

Tallinna Pedagoogikaülikool

Informaatika osakond

XP-metoodika juurutamisest väikeses eesti tarkvarafirmas.

Magistritöö

Gunnar Piho

Juhendaja: Peeter Normak, professor

Tallinn 2003

Autor : .....” .....” ..... 2003.a.

Juhendaja : .....” .....” ..... 2003.a.

Osakonna juhataja : .....” .....” ..... 2003.a.

# ÜLEVAADE

Tarkvara arendusmetoodikate probleemistik, olles sama või peaaegu sama vana kui tarkvara tootmine, on väga aktuaalne ka täna. Paindmetoodikad\*, standardid ja küpsusmodelid on teemad mille üle arutletakse teaduslikel konverentsidel ning mida kajastatakse prestiižikates ajakirjades.

Käesolevas magistritöös üldistan *Extreme Programming* metoodika (edaspidi XP-metoodika) juurutamise kogemust väikeses eesti tarkvarafirmas.

Väitekirja esimeses osas annan lühikese ülevaate tarkvara arendusprotsessist, arendusprotsessi hindamise kriteeriumidest ja kaasaegsetest väikefirmale sobivatest arendusmetoodikatest.

Teises osas hindan XP-metoodikat lähtuvalt SW-CMM (*Capability Maturity Model for Software*) teisest tasemest ja näitan kuidas XP-metoodikat kasutades on võimalik SW-CMM teise taseme nõudeid rahuldada.

Töö kolmandas osas analüüsin eesti tarkvaraarendajate suhtumist SW-CMM teise taseme nõuetesse ning XP-metoodikas kasutatavatesse meetoditesse. Samuti analüüsin eesti tarkvaraarendajate kogemusi lähtuvalt SW-CMM teise taseme nõuetest ning XP-metoodikast.

Magistritöö neljas osa on juhtumianalüüs, kus analüüsin XP-metoodika juurutamisel saadud kogemusi tarkvara arendusfirma Systek Informationsystems OÜ näitel.

---

\* Siin edaspidi jälgin mõistete “*agile method*” ja “*lightweight method*” eesti keelde tõlkimisel professor Leo Võhandu soovitusi ja tõlgisin neid mõlemaid eesti keelde kui “paindmeetod”. Analooiliselt kasutan ka mõistet “paindmetoodika”. Mõisteid “*heavy methods*”, “*monumental method*” eestikeelseteks vasteteks kasutan tema soovitusel mõistet “tardmeetod”. Analooiliselt siis ka “tardmetoodika”.

# TÄNUSÕNAD

Tänu Tallinna Pedagoogikaülikoolile ja eriti ülikooli Informaatika osakonnale IT juhtimise magistriõppe korraldamise eest.

Suured tänud õppejõududele (tähestikulises järjekorras) ...

Tiit Elenurm, Matti Heidmets, Indrek Hiie, Eero Johannes, Tõnu Lehtsaar, Paul Leis, Ants Leitmäe, Katrin Niglas, Peeter Normak, Priit Parmakson, Aleksander Pulver, Harri Roots, Indrek Tart, Jaan Teng, Jaak Tepandi, Avo-Rein Tereping

... teie vaeva eest kursuste ettevalmistamisel ja nende läbiviimisel.

Suured tänud juhendajale, professor Peeter Normakule, abi ning toetuse eest antud töö kirjutamisel ja Katrin Niglasele andmetöötuse alaste konsultatsioonide eest.

Tänu firma Systek Informationsystems OÜ omanikele ja töökaaslastele toetuse eest.

Tänu kursusekaaslastele, Nele Pihlakule, kahe õpinguaasta jooksul minuga koos tehtud aine ja kursusetööde eest minu väitekirjaga kaasnevatel teemadel.

Tänu oma perele toetuse ja abi eest. Tänu abikaasa Sirjele abi eest antud töö keelelise korrigeerimise eest. Tänu poeg Rasmusele abi eest testide elektroonse versiooni valmistamise ja nende internetis kättesaadavaks tegemise ning antud töö inglise keelse kokkuvõtte kirjutamise eest. Tänu Paulile, Laurale ja Sandrale kannatlik olemise eest.

# SISUKORD

Ülevaade .....	3
Tänu sõnad .....	4
Sisukord .....	5
1 Sissejuhatus.....	7
1.1 Probleem ja selle aktuaalsus .....	7
1.1.1 Töö eesmärgid.....	7
1.1.2 Töö aktuaalsus. ....	8
1.1.3 Kasutatud uurimismeetodid .....	8
1.1.4 Analoožilised tööd .....	8
1.1.5 Systek Informationsystems OÜ .....	9
1.2 Põhimõisted.....	10
1.2.1 Tarkvara arendusprotsess.....	10
1.2.2 Olulised ajalooetapid tarkvara arenduses.....	11
1.2.3 Metodoloogia, meetodika ja meetod tarkvaraarenduses.....	12
1.2.4 Tarkvara arendusprotsessi täiustamine. ....	12
1.2.5 Tarkvaraprojekti neli mõõdet.....	13
1.2.6 Väike tarkvarafirma .....	14
1.3 Arendusprotsessi hindamise kriteeriumid.....	14
1.3.1 SW-CMM .....	14
1.3.2 CMMI .....	15
1.3.3 ISO 12207 .....	15
1.3.4 ISO 15504 .....	16
1.3.5 Miks antud töös on lähtutud CMM nõuetest?.....	16
1.4 Arendusmetoodikad .....	17
1.4.1 Paind- ja tardmetoodikad. ....	18
1.4.2 Paind- ja tardmetoodikate sobivus tarkvara arendamiseks .....	19
1.4.3 Tuntumad väikefirmadele sobivad paindmetoodikad.....	21
2 SW-CMM teine tase ja XP-metoodika .....	25
2.1 SW-CMM tasemed. ....	25
2.1.1 Esimene tase.....	25
2.1.2 Teine tase .....	26
2.1.3 Kolmas tase.....	26
2.1.4 Neljas ja viies tase.....	28
2.2 SW-CMM struktuur .....	28
2.2.1 Tegevuskava ja vastutus .....	30
2.2.2 Tingimuste loomine .....	30
2.2.3 Tegevuste korraldamine.....	30
2.2.4 Tulemuste mõõtmine .....	30
2.2.5 Kontroll ja järeltöö tegemine.....	30
2.3 Ülevaade XP-metoodikast .....	31
2.4 XP-metoodika võimaldab täita SW-CMM teise taseme nõudeid.....	33
2.4.1 Nõuete haldamine .....	34
2.4.2 Tarkvaraprojekti planeerimine.....	36
2.4.3 Tarkvaraprojekti monitooring.....	39
2.4.4 Allhangete juhtimine.....	43
2.4.5 Kvaliteedi tagamine .....	43
2.4.6 Konfiguratsioonide haldamine.....	47
3 Empiiriline uuring - Eesti arendajad, CMM ja XP-metoodika.....	51
3.1 Küsitluse eesmärk .....	51

3.2	Küsimustik .....	51
3.2.1	Taustinformatsiooniga seotud küsimused.....	51
3.2.2	Arendusprotsessiga seotud küsimuste koostamise põhimõtted .....	51
3.2.3	Arendusprotsessi meetoditega seotud küsimused.....	53
3.2.4	Teise ja kolmanda osa vastusevariandid .....	53
3.2.5	Arendajate hirmudega seotud küsimused .....	53
3.3	Küsitluse läbiviimine. ....	53
3.4	Andmete töötlemise vahendid.....	54
3.5	Vastajad ja nende taust.....	54
3.6	Küsitlusele vastanute rühmitamine .....	57
3.7	Andmete töötlemine .....	58
3.7.1	Arendusprotsessiga seotud küsimuste rühmitamine .....	58
3.7.2	Arendusmeetoditega seotud küsimuste rühmitamine .....	60
3.7.3	Küsimuste “arendajate hirmud” rühmitamine. ....	61
3.8	Tulemused.....	61
3.8.1	Arendusprotsessiga seotud küsimused.....	62
3.8.2	Arendusmetoodikatega seotud küsimused .....	67
3.8.3	Hirmudega seotud küsimused .....	73
3.9	Ankeedi kokkuvõte ja järeldused.....	75
4	XP juurutamisest firmas systek informationsystems .....	79
4.1	Kuidas ja miks asi algas? .....	79
4.2	Mida ja kuidas me oleme teinud? .....	80
4.2.1	Planeerimine .....	80
4.2.2	Disainimine .....	83
4.2.3	Programmeerimine.....	84
4.2.4	Testimine.....	86
4.3	Kaugel me oleme SW-CMM teisest küpsustasemest?.....	88
	Kokkuvõte.....	93
	Töö võimalikud edasiarendamised .....	95
	Kasutatud kirjandus .....	96
	Summary .....	99
	Lisad.....	102
	Lisa.1. SW-CMM teise taseme nõuded. ....	102
	Lisa.2. XP-metoodika .....	119
	Lisa.3. Küsimustik .....	125
	Lisa.4. Andmetöötlus .....	129
	Lisa.5. Näited XP meetodite kasutamisest firmas Systek.....	141
	Lisa.6. Tarkvaraprojekti test .....	146

# 1 SISSEJUHATUS

## 1.1 Probleem ja selle aktuaalsus

### 1.1.1 Töö eesmärgid

Antud töö eesmärgid on välja kujunenud minu praktilisest tegevusest väikese tarkvara arendusfirma juhina. Alljärgnevalt need liiga tihti korduma hakanud probleemid, millele tuli lahendus leida:

- Müügiga tegelevad inimesed suruvad peale ebareaalseid tähtaegu;
- Projektid venivad;
- Kiirustamisest tekivad vead, kuna testimisele jääb vähe aega;
- Vigade parandamine toob tihti juurde vigu kohtades, mis enne töötasid;
- Arendajad on depressioonis tähtaegade ja pidevate ületundide pärast;
- Ebareaalsetest tähtaegadest ja üleväsimusest tingituna on arendajate motivatsioon langenud;
- Kliendi tarkvaralised soovid pole arusaadavad, või osutuvad vääradeks või valesti mõistetuteks alles rakenduse ekspluateerimise ajal
- Kliendid on närvilised, suhtlemine nendega muutub väga tihti destruktiivseks.

Loetelu võib jätkata. Tõenäoliselt on analoogilised probleemid tuttavad igale tarkvara arendamisega tegelevale inimesele.

2001 aasta sügisel, kui alustasin magistriõpinguid Tallina Pedagoogikaülikooli IT juhtimise erialal, tegin ka ettekande firma Systek Informationsystems OÜ omanikele ja kaastöötajatele ülal loetletud probleemidest ning pakkusin välja võimaliku lahenduse arendusmetoodika *Extreme Programming* (edaspidi XP) juurutamise näol firmas.

Oma magistritöös üldistan XP juurutamise kogemust ja vastan järgmistele küsimustele:

- Kas XP metoodika sobib SW-CMM (*Capability Maturity Model for Software*) [CMM] teise taseme nõuete saavutamiseks?
- Kuidas juurutada XP metoodikat väikeses eesti tarkvara arendusfirmas?

Töös annan lühikese ülevaate tarkvara arendusprotsessist, arendusprotsessi hindamise kriteeriumidest ja kaasaegsetest väikefirmale sobivatest arendusmetoodikatest.

Töös analüüsin:

- XP metoodikat lähtuvalt SW-CMM teisest tasemest;
- eesti arendajate SW-CMM ja XP hoiakuid ning kogemusi;
- XP juurutamise juhtumit firmas Systek Informationsystems OÜ.

XP valiku peamiseks põhjuseks on antud metoodika kompleksne lähenemine väikefirma tarkvara arendusprotsessile (vaata 2.1.1) . SW-CMM valiku kriteeriumiks (vaata 1.3.5) on aga selle dokumendi sisemine loogika, mis läheneb firma tarkvara arendusprotsessi täiustamisele ning arendamisele lähtuvalt tarkvarafirma kogemustest.

### 1.1.2 Töö aktuaalsus.

Käesoleva töö temaatika – tarkvara arendusprotsess ja selle täiustamine eesmärgiga toota tarkvara kvaliteetsemalt, odavamalt ja kiiremini – on kogu maailmas jätkuvalt aktuaalne. Pidevalt ilmub selleteemalisi raamatuid, kogumikke, artikleid, dissertatsioone, ...

Eesti kontekstis pean tööd aktuaalseks järgmistel põhjustel:

- Vähe on antud temaatikaga haakuvat emakeelset erialakirjandust;
- Kümme aastat tarkvaratootmist iseseisvas Eestis on nii mõnegi firma viinud süsteemse arendusmetoodika juurutamise vajaduse tunnetamisele (hüpotees leiab kinnitust ka antud töös)
- Autorile on teada ainult üks arendusmetoodika rakendamise juhtumianalüüsi käsitlev uurimustöö Eestis [SEEBBA2001]. Asko Seeba käsitleb selles töös unifitseeritud tarkvaraarendamise protsessi (RUP) rakendamist. RUP on eelkõige suurfirmadele mõeldud arendusmetoodika.
- Suurem osa firmadest ja umbes pooled arendajad töötavad väikefirmades (hüpotees leiab kinnitust ka antud töös). Suurfirmadele mõeldud metoodikad ei ole väikefirmades rakendatavad.

Firma Systek Informationsystems OÜ kontekstis on temaatika aktuaalne tema seotuse tõttu firma igapäevaste probleemidaga tarkvara arendamisel.

Mulle isiklikult on antud teema südamelähedane ja aktuaalne minu igapäevaste töökohustuste tõttu.

Kokkuvõttes võiks antud töö huvi pakkuda kõikidele neile, kellel on soov täiustada tarkvara arendusprotsessi firmas või grupis. Töö kirjutamisel olen püüdnud anda piisava ülevaate nii arendusprotsessist, selle hindamise kriteeriumitest kui ka kaasaegsetest metoodikatest selleks, et töö või selle mingi osa oleks kasutatav ka sissejuhatava, kuid ammendava õppevahendina antud temaatikasse.

### 1.1.3 Kasutatud uurimismeetodid

Antud töös on kasutatud järgmisi meetodeid:

- Töö erialase kirjandusega;
- Tarkvara arendajate küsitlemine;
- Juhtumi analüüs.

Töö teoreetiline osa tugineb tööle erialase kirjandusega. Eesti arendajate SW-CMM ja XP hoiakuid ning kogemusi analüüsin lähtuvalt koostatud ja arendajate hulgas läbi viidud küsimustikule. Juhtumianalüüsi kasutan firmas Systek Informationsystems OÜ XP- metoodika juurutamise kogemuste analüüsimisel.

### 1.1.4 Analoogilised tööd

Alljärgnevalt ülevaade mõningatest antud tööga analoogilistest töödest.



Olen valinud suurest hulgast kõikidest internetist kättesaadavatest analoogilistest töödest (ja piiratud hulgast nendest töödest, millega suutsin tutvuda ) välja kaks tööd.

Asko Seeba töö [SEEBA2001], kui ainukese mulle teada eestikeelse antud valdkonda kuuluva töö ning Daniel Karlström poolt kirjutatud töö [KARLSTRÖM2002], kui minu tööga temaatikalt väga sarnase töö.

#### 1.1.4.1 Unifitseeritud tarkvaraarendamise protsess ja selle rakendamise juhtumianalüüs

Magistritöö “ Unifitseeritud tarkvaraarendamise protsess ja selle rakendamise juhtumianalüüs” on kirjutanud aastal 2001 Asko Seeba poolt [SEEBA2001].

Väitekirj tutvustab unifitseeritud tarkvaraarendusprotsessi ja selle rakendamist [SEEBA2001] :

- Kirjeldab unifitseeritud protsessi ja seda, miks see protsess tarkvaraprojektides hea kasutada on;
- Kirjeldab juhtumianalüüsi esimestest katsetest rakendada seda protsessi Cybertetica AS-is paari Asko Seeba juhitud projekti peal;
- Räägib sellest, kuidas organisatsiooni tasemel arendusprotsessi juhtida ja kirjeldab SW-CMM-i.

Erinevalt Asko Seeba tööst, kus vaatluse all on eelkõige suurtele firmadele mõeldud unifitseeritud tarkvara arendamise protsessi (RUP) juurutamise küsimused, vaadeldakse käesolevas töös väikefirmadele mõeldud arendusmetoodika XP juurutamist väikeses tarkvara arendusfirmas.

#### 1.1.4.2 Inimeste kaasamine arendusprotsessi täiustamisse

Oma Lundi Ülikoolis 2002. aastal kaitstud teesides (*Licentiate in Engineering*) “*Increasing Involvement in Software Process Improvement*” keskendub töö autor Daniel Karlström järgmistele probleemidele [KARLSTRÖM2002]:

- Kas inimeste kaasamine arendusprotsessi täiustamisse annab efekti ja kui siis millist?
- Millist efekti arendusprotsessi täiustamisele annab paindmetoodikate rakendamine väikeses tarkvara arendusfirmas?
- Kuidas paindmetoodikad tegelikult töötavad?

Käesolev töö ja Daniel Karlström poolt kirjutatud töö käsitlevad samu probleeme. Mõlemas töös on aluseks SW-CMM ja mõlemad vaatlevad XP metoodikat. Selles suhtes on Daniel Karlströmi tulemusi huvitav võrrelda, mida antud töös ka tehakse. Antud töö suurim erinevus Karlströmi tööst on katse üldistada arendajate hoiakuid ja kogemusi. Karlström teeb oma järeldused konkreetselt XP projektides osalenud arendajaid küsitledes.

#### 1.1.5 Syspek Informationsystems OÜ

Antud töös üldistatakse XP-metoodika juurutamise kogemusi firmas Syspek Informationsystems OÜ.

Firma Systek Informationssysteme OÜ loodi 1999. aasta kevadel Saksamaal LV-s tegutseva firma Systek Informationssysteme GmbH (vt. [www.systek.de](http://www.systek.de)) omanike poolt.

Juriidiliselt on tegemist iseseisva firmaga. Praktiliselt on tegemist Saksamaal asuva emafirma tarkvara arendamise ja väljatöötamise osakonnaga. Finantsilistel kaalutlustel lõpetati Saksamaal igasugune uute toodete arendustegevus. Jäeti ainult müük ja hooldustegevus. Kogu arendustegevus toimub Eestis.

Firma alustas kahe arendajaga. 2001. aasta mais lisandus veel kaks arendajat. Et arendajad ei peaks raiskama oma aega administratiivsetele tegevustele, ostetakse raamatupidamise ja firma juhtimise tegevused lepinguga väljast sisse. Üks arendaja täidab projektijuhi ülesandeid.

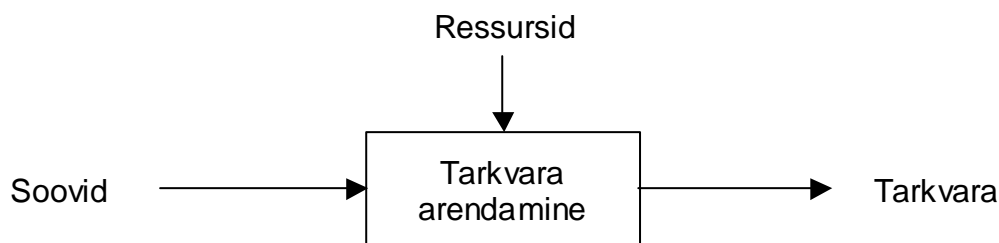
Firma põhitegevuseks on tarkvara tootmine meditsiinilaboratooriumitele. Meie turg on peamiselt Saksamaa Liitvabariik.

Alates 2001 aasta sügisest üritatakse tarkvara arendamise metoodikana juurutada XP-metoodika reegleid ja tehnikaid [Beck2000].

## 1.2 Põhimõisted

### 1.2.1 Tarkvara arendusprotsess

Tuginedes Daniel Karlströmile [KARLSTRÖM2002] on tarkvara arendamine protsess, mille käigus muudetakse kliendi tarkvaralised soovid (ideed, kavad, plaanid, unistused, ...) lähtudes kasutuses olevatest ressurssidest (aeg, teadmised, raha, kasutatavad tehnoloogiad, arendajad, masinad, ...) klienti rahuldavaks testitud ja dokumenteeritud rakenduseks. (Joonis:1).



*Joonis 1. Tarkvara arendamine [KARLSTRÖM2002].*

Kuna tarkvara arendamine on protsess, siis on teda võimalik juhtida nagu ka igat teist protsessi. Lähtuvalt Henri Fayol poolt 1916 aastal (sisuliselt õige ka täna) antud juhtimise definitsioonist tähendab juhtimine ette nägemist ja planeerimist, organiseerimist, korralduste andmist (uue ajal on muudetud see motiveerimiseks) ja kontrollimist.

Iga tarkvara arendusprotsess sisaldab endas ühel või teisel moel järgmisi tegevusi [KARLSTRÖM2002]:

- **Spetsifikatsiooni koostamine** – tehakse selgeks, mida tuleb teha;
- **Arendamine** – tarkvara realiseerimine;
- **Valideerimine** – kontrollitakse, kas rakendus teeb seda, mida algselt kavandati;
- **Muudatuste ja täienduste sisseviimine** – sisuliselt ülahoiu osa, mille käigus tarkvara pidevalt arendatakse vastavalt kliendi tekkivatele uutele vajadustele.

Viimati loetletud tegevused sobivad hästi kokku juhtimise definitsiooniga. Spetsifikatsioonide koostamine pole ju midagi muud, kui ettenägemine ja planeerimine. Arendamine sisaldab endas nii arendustegevuse organiseerimist kui ka arendajate motiveerimist. Valideerimine on tulemuste kontrollimine. Muudatuste ja täienduste sisseviimine on rakenduse praktilise kasutamise tulemusena ilmnevate vigade ja uute vajaduste spetsifitseerimine, arendamine ja kontrollimine.

### 1.2.2 Olulised ajalooetapid tarkvara arenduses.

Kronoloogiliselt on tarkvara arendamises eristatavad kolm etappi (vaata Joonis:2). Käesolevat aega iseloomustatakse kui üleminekut neljandasse etappi ([KARLSTRÖM2002], [BECK2000]).

1. periood	2. periood	3. periood	Täisküpsus	?
kirjuta ja paranda	struktuur-meetodid	protsessi küpsus	Paind-metoodikad	
~1970	~1990	~2000		

Joonis 2. Tarkvara arendusprotsessi etapid [KARLSTRÖM2002].

Esimeses etapis (kuni kuuekümnendate lõpu – seitsmekümnendate alguseni) olid domineerivad väikesemahulised rakendused ning programme kirjutati madalataseme keeltes. Rakenduste väikese mahu tõttu olid suurem osa tänapäeval tuntud arendusprobleemidest tundmata. Lihtsalt kirjutati koodi ja vajadusel parandati vigu. ([KARLSTRÖM2002]).

Teise etapi alguseks loetakse kuuekümnendate lõpul tekkinud kriisi tarkvaraarenduses. Valdav oli struktuurprogrammeerimine. Hakati arendama elementaarseid tarkvaraarendamise mudeleid ja neid ka praktikasse juurutama. Arendati välja koskmudel (Larman ja Basil [LARMAN&BASIL2003] viitavad Winston Royce artiklile [ROYCE1970]). Toodi sisse iteratsioonid (Larman ja Basil [LARMAN&BASIL2003] viitavad mitmele Winston Royce, Harlan Mills jt sellel ajal kirjutatud artiklitele).

Kolmandat etappi (üheksakümnendate aastate algus) seostatakse tarkvara arendusprotsesside täiustamisega ( *SPI - Software Process Improvement*). Loodi mitmeid tarkvara küpsuse ja kvaliteedi tagamise mudeleid. Üks tuntum ja ilmselt ka üks olulisem [KARLSTRÖM2002] on arendusprotsesside küpsusmudeli (CMM – Capability Maturity Model) [CMM] loomine CMU (*Carnegie Mellon University*) SEI (*Software Engineering Institute*) poolt. Loodi ka mitmeid sellekohaseid ISO

standardeid. Suurem osa kolmanda perioodi mudelitest on mõeldud suurtele tarkvarafirmadele [KARLSTRÖM2002]. Nendel mudelitel baseeruvaid arendusmetoodikaid on hakatud viimasel ajal kutsuma tardmetoodikateks (inglise keeles “*heavy methodology*”, “*monumental methodology*”).

Mis on neljanda etapi tunnuseks, on ehk veel raske ütelda. Hinnanguid on erinevaid. Ühelt poolt küpsusmudelite (ja nendel baseeruvate tardmetoodikate) areng ja teiselt poolt paindmetoodikate teke. Tom DeMarko oma eessõnas Kent Becki ja Martin Fowleri raamatule “*Planning Extreme Programming*” [BECK&FOWLER2001] kirjutab:

“XP on praegu kõige tähtsam suund meie erialal. Mulle tundub, et ta muutub tänasele arendajate generatsioonile sama oluliseks kui seda oli eelmisele generatsioonile SEI ja tema tarkvara küpsusmudel”.

### 1.2.3 Metodoloogia, metoodika ja meetod tarkvaraarenduses

Inglise keelses kirjanduses kohtume me mõistetega *heavy methodologies*, *light methodologies*, *agile methodologies*, *XP methodologies* jne. Kui vaadata Eesti Entsüklopeediat [EE6], siis leiame järgnevad sõnaseletused.

**Meetod** – kindlapiiriline praktilise tegevuse või tunnetuse organiseerimise viis.

**Metoodika** – õpetus mingi tegevusvaldkonna meetodeist; töö tegemise otstarbekate menetluste kogum.

**Metodoloogia** – filosoofiline õpetus tegelikkuse tunnetamise ja ümberkujundamise viisidest; maailmavaate printsiipide rakendus vaimses loomingus ja praktikas

Seega metodoloogia on mõtteviis, mis tarkvara arendamise seisukohalt oleks siis kas tard- või paindmetodoloogia.

Mõlemaid mõtteviise toetavad teatud tarkvara arendamise metoodikad. Nii on üks paindmetoodikaid *Extreme Programming* mis sisaldab endas teatud hulga praktilisi meetodeid (näiteks *Extreme Testing* ja paarisprogrammeerimine XP metoodikas).

Üldiselt, metoodika – kuidas me asju teeme.

Antud töös huvitab meid eelkõige paindmetodoloogia, XP-metoodika ja XP-metoodikas kasutatavad meetodid.

### 1.2.4 Tarkvara arendusprotsessi täiustamine.

Tarkvara arendusprotsessi täiustamine on tarkvarafirma tegevus eesmärgiga muuta firmas kasutatav arendusmetoodika ja arendusmeetodid firma vajadusi rahuldavaks.

Hea tarkvara arendamise metoodika on metoodika, mis nõuab võimalikult vähe ressursse ja võimaldab [BECK2000] :

1. Kliendile:

- maksimaalse tulemuse igast tema poolt tasutud arendustegevuse nädalast;

- teada plaanidest, plaanide muudatustest ja muudatuste põhjustest: mida, millal ning milliste ressurssidega tehakse;
- näha progressi väljatöötatava süsteemi arengus ja saada tõestus süsteemi töötamise ning kvaliteedi kohta;
- muuta oma arvamusi, soove ja prioriteete liigsete lisakuludeta;
- peatada suvalisel ajal süsteemi arendus ja omandada esialgse funktsionaalsusega ning antud hetkeks valminud kasutamiskõlblik osa tarkvarast.

## 2. Programmeerijale:

- teada kliendi tarkvaralisi soove ja prioriteete;
- teha kvaliteetset tööd ja omada selleks aega;
- saada abi nii kaastöötajatelt, juhtidelt, kui ka klientidelt;
- püstitada ja muuta tähtaegu;
- võtta ise endale kohustusi;

Selleks, et toota tarkvara hästi, peab tarkvara arendamisega tegeleva organisatsiooni struktuur ja selles organisatsioonis valitsev organisatsiooni kultuur arvestama programmeerijate ja klientide hirmudega ning võimaldama vastastikku teineteise õiguste ja kohustuste tunnustamist ning arvestamist [BECK2000].

### 1.2.5 Tarkvaraprojekti neli mõõdet.

Lähtuvalt Kent Beck töödest ([BECK1999], [BECK2000]) on tarkvaraprojektil neli mõõdet:

- **Kvaliteet** - kvaliteedi kriteeriumiks on kliendiga kooskõlastatud nõuete rahuldatus ja testide automatiseerimine. Kvaliteet on fikseeritud suurus ja järeleandmisi teha ei ole võimalik.
- **Maksumus** - raha saab kokku hoida õigete asjade (kliendile kasulik) õigesti tegemise (tippkvaliteet) tulemusena. Järeleandmisi on praktiliselt võimatu teha, sest ühe programmeerija ülalpidamiskulu (hind) on konstantsed suurused.
- **Funktsionaalsus** - kliendi valik, kus klient sõnastab vajadused ja määrab prioriteedid lähtudes oma (raha-) ja programmeerijate (aja-) ressurssidest.
- **Tähtajad** - programmeerija valik, kui kiiresti on võimalik. Aega saab kokku hoida õigete asjade (kliendile kasulik) õigesti tegemise (tippkvaliteet) tulemusena.

Kokkuvõttes tähendab see seda, et need neli muutujat ei ole sõltumatud muutujad. Seega korraga neid muutujaid ühes ja samas projektis fikseerida ei ole võimalik. Kvaliteet ja maksumus on suhteliselt fikseeritud suurused, milles järeleandmisi teha ei saa. Kehva kvaliteet ei rahulda klienti ja viib alla arendajate motivatsiooni teha head tööd. Arendajate ja firma ülalpidamiskulud on samuti suhteliselt konstantsed suurused.

Jäävad järele realiseeritavad funktsioonid ja tähtajad. Kent Beck väidab, et mõlema korraga fikseerimine ei ole võimalik ega arukas. Kui fikseeritakse funktsionaalsus,

siis peavad tähtajad jääma lahtiseks. Kuni funktsionaalsus pole korralikult realiseeritud on keeruline hinnata palju selleks tegelikult aega läheb.

Kui fikseeritakse tähtaeg, tuleb lahtiseks jätta realiseeritav funktsionaalsus ja alustada kõige olulisemate funktsioonide realiseerimisega.

### 1.2.6 Väike tarkvarafirma

Mark C. Paulk annab oma artiklis [PAULK2001] väikeste arendusprojektide definitsioonid (vt. Tabel.1).

„Väikese“ tüüp	Arendajate arv	Projekti pikkus
<b>Väike</b>	<b>3-5</b>	<b>6 kuud</b>
<b>Väga väike</b>	<b>2-3</b>	<b>4 kuud</b>
<b>Pisike</b>	<b>1-2</b>	<b>2 kuud</b>
<b>Individuaalne</b>	<b>1</b>	<b>1 nädal</b>
<b>Naeruväärne!</b>	<b>1</b>	<b>1 tund</b>

Tabel 1. „Väikese“ projekti definitsioon [PAULK2001].

Samas artiklis viitab ta Johnson ja Brodman artiklile [JOHNSON1998], kus väikeseks firmaks peetakse kuni 50 arendajaga firmat ja väikeseks projektiks kuni 20 arendajaga projekti.

Antud töös mõistetakse väikese arendusprojekti all 3-5 arendajaga 6 kuni 18 kuud kestvaid projekte.

## 1.3 Arendusprotsessi hindamise kriteeriumid

### 1.3.1 SW-CMM

SW-CMM (*Capability Maturity Model for Software*) on SEI (*Software Engineering Institute*) poolt loodud mudel selleks, et hinnata tarkvara arendamise ja haldamise kvaliteeti.

SEI on USA föderaalvalitsuse poolt finantseeritav ja Pittsburgis (Pennsylvania osariik) asuva *Carnegie Mellon University* poolt hallatav teadusasutus, mille uurimisobjektiks on tarkvara ja tarkvaraga kaasnev.

SW-CMM on kokku pandud mahuka uurimuse põhjal. On uuritud tarkvarafirmade kogemusi ja neid üldistatud.

Kenneth M. Dymondi [DYMOND1998] järgi tugineb SW-CMM neljale lihtsale kontseptsioonile:

- Iga tarkvarafirma areneb aja jooksul ja tema poolt kasutatav arendusmetoodika muutub pidevalt. On olemas võimalus süstemaatiliselt parandada ja täiendada tarkvara väljatöötamise ja haldamise meetodeid;
- Tarkvara väljatöötamise protsessi arengus on eristatavad erinevad küpsusastmed;

- Areng on pidev selles mõttes, et ühtegi taset pole võimalik vahele jätta. Enne kui kõik madalama taseme tegevused pole lihvitud, on järgmisele tasemele üleminek võimalik. Firma isiklik kogemus on peamine. Mehhaaniline reeglite täitmine ei aita;
- Entroopia seadused on üldkehtivad ja kehtivad ka tarkvara arendusprotsesside juures: iga asi liigub korrapäratuse poole, kui ei rakendata piisavaid jõupingutusi korra säilitamiseks ehk kui ei tehta piisavalt pingutusi tarkvaraprotsessi küpsuse parandamiseks, siis asjad muutuvad halvemaks.

Ilmselt on mõistlik siinkohal rõhutada, et SW-CMM ei räägi sõnekestki, kuidas tuleb teha. SW-CMM asetab rõhu küsimusele “mida tuleb teha”. See “kuidas” on aga iga firma enda probleem ja sõltub firma teadmistest ja kogemustest.

### 1.3.2 CMMI

CMMI (*Capability Maturity Model Integration*) on SEI (*Software Engineering Institute*) poolt loodud küpsusmodelite edasiarendus ja integratsioon. CMMI püüab integreerida olemasolevad küpsusmodelid (SW-CMM® - *Capability Maturity Model for Software*, SECM - *Systems Engineering Capability Model* ning IPD-CMM - *Integrated Product Development Capability Maturity Model*) ühtseks tervikuks.

Nagu kõik küpsusmodelid tugineb CMMI [CMMI] suure hulga firmade kogemuste üldistamisele. Võrreldes varasemate küpsusmodelitega pakub CMMI järgnevat:

- Rohkem kooskõla äri vajadustega;
- Suuremat rõhku kasutaja vajaduste rahuldamisele;
- Ühendada erinevaid kogemusi (mõõtmistes, riski juhtimises, varustamises, ..) ühtseks tervikuks;
- Suuremat küpsust protsessidele;
- Rohkem kooskõla ISO standarditega.

Tarkvara tootmise seisukohalt (ja silmas pidades väikefirmade vajadusi) on oluline eelkõige 643 lehekülge pikk dokument “CMMI for Software Engineering” [CMMI-SW].

### 1.3.3 ISO 12207

Infotehnoloogia standard ISO 12207 – “Tarkvara elutsükli protsessid” annab tervikliku ja kõikehõlmava käsitluse kogu tarkvara elutsüklist.

Standard defineerib tarkvara elutsükli terminoloogiat ja on kasutatav tarkvaraga seotud protsesside kirjeldamisel ning nende organiseerimisel.

Standardis on kirjeldatud [TEPANDI2003] :

- 5 primaarprotsessi: hankimine, tarnimine, arendus, ekspluatatsioon ja hooldus;
- 8 abiprotsessi: dokumenteerimine, konfiguratsiooni haldus, kvaliteedi tagamine, nõuetele vastavuse tõendamine (*verification*), nõuetele vastavuse väljaselgitamine (*validation*), koostoime analüüs (*joint review*), auditeerimine, probleemilahendus (*problem solving*);
- 4 organisatsioonilist protsessi: haldus, infrastruktuur, täiustamine ja koolitus.

Hankimise protsess koosneb standardi ISO 12207 järgi algatamisest, pakkumiskutse koostamisest, lepingu sõlmimisest ja uuendamisest, tarnija seirest, vastuvõtmisest ning lõpetamisest.

Tarnimise protsess koosneb algatamisest, vastuse koostamisest, lepingust, plaanimisest, täitmisest ja juhtimisest, läbivaatlusest ja hindamisest, üleandmisest ning lõpuleviimisest.

Arendusprotsess koosneb vastavalt protsessi teostamisest, süsteeminõuete analüüsist, süsteemi arhitektuuri projekteerimisest, tarkvaranõuete analüüsist, tarkvara arhitektuuri projekteerimisest, tarkvara detailprojekteerimisest, tarkvara programmeerimisest ja testimisest, tarkvara integreerimisest, tarkvara kvalifitseerimistestimisest, süsteemi integreerimisest, süsteemi kvalifitseerimistestimisest, tarkvara installeerimisest ning tarkvara vastuvõtmise toetamisest.

Ekspluatatsioon koosneb protsessi teostamisest, katseekspluatatsioonist, süsteemi ekspluatatsioonist, kasutaja toetamisest.

Hoolduse osad on protsessi teostamine, probleemide ja muutmise analüüs, muudatuste teostamine, hoolduse läbivaatus ja kinnitamine, migratsioon, tarkvara mahavõtt.

Kvaliteedi tagamine koosneb protsessi teostamisest, toodete tagamisest ning protsesside tagamisest.

Lõpetuseks olgu öeldud, et antud standard on tõlgitud ka eesti keelde ja vastu võetud Eesti Vabariigi standardina (EVS-ISO / IEC 12207).

#### **1.3.4 ISO 15504**

Infotehnoloogia standard ISO 15504 – “Tarkvara protsessi hindamine” on raamistik tarkvara protsesside (soetamine, arendus, tugi,...) hindamiseks. Sama standard on tuntud ka SPICE [SPICE] nime all. Sarnaselt SW-CMM nõuetele on ka siin määratud skaala, mis mõõdab suutlikkust ehk võimekust (ing. k. *capability*). Standardid ISO 12207 ja 15504 on heas kooskõlas.

#### **1.3.5 Miks antud töös on lähtutud CMM nõuetest?**

Kui lähtuda võrdlevatest töödest, kus on analüüsitud omavahel ISO 15504 ja SW-CMM nõudeid [PAULK1999] või ka kõiki (ISO 12207, 15504 ja SW-CMM) omavahel [TLSM1998] - selgub, et pole vahet, millist nendest tarkvara arendusprotsessi täiustamisel aluseks võtta. Nende ühisosa on väga suur ja erinevused marginaalsed [TLSM1998]. USA ja India lähtuvad rohkem CMM-st, Euroopa aga rohkem ISO-st.

Samas on täheldatav tendents nende omavahelises lähenemises ja kooskõlla viimises. CMMI lubab seda oma koduleheküljel [CMMI], SPICE omal [SPICE].

Antud töös on lähtutud SW-CMM teise taseme nõuetest. Valiku esimeseks, kui mitte peamiseks põhjuseks on antud dokumendi lühidus ja selgus. SW-CMM on kokku



umbes 500 lehekülge. Sellest väikefirmale esialgselt arendusprotsessi täiustamiseks piisav teine tase aga ainult umbes 100 lehekülge.

Valiku teiseks, kuid peamiseks, põhjuseks on SW-CMM enda sisemine loogika (vt. antud töö osa 1.3.1 - viide Kenneth M. Dymondi'le ning osa 2.1), mis võimaldab (ja samas ka eeldab) tarkvara arendusprotsessi pidevat täiustamist lähtuvalt firma kogemustest.

Kolmandaks põhjuseks on suure hulga tarkvarafirmade töötamine (vaata 2.1.1) SW-CMM esimesel tasemel. Seega SW-CMM teise taseme nõuded ja sellele tasemele jõudmine võib huvi pakkuda paljudele arendusfirmadele.

Viimaseks (mitte küll teaduslikult põhjendatud) kriteeriumiks on isiklik sümpaatia antud dokumendi suhtes, mis on tingitud tema lühidusest, selgusest ja sisemisest loogikast.

## 1.4 Arendusmetoodikad

Toetudes *Craig Larman'i ja Victor R. Baily* artiklile [LARMAN&BASIL2003] on tarkvara arendusmetoodikaid pidevalt uuritud, parandatud, täiendatud. Tarkvara väljatöötamine on aga endiselt väga riskantne valdkond. Väga suur osa alustatud projektidest ei saa kunagi valmis. Väga suur osa valminud projektidest on nõudnud ressursse oluliselt rohkem, kui esialgselt planeeritud [NORMAK2001].

Seoses tarkvara arendustegevuse massilise jõudmisega äriprotsessidesse, on nõuded tarkvaraarendusele muutunud. Ei ole enam võimalik fikseerida tarkvarale esitatavaid nõudeid ainult arendustegevuse esimeses, planeerimise etapis. Seega klassikaline arendustegevuse koskmudel enam ei tööta. Nõudeid ja koos sellega ka tarkvara ennast peab olema võimalik muuta kogu arendustegevuse käigus.

Arendusmetoodikad on jäävalt "kuum" teema. 2001.aasta veebruaris löid 17 tarkvara arendustegevust uurivat teadlast, praktikut ja vastavateemaliste raamatute autorit USA-s ühenduse "*The Agile Alliance*" (<http://www.agilealliance.org>). Ühenduse eesmärk on välja töötada paremad ja lihtsamad metoodikad tarkvara arendamiseks. Ajakirja "*Upgrade*" 2002. aasta aprilli number <http://www.upgrade-cepis.org/issues/2002/2/upgrade-vIII-2.html> oli pühendatud täielikult ühe tuntuma arendusmetoodika (*Extreme Programming*) tutvustamisele. *IEEE Computer Society* poolt väljaantava ajakirja "*Computer*" 2003. aasta juuni number oli pühendatud paindmetoodikatele. Ka ajakirja "*Upgrade*" värske number (August 2003) <http://www.upgrade-cepis.org/issues/2003/4/upgrade-vIV-4.html> on pühendatud tarkvara arendamisele.

Erialases kirjanduses jagatakse tarkvara arendusmetoodikaid kahte suurde gruppi:

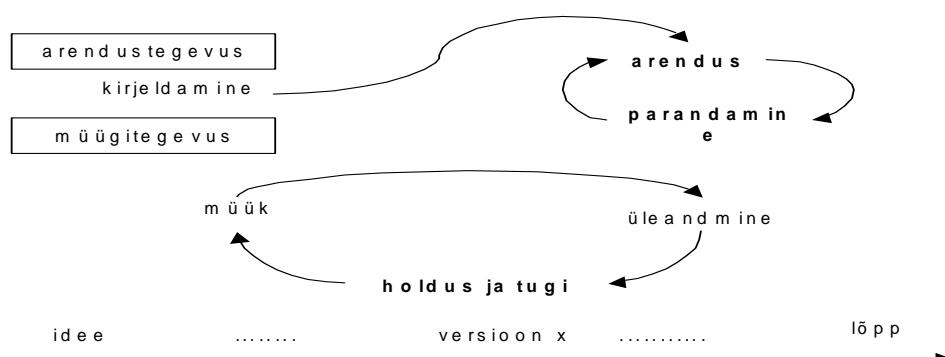
Tardmetoodikad ja paindmetoodikad (vt. alajaotus 1.2.3). Käesolevas osas on toodud tard- ja paindmetoodikate tunnused ja põhilised erinevused nende vahel. Antud osas olen tuginenud Karol Frühauf [FRÜHAUF2001] tööle.

### 1.4.1 Paind- ja tardmetoodikad.

Tarkvara kui produkti eluiga (vt.Joonis.3) on sarnane nii tard-, kui ka paindmetoodikate korral. Asi hakkab ideest, siis see idee teostatakse ja paisatakse turule toote esimene versioon. Sellele järgnevad versiooni parandused, uuendused ja muudatused, kuni ükskord pole toote edasiarendus enam otstarbekas ja asi tuleb lõpetada.

Eluea sees toimub paralleelselt kaks protsessi:

1. **arendustegevus**, kus seda ideed edasi arendatakse ja selle realiseerimist täiustatakse
2. **müügitegevus**, kus seda toodet müüakse, antakse ostjale üle ja pakutakse ostjale nii hooldust kui kasutaja tuge.



Joonis 3. Produkti eluiga [FRÜHAUF2001]

Tabelis 2 on toodud põhilised erinevused paind- ja tardmetoodikate vahel. Neid erinevusi võib kokkuvõtlikult sõnastada järgnevalt : kui tardmetoodikad rõhutavad asjade õigesti tegemist siis paindmetoodikad rõhutavad õigete asjade tegemist.

Tabelis 3 on toodud peamised rõhuasetused tard- ja paindmetoodikate puhul. Kui tardmetoodikad asetavad rõhu protsessidele ja plaanidele, siis paindmetoodikad rõhutavad inimest ja muutuvaid olukordi.

Tabelis 4 on toodud paind- ja tardmetoodikatele omased tunnused .

Paindmetoodikad	Tardmetoodikad
Kaks projekti pole kunagi sarnased	Projektide struktuurid on sarnased ja projekti läbiviimist on võimalik standardiseerida.
Kliendi osalemine projektis on hädavajalik	Kliendi osalemine projektis on ebatõenäoline. Klient esitab projekti algul fikseeritud tingimused/nõudmised ja võtab projekti lõppedes projekti tulemi vastu.
Enne projekti algust fikseeritakse nõuded üldjoontes. Parasjagu töös oleva ülesande nõuded täpsustatakse.	Projektile esitatavad kõik nõuded peavad olema täpselt ja peensusteni kirjeldatud ning fikseeritud enne

	projekti algust.
Disainitakse ainult parasjagu töös olevat ülesannet. Homsete ülesannetega suurt pead ei vaevata. Vajaduse korral muudetakse disaini töö käigus pidevalt.	Rakenduse täpne arhitektuur fikseeritakse enne disainimise ja kodeerimise algust.
Klient saab keskmiselt iga 6 nädala tagant midagi temale vajalikku ja reaalselt kasutatavat.	Enne, kui klient saab midagi kasutatavat näha, kulub palju aega.
“Proovi” ja “proovi uuesti” on peamised kriteeriumid	Asju tuleb õigesti teha esimese korraga.

Tabel 2. *Paind- ja tardmetodoloogia erinevused [FRÜHAUF2001]*

<b>Paindmetoodikad</b>	<b>Tardmetoodikad</b>
Põhimõtted	Protsessi mudel ja kirjeldus
Tegelikud vajadused	Projekti plaan
Inimeste oskused	Protsessi tulemus

Tabel 3. *Paind- ja tardmetodoloogia rõhuasetused [FRÜHAUF2001]*

	<b>Paindmetoodika</b>	<b>Tardmetoodika</b>
<b>Teadmised</b>	Määramata, on inimestes	Fikseeritud, on protsessis
<b>Kommunikatsioon</b>	Inimene inimesega	Dokumentatsioon
<b>Koostöö</b>	Tihe, pidev	Rohkem või vähem tavaline
<b>Suhtumine muudatustesse</b>	Muudatused on täiesti tavaline nähtus	Muudatusi ei taluta
<b>Suhtumine riski</b>	Kohanemine riskidega	Riske üritatakse ennetada ja neid vältida
<b>Teostamine</b>	Tehakse ainult niipalju, kui on momendil hädavajalik	Üritatakse realiseerida maksimum vajadusi silmas pidades.
<b>Distsipliin</b>	Informaalne ja kõrge	Formaalne ja madal

Tabel 4. *Paind- ja tardmetoodikate omadused [FRÜHAUF2001]*

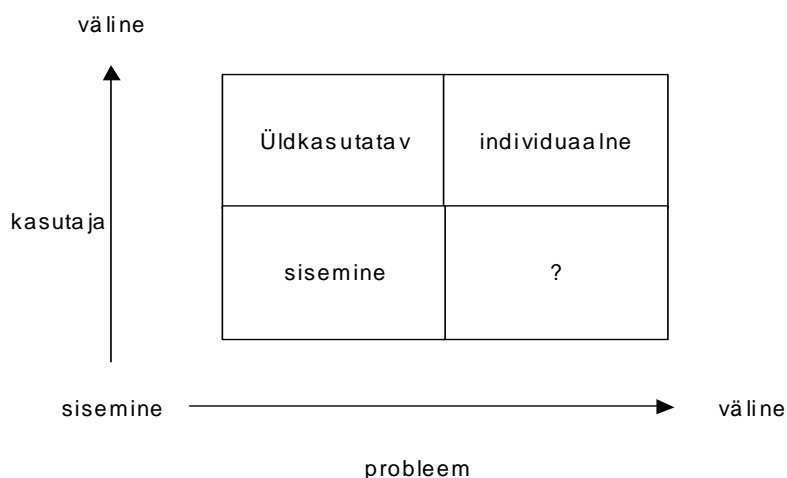
### 1.4.2 Paind- ja tardmetoodikate sobivus tarkvara arendamiseks

Tarkvara liigitamiseks on pakutud välja kahte lähtekohta [FRÜHAUF2001]:

#### 1. Kasutamise kohast lähtuv tarkvara liigitamine:

- **Sisseehitatud tarkvara** - tarkvara on ehitatud mingi riistvara sisse;
- **Informatsioonisüsteemid** - interaktiivsed, andmebaasi rakendused;
- **Süsteemi tarkvara** - teeb mingi riistvara kasutamise lihtsamaks
- **Rakendused** - mingid suvalised eraldiseisvad tarkvara paketid, mis midagi teevad
- **E-rakendused** – teevad midagi enamat, kui näitavad kodulehekülge

2. **Tarnija** seisukohalt saab tarkvara liigitada omatarbeks tehtud (sisemine), individuaalkasutajale tehtud (individuaalne), või turu vajadustest lähtuvalt tehtud (üldkasutatav) tarkvara (vaata joonis 4).



Joonis 4. Tarkvara liigitus kasutaja ja probleemi järgi [FRÜHAUF2001]

Tabel 5 illustreerib paind- ja tardmetoodikate sobivust erinevat liiki tarkvara arendamisel.

	Sisemine	Individuaalne	Üldkasutatav
<b>Sisseehitatud</b>	Monumentaalne	Monumentaalne	Monumentaalne
<b>Infosüsteem</b>	Agiilne	?	Monumentaalne
<b>Süsteemitarkvara</b>			Monumentaalne
<b>Rakendus</b>	Agiilne	?	?
<b>E-rakendus</b>	Agiilne	?	?

Tabel 5. Metoodikate sobivus tarkvara arendamisel [FRÜHAUF2001]

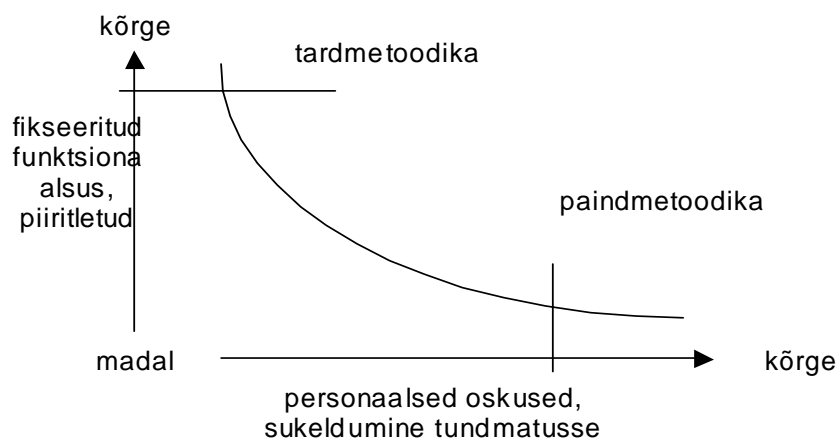
Süsteemitarkvara sisemiseks ja individuaalseks kasutamiseks valmistamine on nonsenss.

“?” - tähendab, et antud tarkvara valmistamisel võib olla edukas nii üks kui teine metodoloogia. Kui tarkvarale esitatud nõuded on täielikult ja lõplikult fikseeritud, ning kui on piisavalt ressursse, siis võib kasutada tardmetoodikat.

