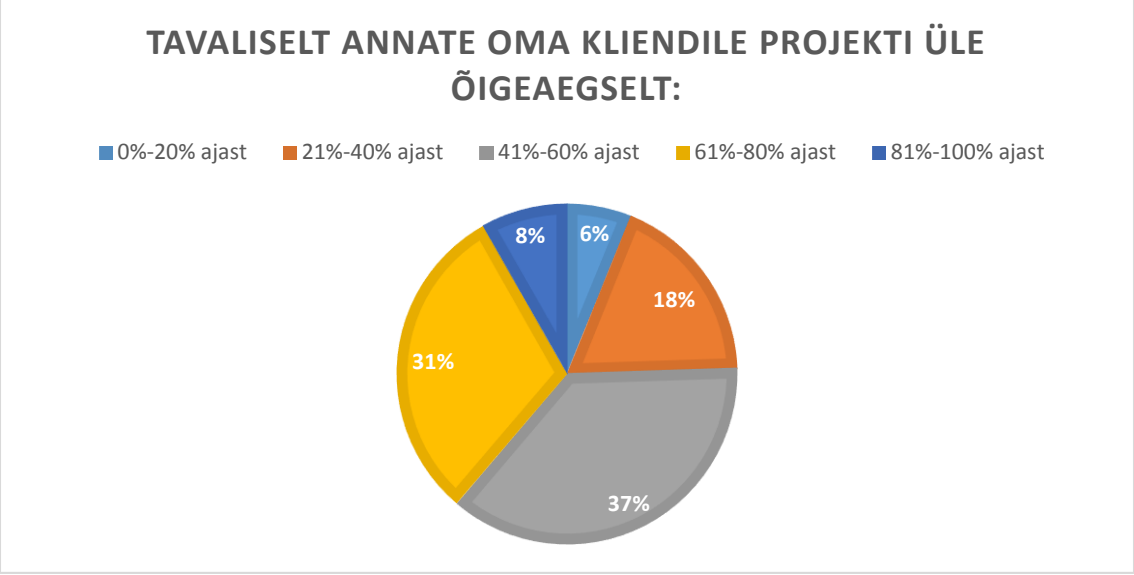
Joonis 10. Kliendile antud ajahinnangu vastavus tegelikkusele.

11. Tavaliselt annate oma kliendile projekti üle varem kui algne ajahinnang lubas:



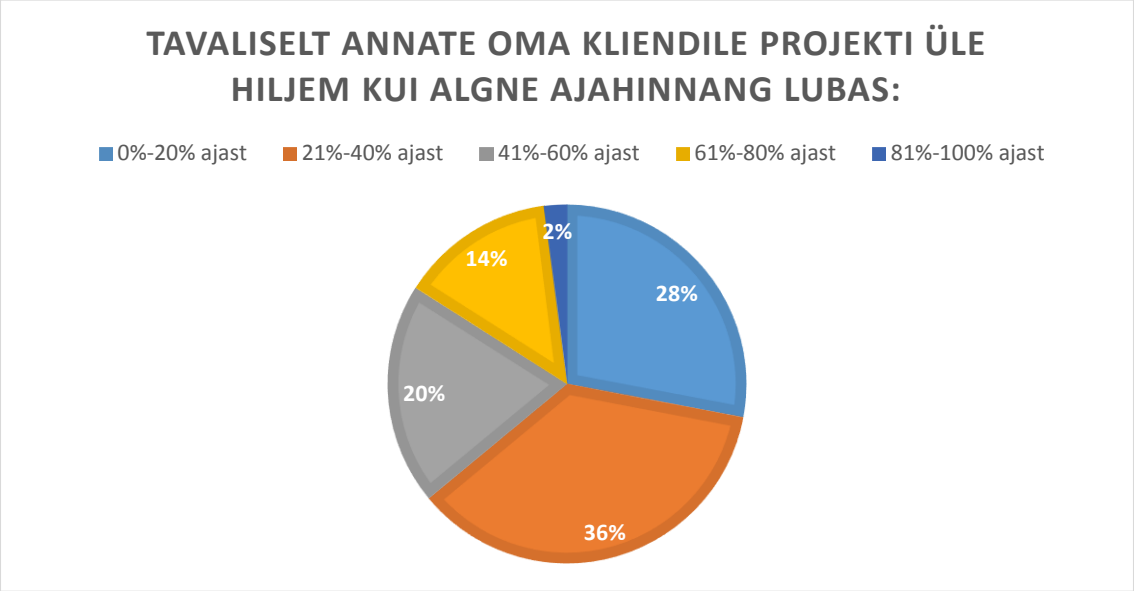
Joonis 11. Kliendile antava ajahinnangu vastavus tegelikkusele: varem üle andmine.

12. Tavaliselt annate oma kliendile projekti üle õigeaegselt:



Joonis 12. Kliendile antava ajahinnangu vastavus tegelikkusele: õigeaegselt üle andmine.