Alljärgnevalt on loetletud võtmeküsimused paind- või tardmetoodika valikus:

- Kuivõrd dokumenteeritud peab protsess olema?
- Millised on nõuded turvalisusele ja töökindlusele?
- Kas produkti üleandmine kliendile on ühekordne või pidev tegevus?
- Kas on ette näha produktile esitatavate nõuete pidevat muutumist?
- Kas probleem on lõpuni mõistetav ja arusaadav?
- Kui innovaatiline on produkt?
- Kui kiiresti valitud tehnoloogia areneb?

Joonisel 5 on toodud seos metoodika ja lahendatava ülesande eripärade vahel.



Joonis 5. Seos metoodika ja lahendatava ülesande eripära vahel [FRÜHAUF2001]

### 1.4.3 Tuntumad väikefirmadele sobivad paindmetoodikad

#### 1.4.3.1 XP – Extreme Programming

XP-metoodika on loodud Kent Beck [BECK2000] poolt. XP- metoodika alustaladeks [FOWLER2000] on inimestevaheline suhtlemine, tagasiside rakenduse kasutajalt, lihtsus ning arendajate südikus (*courage*). XP-metoodika sisaldab endas umbes tosinat meetodit (mõningad allikad toovad välja ligi kakskümmend meetodit [XP]) millest olulisemad on testidel tuginev arendus (*Test Driven Development*) [BECK2003], lühiajaline planeerimine ja plaanide pidev korrigeerimine [BECK&FOWLER2001] ja rakenduse tulevase (või potentsiaalse) kasutaja osalemine arendustegevuses.

#### 1.4.3.2 SCRUM

SCRUM-metoodika peamiseks eesmärgiks on kasutaja nõuete muutumisest tekkivate negatiivsete tagajärgede minimiseerimine 30 päevaste iteratsioonitsükli abil. Selliseid tsikleid nimetatakse “sprintideks”. Siinkohal olgu märgitud termini *sprint* üks tähendus: sprint on suhteliselt lühiajaline intensiivne arendustsükkel, kus koos töötavad tavaliselt eraldi rühmadesse kuuluvad arendajad.

SCRUM-metoodika on väga sarnane teistele paindmetoodikatele ja eriti XP-metoodikale. Ka siin on oluline arendajate vaheline kommunikatsioon (igapäevased arendajate koosolekud – *Scrum Meetings*) ning kasutaja käest saadav tagasiside (“sprint” lõpeb töö demonstreerimisega kasutajale – *Scrum Demo*).

Erinevalt XP-metoodikast, et käsitle SCRUM-metoodika testimist ja koodi kirjutamist. Seetõttu peaks SCRUM-metoodika sobima hästi kokku XP-metoodika koodi kirjutamise tegevustega [FOWLER2000].

Lähemalt vaata SCRUM-metoodika kohta artikleid [SUTHERLAND2002], [BEEDLE] ning [SCRUM].

#### 1.4.3.3 Cockburni *Crystal-pere* metoodikad

Metoodikate autor Alistair Cockburn usub, et erinevad projektid vajavad erinevaid metoodikaid ning sellepärast on tegemist mitte ühe metoodikaga, vaid metoodikate perega [FOWLER2000]. Oluline tegur metoodikate valikuks on arendajate arv projektis ja projekti vigade hind (inimelud, oluline rahaline väärtus, mitteoluline rahaline väärtus, mugavus) [LEIS2001].

Ka Crystal-metoodikad on inimkesksed, kuid seda natuke teisel moel, kui XP-metoodika: Cockburni arvates ei suuda loovisiksused (keda tarkvara arendajad kindlasti on) eriti alluda distsipliinile. Kui XP-metoodika on suhteliselt kõrge distsipliiniga metoodika, siis Crystal-metoodikad üritavad minimiseerida distsipliini tasemele, kus tarkvaraarendus on veel edukas [FOWLER2000].

Ka ei nõua Crystal-metoodikad sellist intensiivsust nagu seda nõuab XP-metoodika [FOWLER2000], mistõttu on antud metoodikad Cockburni arvates suupärasemad suuremale hulgale arendajatele.

Lähemalt vaata Crystal-metoodikate kohta Alistair Cockburni koduleheküljelt [COCKBURN].

#### 1.4.3.4 Adaptive Software Development

Jim Highsmith, oma raamatus, *Adaptive Software Development* (Dorset House, 1999), rakendab tarkvaraarenduses keerulistes kohanduvates süsteemides (*complex adaptive systems*) kasutatavat kaose teooriat [FOWLER2000].

ASD-metoodika ei anna mingeid praktilisi nõuandeid, nagu seda teeb XP-metoodika. ASD on pigem filosoofiline arusaam. ASD aluseks on kolm faasi: oletus (*speculation*), koostöö (*collaboration*) ja õppimine (*learning*) [LEIS2001].

Traditsiooniliselt on plaanist kõrvalekalded vead, mida on vaja paranda. Highsmith käsitleb planeerimist kohandavas keskkonnas mitteprognoositavana. Sellises määramatus keskkonnas on toimetulekuks vajalik inimestevaheline koostöö [LEIS2001].

2001 aasta veebruaris teatati ASD- ja Crystal-metoodikate ühendamisest.

#### 1.4.3.5 Lean Development

*Lean Development* (LD) metoodika on loodud Bob Charette'i poolt. Tema põhiprintsiipideks on kliendi vajaduste rahuldamine (kõrgeim prioriteet). Analoogiliselt XP-metoodikale on ka LD-metoodikas oluline kliendi osalemine tarkvaraarenduses. Rõhutatakse minimaalsust, meeskonnatööd, paindlikkust kliendi soovide realiseerimisel [LEIS2001].

LD-metoodika seitse rõhuasetust [POPPENDIECK2003]:

- Elimineeri mitteoluline
- Suurenda õppimisvõimet
- Otsusta nii hilja kui võimalik

- Tarni nii ruttu, kui võimalik
- Usalda meeskonda
- Arenda tervikut
- Näe tervikut

Lähemalt LD-metoodikast vaata [POPPENDIECK2003].

#### 1.4.3.6 Feature Driven Development

FDD- metoodika on loodud OO-guru Peter Coad'i poolt [FOWLER2000]. Paul Leis lisab FDD autorite hulka ka Jeff De Luca [LEIS2001]. Ka FDD nurgakiviks on lühikesed (antud metoodikas kahenädalased) iteratsioonid. FDD kirjeldab viit protsessi. Kolm protsessi alustavad projekti ( üldmudeli (*Overall Model*) loomine, omaduste/atribuutide nimekirja (*Features List*) koostamine ning planeerimine vastavalt koostatud omaduste nimekirjale). Ülejäänud kaks protsessi (kavandamine vastavalt omaduste nimekirjale ning teostamine vastavalt omadustele) täidetakse aga igas iteratsioonis ([FOWLER2000], [LEIS2001]).

FDD-metoodikas on kahte liiki arendajaid: klasside omanikud ja peaprogrammeerijad (vilunud arendajad, arendusprotsessi koordinaator/projektijuht). Peaprogrammeerija üldjuhul koodi ei kirjuta. Peaprogrammeerija ülesanne on realiseerida ärinõudeid. Ülesande täitmiseks moodustab peaprogrammeerija nõude-meeskonna. Nõude-meeskond koosneb klasside omanikest, kes programmeerivad vajalikud klassid [LEIS2001].

Lähemalt vaata FDD-metoodikast vastavalt koduleheküljelt [FDD].

#### 1.4.3.7 DSDM (Dynamic System Development Method)

DSDM-metoodikale pandi alus 1994. aastal Inglismaal 17 asutajafirma poolt. Selle metoodika arendamist korraldab eriorganisatsioon, mis on välja andnud hulgaliselt metodoloogilisi materjale, korraldab väljaõpet ning sertifitseerimist.

DSDM-metoodika esimeseks etapiks on rakendatavuse analüüs, mis seisneb :

- äriprotsessist arusaamisest (lühikesed tööseminarid)
- loodava süsteemi arhitektuuri ja projektiplaani koostamisest

Edasi järgnevad kolm seotud tsüklit :

- funktsionaalse mudeli tsüklis luuakse analüüsidokumentatsioon ja prototüübid;
- kavandamise ja kodeerimise tsüklis luuakse kasutatav süsteem;
- evitamise tsüklil tagab süsteemi kasutusse võtmise.

DSDM-metoodikas on analoogiliselt XP-metoodikale oluline kõrge kvaliteet, kasutaja muutuvate nõudmiste rahuldamine, kasutaja aktiivne osalemine, sagedased üleandmised ja testimise. Nagu paindmetoodikates ikka, on kas siin iteratsioonid lühikesed [LEIS2001].

Lähemalt vaata DSDM-metoodikast vastavalt koduleheküljelt [DSDM]

#### 1.4.3.8 Unified Process (UP) kui paindmetoodika

Tegemist on Craig Larman'i raamatul "Applying UML and Patterns" [LARMAN1998] tugineva tarkvara arendamise metoodikaga [LEIS2001].

UP-metoodika põhiidee peitub arendusfaasides, iteratsioonides ning arendusprotsessi käigus loodavates mudelites.

UP-metoodika kirjeldab järgmisi arendusfaase [LARMAN1998]:

- Planeerimine ja täpsustamine
- Analüüs
- Disain
- Konstrueerimine
- Testimine

Planeerimise ja täpsustamise faasiga algab iga projekt. Analüüsi, disainimise ja konstrueerimise ja testimise faasi korratakse aga igas iteratsioonis.

UP-metoodika näeb ette mitmesuguste mudelite koostamist millest olulisemad on [LARMAN1998] :

- Use Case mudelid (*Use Cases*)
- Staatilised struktuuri diagrammid (*Static Structure Diagrams*)
- Järgnevuse diagrammid (*Sequence Diagrams*)
- Koostöö diagrammid (*Collaboration Diagrams*)
- Klasside diagrammid (*Class Diagrams*)

Paul Leis [LEIS2001] peab antud metoodikat Eesti tingimustes (ja mitte ainult) üheks perspektiivikamaks.



## 2 SW-CMM TEINE TASE JA XP-METOODIKA

Minu esimene kokkupuude SW-CMM-iga oli põgus ja siis jäi sellest mulje kui mingist suurtele ja väga suurtele tarkvarafirmadele mõeldud metoodikast või standardist.

Täna, peale originaaltekstidega tutvumist võin tuginedes Mark C. Paulk artiklile “Using CMM in small organizations” [PAULK1998] julgelt väita, et SW-CMM on oluline ka väikefirmadele .

SW-CMM on kogum tegevusi, mida tarkvarafirmad järgivad, või mida nad võiksid järgida selleks, et tulemus, arendatav tarkvara, rahuldaks paremini ja täielikumalt kliendi vajadusi.

SW-CMM rakendamine ei sõltu firma suurusest. Küll aga sõltub firma suurusest, võimalustest ja vajadustest see, kuidas neid SW-CMM poolt kirja pandud tegevusi ellu viiakse.

SW-CMM on piisavalt hea selleks, et endale selgeks teha arendusprotsess kui selline (ainult 500 lehekülge) ja ta on ka piisavalt paindlik, et temas sisalduvaid nõuandeid järkjärgult firmas juurutada

Antud osas püüan näidata, et XP-metoodikat korrektselt järgides on firmal lootus töötada vähemalt SW-CMM teisel tasemel.

### 2.1 SW-CMM tasemed.

SW-CMM järgi töötavad tarkvarafirmad erinevatel tasemetel. Kokku on kirjeldatud viit taset.

#### 2.1.1 Esimene tase

SW-CMM esimest taset nimetatakse *Initial* (algne, algeline) tasemeks ja seda protsessi iseloomustavad sõnad “ettearvamatu” ja “kontrollimatu”. Praktikas tähendab see seda, et osa projekte õnnestub väga hästi ja osad lähevad täielikult untsu. Ja keegi ei tea, miks.

Suurem osa tarkvarafirmasid (hinnanguliselt umbes 75 %) töötab SW-CMM skaala esimesel tasemel. Seega suurem osa maailmas loodavast tarkvarast luuakse firmade poolt, kes töötavad SW-CMM esimesel tasemel [DYMOND1998].

Ei ole korrektne ütelda, et loodav tarkvara oleks sellepärast kehva või et töötades SW-CMM esimesel tasemel pole võimalik head tarkvara luua.

Küll aga väidab SW-CMM, et sellisel tasemel loodud tarkvara korral ei ole ressurssidega efektiivselt ümber käidud. Samas ei välistata ka võimalust, et teatud tingimustel (näiteks lähteülesande väga kiire muutumise tingimustes) võib töötamine sellel tasemel olla ainuõige [DYMOND1998].

Selleks, et töötada SW-CMM esimesel tasemel, ei pea tarkvarafirma tegema mitte mingisuguseid jõupingutusi ega omama mitte mingisuguseid kogemusi.

Samas alustavad kõik tarkvarafirmad oma tööd just nimelt sellel tasemel. Ja ainult kogemus ning järjepidev töö tarkvaraprotsessi arendamisel võimaldab firmal liikuda järgmisele tasemele [DYMOND1998].

### 2.1.2 Teine tase

SW-CMM skaala teisel tasemel töötab hinnanguliselt umbes 15% tarkvarafirmadest maailmas [DYMOND1998].

Seda taset kutsutakse **Repeatable** (korratav). Iseloomulik on see, et antud tasemel töötavad firmad suudavad oma eelnevaid kogemusi rakendada uutes tarkvaraprojektides.

Sellel tasemel töötavad firmad suudavad juhtida (kavandada, korraldada ja kontrollida) järgmisi tarkvara arendustegevuse alamprotsesse [CMM]:

- **Nõuete haldamine** - suudetakse kavandada, korraldada ja kontrollida tarkvara lähteülesandest tulenevaid nõudmisi ja nõudmiste muudatusi;
- **Tarkvaraprojekti planeerimine** - vastavalt lähteülesannetele suudetakse kavandada, korraldada ja kontrollida arendustegevuse planeerimist ning sellega kaasnevat;
- **Arendusprotsessi monitooring** - suudetakse kavandada, korraldada ja kontrollida tehtud plaanide igapäevast täitmist;
- **Kvaliteedi tagamine** – suudetakse kavandada, korraldada ja kontrollida loodava tarkvara kvaliteeti;
- **Konfiguratsioonide haldamine** – suudetakse kavandada, korraldada ja kontrollida tarkvara konfiguratsioone;
- **Allhangete juhtimine** – suudetakse kavandada, korraldada ja kontrollida teiste firmade poolt teostatavaid allhankeid.

Seega iseloomustab SW-CMM teisel tasemel töötavaid firmasid eelkõige juhtimise ja distsipliini sisseviimine arendusprotsessi. Selle tulemusena on edukad tarkvara projektid (maksumuse, tähtaegade ja kliendi nõuete rahuldamise mõistes) sellel tasemel töötavate firmade jaoks tavaline ja normaalne.

Sellel tasemel saab töötada tarkvarafirma, kellel on piisavalt teadmisi tarkvara protsessi juhtimisest, piisavalt kogemusi projektide läbiviimisel ja ka piisavalt negatiivseid kogemusi selleks, et mitte laskuda tagasi esimesele tasemele.

SW-CMM teise taseme põhjalikum ülevaade on toodud antud töö lisas 1.

### 2.1.3 Kolmas tase

Kolmandat taset kutsutakse **Defined** (määratud) tasemeks ja sellel tasemel töötavates firmades on tarkvara protsess määratud ja defineeritud organisatsiooni tasemel [DYMOND1998].

Lisaks teise taseme nõuetele on kolmandal tasemel töötamiseks olulised järgmised nõuded [CMM]:

### **1. Tarkvara arendusprotsessi juhtimine organisatsiooni tasandil:**

- Tarkvara protsessi arendamine ja täiustamine on koordineeritud organisatsiooni tasandil;
- On olemas organisatsiooni standard, mida jälgitakse ja millega iga konkreetse arendusprojekti tugevust või nõrkust hinnatakse;
- Organisatsiooni tasandil on planeeritud tarkvara arendusprotsessi arendamine ning täiendamine.

### **2. Organisatsioonisisese arendusprotsessi standardi olemasolu:**

- Organisatsioonil on olemas arendusprotsessi standard ja selle standardiga pidevalt tegeletakse
- Standardi kasutamisega seotud informatsiooni kogutakse, töödeldakse ja nii töödeldud kui töötlemata informatsioon on kättesaadav vajalikele inimestele.

### **3. Pidev töötajate koolitus:**

- Koolitustegevus on planeeritud lähtudes tarkvara protsessi juhtimise ja töötajate tehniliste teadmiste vajadustest;
- Koolitustegevust viiakse vastavalt sellele plaanile ka reaalselt läbi;
- Töötajad saavad koolitust lähtuvalt nende esitatavatest nõuetest

### **4. Integreeritud tarkvara juhtimine**

- Iga konkreetse tarkvaraprojekti jaoks luuakse just seda projekti rahuldav arendusprotsess, lähtudes üldisest organisatsiooni arendusprotsessi standardist
- Igat konkreetset projekti planeeritakse ja juhitakse vastavalt sellele projektile kohandatud nõuetest

### **5. Tarkvaratehnika**

- Tarkvaratehnika on määratud, ühtne ja kasutatakse järjekindlalt kõikides tarkvaraprojektides;
- Tarkvara planeerimise, analüüsi, disainimise, realiseerimise ja testimise tööd on omavahel kooskõlastatud.

### **6. Tarkvara tootmisgruppide vaheline koostöö**

- Tarkvarale esitatavad nõuded (kliendi vajadused) on kooskõlastatud; kõikide asjasse pühendatud osapooltega;
- Gruppide kohustused on omavahel kokku lepitud ja kõikidele teada
- Projekteerimisega tegelevad grupid määravad, kontrollivad ja lahendavad gruppidevahelisi seoseid

### **7. Tagasiside ja kontroll**

- Tagasiside ja kontrolli mehhanismid on planeeritud;
- Vead igas etapis püütakse kinni ja parandatakse.

SW-CMM skaala kolmandal tasemel töötab hinnanguliselt umbes 10% tarkvarafirmadest maailmas [DYMOND1998].

#### 2.1.4 Neljas ja viies tase

SW-CMM neljandal ja viiendal tasemel töötab maailmas 2002 oktoobri seisuga kokku ainult 146 firmat [HIGHMATORGS]. Sellest 72 firmat töötab neljandal tasemel ja 74 firmat viiendal tasemel. Absoluutne ülekaal on India tarkvarafirmade käes, kus neljandal tasemel töötab 27 firmat ja viiendal tasemel 50 firmat. Kui lisada siia veel USA firmad vastavalt 39 (4.tase) ja 20 (5.tase), siis ega teistele suurt midagi ei jäägi.

Lisaks SW-CMM teise ja kolmanda taseme nõuete täitmisele kuuluvad neljandale ehk **Managed** (juhitud) taseme juurde järgmiste alamprotsesside juhtimine:

1. Tarkvara kvaliteedi täielik juhtimine
2. Tarkvara arendusprotsessi kvantitatiivne juhtimine

Öeldakse ka, et neljandal tasemel töötades on tarkvara arendusprotsess kvantitatiivselt mõistetav ja kontrollitav [DYMOND1998].

SW-CMM viiendal ehk **Optimizing** (optimeeritud) tasemel lisandub kõigele eelnevale ka tarkvara arendusprotsessi pidev täiustamine, mis tagatakse alljärgnevate alamprotsesside juhtimisega:

1. Tehnoloogia muudatuste juhtimine
2. Arendusprotsessi muudatuste juhtimine

Neljandast ja viiendast tasemest ei ole mõtet enne isegi mitte rääkida, kui pole vähemalt SW-CMM kolmandal tasemel töötamise kogemust. Seepärast on käesolevas töös vaatluse all eelkõike SW-CMM teine tase ja sellest tulenev.

## 2.2 SW-CMM struktuur

SW-CMM defineerib järgmised mõisted:

- 5 erinevat küpsuse taset (*Levels*)
- 18 erinevat arendustegevuse protsessi (*Key Process Area*)
- 52 eesmärki (*Goals*)
- 316 tegevust (*Practices*)

Defineeritud 18 protsessi on jagatud tasemete vahel nii, et teisele tasemele kuulub 6 protsessi; kolmandal tasemel lisandub neile kuuele veel seitse protsessi; neljas tase toob eelnevatele juurde kaks protsessi; ning viies ja viimane tase lisab kolm protsessi. Keskmiselt on igas protsessis defineeritud 3 – 4 antud protsessi olulist eesmärki ning nende eesmärkide saavutamiseks on sõnastatud sõltuvalt protsessist 9 kuni 22 praktilist tegevust.

SW-CMM üldist struktuuri illustreerib tabel 6.

## CMM - Capability Maturity Model for Software

Levels	Key Process Area	Goals	Co	Ab	Ac	Me	Ve	
5	<i>Optimizing</i>							
	DP Defect Prevention	3	2	4	8	1	3	18
	TCM Technology Change Management	3	3	5	8	1	2	19
	PCM Process Change Management	3	2	4	10	1	2	19
4	<i>Managed</i>							
	QPM Quantitative Process Management	3	2	5	7	1	3	18
	SQM Software Quality Management	3	1	3	5	1	3	13
3	<i>Defined</i>							
	OPF Organization Process Focus	3	3	4	7	1	1	16
	OPD Organization Process Definition	2	1	2	6	1	1	11
	TP Training Program	3	1	4	6	2	3	16
	ISM Integrated Software Management	2	1	3	11	1	3	19
	SPE Software Product Engineering	2	1	4	10	2	3	20
	IC Inter-group Coordination	3	1	5	7	1	3	17
	PR Peer Reviews	2	1	3	3	1	1	9
2	<i>Repeatable</i>	20	9	25	62	6	19	121
	RM Requirements Management	2	1	4	3	1	3	12
	SPP Software Project Planning	3	2	4	15	1	3	25
	PTO Software Project Tracking and Oversight	3	2	5	13	1	3	24
	SQA Software Quality Assurance	4	1	4	8	1	3	17
	SCM Software Configuration Management	4	1	5	10	1	4	21
	SSM Software Subcontract Management	4	2	3	13	1	3	22
1	<i>Initial</i>	52	28	71	150	20	47	316

Tabel 6. SW-CMM struktuur

Ve	Verifying implementation	kontrollimine
Me	Measurement and Analysis	Mõõtmine
Ac	Activities to perform	Tegevus
Ab	Ability to perform	Eeldused
Co	Commitment to perform	Tegevuskava

Tabel 7. SW-CMM praktiliste tegevuste liigitus

Igas protsessis on viit liiki praktilisi tegevusi (Tabel.7). SW-CMM praktilised tegevused on heas kooskõlas juhtimisteooriast tuntud juhtimise mõistega. Juhitaks protsesse ja juhtimine tähendab planeerimist (tegevuskava loomist), organiseerimist (eelduste loomist ja tegevuste organiseerimist) ning kontrollimist (tegevuste mõõtmist ja tulemuste kontrollimist).

### 2.2.1 Tegevuskava ja vastutus

Kõigepealt tuleb **kirjeldada tegevus ja määrata vastutus** (Commitment to Perform). Üldjuhul on igas protsessis 1-2 vastutusega seotud praktilist tegevust, mis kõik on teineteisega väga sarnased. Üks vastutusega seotud praktiline tegevus ütleb, et “antud tegevuse läbiviimiseks peab tarkvarafirmal olema kirja pandud tegevuskava ehk üldine plaan mida tuleb teha” ja teine (kui ta on) et “projekti juht on vastutav antud tegevuse läbiviimise ja tulemuste eest”.

### 2.2.2 Tingimuste loomine

Peale tegevuste kirjeldamist ja vastutuse määramist **tuleb luua tingimused** (Ability to Perform) nende tegevuste elluviimiseks. Selliseid eeldusega seotud praktilisi tegevusi on igas protsessis 3 – 5. Eeldused on eelkõige seotud ressurssidega. Ei ole mõtet teha plaane, kui pole olemas ressursse nende plaanide elluviimiseks. Mõeldud on ressursse, mis seotud rahaga, ajaga, vahenditega ja inimestega. Samas rõhutatakse just adekvaatsete ehk piisavate ressursside olemasolu [CMM]. Mina loen sellest välja kui minimaalselt vajalike ressursside olemasolu nõuet.

Rõhutada tuleb töötajate koolitusvajaduse toonitamist tingimuste blokis. Ei saa nõuda töötajalt korrektset ülesande sooritust, kui töötaja pole selle töö soorituseks saanud vastavat koolitust.

Ka rõhutatakse tingimuste blokis tegevuste dokumenteerimist. Samas ei räägita kuskil sellest, et dokumentatsiooni peab olema tonnide kaupa. Oluline on, et tegevusest jääks jälg.

### 2.2.3 Tegevuste korraldamine

Suurem osa SW-CMM praktilistest tegevustest langeb just tegevuste korraldamise (Activities to perform) gruppi. Keskmiselt on kirjeldatud igas protsessis kümnekond tegevust, mis tuleb korraldada. Mõnes rohkem, mõnes vähem.

Selle grupi tegevused on kõik konkreetse protsessi kesksed ning kirjeldavad seda protsessi.

### 2.2.4 Tulemuste mõõtmine

Üldjuhul sisaldab iga protsess ühte praktilist tegevust, mis seotud tulemuste mõõtmisega (Measurement and Analysis). See praktiline tegevus on kõikides protsessides sarnane ja nõuab meetrika olemasolu antud protsessi tegevuste mõõtmiseks.

### 2.2.5 Kontroll ja järeluste tegemine

Selle bloki praktilised tegevused rõhutavad pideva kontrolli ja järelvaatamise olemasolu (Verifying implementation) tegevustele endile. Küsimus on selle, et kas antud tegevused on ikka adekvaatsed ja kas nad ikka annavad vajalikke tulemusi. Sügavam mõte antud blokil on selles, et kõikide tegevuste sisu tuleb muuta, kui need tegevused ei anna soovitud tulemusi. Kuid muudatuste sisseviimine peab olema põhjendatud ning samuti kontrolli all.

## 2.3 Ülevaade XP-metoodikast

XP-metoodika on põhiliselt Kent Beck'i poolt loodud tarkvara arendamise metoodika. Üks enam kõmu tekitanud ja tähelepanu köitnud paindmetoodika maailmas.

XP-metoodikale omistatakse järgmisi põhilisi tegevusi [PAULK2001]:

1. **Planeerimise mäng** – kombineerides äripoolse vajadusi ja tehnilise poole võimalusi lepitakse kiiresti kokku järgmise rakenduse versiooni (*release*) piirides. Klient määrab prioriteedid ja funktsionaalsed vajadused ning arendajad hindavad selle teostamisele kuluvat aega;
2. **Lühikesed arendusperioodid** – lihtne ja kliendile esmavajalike võimalustega süsteem tehakse kiiresti valmis ja võimaldatakse kliendil kohe ka selle kasutamist. Järgmised versioonid (samuti lühikesed) laiendavad süsteemi kasutusvõimalusi.
3. **Metafoor** – lühike ja kõikidele arusaadav lugu süsteemist ja selle tööst on süsteemi arendustegevust koos hoidvaks võtmeks.
4. **Lihtne disain** – süsteemi disain hoitakse pidevalt nii lihtne kui vähegi võimalik.
5. **Testimine** – XP-metoodikas on kasutusel testimisel tuginev arendustehnika (*Test Driven Development*) [BECK2003], mille olulised märksõnad on testi kõike, testi alati, testi enne ja testi automaatselt.
6. **Ümberkirjutamine (*refactoring*)** - süsteemi pidev restruktureerimine eesmärgida muuta kood lihtsamaks ja universaalsemaks ning elimineerida dubleerimist.
7. **Paarisprogrammeerimine** – kogu kood kirjutatakse programmeerijate paaride poolt. Kaks programmeerijat istuvad ühe ja sama arvuti taga ning kirjutavad ühel ja samal ajal ühte ja sama koodi.
8. **Koodi ühisomand** – kood ei kuulu ühelegi programmeerijale personaalselt. Igal arendajal on lubaküsimate ja igal ajal õigus muuta ükskõik millist osa koodist.
9. **Pidev integratsioon** – niipea, kui mingi uus funktsioon on realiseeritud integreeritakse see süsteemi tervikuks. Kasvõi mitu korda päevas. Testimisel tuginev arendustehnika kasutamine võimaldab sellise pideva integratsiooni ning välistab võimalikud kaasnevad ebameeldivused.
10. **40-tunnine töönnädal** – reeglina töötatakse nädalas mitte enam, kui 40 tundi. Kunagi ei tehta ületunde kaks nädalat järjest.
11. **Kasutaja osalemine** – reaalne ja tegelik kasutaja on meeskonna liige. On pidevalt kohal, kirjutab kasutajalugusid ja testide stsenaariume, testib rakendust ja vastab arendajate küsimustele.

**12. Koodi standardid** – kõik kasutajad järgivad koodi kirjutamisel ühtseid standardeid. Koodi lugedes ei ole võimalik arendajat identifitseerida

Suurem osa XP-metoodika printsiipidest nagu minimaalsus, lihtsus, lühikesed tsükliid kasutaja osalemine, kodeerimise standardid, testimine on praktilised ja kasutuses ka teistes metoodikates [PAULK2001]. XP- metoodika on aga nendele andnud nn “ekstreemse” tähenduse. Tabelis 8 on toodud Mark C. Paulk’i arusaamine “ekstreemsusest” XP-metoodikas [PAULK2001].

Üldine praktika	Ekstreemsus	XP-metoodika
Koodi läbivaatamine	Koodi läbivaatamist tehakse pidevalt	Paaris programmeerimine
Testimine	Testi kogu aeg	Elementaartestid, sobivustestid.
Disain	Disainimine on arendustegevuse igapäevane osa	Ümberkirjutamine (refactoring)
Lihtsus	Kasuta alati lihtsaimat disaini, mis rahuldab olemasolevat funktsionaalsust	Lihtsaim asi, mis töötab
Arhitektuur	Igaüks võib täpsustada süsteemi arhitektuuri.	Metafoor
Terviku testimine	Integreeri ja testi mitu korda päevas	Pidev integratsioon
Lühikesed iteratsioonid	Iteratsioonid tuleb teha väga lühikesed.	Planeerimise mäng

Tabel 8. „Ekstreemsus“ XP-metoodikas [PAULK2001].

XP-metoodikas on olulised ka rollid, mida arendajad peavad täitma. Kokku on määratud seitse rolli [BECK2000].

1. **Programmeerija (programmer)**, kes kirjutab testid ja realiseerib rakenduse, on põhiline roll XP-metoodikas. XP-metoodikas on kood (nii testid kui rakenduse realisatsioon) peamine dokument ja peamine väljund;
2. **Kasutaja (customer)** kirjutab soovilugusid. Soovilugude abil annab kasutaja programmeerijale teada, mida on vaja teha. Programmeerija oskab programmeerida, kuid kasutaja peab ütlema, mida on vaja programmeerida.
3. **Testija (tester)** ülesanne on aidata kasutajal sõnastada, valida ja ka realiseerida sobivustestid. Ka on testija ülesanne hoolitseda automaatsete testide käivitamise eest. Testid annavad pildi projekti edusammudest.
4. **Jäljekütt (tracker)** paneb kirja ja jälgib igasuguseid arendustegevusega seotud numbrilisi väärtusi: mitu ideaalpäeva on kulunud mingi ülesande realiseerimiseks; mitu ülesannet antud nädalas realiseeriti; mitu ülesannet on vaja veel realiseerida, et püsida graafikus,...
5. **Treener (coach)** on vastutav kogu arendusprotsessi (XP-metoodika järgimise, kvaliteedi, tähtaegade jne) eest. Treener otsustab, kas arendusmeeskond on



“järje peal” või mitte ja viib ellu muudatused (vajadusel ka karmid) selleks, et meeskond järjele saada.

6. **Konsultant** (*consultant*) on tehniliste eriküsimuste spetsialist ja annab meeskonna liikmetele eelkõige tehnilist konsultatsiooni. On meeskonna liige, kes on mingi asja endale selgeks teinud või ajutiselt palgatud spetsialist.
7. **Suur juht** (*Big Boss*) on meeskonna juht ja varustab meeskonda vajalike ressurssidega. Peab omama projektist üldist pilti, olema kursis projekti seisuga. Üldjuhul arendustegevusega ei tegele.

Üks ja sama inimene võib meeskonnas täita ka mitmeid rolle. Põhjalik XP-metoodika kirjeldus on toodud lisas 2.

## 2.4 XP-metoodika võimaldab täita SW-CMM teise taseme nõudeid

Mark C. Paulk SEI-st hindab oma artiklis „Extreme Programming from a CMM perspective“ [PAULK2001] XP sobivust SW-CMM nõuete täitmiseks. Tema tulemused on toodud tabelis 9.

CMM-2	XP
RM – Requirements Management	Hea
SPP – Software Project Planning	Hea
PTO – Software Project Tracking and Oversight	Hea
SQM – Software Quality Assurance	Osaline
SCM – Software Configuration Management	Osaline
SSM – Software Subcontract Management	Puudub
CMM-3	XP
OPF – Organization Process Focus	Osaline
OPD – Organization Process Definition	Osaline
TP – Training Program	Puudub
ISM – Integrated Software Management	Puudub
SPE – Software Product Engineering	Hea
IC – Inter-group Coordination	Hea
PR – Peer Reviews	Hea
CMM-4, 5	XP
DP – Defect Prevention	Osaline
TCM – Technology Change Management	Puudub
PCM – Process Change Management	Puudub
QPM – Quantitative Process Management	Puudub
SQM – Software Quality Management	Puudub

Tabel 9. XP-metoodika sobivus SW-CMM nõuete rahuldamiseks [PAULK2001]

Kokkuvõttes rõhutab Mark C. Paulk, et XP koosneb suurest hulgast praktilistest meetoditest, millede sidumine ühtseks metoodikaks võimaldab tarkvara arendamise protsessile läheneda süsteemselt nii nagu SW-CMM läheneb süsteemselt tarkvara firma kui organisatsiooni arendamise ja täiustamise protsessile [PAULK2001].

Organisatsioonid, kes soovivad tõsta oma küpsust peavad võtma kasulikke ideid mõlemast ning neid ellu viima [PAULK2001].

Alljärgnevalt arutlen võimaluste üle, kuidas XP-metoodika meetodeid kasutades ning vajadusel neid täiendades on võimalik SW-CMM teise taseme võtmeprotsesside eesmärgid ja nõudeid rahuldada.

## 2.4.1 Nõuete haldamine

Nõuete haldamise (RM – Requirements Management) otstarve SW-CMM järgi on projekti meeskonnas ühise ja adekvaatse arusaamise kujundamine antud tarkvara projektiga seotud kliendi vajadustest. Tabelites 10 ja 11 on toodud kokkuvõtlikult kuidas XP-metoodikaga on võimalik SW-CMM nõuete haldamise eesmärgid saavutada ja vajalikud tegevused läbi viia.

XP-metoodika võimaldab SW-CMM nõuete haldamise nõudeid hästi täita [PAULK2001]. XP-metoodikas on nõuete haldamises oluline roll **kliendil** (*customer*), kes valdab väljatöötava tarkvara temaatikat. Klient osaleb vahetult arendustegevuses: kirjutab soovilugusid ning sobivustestide stsenaariume; osaleb analüüsi ja planeerimise koosolekutel; vastab arendajate küsimustele ning testib rakendust. Ideaalses olukorras on klient ka antud rakenduse hilisem reaalne kasutaja.

Kliendi esmane ülesanne XP meeskonnas on lihtsas ja arusaadavas keeles (vältida tuleb tehnilisi termineid) kirjeldada arendajatele oma tarkvaralisi nõudeid, kriteeriume, funktsioone ja soove ning kontrollida arendajate poolt realiseeritud tarkvara vastavust nendele. Tarkvaralised nõuded paneb klient lühidalt jutukestena e. **kliendi soovilugudena** (*story*) kirja.

XP-metoodikas jaotatakse arendusprojekt umbes kuue kuu pikkusteks **väljalaseteks** (*release*). Väljalasked (järjekordse versiooni valmimise periood) jaotuvad 2-3 nädala pikkusteks **iteratsioonideks**. Iteratsioon algab **analüüsikoosolekuga**, kus on arutusel selles iteratsioonis realiseeritavad soovilood. Analüüsikoosolekust võtab osa terve meeskond (järeltuleb ka klient, kui meeskonna liige). Analüüsikoosolekul määratakse, millised soovilood ja millises järjekorras antud iteratsioonis realiseeritakse. Soovilugude ja nende järjekorra määramisel on oluline roll kliendil. Analüüsikoosolekul arutatakse ka, kuidas soovilugu kõige lihtsamalt realiseerida ning hinnatakse sooviloo realiseerimiseks kuluvat aega. Saadud hinnangute järgi koostatakse iteratsiooni plaan. Olulised rollid on siin treeneril ja konsultandil.

Iga kliendi sooviloo kohta kirjutatakse enne sooviloo realiseerimist valmis **sobivustest** (*functional test*). Enne iga ülesande (arendusühik) realiseerimist kirjutatakse ülesande realiseerimist kinnitav **ühiktest** (*unit test*). Sobivustesti stsenaariumi kirjutab klient ning kriteeriumid kooskõlastatakse kliendiga.

Soovilugu loetakse realiseerituks, kui kõik tema koostisülesanded on läbinud ühiktestid ja soovilugu ise läbib täielikult sobivustesti ning tervikuks integreerituna läbib rakendus ka kõikide eelnevalt realiseeritud soovilugude ühiktestid ning sobivustestid. Kuna testid on realiseeritud tarkvaraliselt, on loodava tarkvara testimist võimalik automatiseerida. Testide automaatse käivitamise eest vastutab testija. Lisaks on kliendi ülesandeks ka rakenduse visuaalne testimine.

	<b>SW-CMM</b>	<b>XP-metoodika</b>
1	Projektile esitatud nõuded peavad olema tarkvara arendamise aluseks.	Tarkvara arendamise aluseks on kliendi poolt kirja pandud soovilood.
2	Valmis tarkvara peab vastama esitatud nõuetele.	Klient hoolitseb selle eest, et tarkvara vastaks nõuetele. Kliendi pidev kohalolek, tihe tervikuks integreerimine ja sobivustestid aitavad teda selles.

*Tabel 10. SW-CMM nõuete haldamise eesmärgid ja XP-metoodika*

	<b>SW-CMM</b>	<b>XP-metoodika</b>
<b>Co1</b>	Nõuete haldamine allub dokumenteeritud reeglitele.	Nõuete haldamine allub XP-metoodika reeglitele.
<b>Ab1</b>	Igal projektil on olemas nõuete analüüsi eest vastutav, kes ka teeb kindlaks ja otsustab, kas antud nõue on realiseeritav riisvaraliselt, tarkvaraliselt, või kuidagi teisiti	Kliendi poolt kirja pandud nõudeid analüüsitakse iteratsiooni algul toimival analüüsikoosolekul. Personaalselt vastutab arendusprojekti eest treeneri rolli täitev arendaja.
<b>Ab2</b>	Nõuded dokumenteeritakse	Klient kirjutab soovilugusid.
<b>Ab3</b>	Nõuete haldamiseks on eraldatud adekvaatsed ressursid.	XP-metoodika läbiviimiseks on ressursid tagatud arenduslepinguga.
<b>Ab4</b>	Inimesed on koolitatud tarkvara nõudeid haldama	Inimesed on koolitatud kasutama XP-metoodikat.
<b>Ac1</b>	Analüüsimeeskond töötleb ja analüüsib nõudeid enne realiseerimist	Analüüs teostatakse meeskonna poolt sooviloo analüüsi koosolekul.
<b>Ac2</b>	Analüüsimeeskond võtab nõuded aluseks edasiste plaanide koostamisel ja tarkvara kavandamisel	Analüüsikoosolekul koostatakse iteratsiooni plaan, mis lähtub selles iteratsioonis realiseeritavatest soovilugudest ning ülesannetest.
<b>Ac3</b>	Muudatused nõuetes fikseeritakse, töödeldakse, lisatakse projekti ning antakse kõikidele teada.	Klient paneb muudatused kirja tavaliste soovilugudena ja need realiseeritakse tavalises korras.
<b>Me1</b>	Nõuete haldamise tegevusi mõõdetakse	Jäljekütt fikseerib soovilugude ja ülesannete arvu ning nende realiseerimiseks kulutatud aja, muudatusi sisaldavate soovilugude arvu ja muud meetrikat.
<b>Ve1</b>	Nõuete haldamise protsessi tegevused on tippjuhtkonna tähelepanu all	Tippjuhi rolli täidab suur juht. Nõuete haldamise protsess kajastub mõõtmistes.
<b>Ve2</b>	Projektijuhid tegelevad nõuete haldamise igapäevaste küsimustega.	Treeneri roll XP meeskonnas.
<b>Ve3</b>	Kvaliteedi eest vastutajad jälgivad ja auditeerivad nõuete haldamise protsessi	Loodava tarkvara kvaliteedi eest vastutavad treener ja testija. Protseduurireeglite järgimist jälgib treener. Meeskonna koosolekutel arutatakse puudusi.

*Tabel 11. SW-CMM nõuete haldamise tegevused ja XP-metoodika*

## 2.4.2 Tarkvaraprojekti planeerimine

Tarkvaraprojekti planeerimise (SPP – Software Project Planning) otstarve SW-CMM järgi on põhjendatud ja vettpidavate tarkvara arendamise plaanide koostamine. XP-metoodika võimaldab SW-CMM planeerimise nõudeid hästi rahuldada [PAULK2001].

XP-s kasutatakse mõistet **juhitav arendustegevus**, mille all mõistetakse pidevat asjade kulgemise jälgimist ja vajadusel korrektsioonide tegemist.

XP planeerimise protsessis on äripoole inimeste ja tehnilise poole inimeste rollid kindlalt jaotatud. Kummalgi poolel pole õigust otsustada teise valdusalasse jäävates piirides. Äriiga seotud inimesed teevad ainult äriiga seotud otsuseid; tarkvara arendajad aga ainult tarkvara arendamisega seotud tehnilisi otsuseid.

XP-metoodika rakendamise korral pole võimalik üheaegselt fikseerida kahte asja: rakenduses realiseeritavat funktsionaalsust ja selle valmistamiseks kulutatavat aega (vaata 1.2.5. “Tarkvaraprojekti neli mõõdet”). Kui klient soovib kindlasti fikseerida funktsionaalsuse, jääb arendajatele võimalus kulutada aega niipalju, kui selle kvaliteetseks realiseerimiseks ja testimiseks aega kulub. Enne asja lõplikku realiseerimist ei tea nagunii keegi, kui palju selleks tegelikult aega vaja on.

Kui klient paneb paika tähtsajad, tuleb valmistatavale tarkvarale esitatavaid funktsionaalseid nõudeid pidevalt korrigeerida (kas suurendada või vähendada).

Projekti planeerimise aluseks on kliendi poolt kirja pandud ning tähtsuse järgi reastatud soovilood. Projekt realiseeritakse umbes kuue kuu pikkuste väljalasete (*release*) käigus. Esimeses väljalaskes realiseeritakse kliendile kõige olulisemad soovilood. Oluline on, et juba peale esimese väljalaske valmimist on kliendil võimalik rakendust kasutada. Väljalasked planeeritakse väga üldiselt projekti algusfaasis ja neid plaane korrigeeritakse pidevalt vastavalt tegelikule jõudlusele, kuna [BECK&FOWLER2001]:

1. Kasutajal on alati õigus ja võimalus lisada soovilugusid, muuta soovilugude prioriteete.
2. Arendajal on alati õigus kulutada töö tegemiseks aega vastavalt reaalsele vajadusele.
3. Meeskonnal on alati õigus alaplaneerimise korral tööd juurde võtta ja üleplaneerimise korral vähem prioriteetseid töid järgmisse redaktsiooni edasi lükata.

Esimest väljalaset on kõige raskem planeerida ja selles on ka kõige rohkem möödalaskmisi. Vastavalt meeskonna ja projekti vajadustele planeeritakse väljalasked kas pikemad või lühemad; kasutaja soovilood kas suuremad või väiksemad.

Väljalasked realiseeritakse iteratsioonide (2-3 nädalat) kaupa. Iteratsioon algab analüüsi koosolekuga, kus lahatakse soovilood ülesanneteks ja planeeritakse nende ülesannete realiseerimiseks kulutatav aeg.

Planeerimise aluseks on analoogia ja kogemused ning arendusprotsessi käigus kogutav informatsioon: palju analoogiliste asjade tegemiseks on kunagi aega kulunud.

Sellise informatsiooni kogumise ja haldamisega tegeleb jäljekütt. Rõhutada tuleb, et XP-s planeeritakse arendustegevus meeskonnas ühiselt. Konkreetsed ülesanded ja vastutuse võtab aga iga programmeerija isiklikult vastavalt enda kogemustele ja jõudlusele.

Kui on planeeritud rohkem, kui teha suudetakse, siis seda ka täpselt selliselt võetakse: vähe pole mitte aega, vaid planeeritud on liiga palju. Ei ole võimalik anda aega juurde. On võimalik vähem teha [BECK&FOWLER2001].

Planeerimisel kasutatakse lihtsaid töövahendeid: paber, pliiats ja tahvel on piisavalt head planeerimise vahendid. Inimeste vaheline kommunikatsioon on planeerimise juures tunduvalt tähtsam, kui kõikvõimalikud "võluvahendid".

Mõningate SW-CMM planeerimise nõuetega tuleb XP-metoodikat järgival tarkvarafirmal veel lisaks arvestada, kui soovitakse ka rahuldada SW-CMM teise taseme nõudeid. Sellised nõuded on:

- Ac2 - Kui tarkvara arendamine on üks osa kogu teostatavast projektist, siis alustatakse tarkvara planeerimist samaaegselt ning planeeritakse paralleelselt üldise plaaniga;
- Ac4 - Vastutuse delegeerimine isikutele ja gruppidele väljaspool organisatsiooni on allutatud dokumenteeritud protseduuridele ja viiakse läbi tippjuhtkonna poolt;
- Ac15 - Plaanid dokumenteeritakse ja säilitatakse.

XP-metoodika soodustab ning loob kõik eeldused nende nõuete täitmiseks.

Tabelites 12 ja 13 on toodud kokkuvõtlikult kuidas XP-metoodikaga on võimalik SW-CMM planeerimise eesmärgid saavutada ning vajalikud tegevused läbi viia.

	<b>SW-CMM</b>	<b>XP-metoodika</b>
1	Tarkvaraprojekt tuleb planeerida vastavalt etteantud eelarvele.	Tarkvaraprojekt planeeritakse vastavalt kliendiga sõlmitud lepingule kas funktsionaalsusest või tähtaegadest lähtuvalt. Funktsionaalsusest lähtuva planeerimise korral on eelarve maht lahtine; fikseeritud eelarve mahust lähtuva planeerimise korral on realiseeritavate funktsioonide hulk lahtine.
2	Tegevused ja vastustus tarkvaraprojektis tuleb planeerida ja dokumenteerida.	XP-metoodika ja tarkvaraprojekti planeerimise põhimõtted on kliendiga kooskõlastatud ja lepinguga määratud.
3	Arendustegevusega seotud grupid (ka klient) on teadlikud ja nõus tähtaegade ning vastutusega.	Klient ja kogu arendusmeeskond on teadlikud rakendatava metoodika olemusest, vastastikustest õigustest ja kohustustest.

*Tabel 12. SW-CMM tarkvaraprojekti planeerimise eesmärgid ja XP-metoodika*

	<b>SW-CMM</b>	<b>XP-metoodika</b>
<b>Co1</b>	Projekti juht vastutab planeerimise ja vastutuste kooskõlastamise eest	Treener täidab XP-s projektijuhi rolli.
<b>Co2</b>	Planeerimisel lähtutakse kehtivatest kirjalikest tavadest ja standarditest	Planeeritakse vastavalt XP-metoodika planeerimise meetodeid kasutades.
<b>Ab1</b>	Projekti jaoks on olemas dokumenteeritud ja kinnitatud tellimus	Tarkvara väljatöötamise aluseks on kliendiga sõlmitud arendusleping.
<b>Ab2</b>	Arendusplaani väljatöötamise eest on määratud vastutaja	Treener vastutab kogu projekti, seega ka arendusplaani väljatöötamise eest.
<b>Ab3</b>	Planeerimise läbiviimiseks on olemas adekvaatsed ressursid	XP-metoodika läbiviimiseks on ressursid tagatud arenduslepinguga.
<b>Ab4</b>	Inimesed on koolitatud ja nad oskavad tarkvaraprojekte planeerida	Inimesed on koolitatud kasutama XP-metoodikat.
<b>Ac1</b>	Tarkvara arendajad osalevad projekti ettevalmistavas töös	Arendusmeeskond osaleb projekti ettevalmistavas töös.
<b>Ac2</b>	Kui tarkvara arendamine on üks osa kogu teostatavast projektist, siis alustatakse tarkvara planeerimist samaaegselt ning planeeritakse paralleelselt üldise plaaniga	Tarkvarafirma, kes soovib SW-CMM teise taseme nõudeid rahuldada peab selle nõude lisaks juurutama.
<b>Ac3</b>	Arendajad osalevad üldistes aruteludes kogu projekti elutsükli vältel;	Arendusmeeskond osaleb kogu elutsükli vältel projekti üldistes aruteludes.
<b>Ac4</b>	Vastutuse delegeerimine isikutele ja gruppidele väljaspool organisatsiooni on allutatud dokumenteeritud protseduuridele ja viiakse läbi tippjuhtkonna poolt	Tarkvarafirma, kes soovib SW-CMM teise taseme nõudeid rahuldada peab selle nõude lisaks juurutama.  Suur juht täidab tippjuhi rolli.
<b>Ac5</b>	Tarkvara elutsükkel on jagatud väiksemateks osadeks;	XP-metoodikas on arendusühikuteks ülesanded, soovilood, iteratsioonid ja redaktsioonid.
<b>Ac6</b>	Tarkvara arendusplaani väljatöötamise aluseks on organisatsioonis dokumentaalselt kehtestatud standardid ja reeglid	Arendusplaani väljatöötamise aluseks on XP planeerimise meetodid.
<b>Ac7</b>	Tarkvara väljatöötamise plaan on kirjalik dokument	Tarkvara väljatöötamise plaan on kõikidele kättesaadav kirjalik dokument.
<b>Ac8</b>	Plaani koostamiseks ja täitmise kontrollimiseks on tegevused määratud;	Vastavad tegevused on: projekti üldise mahu hindamine; projekti redaktsioonideks jaotamine; iteratsioonide planeerimine; ülesannetele kulutatud aja mõõtmine; ülesannete ja soovilugude testimine.
<b>Ac9</b>	Plaanis kajastuvad ka peamiste tööde mahud	Plaanis on kajastatud soovilood ja nende realiseerimiseks planeeritud aeg.
<b>Ac10</b>	Maksumus ja tähtajad planeeritakse	Maksumus ja tähtajad planeeritakse

	lähtuvalt dokumenteeritud reeglitest	XP-metoodikast lähtuvalt.
<b>Ac11</b>	Arvutusvõimsused planeeritakse lähtuvalt dokumenteeritud reeglitest	Arvutusvõimsused hinnatakse lähtuvalt eelnevatest kogemustest. Testid kinnitavad piisavuse.
<b>Ac12</b>	Projekti ajakava planeeritakse lähtuvalt dokumenteeritud reeglitest	Kas fikseeritakse aeg ja lahtine on funktsionaalsus, või fikseeritakse funktsionaalsus ja lahtine on aeg.
<b>Ac13</b>	Maksumusega, ressurssidega, tähtaegadega ja tehniliste aspektidega seotud riskid määratakse, hinnatakse ja dokumenteeritakse	XP tugineb tõsiasjal, et arendustegevuses võib kõik minna untsu (vähemalt mitte nii, nagu planeeritakse). Kogu XP metoodika on üles ehitatud selleks, et arendustegevuse riske maandada.
<b>Ac14</b>	Plaanid tehakse nii tarkvara arendamiseks, kui ka arendamist toetavateks tegevusteks	Testimine ja testide automatiseerimine on tihedalt integreeritud XP arendustegevuses.
<b>Ac15</b>	Plaanid dokumenteeritakse ja säilitatakse	Tarkvarafirma, kes soovib SW-CMM teise taseme nõudeid rahuldada peab selle nõude lisaks juurutama.
<b>Me1</b>	Planeerimistegevusi mõõdetakse ja saadud tulemusi kasutatakse hilisemates planeerimistöödes	Jäljekütt fikseerib soovilugude ja ülesannete arvu, ülesannete ja soovilugude realiseerimiseks kulutatud aega, muudatusi sisaldavate soovilugude arvu ja muud meetrikat.
<b>Ve1</b>	Planeerimisega seotud tegevused on tippjuhtkonna tähelepanu all.	Tippjuhi rolli täidab suur juht.
<b>Ve2</b>	Igapäevast planeerimistegevust koordineerib projekti juht.	XP meeskonnas täidab projektijuhi rolli treener.
<b>Ve3</b>	Kvaliteedi eest vastutav grupp jälgib ja auditeerib planeerimistegevusi selleks, et tagada adekvaatsus.	Loodava tarkvara kvaliteedi eest vastutab testija. Protseduurireeglite järgimist jälgib treener. Meeskonna koosolekutel arutatakse puudusi.

Tabel 13. SW-CMM tarkvaraprojekti planeerimise tegevused ja XP-metoodika

### 2.4.3 Tarkvaraprojekti monitooring

Tarkvaraprojekti monitooringu (PTO – Software Project Tracking and Oversight) otstarve SW-CMM järgi on adekvaatse ülevaate saamine arendustegevuse käigust selleks, et oleks võimalik vastu võtta õigeid otsuseid.

XP-metoodika võimaldab SW-CMM tarkvaraprojekti monitooringu nõudeid hästi rahuldada [PAULK2001].

Iteratsiooni planeerimise koosolekul, kus sellesse iteratsiooni kuuluvad soovilood jaotatakse mõne päeva jooksul realiseeritavateks ülesanneteks, märgib iga arendaja endale ülesanded, mida just tema konkreetselt realiseerib ja palju ta iga konkreetse ülesande jaoks aega vajab.

Ülesanne on XP-metoodika väikseim realiseeritav osa. Ülesanne realiseeritakse, testitakse ja integreeritakse rakendusse programmeerijate paari poolt.

Ülesande realiseerimiseks planeeritavat ja tegelikult kulutatud aega mõõdetakse **ideaalsetes arenduspäevades**. Ideaalne arenduspäev on keskmiselt kaheksa tundi pikk ja selle aja jooksul arendaja ei tee muud, kui kirjutab koodi. Ühte iteratsiooni (kaks nädalat) ei mahu kunagi kümme ideaalset arenduspäeva. Hea, kui keskmiselt on ühes iteratsioonis ühel arendajal 5-7 sellist arenduspäeva. Peale koodi kirjutamise peab arendaja ka veel osalema planeerimiskoosolekutel, suhtlema teistega ja jälgima teisi (paarisprogrammeerimise nõue).

Kõikide arendajate ideaalsete arenduspäevade summat ühes iteratsioonis nimetatakse projekti **arenduskiiruseks**.

XP korral realiseeritakse rakendus programmeerijate poolt ühiselt soovilugude kaupa. Rakendus integreeritakse tervikuks aga ülesannete kaupa. Alati, kui programmeerija on lõpetanud ülesande realiseerimise ja testimise, integreerib ta enda poolt realiseeritud ülesande ja testi koodi rakenduse testimise arvutisse (sellise arvuti olemasolu on XP korral kohustuslik) ja kontrollib et rakendus läbiks veel ka kõik eelnevad testid.

XP metoodikat rakendades on seega võimalik mõõta ja kasutada projekti kulgemise jälgimiseks järgmisi meetrikaid:

- Planeeritud ja realiseeritud soovilugude arv kokku, väljalaskes, iteratsioonis;
- Planeeritud ja realiseeritud ülesannete arv kokku, väljalaskes, iteratsioonis, soovilugudes, ideaalses arenduspäevas;
- Ideaalsete arenduspäevade arv iteratsioonis;
- Projekti kiiruse muutumised;

Paar korda nädalas fikseerib üks jäljekütt tegeliku olukorra: kas arendajad on enda võetud kohustuste graafikus või mitte.

Iga päev toimub **välkkoosolek** (seistes; umbes 15 minutit), kus iga arendaja annab olukorrast lühikese ülevaate. Nii on tegelik olukord kõikidele teada. Ka on kõikidele nähtav ja kättesaadav projekti, redaktsiooni ja iteratsiooni kulgu kajastavad tabelid ja graafikud.

Lõpetatud soovilugudes väljatulnud vigade parandamine planeeritakse soovilugudena ühistel alustel: nii antakse kasutajale võimalus valida uute funktsionaaluste lisamise ja vigade likvideerimise vahel.

Tarkvaraprojekti monitooringu nõuded on põhiliselt kaetud järgmiste XP meetoditega:

- Projekti kiiruse määramise, mõõtmise ja korrigeerimisega;
- Sobivus- ja ühiktestide automatiseerimisega;
- Ülevaadete koostamisega.



Tabelites 14 ja 15 on toodud kokkuvõtlikult kuidas XP-metoodikaga on võimalik SW-CMM tarkvaraprojekti monitooringu eesmärgid saavutada ning vajalikud tegevused läbi viia.

	<b>SW-CMM</b>	<b>XP-metoodika</b>
1	Tulemuste ja jõudluse võrdlemine tehtud plaanidega;	Mõõdetakse soovilugude ja ülesannete realiseerimiseks planeeritud ja tegelikult kulutatud aega.
2	Meetmete rakendamine juhul, kui plaan ja tegelikkus drastiliselt erinevad;	Kui plaan ja tegelikkus erinevad, muudetakse plaani.
3	Muudatused vastutusalades tuleb kooskõlastada gruppide ja isikute vahel.	Plaanide koostamisest ja muutmisest võtavad osa kõik meeskonna liikmed. Kliendi esindaja on arendusmeeskonna liige.

Tabel 14. SW-CMM tarkvaraprojekti monitooringu eesmärgid ja XP-metoodika

	<b>CMM</b>	<b>XP</b>
<b>Co1</b>	Projekti juht on vastutav projekti käekäigu ja tulemuste eest	Treener on vastutav kogu projekti käekäigu eest.
<b>Co2</b>	Tarkvaraprojekti monitooringul jälgitakse firma siseseid dokumenteeritud nõudeid ja standardeid	Jälgitakse XP nõudeid. On olemas firma sisene kõikidele kohustuslik koodi standard.
<b>Ab1</b>	On olemas dokumenteeritud arendusprojekti plaan	On olemas projekti, väljalaske ja iteratsioonide plaanid.
<b>Ab2</b>	Projekti juht delegeerib konkreetsete ülesannete täitmiseks vastutust konkreetsetele isikutele ja gruppidele	Iteratsioonide planeerimise koosolekul võtab iga arendaja endale ise kohustused.
<b>Ab3</b>	Projekti monitooringuks on olemas ressursid ja rahalised vahendid	XP eeldab minimaalselt vähemalt kahte asja: planeerimine ja testimine. Planeerimine ja testimine on XP-ga organiliselt seotud. Kui kliendiga sõlmitakse leping töö teostamiseks, on ka saadud kliendi nõusolek projekti ressurssidega katmiseks.
<b>Ac1</b>	Dokumenteeritud arendustegevuse plaan on arendusprojekti monitooringu aluseks	Arendustegevuse plaan pannakse paika peale kliendiga lepingu sõlmimise, kus fikseeritakse esialgne ja üldine plaan: mida teha ja palju see umbes aega võtab. See plaan jääb aluseks. Hiljem seda plaani pidevalt muudetakse, parandatakse ja täiendatakse vastavalt reaalsele olukorrale.
<b>Ac2</b>	Plaani muutmiseks on kehtestatud dokumenteeritud reeglid	Plaani muutmine on täiesti loomulik XP osa. Kliendil on õigus plaani pidevalt muuta, ilma spetsiaalsete lisakulutusteta. Plaani muutmine kajastub soovilugudes või nende

		järjekordades.
<b>Ac3</b>	Tarkvara arendusprojektiga seotud vastutuse määramine ja vastutuse muutmine indiviididele ja gruppidele väljaspool organisatsiooni kuulub tippjuhtkonna vastutusalasse ning on allutatud dokumenteeritud reeglistikule	XP ei kirjelda suhteid väljaspool arendusmeeskonda. Kuid ka ei välista sellise reeglistiku sisseviimist vastavalt vajadusele.
<b>Ac4</b>	Projekti mõjutavad muudatused vastutusalades kooskõlastatakse arendusgrupiga ja teiste asjast huvitatud gruppidega (kaas arvatud klient)	Kõiki projektiga seotud küsimusi arutatakse arendusmeeskonnaga ühiselt. Arendusmeeskonna orgaaniline liige on klient (või tema esindaja).
<b>Ac5</b>	Tööde mahtusid jälgitakse. Vajadusel korrigeerimised	Tööde mahtusid planeeritakse alates projekti alustamise koosolekust ja lõpetades soovilugude ülesanneteks jaotamise koosolekuga. Tegelikult kulutatud aega mõõdetakse ja fikseeritakse. Plaane korrigeeritakse pidevalt.
<b>Ac6</b>	Saavutusi ja kulutusi jälgitakse. Vajadusel korrigeerimised	Kõik ülesanded peavad läbima ühiktesti ja kõik soovilood peavad läbima sobivustesti. Testid säilitatakse ning on automaatselt käivituvad. Seega on projekti saavutused pidevalt fikseeritud ja tõestatavad ning saavutuste ja kulutuste suhe suhteliselt hästi hinnatav.
<b>Ac7</b>	Projekti kriitilisi arvutusvõimsusi jälgitakse. Vajadusel korrigeerimised	Ideoloogia on: kõigepealt tuleb saada rakendusele “pilt ette” ja seejärel korrigeerida vajadusel jõudlust. Pideva testimise ja integreerimise abil üritatakse kriitilistele kohtadele (sealhulgas ka arvutusvõimsustele) kiiresti jälile saada.
<b>Ac8</b>	Projekti ajagraafikut jälgitakse. Vajadusel korrigeerimised	Projekti ajagraafikut muudetakse koostöös kliendiga pidevalt vastavalt reaalsele jõudlusele.
<b>Ac9</b>	Kasutatavaid arendustehnikaid ja nende sobivust jälgitakse. Vajadusel korrigeerimised	XP eeldab, et metoodikat ennast pidevalt jälgitakse, täiendatakse ja korrigeeritakse vastavalt vajadusele.
<b>Ac10</b>	Maksumusega, ressurssidega, tähtaegadega ja arendustehnikatega kaasnevaid riske jälgitakse. Vajadusel korrigeerimised	XP ideoloogia aluseks on tõsiasi, et arendustegevus on riskantne ja alati on võimalik millegi untsu minek. Riskide maandamiseks alustatakse arendustegevust kõige tähtsamatest ja kõige keerulisematest ülesannetest. Protsessi pidev jälgimine annab hea ülevaate tegelikust olukorrast. Kohe

		kui plaanid ei vasta reaalsusele, korrigeeritakse plaane.
Ac11	Arendustegevusega seotud andmeid kogutakse ja säilitatakse edasise parema planeerimise eesmärgil	XP-metoodika eeldab andmete säilitamist ja nende kasutamist edasisel planeerimisel.
Ac12	Arendusgrupp teeb perioodilisi kokkuvõtteid arendustegevusest ja arendustegevuse vastavusest plaanidele	Toimuvad projekti, väljalaske, iteratsiooni ja päeva alustamise koosolekud.
Ac13	Projekti ametlikud läbivaatamised toimuvad perioodiliselt ning vastavalt kehtestatud protseduurireeglitele	Toimuvad projekti, väljalaske, iteratsiooni ja päeva alustamise koosolekud.
Me1	Arendusprotsessi mõõdetakse. Tulemusi kasutatakse olukorra hindamiseks	Mõõdetakse ja fikseeritakse ülesannetele planeeritud aega ja ülesannete täitmiseks tegelikult kulutatud aega; Mõõdetakse ja fikseeritakse soovilugude realiseerimiseks planeeritud aega ja tegelikult realiseeritud soovilugude hulka igal ajamomendil. Mõõdetakse projekti kiiruse muutumist.
Ve1	Tarkvara arendamine on pidevalt tippjuhtkonna huviorbiidis;	Tippjuhi rolli täidab suur juht, kes on meeskonna liige.
Ve2	Tarkvara arendamise igapäevast juhtimist teostab vastava projekti juht;	XP meeskonnas täidab projektijuhi rolli treener.
Ve3	Kvaliteedi grupp jälgib ja auditeerib arendustegevust.	Loodava tarkvara kvaliteedi eest vastutab testija. Protseduurireeglite järgimist jälgib treener. Meeskonna koosolekutel arutatakse puudusi.

Tabel 15. SW-CMM tarkvaraprojekti monitooringu tegevused ja XP-metoodika

#### 2.4.4 Allhangete juhtimine

Allhangete juhtimise (SSM – Software Subcontract Management) otstarve SW-CMM järgi on leida parimaid alltöövõtjaid ja neid efektiivselt juhtida.

Allhangete juhtimine on ainuke SW-CMM protsessidest, mis pole kohustuslik juhul, kui organisatsioon ei tegele alltöövõttudega.

XP-metoodika allhangete juhtimist ette ei näe. Antud töös allhangete juhtimisega seonduvat probleemistikku ei puudutata.

#### 2.4.5 Kvaliteedi tagamine

Kvaliteedi tagamise (SQA – Software Quality Assurance) eesmärgiks on anda erapooletu hinnang protsessi käigust ja tagada nii toodangu kvaliteet, kui ka tagada, et tootmiseks kasutatav metoodika oleks kvaliteetne (ei raiskaks ressursse).

SQA annab tippjuhtkonnale võimaluse olla informeeritud juhul, kui tarkvara arendusprojekti seire (üks CMM põhiprotsessidest) PTO (*Software Project Tracking*

*and Oversight*) ei anna adekvaatset tulemust ([DYMOND1998] lk 2-36). XP-metoodika võimaldab SW-CMM kvaliteedi tagamise nõudeid rahuldavalt täita [PAULK2001].

Arendatava tarkvara kvaliteet on XP-s väga oluline. XP ideoloogid väidavad, et väiksema kui tippkvaliteediga pole rahul ei klient, ega ka arendaja. Arendajal peab olema aega teha asju kvaliteetselt. Vastasel korral kaob arendajal igasugune motivatsioon.

XP metoodika eesmärgiks on koostöös kliendiga toota võimalikult odavalt, võimalikult kiiresti ja tippkvaliteetselt kliendile kasulikku arvutitarkvara keskkonnas, kus programmeerijad ja kliendid vastastikku tunnustavad teineteise õigusi.

- **Toota kasulikku arvutitarkvara** tähendab aidata kliendil leida tema vajadusi rahuldav lihtsaim ja parim lahendus.
- **Toota kvaliteetselt** tähendab olla kindel ja veenda klienti tarkvara kui terviku ja iga selle üksiku osa töökindluses.
- **Toota kiiresti ja odavalt** tähendab kulutada aega ja raha ainult kliendile vajalike lahenduste kvaliteetseks väljatöötamiseks.

**Kliendil on õigus** [BECK2000]:

- saada maksimaalne tulemus igast tema poolt tasutud arendustegevuse nädalast;
- teada plaanidest, plaanide muudatustest ja muudatuste põhjustest: mida, millal ning milliste ressurssidega tehakse;
- näha progressi väljatöötatava süsteemi arengus ja saada tõestus süsteemi töötamise ning kvaliteedi kohta;
- muuta oma arvamusi, soove ja prioriteete liigsete lisakuludeta;
- peatada suvalisel ajal süsteemi arendus ja omandada esialgse funktsionaalsusega ning antud hetkeks valminud kasutamiskõlblik osa tarkvarast.

**Programmeerijal on õigus** [BECK2000]:

- teada kliendi tarkvaralisi soove ja prioriteete;
- teha kvaliteetsed tööd ja omada selleks aega;
- saada abi nii kaastöötajatelt, juhtidelt, kui ka klientidelt;
- püstitada ja muuta tähtaegu;
- võtta ise endale kohustusi;

XP-metoodikas tagatakse kvaliteet järgnevate tegevustega:

- dokumenteeritud ja kõikidele kohustuslik koodi kirjanemise standard;
- paaris programmeerimine;
- treeneri, testija ja kliendi rollid meeskonnas;
- kliendi kohalolek;
- meeskonna ühisvastutus, ülesannete ja soovilugude testimine;
- aja võimaldamine tarkvara kvaliteetseks realiseerimiseks.

Selleks, et rangelt SW-CMM nõudeid täita, tuleb firmal XP-metoodikat täiendada meetoditega, mis kataksid järgmised SQA nõuded:

- Me1 - Kvaliteedi tagamise tegevusi mõõdetakse ja hinnatakse nende otstarbekust;
- Ve3 - Sõltumatud eksperdid hindavad perioodiliselt kvaliteedi alast tegevust firmas.

Tabelites 16 ja 17 on toodud kokkuvõtlikult kuidas XP-metoodikaga on võimalik SW-CMM kvaliteedi tagamise eesmärgid saavutada ning vajalikud tegevused läbi viia.

	SW-CMM	XP-metoodika
1	Kvaliteedi tagamine on planeeritud.	XP eesmärk on kvaliteetne tarkvara. XP sisaldab kvaliteedi tagamise meetodeid (kliendi pidev kohalolek, testidel tuginev arendamine, arendusmetoodika järgimine).
2	Nii toodangule, kui ka tootmises kasutatavatele meetoditele antavad hinnangud peavad olema adekvaatsed.	Kliendi osalemine tarkvara väljatöötamise protsessis suurendab kliendi nõuetele vastava tarkvara väljatöötamise tõenäosust. Kõige adekvaatsema hinnangu tarkvarale saab anda klient (kas tarkvara on talle kasulik või mitte). Kõige adekvaatsema hinnangu arendustegevuses kasutatavatele meetoditele saavad anda arendajad (kas antud meetodid võimaldavad toota kliendile kasulikku ja kvaliteetset tarkvara või mitte)
3	Kvaliteedi nõuded peavad olema kõikidele teada ja kättesaadavad.	On olemas koodi standard; on olemas kohustus kirjutada teste; tarkvara peab läbima kõik testid.
4	Väljapoole nõudeid kuuluvad küsimused jäävad tippjuhtkonna lahendada.	Väljapoole nõudeid kuuluvad küsimused lahendatakse kliendi ja meeskonna vahelises koostöös. Vajadusel lahendavad küsimused treener ning suur juht.

Tabel 16. SW-CMM kvaliteedi tagamise eesmärgid ja XP-metoodika

	SW-CMM	XP-metoodika
<b>Co1</b>	Kvaliteedi tagamise aluseks on olemas dokument.	Aluseks on XP-metoodika.
<b>Ab1</b>	Kvaliteedi eest vastutajad on olemas.	Treener, klient ja testija.
<b>Ab2</b>	Ressursid ja rahalised vahendid kvaliteedi tagamiseks on olemas	Ressursid tagatakse arenduslepinguga.
<b>Ab3</b>	Inimesed on kvaliteedi tagamiseks koolitatud	Inimesed on koolitatud XP tehnikaid kasutama.
<b>Ac1</b>	Projekti kvaliteedi plaan on koostatud	Kvaliteedi plaan on koostatud

	vastavalt firmas kehtivatele dokumenteeritud nõuetele ja standarditele	vastavalt XP-meetodikale.
<b>Ac2</b>	Kvaliteedi tagamiseks on olemas plaan	Kvaliteedi tagamiseks on olemas arendustegevuse meetodid: koodi standard; koodi ülevaatamine (paaris programmeerimine), pidev integreerimine ja totaalne testimine.
<b>Ac3</b>	Kvaliteedi eest vastutavad inimesed osalevad plaanide, standardite ja protseduuride väljatöötamise ja läbivaatamise juures	Kogu meeskond osaleb plaanide, standardite ja protseduuride väljatöötamise ning läbivaatamise juures.
<b>Ac4</b>	Kvaliteedi eest vastutavad inimesed vaatavad pidevalt läbi arendustegevusi, et kindlustada vastavus nõuetega	Treener jälgib pidevalt arendustegevuste vastavust XP-meetodikale. Perioodiliselt toimuvad ühised metoodika läbivaatamised
<b>Ac5</b>	Kvaliteedi grupp auditeerib väljatöötatud tarkvara	Iga arendaja töö on teise arendaja pideva sõbraliku kontrolli ja läbivaatuse objektiks (paaris programmeerimine). Inimeste rollid meeskonnas pidevalt muutuvad. Kõik saavad kõiki auditeerida. Treener, klient ja testija moodustavad kvaliteedi grupi
<b>Ac6</b>	Kvaliteedi grupp koostab perioodilisi aruandeid ja teeb need arendusgrupile teatavaks	Nii metoodika, kui ka tarkvara ise on meeskonna pideva järelvalve ja arutluse objektiks. Metoodikat hinnatakse lähtuvalt tema toimimisest; tarkvara hinnatakse lähtuvalt kliendi rahulolust.
<b>Ac7</b>	Puudujäägid töös ja produktis kõrvaldatakse vastavalt kooskõlastatud ja dokumenteeritud protseduurireeglitele;	Puudujäägid töös ja produktis kõrvaldatakse üldjuhul kliendi ja arendajate koostööna. Kui leitakse viga tarkvaras, siis kirjutatakse selle likvideerimiseks soovilugu ja realiseeritakse see üldistel alustel.
<b>Ac8</b>	Kvaliteedi grupp teeb tihedat koostööd kliendi kvaliteedi grupiga andes ülevaateid ja koostades raporteid;	Klient on arendusmeeskonna liige. Klient otsustab ja vastutab nõuete eest ja vastuvõetava töö kvaliteedi eest. XP ei välista kliendi poolt ülevaadete ja aruannete kirjutamist. Pigem on see soovitatav, sest eeldatakse, et klient esindab firmat, kellele töö tehakse
<b>Me1</b>	Kvaliteedi tagamise tegevusi mõõdetakse ja hinnatakse nende otstarbekust;	Kvaliteedi tagamiseks eraldi spetsiaalseid mõõtmisi ei tehta. Küll aga on olemas kaudsed mõõtmised, kui eeldada, et kvaliteet on XP prioriteet.
<b>Ve1</b>	Kvaliteedi tagamise protsess on	Tippjuhi rolli täidab suur juht, kes on

	pideva tippjuhtkonna tähelepanu all;	meeskonna liige.
Ve2	Kvaliteedi tagamise igapäevast tööd korraldab projekti juht;	XP meeskonnas täidab projektijuhi rolli treener.
Ve3	Sõltumatud eksperdid hindavad perioodiliselt kvaliteedi alast tegevust firmas.	XP ei välista perioodilist kvaliteedialast auditeerimist sõltumatute ekspertide poolt.

Tabel 17. SW-CMM kvaliteedi tagamise tegevused ja XP-metoodika

## 2.4.6 Konfiguratsioonide haldamine.

Konfiguratsioonide haldamise (SCM – Software Configuration Management) otstarve SW-CMM järgi on kogu tarkvara elutsükli jooksul korrastada erinevate tarkvara toodete integreerumist tervikuks. XP-metoodika võimaldab SW-CMM konfiguratsiooni haldamise nõudeid rahuldavalt täita [PAULK2001].

XP-s on konfiguratsioonide haldamisele suunatud järgmised tegevused: koodi ühisomand, väljalasked ja pidev integreerimine.

XP-s nimetatakse väljatöötatava tarkvara konfiguratsioone väljalaseteks (*release*). Tarkvara arendatakse väljalasete kaupa.

XP-s toimub tarkvara pidev integreerimine tervikuks. Tervikuks integreeritud tarkvara on kaetud testidega. Selle aluseks on testidel tuginev arendus [BECK2003].

XP eeldab, et tarkvara integreeritakse tervikuks selleks eraldatud spetsiaalses riistvaralises keskkonnas. Seda osa riistavarast arendustegevuseks (koodi kirjutamiseks) ei kasutata. Selles keskkonnas ainult testitakse tarkvara ja hoitakse viimast töökorras versiooni. Aga samuti ka kõiki eelnevate väljalasete viimast versiooni.

Lisaks viimase töökorras versioonile tuleb hoida alles ka kõik eelnevad töökorras versioonid. Töökorras versioonide haldamiseks sobib kasutada versioonihaldustarkvara. Versioone hoitakse kas testikeskkonnas või (veel parem) selleks eraldi loodud keskkonnas.

Kuigi loodava tarkvara konfiguratsioonide haldamine on XP loomulik osa, tuleb CMM nõuete rahuldamiseks XP-d täiendada arendustegevuses kasutatava tarkvara konfiguratsioonide haldamisega.

Kasutatava tarkvara konfiguratsioonide haldamise juures on kaks olulist asja:

- kõik meeskonnaliikmed peavad arendustegevuses kasutama ühe ja sama arendustarkvara ühte ja sama versiooni;
- rakendust tuleb testida kõikides nendes operatsioonisüsteemides ja nende versioonides, kus rakendust planeeritakse reaalselt kasutada.

SW-CMM nõuded, milles tuleb XP-metoodikat kasutaval firmal täiendavalt kasutatava tarkvara konfiguratsioonide haldamise meetodeid välja töötada on:

- Co1 - Konfiguratsioonide (nii loodava kui ka kasutatava tarkvara) haldamise aluseks firmas on vastav dokument;
- Ab1 - Konfiguratsioonide haldamine on allutatud kontrollile ning on olemas selle eest vastutavad isikud
- Ab5 - Tarkvara arendusmeeskonna liikmed ja teised asjast huvitatud osapooled on koolitatud konfiguratsioonide haldamise vajadustest lähtuvalt
- Ac1 - Iga tarkvaraprojekti jaoks on koostatud konfiguratsioonide haldamise plaan vastavalt firmas kokkulepitud ja dokumenteeritud protseduuri reeglitele;
- Ac5 - Muudatuste nõuded ja probleemid iga konfiguratsioonidega seotud küsimuse korral on määratud, fikseeritud, läbi vaadatud, kinnitatud ning säilitatud vastavalt dokumenteeritud protseduuridele;
- Ac6 - Konfiguratsioonide muudatused viiakse läbi ja kontrollitakse vastavalt dokumentides kehtestatud korrale;
- Ac8 - Konfiguratsioonide kohta peetakse arvet vastavalt dokumenteeritud protseduuridele;
- Me1 - Konfiguratsioonide haldamise tegevusi mõõdetakse;

Täiesti on XP-metoodika poolt katmata järgmised SW-CMM konfiguratsioonide haldamise nõuded:

- Ac4 - On määratud tarkvara tooted, millede konfiguratsioonid peavad olema hallatud;
- Ac9 - Standardsed raportid konfiguratsiooni muudatuste teavitamise kohta on välja töötatud ja asjaosalistele kättesaadavaks tehtud;
- Ac10 - Tarkvara konfiguratsioonide audit viiakse läbi vastavalt dokumentaalselt kehtestatud korrale.

Tabelites 18 ja 19 on toodud kokkuvõtlikult kuidas XP-metoodikaga on võimalik SW-CMM konfiguratsioonide haldamise eesmärgid saavutada ning vajalikud tegevused läbi viia.

	<b>CMM</b>	<b>XP</b>
1	Konfiguratsioonide haldamise tegevused on planeeritud.	Väljatöötatava tarkvara erinevaid konfiguratsioone XP-s nimetatakse väljalaseteks. XP-metoodikas arendatakse tarkvara väljalasete kaupa. Väljalasete haldamine on planeeritud
2	Erinevad tarkvara osad on määratletud, kontrollitavad ja kättesaadavad.	XP eeldab pidevat integreerimist tervikuks ja töökorras versioonide säilitamist.
3	Muudatused konfiguratsioonides on kontrolli all.	Väljalasete ja versioonide sisu on teada realiseeritud ja teste läbinud soovilugude ning ülesannete järgi.
4	Asjasse pühendatud isikud ja grupid on informeeritud olukorrast.	Arendusmeeskond (ja klient selle liikmena) on teadlik jooksvast (ja ka eelnevatest) väljalasetest ning versioonidest.

*Tabel 18. SW-CMM konfiguratsioonide haldamise eesmärgid ja XP-metoodika*



	<b>CMM</b>	<b>XP</b>
<b>Co1</b>	Konfiguratsioonide (nii loodava kui ka kasutatava tarkvara) haldamise aluseks firmas on vastav dokument;	XP-metoodika eeldab väljatöötatava tarkvara konfiguratsioonide haldamist. Kasutatava tarkvara haldamine nõuab eraldi reguleerimist
<b>Ab1</b>	Konfiguratsioonide haldamine on allutatud kontrollile ning on olemas selle eest vastutavad isikud	Loodava tarkvara konfiguratsioone haldab testija. Testija rolli laiendamine ka kasutatava tarkvara konfiguratsioonide haldamisele lahendab olukorra.
<b>Ab2</b>	Iga projekti juures on loodud grupp, kes koordineerib ja viib läbi konfiguratsioonide alast tööd	Grupi suurust CMM ei fikseeri. Grupp võib koosneda ka ühest osalise koormusega inimesest. Testija rolli laiendamine lahendab olukorra.
<b>Ab3</b>	On eraldatud adekvaatsed ressursid ja rahalised vahendid selleks, et konfiguratsioonide haldamisega seotud tegevusi läbi viia	XP eeldab minimaalselt versioonihaldus tarkvara. Rahalised vahendid tuleb tagada tarkvara arenduslepingutega.
<b>Ab4</b>	Konfiguratsioonide haldamisega ja konfiguratsiooni muudatuste läbiviimisega tegelevad inimesed on saanud vastava koolituse	Testija (või keegi teine meeskonnast) peab saama vastava koolituse.
<b>Ab5</b>	Tarkvara arendusmeeskonna liikmed ja teised asjast huvitatud osapooled on koolitatud konfiguratsioonide haldamise vajadustest lähtuvalt	Väljatöötatava projekti konfiguratsioonihaldus on XP osa. Arenduses kasutatava tarkvara konfiguratsioonihaldus tuleb täiendusena firma XP metoodikasse sisse viia.
<b>Ac1</b>	Iga tarkvaraprojekti jaoks on koostatud konfiguratsioonide haldamise plaan vastavalt firmas kokkulepitud ja dokumenteeritud protseduuri reeglitele	On olemas väljalasete valmimise plaan ja ülesannete tervikuks integreerimise reeglistik. Arendustegevuses kasutatava tarkvara versioonide muutmise reeglid tuleb aga firmal välja töötada.
<b>Ac2</b>	Tarkvaraprojekti konfiguratsiooni plaan on aluseks konfiguratsioonidega seotud tegevuste läbiviimisel	Arendustegevuse plaan on aluseks väljalaske versioonide.
<b>Ac3</b>	On olemas konfiguratsioonide haldamise süsteem ja inimeste oskused seda süsteemi kasutada	XP-metoodika eeldab versioonihaldustarkvara olemasolu ja kasutamist arendajate poolt.
<b>Ac4</b>	On määratud tarkvara tooted, millede konfiguratsioonid peavad olema hallatud;	Selline nimekiri tuleb firmal koostada
<b>Ac5</b>	Muudatuste nõuded ja probleemid iga konfiguratsioonidega seotud küsimuse korral on määratud, fikseeritud, läbi vaadatud, kinnitatud ning säilitatud vastavalt dokumenteeritud protseduuridele;	Kasutatava tarkvara jaoks tuleb selline kord lisaks XP meetoditele kehtestada. Arendatava tarkvara konfiguratsioonide muudatused alluvad tavalisele XP soovilugude realiseerimise meetodile.

Ac6	Konfiguratsioonide muudatused viiakse läbi ja kontrollitakse vastavalt dokumentides kehtestatud korrale;	Kasutatava tarkvara jaoks tuleb selline kord lisaks XP meetoditele kehtestada. Arendatava tarkvara konfiguratsioonide muudatused alluvad tavalisele XP soovilugude realiseerimise meetodile.
Ac7	Väljatöötatud tarkvara konfiguratsioonide kontroll ja evitamine on allutatud dokumenteeritud protseduuriereglitele;	On XP-metoodika osa ja tagatud tervikuks integreerimise ning testidel tugineva arendusega.
Ac8	Konfiguratsioonide kohta peetakse arvet vastavalt dokumenteeritud protseduuridele;	Kasutatava tarkvara jaoks tuleb selline kord lisaks XP meetoditele kehtestada. Arendatava tarkvara konfiguratsioonide muudatused alluvad tavalisele XP soovilugude realiseerimise meetodile.
Ac9	Standardsed raportid konfiguratsiooni muudatuste teavitamise kohta on välja töötatud ja asjaosalistele kättesaadavaks tehtud;	Selline kord tuleb sisse viia.
Ac10	Tarkvara konfiguratsioonide audit viiakse läbi vastavalt dokumentaalselt kehtestatud korrale.	Selline kord tuleb kehtestada.
Me1	Konfiguratsioonide haldamise tegevusi mõõdetakse;	Arendatava tarkvara konfiguratsiooni haldamise tegevusi mõõdetakse. Kasutatava tarkvara jaoks tuleb meetrika kehtestada.
Ve1	Konfiguratsioonide haldamine on allutatud tippjuhtkonna pideva tähelepanu alla;	Tippjuhi rolli täidab suur juht, kes on meeskonna liige.
Ve2	Konfiguratsioonide haldamise igapäevaste tegevustega tegeleb projekti juht;	XP meeskonnas täidab projektijuhi rolli treener.
Ve3	Konfiguratsioonide eest vastutav grupp auditeerib perioodiliselt kasutatavat tarkvara selleks, et veenduda dokumentide vastavust reaalsusele;	Tarkvara konfiguratsioonid on määratavad ja kontrollitavad testidega. Seega auditeeritakse konfiguratsioonide vastavust pidevalt uute ülesannete tervikuks integreerimise ajal.
Ve4	Tarkvara kvaliteedi grupp jälgib ja auditeerib perioodiliselt konfiguratsiooniga seotud tegevuste vastavust reaalsele vajadustele.	XP-metoodika jälgimine ja selle arutelud on XP-metoodika osa.

Tabel 19. SW-CMM konfiguratsioonide haldamise tegevused ja XP-metoodika

### **3 EMPIIRILINE UURING - EESTI ARENDAJAD, CMM JA XP-METOODIKA**

Selleks, et saada empiirilise uuringu jaoks lähteandmeid, oli vaja läbi viia vastav küsitlus.

#### **3.1 Küsitluse eesmärk**

Küsitluse eesmärgiks oli saada ülevaade tarkvara arendamise metoodikate kasutamise tegelikust olukorrast Eestis ning arendustegevusega seotud inimeste ootustest ja hoiakutest tarkvara arendamise metoodikate suhtes.

Konkreetselt:

1. Kas eesti tarkvaraarendajad peavad SW-CMM (Capability Maturity Model for Software) [CMM] teise taseme nõudeid tarkvaraarenduse seisukohalt oluliseks.
2. Mil määral on eesti tarkvaraarendajad oma praktilises arendustegevuses SW-CMM teise taseme nõuete järgimist kohanud või neid ise järginud.
3. Kas eesti tarkvaraarendajad peavad XP-metoodikas (*Extreme programming*) [BECK2001] kasutatavaid meetodeid tarkvaraarenduse seisukohalt oluliseks .
4. Mil määral on eesti tarkvaraarendajad oma praktilises arendustegevuses XP meetodite järgimist kohanud või neid ise järginud.
5. Kuivõrd arvestatavad on eesti tarkvaraarendajate hulgas hirmud, mida Kent Beck peab arendajate põhihirmudeks [BECK2001].

Nendele küsimustele vastuse saamiseks koostasın küsimused lähtuvalt SW-CMM teisest tasemest ning XP-metoodikas kasutatavatest meetoditest. Küsimustele palusin vastata tarkvara arendamisega seotud inimestel.

#### **3.2 Küsimustik**

Küsimustik koosnes neljast osast (vt. Lisa 3).

##### **3.2.1 Taustinformatsiooniga seotud küsimused**

Esimeses osas on küsimused, mis on seotud tarkvara arendajate igapäevase tegevuse ja kogemustega. Selle osa eesmärgiks on saada taustinformatsiooni ankeedile vastanute kohta. Soovisin teada, kas vastaja on tippjuht, projektijuht või arendaja. Millised on vastaja igapäevased põhitegevused? Palusin vastajal hinnata oma teadmisi tarkvara arendusprotsessidest ja metoodikatest. Tundsin huvi vastaja tööalase kogemuse, hariduse, arendajate arvu kohta firmas ja tüüpilises arendusprojektis ning palusin kirja panna firmas kasutatava või planeeritava arendusmetoodika.

##### **3.2.2 Arendusprotsessiga seotud küsimuste koostamise põhimõtted**

Teises osas on üldised tarkvara arendamise protsessiga seotud küsimused. Teise osa küsimuste koostamisel püüdsin lähtuda SW-CMM teise taseme nõuetest tarkvara arendusprotsessile. Täielikult on välja jäetud need SW-CMM teise taseme nõuded,

mis on seotud allhangete osaga (*SSM - Software Subcontract Management*) tarkvara arendamises. Allhangete väljajätmise põhjuseks on asjaolu, et allhangete osa pole nendele arendusfirmadele kohustuslik, kes alltöövõtjaid ei kasuta. Antud töös lähtun aga igal pool sellisest kitsendusest arendusfirmale.

Seega puudutavad teise osa küsimused alljärgnevaid SW-CMM teise taseme protsesse (*Key Process Area*):

1. Nõuete haldamine (*RM - Requirements Management*).
2. Tarkvaraprojekti kavandamine (*SPP - Software Project Planning*)
3. Tarkvaraprojekti monitooring (*PTO - Software Project Tracking and Oversight*)
4. Tarkvara kvaliteedi tagamine (*SQA - Software Quality Assurance*)
5. Tarkvara konfiguratsioonide haldamine (*SCM - Software Configuration Management*)

Kuna SW-CMM teise taseme nõudeid (mõistes tegevused, mida arendusmeeskond peab tegema või jälgima) on liiga palju (kokku 121; ilma SSM nõueteta 99) ja vabatahtliku küsimustiku maht suhteliselt piiratud, üritasin võimalikult väikese küsimuste arvuga haarata maksimumi. Kuna kõikide SW-CMM protsesside nõuded on sarnaselt struktureeritud ning paljudes asjades dubleerivad oli SW-CMM nõuete kokkusurumine vastavalt antud küsimustiku eesmärkidele (pildi saamine arendajate hoiakutest ja kogemustest) suhteliselt lihtne. Kokkusurumist hõlbustas asjaolu, et ...

1. Igas protsessis on 1-2 tegevuskavaga (*Co – Commitment to perform*) seotud nõuet, millest üks nõuab protseduuri olemasolu ja teine (kui on) vastutaja olemasolu. Küsimustiku teise osa küsimused 1) ja 3) on seotud tegevuskavaga.
2. Igas protsessis on 3-5 ressursside (*Ab – Ability to perform*) olemasolu nõudvad nõuded. Ressursside all mõeldakse nii raha, aega, töövahendeid kui ka inimeste koolitust. Küsimustiku teise osa küsimused 1), 2) ja 3) on seotud selle osaga.
3. Suurem osa protsessi spetsiifilisi nõudeid langeb tegevuste (*Ac – Activities to perform*) osasse. Selle osa nõudeid oli kõige raskem kokku suruda.
  - Küsimustiku teise osa küsimused 6), 7) ja 8) on koostatud nii, et nad kataksid nõuete haldamise (*RM*) tegevused Ac1 kuni Ac3 .
  - Küsimustiku teise osa küsimused 9) kuni 16) on koostatud nii, et nad kataksid arendusprotsessi planeerimise (*SPP*) tegevused Ac1 kuni Ac15
  - Küsimustiku teise osa küsimused 9) kuni 20) on koostatud nii, et nad kataksid ka olulisema osa arendusprotsessi jälgimise (*PTO*) tegevustest Ac1 kuni Ac13
  - Küsimustiku teise osa küsimused 18) kuni 21) on koostatud nii, et nad kataksid kvaliteedi tagamise (*SQA*) tegevused Ac1 kuni Ac8
  - Küsimustiku teise osa küsimused 22) kuni 24) on koostatud nii, et nad kataksid konfiguratsioonide haldamise (*SCM*) tegevused Ac1 kuni Ac10.

4. Igas protsessis on 1 mõõtmistega (*Me – Measurement and Analysis*) seotud nõue. Küsimustiku teise osa küsimus 4) on seotud mõõtmistega.
5. Igas protsessis on 3 - 4 kontrolli ja järeluste tegemisega (*Ve – Verifying implementation*) seotud nõuet. Küsimustiku teise osa küsimused 4) ja 5) katavad kontrolli ja järeluste nõudeid.

### 3.2.3 Arendusprotsessi meetoditega seotud küsimused.

Kolmanda osa küsimuste aluseks olid põhiliselt arendusmetoodikas XP (*Extreme Programming*) [BECK2001] kasutatavad meetodid. Kokku on arendusprotsessi meetoditega seotud osas 16 küsimust.

### 3.2.4 Teise ja kolmanda osa vastusevariandid

Igale teise (arendusprotsess) ja kolmanda (arendusmeetodid) osa küsimusele pidi vastaja andma hinnangu kahel erineval skaalal.

1. Kas tegevus on projektile edu tagamiseks oluline, ebaoluline või pidurdav?
2. Kas firmas, kus arendaja töötab, on antud põhimõtte/tegevus tarkvara arendusprojektides tavaline, harvaesinev või mitte kasutatav.

### 3.2.5 Arendajate hirmudega seotud küsimused

Kent Beck väidab oma raamatus “Extreme Programming Explained, Embrace Change” (Addison-Wesley, 2000), et tarkvara arendusmetoodikad püüavad taltsutada arendustegevusega seotud hirme selleks, et võimaldada tarkvara arendajatele rahulik uni. Arendajate hirmudega seotud osas oli 9 küsimust.

Igale küsimusele sai vastata kas “Tihti”, “Harva” või “Mitte kunagi”.

## 3.3 Küsitluse läbiviimine.

Elektroonne küsimustik oli üles pandud aadressile <http://cafe.ee/~rasmus/gunnar/>.

Kuna sooviks oli saada võimalikult paljude tarkvara arendusega tegelevate inimeste hinnanguid ja arvamusi võimalikult erinevatest firmadest ja võimalikult erinevate kogemustega, levitasin informatsiooni ankeedi asukoha kohta koos palvega leida aega ning vastata minu küsimustele alljärgnevaid kanaleid pidi:

1. Tuttavate arendajate hulgas;
2. Õpingukaaslaste hulgas;
3. Uudistegruppides ee.prog ja ee.prog.delphi;
4. Listis [teadus.tcs@lists.ut.ee](mailto:teadus.tcs@lists.ut.ee)
5. Listis [eits-liikmed@fut.ee](mailto:eits-liikmed@fut.ee)
6. Personaalne e-kiri aktiivsetele EITS liikmetele

Tuttavad arendajad ja õpingukaaslased olid esimesed, kellele oma palve ankeedi levitamiseks esitasin. Seda kanalit pidi laekus ligikaudu paarkümmend vastatud ankeeti. EITS (Eesti Infotehnoloogia Selts) listis teavitamise ja aktiivsetele liikmetele personaalse e-kirja saatmise tulemusena laekus hinnanguliselt umbes kümmekond

tööd. Üksikud ankeedid laekusid listis teadus.tcs teavitamise järel. Kõige suurema vastukaja sain aga informatsiooni levitamise tulemusena eesti programmeerijatele mõeldud uudistegruppides ee.prog ja ee.prog.delphi. Siit laekus hinnanguliselt umbes nelikümmend tööd.

Ankeedid laekusid elektroonselt minu emaili aadressile.

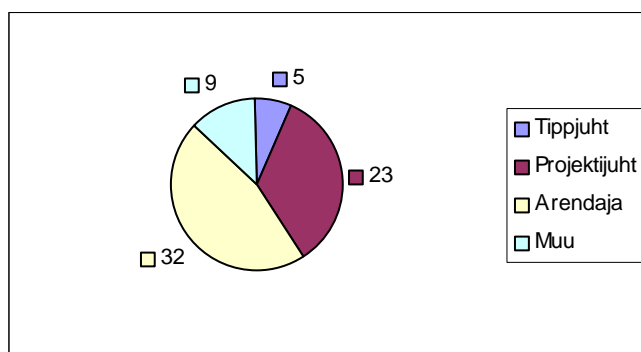
### 3.4 Andmete töötlemise vahendid

Andmete töötlemiseks kasutasin vahendeid MS EXCEL ja SPSS 11.5 for Windows.

### 3.5 Vastajad ja nende taust

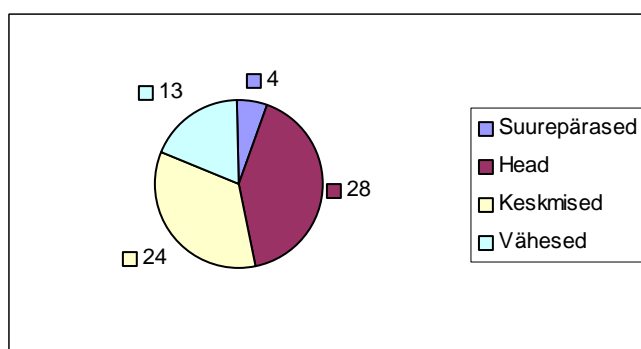
Kokku vastas minu poolt koostatud küsimustele 69 inimest. Vastuseid laekus 38-lt erinevalt interneti aadressilt Põhiliselt laekus aadressilt üks kuni kaks ankeeti. Ühelt aadressilt (üldkasutatav estpak.ee) laekus üksteist ankeeti (nendest 6 oli minu poolt sisestatud paberkandjal laekunud vastust) ja ühelt aadressilt (kuulub ühele tarkvarafirmale) laekus 10 ankeeti. Joonised 6 kuni 14 illustreerivad vastajate tausta.

Arendajaid ja juhte (tipp- ning projektijuhte kokku) oli vastanute hulgas enamvähem võrdselt.



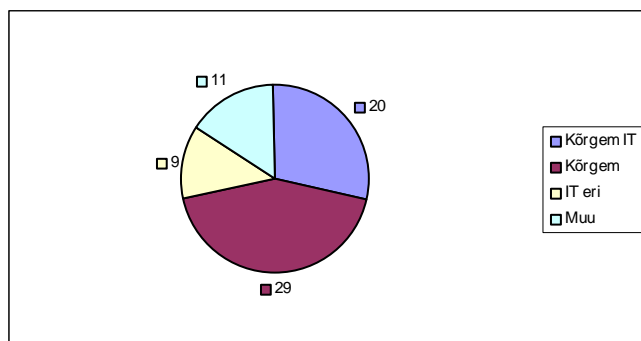
Joonis 6. Ankeedile vastanute amet.

Enamik vastanuid hindas oma teadmisi arendusmetoodikatest kas headeks või keskmisteks.



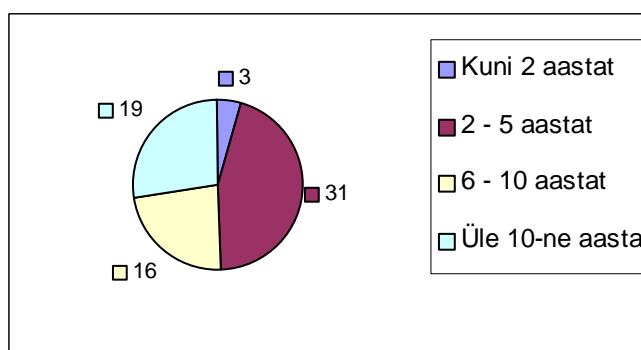
Joonis 7. Ankeedile vastanute teadmised metoodikast (enesehinnang).

Ligi kolmveerand vastanutest omas kõrgemat haridust ja ligi pooltel oli IT haridus (kas kõrgem- või eriharidus).



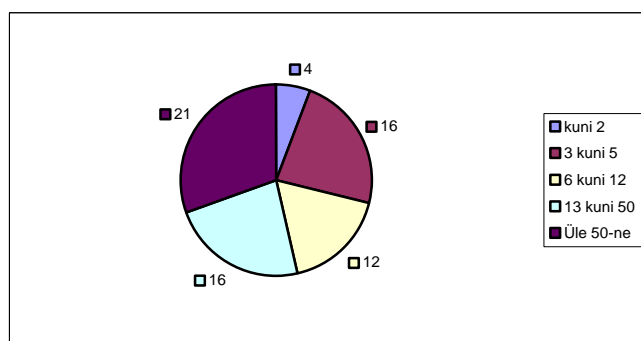
Joonis 8. Ankeedile vastanute haridus.

Umbes pooltel oli arendusalast kogemust kuni 5 aastat ja pooltel rohkem kui viis aastat. Umbes veerand vastanutest omas rohkem kui kümneaastast arendusalast kogemust.



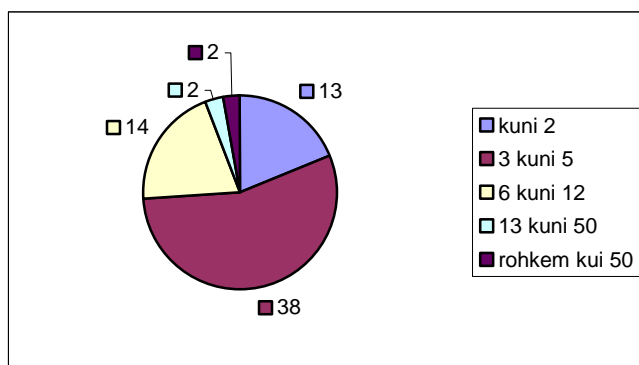
Joonis 9. Ankeedile vastanute arendusalane kogemus.

Kui võtta väikeste, keskmiste ja suurte firmade piirideks vastavalt 5 ja 50 arendajat, siis kolmandik vastanutest oli väikefirmadest, kolmandik keskmise suurusega firmadest ja kolmandik suurfirmadest.



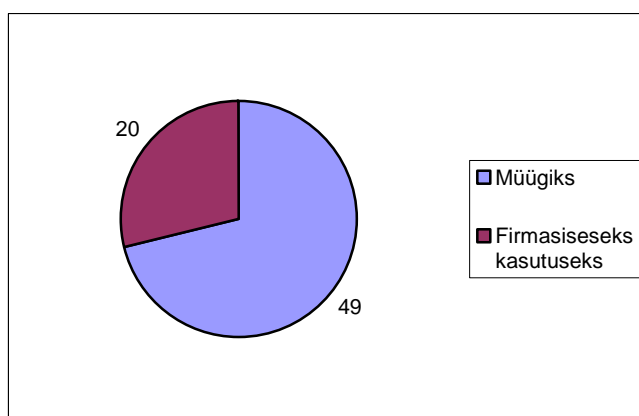
Joonis 10. Arendajaid firmas, kus ankeedile vastanu töötab.