13. Tavaliselt annate oma kliendile projekti üle hiljem kui algne ajahinnang lubas:



Joonis 13. Kliendile antava ajahinnangu vastavus tegelikkusele: hiljaks jäämine.

2.2 Küsimustiku analüüs

Küsitluse tulemustest selgus, et enamus vastajatest annab tarkvara projektile lõpliku kuluhinnangu tunde järgi. Samuti selgus, et enamik vastajatest oli vähemalt kuulnud planeerimise pokkerist ning seda ka kasutanud. MoSCoW analüüsi kasutab pidevalt vaid 10% inimestest, kuid enamik oli siiski sellest vähemalt kuulnud. Vastajate valimist võib järeldada, et enamik Eestis tegutsevatest tarkvaraarenduse ettevõtetest ei kasuta meetodit COCOMO II, sest vaid 8% vastajatest oli sellest kuulnud, kuid keegi polnud seda kasutanud.

Valdav osa küsimustikule vastajatest on tegelenud projektide ajahinnangu määramisega üle aasta ning 90% vastajatest kasutab agiilset metoodikat tarkvara arendamiseks. Nendel vastajatel, kes hilinevad projektide üle andmisega kliendile kõigest 0%-20% kordadest, on ajahindamisega üldiselt (71 protsendil) rohkem kui 2 aastat kogemust ning kõik kasutavad agiilset meetodit arendamiseks.

2.2.1 Planeerimise pokkeri statistika

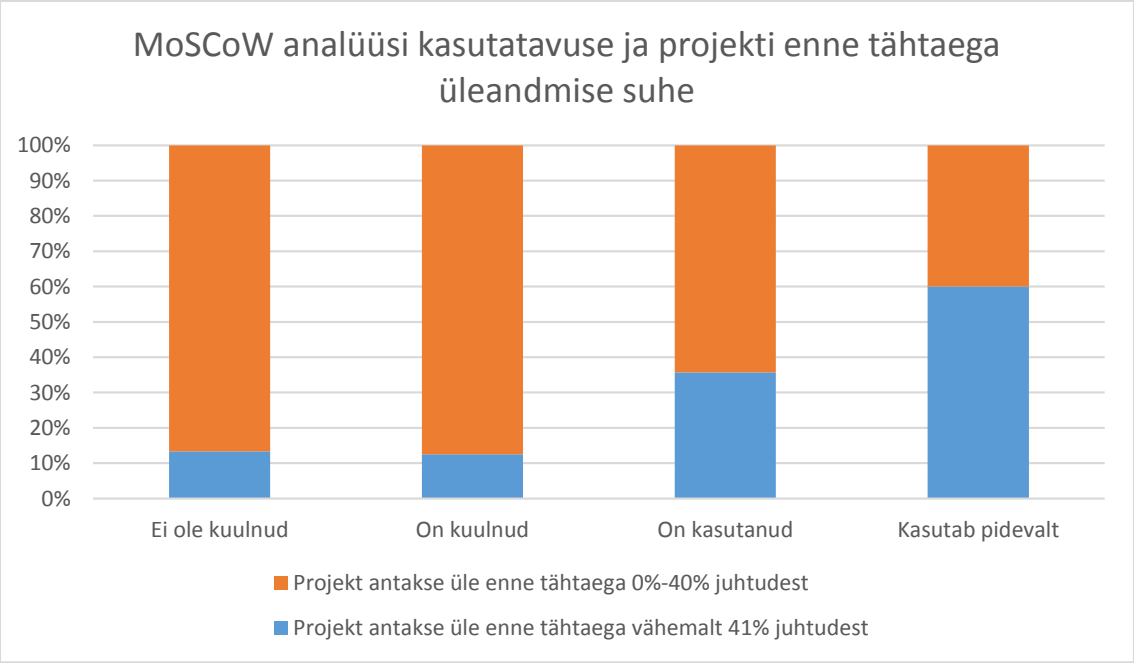
75% inimestest, kes kasutavad pidevalt planeerimise pokkerit, ütlevad, et vähemalt 41% või rohkem nende töödest saab üle antud õigeaegselt. Samuti on pidevalt planeerimise

pokkeri kasutajate hulgas näha väiksemat tendentsi hilineda projekti üle andmisele kliendile. Nimelt inimestest, kes ei ole kuulnud planeerimise pokkerist suudavad ainult 46% anda projekti üle 0%-20%lise hilinemise tõenäosusega, samas kui inimesed, kes kasutavad pidevalt planeerimise pokkerit, nendest suudab 63% inimesi anda projekti üle 0%-20%lise hilinemise tõenäosusega. Ehk need inimesed, kes ei ole kuulnud planeerimise pokkerist jäävad projekti üle andmisega suurema tõenäosusega hiljaks kui need, kes seda meetodit pidevalt kasutavad.