Suuremale osale arendusprojektidest tundub aga XP-metoodika vähemalt arendajate arvu vaadates sobivat. Üle 12-ne arendajaga projekte on marginaalne osa ja rohkem kui pooltes arendusprojektide osaleb 3 kuni 5 arendajat.



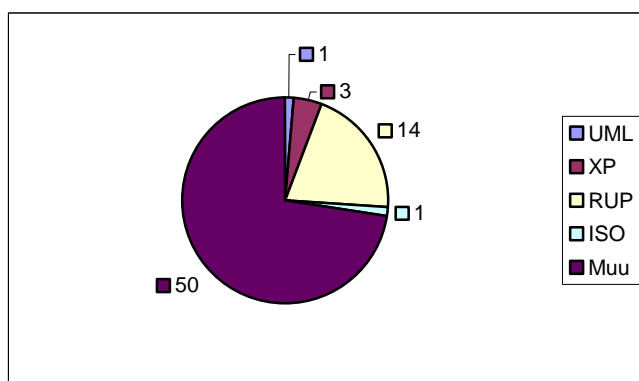
Joonis 11. Arendajaid keskmiselt arendusprojektis, kus ankeedile vastanu töötab.

Kolmandik arendajaid töötab firmades, kus tarkvara tehakse firmasiseseks kasutamiseks ja 2/3 firmadest toodab tarkvara müügiks.



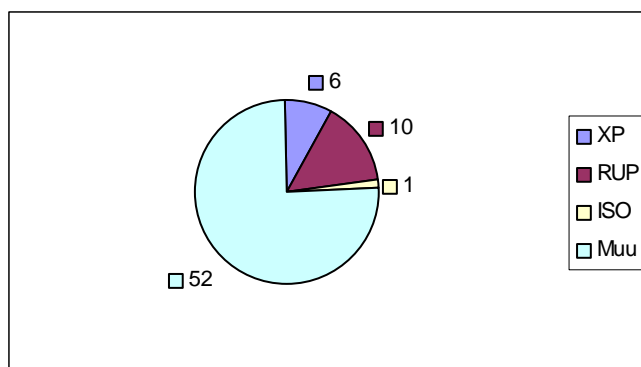
Joonis 12. Tarkvara arendamise eesmärk ankeedile vastanu firmas.

Kolmveerand vastanutest leidis, et nende firmas ei kasutata ega plaanita ka lähiajal juurutada mingit metoodikat. Metoodikatest valitseb RUP. Ka XP-metoodika on nähtaval (sektoridiagrammid on tehtud saadud vastuste alusel).





Joonis 13. Kasutatav arendusmetoodika ankeedile vastanu firmas.



Joonis 14. Planeeritav arendusmetoodika ankeedile vastanu firmas.

### 3.6 Küsitlusele vastanute rühmitamine

Esimene ülesanne, mille ma ankeetide töötlemisel endale püstitasin, oli rühmitada vastajaid nende taustandmete järgi. Hüpotees oli, et tippjuhtidel ja projektijuhtidel võiks olla IT alane kõrgharidus ja suuremad arendusalased kogemused.

Kuna objekte oli suhteliselt vähe, kasutasin hierarhilist klasterdamist. Lähtuvalt Katrin Niglase koostatud klasteranalüüsi juhendist [NIGLAS] valisin klasterdamise matemaatiliseks meetodiks klasterdamise mediaani järgi (*Median clustering*) ning sobivaks sarnasuse ehk kauguse mõõduks valisin eukleidese ruutkauguse (*Squared Euclidean distance*). Kuna kõik klasterdamise aluseks olevad tunnused (haridus, kogemused, arendajate arv firmas ja gruppis, amet ning enesehinnang metoodikate tundmises) on mõõdetud järjestatavatel, kuid erinevatel, skaaladel – valisin vastavalt eelmainitud juhendile “Z-scores” väärtuste standardiseerimise meetodiks. Tunnuseid, mis ei ole mõõdetud järjestataval skaalal (metoodika kasutamisega või plaaniga metoodika lähiajal juurutada, põhitegevused) rühmitamise juures ei ma ei kasutanud. Objektid lasin paigutada 10-sse gruppi.

Tulemuseks oli kaks suurt (ühes 27 teises 28 vastanut) ja 8 marginaalset rühma (Lisa.4: Vastanute rühmitamine taustandmete järgi).

Gruppide võrdlemisel (vaata Lisa.4: Vastanute taustatunnused gruppides) selgus, et esimeses grupis (kokku 27 vastanut) on valdavalt IT kõrgharidusega, suuremate kogemustega (vähemalt 6 aastat), põhiliselt tipp- või projektijuhid. Selle grupi enesehinnang metoodikate tundmises oli suhteliselt kõrge (hinnatud kas “headeks” või “suurepäraseks”). Sellele grupile panin nimeks JUHID.

Teises grupis (kokku 28 vastanut) on valdavalt mitte IT kõrgharidusega, väiksemate kogemustega (kuni 5 aastat) põhiliselt arendajad. Selle grupi enesehinnang metoodikate tundmises oli suhteliselt tagasihoidlik (hinnatud kas “vähesed” või “keskmised”). Sellele grupile panin nimeks AREDAJAD.

Kolmandasse gruppi panin kõik objektid, kes esimesse ja teise gruppi ei sobinud. Sellele grupile panin nimeks TEISED.

Kokkuvõtet rühmitamisest vaata alajaotusest 3.9 ja tabelist 25.

### 3.7 Andmete töötlemine

Eesmärgiga vähendada analüüsitavate tunnuste arvu üritasin grupeerida ja ühendada omavahel need tunnused, mis vastajate hinnangul on arendusprotsessile edu tagamise seisukohalt võrdse kaaluga või mida reaalses arendusprotsessides ühesuguse sagedusega kohtab.

Ka siin kasutasin hierarhilist klasterdamist. Lähtuvalt eesmärgist valisin klasterdamise matemaatiliseks meetodiks klasterdamise mediaani järgi (*Median clustering*) ning sobivaks sarnasuse ehk kauguse mõõduks eukleidese ruutkauguse (*Squared Euclidean distance*).

#### 3.7.1 Arendusprotsessiga seotud küsimuste rühmitamine

Arendusprotsessiga seotud küsimuste rühmitamine vastuste järgi, kus vastanud hindasid antud väidet skaalal – arendusprotsessi eduks “oluline”, “mitteoluline” või “pidurdav” – tulemust, mida ma interpreteerida oskaks või mille järgi tunnuseid grupeerida saaks, ei andnud (Vaata Lisa.4: Küsimuste “oluline arendusprotsessis” rühmitamine). Põhjus selles, et peaaegu kõik vastanutest peavad kõiki väiteid “oluliseks” (Tabel 20). Selline antud küsimustele vastamine oli ka eeldatav, kuna küsimused sisaldasid CMM nõudeid. Kaks kõige vähem oluliseks peetud küsimust olid seotud sõltumatu kvaliteedi tagamise ning konfiguratsioonide auditeerimisega.

Vastajate arv, kes pidasid väidet oluliseks		Hinnangu saanud väidete arv
Absoluutarv	Vastajate osakaal	
69	100 %	8
68	98,6 %	3
67	97 %	4
66	95,7 %	4
65	94,2 %	2
63	91 %	1
59	85,5 %	1
58	84,1 %	1

Tabel 20. Vastajate arv, kes pidasid tegevust arendusprotsessis oluliseks

Arendusprotsessiga seotud küsimuste rühmitamine vastuste järgi, kus vastanud hindasid antud väidet skaalal - firmas, kus arendaja töötab, kasutatakse antud meetodit tarkvaraarenduses “tihti”, “harva” või “mitte kunagi” – andis aga võimaluse tunnuste rühmitamiseks. (Lisa.4: Küsimuste “arendusprotsessis kasutatakse” rühmitamine).

Moodustasin järgmised tunnuste grupid (Tabel.21) nii, et tunnuse grupi väärtuseks sai temasse kuuluvate osatunnuste väärtuste keskmine.

Tunnustegrupp “nõuete haldamine” on heas loogilises seoses ka küsimuste koostamise põhimõtetega. Kuna küsimused (vt. 3.2.2.) 6) – 8) on koostatud nii, et nad kataksid CMM nõuete haldamise põhimõtteid. Küsimus 3) (projektijuht) kuulub sisuliselt kõikide CMM osade juurde. Erinevus on ainult küsimuses 10), mis küsimustiku koostamise põhimõtteid arvestades oleks pidanud sattuma planeerimise

gruppi. Tema sattumise põhjuseks nõuete haldamise rühma pean ka üheks arendusprotsessi küpsuse näitajaks (sellest aga pikemalt järeltunde osas).

Tunnuste rühma “Projekti planeerimine” paigutatud küsimused on peamiselt seotud projekti planeerimisega. Lisaks veel protseduurireeglite ja ressursside nõue (küsimus 1) ning inimeste koolitamise nõue (küsimus 2), mis on SW-CMM järgi sisuliselt seotud kõikide alamprotsessidega tarkvara arenduses.

Tunnuste rühma “Tulemuste jälgimine” paigutatud küsimused on seotud arendusprotsessi lõpptulemustega – tuntakse huvi toodangu kvaliteedi, tema vastavuse kohta püstitatud nõuetele ja versioonide ning konfiguratsioonide vastu. SW-CMM järgi kuuluvad antud nõuded kvaliteedi tagamise ning konfiguratsioonide haldamise osaprotsesside koosseisu.

Tunnusterühma “Projekti monitooring” paigutatud küsimused on seotud arendusprojekti enese järgimisega. Kas plaanidest peetakse kinni, kuhu on jõutud. Rühma “Tulemuste audit” tunnuseks on sõna “audit” kvaliteedi tagamise ja versioonide haldamise protsessis. Projektis kasutatava meetodika läbivaatamine ja selle muutmine vastavalt vajadustele ühegi teise tunnustega kokku ei sobinud.

<b>Nõuete haldamine</b>	<ul style="list-style-type: none"> <li>• <b>Projektil on juht</b> ( Küsimus: 3)</li> <li>• <b>Nõudeid hallatakse</b> (6).</li> <li>• <b>Nõudeid analüüsitakse</b> (7)</li> <li>• <b>Nõuete muudatusi arvestatakse</b> (8).</li> <li>• <b>Arendajad osalevad aruteludes</b> (10).</li> </ul>
<b>Projekti planeerimine</b>	<ul style="list-style-type: none"> <li>• <b>Protseduurireeglid ja ressursid on olemas</b> (1).</li> <li>• <b>Inimesed on koolitatud</b> (2).</li> <li>• <b>Hallatavad elutsüklid</b> (9).</li> <li>• <b>Plaan on dokument</b> (11).</li> <li>• <b>Plaanis kajastuvad vajadused</b> (12).</li> <li>• <b>Kõik on plaanidega nõus</b> (14).</li> </ul>
<b>Tulemuste jälgimine</b>	<ul style="list-style-type: none"> <li>• <b>Kvaliteedinõuded on olemas</b> (18).</li> <li>• <b>Kvaliteedi eest vastutajad on olemas</b> (19).</li> <li>• <b>Kvaliteedi eest vastutajad teevad tööd</b> (20).</li> <li>• <b>Versioonide plaanid on olemas</b> (22).</li> <li>• <b>Versioone ja konfiguratsioone hallatakse</b> (23).</li> </ul>
<b>Projekti monitooring</b>	<ul style="list-style-type: none"> <li>• <b>Mõõtmisi sooritatakse</b> (4).</li> <li>• <b>Reeglid plaanide hindamiseks on olemas</b> (13).</li> <li>• <b>Riske hallatakse</b> (15).</li> <li>• <b>Reeglid plaanide muutmiseks on olemas</b> (16).</li> <li>• <b>Korrigeerimisreeglid on olemas</b> (17).</li> </ul>
<b>Tulemuste audit</b>	<ul style="list-style-type: none"> <li>• <b>Kvaliteediaudit</b> (21).</li> <li>• <b>Versioonide ja konfiguratsioonide audit</b> (24).</li> </ul>
<b>Meetodika läbivaatamine</b>	<ul style="list-style-type: none"> <li>• <b>Meetodika läbivaatamine</b> (5)</li> </ul>

Tabel 21. Uute tunnuste moodustamine arendusprotsessiga seotud küsimuste järgi (number näitab küsimuse numbrit küsimustikus )

### 3.7.2 Arendusmeetoditega seotud küsimuste rühmitamine

Arendusmeetoditega seotud küsimuste rühmitamine vastuste järgi, kus vastanud hindasid antud väidet skaalal – arendusprotsessi eduks “oluline”, “mitteoluline” või “pidurdav” – ei andnud samuti mingit sellist tulemust (Vaata Lisa.4: Küsimuste “oluline arendusmeetodika” rühmitamine). Põhjus selles, et vastanute hinnangud langesid peaaegu kokku (Tabel.22). Kõige kehvema hinnangu sai paarisprogrammeerimine (37 olulist) ning “pidevate kanglastegude tegemine” (46 olulist).

Vastajate arv, kes pidasid väidet oluliseks		Hinnangu saanud väidete arv
Absoluutarv	Vastajate osakaal	
68	98,6 %	2
67	97,1 %	1
65	94,2 %	1
64	92,8 %	2
63	92,3 %	2
61	88,4 %	1
59	85,5 %	2
58	84,1 %	1
57	82,6 %	1
56	81,2 %	1
46	66,7 %	1
37	53,6 %	1

Tabel 22. Vastajate arv, kes pidasid meetodeid oluliseks

Arendusmeetoditega seotud küsimuste rühmitamine vastuste järgi, kus vastanud hindasid antud väidet skaalal - firmas, kus arendaja töötab, kasutatakse antud meetodit tarkvaraarenduses “tihti”, “harva” või “mitte kunagi” – andis aga võimaluse küsimuste rühmitamiseks. (Vaata Lisa.4: Küsimuste “meetodikat kasutatakse” rühmitamine).

Moodustasin järgmised tunnuste grupid (Tabel.23) nii, et tunnuse väärtuseks saab temasse kuuluvate osatunnuste väärtuste keskmine.

Rühma “XP meetodid” paigutasid küsimused meetodite kohta, mida võiks nimetada XP-metoodika tuumik meetoditeks.

Rühma “Tavalised meetodid” paigutasid küsimused, mis minu arust on igas vähegi arvestatavas ja kogemustega tarkvara arendusfirmas (kogemustega tarkvaraarendaja poolt) kasutusel.

Rühma nimetust “Häkkeri meetodid” ei tule võtta mingil juhul halvustavalt.

Koodi standardite ja paarisprogrammeerimise meetodid ühegi teise tunnusega kokku ei sobinud.

XP meetodid	<ul style="list-style-type: none"> <li>• <b>Soovilood</b> (Küsimus 4).</li> <li>• <b>Testid</b> (9).</li> <li>• <b>Pidev Integreerimine</b> (10)</li> <li>• <b>Ühiskood</b> (6).</li> <li>• <b>Ümberkirjutamine</b> (12).</li> </ul>
Tavalised meetodid	<ul style="list-style-type: none"> <li>• <b>Lihtne disain</b> (11) .</li> <li>• <b>Optimeeri lõpus</b> (14) .</li> <li>• <b>Kliendi osavõtt</b> (3) .</li> <li>• <b>Pühendunud inimesed</b> (15) .</li> <li>• <b>Lühikesed väljalasked</b> (8) .</li> </ul>
Häkkeri meetodid	<ul style="list-style-type: none"> <li>• <b>Pidevad kangelasteod</b> (16) .</li> <li>• <b>Lihtsad vahendid</b> (13) .</li> </ul>
Arukad meetodid	<ul style="list-style-type: none"> <li>• <b>Ületundide vältimine</b> (1) .</li> <li>• <b>Süsteemi metafoor</b> (2) .</li> </ul>
Koodi standard	<ul style="list-style-type: none"> <li>• <b>Koodi standard</b> (5).</li> </ul>
Paarisprogrammeerimine	<ul style="list-style-type: none"> <li>• <b>Paarisprogrammeerimine</b> (7)</li> </ul>

Tabel 23. Uute tunnuste moodustamine arendusmeetoditega seotud küsimuste järgi (number näitab küsimuse numbrit küsimustikus )

### 3.7.3 Küsimuste “arendajate hirmud” rühmitamine.

Klasteranalüüsi tulemusena (Vaata Lisa.4) moodustan järgmised tunnuste grupid (Tabel 24) nii, et tunnuse väärtuseks saab temasse kuuluvate osatunnuste väärtuste keskmine.

<b>Rumal olemise hirm</b>	<ul style="list-style-type: none"> <li>• ... pole tarkvara arendamiseks piisavalt tark</li> <li>• ... pole tarkvara arendamiseks piisavaid teadmisi</li> </ul>
<b>Aja nappuse hirm</b>	<ul style="list-style-type: none"> <li>• ... pead nõustuma tähtaegadega, mis pole reaalsed</li> <li>• ...pead ohverdama kvaliteedi tähtaegade kasuks.</li> <li>• ...edu saavutamiseks ei ole piisavalt aega.</li> </ul>
<b>Abi mittesaamise hirm</b>	<ul style="list-style-type: none"> <li>• ...oled pandud vastutama lootusetute olukordade eest.</li> <li>• ... keeruliste probleemide korral ei ole abi kuskilt loota.</li> </ul>
<b>Mõttetu töö hirm</b>	<ul style="list-style-type: none"> <li>• ... arendatavat rakendust pole tegelikult mitte kellelegi vaja.</li> </ul>
<b>Kasutaja nõuete mittemõistmise hirm</b>	<ul style="list-style-type: none"> <li>• ... Sa ei saa lõppkasutaja tegelikest vajadustest aru</li> </ul>

Tabel 24. Tunnuste „arendajate hirmud“ rühmitamine

## 3.8 Tulemused

Tulemustena vaatlen moodustatud uute tunnuste (tabelid 21, 23 ja 24) keskmisi väärtusi moodustatud rühmadel (vt. 3.6 ).

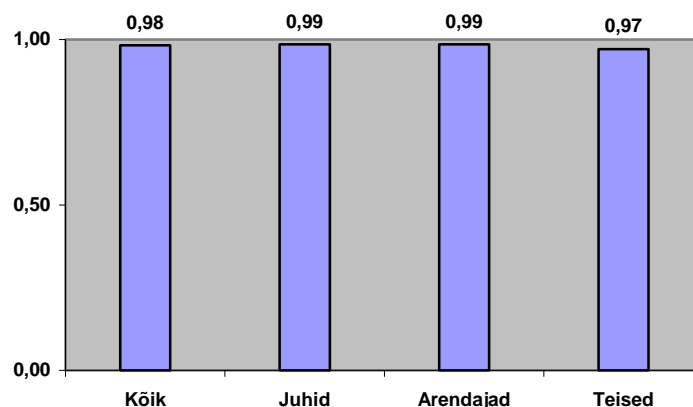
### 3.8.1 Arendusprotsessiga seotud küsimused.

#### 3.8.1.1 Nõuete haldamine

Tunnuste rühm “*Nõuete haldamine*” sisaldab vastanute hinnangut viiele ankeedis esitatud arendusprotsessiga seotud väitele (Tabel 21). Lisaks nõuete haldamise tegevustele sattusid siia rühma ka projektijuhi olemasolu nõue, mis SW-CMM järgi on kõikide osaprotsesside koostisosa ning arendajate osalemise nõue projektiga seotud aruteludes, mis SW-CMM järgi on planeerimise osa .

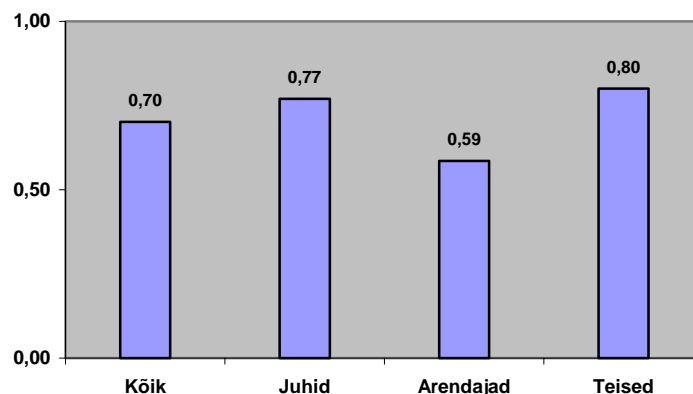
Ehk on projektijuhi olemasolu ning arendajate osalemise nõude sattumise põhjuseks *nõuete haldamise rühma* asjaolu, et projektijuht, kasutaja nõuetega tegelemine ja arendajate osalemine aruteludes on minimaalne ja kõige tavalisem, mida tarkvara arendusprojektides tehakse.

Nõuete haldamisega seotud tegevusi pidasid ankeedile vastanud arendusprojektile edu tagamise seisukohalt oluliseks (Joonis 15).



Joonis 15. Nõuete haldamine on oluline (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Ka peeti *nõuete haldamise rühma* kuuluvaid tegevusi arendusprojektides suhteliselt tavalisteks tegevusteks (joonis 16).



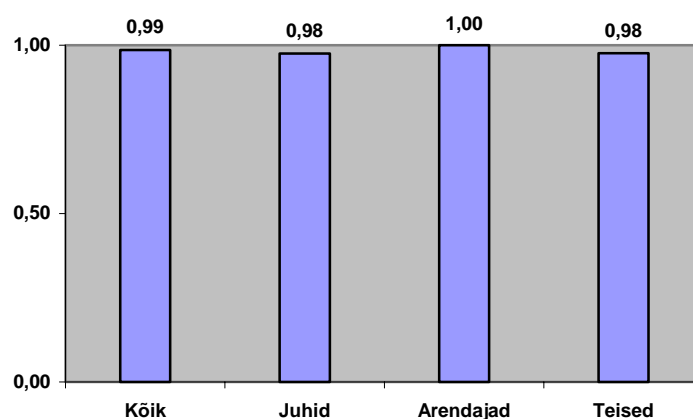
Joonis 16. Nõudeid hallatakse suhteliselt hästi (1 – tavaline; 0 – harva; -1 – pole kasutuses).

Juhid näevad nõuete haldamise tegevusi reaalsetes projektides natuke rohkem kui arendajad.

### 3.8.1.2 Projekti planeerimine.

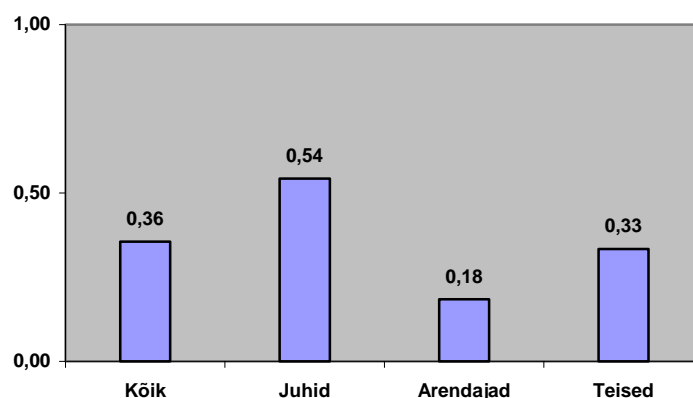
Tunnuste rühm “*Projekti planeerimine*” sisaldab vastanute hinnangut kuuele ankeedis esitatud arendusprotsessiga seotud väitele (Tabel 21). Lisaks planeerimise nõuetele sattusid siia gruppi ka protseduurireeglite ja ressursside nõue ning inimeste koolitamise nõue, mis SW-CMM teise taseme järgi on kõikide osaprotsesside koostisosad.

Planeerimisega seotud tegevusi pidasid ankeedile vastanud arendusprojektile edu tagamise seisukohalt oluliseks (Joonis 17).



Joonis 17. Planeerimine on oluline (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Planeerimise rühma kuuluvaid tegevusi arendusprojektides alati aga ei tehta (joonis 18). Vähemalt jääb üldine keskmine (0,36) alla nõuete haldamise rühma vastavale näitajale (0,70).



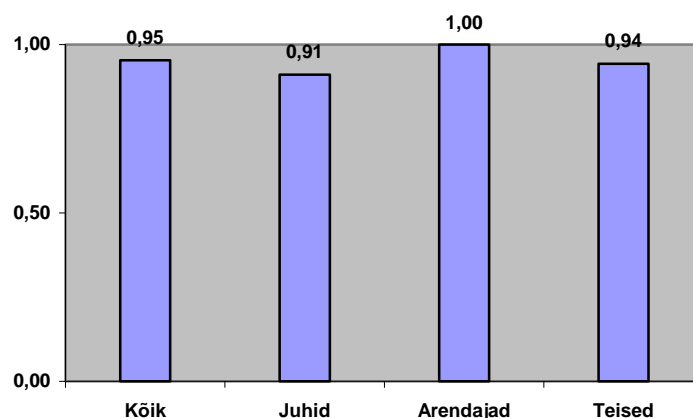
Joonis 18. Alati ei planeerita (1 – tavaline; 0 – harva; -1 – pole kasutuses).

Juhid näevad planeerimise tegevusi reaalsetes projektides rohkem kui arendajad, kelle hinnang kaldub juba harvaesineva piirimaile.

### 3.8.1.3 Tulemuste jälgimine

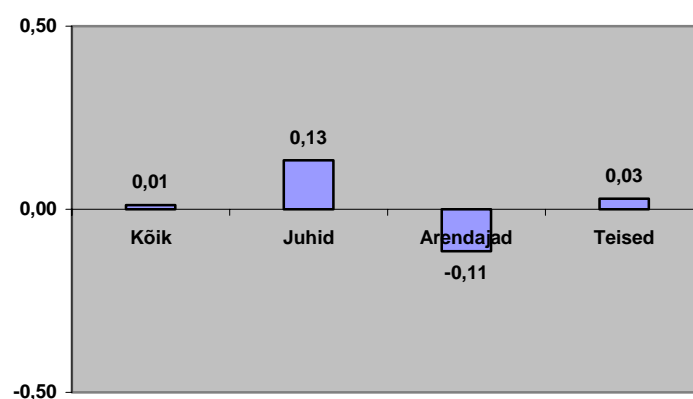
Tunnuste grupp “Tulemuste jälgimine” sisaldab vastanute hinnangut viiele ankeedis esitatud arendusprotsessiga seotud väitele (Tabel 21). Siia gruppi langesid tegevused, mis SW-CMM teise taseme järgi kuuluvad kvaliteedi tagamise ning konfiguratsiooni haldamise osaprotsesside koosseisu ja on seotud arendusprotsessi tulemuste jälgimisega.

Tulemuste jälgimistega seotud tegevusi pidasid ankeedile vastanud arendusprojektile edu tagamise seisukohalt oluliseks (Joonis 19).



Joonis 19. Tulemuste jälgimine on oluline (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Tulemuste jälgimise gruppi kuuluvaid tegevusi tehakse arendusprojektides aga harva (joonis 20).



Joonis 20. Tulemusi jälgitakse harva (1 – tavaline; 0 – harva; -1 – pole kasutuses).

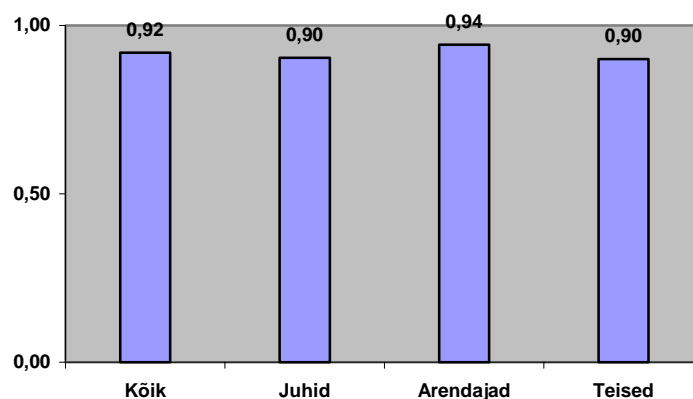
Ka siin on juhid hinnangute andmisel arendajatest optimistlikumad.

### 3.8.1.4 Projekti monitooring

Tunnuste grupp “Projekti monitooring” sisaldab vastanute hinnangut viiele ankeedis esitatud arendusprotsessiga seotud väitele (Tabel 21).

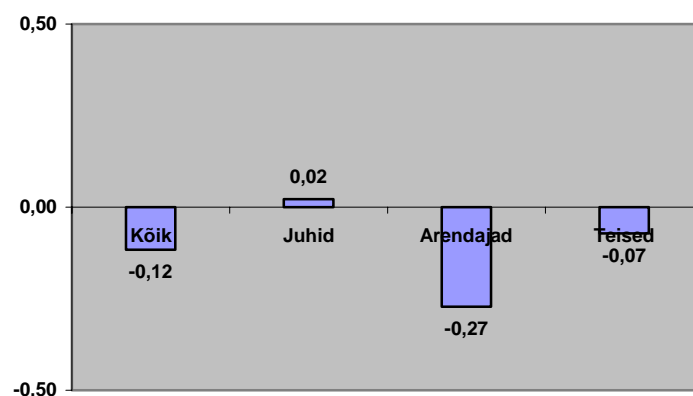


Ehk on projekti monitooringuga seotud tegevused natuke tagasihoidlikuma hinnangu (Joonis 21) oma keskmise tulemusega 0,92 kui nõuete haldamise (keskmine 0,98) või planeerimise (keskmine 0,99) tegevused.



Joonis 21. Projekti monitooring on oluline (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Projekti monitooringu tegevused on arendusprojektides harva esinevad tegevused (joonis 22).



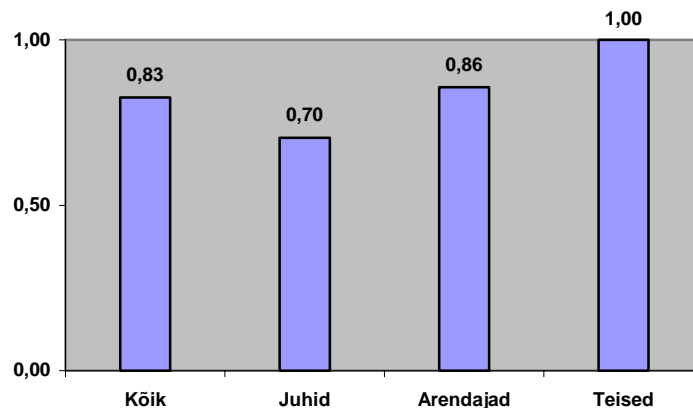
Joonis 22. Projekti monitooring on harvaesinev (1 – tavaline; 0 – harva; -1 – pole kasutuses).

Ka siin on juhid hinnangute andmisel arendajatest optimistlikumad.

### 3.8.1.5 Metoodika läbivaatamine

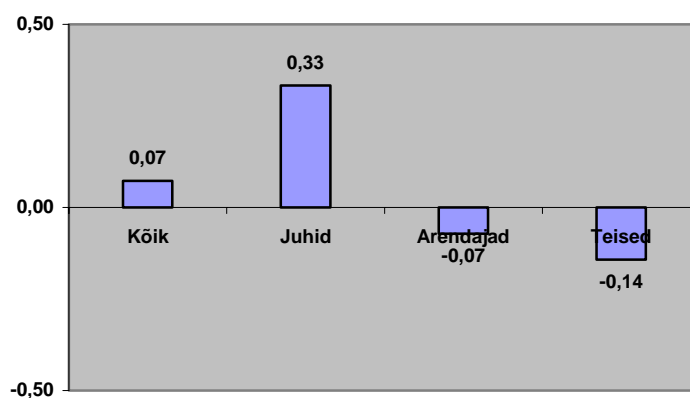
Tunnus “Metoodika läbivaatamine” ühegi teise tunnusega gruppi ei moodustanud (Tabel 21).

Metoodika läbivaatamist peeti arendusprojekti edukuse seisukohal suhteliselt oluliseks (Joonis 23). Juhid hindavad metoodika läbivaatamist arendusprojekti edu seisukohalt vähem oluliseks kui arendajad.



Joonis 23. Metoodika läbivaatamine on oluline (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Metoodika läbivaatamine on arendusprojektides harva esinev tegevus (joonis 24). Juhtide hinnangul aga tehakse seda sagedamini, kui seda tehakse arendajate hinnangul.

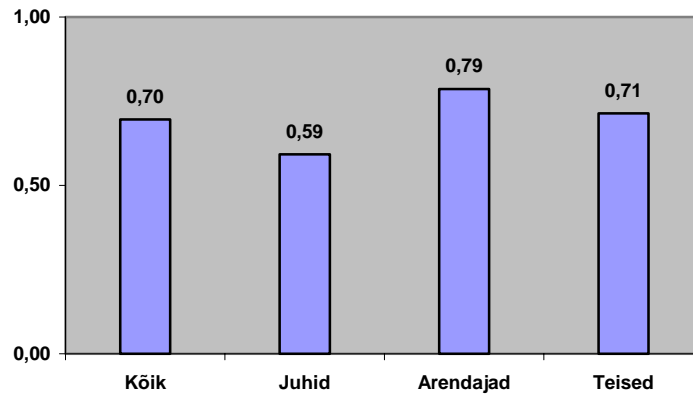


Joonis 24. Metoodika läbivaatamine on harvaesinev (1 – tavaline; 0 – harva; -1 – pole kasutuses).

### 3.8.1.6 Tulemuste audit

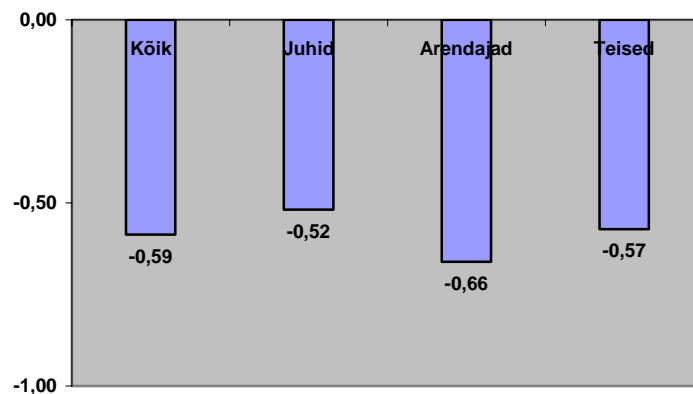
Tunnuste grupp “Tulemuste audit” sisaldab vastanute hinnangut kahele ankeedis esitatud arendusprotsessiga seotud väitele (Tabel 21). Sellesse gruppi paigutusid nõuded, mis on seotud konfiguratsioonide auditiga ning sõltumatu kvaliteedi auditiga.

Võrreldes eelnevatega (keskmised üle 0.9) sai tulemuste audit oma keskmisega 0,7 palli arendusprojekti edukuse seisukohalt mitte enam nii kõrge hinnangu (Joonis 25). Üllatav on juhtide tagasihoidlikum hinnang võrreldes arendajatega. Kas võib siit välja lugeda arendajate soovi anda pingevabalt projekti tegelikust olukorrast ülevaade sõltumatule audiitorile.



Joonis 25. Tulemuste audit pole nii oluline (1 – oluline; 0-ebaoluline; -1 –pidurdav).

Tulemuste audit on arendusprojektides suhteliselt mittekasutatav tegevus (joonis 26).



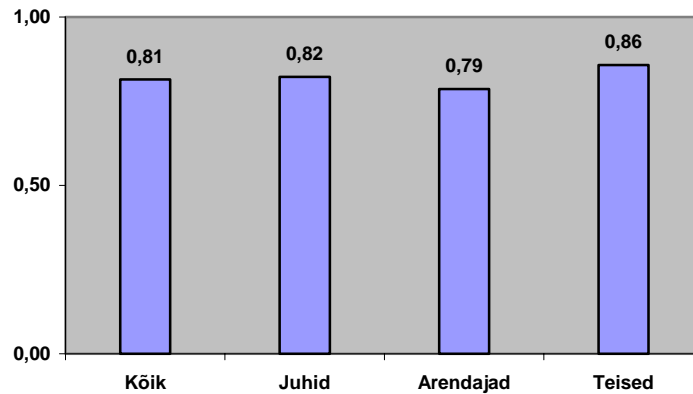
Joonis 26. Tulemuste audit pole kasutuses (1-tavaline; 0-harva; -1-pole kasutuses).

### 3.8.2 Arendusmetoodikatega seotud küsimused

#### 3.8.2.1 Tavalised meetodid.

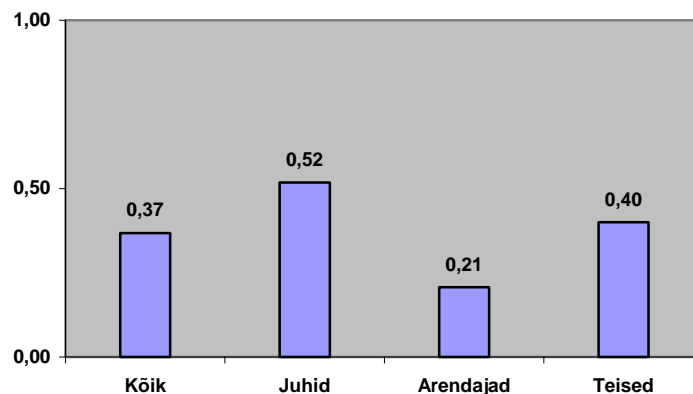
Tunnuste grupp “Tavalised meetodid” sisaldab vastanute hinnangut viiele ankeedis esitatud arendusmeetoditega seotud väitele (Tabel 23). Sellesse gruppi paigutasid XP-metoodika sellised meetodid, mis on sarnase tähendusega kasutuses ka teistes metoodikates. Nendeks meetoditeks on disaini hoidmine lihtsana, optimeerimine lõpus, kliendi osalemine arendustegevuses, lühikesed väljalasked ja töösse pühendunud inimeste osalemine arendustegevuses.

Tavalisi meetodeid peeti arendusprojektile edu tagamise seisukohalt olulisteks meetoditeks (Joonis 27). Ka Daniel Karlström [KARLSTRÖM2002] saab oma töö tulemuseks antud meetoditele suhteliselt kõrge hinnangu. Skaalal väga positiivne, positiivne, neutraalne, negatiivne ja väga negatiivne sai lihtne disain positiivse hinnangu. Kliendi kohalolekut ja lühikesi väljalaskeid pidasid tema uuritavad väga positiivseks. Optimeerimist lõpus ja töösse pühendunud inimeste osatähtsust tema töö ei kajastanud.



Joonis 27. Tavalised meetodid on olulised (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Arendusprojektides on tavalised meetodid (joonis 28) tugevamini esindatud (keskmine 0,37) kui teistesse gruppidesse jäävad meetodid oma keskmistega alla 0,25 punkti. Arendajate hinnang (0,21) on nagu tavaliselt juhtide hinnangust (0,52) madalam.



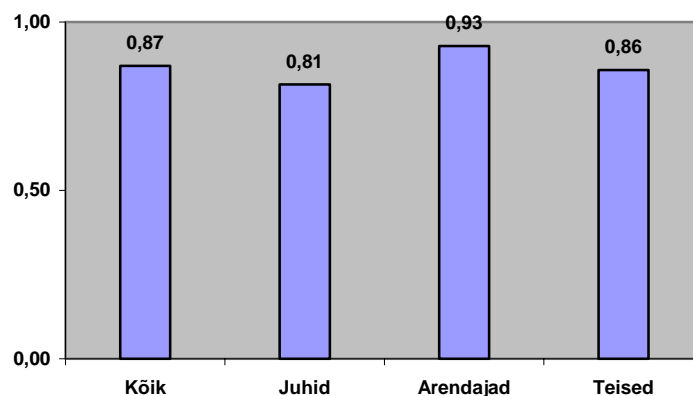
Joonis 28. Tavalisi meetodeid vahel järgitakse (1 – tavaline; 0 – harva; -1 – pole kasutuses).

### 3.8.2.2 Arukad meetodid.

Ma pean arukaks omada projektis selgesõnalist ja kõikidele arusaadavat lühikokkuvõtet või metafoori ning arendajate töökoormuse hoidmist 40 tunni piires nädalas. Just neid kahte XP-metoodika meetodit antud tunnuste grupp – “Arukad meetodid” – sisaldab (Tabel 23).

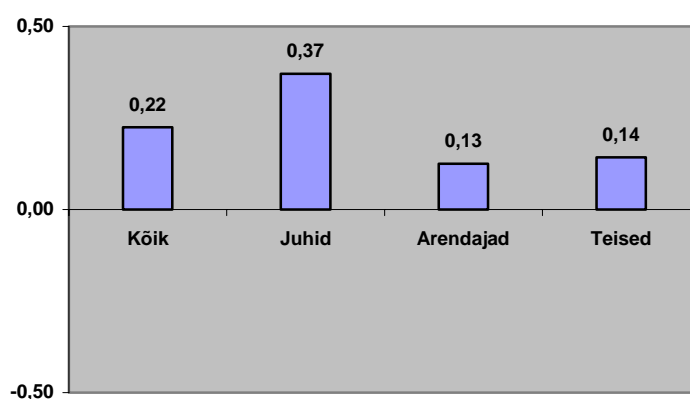
Arukaid meetodeid peeti arendusprojektile edu tagamise seisukohalt olulisteks (Joonis 29). Ehk on arendajad väsimuse laastavat mõju ning kasutaja nõuetest mitte arusaamist (vaata 3.8.3) omal nahal rohkem tunda saanud, kui juhid, ning seetõttu peavad antud tunnuseid juhtidest olulisemaks.

Karlström [KARLSTRÖM2002] saab aga oma töös antud meetoditele suhteliselt keskpärase hinnangu. 40-tunnist tööädalat peetakse neutraalseks ja metafoori olemasolu negatiivseks.



Joonis 29. Arukad meetodid on olulised (1 – oluline; 0-ebaoluline; -1 –pidurdav).

Arendusprojektides on arukate meetodite kasutamine suhteliselt harvaesinev tegevus (Joonis 30). Jälle näevad juhid neid tegevusi arendusprojektides rohkem (0,37) kui arendajad (0,13).

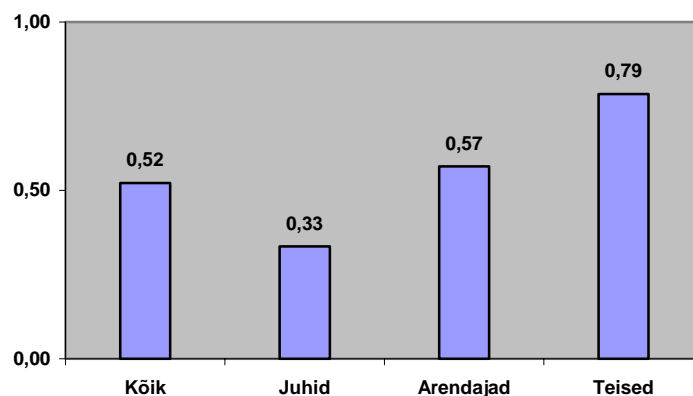


Joonis 30. Arukad meetodid on harvaesinevad (1-tavaline; 0-harva; -1-pole kasutuses).

### 3.8.2.3 Häkkimise meetodid.

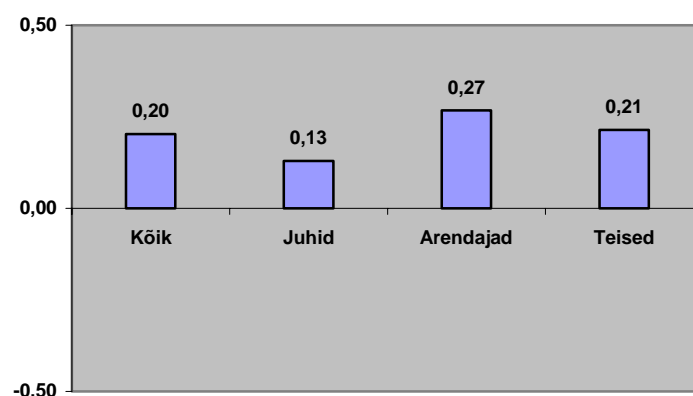
Tunnuste grupp “Häkkimise meetodid” sisaldab vastanute hinnangut kahele ankeedis esitatud arendusmeetoditega seotud väitele (Tabel 23). Sellesse gruppi paigutus XP-metoodika lihtsate vahendite kasutamise meetod ja minu poolt ankeeti sisse pandud arendusmetoodikaid suhteliselt välistav “pidevate kangelastegude tegemise” meetod.

Häkkimise meetodeid arendusprojektile edu tagamise seisukohalt väga olulisteks ei peetud (Joonis 31). Juhid pidasid neid meetodeid vähem oluliseks (0,33) kui arendajad (0,57). Ehk mängib selles erinevuses teatud rolli juhtide suurem usk arendust toetavatesse vahenditesse ning arendajate teadmine, et vahel on “intelligentne häkk” suhteliselt edu toov tegevus (minu isiklikule kogemusele tuginev väide). Määramatu grupp “Teised” pidasid aga antud meetodeid suhteliselt oluliseks.



Joonis 31. Häkkimise meetodid väga olulised pole (1 – oluline; 0-ebaoluline; -1 –pidurdav).

Arendusprojektides on häkkimise meetodite kasutamine suhteliselt harvaesinev tegevus (Joonis 32). Samas on nad oma keskmise näitajaga (0.20) vist tugevamini esindatud kui XP meetodid (-0,12) ning nõrgemini esindatud kui tavalised (0,37) meetodid.



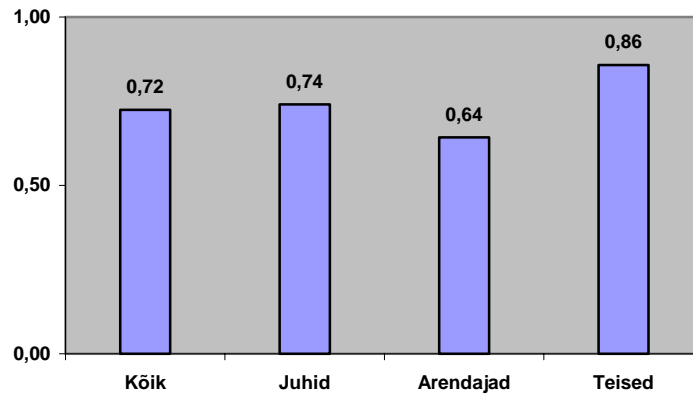
Joonis 32. Häkkimise meetodeid vahel järgitakse (1-tavaline; 0-harva; -1-pole kasutuses).

### 3.8.2.4 Koodi standard

Tunnus “Koodi standard” ühegi teise tunnusega gruppi ei moodusta (Tabel 23).

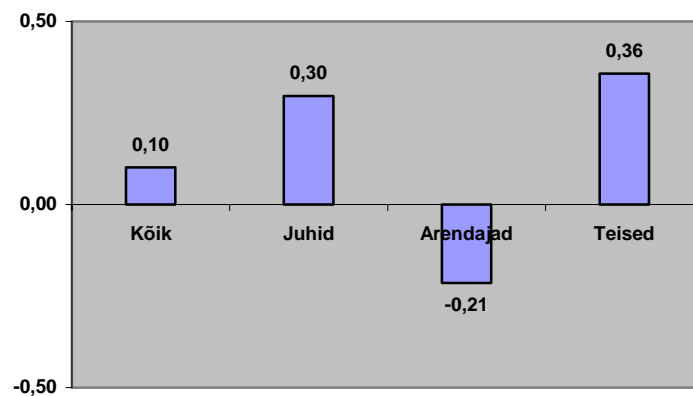
Koodi standardite olemasolu peetakse arendusprojekti edukuse seisukohalt suhteliselt oluliseks (Joonis 33). Ehk on marginaalse erinevuse põhjuseks juhtide ja arendajate hinnangutes arendajate soov olla piirangutest suhteliselt vaba.

Ka Karlströmi [KARLSTRÖM2002] grupp leiab, et koodi standarditel on positiivne efekt.



Joonis 33. Koodi standard on oluline (1 – oluline; 0-ebaoluline; -1 –pidurdav).

Koodi standard on arendusprojektides harvaesinev nähtus (joonis 24). Täiesti tuntav vahe on juhtide (0,34) ja arendajate (-0,21) hinnangutes. Kui juhid leiavad, et koodi standardeid pigem ikka kasutatakse, siis arendajad seda ei leia.

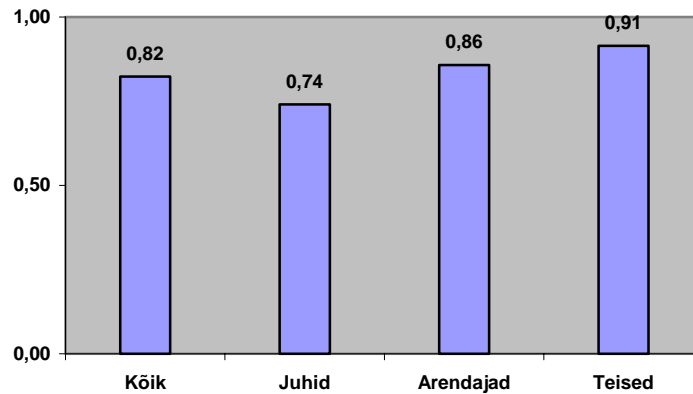


Joonis 34. Koodi standard on harvaesinev (1-tavaline; 0-harva; -1-pole kasutuses).

### 3.8.2.5 XP meetodid.

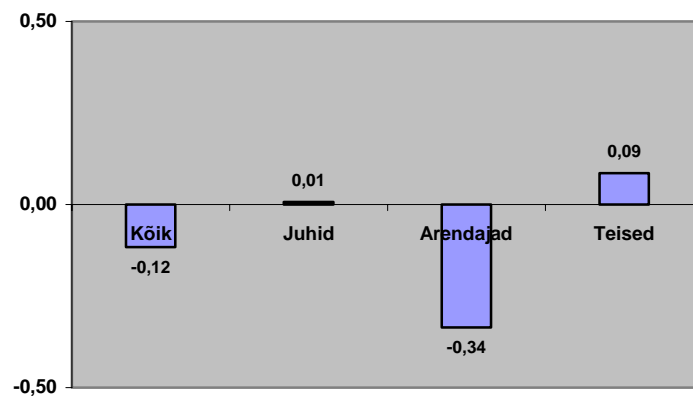
Tunnuste grupp “XP meetodid” sisaldab vastanute hinnangut viiele ankeedis esitatud arendusmeetoditega seotud väitele (Tabel 23). Sellesse gruppi paigutusid XP-metoodika seisukohalt olulise tähtsusega meetodid: soovilood, testidel tuginev arendus, pidev tervikuks integreerimine, koodi ühisomandus ja koodi pidev ümberkirjutamine (*refactoring*).

XP meetodeid peeti arendusprojektile edu tagamise seisukohalt olulisteks meetoditeks (Joonis 35). Karlströmi [KARLSTRÖM2002] grupi vastavad hinnangud on: testid – väga positiivne; pidev tervikuks integreerimine – positiivne; koodi ühisomand – positiivne; koodi pidev ümberkirjutamine – positiivne. Hinnangut soovilugude kasutamisele Karlström välja ei toonud.



Joonis 35. XP meetodid on olulised (1 – oluline; 0 – ebaoluline; -1 – pidurdav).

Arendusprojektides on aga XP-meetodid harvaesinevad (joonis 36). Arendajate hinnang (-0,34) on juhtide hinnangust (0,01) madalam. Kui juhid leiavad, et antud meetodeid mõnikord tehakse, siis arendajate arvates kohtab neid “vähem kui harva”.



Joonis 36. XP meetodid järgitakse harva (1 – tavaline; 0 – harva; -1 – pole kasutuses).

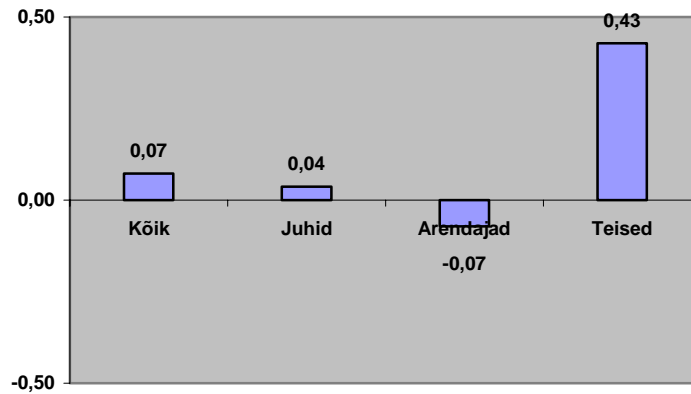
### 3.8.2.6 Paarisprogrammeerimine.

Tunnus “Paarisprogrammeerimine” ühegi teise tunnusega gruppi ei moodusta (Tabel 23).

Paarisprogrammeerimist peetakse arendusprojekti edukuse seisukohalt ebaoluliseks (Joonis 37). Erandi moodustab vastajate grupp “Teised”, mis sai kokku pandud kõikidest nendest vastajatest, kes juhtide ja arendajate gruppi ei sobinud. Kindlasti mõjutab seda tugevasti kõikide XP-metoodikaga (kolm kokku) kokku puutunud vastaja asumine selles grupis.

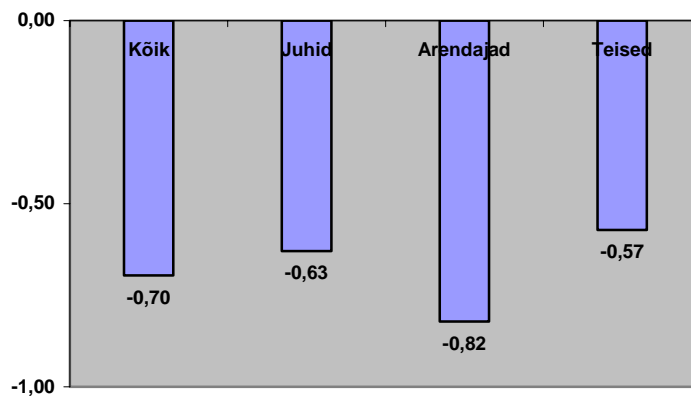
Karlströmi [KARLSTRÖM2002] 10-ne liikmeline grupp peab paarisprogrammeerimise efekti väga positiivseks.





Joonis 37. Paarisprogrammeerimist peetakse ebaoluliseks (1 – oluline; 0-ebaoluline; -1 – pidurdav).

Paarisprogrammeerimist arendusprojektides ei harrastata (Joonis 38).



Joonis 38. Paarisprogrammeerimine pole kasutuses (1-tavaline; 0-harva; -1-pole kasutuses).

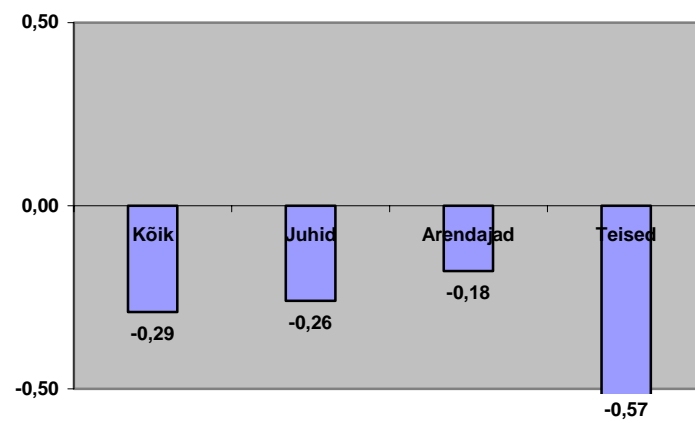
### 3.8.3 Hirmudega seotud küsimused

Eesti arendajad ei tunne hirmu selle ees, et nende arendatavat rakendust mitte kellelegi vaja pole (Joonis 39). Eriti tundmatu on see hirm grupil, mis sai moodustatud vastajatest, kes moodustatud “Juhtide” ja “Arendajate” gruppi ei sobinud. Lõppkasutaja nõuete mittemõistmise hirmu (Joonis 40) harva ikka tuntakse. Arendajad natuke rohkem (0,25) kui juhid ( -0,11).

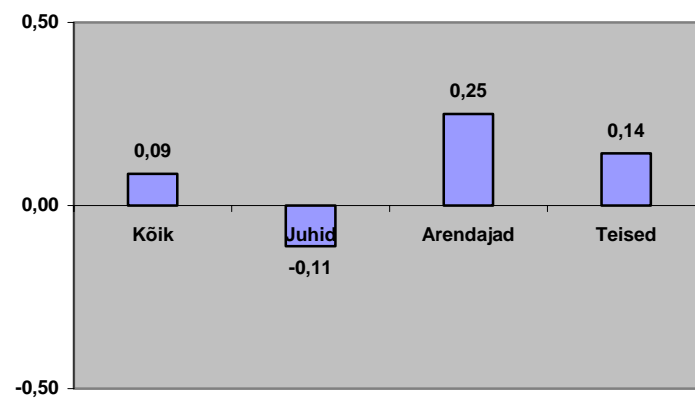
Aja nappuse hirmu ( ebareaalsed tähtajad, kvaliteedi ohverdamine tähtaegade kasuks ning edu saavutamiseks ei jätku aega) (tabel 24) tuntakse suhteliselt tihti (Joonis 42). Abi mittesaamise hirmu (lootusetud olukorrad, üksi keeruliste probleemidega) (tabel 24) vahel tuntakse (Joonis 43). Juhtide grupp tunneb antud hirmu vähem (-0,26) , kui arendajad (-0,09) .

Seda, et ta pole tarkvara arendamiseks piisavalt tark ning tal pole tarkvara arendamiseks piisavalt teadmisi (Tabel 24) harva tuntakse (Joonis 41). Arendajad

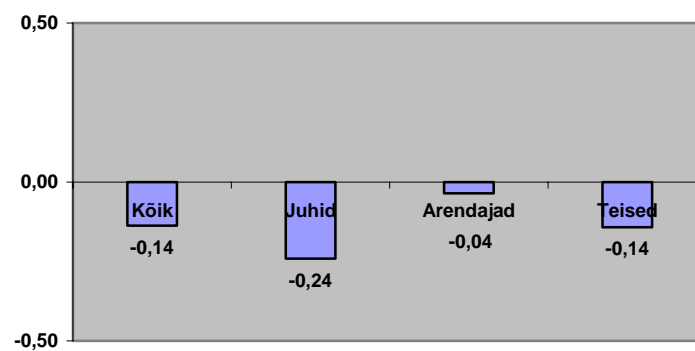
sagedamini, kui juhid. Ehk on erinevuse põhjus juhtide suuremas arendustegevuse alases kogemuses ning IT mõistes paremas hariduses (vaata 3.6).



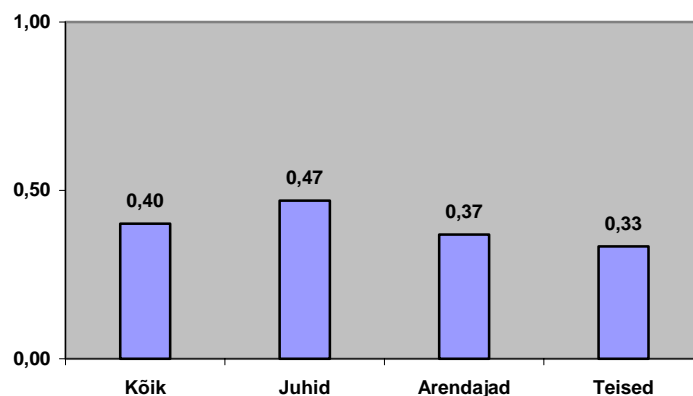
Joonis 39. Mõttetu töö hirm (1 –tihti; 0-harva; -1 –mitte kunagi).



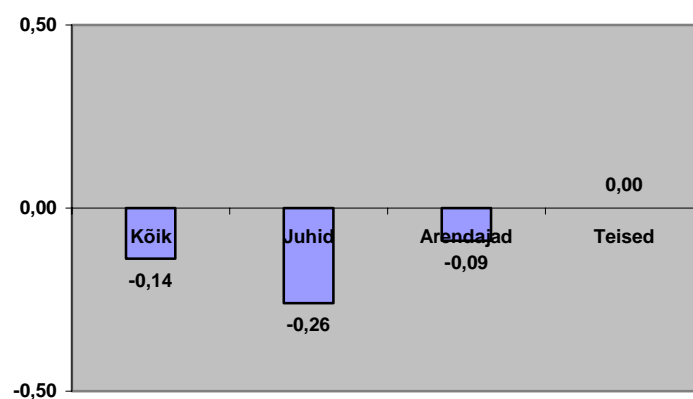
Joonis 40. Kasutaja nõuete mittemõistmise hirm (1 –tihti; 0-harva; -1 –mitte kunagi).



Joonis 41. Rumal olemise hirm (1 –tihti; 0-harva; -1 –mitte kunagi).



Joonis 42. Aja nappuse hirm (1 –tihti; 0-harva; -1 –mitte kunagi).



Joonis 43. Abi mittesaamise hirm (1 –tihti; 0-harva; -1 –mitte kunagi).

### 3.9 Ankeedi kokkuvõte ja järeldused

69 ankeedile vastanud eesti tarkvara arendajat paigutasid kahte gruppi (vaata 3.6 ), mida nimetasin tinglikult JUHTIDEKS ja ARENDAJATEKS . Kolmandasse gruppi paigutasin kõik ülejäänud.

Tabel 25 annab rühmade iseloomustuse.

	JUHID	ARENDAJAD
Amet	57% kõikidest ankeedile vastanud juhtidest kuulub antud rühma; juhtide osakaal antud rühmas on 59%	67 % kõikidest ankeedile vastanud arendajatest kuulub antud rühma; arendajate osakaal antud rühmas on 75%
Teadmised metoodikast	65% kõikidest headest ja suurepäraastest hinnangutest kuulub antud rühma; heade ja suurepäraaste hinnangute osakaal antud rühmas on 78%.	67% kõikidest vähestest ja keskmistest hinnangutest sattus siia rühma; nende osakaal antud rühmas on 89%
Haridus	65% kõikidest IT kõrgharidust	10% kõikidest IT kõrgharidust

	omavatest kuulub antud rühma; IT ja kõrgema hariduse osakaal antud rühmas on 85%	omavatest kuulub rühma; 67% kõikidest IT eriharidust omavatest kuulub rühma ; IT ja kõrgema hariduse osakaal rühmas on 50%
Kogemus	60% kõikidest 6 ja enam aastat arendusalast kogemust omavatest kuulub antud rühma; nende osakaal rühmas on 78%	61% kõikidest kuni 5 aastat arendusalast kogemust omavatest kuulub antud rühma; nende osakaal selles rühmas on 71%.

*Tabel 25. Rühmade iseloomustus*

Ankeedis arendusprotsessiga seotud 24 küsimust paigutasin projektides kasutatavuse järgi kuude tunnuste rühma (vaata 3.7.1).

Tabel 26 annab kokkuvõtte vastajate hinnangutest rühmade lõikes lähtuvalt nõude olulisusest tarkvaraprojekti edu tagamiseks. Võib teha järelduse, et eesti tarkvaraarendajad peavad SW-CMM teise taseme nõudeid arendusprojekti edu tagamiseks olulisteks. Natuke teistest vähem peetakse lugu auditeerimise nõuetest. Selline arendajate hoiak annab minu arust hea aluse meetodikate juurutamise abil tarkvaraprojektide edukuse suurendamiseks ja arendusprojektide küpsusastme tõstmiseks vähemalt SW-CMM teise taseme nõuetele vastavaks.

	Kokku	Juhid	Arendajad	Teised
Nõuete haldamine	0,98	0,99	0,99	0,97
Projekti planeerimine	0,99	0,98	1,00	0,98
Tulemuste jälgimine	0,95	0,91	1,00	0,94
Projekti monitooring	0,92	0,90	0,94	0,90
Metoodika läbivaatamine	0,83	0,70	0,86	1
Tulemuste audit	0,70	0,59	0,79	0,71

*Tabel 26. Hinnangud arendusprojekt edu tagavatele tegevustele*

Tabelis 27 (Üksikute taustatunnuste kohta vaata koondandmeid lisas 4) on kokkuvõtte tarkvaraprojektide reaalse olukorra kohta antud hinnangutest. Vastajate hinnangutele tuginedes on nõuete haldamise tegevused eesti tarkvarafirmades läbiviidavates tarkvaraprojektides pigem tavalised kui harvaesinevad. Ka projekti planeerimise tegevusi nähakse arendusprojektides suhteliselt tihti.

Harva esineb projektides metoodika läbivaatamist ja kohandamist; tulemuste läbivaatamist (kvaliteedi alast tegevust ja konfiguratsioonide haldamist) ning projekti monitooringut.

Auditeerimine on aga tarkvaraprojektides vastajate hinnangutele tuginedes praktiliselt tundmatu. Süмптоmaatiline on arendajate grupi poolt antud madalamad hinnangud võrreldes juhtide rühmaga.

Kokkuvõtteks võib öelda, et SW-CMM teise taseme nõuete täitmine on enamikule eesti arendusfirmadele piisavalt suur väljakutse.

	Kokku	Juhid	Arendajad	Teised
Nõuete haldamine	0,70	0,77	0,59	0,80
Projekti planeerimine	0,36	0,54	0,18	0,33
Metoodika läbivaatamine	0,07	0,33	-0,07	-0,14
Tulemuste jälgimine	0,01	0,13	-0,11	0,03
Projekti monitooring	-0,12	0,02	-0,27	-0,07
Tulemuste audit	-0,59	-0,52	-0,66	-0,57

Tabel 27. Hinnangud arendusprojekti tegevustele

Rühmitatud tunnuste omavahelise korreleeruvuse hindamiseks kasutasin Pearsoni korrelatsioonikordajaid (vt.Lisa.4). Suhteliselt tugevalt (0,645) korreleeruvad *projekti planeerimine* ja *tulemuste jälgimine*. Nende mõlemaga korreleerub hästi ka *projekti monitooring*: Pearsoni korrelatsioonikordajad vastavalt 0,622 (*tulemuste jälgimine*) ning 0,606 (*projekti planeerimine*). Omakorda nendele lisandub suhteliselt kõrgete kordajatega *tulemuste audit*: 0,571 (*tulemuste jälgimine*); 0,476 (*projekti planeerimine*) ning 0,463 (*projekti monitooring*). *Nõuete haldamine* korreleerub teistega kõige halvemini (maksimaalne 0,377).

Siit võiks järeldada, et kui peale *nõuete haldamise* tegeldakse projektis ka veel *projekti planeerimisega*, siis tõenäoliselt on selles projektis vähemalt osaliseltki kaetud ka *projekti monitooringu* ning *tulemuste jälgimise* (kvaliteedi tagamise ja konfiguratsioonide haldamise) tegevused (Vaata ka hüpoteetilisi küpsusmudeleid antud töö kokkuvõttest).

Üksikute tunnuste omavahelised korreleeruvused on toodud lisas 4.

Kui jätta välja paarisprogrammeerimine (peetakse mitteoluliseks aga ka mitte takistavaks) , siis teised XP-metoodika meetodid on ankeedile vastanute hinnangul eesti arendajate poolt täiesti vastuvõetavad meetodid tarkvaraprojekti edu tagamiseks (Tabel 28).

	Kokku	Juhid	Arendajad	Teised
XP meetodid	0,82	0,74	0,86	0,91
Tavalised meetodid	0,81	0,82	0,79	0,86
Arukad meetodid	0,87	0,81	0,93	0,86
Koodi standard	0,72	0,74	0,64	0,86
Häkkeri meetodid	0,52	0,33	0,57	0,79
Paarisprogrammeerimine	0,07	0,04	-0,07	0,43

Tabel 28. Hinnangud XP-meetoditele projektile edu tagamise seisukohalt.

Samuti võib vastanute hinnangutele tuginedes väita, et XP-metoodika range juurutamine eesti tarkvarafirmade arendusprojektides võib olla edukas ja ka mõttekas. Ühelt poolt mõttekas sellepärast, et suurem osa projektidest viiakse läbi kuni 12 liikmeliste (vaata joonis 11) meeskondade poolt (täiesti aktsepteeritav XP-meeskonna suurus). Teiselt poolt mõttekas kuna XP-metoodikaga on võimalik suhteliselt hästi rahuldada SW-CMM teise taseme nõudeid (vaata 2.4) ning annab ka hea baasi edasiarenemiseks ja SW-CMM kolmanda taseme nõuete rahuldamiseks (Tabel 9)

Edukas ka sellepärast, et XP-metoodika meetodid (paarisprogrammeerimine välja arvatud) pole vastanute hinnangul arendusprojektides mitte täiesti tundmatud (Tabel 29). Üksikute taustatunnuste kohta vaata XP metoodika kasutamise koondandmeid lisas 4.

	Kokku	Juhid	Arendajad	Teised
Tavalised meetodid	0,37	0,52	0,21	0,40
Arukad meetodid	0,22	0,37	0,13	0,14
Häkkeri meetodid	0,20	0,13	0,27	0,21
Koodi standard	0,10	0,30	-0,21	0,36
XP meetodid	-0,12	0,01	-0,34	0,09
Paarisprogrammeerimine	-0,70	-0,63	-0,82	-0,57

*Tabel 29. Hinnangud XP-meetodite sageduse kohta projektides*

Vastanute hinnangul on eesti tarkvaraarendajatel hirm aja vähesuse ees (3.8.3). Mõttetu töö hirmu tuntakse kõige vähem. Kasutaja nõuete mittemõistmise, rumal olemise ja abi mitteraamimise hirmu tuntakse harva. Kent Beck'i [BECK2000] väitel aitab kõikide nende hirmude vastu metoodika juurutamine.

	Kokku	Juhid	Arendajad	Teised
Aja nappuse hirm	0,40	0,47	0,37	0,33
Kasutaja nõuete mittemõistmise hirm	0,09	-0,11	0,25	0,14
Rumal olemise hirm	-0,14	-0,24	-0,04	-0,14
Abi mitteraamimise hirm	-0,14	-0,26	-0,09	0,00
Mõttetu töö hirm	-0,29	-0,26	-0,18	-0,57

*Tabel 30. Eesti arendajate hirmud*

## 4 XP JUURUTAMISEST FIRMAS SYSTEK INFORMATIONSYSTEMS

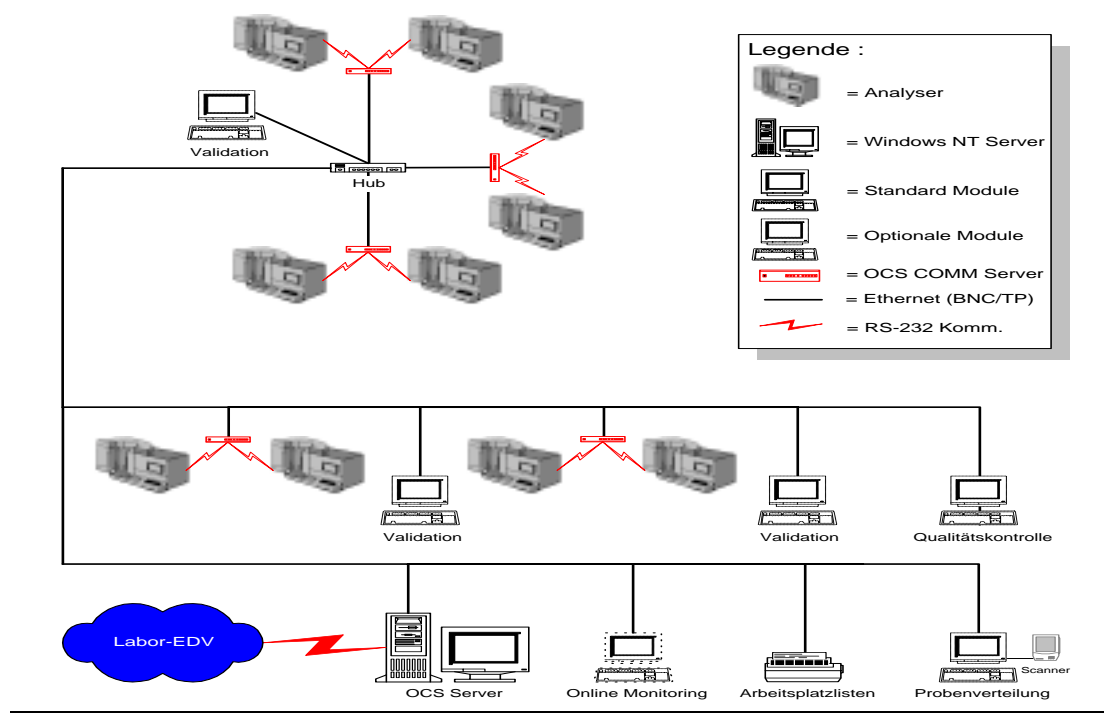
### 4.1 Kuidas ja miks asi algas?

Kui mind 1999. aasta kevadel firma Systek Informationsystems OÜ projektijuhiks (tegevdirektori ülesannetes) palgati, võtsin endaga oma eelnevast töökohast kaasa tuttava tarkvaraarendaja, kellega meil olid arendustegevusele suhteliselt sarnased vaated. Kuna eelmisest töökohast sai kaasa võetud suhteliselt negatiivne tarkvaraprojektide juhtimise kogemus, otsustasime kohe alguses arendustegevusele läheneda võimalikult süsteemselt.

Alates firma loomisest peale on arendatud ainult ühte toodet koondnimetusega **Online Control Server** (edaspidi **OCS**) (Joonis.44). Tegemist on meditsiinilaboratooriumites kasutatava tarkvara terviklahendusega väikestele ja keskmise suurusega laboratooriumidele. Oma põhiideelt on OCS standardset liidest (protokolli) omav “intelligentne” laboriseade, mis oskab suhelda erinevate füüsiliste laboratooriumiseadmetega. Täpsemalt OCS funktsioonid ja võimalused vaata Lisa.5.

Lahendus võimaldab ka suurte laboratooriumide vajaduste rahuldamist, kuid suurte ja kallite süsteemide turg on maailmas suhteliselt täis. Samas puuduvad head, odavad ja töökindlad lahendused väikestele ja keskmistele laboratooriumidele.

Täpsustuseks võib öelda, et kui kõik laboratooriumid Eestis koondada ühte kohta (mis ilmselt oleks otstarbekas) kokku, siis saaksime ühe maailma (üldistused Saksamaa Liitvabariigi järgi, mida olen oma silmaga näinud) mõistes väikese laboratooriumi, kus oleks mõttekas OCS kasutamine.



Joonis 44. Online Control Server

2001.aasta lõpus oli OCS esimene versioon valmis ning väljas Saksamaal igal aastal toimuval rahvusvahelisel näitusel MEDICA (vt.<http://www10.medica.de>). Sama aasta lõpul sõlmis korporatsioon Sysmex (vt.<http://www.sysmex.com>), üks võimsamaid meditsiinilaboratooriumi seadmeid tootev ja müüv firma maailmas, lepingu OCS hulgiostuks. 2002. aasta aprillis installeeriti katseliselt esimene OCS (vähendatud versioon) ühes Saksamaa LV laboratooriumis. Tänapäevaks on esimese lepingu OCS kasutamise kohta sõlminud ka teine verelaboratooriumide gigant Olympus (vt.<http://www.olympus.com>).

2001. aasta lõpus võeti lisaks tööle kaks arendajat. Jätkati OCS edasiarendamist. 2002. aasta lõpus demonstreeriti (juba korporatsiooni Sysmex paviljonis) ühte OCS osa koondnimetuse *BloodBank* (verepank) all <http://www.systek.de/blutdepot/default.htm>. 2003. aasta augustis valmis OCS kvaliteedi kontrolli (nimetuse MultiLab™-QC all) osa, mis automatiseerib ja abistab meditsiinilaboratooriumi töötajaid kohustusliku verelaboratooriumi kvaliteedi kontrollimise protseduure täita. 2003. aasta septembris läks oma teisele elule 2001. aastal valminud vere katsutite marsrutiseerimise osa (liidetakse edaspidi OCS-iga tervikuks) David. 2003. aasta novembris toimuval järjekordsel MEDICA näitusel on väljas kogu süsteem tervikuna.

Tarkvara arendustegevuse planeerimist alustasime Craig Larman'i raamatu "*Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*" [LARMAN1998] järgi. Paul Leis nimetab sellist meetodikat agiilseks UP-ks [LEIS2001] (Seda meetodikat on lühidalt iseloomustatud ka antud töö alajaotuses 1.4.3.8).

Algul me püüdsime vajalikke mudeleid joonistada erinevate vahenditega (SELECT, VISIO, ModelMaker, Rational). Hiljem sellest loobusime ja järele jäi ainult paber, pliiats, tahvel ja kriit.

Peagi selgus, et lihtsalt mudelite väljajoonistamisest ei piisa, kuna kahte põhiprobleemi – kasutaja nõuetest arusaamist ning tekkivaid vigu loodud tarkvaras – sellega lahendada pole võimalik.

Alates 2001. aasta sügisest üritame tarkvara arendamise meetodikana juurutada XP reegleid ja tehnikaid [Beck2000]. XP juurde jõudmine oli meil analoogiline sellega, kuidas Cybernetica AS-is jõuti unifitseeritud protsessini [SEEBA2001]: peale vastavateemalise raamatu lugemist toimub selle tutvustamine töökaaslastele ning ühine arutelu.

## **4.2 Mida ja kuidas me oleme teinud?**

Alljärgnevalt tarkvara arendamise protsessi kirjeldus ja analüüs firmas Systek Informationssysteme OÜ aastatel 1999 kuni 2003. Kirjeldamisel lähtun XP meetodika meetoditest.

### **4.2.1 Planeerimine**

#### **4.2.1.1 Soovilood ( User story)**

Meie kasutaja poolt koostatud sooviloo näide on toodud lisas 5. Tegemist on kommenteeritud pildiga. Selliseid kommentaaridega pilte meie klient



(arendusmeeskonna liige, tellija esindaja) meile pidevalt joonistab. Arenduse algusfaasis teeb ta seda *MS Visio* vahendeid kasutades: kavandab ekraanivormid, mis tema arust on laboratooriumis head kasutada ning lisab kommentaaridena vajalikud selgitused. Arenduse edenedes, kui esimene versioon on juba töökorras, lisatakse kommentaare ja vajakajäämisi otse rakendusest kopeeritud ekraanivormidele.

Meie kasutaja soovilood (soovilugu on üks kommentaaridega pilt) paiknevad kasutaja soovidokumendis (MS Word dokument). Soovidokumendis on üks või mitu kasutaja seisukohalt omavahel seotud soovilugu. Soovidokumentidel on nimi ja soovidokumendid säilitatakse vastava väljalaske (arendusperiood) soovidokumentide kataloogis.

Üldine reegel (see on kujunenud, keegi pole seda kehtestanud) on: iga kasutaja vormi kohta on üldjuhul üks soovidokument. Seda dokumenti ei parandata teisiti, kui sellele dokumendile järgnevate märkustega. Kui soovidokumendi nimi on *QCGeneral*, siis temale järgnevad märkusi (vead, täiendused, arusaamatused, parandused,...) sisaldavad dokumendid kannavad nime *Remarks\_QcGeneral\_001*, ... *\_002* jne.

Kasutaja soovilugude sisseviimine praktilisse arendustegevusse lõi võimaluse konstruktiivseks suhtlemiseks töö tellija (antud konkreetsel juhul firma äripool) ja arendajate vahel. Paranes arendajate arusaam tarkvarale esitatavatest nõuetest ning suurenes tellija rahulolu eelkõige just rakenduse väljanägemise suhtes.

Samuti paranes soovilugude sisseviimisega tunduvalt tellija üldine suhtumine arendajatesse. Kuna klient on ikkagi eelkõige tellija esindaja, siis soovilugude joonistamise ja kirjutamise kaudu hakkas tellija rohkem mõistma arendatava rakenduse mahtu ning aru saama rakendusele kuluvast ajast. Ka oli tellija sunnitud rohkem ja täpsemalt mõtlema rakendusele esitatavatele nõuetele ning neid ka arusaadavalt sõnastama.

Kõik see tõi kaasa tellija ja arendajate parema teineteisemõistmise ning parema töökeskkonna.

Peeter Normak [NORMAK2001], tuginedes *Standish Group* poolt 1994 aastal läbi viidud uurimusele, rõhutab tarbijate kaasamise olulisust projekti edu tähtsaimaks faktoriks.

#### 4.2.1.2 Väljalasked ja nende planeerimine

Lühikesi väljalaskeid oleme me kasutanud enne, kui me XP metoodikat teadlikult juurutama hakkasime. Olen täiesti nõus Karlströmi [KARLSTRÖM2002] poolt saadud tulemustega: lühikeste väljalasete sisseviimine arendustegevusse on lihtne, ning sisseviimisest saadav efekt on väga positiivne.

Ilmselt teatud rakenduse spetsiifikat arvestades ei saa me väljalaskeid siiski väga lühikesteks teha. Kuna tegemist on müügiks mõeldud produktiga peab enne selle turustamise algust olema toode potentsiaalsetele klientidele tõeliselt kasulik. Meie poolt kasutatavate väljalasete pikkused kõiguvad keskmiselt poolest aastast ühe aastani. Erandi moodustas OCS testi versioon, mida arendati kokku 17 kuud ja mis lõppes testimisega (turustama ei hakatud).

Tabelis 31 on toodud OCS väljalasked.

Toode	Arenduse algus	Müügi algus	Väljalaske pikkus kuudes
OCS testi versioon	1999 aprill	2000 august (müüki ei läinud, Lõppes testiga)	17
David (OCS marsruuter) esimene versioon	2000 jaanuar	2000 juuni	6
OCS esimene versioon	2000 nov	2001 nov	12
BloodBank	2002 mai	2002 nov	6
Multilab (OCS klient) eelversioon	2002 mai	2002 nov (turustama ei hakatud, ainult demonstreeriti näitusel)	6
Multilab – QC (kvaliteedikontroll)	2002 nov	2003 august	12
David teine versioon	2003 juuni	2003 september	4
Multilab – testide valideerimine	2003 august	2003 nov (plaan)	5
Multilab (OCS klient) esimene täisversioon	2002 nov	2003 nov (plaan)	14

*Tabel 31. OCS väljalasked*

Väljalasked planeeritakse lähtuvalt äri vajadusi silmas pidades. Seega on fikseeritud tähtajad. Tähtajad on seni olnud seotud kas MEDICA igaaastase näitusega novembris, või mingi müügilepinguga, mille äripool sõlmib. Tähtajast üleminek pole olnud praktiliselt võimalik. Samas on olnud võimalik vähendada või lihtsustada (ja seda me oleme ka teinud) rakenduse funktsionaalsust vastavalt kokkuleppele ja lisada tootele uut funktsionaalsust juba toote ekspluateerimise ajal.

Väljalasete funktsionaalsus planeeritakse väga üldiselt. Väljalase algab planeerimise koosolekuga kas Eestis või Saksamaal ja sellest võtavad osa kõik äripoolle ning kõik arenduse poole inimesed. Kokku 6 kuni 8 inimest vastavalt vajadustele. Tüüpiline väljalaske planeerimise koosolek kestab umbes 12 tundi, mille sisse mahub ka kaks korralikku einet korralikus restoranis. Väljalaske koosoleku ajal äri poole inimesed räägivad suurtest plaanidest ning arendajad püüavad asjale üldjoontes (siiski võimalikult täpselt) pihta saada.

Väljalaske koosoleku tulemusena lepitakse kokku (mõistes arendajad saavad teada) tähtaja ning enam vähem realiseerimist vajava funktsionaalsuse.

#### 4.2.1.3 Iteratsioonid ja nende planeerimine

Kuigi kahenädalased (vahel ka lühemad) iteratsioonid on meil kasutusel, tegeleme me iteratsioonide planeerimisega suhteliselt vähe. Me lähtume kliendi vajadustest, ning üritame võimalikult palju ühes iteratsioonis ära teha. Niipalju siiski on, et me jagame omavahel ülesandeid, kes mille realiseerib. Kuna me oleme seni arendajatele kätte andnud tunduvalt suuremaid tükke, kui XP seda eeldaks, siis on seda ka suhteliselt hõlbus teha. Igal on oma lõik.

Mida me mõistame iteratsiooni all. Iteratsioon on kahe kliendile (roll meeskonnas) testimiseks saadetud rakenduse versiooni vaheline aeg. Tavaline iteratsioon on meil kaks nädalat pikk, algab teisipäeval (tavaliselt peale lõunat) ning lõpeb esmaspäeval millal iganes. Oluline on, et klient saaks uue versiooni koos realiseeritud ning kasutaja seisukohalt uute funktsioonidega kätte. Läheb siis nii kaua kui läheb. Tavaliselt üritatakse siiski asi reedeks ära lõpetada (või hädavajadusel nädalavahetused appi võtta).

Iteratsioonid on tavaliselt üle planeeritud. Pole vist veel peaaegu kunagi olnud, et kõik iteratsiooni planeeritud (lugeda kliendi poolt soovitud) soovilood realiseeritud saaks. Selles suhtes on aga selgus. Soovilood on kliendil tähtsuse järjekorda pandud (Lisa.5). Alustatakse lehe ülemisest otsast. Tehakse niipalju kui jõutakse.

Iteratsioonidel on meil selline omadus, et mida lähemale hakkab jõudma tähtaeg, seda lühemaks iteratsioonid jäävad. Väljalaske lõpus võib juhtuda, et iteratsiooni pikkus on ainult pool päeva.

Samas on iteratsioonide kasutamine aidanud meil vältida ajahätta jäämisest tingitud negatiivseid kõrvalefekte – suhtlemise vähenemist juhtide ja klientidega – mida Normak [NORMAK2001] toob välja kui tarkvaraprojektide edu ühe kriitilise faktori.

## **4.2.2 Disainimine**

### **4.2.2.1 Rakenduse metafoor**

Kui vaadata metafoori mitte ainult kui lühikest kokkuvõtet projektist [BECK2000] vaid ka kui arhitektuuri elementi, nagu seda vaatab Mark C. Paulk (üks vaieldamatutest tarkvaratehnika gurudest) [PAULK2001], oli süsteemi metafoori sisseviimine tarkvara tootmisse üks esimesi juurutatud XP meetodeid. Juurutamise ajal me ei teadnud, et tegemist on XP metoodika mõistes metafooriga. OCS metafoor sisaldab kaheksat OCS põhifunktsiooni ja kahte illustreerivat joonist (Lisa.5). 1999 aastal loodud pildid ja siis sõnastatud põhifunktsioonid on tänaseni muutmata kujul kehtivad ja kasutuses. Siinkohal ma ei mõista miks Karlströmi [KARLSTRÖM2002] grupp leidis, et metafoori juurutamine keeruline on.

### **4.2.2.2 Lihtsad disainimise vahendid**

Meie kogemus ütleb, et häid disainimise (üldiselt arendust toetavaid vahendeid) pole, või on nad nii kallid, et nende kasutamine vähemalt väikestes firmades ei tasu ära. Kõik võimalikud vahendid nõuavad kõigepealt väga head koolitust ja seejärel hakkavad vahendid suhteliselt julmalt peale suruma oma loogikat (on see kokkuvõttes hea või halb, ei oska öelda) XP-metoodika eelistab võimalikult lihtsate vahendite kasutamist igal pool, kus vähegi võimalik ja seda me ka teeme. Kuigi me oleme proovinud nii mitmegi vahendi võimalusi (Select, Visio, Rational, ModelMaker) oleme nendest kõikidest loobunud. Tahvli, paberi ja pliiatsi kasutamine on osutunud parimaks ja täiesti piisavaks vahendiks rakenduse disainimisel.

### **4.2.2.3 Minimaalne funktsionaalsus**

Siin me jälgime raudset reeglit: kui pole küsitud, et tee. Teeme ainult ja täpselt seda, mille klient on soovilugudesse joonistanud. Kõik see kehtib ka andmebaasi

disainimise kohta. Igaks juhaks ei pane me ühtegi välja andmebaasi tabelisse ega ühtegi funktsionaalsust rakendusse lisaks. Kui vaja – lisame. Selleks, et pääseda andmebaasi versioonide kontrollist, on Multilab'i lisatud andmebaasi *upgrade* funktsionaalsus, mis kontrollib, kas antud andmebaasi *layout* on vastavuses konkreetse rakenduse versiooniga. Kui vastavust pole, lisatakse vajalikud tabelid või tabelitesse vajalikud väljad.

#### 4.2.2.4 Ümberkirjutamine (*Refactoring*)

XP näeb ette järgmise koodi kirjutamise tsükli: testi kirjutamine – funktsionaalsuse realiseerimine – ümberkirjutamine.

Ümberkirjutamise eesmärk on hoida rakenduse disain igal ajahetkel nii lihtne ja lühike kui vähegi võimalik. Ümberkirjutamise käigus minimeeritakse dubleerimisi koodis (tuntud *copy and paste* tehnoloogia kaasprodukt) ning muudetakse disaini.

Kuigi kõik meeskonna liikmed on veendunud ümberkirjutamise vajalikkuses, ei tehta seda pidevalt. Ütleme ausalt. Me tegeleme sellega praegu ainult siis, kui selleks on aega või tekib vääramatu vajadus koodi puhastamiseks ning ümberdisainimiseks. OCS tuuma on selle aja jooksul ümber neli korda; OCS laboriseadmete andmevahetusprotokollide baastaset on ümber kirjutatud kolm korda; Multilab karkassi on ümber kirjutatud 5 korda.

Sellega, et ümberkirjutamisel on positiivne efekt, nagu seda leiab Karlström [KARLSTRÖM2002], ma nõustun. Seda, et ümberkirjutamise sisseviimine arendustegevusse oleks lihtne, ma aga enda kogemustele tuginedes väita ei saa.

### 4.2.3 Programmeerimine

#### 4.2.3.1 Kliendi pidev kohalolek.

Klient liitus arendusgrupiga 2001.aastal. Tegemist on Saksamaa Liiduvabariigi ühe väikese laboratooriumi endise juhataja asetäitjaga, kes on suurepäraselt kursis verelaboratooriumi tegelikest vajadustest. Kliendi alaline resideerimispaik on Saksamaal, kuid see ei takista meil iga päev kliendiga suhelda. Me kasutame suhtlemisvahendina täiesti tasuta saadava *MSN Messenger* võimalusi. Sellel on see väga hea omadus, et kui keegi (või ka mõned) arendajatest suhtlevad kliendiga, saavad teised segamatult (kogu nelja liikmeline arendajate meeskond kasutab ühte tööruumi) oma tööd teha.

Mõned kuud aastas (arenduse etappide algused) viibib meie klient ka Eestis.

Arendustegevuses kliendi kasutamise kogemused langevad täielikult kokku Karlströmi [KARLSTRÖM2002] poolt saadud tulemustega. Selle meetodi juurutamine on väga lihtne – tuleb leida ainult sobiv inimene ning vahendid temale tasumiseks. Samas on kliendi kasu arendusprojekti edu tagamisel raske üle hinnata.

Meie kogemused ütlevad ka seda, et kliendi olematud teadmised nii tarkvara arendamisest kui arendajate kasutatavast tehnilisest keelest on pluss.

#### 4.2.3.2 Kokkulepitud standardid

Systek Informationsystems OÜ algusaastal me töötasime koodi standardite abil. Tegime isegi vastava dokumendi. Tänapäevaks on see dokument kadunud ja ega me sellest ka eriti hooli. Siiski on kuidagi vaikimisi kokkuleppele jõutud mõningates reeglites, mida jälgitakse koodi kirjutamisel:

- Kood peab olema lihtne;
- Heal koodil pole kommentaare v.a (mõningaid asju ikka me kommenteerime);
- Jälgime Delphi “standardeid” kuna kasutame arendusvahendina Borland Delphi arenduskeskkonda;
- Vältime pikki (rohkem kui 50 koodirida) meetodeid ja pikki ridu;
- Nimed (tabelid, klassid, meetodid, funktsioonid) peavad olema arusaadavad ilma kommentaarideta;
- Vältime igasuguseid trikke ja nippe.

Meie arendajad on koodi järgi identifitseeritavad (erinevalt XP-metoodika soovitusetest), kuid kõikide poolt kirjutatud kood on kõikidele teistele arusaadav ja mõistetav. Ja see on peamine. Mingi individuaalsus peab igale programmeerijale meie arvates siiski jääma. Seda, et koodi standardi sisseviimine vastavalt Karlströmi [KARLSTRÖM2002] tulemustele firmas pingutust nõuab langeb kokku ka meie kogemustele. Seda, et see nüüd eriti suurt efekti annaks, me pole leidnud. Seni kuni programmeerija ei nimeta funktsioone ja tabeleid “kilu” ja “kala” (näited reaalsest elust) ning ei kasuta dešifreerimist vajavaid lühendeid (xsk2, aabbcc, ...) ja kirjutab vähemalt nii, et ta ise asjast igal ajal (ka kolme aasta pärast) aru saab, peaks kõik olema ok.

#### 4.2.3.3 Paarisprogrammeerimine

Paarisprogrammeerimist XP mõistes me ei tee. Kuigi paarisprogrammeerimine on kirjanduse järgi väga efektiivne meetod ning väga lihtne juurutada (ka Karlströmi [KARLSTRÖM2002] jõuab sellisele järeldusele), on sellel ka teatavad negatiivsed kõrvaleffektid, mida me ei soovi. Meil on arendajad suhteliselt vabad inimesed. Tulevad tööle millal tahavad ja lahkuvad millal tahavad. Ka on kõikidele arendajatele loodud kodus töötamise võimalused. Paarisprogrammeerimine eeldaks aga ühist tööpäeva algust ning ka ühist tööpäeva lõppu (vähemalt programmeerijate paarile).

Samas on paarisprogrammeerimine tõhus ja hea vahend programmeerite viimiseks ühisele laienele siis, kui neil on vaja koos ühe ja sama projektiga tegeleda. Selles mõttes me seda meetodit rakendame. Seega on meil paarisprogrammeerimine kasutusel mitte niivõrd arendusmeetodina, kuivõrd õpetamise ja kogemuste vahetamise meetodina.

Ehk on paarisprogrammeerimise suhtes eesti arendajatel mingi psühholoogiline tõrge. Antud väitekirja raames läbi viidud arendajate küsitlus näitas, et paarisprogrammeerimisest peeti võrreldes teiste meetoditega kõige vähem lugu (3.8.2.6).

#### 4.2.3.4 Pidev integreerimine

Sarnaselt Karlströmi [KARLSTRÖM2002] tulemustele ütleb ka meie kogemus, et pidev tervikuks integreerimine on lihtne juurutada ning annab efekti. Selleks on vaja ainult versioonihaldustarkvara (meil kasutusel Borland Delphi'ga kaasas olev lisavõimalus ) ning soovitatavalt ka ühte arvutit, kus hoitakse viimast töötavat versiooni. Soovitatav on, et selle arvutiga midagi muud ei tehta. Meie nii ka teeme. Tõsi; vahel on selle pideva integreerimisega ka probleeme, kuid me usume need lahenduvad niipea, kui meil on õnnestunud täielikult juurutada testidel tuginev arendustegevus. Kuni testidel tuginevat arendustegevust järjekindlalt ei rakendata, on vahel (nagu meie kogemus ütleb) probleeme. Samas ei ole me nende probleemide tõttu siiski pidevast integreerimisest loobuma.

Ka Peeter Normak [NORMAK2001] rõhutab tervikuks integreerimist kui ühte olulist faktorit tarkvaraprojekti edu tagamise seisukohalt, kuna hiline tervikuks integreerimine võib kaasa tuua lisaressursside vajaduse liideste ümbertegemiseks.

#### 4.2.3.5 Kood on kollektiivne omand

Jah. Selle sisseviimine on samuti suhteliselt tülikas. Seda leiavad ka Karlströmi [KARLSTRÖM2002] uuritavad. Samas on ikka hea küll, kui iga arendaja on võimeline igat kohta vajaduselt parandama ja täiendama. Praegu firmas aktuaalse Multilab projektiga töötavad korraga kõik neli arendajat.

Kui me olime sunnitud tegema algust selle koodi kollektiivse omandiga, oli ikka parasjagu vaeva ja jama. Asjad läksid kaduma, ei tea kust ilmusid täiesti valed ja vanad koodiosad uude koodi jne. Kuid tasapisi asjad paranesid. Inimesed õpivad - ja väga kiiresti.

Samas kui me ei praegu ei kasutaks pidevat integreerimist ning koodi kollektiivset omandust, oleks probleeme ilmselt palju rohkem.

#### 4.2.3.6 Ületundide vältimine

Ületundidega pole meil kunagi probleeme olnud ja ega me sellest probleeme ka tekita. Nagu juba mainitud on meil arendajad vabad inimesed: tulevad tööle millal tahavad ja lahkuvad millal tahavad. Kui on väsinud, võtavad vaba päeva või kaks. Kui vaja ja jaksavad, töötavad kaua

Probleem on meil aga puhkustega. Planeeri või mitte, korralist puhkust planeeritud ajal on praktiliselt võimatu saada. Selles suhtes tuleb midagi lähiajal ette võtta. Arendajad vajaksid vähemalt kolme nädalat totaalset puhkust ilusal ajal suvel ning nädalast puhkust Jõulude ajal. Seda viimast me oleme ka suutnud tagada.

#### 4.2.4 Testimine.

Testimine on XP kõige tähtsam osa. Kui testimist ei tee, pole ka XP-d. Kuna ka meie pole veel täielikult testidel tuginevat arendustegevust juurutanud, on ka meie XP mitte päris see, mis ta olema peab.

Samas ripuvad testimisest ära ka mitmed teised XP meetodid: koodi ühisomand ja tihe tervikuks integreerimine. Mõlemad nimetatud meetoditest vajavad kontrolli – kas kõik mis enne töötab töötab ka praegu.

Midagi me siiski oleme katsetanud ja proovinud. OCS laboriseadmete protokollide osa on peaaegu täielikult testidega kaetud. Multilab ja QC osad on kaetud osaliselt.

Testimise sisseviimine on raskem, kui esialgu arvata võiks. Ka Karlströmi [KARLSTRÖM2002] grupp on samal arvamusel. Samas on selle efekt ilmselt kõikidest XP meetoditest kõige suurem.

Samas tagab läbimõeldud ja automatiseeritud testimine tarkvarale kõrge kvaliteedi. Kvaliteedikindluse osatähtsust tarkvaraprojekti edu tagamiseks on rõhutatult välja toodud ka Peeter Normaku loengukonspektis [NORMAK2001].

Suhteliselt lihtsam on testimist sisse viia sinna, kus pole tegemist kasutajaliidesega. Selle pärast ongi meil protokollid testidega hästi kaetud. Kokku on protokollide testimiseks kirjutatud 272 automaatselt käivitavat testi. Need testid testivad kokku 17-ne erinevat laboratooriumiseadme protokolle. Esimesele kümnele protokollile sai kirjutatud ainult sobivustestid ja sedagi tagant järele. Nendest kümnest protokollist kaks osutusid reaalses töös hiljem mõningates reaalselt tekkivates olukordades vigasteks. Vigade parandamiseks ja nende ülesotsimiseks sai mõlemale protokollile kirjutatud juurde ligikaudu paarkümmend ühiktesti. Hiljem kirjutatud 7 protokollile on aga kirjutatud korrektselt lähtuvalt testidele tuginevast arendusmetoodikast.

Kasutajaliidese testimine on aga suhteliselt suur peavalu, kuna lihtsalt ei oska ning ka vastav materjal kirjanduse näol et õppida saaks on praktiliselt veel olematu. Siiski on meil kokku kirjutatud 296 testi ja me jätkame pingutusi.

Üks tõsine probleem testimise juurutamisel on mõttemallide muutmises: kuidas ikkagi kirjutada testi enne, kui seda osa, mida testida. Kuigi selle kohta on olemas suhteliselt hästi kirjutatud raamat [BECK2003], on selle loogika omaksvõtt ja selle rakendamine oma igapäevases arendustegevuses suhteliselt vaevaline.

Samas ma näen sellel arendusmetoodikal ja selle juurutamisel suurt perspektiivi nii üldiselt, kui ka oma firmas. Hea meel on, et ka minu kolleegid – arendajad – on sama meelt.

Olgu veel lisatud, et testidel tuginev arendamine suurendab rakenduse kirjutamisele kuluvat aega kirjanduse andmetel 30-50%. Kogemused kinnitavad seda. Samas, annab see kindlustunde, et see, mis on testitud, ka töötab. Eriti oluline on automaatsete testide olemasolu siis, kui midagi tuleb rakenduses muuta.

Ja veel üks märkus: testide kirjutamine ei vabasta kasutajaliideste inimese poolt läbi viidavast visuaalsest testimisest ega rakenduse katsetamist reaalsust matkivates testikeskkondades. Mõlemaid me ka kasutame. Visuaalselt testib kasutajaliideseid klient; laboratooriumi matkimiseks oleme kirjutanud virtuaalseid laboriseadmeid, mis matkivad eelmise päeva reaalsete logide järgi tegelikku olukorda laboratooriumis.

Testide kirjutamiseks ja automatiseerimiseks (vt. Lisa.5) kasutame rakendust DUnit [DUNIT].

### 4.3 Kaugel me oleme SW-CMM teisest küpsustasemest?

Me oleme jõudnud arusaamale, et kvaliteetse tarkvara loomiseks on arendusmetoodika kasutamine hädavajalik ning oleme firmas Systek Informationsystems OÜ alustanud tarkvara arendusmetoodika juurutamisega.

Me õpime, katsetame, omandame ja sobitame erinevaid XP metoodika meetodeid. Me ei kiirusta. Põhiline eesmärk on siiski toota kliendile kasulikku tarkvara.

Meie eesmärk on sammhaaval liikudes jõuda SW-CMM teise taseme (ja edaspidi vastava CMMI taseme) nõuete rahuldamiseni ehk lähema kahe kuni kolme aasta jooksul. Tabelites 32 – 36 on toodud enesehinnanguna meie tänane seis. Kõige tugevamalt on meil täidetud nõuete haldamisega seonduv ja kõige kehvemini kvaliteedi tagamise tagavused.

Suure edasimineku tarkvara arendusprotsessi parandamisele firmas Systek annab testidel tugineva arendusmetoodika järjekindel sisseviimine. Siis muutuvad võimalikuks ja mõttekaks ka meetrikate defineerimine ning rohkem kvantitatiivsete (mõistes objektiivselt mõõdetavate) hinnangute andmine projektile.