Kõik inimesed, kes annavad oma projekti üle õigeaegselt 81%-100% kordadest, on kuulnud või kasutavad pidevalt planeerimise pokkerit.

2.2.2 MoSCoW analüüsi statistika

MoSCoW analüüsist on vähemalt kuulnud 70% küsitluses osalenutest ning selgus, et inimesed, kes ei olnud kuulnud sellest meetodist, andsid keskmiselt 42%-61% juhtudest projekti üle kliendile õigeaegselt, samas kui inimesed, kes pidevalt kasutasid MoSCoW analüüsi andsid keskmiselt 53% kuni 72% juhtudest projekti kliendile üle õigeaegselt. Sarnaselt kasvas ka inimeste hulk, kes suudavad varem anda tarkvara kliendile üle enne algset tähtaega (Joonis 14).



Joonis 14. MoSCoW analüüsi ja projekti enne tähtaega üleandmise suhe.

2.2.3 Kokkuvõttev analüüs ja soovitused

Kulu hindamise metoodikate mitte tundmine ja kasutamine mõjutab kulu hindamise täpsust. Küsimustiku vastustest saab järeldada, et need inimesed, kes ei ole kuulnud ühestki meetodist, mida küsimustikus käsitleti, annavad projekti kliendile keskmisest harvemini õigeaegselt või varem üle. Sellest saab teha järelduse – olles teadlik erinevatest meetoditest, mida kasutada, osatakse ka paremaid hinnanguid anda.

Küsimustikust saab kinnitust ka see, et kasutades kas planeerimise pokkerit või MoSCoW analüüsi saadi tihedamini projekt õigeaegselt kliendile üle antud. MoSCoW analüüsi ja planeerimise pokkerit saab kasutada agiilsete arendamismetoodika puhul. Siit tuleb ka järgmine soovitus: kui erinevate meetodite kohta on uuritud, siis tuleks valida, pidades silmas kasutatavat arendusmetoodikat, meeskonnale sobivaim ning kasuta seda.

Kui soovitakse oma meeskonnas alustada kulu hindamise meetodite kasutamist, siis selleks tuleks alustada kindlasti mõnest lihtsamast meetodist, mis meelestaks meeskonna positiivselt ning tooks välja kuluhindamise head küljed. Sel juhul on meeskonnal lihtsam hindamisse sisse elada ning hiljem on võimalus liikuda edasi keerulisemate ja täpsemate peale.

Kokkuvõte

Käesoleva tööga tutvustati ning võrreldi tarkvaraarenduse kulu hindamise metoodikaid ning anti soovitusi, kuidas hinnangute andmist täpsemaks muuta. Sellega täidab töö oma eesmärki: võrrelda tarkvaraarenduse kulu hindamise metoodikaid ning anda soovitusi, et aidata kaasa hinnangute täpsustumisele Eestis. Töö esimeses pooles anti lühike ülevaade viiest erinevast kulu hindamise metoodikast: planeerimise pokker, kasutusjuhu punktid, COCOMO ja COCOMO II, tõestusel põhinev planeerimine ja MoSCoW analüüs. Peatükk lõpetati neid võrdleva tabeliga

Teine osa tööst keskendub praegustele tarkvaraarendamise kulu hindamise kommetele Eestis ning kuidas neid mõjutavad eelmainitud metoodikad. Seoste leidmiseks koostati küsimustik, millele vastas 50 inimest. Küsimustiku vastustest sai järeldada, et erinevate meetodite teadmine ja kasutamine aitab kaasa täpsemalt tarkvaraarenduse kulu hindamisele. Teine osa lõppeb soovitustega, mida järgides, on võimalik tarkvaraarenduse kulu hindamist täpsustada.

Töö edasiarendus oleks luua detailsem ülevaade tarkvaraarenduse kulude hindamise metoodikatest ja nende sobivusest erinevat tüüpi tarkvaraarendus ettevõtetele.

Kasutatud kirjandus

[1] Cohn Mike. Agile Estimating and Planning.United States of America: Prentice Hall, 2005. (May 1, 2016)

[2] Chemuturi Murali.Softare Estimation Best Practices, Tools & Techniques: A Complete Guide for Software Project Estimators. United States of America : J. Ross Publishing, 2009. (May 18, 2016)

[3] Boehm Barry W. Software Engineering Economics. United States of America: Prentice-Hall, 1981. (May 5, 2016)

[4] Sommerville Ian. Software Engineering. – 2007. England: Pearson Education Limited. (March 7, 2016)

[5] Joel Spolsky. Evidence Based Scheduling. – Joel on Software 2007 [WWW] http://www.joelonsoftware.com/items/2007/10/26.html (May 10, 2016)

[6] Paul Barnes. Software Costs Estimation In Agile Project Management. — Toptal 2015, [WWW] https://www.toptal.com/agile/software-costs-estimation-in-agile-project-management (March 7, 2016)