Lisas 6 on toodud firma Systek Informationsystems OÜ enesehinnang tarkvaraprojektide kohta lähtuvalt Peeter Normaku [NORMAK2001] loengukonspektis toodud testi alusel. Kuigi selle skaala järgi on meil tarkvara projektidega asjad korras, ei ole me asjadega ise sugugi veel rahul.

	SW-CMM	Enesehinnang
<b>Co1</b>	Nõuete haldamine allub dokumenteeritud reeglitele.	Hea
<b>Ab1</b>	Igal projektil on olemas nõuete analüüsi eest vastutav, kes ka teeb kindlaks ja otsustab, kas antud nõue on realiseeritav riisvaraliselt, tarkvaraliselt, või kuidagi teisiti	Hea
<b>Ab2</b>	Nõuded dokumenteeritakse	Hea
<b>Ab3</b>	Nõuete haldamiseks on eraldatud adekvaatsed ressursid.	Hea
<b>Ab4</b>	Inimesed on koolitatud tarkvara nõudeid haldama	Hea
<b>Ac1</b>	Analüüsimeeskond töötleb ja analüüsib nõudeid enne realiseerimist	Hea
<b>Ac2</b>	Analüüsimeeskond võtab nõuded aluseks edasiste plaanide koostamisel ja tarkvara kavandamisel	Hea
<b>Ac3</b>	Muudatused nõuetes fikseeritakse, töödeldakse, lisatakse projekti ning antakse kõikidele teada.	Hea
<b>Me1</b>	Nõuete haldamise tegevusi mõõdetakse	Osaline
<b>Ve1</b>	Nõuete haldamise protsessi tegevused on tippjuhtkonna tähelepanu all	Hea
<b>Ve2</b>	Projektijuhid tegelevad nõuete haldamise igapäevaste küsimustega.	Hea
<b>Ve3</b>	Kvaliteedi eest vastutajad jälgivad ja auditeerivad nõuete haldamise protsessi	Osaline

Tabel 32. SW-CMM nõuete haldamise tegevused firmas Systek



	<b>SW-CMM</b>	<b>Enesehinnang</b>
<b>Co1</b>	Projekti juht vastutab planeerimise ja vastutuste kooskõlastamiste eest	Hea
<b>Co2</b>	Planeerimisel lähtutakse kehtivatest kirjalikest tavadest ja standarditest	Osaline
<b>Ab1</b>	Projekti jaoks on olemas dokumenteeritud ja kinnitatud tellimus	Hea
<b>Ab2</b>	Arendusplaani väljatöötamise eest on määratud vastutaja	Hea
<b>Ab3</b>	Planeerimise läbiviimiseks on olemas adekvaatsed ressursid	Osaline
<b>Ab4</b>	Inimesed on koolitatud ja nad oskavad tarkvaraprojekte planeerida	Osaline
<b>Ac1</b>	Tarkvara arendajad osalevad projekti ettevalmistavas töös	Hea
<b>Ac2</b>	Kui tarkvara arendamine on üks osa kogu teostatavast projektist, siis alustatakse tarkvara planeerimist samaaegselt ning planeeritakse paralleelselt üldise plaaniga	Puudub vajadus
<b>Ac3</b>	Arendajad osalevad üldistes aruteludes kogu projekti elutsükli vältel;	Hea
<b>Ac4</b>	Vastutuse delegeerimine isikutele ja gruppidele väljaspool organisatsiooni on allutatud dokumenteeritud protseduuridele ja viiakse läbi tippjuhtkonna poolt	Puudub vajadus
<b>Ac5</b>	Tarkvara elutsükkel on jagatud väiksemateks osadeks;	Hea
<b>Ac6</b>	Tarkvara arendusplaani väljatöötamise aluseks on organisatsioonis dokumentaalselt kehtestatud standardid ja reeglid	Osaline
<b>Ac7</b>	Tarkvara väljatöötamise plaan on kirjalik dokument	Osaline
<b>Ac8</b>	Plaan koostamiseks ja täitmise kontrollimiseks on tegevused määratud;	Osaline
<b>Ac9</b>	Plaanis kajastuvad ka peamiste tööde mahud	Osaline
<b>Ac10</b>	Maksumus ja tähtajad planeeritakse lähtuvalt dokumenteeritud reeglitest	Puudulik
<b>Ac11</b>	Arvutusvõimsused planeeritakse lähtuvalt dokumenteeritud reeglitest	Puudulik
<b>Ac12</b>	Projekti ajakava planeeritakse lähtuvalt dokumenteeritud reeglitest	Osaline
<b>Ac13</b>	Maksumusega, ressurssidega, tähtaegadega ja tehniliste aspektidega seotud riskid määratakse, hinnatakse ja dokumenteeritakse	Puudulik
<b>Ac14</b>	Plaanid tehakse nii tarkvara arendamiseks, kui ka arendamist toetavateks tegevusteks	Puudulik
<b>Ac15</b>	Plaanid dokumenteeritakse ja säilitatakse	Osaline
<b>Me1</b>	Planeerimistegevusi mõõdetakse ja saadud tulemusi kasutatakse hilisemates planeerimistöödes	Puudulik
<b>Ve1</b>	Planeerimisega seotud tegevused on tippjuhtkonna tähelepanu all.	Hea
<b>Ve2</b>	Igapäevast planeerimistegevust koordineerib projekti juht.	Hea
<b>Ve3</b>	Kvaliteedi eest vastutav grupp jälgib ja auditeerib planeerimistegevusi selleks, et tagada adekvaatsus.	Puudulik

Tabel 33. SW-CMM tarkvaraprojekti planeerimise tegevused firmas System.

	<b>CMM</b>	<b>Enesehinnang</b>
<b>Co1</b>	Projekti juht on vastutav projekti käekäigu ja tulemuste eest	Hea
<b>Co2</b>	Tarkvaraprojekti monitooringul jälgitakse firma siseseid dokumenteeritud nõudeid ja standardeid	Osaline
<b>Ab1</b>	On olemas dokumenteeritud arendusprojekti plaan	Osaline
<b>Ab2</b>	Projekti juht delegeerib konkreetsete ülesannete täitmiseks vastutust konkreetsetele isikutele ja gruppidele	Hea
<b>Ab3</b>	Projekti monitooringuks on olemas ressursid ja rahalised vahendid	Osaline
<b>Ac1</b>	Dokumenteeritud arendustegevuse plaan on arendusprojekti monitooringu aluseks	Osaline
<b>Ac2</b>	Plaani muutmiseks on kehtestatud dokumenteeritud reeglid	Osaline
<b>Ac3</b>	Tarkvara arendusprojektiga seotud vastustuse määramine ja vastutuse muutmine indiviididele ja gruppidele väljaspool organisatsiooni kuulub tippjuhtkonna vastutusalasse ning on allutatud dokumenteeritud reeglistikule	Puudub vajadus
<b>Ac4</b>	Projekti mõjutavad muudatused vastutusalades kooskõlastatakse arendusgrupiga ja teiste asjast huvitatud gruppidega (kaas arvatud klient)	Hea
<b>Ac5</b>	Tööde mahtusid jälgitakse. Vajadusel korrigeerimised	Hea
<b>Ac6</b>	Saavutusi ja kulutusi jälgitakse. Vajadusel korrigeerimised	Osaline
<b>Ac7</b>	Projekti kriitilisi arvutusvõimsusi jälgitakse. Vajadusel korrigeerimised	Osaline
<b>Ac8</b>	Projekti ajagraafikut jälgitakse. Vajadusel korrigeerimised	Hea
<b>Ac9</b>	Kasutatavaid arendustehnikaid ja nende sobivust jälgitakse. Vajadusel korrigeerimised	Osaline
<b>Ac10</b>	Maksumusega, ressurssidega, tähtaegadega ja arendustehnikatega kaasnevaid riske jälgitakse. Vajadusel korrigeerimised	Osaline
<b>Ac11</b>	Arendustegevusega seotud andmeid kogutakse ja säilitatakse edasise parema planeerimise eesmärgil	Osaline
<b>Ac12</b>	Arendusgrupp teeb perioodilisi kokkuvõtteid arendustegevusest ja arendustegevuse vastavusest plaanidele	Hea
<b>Ac13</b>	Projekti ametlikud läbivaatamised toimuvad perioodiliselt ning vastavalt kehtestatud protseduurireeglitele	Hea
<b>Me1</b>	Arendusprotsessi mõõdetakse. Tulemusi kasutatakse olukorra hindamiseks	Osaline
<b>Ve1</b>	Tarkvara arendamine on pidevalt tippjuhtkonna huviorbiidis;	Hea
<b>Ve2</b>	Tarkvara arendamise igapäevast juhtimist teostab vastava projekti juht;	Hea
<b>Ve3</b>	Kvaliteedi grupp jälgib ja auditeerib arendustegevust.	Puudulik

*Tabel 34. SW-CMM tarkvaraprojekti monitooringu tegevused firmas Systek*

	SW-CMM	Enesehinnang
<b>Co1</b>	Kvaliteedi tagamise aluseks on olemas dokument.	Puudulik
<b>Ab1</b>	Kvaliteedi eest vastutajad on olemas.	Osaline
<b>Ab2</b>	Ressursid ja rahalised vahendid kvaliteedi tagamiseks on olemas	Puudulik
<b>Ab3</b>	Inimesed on kvaliteedi tagamiseks koolitatud	Puudulik
<b>Ac1</b>	Projekti kvaliteedi plaan on koostatud vastavalt firmas kehtivatele dokumenteeritud nõuetele ja standarditele	Puudulik
<b>Ac2</b>	Kvaliteedi tagamiseks on olemas plaan	Puudulik
<b>Ac3</b>	Kvaliteedi eest vastutavad inimesed osalevad plaanide, standardite ja protseduuride väljatöötamise ja läbivaatamise juures	Puudulik
<b>Ac4</b>	Kvaliteedi eest vastutavad inimesed vaatavad pidevalt läbi arendustegevusi, et kindlustada vastavus nõuetega	Osaline
<b>Ac5</b>	Kvaliteedi grupp auditeerib väljatöötatud tarkvara	Puudulik
<b>Ac6</b>	Kvaliteedi grupp koostab perioodilisi aruandeid ja teeb need arendusgrupile teatavaks	Puudulik
<b>Ac7</b>	Puudujäägid töös ja produktis kõrvaldatakse vastavalt kooskõlastatud ja dokumenteeritud protseduurireeglitele;	Osaline
<b>Ac8</b>	Kvaliteedi grupp teeb tihedat koostööd kliendi kvaliteedi grupiga andes ülevaateid ja koostades raporteid;	Puudulik
<b>Me1</b>	Kvaliteedi tagamise tegevusi mõõdetakse ja hinnatakse nende otstarbekust;	Puudulik
<b>Ve1</b>	Kvaliteedi tagamise protsess on pideva tippjuhtkonna tähelepanu all;	Osaline
<b>Ve2</b>	Kvaliteedi tagamise igapäevast tööd korraldab projekti juht;	Osaline
<b>Ve3</b>	Sõltumatud eksperdid hindavad perioodiliselt kvaliteedi alast tegevust firmas.	Puudulik

Tabel 35. SW-CMM kvaliteedi tagamise tegevused firmas Systek.

	SW-CMM	Enesehinnang
<b>Co1</b>	Konfiguratsioonide (nii loodava kui ka kasutatava tarkvara) haldamise aluseks firmas on vastav dokument;	Osaline
<b>Ab1</b>	Konfiguratsioonide haldamine on allutatud kontrollile ning on olemas selle eest vastutavad isikud	Osaline
<b>Ab2</b>	Iga projekti juures on loodud grupp, kes koordineerib ja viib läbi konfiguratsioonide alast tööd	Osaline
<b>Ab3</b>	On eraldatud adekvaatsed ressursid ja rahalised vahendid selleks, et konfiguratsioonide haldamisega seotud tegevusi läbi viia	Osaline
<b>Ab4</b>	Konfiguratsioonide haldamisega ja konfiguratsiooni muudatuste läbiviimisega tegelevad inimesed on saanud vastava koolituse	Osaline
<b>Ab5</b>	Tarkvara arendusmeeskonna liikmed ja teised asjast huvitatud osapooled on koolitatud konfiguratsioonide haldamise vajadustest lähtuvalt	Osaline

Ac1	Iga tarkvaraprojekti jaoks on koostatud konfiguratsioonide haldamise plaan vastavalt firmas kokkulepitud ja dokumenteeritud protseduuri reeglitele	Osaline
Ac2	Tarkvaraprojekti konfiguratsiooni plaan on aluseks konfiguratsioonidega seotud tegevuste läbiviimisel	Hea
Ac3	On olemas konfiguratsioonide haldamise süsteem ja inimeste oskused seda süsteemi kasutada	Hea
Ac4	On määratud tarkvara tooted, millede konfiguratsioonid peavad olema hallatud;	Hea
Ac5	Muudatuste nõuded ja probleemid iga konfiguratsioonidega seotud küsimuse korral on määratud, fikseeritud, läbi vaadatud, kinnitatud ning säilitatud vastavalt dokumenteeritud protseduuridele;	Osaline
Ac6	Konfiguratsioonide muudatused viiakse läbi ja kontrollitakse vastavalt dokumentides kehtestatud korrale;	Osaline
Ac7	Väljatöötatud tarkvara konfiguratsioonide kontroll ja evitamine on allutatud dokumenteeritud protseduuri reeglitele;	Osaline
Ac8	Konfiguratsioonide kohta peetakse arvet vastavalt dokumenteeritud protseduuridele;	Osaline
Ac9	Standardsed raportid konfiguratsiooni muudatuste teavitamise kohta on välja töötatud ja asjaosalistele kättesaadavaks tehtud;	Osaline
Ac10	Tarkvara konfiguratsioonide audit viiakse läbi vastavalt dokumentaalselt kehtestatud korrale.	Puudulik
Me1	Konfiguratsioonide haldamise tegevusi mõõdetakse;	Puudulik
Ve1	Konfiguratsioonide haldamine on allutatud tippjuhtkonna pideva tähelepanu alla;	Hea
Ve2	Konfiguratsioonide haldamise igapäevaste tegevustega tegeleb projekti juht;	Hea
Ve3	Konfiguratsioonide eest vastutav grupp auditeerib perioodiliselt kasutatavat tarkvara selleks, et veenduda dokumentide vastavust reaalsusele;	Osaline
Ve4	Tarkvara kvaliteedi grupp jälgib ja auditeerib perioodiliselt konfiguratsiooniga seotud tegevuste vastavust reaalsele vajadusele.	Puudulik

Tabel 36. SW-CMM konfiguratsioonide haldamise tegevused firmas Syspek.

## KOKKUVÕTE

Tarkvara arendusmetoodika juurutamine firmas ei ole ühekordne tegevus. Seda ei saa teha hoogtööna. Ka ei ole mõtet (ja vist pole ka võimalik) juurutada arendusmetoodikat lihtsalt arendusmetoodika enese pärast. Kui firmas pole jõutud arusaamisele, et metoodika rakendamine arendustegevuses aitab lõppkokkuvõttes luua kiiremini, odavamalt ja kvaliteetsemat tarkvara, pole arendusmetoodika juurutamisele ja selle praktilisele kasutamisele tarkvaraarenduses mõtet ei aega ega vaeva kulutada [BECK&FOWLER2001]. Hea aeg metoodika juurutamiseks on saabunud siis, kui nii arendajad, juhid kui ka firma omanikud on kõik jõudnud arusaamisele, et ilma metoodikata enam hakkama ei saa. Sellele tõdemusele jõudmiseks (veendumuseks, mitte raamatutest loetud ja päheõpitud tõeks) on aga vaja praktilist (ka teatud hulka negatiivseid) arendustegevuse kogemusi.

Kui arendusmetoodika vajalikkusest pole aru saanud arendajad ning ei hakka seda praktiliselt kasutama, või pole arendusmetoodika vajalikkusest aru saanud juhid (omanikud, tegevjuhid) ning ei varusta arendusprojekte arendusmetoodika korrektseks läbiviimiseks vajalike ressurssidega (eelkõige raha ja aeg) jääb metoodika seisma ainult firma reklaami stiilis: “Meie firma kasutab arendusmetoodikana ... metoodikat” või “Meie firma arendusmetoodika vastab ... standarditele”. Kahuks pole sellistest reklaamlausest kasu ei arendajatel ega ka tarkvara tellijatel.

Arendusmetoodika juurutamisega mitte ei lõpe, vaid alles algab lõputu töö arendustegevuse muutmiseks kvaliteetsemaks ning tulutoovamaks.

Käesolevas magistritöös on püütud üldistada XP-metoodika juurutamise kogemusi firmas *Systek Information Systems OÜ*. Ma valisime XP metoodika, kuna XP:

- On mõeldud väikestele ja keskmise suurusega arendusmeeskondadele;
- Ei vaja erilist bürokraatiat;
- Ei vaja erilisi ja kalleid vahendeid;
- On “tervest mõistusest” lähtuv ja arusaadav;
- Sisaldab hästi läbimõeldut ja lihtsat testimise metoodikat
- Sisaldab hästi läbimõeldut ja lihtsat planeerimise metoodikat;
- Sisaldab hästi läbimõeldut ja lihtsat tarkvara arendustegevuse mõõtmise metoodikat;
- Hindab inimesi enam kui reegleid ja protseduure;
- Hindab töötavat tarkvara enam kui dokumentatsiooni;
- Puuduvad karmid reeglid ja on võimalik järkjärguline juurutamine.

Käesolevates teesides olen ma püüdnud näidata, et XP-metoodika meetodid:

- Võimaldavad rahuldada SW-CMM teise taseme nõudeid;
- On küsitletud eesti tarkvaraarendajate arvates olulised arendusprojekti edukust silmas pidades;
- Sobivad kasutada enamiku küsitletud eesti arendajate arendusprojektides

Ka võib antud töö tulemusena vähemal kaudselt järeldada, et enamik Eestis läbiviidavatest arendusprojektidest SW-CMM teise taseme nõudeid ei rahulda. Küll aga on küsitletud eesti tarkvaraarendajate arvates SW-CMM teise taseme nõuded

arendusprojektidele edu tagavad. Seega võiks SW-CMM teise taseme nõuete täitmine olla piisavalt ambitsioonikas väljakutse paljudele eesti tarkvarafirmadele. Miks mitte seda teha XP-metoodikat juurutades.

Lähtuvalt antud töös läbiviidud küsitlusele praktilistest tarkvaraarenduse kogemustest võiks moodustada järgmised hüpoteetilised eesti arendusprojektide küpsustasemed:

- **SW-CMM-2-nõuded** – sellel tasemel töötavates projektides on olemas personaalse vastutusega isik, arendajad osalevad projektiga seotud aruteludes ning toimub tarkvarale esitatavate nõuete haldamine. Suurem osa eesti tarkvarafirmadest töötab eeldatavalt sellel tasemel;
- **SW-CMM-2-plaanid** – sellel tasemel töötavates projektides on (lisaks küpsustaseme SW-CMM-2-nõuded omadustele) eraldatud projekti läbiviimiseks adekvaatsed ressursid, inimesed on koolitatud ning tegeletakse projekti planeerimisega. Ka sellel tasemel peaks töötama piisav hulk tarkvarafirmasid Eestis;
- **SW-CMM-2-tulemused** – sellel tasemel töötavates projektides on lisaks nõuete haldamisele ja projekti planeerimisele kaetud ka projekti monitooringu tegevused ning mõningad kvaliteedi tagamise ning konfiguratsioonide haldamise tegevused. Kui üldse, siis on sellel tasemel töötavaid firmasid Eestis vähe;
- **SW-CMM-2** – sellel tasemel töötavates projektides on rahuldatud kõik SW-CMM teise taseme nõuded. Sellel tasemel töötavaid firmasid läbiviidud empiirilise uuringuga hõlmatud firmade hulgas ei olnud.

Toon veel ära minu arvates olulised metoodika juurutamise võimalikud karid:

- Vähene motivatsioon metoodikat juurutada;
- Inimeste mugavus;
- Vähene järjekindlus metoodika juurutamisel;
- Programmeerijate iseloomudest tekkiv vastuseis;
- Soov liiga kiiresti näha tulemusi;
- Soov ilma vajalike ressurssideta metoodikat juurutada;
- Soov ilma inimesi koolitamata ning motiveerimata metoodikat juurutada;
- Liiga kiire juurutamine.

Olgu lõpetuseks toodud üks lõik Kent Beck'i poolt Addison-Wesley raamatuteseria "The XP Series" nõuandja eessõnast ...

"Kuigi XP-d esitatakse tihti kui loetelu praktilistest tegevustest, et ole XP finiši joon . Ei ole võimalik saavutada paremaid tulemusi XP tegevusi juurutades ja viimistledes seni kuni te pole jõudnud tõdemuseni: XP on stardijoon. XP esitab küsimuse – kui vähe on võimalik teha selleks, et ikkagi veel toota suurepärasest tarkvara. "

Kui ei ole töötavat tarkvara, on kõik teised asjad mitteolulised.

Excuse me, I gotta go program [BECK2000].

## **TÖÖ VÕIMALIKUD EDASIARENDAMISED**

Pakun välja neli probleemi, millega kas antud tööst lähtuvalt või antud töös puudutatud küsimustega kaasnevalt võiks tegeleda.

### **Milline on eesti tarkvaraarendaja?**

Soomlased on uurinud ja uurivad oma IT inimesi. Miks mitte uurida eesti tarkvaraarendajaid? Kes nad on? Kellele nad töötavad? Millistes firmades (suured, keskmised, väikesed) nad töötavad? Millised on nende põhiprobleemid? Kas nad eristuvad kuidagi teistest (tavainimestest)? Kas nad eristuvad kuidagi oma (Euroopa, Ameerika) ametivendadest? Millised on nende teadmised ja oskused võrreldes Euroopa ja Ameerika kolleegidega? ... Küsimusi võiks olla palju.

### **Milline on tarkvara arendusprojektide küpsustase Eestis?**

Kuidas Eestis tarkvara toodetakse? Millisel küpsustasemel seda tehakse? Millised on arengusuunad, kuhu firmad pürgivad?

### **Kas on mõttekas ja kui siis kuidas ühendada omavahel XP-metoodika kliendi soovilood ning kasutaja juhend?**

XP-metoodika järgi kuuluvad kliendi poolt kirjutatud soovilood, peale realiseerimist, hävitamisele. Kas ja kuidas saaks kasutada kliendi soovilugusid kasutaja juhendi kirjutamiseks või ehk veel parem - kuidas panna klienti kasutaja soovilugu kirjutama ning programmeerijaid selle kliendi poolt kirjutatud juhendi järgi tarkvara programmeerima.

### **Kuidas testidel tuginevat arendusmetoodikat läbi viia?**

Eesmärgiks oleks testidel tugineva arendusmetoodika süvaanalüüs lähtuvalt testimise teooriast ning testimise mustrite väljatöötamine. Analüüsida tuleks ka testidel tugineva arendusmetoodika eeliseid ning puudusi võrreldes testimiseks mõeldud spetsiaalvahenditega.

## KASUTATUD KIRJANDUS

1. [EE6] Eesti Entsüklopeedia, 2.trükk, VI köide. –Tln.: Valgus 1992.
2. [LEIS2001] **Paul Leis. Agiilmetoodikad IT juhile .**  
<http://www.tud.ttu.ee/material/leis/Tarkvaratehnika/2002/Agiilmetoodikad%20IT%20juhile%201.doc>
3. [NIGLAS] **Katrin Niglas**, Klasteranalüüs, käsikiri.
4. [NORMAK2001] **Peeter Normak**, Tarkvaraprojektide juhtimine. Loengukonspekt. TPÜ, Tallinn 2001.
5. [TEPANDI2003] **Jaak Tepandi**, Tarkvara kvaliteet ja standardid, Kursuse versioon 09. 01. 2003, TTÜ informaatikainstituut, Tallinn 2003.
6. [SEEBA2001] **Asko Seeba**, “Unifitseeritud tarkvaraarendamise protsess ja selle rakendamise juhtumianalüüs”, Magistritöö, 2001, Tartu Ülikool, Tartu.
7. [BECK1999] **Kent Beck**, Optional Scope Contracts, 1999,  
<http://www.xprogramming.com/ftp/Optional+scope+contracts.pdf>
8. [BECK2000] **Kent Beck, Extreme Programming Explained**, Embrace Change, Addison-Wesley, 2000, ISBN 0201616416
9. [BECK&FOWLER2001] **Kent Beck, Martin Fowler. Planning Extreme Programming**, Addison-Wesley, 2001, ISBN 0201710919
10. [BECK2003] **Kent Beck, Test-Driven Development: By Example**, Addison-Wesley, 2003, ISBN 0-321-14653-0
11. [BEEDLE] **Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, Jeff Sutherland. SCRUM: An extension pattern language for hyperproductive software development.**  
[http://jeffsutherland.com/scrum/scrum\\_plop.pdf](http://jeffsutherland.com/scrum/scrum_plop.pdf)
12. [CMM] **Capability Maturity Model® (SW-CMM®) for Software**  
<http://www.sei.cmu.edu/cmm/cmm.sum.html>
13. [CMMI] **CMMI® General Information (SW-CMM®) for Software**  
<http://www.sei.cmu.edu/cmmi/general/>
14. [CMMI-SW] **CMMI<sup>SM</sup> for Software Engineering.**  
<http://www.sei.cmu.edu/cmmi/models/sw-continuous.doc>
15. [CMM1993] **Key Practices of the Capability Maturity Model<sup>SM</sup>**, Version 1.1. <http://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr25.93.pdf>
16. [COCKBURN] **Alistair Cockburn kodulehekül**  
<http://www.members.aol.com/acockburn/>



17. [DSDM] **"Dynamic System Development Method" metoodika kodulehekülg** <http://www.dsdm.org/>
18. [DUNIT] DUnit. Extreme testing for Delphi.  
<http://sourceforge.net/projects/dunit/>
19. [DYMOND1998] **Kenneth M.Dymond. A Guide to the CMM. Understanding the Capability Maturity Model for Software.** 1998. Process Training International Inc. Annapolis, Maryland 21404 USA
20. [FDD] **Feature Driven Development metoodika kodulehekülg**  
<http://www.featuredrivendevelopment.com/>
21. [FOWLER2000] **Martin Fowler, Put Your Process on a Diet**, Software Development Magazine, December 2000  
<http://www.sdmagazine.com/documents/s=737/sdm0012a/0012a.htm?temp=kP2hd6O8Yb>
22. [FRÜHAUF2001] Karol Frühauf, Is there something like light maintenance? 5<sup>th</sup> European Conference on Software Maintenance and Reengineering, Lissabon, March 14, 2001.
23. [HIGHMATORGS] **List of High Maturity Organizations.**  
<http://www.sei.cmu.edu/activities/cmm/high-maturity/HighMatOrgs.pdf>
24. [JOHNSON1998] **Donna L. Johnson and Judith G. Brodman**, "Applying the CMM to Small Organizations and Small Projects," Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.
25. [KARLSTRÖM2002] **Daniel Karlström**, "Increasing Involvement in Software Process Improvement", Lund 2002,  
<http://www.lucas.lth.se/publications/pub2002/021106daniel.pdf>.
26. [LARMAN1998] **Craig Larman, Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design**, Prentice Hall PTR, 1998, *ISBN 0-13-748880-7*
27. [LARMAN&BASIL2003] **Craig Larman, Victor R. Basil**, "Iterative and Incremental Development: A Brief History", Computer, June 2003, pp. 47-56.
28. [PAULK1998] **Mark C. Paulk. Using CMM in small organizations.**  
<http://www.sei.cmu.edu/activities/cmm/papers/cmm-small.pdf>
29. [PAULK1999] **Mark C. Paulk. Analyzing the Conceptual Relationship Between ISO/IEC 15504 (Software Process Assessment) and the Capability Maturity Model for Software.** 1999 International Conference on Software Quality, Cambridge, MA.  
<http://www.sei.cmu.edu/activities/cmm/papers/iso15504-cmm99.pdf>

30. [PAULK2001] **Mark C. Paulk. Extreme Programming from a CMM Perspective.** Paper for XP Universe, Raleigh, NC 23-25 July 2001.  
<http://www.sei.cmu.edu/cmm/papers/xp-cmm-paper.pdf>
31. [POPPENDIECK2003] **Mary Poppendieck, Tom Poppendieck. Lean Software Development: An Agile Toolkit for Software Development Managers,** Addison Wesley, 2003. <http://www.poppendieck.com/ld.htm>
32. [ROYCE1970] **W. Royce,** “Managing the Development of Large Software Systems”, Proc. Weston, IEEE CS Press, 1970, pp 328-339
33. [SCRUM] **SCRUM Software Development Process.**  
<http://www.controlchaos.com/scrumwp.htm>
34. [SPICE] Software Process Improvement and Capability dEtermination.  
<http://www.sqi.gu.edu.au/spice/>
35. [SUTHERLAND2002] **Jeff Sutherland, SCRUM Hyperproductive Software Development Method.** <http://jeffsutherland.com/scrum/>
36. [TLSM1998] Top-Level Standards Map.  
<http://www.sei.cmu.edu/activities/cmm/docs/standards-map.pdf>
37. [XP] **Extreme Programming:** A gentle introduction.  
<http://www.extremeprogramming.org/>

## SUMMARY

The question of software development methodologies, being as old or nearly as old as software development itself, still is actual today. Agile methodologies, development standards and project maturity models are consequent being discussed on scientific conferences and are the subjects of prestigious magazines.

In my masters thesis I generalize the experiences of bringing in Extreme Programming (XP) methodology in a small Estonian software company. In the first part I give a short overview on software development process, the criteria's of evaluating a development process and modern developing methodologies suitable for small companies.

The second part concentrates on evaluating XP methodology using the second level of SW-CMM (Capability Maturity Model for Software). I also show how the demands of SW-CMM second level can be satisfied using XP methodology.

In the third part I analyze the experiences and attitude of Estonian software developers over the demands of SW-CMM second level and methods used in XP-methodology.

The fourth part of my master's thesis is a case study, where I analyze the experiences achieved during introducing XP methodologies based on the example of Syspek Informationsystems OÜ.

Introducing software development methodology is not a one-time activity and it cannot be done as shock work. It is not reasonable (nor possible) bringing in development methodology just for the sake of it. There is no point spending time and effort on bringing in and practical use of a development methodology, if the company has not come to understanding that a methodology in development can help produce software faster, cheaper and with better quality [BECK&FOWLER2001]. When all the developers, leaders and even owners have understood that they cannot go on without a methodology, then it is a good time to bring one in. But in order reach that understanding and certainty, practical experiences of developing, even negative of some measure, are needed.

If the necessity of development methodology is not understood by developers, who would not start using it in practice, or by leaders (owners, managers), who would not start providing projects with needed resources (time and money above all) for correctly carrying out the methodology, then the methodology will most likely only produce the slogans like "Our firm uses ... methodology" or "The development methodology of our firm measures up to ... standards" which really are not beneficial for the developers nor the clients.

Introducing a methodology in development process does not mean that the work is done, on the contrary - it marks the beginning of long and hard work changing development process into better quality and more rewarding.

In my masters thesis I have tried to generalize the experiences introducing the XP-methodology in a software company Systek Informationsystems OÜ. We chose XP, because XP:

- Is suitable for small- and middle-sized development teams;
- Works without special bureaucracy;
- Works without expensive and distinguished tools;
- Is understandable to "common sense";
- Includes simple and well-advised methods for testing developed software;
- Includes forethoughtful software project planning methods;
- Includes well considered project tracking and oversight methods;
- Respects individuals and interactions over processes and tools;
- Respects working software over comprehensive documentation;

In current thesis I have shown that:

- XP methodology satisfies the needs of SW-CMM maturity level 2 (The repeatable process);
- Most Estonian developers respect XP methods (pair programming is an exception);
- XP methodology is usable in most software projects in Estonia.

Based on my thesis it is possible to conclude that most of Estonian software projects do not meet the needs of SW-CMM maturity level 2. However, the questioned Estonian software developers ensure that following the needs of SW-CMM maturity level 2 can bring success to development projects. For most companies in Estonia the challenge of implementing XP and starting to develop software according to SW-CMM maturity level 2 can be quite an ambition in the foreseeable future.

By current empirical studies it is possible to form the following hypothetical maturity sublevels of Estonian software companies:

- **SW-CMM-2-requirements.** Development projects include a project manager with personal responsibilities on the project; the developers are well informed and participate in general project discussions; the software requirements are mostly managed. Most Estonian software companies supposedly work on this level.
- **SW-CMM-2-plans.** In addition to prior adequate resources to carry on the project are provided, people are trained and the project plans are being created. There should also be quite a sufficient amount of companies in Estonia working on this level;
- **SW-CMM-2-results.** In addition to managed requirements and planned projects the software project activities are monitored and control actions, also some software quality assurance and configuration management activities are being carried out. If any, then only a few companies in Estonia are working on this level;
- **SW-CMM-2.** This is the complete SW-CMM maturity level 2. By current empirical studies there are no companies in Estonia working on this level.

Some possible drawbacks, which I consider important:

- Scarce motivation and little consistency to carry out the implementation of a methodology;
- Little development experience;
- Negligence;
- Opposition of characters of individuals;
- Wish to see the results too fast;
- Introducing without adequate resources;
- Introducing without motivating and educating developers;
- Too fast introducing.

Following is from Kent Beck's (series advisor) foreword to Addison-Wesley "The XP Series" books ...

"Although XP is often presented as a list of practices, XP is not a finish line. You don't get better and better grades at doing XP until you finally receive the coveted gold star. XP is a starting line. It asks the question 'How little can we do and still build great software?' "

Working software is the primary measure of progress. If no working software all other things are mean less.

Excuse me, I gotta go program [BECK2000].

# LISAD

## Lisa.1. SW-CMM teise taseme nõuded.

### Nõuete haldamine (RM – Requirements Management)

#### *Requirements Management*

The purpose of Requirements Management is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project.

#### *Goals*

- G1 System requirements allocated to software are controlled to establish a baseline for software engineering and management use.
- G2 Software plans, products, and activities are kept consistent with the system requirements allocated to software.

#### *Commitment to perform*

- Co1 The project follows a written organizational policy for managing the system requirements allocated to software.
  - 1 The allocated requirements are documented
  - 2 The allocated requirements are reviewed by the software managers and other affected groups (system test, software engineering, system engineering, software quality assurance, software configuration management, documentation support)
  - 3 The software plans; work products, and activities are changed to be consistent with changes to the allocated requirements.

#### *Ability to perform*

- Ab1 For each project, responsibility is established for analyzing the system requirements and allocating them to hardware, software, and other system components.
  - 1 Managing and documenting the system requirements and their allocation throughout the project's life
  - 2 Effecting changes to the system requirements and their allocation.
- Ab2 The allocated requirements are documented.
  - 1 The nontechnical requirements (i.e., the agreements, conditions, and/or contractual terms) that affect and determine the activities of the software project.
  - 2 The technical requirements for the software (end user, operator, support, or integration functions; performance requirements; design constraints; programming language; and interface requirements. )
  - 3 The acceptance criteria that will be used to validate that the software products satisfy the allocated requirements.
- Ab3 Adequate resources and funding are provided for managing the allocated requirements.
  - 1 Individuals who have experience and expertise in the application domain and in software engineering are assigned to manage the allocated requirements.
  - 2 Tools to support the activities for managing requirements are made available. (Spreadsheet programs, tools for configuration management, tools for traceability, and tools for test management.)
- Ab4 Members of the software engineering group and other software-related groups are trained to perform their requirements management activities.

#### *Activities performed*

- Ac1 The software engineering group reviews the allocated requirements before they are incorporated into the software project.
  - 1 Incomplete and missing allocated requirements are identified.

- 2 The allocated requirements are reviewed to determine whether they are: feasible and appropriate to implement in software, clearly and properly stated, consistent with each other, and testable.
  - 3 Any allocated requirements identified as having potential problems are reviewed with the group responsible for analyzing and allocating system requirements, and necessary changes are made.
  - 4 Commitments resulting from the allocated requirements are negotiated with the affected groups.
- Ac2 The software engineering group uses the allocated requirements as the basis for software plans, work products, and activities.
- 1 The allocated requirements are managed and controlled.
  - 2 The allocated requirements are the basis for the software development plan.
  - 3 The allocated requirements are the basis for developing the software requirements.
- Ac3 Changes to the allocated requirements are reviewed and incorporated into the software project.
- 1 The impact to existing commitments is assessed, and changes are negotiated as appropriate.
  - 2 Changes that need to be made to the software plans, work products, and activities resulting from changes to the allocated requirements are: identified, evaluated, assessed for risk, documented, planned, communicated to the affected groups and individuals, and tracked to completion.

### **Measurement and analysis**

Me1 Measurements are made to determine the status of Requirements Management.

- 1 Examples of measurements include: status of each of the allocated requirements; change activity for the allocated requirements; and cumulative number of changes to the allocated requirements, including total number of changes proposed, open, approved, and incorporated into the system baseline.

### **Verifying implementation**

- Ve1 The activities for managing the allocated requirements are reviewed with senior management on a periodic basis.
- Ve2 The activities for managing the allocated requirements are reviewed with the project manager on both a periodic and event-driven basis.
- Ve3 The software quality assurance group reviews and/or audits the activities and work products for managing the allocated requirements and reports the results.
- 1 The allocated requirements are reviewed, and problems are resolved before the software engineering group commits to them.
  - 2 The software plans; work products, and activities are appropriately revised when the allocated requirements change.
  - 3 Changes to commitments resulting from changes to the allocated requirements are negotiated with the affected groups.

## **Tarkvaraprojekti planeerimine (SPP – Software Project Planning).**

### ***Software Project Planning***

The purpose of Software Project Planning is to establish reasonable plans for performing the software engineering and for managing the software project.

#### ***Goals***

- G1 Software estimates are documented for use in planning and tracking the software project.
- G2 Software project activities and commitments are planned and documented.
- G3 Affected groups and individuals agree to their commitments related to the software project.

#### ***Commitment to perform***

*Co1* A project software manager is designated to be responsible for negotiating commitments and developing the project's software development plan.

*Co2* The project follows a written organizational policy for planning a software project.

- 1 The system requirements allocated to software are used as the basis for planning the software project.
- 2 The software project's commitments are negotiated between: the project manager, the project software manager, and the other software managers.
- 3 Involvement of other engineering groups in the software activities is negotiated with these groups and is documented.
- 4 Affected groups review the software projects: software size estimates, effort and cost estimates, schedules, and other commitments.
- 5 Senior management reviews all software project commitments made to individuals and groups external to the organization.
- 6 The project's software development plan is managed and controlled.

***Ability to perform***

*Ab1* A documented and approved statement of work exists for the software project.

- 1 The statement of work covers: scope of the work, technical goals and objectives, identification of customers and end users, imposed standards, assigned responsibilities, cost and schedule constraints and goals, dependencies between the software project and other organizations, resource constraints and goals, and other constraints and goals for development and/or maintenance.
- 2 The statement of work is reviewed by: the project manager, the project software manager, the other software managers, and other affected groups
- 3 The statement of work is managed and controlled.

*Ab2* Responsibilities for developing the software development plan are assigned.

- 1 The project software manager, directly or by delegation, coordinates the project's software planning
- 2 Responsibilities for the software work products and activities are partitioned and assigned to software managers in a traceable, accountable manner.

*Ab3* Adequate resources and funding are provided for planning the software project.

- 1 Where feasible, experienced individuals, who have expertise in the application domain of the software project being planned, are available to develop the software development plan.
- 2 Tools to support the software project planning activities are made available. (Spreadsheet programs, estimating models, and project planning and scheduling programs.)

*Ab4* The software managers, software engineers, and other individuals involved in the software project planning are trained in the software estimating and planning procedures applicable to their areas of responsibility.

***Activities performed***

*Ac1* The software engineering group participates on the project proposal team.

- 1 The software engineering group is involved in: proposal preparation and submission, clarification discussions and submissions, and negotiations of changes to commitments that affect the software project.
- 2 The software engineering group reviews the project's proposed commitments. (Examples of project commitments include: the project's technical goals and objectives; the system and software technical solution; the software budget, schedule, and resources; and the software standards and procedures.)

*Ac2* Software project planning is initiated in the early stages of, and in parallel with, the overall project planning.

*Ac3* The software engineering group participates with other affected groups in the overall project planning throughout the project's life.

- 1 The software engineering group reviews the project-level plans.

*Ac4* Software project commitments made to individuals and groups external to the organization are reviewed with senior management according to a documented procedure.

*Ac5* A software life cycle with predefined stages of manageable size is identified or defined.



- 1 Examples of software life cycles include: waterfall, overlapping waterfall, spiral, serial build, and single prototype/overlapping waterfall.
- Ac6* The project's software development plan is developed according to a documented procedure.
- 1 The software development plan is based on and conforms to: the customer's standards, as appropriate; the project's standards; the approved statement of work; and the allocated requirements.
  - 2 Plans for software-related groups and other engineering groups involved in the activities of the software engineering group are negotiated with those groups, the support efforts are budgeted, and the agreements are documented.
  - 3 Plans for involvement of the software engineering group in the activities of other software-related groups and other engineering groups are negotiated with those groups, the support efforts are budgeted, and the agreements are documented.
  - 4 The software development plan is reviewed by: the project manager, the project software manager, the other software managers, and other affected groups.
  - 5 The software development plan is managed and controlled.
- Ac7* The plan for the software project is documented. The software development plan covers:
- 1 The software project's purpose, scope, goals, and objectives.
  - 2 Selection of a software life cycle.
  - 3 Identification of the selected procedures, methods, and standards for developing and/or maintaining the software. (Examples of software standards and procedures include: software development planning, software configuration management, software quality assurance, software design, problem tracking and resolution, and software measurement.)
  - 4 Identification of software work products to be developed.
  - 5 Size estimates of the software work products and any changes to the software work products.
  - 6 Estimates of the software project's effort and costs.
  - 7 Estimated use of critical computer resources.
  - 8 The software project's schedules, including identification of milestones and reviews.
  - 9 Identification and assessment of the project's software risks.
  - 10 Plans for the project's software engineering facilities and support tools.
- Ac8* Software work products that are needed to establish and maintain control of the software project are identified.
- Ac9* Estimates for the size of the software work products (or changes to the size of software work products) are derived according to a documented procedure. The procedure typically specifies that:
- 1 Size estimates are made for all major software work products and activities.
  - 2 Software work products are decomposed to the granularity needed to meet the estimating objectives.
  - 3 Historical data are used where available.
  - 4 Size estimating assumptions are documented.
  - 5 Size estimates are documented, reviewed, and agreed to.
- Ac10* Estimates for the software project's effort and costs are derived according to a documented procedure. This procedure typically specifies that:
- 1 Estimates for the software project's effort and costs are related to the size estimates of the software work products (or the size of the changes).
  - 2 Productivity data (historical and/or current) are used for the estimates when available; sources and rationale for these data are documented. The productivity and cost data are from the organization's projects when possible. The productivity and cost data take into account the effort and significant costs that go into making the software work products. (Examples of significant costs that go into making the software work products include: direct labor expenses, overhead expenses, travel expenses, and computer use costs.)
  - 3 Effort, staffing, and cost estimates are based on past experience. Similar projects should be used when possible. Time phasing of activities is derived. Distributions of the effort, staffing, and cost estimates over the software life cycle are prepared.

- 4 Estimates and the assumptions made in deriving the estimates are documented, reviewed, and agreed to.
- Ac11* Estimates for the project's critical computer resources are derived according to a documented procedure. This procedure typically specifies that:
- 1 Critical computer resources (computer memory capacity, computer processor use, and communications channel capacity) for the project are identified.
  - 2 Estimates for the critical computer resources are related to the estimates of: the size of the software work products, the operational processing load, and the communications traffic.
  - 3 Estimates of the critical computer resources are documented, reviewed, and agreed to.
- Ac12* The project's software schedule is derived according to a documented procedure. This procedure typically specifies that:
- 1 The software schedule is related to: the size estimate of the software work products (or the size of changes), and the software effort and costs.
  - 2 The software schedule is based on past experience. Similar projects are used when possible.
  - 3 The software schedule accommodates the imposed milestone dates, critical dependency dates, and other constraints.
  - 4 The software schedule activities are of appropriate duration and the milestones are of appropriate time separation to support accuracy in progress measurement.
  - 5 Assumptions made in deriving the schedule are documented.
  - 6 The software schedule is documented, reviewed, and agreed to.
- Ac13* The software risks associated with the cost, resource, schedule, and technical aspects of the project are identified, assessed, and documented.
- 1 The risks are analyzed and prioritized based on their potential impact to the project.
  - 2 Contingencies (schedule buffers, alternate staffing plans, and alternate plans for additional computing equipment) for the risks are identified.
- Ac14* Plans for the project's software engineering facilities and support tools are prepared.
- 1 Estimates of capacity requirements for these facilities and support tools (host computers and peripherals for software development, software test computers and peripherals, target computer environment software, and other support software) are based on the size estimates of the software work products and other characteristics.
  - 2 Responsibilities are assigned and commitments are negotiated to procure or develop these facilities and support tools.
  - 3 All affected groups review the plans.
- Ac15* Software planning data are recorded.
- 1 Information recorded includes the estimates and the associated information needed to reconstruct the estimates and assess their reasonableness.
  - 2 The software planning data are managed and controlled.

### ***Measurement and analysis***

- Me1* Measurements are made and used to determine the status of the software planning activities. Examples of measurements include:
- 1 Completions of milestones for the software project planning activities compared to the plan
  - 2 Work completed, effort expended, and funds expended in the software project planning activities compared to the plan.

### ***Verifying implementation***

- Ve1* The activities for software project planning are reviewed with senior management on a periodic basis.
- 1 The technical, cost, staffing, and schedule performance is reviewed.
  - 2 Conflicts and issues not resolvable at lower levels are addressed.
  - 3 Software project risks are addressed.
  - 4 Action items are assigned, reviewed, and tracked to closure.

- 5 A summary report from each meeting is prepared and distributed to the affected groups and individuals.
- Ve2 The activities for software project planning are reviewed with the project manager on both a periodic and event-driven basis.
  - 1 Affected groups are represented.
  - 2 Status and current results of the software project planning activities are reviewed against the software project's statement of work and allocated requirements.
  - 3 Dependencies between groups are addressed.
  - 4 Conflicts and issues not resolvable at lower levels are addressed.
  - 5 Software project risks are reviewed.
  - 6 Action items are assigned, reviewed, and tracked to closure.
  - 7 A summary report from each meeting is prepared and distributed to the affected groups and individuals.
- Ve3 The software quality assurance group reviews and/or audits the activities and work products for software project planning and reports the results.
  - 1 The activities for software estimating and planning.
  - 2 The activities for reviewing and making project commitments.
  - 3 The activities for preparing the software development plan.
  - 4 The standards used for preparing the software development plan.
  - 5 The content of the software development plan.

## **Tarkvaraprojekti monitooring (PTO – Software Project Tracking and Oversight).**

### *Software Project Tracking and Oversight*

The purpose of Software Project Tracking and Oversight is to provide adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.

#### **Goals**

- G1 Actual results and performances are tracked against the software plans.
- G2 Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.
- G3 The activated groups and individuals agree to changes to software commitments.

#### **Commitment to perform**

- Co1 A project software manager is designated to be responsible for the project's software activities and results.
- Co2 The project follows a written organizational policy for managing the software project.
  - 1 A documented software development plan is used and maintained as the basis for tracking the software project.
  - 2 The project manager is kept informed of the software project's status and issues.
  - 3 Corrective actions are taken when the software plan is not being achieved, either by adjusting performance or by adjusting the plans.
  - 4 Changes to the software commitments are made with the involvement and agreement of the affected groups.
  - 5 Senior management reviews all commitment changes and new software project commitments made to individuals and groups external to the organization.

#### **Ability to perform**

- Ab1 A software development plan for the software project is documented and approved.
- Ab2 The project software manager explicitly assigns responsibility for software work products and activities. The assigned responsibilities cover:
  - 1 The software work products to be developed or services to be provided.
  - 2 The effort and cost for these software activities.

- 3 The schedule for these software activities.
- 4 The budget for these software activities.
- Ab3 Adequate resources and funding are provided for tracking the software project.
  - 1 The software managers and the software task leaders are assigned specific responsibilities for tracking the software project.
  - 2 Tools (spreadsheet programs, and project planning/scheduling programs) to support software tracking are made available.
- Ab4 Software managers are trained in managing the technical and personnel aspects of the software project.
- Ab5 First line managers receive orientation (the project's software engineering standards and procedures, and the project's application domain) in the technical aspects of the software project.

***Activities performed***

- Ac1 A documented software development plan is used for tracking the software activities and communicating status.
- Ac2 The project's software development plan is revised according to a documented procedure. This procedure typically specifies that:
  - 1 The software development plan is revised, as appropriate, to incorporate plan refinements and incorporate plan changes, particularly when plans change significantly.
  - 2 The software development plan is updated to incorporate all new software project commitments and changes to commitments.
  - 3 The software development plan is reviewed at each revision.
  - 4 The software development plan is managed and controlled.
- Ac3 Software project commitments and changes to commitments made to individuals and groups external to the organization are reviewed with senior management according to a documented procedure.
- Ac4 Approved changes to commitments that affect the software project are communicated to the members of the software engineering group and other software-related groups.
- Ac5 The size of the software work products (or size of the changes to the software work products) is tracked, and corrective actions are taken as necessary.
  - 1 Sizes for all major software work products (or the size of the changes) are tracked.
  - 2 Actual size of code (generated, fully tested, and delivered) is compared to the estimates documented in the software development plan.
  - 3 Actual units of delivered documentation are compared to the estimates documented in the software development plan.
  - 4 Overall projected size of the software work products (estimates combined with actual) is refined, monitored, and adjusted on a regular basis.
  - 5 Changes in size estimates of the software work products that affect software commitments are negotiated with the affected groups and are documented.
- Ac6 The project's software effort and costs are tracked, and corrective actions are taken as necessary.
  - 1 Actual expenditures of effort and costs over time and against work completed are compared to the estimates documented in the software development plan to identify potential overruns and under runs.
  - 2 Software costs are tracked and compared to the estimates documented in the software development plan.
  - 3 Effort and staffing are compared to the estimates documented in the software development plan.
  - 4 Changes in staffing and other software costs that affect software commitments are negotiated with the affected groups and are documented.
- Ac7 The project's critical computer resources are tracked, and corrective actions are taken as necessary.

- 1 The actual and projected use of the project's critical computer resources are tracked and compared to the estimates for each major software component as documented in the software development plan.
  - 2 Changes in estimates of critical computer resources that affect software commitments are negotiated with the affected groups and are documented.
- Ac8* The project's software schedule is tracked, and corrective actions are taken as necessary.
- 1 Actual completion of software activities, milestones, and other commitments is compared against the software development plan.
  - 2 Effects of late and early completion of software activities, milestones, and other commitments are evaluated for impacts on future activities and milestones.
  - 3 Software schedule revisions that affect software commitments are negotiated with the affected groups and are documented.
- Ac9* Software engineering technical activities are tracked, and corrective actions are taken as necessary.
- 1 Members of the software engineering group report their technical status to their first-line manager on a regular basis.
  - 2 Software release contents for successive builds are compared to the plans documented in the software development plan.
  - 3 Problems identified in any of the software work products are reported and documented.
  - 4 Problem reports are tracked to closure.
- Ac10* The software risks associated with cost, resource, schedule, and technical aspects of the project are tracked.
- 1 The priorities of the risks and the contingencies for the risks are adjusted as additional information becomes available.
  - 2 High-risk areas are reviewed with the project manager on a regular basis.
- Ac11* Actual measurement data and replanning data for the software project are recorded.
- 1 Information recorded includes the estimates and associated information needed to reconstruct the estimates and verify their reasonableness.
  - 2 The software replanning data are managed and controlled.
  - 3 The software planning data, replanning data, and the actual measurement data are archived for use by ongoing and future projects.
- Ac12* The software engineering group conducts periodic internal reviews to track technical progress, plans, performance, and issues against the software development plan. These reviews are conducted between:
- 1 The first-line software managers and their software task leaders.
  - 2 The project software manager, first-line software managers, and other software managers, as appropriate.
- Ac13* Formal reviews to address the accomplishments and results of the software project are conducted at selected project milestones according to a documented procedure. These reviews:
- 1 Are planned to occur at meaningful points in the software project's schedule, such as the beginning or completion of selected stages.
  - 2 Are conducted with the customer, end user, and affected groups within the organization, as appropriate.
  - 3 Use materials that are reviewed and approved by the responsible software managers.
  - 4 Address the commitments, plans, and status of the software activities.
  - 5 Result in the identification and documentation of significant issues, action items, and decisions.
  - 6 Address the software project risks.
  - 7 Result in the refinement of the software development plan, as necessary.

### ***Measurement and analysis***

**Me1** Measurements are made and used to determine the status of the software tracking and oversight activities.

### ***Verifying implementation***

**Ve1** The activities for software project tracking and oversight are reviewed with senior management on a periodic basis.

- 1 The technical, cost, staffing, and schedule performance are reviewed.
- 2 Conflicts and issues not resolvable at lower levels are addressed.
- 3 Software project risks are addressed.
- 4 Action items are assigned, reviewed, and tracked to closure.
- 5 A summary status report from each meeting is prepared and distributed to the affected groups.

**Ve2** The activities for software project tracking and oversight are reviewed with the project manager on both a periodic and event-driven basis.

- 1 Affected groups are represented.
- 2 The technical, cost, staffing, and schedule performance is reviewed against the software development plan.
- 3 Use of critical computer resources is reviewed; current estimates and actual use of these critical computer resources are reported against the original estimates.
- 4 Dependencies between groups are addressed.
- 5 Conflicts and issues not resolvable at lower levels are addressed.
- 6 Software project risks are addressed.
- 7 Action items are assigned, reviewed, and tracked to closure.
- 8 A summary report from each meeting is prepared and distributed to the affected groups.

**Ve3** The software quality assurance group reviews and/or audits the activities and work products for software project tracking and oversight and reports the results

- 1 The activities for reviewing and revising commitments.
- 2 The activities for revising the software development plan.
- 3 The content of the revised software development plan.
- 4 The activities for tracking the software project's cost, schedule, risks, technical and design constraints, and functionality and performance.
- 5 The activities for conducting the planned technical and management reviews.

## **Kvaliteedi tagamine (SQM – Software Quality Assurance).**

### ***Software Quality Assurance***

The purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and of the products being built.

**G1** Software quality assurance activities are planned.

**G2** Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively

**G3** Affected groups and individuals are informed of software quality assurance activities and results.

**G4** Senior management addresses non-compliance issues that cannot be resolved within the software project.

### ***Commitment to perform***

**Co1** The project follows a written organizational policy for implementing software quality assurance (SQA). This policy typically specifies that:

- 1 The SQA function is in place on all software projects.
- 2 The SQA group has a reporting channel to senior management that is independent of: the project manager; the project's software engineering group; the other software-related groups (configuration management, documentation support)

- 3 Senior management periodically reviews the SQA activities and results.

***Ability to perform***

*Ab1* A group that is responsible for coordinating and implementing SQA for the project exists.

*Ab2* Adequate resources and funding are provided for tracking the performing the SQA activities.

- 1 A manager is assigned specific responsibilities for the project's SQA activities.
- 2 A senior manager, who is knowledgeable in the SQA role and has the authority to take appropriate oversight actions, is designated to receive and act on software noncompliance items. (All managers in the SQA reporting chain to the senior manager are knowledgeable in the SQA role, responsibilities, and authority.)
- 3 Tools to support the SQA activities are made available.

*Ab3* Members of the SQA group are trained to perform their SQA activities. Examples of training include:

- 1 Software engineering skills and practices
- 2 Roles and responsibilities of the software engineering group and other software-related groups;
- 3 Standards, procedures, and methods for the software project;
- 4 Application domain of the software project
- 5 SQA activities, procedures, and methods
- 6 Involvement of the SQA group in the software activities
- 7 Effective use of SQA methods and tools
- 8 Interpersonal communications

***Activities performed***

*Ac1* A SQA plan is prepared for the software project according to a documented procedure. This procedure typically specifies that:

- 1 The SQA plan is developed in the early stages of, and in parallel with, the overall project planning.
- 2 The affected groups and individuals review the SQA plan. (Affected groups and individuals are: the project software manager, other software managers, the project manager, customer SQA representative, the senior manager to whom the SQA group reports noncompliance issues, software engineering group)

- 3 The SQA plan is managed and controlled.

*Ac2* The SQA group's activities are performed in accordance with the SQA plan.

- 1 Responsibilities and authority of the SQA group
- 2 Resource requirements for the SQA group (staff, tools, facilities)
- 3 Schedule and funding of the project's SQA group activities.
- 4 The SQA group's participation in establishing the software development plan, standards, and procedures for the project.
- 5 Evaluations to be performed by the SQA group
- 6 Audits and reviews to be conducted by the SQA group
- 7 Projects, standards and procedures to be used as the basis for the SQA group's reviews and audits.
- 8 Procedures for documenting and tracking noncompliance issues to closure.
- 9 Documentation that the SQA group required to produce.
- 10 Method and frequency of providing feedback to the software engineering group and other software-related groups on SQA activities.

*Ac3* The SQA group participates in the preparation and review of the project's software development plan, standards, and procedures.

- 1 The SQA group provides consultation and review of the plans, standards, and procedures with regards to organizational policy, externally imposed standards, standards that are appropriate for use by the project, topics that should be addressed in the software development plan.

- 2 The SQA group verifies that plan, standards, and procedures are in place and can be used to review and audit the software project.
  - Ac4 The SQA group reviews the software engineering activities to verify compliance.
  - Ac5 The SQA group audits designated software work products to verify compliance.
    - 1 The deliverable software products are evaluated before they are delivered to the customer.
    - 2 The software work products are evaluated against the designated software standards, procedures, and contractual requirements.
    - 3 Deviations are identified, documented, and tracked to closure.
    - 4 Corrections are verified.
  - Ac6 The SQA group periodically reports the results of its activities to the software engineering group.
  - Ac7 Deviations identified in the software activities and software work products are documented and handled according to a documented procedure.
    - 1 Deviations from the software development plan and the designated project standards and procedures are documented and resolved with the appropriate software task leaders, software managers, or project manager, where possible.
    - 2 Deviations from the software development plan and the designated project standards and procedures not resolvable with the software task leaders, software managers, or project manager are documented and presented to the senior manager designated to receive noncompliance items.
    - 3 Noncompliance items presented to the senior manager are periodically reviewed until they are resolved.
    - 4 The documentation of noncompliance items is managed and controlled.
  - Ac8 The SQA group conducts periodic reviews of its activities and findings with the customer's SQA personnel, as appropriate.
- Measurement and analysis*
- Me1 Measurements are made and used to determine the cost and schedule status of the SQA activities. Examples of measurements include:
- 1 Completions of milestones for the SQA activities compared to the plan;
  - 2 Work completed, effort expended, and funds expended in the SQA activities compared to the plan; and
  - 3 Numbers of product audits and activity reviews compared to the plan.
- Verifying implementation*
- Ve1 The SQA activities are reviewed with senior management on a periodic basis.
- Ve2 The SQA activities are reviewed with the project manager on both a periodic and event-driven basis.
- Ve3 Experts independent of the SQA group periodically review the activities and software work products of the project's SQA group.

## **Konfiguratsoonide haldamine (SCM – Software Configuration Management).**

### *Software Configuration Management*

The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

- G1 Software configuration management activities are planned.
- G2 Selected software work products are identified, controlled, and available.
- G3 Changes to identified software work products are controlled.
- G4 Affected groups and individuals are informed of the status and content of software baselines.

### *Commitment to perform*

- Co1 The project follows a written organizational policy for implementing software configuration management (SCM). This policy typically specifies that:



- 1 Responsibility for SCM for each project is explicitly assigned.
- 2 SCM is implemented throughout the project's life cycle.
- 3 SCM is implemented for externally deliverable software products, designated internal software work products, and designated support tools used inside the project (e.g., compilers).
- 4 The projects establish or have access to a repository for storing configuration items/units and the associated SCM records.
- 5 The software baselines and SCM activities are audited on a periodic basis.

***Ability to perform***

- Ab1* A board having the authority for managing the project's software baselines (i.e., a software configuration control board - SCCB) exists or is established. The SCCB:
- 1 Authorizes the establishment of software baselines and the identification of configuration items/units.
  - 2 Represents the interests of the project manager and all groups who may be affected by changes to the software baselines.
  - 3 Reviews and authorizes changes to the software baselines.
  - 4 Authorizes the creation of products from the software baseline library.
- Ab2* A group that is responsible for coordinating and implementing SCM for the project (i.e., the SCM group) exists. The SCM group coordinates or implements:
- 1 Creation and management of the project's software baseline library.
  - 2 Development, maintenance, and distribution of the SCM plans, standards, and procedures.
  - 3 The identification of the set of work products to be placed under SCM.
  - 4 Management of the access to the software baseline library.
  - 5 Updates of the software baselines.
  - 6 Creation of products from the software baseline library.
  - 7 Recording of SCM actions.
  - 8 Production and distribution of SCM reports.
- Ab3* Adequate resources and funding are provided for performing the SCM activities.
- 1 A manager is assigned specific responsibilities for SCM.
  - 2 Tools to support the SCM activities are made available.
- Ab4* Members of the SCM group are trained in the objectives, procedures, and methods for performing their SCM activities.
- Ab5* Members of the software engineering group and other software-related groups are trained to perform their SCM activities (standards, procedures, methods and tools)

***Activities performed***

- Ac1* A SCM plan is prepared for each software project according to a documented procedure. This procedure typically specifies that:
- 1 The SCM plan is developed in the early stages of, and in parallel with, the overall project planning.
  - 2 The affected groups review the SCM plan.
  - 3 The SCM plan is managed and controlled.
- Ac2* A documented and approved SCM plan is used as the basis for performing the SCM activities. The plan covers:
- 1 The SCM activities to be performed, the schedule of activities, the assigned responsibilities, and the resources required (including staff, tools, and computer facilities).
  - 2 The SCM requirements and activities to be performed by the software engineering group and other software-related groups.
- Ac3* A configuration management library system is established as a repository for the software baselines. This library system:
- 1 Supports multiple control levels of SCM.

- 2 Provides for the storage and retrieval of configuration items/units.
  - 3 Provides for the sharing and transfer of configuration items/units between the affected groups and between control levels within the library.
  - 4 Helps in the use of product standards for configuration items/units.
  - 5 Provides for the storage and recovery of archive versions of configuration items/units.
  - 6 Helps to ensure correct creation of products from the software baseline library.
  - 7 Provides for the storage, update, and retrieval of SCM records.
  - 8 Supports production of SCM reports.
  - 9 Provides for the maintenance of the library structure and contents.
- Ac4* The software work products to be placed under configuration management are identified.
- 1 The configuration items/units are selected based on documented
  - 2 The configuration items/units are assigned unique identifiers.
  - 3 The characteristics of each configuration item/unit are specified.
  - 4 The software baselines to which each configuration item/unit belongs are specified.
  - 5 The point in its development that each configuration item/unit is placed under configuration management is specified.
  - 6 The person responsible for each configuration item/unit (i.e., the owner, from a configuration management point of view) is identified.
- Ac5* Change requests and problem reports for all configuration items/units are initiated, recorded, reviewed, approved, and tracked according to a documented procedure.
- Ac6* Changes to baselines are controlled according to a documented procedure. This procedure typically specifies that:
- 1 Reviews and/or regression tests are performed to ensure that changes have not caused unintended effects on the baseline.
  - 2 Only configuration items/units that are approved by the SCCB are entered into the software baseline library.
  - 3 Configuration items/units are checked in and out in a manner that maintains the correctness and integrity of the software baseline library.
- Ac7* Products from the software baseline library are created and their release is controlled according to a documented procedure. This procedure typically specifies that:
- 1 The SCCB authorizes the creation of products from the software baseline library.
  - 2 Products from the software baseline library, for both internal and external use, are built only from configuration items/units in the software baseline library.
- Ac8* The status of configuration items/units is recorded according to a documented procedure. This procedure typically specifies that:
- 1 The configuration management actions are recorded in sufficient detail so that the content and status of each configuration item/unit are known and previous versions can be recovered
  - 2 The current status and history (i.e., changes and other actions) of each configuration item/unit are maintained.
- Ac9* Standard reports documenting the SCM activities and the contents of the software baseline are developed and made available to affected groups and individuals.
- Ac10* Software baseline audits are conducted according to a documented procedure. This procedure typically specifies that:
- 1 There is adequate preparation for the audit.
  - 2 The integrity of software baselines is assessed.
  - 3 The structure and facilities of the configuration management library system are reviewed.
  - 4 The completeness and correctness of the software baseline library contents are verified.
  - 5 Compliance with applicable SCM standards and procedures is verified.
  - 6 The results of the audit are reported to the project software manager.
  - 7 Action items from the audit are tracked to closure.

***Measurement and analysis***

*Me1* Measurements are made and used to determine the status of the SCM activities. Examples of measurements include:

- 1 Number of change requests processed per unit time;
- 2 Completions of milestones for the SCM activities compared to the plan; and
- 3 Work completed, effort expended, and funds expended in the SCM activities.

***Verifying implementation***

*Ve1* The SCM activities are reviewed with senior management on a periodic basis.

*Ve2* The SCM activities are reviewed with the project manager on both a periodic and event-driven basis.

*Ve3* The SCM group periodically audits software baselines to verify that they conform to the documentation that defines them.

*Ve4* The software quality assurance group reviews and/or audits the activities and work products for SCM and reports the results.

## **Allhangete juhtimine (SSM – Software Subcontract Management).**

### ***Software Subcontract Management***

The purpose of Software Subcontract Management is to select qualified software subcontractors and manage them effectively.

*G1* The prime contractor selects qualified software subcontractors.

*G2* The prime contractor and the software subcontractor agree to their commitments to each other.

*G3* The prime contractor and the software subcontractor maintain ongoing communications.

*G4* The prime contractor tracks the software subcontractor's actual results and performance against its commitments.

### ***Commitment to perform***

*Co1* The project follows a written organizational policy for managing the software subcontract. This policy typically specifies that.

- 1 Documented standards and procedures are used in selecting software subcontractors and managing the software subcontracts.
- 2 The contractual agreements form the basis for managing the subcontract.
- 3 Changes to the subcontract are made with the involvement and agreement of both the prime contractor and the subcontractor.

*Co2* A subcontract manager is designated to be responsible for establishing and managing the software subcontract.

- 1 The subcontract manager is knowledgeable and experienced in software engineering or has individuals assigned who have that knowledge and experience.
- 2 The subcontract manager is responsible for coordinating the technical scope of work to be subcontracted and the terms and conditions of the subcontract with the affected parties.
- 3 The subcontract manager is responsible for: selecting the software subcontractor, managing the software subcontract, and arranging for the post-subcontract support of the subcontracted products.

### ***Ability to perform***

*Ab1* Adequate resources and funding are provided for selecting the software subcontractor and managing the subcontract.

- 1 Software managers and other individuals are assigned specific responsibilities for managing the subcontract.
- 2 Tools (estimating models, spreadsheet programs, and project management and scheduling programs) to support managing the subcontract are made available.

*Ab2* Software managers and other individuals who are involved in establishing and managing the software subcontract are trained to perform these activities.

- 1 Examples of training include: preparing and planning for software subcontracting, evaluating a subcontract bidder's software process capability, evaluating a subcontract bidder's software estimates and plans, selecting a subcontractor, and managing a subcontract.
- Ab3 Software managers and other individuals who are involved in managing the software subcontract receive orientation in the technical aspects of the subcontract.
  - 1 Examples of orientation include: (application domain, software technologies being applied, software tools being used, methodologies being used, standards being used, and procedures being used).

**Activities performed**

- Ac1 The work to be subcontracted is defined and planned according to a documented procedure. This procedure typically specifies that.
  - 1 The software products and activities to be subcontracted are selected based on a balanced assessment of both technical and nontechnical characteristics of the project.
  - 2 The specification of the work to be subcontracted and the standards and procedures to be followed are derived from the project's: statement of work, system requirements allocated to software, software requirements, software development plan, and software standards and procedures.
  - 3 A subcontract statement of work is: prepared, reviewed, agreed to, and revised when necessary, and managed and controlled.
  - 4 A plan for selecting a subcontractor is prepared concurrent with the subcontract statement of work and is reviewed, as appropriate.
- Ac2 The software subcontractor is selected, based on an evaluation of the subcontract bidders' ability to perform the work, according to a documented procedure. This procedure covers the evaluation of:
  - 1 Proposals submitted for the planned subcontract.
  - 2 Prior performance records on similar work, if available.
  - 3 The geographic locations of the subcontract bidders' organizations relative to the prime contractor.
  - 4 Software engineering and software management capabilities.
  - 5 Staff available to perform the work.
  - 6 Prior experience in similar applications, including software expertise on the subcontractor's software management team.
  - 7 Available resources.
- Ac3 The contractual agreement between the prime contractor and the software subcontractor is used as the basis for managing the subcontract. The contractual agreement documents:
  - 1 The terms and conditions.
  - 2 The statement of work.
  - 3 The requirements for the products to be developed.
  - 4 The list of dependencies between the subcontractor and the prime contractor.
  - 5 The subcontracted products to be delivered to the prime contractor.
  - 6 The conditions under which revisions to products are to be submitted.
  - 7 The acceptance procedures and acceptance criteria to be used in evaluating the subcontracted products before the prime contractor accepts them.
  - 8 The procedures and evaluation criteria to be used by the prime contractor to monitor and evaluate the subcontractor's performance.
- Ac4 A documented subcontractor's software development plan is reviewed and approved by the prime contractor.
  - 1 This software development plan covers (directly or by reference) the appropriate items from the prime contractor's software development plan.
- Ac5 A documented and approved subcontractor's software development plan is used for tracking the software activities and communicating status.

- Ac6* Changes to the software subcontractor's statement of work, subcontract terms and conditions, and other commitments are resolved according to a documented procedure.
- 1 This procedure typically specifies that all affected groups of both the prime contractor and the subcontractor are involved.
- Ac7* The prime contractor's management conducts periodic status/coordination reviews with the software subcontractor's management.
- 1 The subcontractor is provided with visibility of the needs and desires of the product's customers and end users, as appropriate.
  - 2 The subcontractor's technical, cost, staffing, and schedule performance is reviewed against the subcontractor's software development plan.
  - 3 Computer resources designated as critical for the project are reviewed; the subcontractor's contribution to the current estimates are tracked and compared to the estimates for each software component as documented in the subcontractor's software development plan.
  - 4 Critical dependencies and commitments between the subcontractor's software engineering group and other subcontractor groups are addressed.
  - 5 Critical dependencies and commitments between the prime contractor and the subcontractor are addressed.
  - 6 Nonconformance to the subcontract is addressed.
  - 7 Project risks involving the subcontractor's work are addressed.
  - 8 Conflicts and issues not resolvable internally by the subcontractor are addressed.
  - 9 Action items are assigned, reviewed, and tracked to closure.
- Ac8* Periodic technical reviews and interchanges are held with the software subcontractor. Those reviews:
- 1 Provide the subcontractor with visibility of the customers and end users' needs and desires, as appropriate.
  - 2 Monitor the subcontractor's technical activities.
  - 3 Verify that the subcontractor's interpretation and implementation of the technical requirements conform to the prime contractor's requirements.
  - 4 Verify that commitments are being met.
  - 5 Verify that technical issues are resolved in a timely manner.
- Ac9* Formal reviews to address the subcontractor's software engineering accomplishments and results are conducted at selected milestones according to a documented procedure. This procedure typically specifies that:
- 1 Reviews are preplanned and documented in the statement of work.
  - 2 Reviews address the subcontractor's commitments for, plans for, and status of the software activities.
  - 3 Significant issues, action items, and decisions are identified and documented.
  - 4 Software risks are addressed.
  - 5 The subcontractor's software development plan is refined, as appropriate.
- Ac10* The prime contractor's software quality assurance group monitors the subcontractor's software quality assurance activities according to a documented procedure. This procedure typically specifies that:
- 1 The subcontractor's plans, resources, procedures, and standards for software quality assurance are periodically reviewed to ensure they are adequate to monitor the subcontractor's performance.
  - 2 Regular reviews of the subcontractor are conducted to ensure the approved procedures and standards are being followed.
  - 3 The subcontractor's records of its software quality assurance activities are periodically audited to assess how well the software quality assurance plans, standards, and procedures are being followed.

- Ac11* The prime contractor's software configuration management group monitors the subcontractor's activities for software configuration management according to a documented procedure. This procedure typically specifies that:
- 1 The subcontractor's plans, resources, procedures, and standards for software configuration management are reviewed to ensure they are adequate.
  - 2 The prime contractor and the subcontractor coordinate their activities on matters relating to software configuration management to ensure that the subcontractor's products can be readily integrated or incorporated into the project environment of the prime contractor.
  - 3 The subcontractor's software baseline library is periodically audited to assess how well the standards and procedures for software configuration management are being followed and how effective they are in managing the software baseline.
- Ac12* The prime contractor conducts acceptance testing as part of the delivery of the subcontractor's software products according to a documented procedure.
- 1 The acceptance procedures and acceptance criteria for each product are defined, reviewed, and approved by both the prime contractor and the subcontractor prior to the test.
  - 2 The results of the acceptance tests are documented.
  - 3 An action plan is established for any software product that does not pass its acceptance test.
- Ac13* The software subcontractor's performance is evaluated on a periodic basis, and the evaluation is reviewed with the subcontractor.

***Measurement and analysis***

- Me1* Measurements are made and used to determine the status of the activities for managing the software subcontract.
- 1 Examples of measurements include: costs of the activities for managing the subcontract compared to the plan, actual delivery dates for subcontracted products compared to the plan, and actual dates of prime contractor deliveries to the subcontractor compared to the plan.

***Verifying implementation***

- Ve1* The activities for managing the software subcontract are reviewed with senior management on a periodic basis.
- Ve2* The activities for managing the software subcontract are reviewed with the project manager on both a periodic and event-driven basis.
- Ve3* The software quality assurance group reviews and/or audits the activities and work products for managing the software subcontract and report the results.
- 1 The activities for selecting the subcontractor.
  - 2 The activities for managing the software subcontract.
  - 3 The activities for coordinating configuration management activities of the prime contractor and subcontractor.
  - 4 The conduct of planned reviews with the subcontractor.
  - 5 The conduct of reviews that establish completion of key project milestones or stages for the subcontract.
  - 6 The acceptance process for the subcontractor's software products.

## Lisa.2. XP-metoodika

XP-metoodika ülevaate kirjutamisel on lähtutud materjalidest ja loogikast, mis on leitavad aadressilt: <http://www.extremeprogramming.org/>

### Planeerimine

#### Soovilood ( *User story* )

- Kasutatakse põhjalike tehniliste tingimuste asemel.
- On kasutaja poolt kirjutatud kokkuvõtlikud jutud sellest, mida süsteem tegema peab.
- Kirjeldavad üldjuhul kasutaja vajadusi (mitte andmebaasi ja kasutajaliidese disaini).
- Peavad olema teineteisest võimalikult sõltumatud.
- On lühikesed (ideaalis 3 lauset) ja kirjutatud normaalses kõnekeeles.
- Sisaldavad parasjagu niipalju täpsust et planeerida ligikaudselt projektile kulutatavat aega ja projekti versioone (versiooni suurus on ligikaudu 80 lugu).
- Arendajad hindavad sooviloo realiseerimiseks ja täielikuks testimiseks vajalikku aega arenduse ideaalpäevades.
- Ideaalpäevad on sellised arendaja tööpäevad, kus kõik on selge ja saab kirjutada ainult koodi.
- Sooviloo realiseerimise ja testimise optimaalseks ajaks peetakse 3-9 arendaja ideaalpäeva.
- Ühete nädalasse mahub umbes kolm arendaja ideaalpäeva
- Tavaliste koodi testide kõrval kirjutatakse iga sooviloo kohta ka vähemalt üks selle realiseerimist kinnitav test.
- Kasutaja detailiseerib loo peensusteni alles loo realiseerimise ajal.

#### Väljalasete planeerimine

- Projekt kui tervik jaotatakse osadeks ehk väljalaseteks (*release*). Iga väljalase peab olema kasutaja seisukohalt reaalselt kasutatav reaalses töös.
- Väljalasete planeerimisel teevad äripoolle inimesed ainult äripoollega seotud otsuseid (mida on äri seisukohalt vaja) ja tehnika poole inimesed ainult tehnilise poole otsuseid (kui palju see ressursse võtab).
- Arendajad hindavad iga sooviloo realiseerimiseks ja testimiseks vajalikku aega arendaja ideaalpäevades.
- Kunagi ei ole reaalses nädalas viit kasutaja ideaalpäeva. Pigem on see kaks või kolm, kuna asjast tuleb ka aru saada.
- Seega kui soovilugu eeldab 5 ideaalpäeva, siis tähendab see seda, et reaalselt kulub ühel kasutajal selle tegemiseks peaaegu kaks nädalat.
- Kasutajad otsustavad milliseid soovilood realiseeritakse esimeses, teises ja milliseid viimases väljalaskes. Ärilises mõttes olulisemad alati enne.
- Planeerimise neljast muutujast (funktsionaalsus, aeg, maksumus, kvaliteet) saab äripool reaalselt fikseerida ainult ühte (kas funktsionaalsust või aega). Kui äripool fikseerib funktsionaalsuse, peab arendajatel olema võimalus määrata aeg ja vastupidi.
  - Funktsionaalsus – mida ja kui palju on vaja teha
  - Kvaliteet – kui hea peab süsteem olema ja kui hästi testitud. Väiksema kui tippkvaliteediga toodang ei rahulda ei klienti ega ka arendajat.
  - Raha – mitut inimest ja kui pikalt on võimalik kasutada. Ka siin puudub praktiliselt mängumaa, sest ühe arendaja ülalpidamiskulud on suhteliselt konstantsed.
  - Aeg – millal peab valmis olema
- Väljalasete planeerimisel tuleb jõuda osapoolte konsensuseni.

#### Väljalasked

- Esimese kasuliku ja töökõlbuliku väljalaske peaks kasutaja saama vähemalt kuuenda arenduskuu lõpul.
- Iga uus väljalase umbes iga kuue kuu tagant.
- Äriliselt kõige vajalikumad ja tehniliselt kõige raskemad soovilood tuleb planeerida võimalikult varajastesse väljalasetesse.

## Projekti iteratsioonid.

- Projekti versioonid realiseeritakse umbes 12-ne ühepikkuse iteratsiooni (perioodi) käigus
- Iteratsiooni sobilik pikkus on 1-3 nädalat. Iga iteratsiooni on sobilik alustada iteratsiooni planeerimisega, kus täpsustatakse ka kasutaja soove.
- Ühes iteratsioonis realiseeritakse üks või mõned kasutaja soovilood. soovilood realiseeritakse ülesannetena. Iga kasutaja lugu kirjutatakse lahti vajalikuks koguseks ülesanneteks. Ülesande sobilik pikkus on keskmiselt 1 - 3 arendaja ideaalpäev.
- Igas iteratsioonis tehakse just ja ainult selles iteratsioonis planeeritud asju.
- Iga arendaja valib endale tööd (ülesanded) ja koos sellega kohustub võetud ülesanded iteratsiooni käigus realiseerima ja testima.
- Iteratsiooni käigus tuleb jälgida ja fikseerida tööde kulgu. Kui palju iga arendaja on oma ülesannete peale ideaalpäevi kulutanud ja palju ta arvab veel kuluvat.
- Iteratsiooni plaanidest on kasulik kinni pidada. Kui on selgunud, et siiski on liiga palju või liiga vähe planeeritud, tuleb plaani kohe muuta (vähendada, suurendada ) ja sellest ka klienti teavitada.
- Ülesandeid täidetakse (realiseeritakse ja testitakse) ühekaupa alustades kasutajale kõige vajalikumast.
- Iteratsiooni planeerimisel võetakse aluseks töö hulk eelmises iteratsioonis ja planeeritakse käesolevasse sama palju.

## Projekti jõudlus

- Projekti jõudlus näitab meeskonna poolt tehtava töö hulka projekti ühes iteratsioonis.
- Ideaalne arenduspäev on päev ainult kodeerimisele ja testimisele (on selge mida teha, kuidas teha, milliste vahenditega teha ja kuidas neid vahendeid kasutada)
- Ühe arendaja jõudlus on tema poolt tehtavate ideaalpäevade arv ühes iteratsioonis. Mõni arendaja suudab rohkem, mõni vähem. Meeskonna liikmete ideaalpäevad ühes iteratsioonis kokku annavad projekti jõudluse.
- Kasutaja määrab iteratsioonis realiseeritavad kasutajalood vastavalt projekti jõudlusele. Ühte iteratsiooni planeeritakse ainult niipalju, kuipalju meeskond on võimeline.
- Projekti jõudluse järgi pole võimalik mõõta kahte arendajate gruppi ega ühe ja sama grupi kahte projekti. Tegemist on ainult projekti sisemise mõõduga selleks, et oleks võimalik tajuda kulgemist.
- Projekti jõudlus on kõikuv suurus.

## Iteratsioonide planeerimine

- Iga iteratsioon algab koosolekuga, kus tehakse iteratsiooni plaan.
- Iteratsiooni valitakse kasutaja lood vastavalt kasutaja poolt antud prioriteetidele ja arendajate poolt hinnatud raskusastmele (keerukus ja uudsus)
- Iteratsiooni planeerimise koosolekul fikseeritakse kasutaja poolt ka realiseerimist kinnitavate testide stsenaariumid.
- Iteratsiooni planeeritakse töid hinnanguliselt sama palju, kui palju jõuti eelmisel iteratsioonil teha.
- Kasutaja soovilood ja realiseerimist kinnitavate testide stsenaariumid sõnastatakse teineteisest sõltumatuteks (et erinevad arendajad saavad korraga realiseerida) ülesanneteks. Ülesande realiseerimiseks on ideaalis 1 kuni 3 ideaalpäeva. Lühemad kui ühe päevased ülesanded grupeeritakse, pikemad kui kolme päevased sõnastatakse alamülesanneteks.
- Erinevalt kasutaja soovilugudest, mis on kirjutatud tavalises inimkeeles, on ülesanded kirjutatud tehnilises keeles. Teineteist osaliselt või täielikult dubleerivad ülesanded liidetakse üheks.
- Ülesanded moodustavad iteratsiooni detailse plaani
- Arendaja, kes võtab endale kohustuse mingi ülesanne realiseerida, omab õigust ütelda, palju see temal aega võtab. Töö saab see, kelle arvates on realiseerimiseks sooritatav aeg lühim.
- Pole midagi hullu, kui peale iga kolme kuni viit iteratsiooni kasutaja kirjutab kasutajalugusid ümber (parandab, täiendab, loobub) või muudab versiooni plaani. Kõik on lubatud. Kasutaja määrab mida on vaja teha. Arendajad muudavad vastavalt sellele disaini ja koodi (ka juba olemasolevat).



## Inimeste ümberpaigutamine.

- Inimesed peavad tegelema erinevate töödega ja saama erinevaid kogemusi.
- Kui inimene, kes ainult üksinda tunneb mingit valdkonda, lahkub või selles valdkonnas on liiga palju teha, väheneb projekti jõudlus drastiliselt.
- Kellegi unikaalsed teadmised võivad olla projektile hukatuslikud.
- Paarisprogrammeerimine (õpilane-õpetaja) võimaldab inimesi koolitada, anda nendele erinevaid ja mitmekülgseid teadmisi vähendamata projekti jõudlust.
- Eesmärgiks on iga üksiku grupi liikme piisavad teadmised (selleks et muuta, parandada ja edasi arendada) ja oskused igast üksikust koodi osast.

## Koosolekud.

- Tüüpilise koosoleku eesmärk on olukorrast (probleemid, saavutused) teavitamine.
- Välkkoosolekud (seistes) iga päeva alguses on mõningates meeskondades traditsiooniks.

## Metoodika täiustamine.

- XP ei ole kivistunud reeglite hulk. XP on mõtteviis selleks, et oma tööd paremini teha. Kui ikkagi midagi ei toimi, siis pole seda vaja punnitada.
- XP ei ole lõppeesmärk, vaid on alguspunkt. Täiusta meeskonnaga oma metoodikat pidevalt, arutage, mis toimib ja mis mitte. Mis on kasulik ja mis ajaraiskamine.
- Ka siin kehtib paindmetodoloogia : inimesed ja töötav kood kõigepealt.

## Disainimine

### Lihtsus

- Lihtne disain nõuab alati vähem aega lõpetamiseks, kui keeruline.
- Alati tee lihtsaim võimalik asi, mis töötab ja kasutaja tingimusi rahuldab.
- Kui leiad kunagi hiljem mingi lihtsama lahenduse, siis muuda kohe disaini.
- Alati on odavam lihtsustada kohe, kui kunagi hiljem.
- Ära lisa funktsionaalsust enne, kuni see on hädavajalik
- Disain peab olema lihtne aga mitte “cool”.

### Rakenduse metafoor

- Rakenduse metafoor peab olema lühike, lihtne, kõigile (ka täiesti võõrale) üheselt arusaadav mõnesõnaline selgitus rakendusest ( näited: verelabori server, tootmisliini IS)
- Kasuta rakenduse metafoori selleks, et hoida meeskonda ühel lainel klassidele ja meetoditele nimetuste andmisel ( LaboriSeade, LaboriSeadmeProtokoll, ...)
- Objektidele ja meetoditele nimetuste andmisel on oluline osa disaini lihtsusel, isedokumenteeriva koodi kirjutamisel ja hilisemal koodist probleemideta arusaamisel.

### Lihtsad disainimise vahendid

- Rakenduse disainimisel kasuta lihtsaid vahendeid (tahvel, paber, pliiats on vahel täiesti piisavad).
- Keerulised vahendid (spetsiaalprogrammid) sunnivad tavaliselt peale asju mida tegelikult vaja pole.
- Rakenduse disainimisel kasuta lihtsaid tehnikaid (CRC – *Class, Responsibilities, and Collaboration* kaardid, UML mudelid, ...)
- Kõige parem disaini dokumenteerimise vahend on selge, lihtne, arusaadav ilma kommentaarideta kood.

### Prototüübid riskide vähendamiseks.

- Prototüübi selleks, et saada kogemusi ja kindlust nii kasutatava tehnika kui ka disaini sobivuse kohta.
- Prototüüp on väga lihtne programm selleks, et katsetada potentsiaalset lahendust

- Prototüübis keskendu ainult katsetatavale lahendusele. Kõik muu jäta kõrvale. Ära ürita prototüüpi hiljem oma rakenduses kasutada. Üldjuhul see ei kõlba.
- Kui süsteemi seisukohast on vaja midagi pikemalt katsetada, pane üks programmeerijate paar seda spetsiaalselt nädalaks või paariks tegema.

### Minimaalne funktsionaalsus.

- Keskendu ainult kliendi poolt nõutud funktsionaalsuste realiseerimisele.
- Ära lisa ühtegi funktsionaalsust, mida sa arvad et kunagi hiljem on see kasulik
- Keskmises rakenduses kasutatakse ainult 10% selles rakenduses olevatest võimalustest (seega 90% selle rakenduse arendamiseks kulutatud aeg on raisatud aeg)

### Ümberkirjutamine (*Refactoring*)

- Alati, kui võimalik ja näed, vähenda koodi pikkust, korista koodist ülearused osad, elimineeri funktsionaalsust, mida ei kasutata ja uuenda disaini – see ongi ümberkirjutamine e. *refactoring*.
- Ümberkirjutamine hoiab kokku hilisemat aega ja parandab koodi kvaliteeti.
- Ümberkirjutamine lihtsustab koodi.
- Tegele ümberkirjutamisega pidevalt projekti eluea jooksul.
- Kirjutatud testid toetavad ümberkirjutamist ja annavad kindluse, et midagi ei rikutud ümberkirjutamise käigus ära.

## Programmeerimine

### Kliendi pidev kohalolek.

- Reaalne klient (inimene, kes teab mida ja milleks on süsteemis tegelikult vaja) ja kes hiljem seda süsteemi ka kasutama hakkab, peab olema meeskonna liige selleks, et igas projekti faasis oleks võimalus kliendiga näost näkku suhelda ja küsida.
- Kliendi ülesanneteks projektis on soovilugude kirjutamine, realiseerimist kinnitavate testi stsenaariumite koostamine, soovilugude ja ülesannete prioriteetide hindamine, testimiseks vajalike andmete loomine, rakenduse katsetamine ja arendustegevusele hinnangu andmine kuni projekti peatamiseni välja.

### Kokkulepitud standardid.

- Kood peab olema kirjutatud ranges vastavuses kokkulepitud standarditega.
- Koodi järgi arendaja identifitseerimine peab olema võimalik.
- Standarditega vastavuses olev kood hoiab kokku meeskonna aega.
- Hea kood on lihtne, ilma trikkideta kood, millest iga arendaja (nii oma, kui võõras) probleemideta ka aastate pärast aru saab

### Alusta testi kirjutamisega.

- Enne kui lisada objekti mingi meetod või kirjutada mingi eraldiseisev funktsioon/protseduur, on mõttekas kirjutada testprogramm selle meetodi või funktsiooni/protseduuri testimiseks.
- Ainult erandjuhul võib koodi kirjutada enne, kui testi selle koodi testimiseks
- Test enne ja kood pärast (*Test Driven Development*) lihtsustab koodi, täpsustab enne koodi kirjutamist tegelikud vajadused, kõrvaldab arusaamatused, annab otsese tagasiside tulemustest.
- Kõigepealt kirjutatakse üks test selleks, et defineerida jooksva ülesande mingi aspekt. Seejärel kirjutatakse lihtsaim kood, mis seda testi rahuldab. Kirjutatakse järgmine test mingi järgmise aspekti jaoks jne. Kuni midagi enam testida ei ole.
- Niimoodi kirjutatud kood saab lihtne ning realiseerib ainult vajalikud funktsionaalsused.
- Teste on võimalik kasutada ka koodi kasutamise selgitamiseks teistele arendajatele.

### Paaris programmeerimine

- Paaris programmeerimine tähendab, et kaks arendajat istuvad ühe ja sama kuvari taga, neil on kahepeale üks klaviatuur, üks hiir ja nad töötavad ühe ja sama koodi kallal.

- Kasuta paaris programmeerimist, see vähendab vigu, hoiab koodi lihtsa ja võimaldab programmeerijatel õppida teineteise oskustest.
- Paaris programmeerimine hoiab aega kokku. Kaks programmeerijat ühe arvuti taga kirjutavad üheskoos samapalju, kuid võrratult parema kvaliteediga koodi kui nad suudaksid seda eraldi kirjutades.
- Paaris programmeerimisel on ühe programmeerija ülesanne realiseerida koodi (objekti meetodit) ja teise ülesanne mõelda strateegiliselt (mida see meetod objektile annab)
- Paaris programmeerimist on raske juurutada väljakujunenud harjumuste tõttu, kuid rakendades ja omandades kogemusi on väga efektiivne.

## Pidev integreerimine tervikuks

- XP eeldab reegleid süsteemi integreerimiseks tervikuks.
- Kui iga arendaja omab koopi ühisest koodist (versioonikontrolli programmid võimaldavad) ja kuskil serveris hoitakse jooksvat tervikut võib ikkagi tekkida probleeme süsteemi integreerimiseks.
- Probleemide vältimiseks peaks olema veel üks kliendarvuti, milles arendustegevust ei teostata ja mida kasutatakse ainult integreerimiseks. Seega on kindlustatud, et integreerimisega tegeleb ainult üks arendajate paar korraga.
- Arendajad lähevad integratsiooni-arvuti taha, lisavad enda koodi ja testivad kogu süsteemi. Kui kõik testid läbitakse jäetakse uus (testitud ja töötav) versioon integratsiooni-arvutisse. Kui teste ei läbita parandatakse koodi või taastatakse endine töötav versioon.
- Kasulik on integreerida pigem tihedamini, kui harvemini.

## Kood on kollektiivne omand

- Igal meeskonna liikmel on õigus teha parandusi ja muudatusi mistahes koodi osas küsimata selleks kelleltki luba.
- Kogu meeskond on vastutav süsteemi kui terviku osas. Süsteemil puudub peaarhitekt.
- Iga süsteemi lisatud meetod on testitud. Testida tuleb ka iga sisse viidav muudatus ja parandus.
- Ka peale muudatuste ja paranduste sisseviimist peab süsteem ikkagi läbima kõik testid 100%-selt.
- Koodi kollektiivne omand võimaldab projektil ellu jääda ka peale suvalise arendaja lahkumist.

## Optimeerimine lõpus.

- Kõigepealt pane asjad tööle, kontrolli, et asjad töötavad õigesti ja alles seejärel vaata, kas asja on vaja kiiremaks teha

## Ületundide vältimine.

- Ületunnid vähendavad meeskonna motivatsiooni ja jõudlust.
- Projekt, mille jaoks tuleb teha ületunde hilineb olenemata sellest, mida ette võetakse.
- Ressursside lisamine uute inimeste näol ainult aeglustab projekti kulgu.
- Ainus lahendus on projekti funktsionaalsuse vähendamine (kvaliteeti vähendada ja teste tegemata jätta ei tohi) või aja lisamine.

## Testimine.

Testimisel on XP-metoodikas rõhutatult eriline, oluline ja tähtis koht. Kui XP testimist arendusmeeskonnas juurutatud ei ole, siis ei saa ka rääkida XP kui tarkvara arendamise metoodika kasutamisest antud meeskonnas. XP rõhutab seda, et iga kliendile üleantud funktsioon peab olema 100%-liselt testitud. Kui funktsioon testitud ei ole, siis rangelt võttes ei ole ka seda funktsiooni.

XP testimise metoodika rõhutab, et enne, kui asuda mingi funktsiooni kirjutamise või muutmise juurde, tuleb kirjutada test selle funktsiooni kontrollimiseks.

## Ühiktest (*unit test*)

- Süsteemi igat üksikut elementi (meetodit) tuleb testida. Välja võib jätta ainult üksikud triviaalsed meetodid.
- Kõiki süsteemi elementide testimiseks kirjutatud teste peab saama käivitada automaatselt. Selleks on olemas vastavad raamprogrammid või tuleb sellised programmid ise kirjutada.
- Testid kirjutatakse enne, kui kirjutatakse meetod, mida selle testiga testitakse.
- Saabuv tähtaeg ei ole argument selleks, et teste mitte kirjutata.
- Kui meetodit testitud ei ole, siis seda meetodit ka ei ole.
- Testid kulutavad aega arendusel, kuid hoiavad võrratult rohkem aega kokku hilisemal vigade otsimisel.
- Testid võimaldavad koodi ühisomandust: s.t. mitmel inimesel ühiselt töötada ühe ja sama rakenduse (koodi) kallal
- Süsteem peab läbima kõik selle süsteemi osade testimiseks kirjutatud testid.
- Testid võimaldavad peale koodis muudatuste tegemist veenduda, et midagi ei rikutud ära.
- Testid võimaldavad lühikeste ajavahemike järel integreerida osasid tervikuks ja veenduda süsteemi kui terviku töötamises.
- On vähe tõenäoline, et test on vale ja kood õige; samuti on vähetõenäoline, et nii test kui ka testitav kood on mõlemad vigased ja seda ei märgata.

## Kui leitakse viga, kirjutatakse test

- Kui avastatakse mingi viga, kirjutatakse test selle vea edaspidiseks vältimiseks.
- Kui viga on soovilooloo loogikas, tuleb kirjutada selle vea edaspidiseks vältimiseks realiseerimist kinnitav test.

## Sobivustest (*functional test*)

- Kasutaja kirjutab iga sooviloo testimiseks stsenaariumi, millega annab mõõdupuu selle soovilooloo korrektse realiseerimise mõõtmiseks.
- Selle stsenaariumi järgi kirjutatakse vahetult enne loo realiseerimist üks või mitu selle loo korrektset realiseerimist kinnitavat testi.
- Soovilugu on realiseeritud siis, kui rakendus läbib kõik antud loo jaoks kirjutatud testid ning ka kõikide eelnevate lugude jaoks kirjutatud testid.
- Soovilugude realiseerimist kinnitavate testide (sobivustestide) järgi mõõdetakse projekti kulgu. Kui test kirjutatud ja soovilugu nii realiseeritud, et kõiki testid on läbitud, siis on linnuke kirjas. Enne ei ole.

## Lisa.3. Küsimustik

### Hea kolleeg.

Olen Tallinna Pedagoogikaülikooli IT juhtimise magistrant. Minu magistritöö teemaks on tarkvara arendusmetoodika valimine ja juurutamine tarkvara arendusfirmas. Seoses sellega olen huvitatud eesti tarkvaraarendajate arvamustest ja Eestis sellel alal juurdunud praktikast.

Olen äärmiselt tänulik, kui leiad võimaluse täita lisatud küsimustik. Küsimustiku täitmine võtab aega umbes 20 minutit.

Käesolevaga kinnitan, et saadud andmeid kasutan ainult magistritöö raames. Töö tulemuseks on üldistatud andmed, mida olen nõus kõikide ankeedi täitjatega jagama.

Küsimustik koosneb neljast osast. Esimese osa küsimused on seotud Sinu igapäevase tegevuse ja kogemustega. Teises osas on üldised tarkvara arendusprotsessiga seotud väited. Kolmanda osa küsimused on seotud tarkvara arendusprotsessi meetoditega. Neljanda osa küsimused on seotud Sinu isiklike tunnetega.

Loodan Sinu abile.

Gunnar Piho

TPÜ IT juhtimise magistrant.

### Erialase tegevuse ja kogemusega seotud küsimused.

#### 1. Kes sa oled?

☐ Tippjuht ☐ Projekti- või grupijuht ☐ Tarkvara arendaja ☐ Muu (Täpsusta palun)

#### 2. Mis on Sinu põhitegevus (ed)?

<input type="checkbox"/> Spetsifikatsioonide koostamine	<input type="checkbox"/> Analüüs ja disain
<input type="checkbox"/> Konfiguratsioonide haldamine	<input type="checkbox"/> Kodeerimine
<input type="checkbox"/> Testimine	<input type="checkbox"/> Arendusprotsessi täiustamine
<input type="checkbox"/> Integreerimine	<input type="checkbox"/> Kvaliteedi tagamine
<input type="checkbox"/> Juhtimine	<input type="checkbox"/> Muu (Täpsusta palun)

#### 3. Milliseks pead oma teadmisi tarkvara arendusprotsessi ja metoodikate alal?

☐ Suurepärased ☐ Head ☐ Keskmised ☐ Vähesed ☐ Olematud

#### 4. Kui suur on Sinu tööalane kogemus tarkvara arendamisel?

☐ kuni 2 aastat ☐ 2-5 aastat ☐ 6-10 aastat ☐ rohkem kui 10 aastat

#### 5. Milline on Sinu haridus?

☐ kõrgem IT ☐ kõrgem ☐ IT eri ☐ muu

#### 6. Mitu inimest on firmas, kus Sa töötad, seotud tarkvara arendamisega?

☐ kuni 2 ☐ 3-5 ☐ 6-12 ☐ 13-50 ☐ rohkem kui 50

#### 7. Mitu inimest on meeskonnas (keskmiselt meeskondades), kus Sa töötad, seotud tarkvara arendamisega?

☐ kuni 2 ☐ 3-5 ☐ 6-12 ☐ 13-50 ☐ rohkem kui 50

#### 8. Firmas, kus Sa töötad, arendatakse tarkvara ... ?

☐ peamiselt müügiks. ☐ peamiselt firmasiseseks kasutamiseks.

#### 9. Milline tarkvara arendusmetoodika on kasutusel firmas, kus Sa töötad?

10. Millist tarkvara arendusmetoodikat üritatakse firmas, kus Sa töötad, lähiajal juurutada?

## Tarkvara arendusprotsessiga seotud küsimused.

Alljärgnevalt on loetletud mõningad tarkvara arendusprotsessiga seotud tegevused. Hinda tegevusi lähtuvalt enda kogemustest ja vasta:

1. Kas tegevus on projektile edu tagamiseks O - oluline, E - ebaoluline või P - pidurdav?
2. Kas firmas, kus Sa töötad, on antud tegevus tarkvara arendusprojektides T – tavaline, H – harvaesinev või M – mitte kasutatav.

Eduks ...				Esineb...		
O	E	P		T	H	M
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projektid viiakse läbi vastavalt protseduurireeglitele, mis on kõikidele asjaosalistele teada ning selleks on eraldatud adekvaatsed ressursid.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Inimesed on koolitatud selleks, et projekti on võimalik vastavalt kehtestatud reeglitele ja korrale läbi viia.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projektile on kinnitatud otsustamise õigusega ja personaalse vastutusega juht.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projekti käigus mõõdetakse projektiga seonduvaid tegevusi vastavalt defineeritud meetrikatele selleks, et teha projektist ja selle kulgemisest perioodiliselt adekvaatseid kokkuvõtteid ning planeerida järgnevaid projekte.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Perioodiliselt auditeeritakse (vajadusel muudetakse) projekti läbiviimiseks kehtestatud reegleid ja arendustegevuses kasutatavaid meetodeid.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kasutaja poolt esitatud nõudeid, muudatusettepanekuid nõuetesse ning nõuete muudatusi hallatakse. Kõiki asjaosalisi hoitakse nõuete ja nende muudatustega kursis.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nõudeid analüüsitakse ja kasutatakse tarkvara kavandamisel ning plaanide koostamisel.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Muudatused nõuetes on aluseks plaanide korrigeerimisele.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Planeerimise käigus jaotatakse arendatava tarkvara elutsükkel väiksemateks hallatavateks osadeks.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Arendajad osalevad projekti ettevalmistavas töös, planeerimisel ja aruteludes kogu projekti elutsükli jooksul.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Konkreetselt tarkvara arendamise plaan on kõikidele asjaosalistele kättesaadav kirjalik dokument.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Plaanides on kajastatud arendustegevusega seotud tööde loetelud, arendamist toetavate tegevuste loetelud, nende mahud ja vastutajad.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Plaanides on kajastatud plaani täitmise kontrollimiseks vajalikud protseduurid, tegevused ja kriteeriumid hinnangute andmiseks.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tarkvara arendamisega seotud osapooled (ka arendajad) on teadlikud ja nõus tähtaegade ning vastutusega.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Planeerimise käigus koostatakse riskide loetelu ja hinnatakse riskide tõenäosust. Riske jälgitakse kogu arendusprotsessi käigus.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Arendusplaani (tähtajad, vastutus,...) muutmiseks on kehtestatud reeglid. Arendusplaani muudatustest antakse kõikidele osapooltele teada.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Selleks, et rakendada adekvaatseid meetmeid juhul, kui plaanid ja tegelikkus teineteisest oluliselt erinevad, on projekti jälgimiseks kehtestatud kõikidele osapooltele teada reeglid.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tarkvara kvaliteedi tagamiseks on kehtestatud nõuded. Kvaliteedi tagamiseks vajalikud tegevused on reglementeeritud, planeeritud ja kõikidele osapooltele teada.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kvaliteedi tagamise eest vastutajad osalevad nii tarkvara planeerimise kui ka arendustegevusega seotud reeglite	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

			väljatöötamise ja läbivaatamise protsessis.			
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kvaliteedi eest vastutajad auditeerivad loodavat tarkvara ning vaatavad perioodiliselt läbi arendusega seotud tegevusi selleks, et kindlustada vastavus nõuetele. Tulemustest antakse kõikidele osapooltele teada.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sõltumatud eksperdid hindavad perioodiliselt kvaliteedi alast tegevust firmas.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projektil on olemas ja kõikidele osapooltele teada versioonide valmimise, levitamise ja haldamise plaan.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Versioonide ja konfiguratsioonide kirjeldused dokumenteeritakse ja säilitatakse. Muudatustest antakse osapooltele teada.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Perioodiliselt viiakse läbi versioonide ja konfiguratsioonide auditeid selleks, et kontrollida tegelikkuse vastavust dokumentidele.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Tarkvara arendusprotsessi meetoditega seotud küsimused.

Alljärgnevalt on loetletud mõningad tarkvara arendusprotsessi meetoditega seotud tegevused. Hinda tegevusi lähtuvalt enda kogemustest ja vasta:

1. Kas meetod on projektile edu tagamiseks O - oluline, E - ebaoluline või P - pidurdav?
2. Firmas, kus Sa töötad, on antud meetod tarkvara arendusprojektides T – tavaline, H – harvaesinev või M – mitte kasutatav.

Eduks ...				Esineb...		
O	E	P		T	H	M
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Reeglina on arendajate töökoormus 40 tundi nädalas. Ületunde ei tehta kunagi kaks nädalat järjest.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projektil on olemas selgesõnaline ja kõikidele mõistetav lühikokkuvõte.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Arendatava tarkvara tulevane (tegelik, potentsiaalne) kasutaja (näiteks elukutseline raamatupidaja, kui arendatakse raamatupidamise programmi) on arendusmeeskonna liige ja täidab "intelligentse spetsifikatsiooni" rolli.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tarkvarale esitatavad nõuded on kirja pandud soovilugudena. Ideaalne soovilugu on kolm lauset pikk, ei sisalda tehnilisi termineid, on kirjutatud kasutaja poolt, kirjeldab süsteemi ühte konkreetset tegevust ning on realiseeritav ja testitav kolme päeva jooksul.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Meeskonna liikmed järgivad koodi kirjutamisel ühtseid reegleid selliselt, et koodi lugedes pole võimalik kindlaks teha, kes programmeerijatest on selle kirjutanud.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kood on meeskonna ühine omand. Jälgitakse reeglit, et ühegi arendaja lahkumine ei halvaks projekti käekäiku. Igal arendajal peab olema piisav ülevaade koodi igast osast selleks, et vajadusel seda parandada, täiendada või muuta. Selle saavutamiseks "liiguvad arendajad projektis pidevalt ringi".	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Programmeerijad töötavad paaridena s.t kaks programmeerijat istuvad kõrvuti ühe arvuti taga. Neil on kahepeale üks monitor, üks hiir ja üks klaviatuur. Nad töötavad ühe ja sama probleemi (koodi osa) kallal.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tarkvara arendusühikuks on versioon. Esimene (minimaalse kuid kasutajale hädavajaliku funktsionaalsusega testitud ning müügikõlblik) versioon tehakse valmis lühikese aja (umbes 6 kuud kuni 1 aasta) jooksul. Iga järgnev versioon lisab eelnevale versioonile kasutaja seisukohalt uut tarvilikku funktsionaalsust.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Enne iga funktsiooni (või objekti meetodi) realiseerimist kirjutatakse test. Test kontrollib vastavust esitatud nõuetele. Projekti kõik testid koondatakse ja neid on võimalik ühiselt käivitada. Seega allub kogu kood automaatsele kontrollile.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Rakendust integreeritakse tervikuks tihti. Integreerimist on võimalik teostada ainult selleks eraldatud ühelt tööarvutilt (füüsiliselt kindlas järjekorras). Integreerimise juures lisatakse kõigepealt testid ja seejärel ülejäänud kood. Integreerimise tingimuseks on see, et integreeritud rakendus läbib kõik testid täielikult.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Rakenduse disain hoitakse (mõtlemata tulevikule) nii lihtne, kui antud funktsionaalsuse juures vähegi võimalik. Uut funktsionaalsust lisatakse rakendusse pigem hiljem, kui varem.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kui rakenduses leitakse liigset keerukust, dubleerimist või parem lahendus, muudetakse koodi. Kirjutatud testid tagavad selle, et muudatuse käigus midagi hullemaks ei lähe.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Projekti analüüsimisel, disainimisel, planeerimisel, jne... eelistatakse lihtsaid vahendeid: paber, pliiats, tahvel.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Koodi optimeerimisega (jõudluse probleemidega) tegeletakse peale vajaliku funktsionaalsuse realiseerimist.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Meeskonda valitakse andekad ja oma töösse pühendunud inimesed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Meeskonna liikmed teevad pidevalt kangelastegusid ja ületavad ennast selleks, et täita neile pandud kohustusi.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Lisaküsimused.

Kent Beck väidab oma raamatus “Extreme Programming Explained, Embrace Change” (Addison-Wesley, 2000), et tarkvara arendusmetoodikad püüavad taltsutada arendustegevusega seotud hirme selleks, et võimaldada tarkvara arendajatele rahulik uni. Alljärgnevalt on loetletud väidetavalt tüüpilised tarkvara arendajate hirmud. Palun hinda nende väidete paikapidavust enda puhul.

Kui tihti Sa oled tundnud, et ...

	Tihti	Harva	Mitte kunagi
... pead nõustuma tähtaegadega, mis pole reaalsed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... arendatavat rakendust pole tegelikult mitte kellelegi vaja.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Sa pole tarkvara arendamiseks piisavalt tark.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Sul pole tarkvara arendamiseks piisavaid teadmisi.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... oled pandud vastutama lootusetute olukordade eest.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Sa ei saa lõppkasutaja tegelikest vajadustest aru.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... pead ohverdama kvaliteedi tähtaegade kasuks.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... keeruliste probleemide korral ei ole abi kuskilt loota.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... edu saavutamiseks ei ole piisavalt aega.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

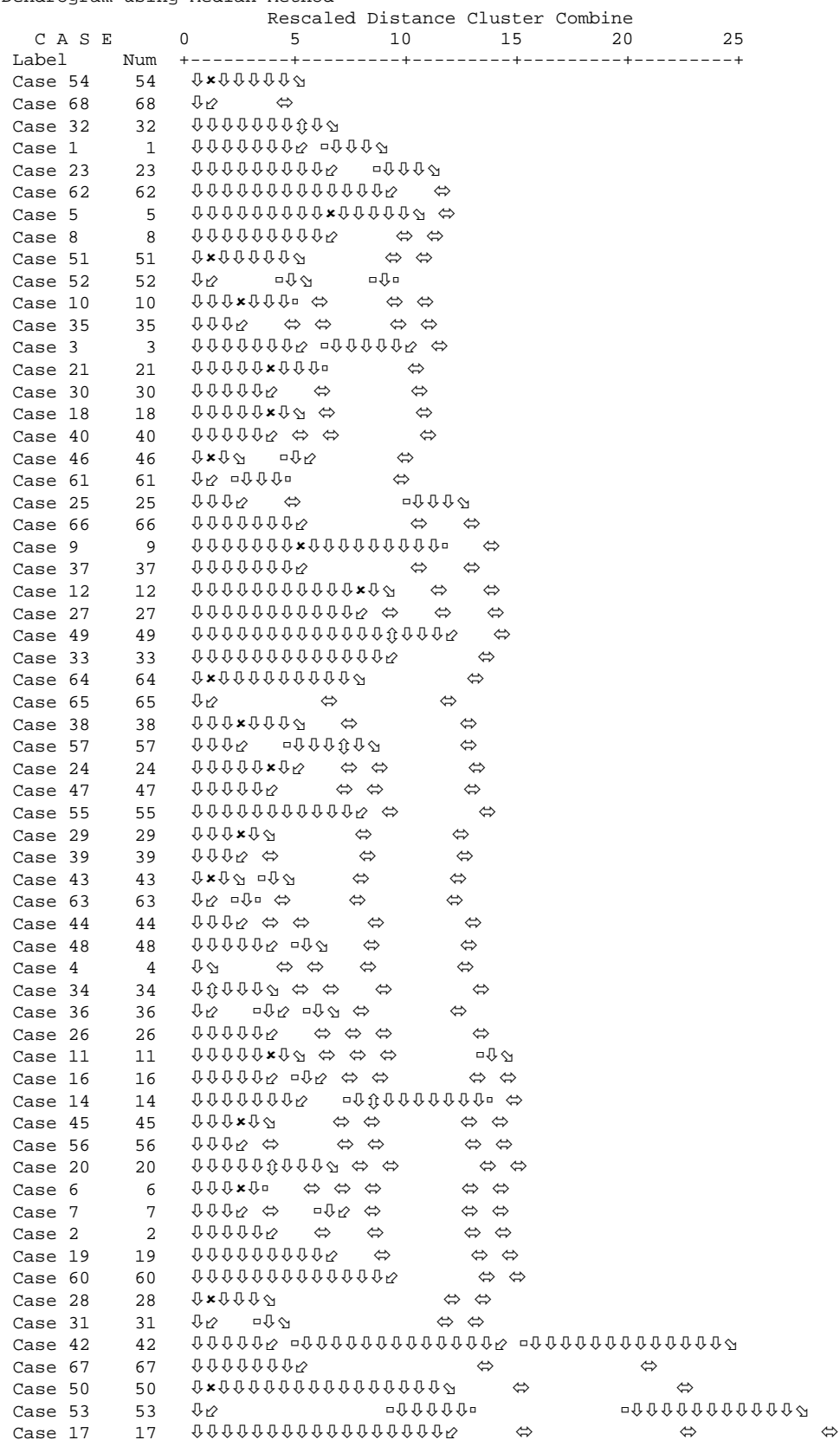


## Lisa.4. Andmetöötlus

### Vastanute rühmitamine taustandmete järgi

\* \* \* \* \* H I E R A R C H I C A L C L U S T E R A N A L Y S I S \* \* \* \* \*

Dendrogram using Median Method



Case 58	58			
Case 41	41			
Case 59	59			
Case 15	15			
Case 13	13			
Case 22	22			
Case 69	69			

```

* * * * * H I E R A R C H I C A L   C L U S T E R   A N A L Y S I S * * * * *

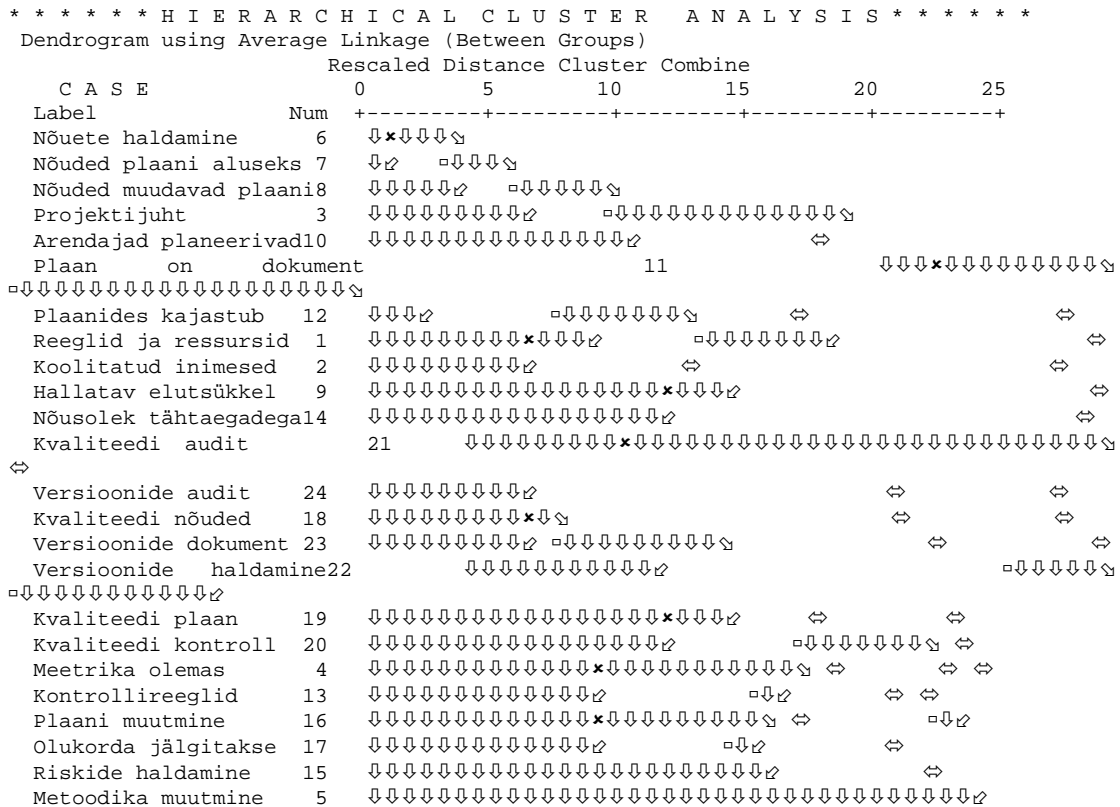
```

### Dendrogram using Average Linkage (Between Groups)

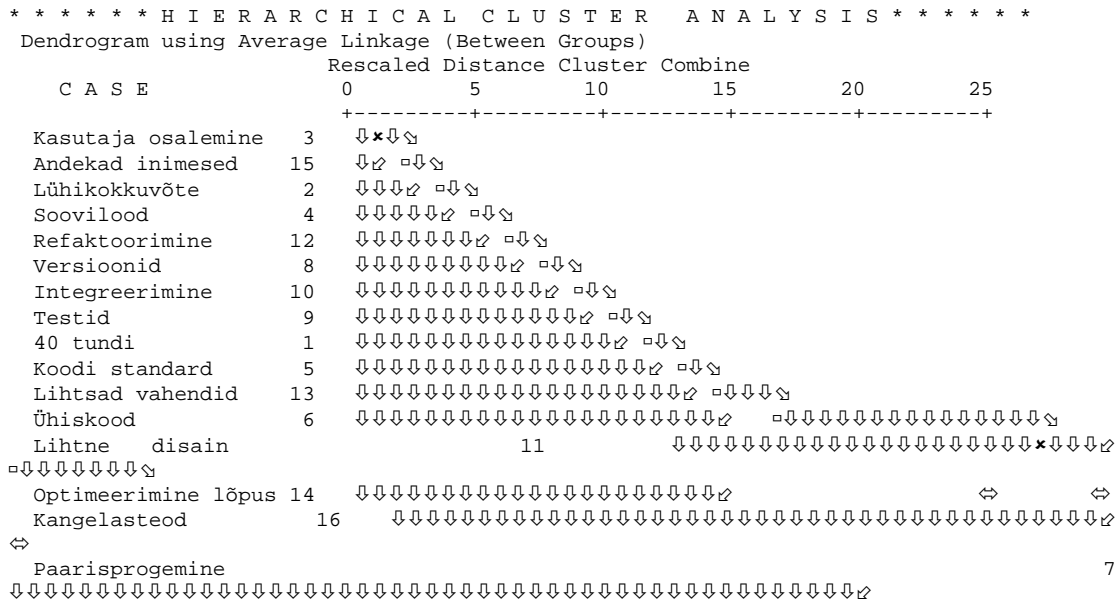
## Rescaled Distance Cluster Combine

C A S E	0	5	10	15	20	25
Label	Num					
Tähtaegadega nõusolek	14					
Kvaliteedi nõuded	18					
Ressursid olemas	1					
Plaan on dokument	11					
Plaanides sisu	12					
Nõuete haldamine	6					
Plaan lähtub nõuetest	7					
Projektijuht	3					
Versioonide dokument	23					
Inimesed koolitatud	2					
Kvaliteedi plaan	19					
Arendajad planeerivad	10					
Versioonide haldamine	22					
Elutsükli hallatakse	9					
Nõuded muudavad plaani	8					
Meetrika on olemas	4					
Olukorda jälgitakse	17					
Riske hallatakse	15					
Kontrollireeglid	13					
Kontrollitakse	20					
Plaan muutmisreeglid	16					
Metoodikat muudetakse	5					
Versioonide audit						
Kvaliteedi audit						

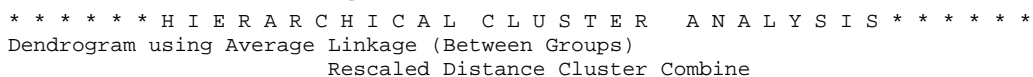
## Küsimuste “arendusprotsessis kasutatakse” rühmitamine

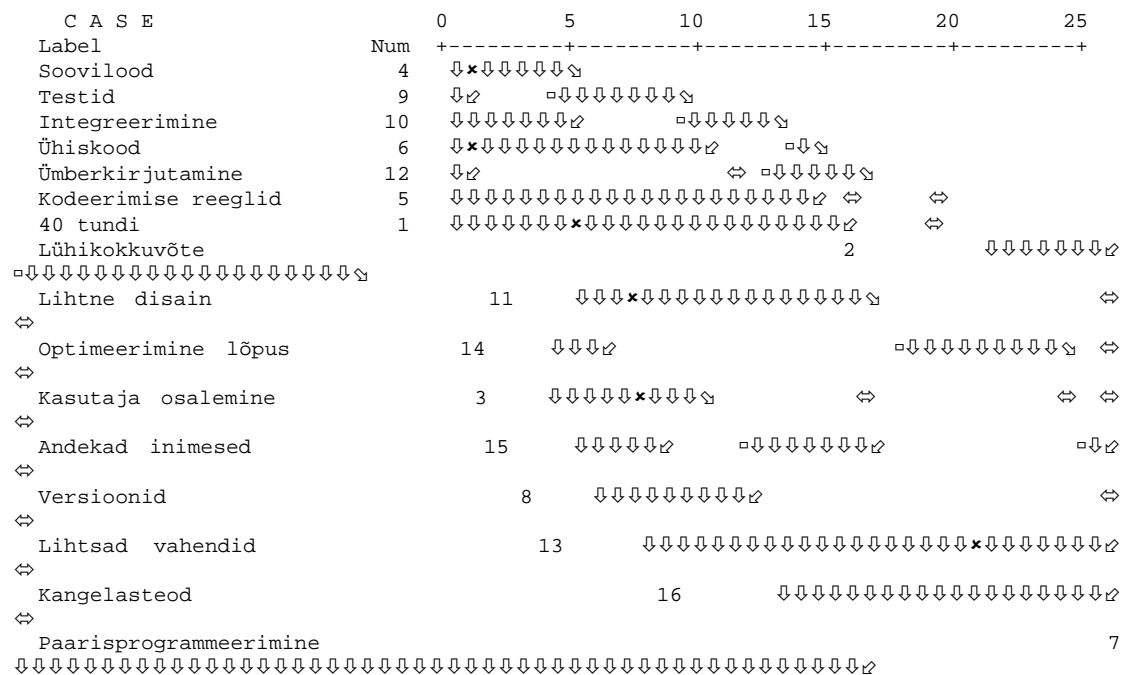


## Küsimuste “oluline arendusmeetod” rühmitamine

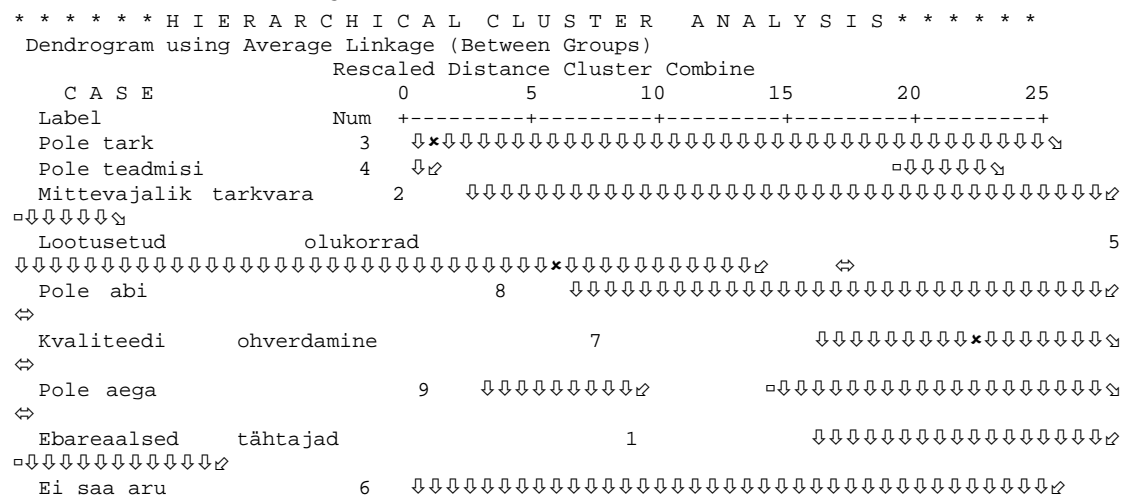


## Küsimuste “arendusprotsessis kasutatakse” rühmitamine





## Küsimuste “arendajate hirmud” rühmitamine



## Vastanute taustatunnused gruppides

### Grupp \* Amet Crosstabulation

		Amet			Total
			Arendaja	Projektijuht	Tippjuht
Grupp	juht		7	14	2
	arendaja	1	21	6	0
	teised	4	4	3	3
Total		9	32	23	5

### Grupp \* Teadmised metoodikatest Crosstabulation

		Teadmised metoodikatest				Total
		Vähesed	Keskmised	Head	Suurepärased	
Grupp	juht	1	4		3	27
	arendaja	8	17	3	0	28
	teised	4	3	6	1	14
Total		13	24	28	4	69

### Grupp \* Arendajaid firmas Crosstabulation

		Arendajaid firmas					Total
			3 kuni 5	6 kuni 12	13 kuni 50	50	
Grupp	juht	1	2	4	11	9	27
	arendaja	0	8		4	8	28
	teised	3	6	0	1	4	
Total		4		12	16	21	

### Grupp \* Haridus Crosstabulation

		Haridus				Total
		Kõrgem IT	Kõrgem	IT eri	Muu	
Grupp	juht		10	2	2	
	arendaja		12	6	8	28
	teised	5	7	1	1	14
Total		20	29	9	11	69

### Grupp \* Kogemus kokku Crosstabulation

		Kogemus kokku				Total
		Kuni 2 aastat	2 kuni 5 aastat	6 kuni 10 aastat	Rohkem kui 10 aastat	
Grupp	juht	0	6	8	13	27
	arendaja	3		6	0	28
	teised	0	6	2	6	14
Total			31	16		69

### Grupp \* Arendajaid grupis Crosstabulation

		Arendajaid grupis					Total
		kuni 2	3 kuni 5	6 kuni 12		rohkem kui 50	
Grupp	juht	2	18		1	0	27
	arendaja	3	17	8	0	0	28
	teised		3	0	1		14
Total			38	14	2	2	69

### Grupp \* Milleks tarkvara luuakse Crosstabulation

		Milleks tarkvara luuakse		Total
		Müügiks	Firmasiseseks	
Grupp	juht		12	27
	arendaja	28	0	28
	teised	6	8	14
Total		49	20	69

### Grupp \* Milline metoodika on kasutusel Crosstabulation

		Milline metoodika on kasutusel					Total
		Muu	UML	XP	RUP	ISO	
Grupp	juht	17	0		9	1	27
	arendaja	24		0		0	28
	teised	9	0		2	0	14
Total		50	1	3	14	1	69

### Grupp \* Milline metoodika on plaanis Crosstabulation

		Milline metoodika on plaanis				
		MUU		RUP	ISO	
	juht	20		7	0	27
	arendaja	21	4	2	1	28
	teised	11	2	1	0	14
Total		52	6	10	1	69

### Rühmitatud tunnuste keskmised vastajate ameti järgi

	Amet			
	Muu	Arendaja		Tippjuht
	Mean		Mean	Mean
Nõuete haldamine	,71	,63	,74	
Projekti planeerimine	-,07		,56	,50
Tulemuste jälgimine	-,13	-,13	,23	,16
Projekti monitooring	-,31	-,24	,09	,08
Tulemuste audit	-,83	-,73	-,30	-,50
Metoodika läbivaatamine	-,33	-,19	,48	,60
XP meetodid	-,11	-,20	-,03	,04
Tavalised meetodid	,24	,30	,41	,84
Häkkeri meetodid	,33	,27	,02	,40
Arukad meetodid	,06	,17	,43	-,10
Koodi standard	,22	-,06	,17	,60
Rumal olemise hirm	-,17	-,17	-,11	,00
Aja nappuse hirm	,59	,33	,38	,60
Abi mittesaamise hirm	,17	-,22	-,13	-,20
Mõttetüü töö hirm	-,44	-,16		,20
Kasutaja nõuete mittemõistmise hirm	-,11	,25	,04	-,40

## Rühmitatud tunnuste keskmised metoodikateadmiste järgi

	Teadmised metoodikatest			
	Vähesed	Keskmiised	Head	Suurepärased
	Mean	Mean	Mean	Mean
Nõuete haldamine	,71	,62	,74	,90
Projekti planeerimine		,26	,47	,71
	,02	-,09		,45
Projekti monitooring		-,25	-,01	-,10
	-,77	-,52	-,57	-,50
Metoodika läbivaatamine		,17	,18	,25
	-,11	-,23		,00
Tavalised meetodid	,40	,29	,40	,50
Häkkeri meetodid	,23	,29	,14	,00
Arukad meetodid	,35	,27		,63
Koodi standard	,15	-,29	,43	,00
	,15	-,08	-,27	-,50
Aja nappuse hirm	,15	,47	,44	,50
Abi mittesaamise hirm		-,04	-,20	-,25
Mõttetü töö hirm	-,54		-,21	-,75
Kasutaja nõuete mittemõistmise hirm	,00	,17	,11	-,25

## Rühmitatud tunnuste keskmised vastanute hariduse järgi

	Haridus			
	Kõrgem IT	Kõrgem	IT eri	
	Mean	Mean	Mean	Mean
Nõuete haldamine	,87	,73	,38	,58
Projekti planeerimine	,53	,42	-,02	,17
Tulemuste jälgimine	,24	-,02	-,18	
Projekti monitooring	,12		-,42	-,20
Tulemuste audit	-,32	-,72	-,78	
Metoodika läbivaatamine	,20	-,03	,00	,18
XP meetodid	,08	-,07	-,33	
Tavalised meetodid	,52	,36	,33	,15
Häkkeri meetodid	,18	,10	,33	,41
Arukad meetodid		,17	,06	-,05
	,45	,07		-,27
Rumal olemise hirm	-,13	-,29		,09
Aja nappuse hirm	,38	,28	,74	,48
Abi mittesaamise hirm	-,18	-,21	,17	-,14
Mõttetü töö hirm	-,50	-,24	,00	
Kasutaja nõuete mittemõistmise hirm	-,15	,17	,56	-,09

## Rühmitatud tunnuste keskmised vastanute kogemuse järgi

	Kogemus kokku			
	Kuni 2 aastat	2 kuni 5 aastat	6 kuni 10 aastat	Rohkem kui 10 aastat
	Mean	Mean	Mean	Mean
Nõuete haldamine	,87	,63	,68	,81
Projekti planeerimine	-,50	,41	,42	,35
Tulemuste jälgimine	-,67	-,06	,08	,18
Projekti monitooring	-,60	-,21	,00	,01
Tulemuste audit	-1,00	-,56	-,75	-,42
Metoodika läbivaatamine	-,67	,00	,19	,21
XP meetodid	-,47	-,21	-,11	,09
Tavalised meetodid	-,47	,39	,30	,52
Häkkeri meetodid	,33	,19	,34	,08
Arukad meetodid	,17	,24	,25	,18
Koodi standard	-1,00	,10	-,06	,42
Rumal olemise hirm	,17	-,13	-,09	-,24
Aja nappuse hirm	,22	,38	,46	,42
Abi mittedaamise hirm	-,67	-,16	-,06	-,08
Mõttetüü töö hirm	,67	-,35	-,25	-,37
Kasutaja nõuete mittemõistmise hirm	,67	,23	,06	-,21

### Arendusprotsessiga seotud rühmitatud tunnuste vahelised Pearsoni korrelatsioonikordajad

Case	Matrix File Input					
	1	2	3	4		6
Nõuete haldamine (1)	,000	,377	,361		,198	,155
Projekti planeerimine (2)		,000	,645		,476	
Tulemuste jälgimine (3)	,361		,000	,622	,571	
Projekti monitooring (4)	,355	,606	,622	,000	,463	,412
Tulemuste audit (5)	,198	,476	,571	,463	,000	,293
	,155	,418	,270	,412		,000

[illegible]



### Arendusmeetoditega seotud rühmitatud tunnuste vahelised Pearsoni korrelatsioonikordajad

Case	Matrix File Input					
	1	2	3	4	5	
XP meetodid (1)	,000	,410		,398	,390	,130
Tavalised meetodid (2)		,000	,033	,187	,321	-,025
Häkkeri meetodid (3)	,013	,033	,000	-,214	,002	,070
Arukad meetodid (4)	,398	,187		,000	,207	,191
Koodi standard (5)		,321	,002	,207	,000	,221
Paaris program. (6)	,130	-,025	,070	,191		,000

XP	1	↓↘
Tavalised	2	↓↑↓↘
Arukad	4	↓↗ □↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↘
Kood		5
Paaris	6	↖ ↓ ↗
Häkk		3

## Hirmudega seotud rühmitatud tunnuste vahelised Pearsoni korrelatsioonikordajad

Case	Matrix File Input				
	1	2		4	5
Rumal olemise hirm (1)	,000	-,009	,198	,110	,046
Aja nappuse hirm (2)	-,009	,000	,410	,119	,176
Abi mittesaamise hirm (3)	,198	,410	,000	-,008	,213
	,110	,119		,000	,180
Kasutaja nõuete mittemõistmise hirm (5)		,176	,213	,180	,000

Aja	2
Abi	3
Nõuete	5
Rumaluse	1
Mõttetuse	4

## Arendusprotsessiga seotud tunnuste vahelised Pearsoni korrelatsioonikordajad

	T1	T2	PKK		T1	T2	PKK
1		12	,742	13	6	7	,543
2	18	23	,688	14	1	15	,504
3	16	17	,673	15	5	9	,501
4	18	22	,655	16	1	16	,498
5	2	11	,629	17	1	14	,495
6	1	2	,595	18	1	24	,477
7	1	4	,575	19	1	21	,476
8	1	18	,573	20	1	5	,472
9	1	13	,563	21	1	6	,457
10	19	20	,560	22	1	3	,417
11	7	8	,557	23	1	10	,263
12	1	19	,556				

Arendusplaan olemas	11	↓×↓↓↓↓↓↓↓↓↓↓	
Plaanides kajastub	12	↓↗	□↓↓↓↓
Inimesed koolitatud	2	↓↓↓↓↓↓↓↓↓↓↓↓	□↓↗
Ressursid olemas	1	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Meetrika olemas	4	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↗↓↗
Kvaliteedinõuded	18	↓↓↓↓↓↓×↓↓↓↓	↔↔
Versioonide dokumendid	23	↓↓↓↓↓↗	□↓↓↓↓↓↓↗↔
Versioonide plaanid	22	↓↓↓↓↓↓↓↓↓↗	□↓↓↓↓↓↓↗
Plaanide kontroll	13	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Kvaliteedi grupp osaleb	19	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Loodava tarkvara audit	20	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Riskide jälgimine	15	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	□
Plaani muutmise reeglid	16	↓↓↓↓↓↓↓↓×↓↓↓↓	↓↗↓↗
Plaanijälgimise reeglid	17	↓↓↓↓↓↓↓↗	↔↔
Kõik plaaniga nõus	14	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	□↓↗
Versioonide audit	24	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Toodangu audit	21	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
Metoodika audit		↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↓↓↓↓↓↓×↓↓↓↗↓↗
Elutsükliid	9	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↔
↔			
Nõuete analüüs	7	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	×↓↗↔
□↓↓↓↓↓↓↓↗			
Nõuete muutus	8	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	□↓↓↓↓↓↓↗
↔↔			
Nõuete haldus	6	↓↓↓↓↓↓↓↓↓↓↓↓↓↓	↗
↔↔			
Projektijuht olemas			3
↓↓↓↓↓↓↓↓↓↓↓↓↓↓		↔	
Arendajad osalevad			10
↓↓↓↓↓↓↓↓↓↓↓↓↓↓			

## Arendusmeetoditega seotud tunnuste vahelised Pearsoni korrelatsioonikordajad

	T1	T2	PKK		T1	T2	PKK
1	11	14	,470	9	1	7	,319
2	8	10	,441	10	1	5	,313
3	1	4	,413	11	1	6	,300
4	1	2	,399	12	1	3	,299
5	1	11	,375	13	1	15	,299
6	1	9	,370	14	1	16	,235
7	6	12	,359	15	1	13	,233
8	5	8	,327				

[illegible]

## Hirmudega seotud tunnuste vahelised Pearsoni korrelatsioonikordajad

	T1	T2			T1	T2	
1	3	4	,811	5	1	5	,352
2	7	9	,512	6	1	6	,257
3	1	7	,433	7	1	3	,219
4	5	8	,430	8	1	2	,180

Pole tark 3  
 Pole teadmisi 4  
 Kvalit. Ohverdamine 7  
 Pole piisavalt aega 9  
 Ebareaalsed tähtajad 1  
 Lootusetud olukorrad 5  
 Ei saa abi 8  
 Arusaamatud vajadused 6  
 Rakendust pole vaja 2

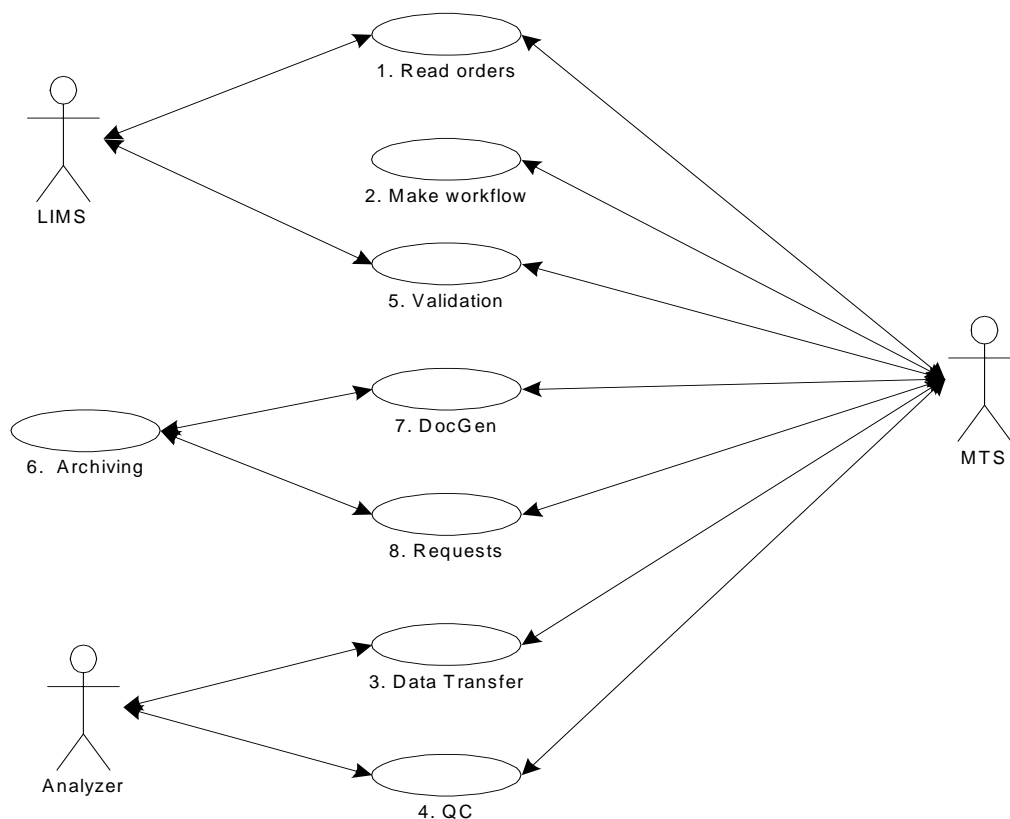
## Lisa.5. Näited XP meetodite kasutamisest firmas Sysstek

### OCS metafoor

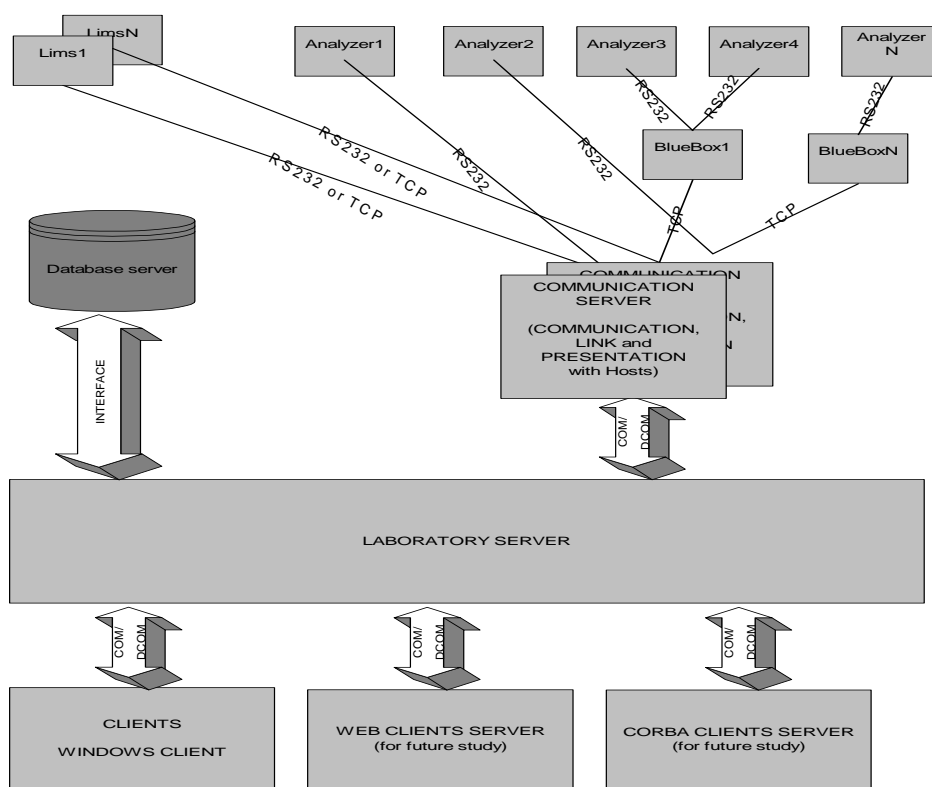
OCS on verelaboratooriumi infosüsteem.

### OCS põhifunktsioonid

1. Laboratooriumi uuringuteks saadetud materjalidega kaasnevate päringute sisestamine (Milliseid parameetreid tuleb uurida?);
2. Materjali katsutite marsrutiseerimise hõlbustamine (Millised seadmed ja millises järjekorras on otstarbekas kasutada?);
3. Elektroonne andmevahetus analüsaatorite ja serveri vahel (Igale seadmele tuleb saata päring teostatavatest mõõtmistest ja saada vastuseks tulemused );
4. Kohustusliku kvaliteedikontrolli hõlbustamine (Perioodiliselt tuleb kontrollida seadmete korrektsust, lastes seadmetel sooritada mõõtmisi etalonverele ja võrrelda tulemusi. );
5. Tulemuste õigsuse kontrollimise ja kinnitamise hõlbustamine (Lõpliku hinnangu analüüsidele annab laboratooriumis töötav arst. Vajadusel kiire raviarsti teavitamine.);
6. Andmete ahiveerimine (Kõike tuleb säilitada kaitseks võimalike tulevaste kohtuprotsesside jaoks.)
7. Kõikvõimalike dokumentide vormistamise hõlbustamine (generaator)
8. Kõikvõimalike päringute genereerimine (Päringute generaator)



## OCS ülesehitus.



Selgitused joonise juurde:

- *LIMS* on verelaboratooriumi infosüsteem, mis on arendatud kolmandate osapoolte poolt;
- *Analyzer* on laboratooriumi seade, mis teostab analüüse, või sorteerib katsuteid;
- *BlueBox* – on PC või mingi muu elektrooniline seade selleks, et ainult RS232 porti omavaid laboratooriumi seadmeid ja/või informatsioonisüsteeme saaks vajadusel ühendada TCP/IP võrku;
- *Communication server*, *Laboratory server*, *Windows client*, *Web Clients Server* ja *CORBA Clients Server* on OCS loogiliselt iseseisvad moodulid.

[illegible]

## Iteratsiooni plaan

Summary\_Remarks\_001\_013.doc - Microsoft Word

File Edit View Insert Format Tools Table Window Help

Normal

MultiLab for Windows

System

Version: 001 Date: 03/05/2003 Page: 1 von 1

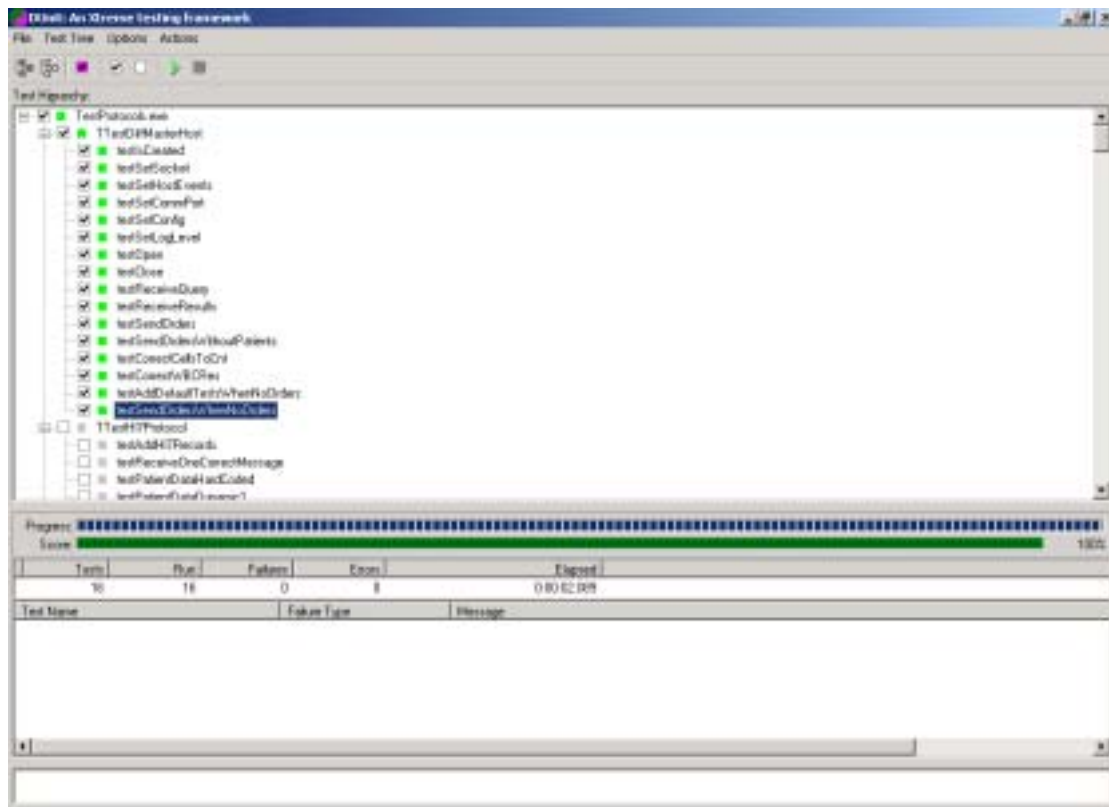
Project: MFW Summary Remarks from 001 until 013

Remarks No / Page	Remark Description	Priority
Remarks 002 / Page 1	In the tree the shortpaths of test	high
Remarks 008 / Page 1	Screen design (RUBAK Grid)	high
Remarks 009 / Page 1	Slowly if I start the module	
	If I want enter new QC Data, I must press "Insert" and then "Edit"	
	If I want enter the first QC Data its not possible, if the empty.	
	Screen	high
Remarks 010 / Page 1	Enter new charge, I must close and open.	
	Data fields	
Remarks 010 / Page 2	Screen	high
Remarks 010 / Page 3	Screen	high
	Buttons	
Remarks 010 / Page 4	Grid - Delete column TYP	high
	Its functionality	
Remarks 010 / Page 5	Screen	high
	Buttons	high
	Button "Save collection"	
Remarks 010 / Page 6	Screen	high
	Buttons	high
	Checkbox "Active"	
Remarks 011 / Page 1	Imprecision form the single value - format	high
Remarks 011 / Page 2-9	Functionality for comments with the wizard	high
Remarks 012 / Page 1	Mistakes IP and IC	
Remarks 012 / Page 2	Comments	high
Remarks 012 / Page 3-4	Cutoff for the first controlcycle	high
Remarks 012 / Page 5	Check for controls without "3s-Rule"	
Remarks 013 / Page 1	Check module Error	high
	Sorting of test in Grid "Qc"	high

Page 1 Sec 1 1/1 At 4cm Ln 1 Col 1 REC TRK EXT OVR English (U.S.)



## Testimise vahend DUnit



## Lisa.6. Tarkvaraprojekti test

Syspek Informationsystems OÜ projektide enesehinnang Peeter Normaku [NORMAK2001] loengukonspektis toodud tarkvaraprojekti testi järgi.

Igale küsimusele tuleb vastata neljapallisüsteemis: 3 "jah" korral, 2 "tõenäoliselt", 1 "pigem mitte" ja 0 "ei" korral.

### Nõuded

1. (3) Kas projektil on olemas selge ja arusaadav lühikokkuvõte või eesmärgiseade?
2. (3) Kas kõik projektimeeskonna liikmed peavad eesmärgi saavutamist realistlikuks?
3. (2) Kas on analüüsitud majanduslikku tulu ja seda, kuidas seda tulu mõõdetakse?
4. (3) Kas projektil on kasutajaliidese prototüüp mis realistlikult ja ilmekalt demonstreeriks tegeliku süsteemi funktsionaalsust?
5. (2) Kas projektis on detailne kirjalik spetsifikatsioon selle kohta, mida tarkvara peaks tegema?
6. (3) Kas projektimeeskond küsitles tarkvara tegelikke lõppkasutajaid projekti koostamisel ja kaasab neid projekti jooksul?

### Kavandamine

7. (1) Kas projektil on detailne kirjalik Tarkvaraarendusplaan?
8. (3) Kas projekti käigus on kavas koostada installatsiooniprogramm, andmete konverteerimise võimalus, muu tarkvara integreerimine, kohtumisi tarbijatega jne?
9. (3) Kas ajagraafikut ja eelarvet täiendati peale viimati sisseviidud muudatusi?
10. (2) Kas projektil on detailsed kirjalikud struktuuri ja disaini kirjeldavad dokumendid?
11. (0) Kas projektis on kirjalik Kvaliteedi Kindlustamise Plaan mis lisaks süsteemi testimisele nõuaks ka disaini ja lähtekoodi ülevaateid?
12. (3) Kas projektis on detailne tarkvara üleandmise plaan, mis kirjeldaks tarkvara rakendamise ja üleandmise faase?
13. (2) Kas projekt sisaldab puhkuseaega, puhkepäevi, haiguspäevi, täienduskoolitust, ning kas on kasutatud alla 100% ressursse?
14. (2) Kas projekt (s.h. ajagraafik) on heaks kiidetud projektimeeskonna poolt, s.t. tegelike täitjate poolt?

### Projekti juhtimine

15. (3) Kas on keegi otsustusõigusega isik tehtud projekti eest vastutavaks ning kas projektil on tema aktiivne toetus?
16. (3) Kas projekti juhi töökoormus lubab tal pühendada projektile piisavalt aega?
17. (3) Kas projektil on piisavalt määratletud alamülesanded, mille täidetus või mittetäidetus on kontrollitav?
18. (2) Kas projekti dokumentatsiooni järgi on alamülesannete täidetus kergelt kontrollitav?
19. (2) Kas projektis on tagasisidekanal, mille kaudu projektiliikmed saavad anonüümselt ette kanda probleemidest?
20. (2) Kas projektis on kirjalik kava tarkvaraspetsifikatsiooni muudatuste jälgimiseks?
21. (3) Kas projektis on Juhtrühm, kellel on lõplik õigus otsustada muudatuste sisseviimiseks või neist keeldumiseks?
22. (3) Kas projekti töömaterjalid (ajagraafikud, töökohustused, projekti täitmise järg jne) kättesaadavad projektimeeskonna igale liikmele?
23. (1) Kas kogu lähtekood allub automaatsele ülevaatussele ja kontrollile?
24. (3) Kas töökeskkond sisaldab projektitäitmiseks vajalikke vahendeid (Veakontrollivahendid, lähtekoodikontroll, projekti juhtimistarkvara)?

### Riskijuhtimine

25. (1) Kas projektis on formuleeritud ka riskide loetelu? Kas seda loetelu on hiljuti täiendatud?
26. (1) Kas projektitäitjate hulgas on inimene, kelle ülesandeks on tekkivate riskide identifitseerimine?
27. (3) Kui kasutatakse alltöövõtte, kas on iga organisatsiooniga suhtlemise kava ning selle eest vastutav inimene (Kui alltöövõtte ei kasutata, siis anda maksimumpunktid)?

### Personal

28. (2) Kas projektimeeskond omab projekti täitmiseks vajaliku kompetentsi?
29. (2) Kas projektimeeskond omab kompetentsi, opereerimaks ärikeskkonnas, kuhu tarkvara on orienteeritud?
30. (2) Kas projektil on liider, kes oleks võimeline projekti edukalt juhtima?

31. (2) Kas kõikide vajalike tööde tegemiseks jätkub piisavalt inimesi?  
32. (3) Kas inimeste omavaheline koostöö klappib?  
33. (3) Kas kõik täitjad on piisavalt projekti täitmisse pühendunud?

#### **Kokkuvõte**

- 76 Esialgne summa.  
1,25 Tegur: 1.5, 1.25 või 1 kui projektis on vastavalt 3 või vähem, 4 kuni 6 või rohkem täistöökohta ekvivalenti.  
95 Lõpptulemus = eelmise kahe arvu korrutis.

#### **Projekti hinnang:**

- 90+ (väljapaistev, edu peaaegu garanteeritud);
- 80-89 (suurepärane, edu tõenäosus väga kõrge);
- 60-79 (hea, üle keskmise, mis tõenäoliselt kas väljub eelarvest või ajalimiidist);
- 40-59 (keskmine, puudujäägid tarkvara funktsionaalsuses, ületab nii aja kui eelarve, projekti täitmine konarlik);
- alla 40 (riskantne, põhinõuete osas tõsised puudujäägid, nõrk kavandamine, järelvalve, riskikäsitus ja personal, enamasti ei õnnestu lõpetada).