LUT University

TUOTANTOTALOUDEN KOULUTUSOHJELMA

Työmäärän arviointi ohjelmistoprojektissa

Effort Estimation in Software Project

Kandidaatintyö

Jaakko Hallikainen

TIIVISTELMÄ

Tekijä: Jaakko Hallikainen

Työn nimi: Työmäärän arviointi ohjelmistoprojektissa

Vuosi: 2019 **Paikka: Lappeenranta**

Kandidaatintyö. LUT-yliopisto, tuotantotalous.
29 sivua, 7 kuvaa ja 1 taulukko.
Tarkastaja(t): Antero Kutvonen.

Hakusanat: Ohjelmistoprojektin hallinta, työmäärän ennustaminen
Keywords: Software project management, effort estimation

Tämän kandidaatintyön tavoitteena on kartoittaa syitä ohjelmistoprojektin työmäärän arviointiin liittyvien ongelmien syitä ja mahdollisesti keksiä ongelmaan ratkaisuja.

Työssä ensin kartoitetaan tyypillisiä ohjelmistoprojektin läpivientiin käytettäviä malleja, kuten vesiputousmallia ja ketteriä menetelmiä. Sitten pureudutaan ohjelmistoprojektien tyypillisiin ongelmiin. Tyypillisten ongelmien jälkeen kartoitetaan ohjelmistoprojektin läpivientiin vaadittavan työmäärän ennustamiseen käytettyjä menetelmiä. Menetelmistä pyritään havaitsemaan ongelmakohtia, ja keksimään niihin ratkaisuja.

Työssä havaittiin, että ohjelmistoprojekti voidaan saada epäonnistumaan todella monella eri tavalla kaikissa ohjelmistoprojektin eri vaiheissa. Myös havaittiin, että tyypillisimmät ongelmat voidaan ratkaista todella huolellisella ja objektiivisella suunnittelulla.

Työssä myös havaittiin, että ohjelmistoprojektin vaatiman työmäärän arviointiin käytettävät menetelmät usein eivät ota kaikkia tarpeellisia työmäärään vaikuttavia seikkoja huomioon. Tästä syystä arviointiin pitäisi kehittää sellainen menetelmä, joka ottaa kaikki työmäärään vaikuttavat tekijät huomioon järkevässä mittasuhteessa. Työmäärän arvioinnissa havaittiin suureksi ongelmaksi se, että

projektin aikataulutuksesta on vastuussa ne henkilöt, joilla ei ole riittävää ymmärrystä ohjelmiston teknisestä toteutuksesta, vaan heidän näkemyksensä on puhtaasti liiketaloudellinen. Lisäksi havaittiin, että projektin läpiviennin mallillakin on oma osuutensa asiaan. Suurten projektien vaatiman työmäärän arviointi on huomattavasti vaikeampaa kuin pienten projektien. Näin ollen suurissa projekteissa ketteriä menetelmiä käytettäessä työmäärän arviointi on huomattavasti mielekkäämpää.

SISÄLLYSLUETTELO

1 JOHDANTO

Viimeaikainen tietojenkäsittelytaitojen kehittyminen on johtanut uuteen aikakauteen, jossa kaikenlaisia laitteita on liitetty verkkoon erilaisten antureiden ja sensoreiden avulla. Verkkoon liitetyt laitteet pystyvät keräämään kaikenlaista tietoa ympäriltään. Lisäksi laitteet pystyvät viestimään muiden laitteiden kanssa, sekä toimimaan käsittelemänsä datan perusteella älykkäästi. Osin tästä syystä ohjelmistoalalla siirrytään suuntaan, jossa ohjelmistojen kompleksisuus on kasvanut huomattavasti. Kompleksisuuden kasvun myötä ohjelmistojen valmistukseen käytettävä työmääräkin kasvanut merkittävästi.

Viimeaikainen teknologian kehitys on avannut aivan uusia liiketoimintamahdollisuuksia. Yritykset pystyvät valmistamaan tuotteitaan huomattavasti kustannustehokkaammin, kun toimintaa pystytään automatisoimaan entistä enemmän. Yrityksillä on myös käytössä entistä enemmän tietoa markkinoilla tapahtuvista muutoksista, sekä omasta tehokkuudestaan. Yritykset pystyvät myös markkinoimaan tuotteitaan entistä tehokkaammin erilaisten verkkokanavien kautta kohdentamalla mainontaa potentiaalisille asiakkailleen. Vastaavasti kuluttajat saavat heitä itseään kiinnostavista tuotteista mainoksia. Kuluttajilla on myös mahdollisuus ostaa haluamiaan tuotteita verkossa, sekä mahdollisuus saada aivan uudenlaista tietoa itsestään, ystävistään, sekä ympäristöstään yhä useammassa laitteessa entistä helpommin ja nopeammin.

McConnell:n (2002) mukaan monissa tutkimuksissa on todettu, että noin kaksi kolmasosaa ohjelmistoprojekteista epäonnistuu (kuva 1). Suurista aikataulunsa ylittävistä projekteista aikataulu venyy noin 25-50 % alkuperäiseen suunnitelmaan nähden. Tyypillisesti mitä suurempi ohjelmistoprojekti on kyseessä, sitä enemmän aikataulu ylittyy (McConnel 2002.) Yksin Suomessa viivästyneet ohjelmistoprojektit tuovat vuosittain kuluja lisää noin 7 miljardia euroa (Järvenpää & Kankare 2013). Suuri osa ohjelmistoprojektien epäonnistumisista johtuu siitä, että projektin vaatimaa työmäärää ei ole osattu arvioida oikein (Juvonen 2018, s. 82-83). Ohjelmistoprojektin viivästymiset näkyvät valmiiden ohjelmistojen ylisuurten hintojen lisäksi valmistavien yritysten valtavana imagotappiona. Ohjelmistoprojekti voi epäonnistua niin huonon vaatimusmäärittelyn, huonon ajankäytön suunnittelun, kuin myös valmiin tuotteen huonon laadun takia.

Kuva 1. Ohjelmistoprojektien onnistumiset ja epäonnistumiset

Tämän kandidaatintyön tarkoituksena on tarkastella kirjallisuudessa suositeltuja malleja ohjelmiston valmistukseen tarvittavan työmäärän arvioimiseen. Työn keskeisimpiä kysymyksiä on, mitä keinoja on yleisesti käytetty ongelman ratkaisemiseksi, mitä keinoja alan yritykset ovat kehittäneet, tai käyttäneet ja miksi löydetyt keinot eivät välttämättä toimi. Työssä pyritään myös määrittämään, miten mallien mahdollisia ongelmakohtia voitaisiin kehittää tulevaisuudessa.

Kirjallisuuskatsauksessa perehdytään yleisiin ohjelmistoprojektien ongelmiin, jonka jälkeen pureudutaan keinoihin, joilla ohjelmistoprojektien työmäärää on arvioitu. Työssä myös pyritään havaitsemaan mahdollisia ongelmakohtia kirjallisuudesta löydetyistä työmäärän ennustamisen menetelmistä. Jos menetelmissä havaitaan selviä ongelmakohtia, pyritään keksimään keinoja ongelmien ratkaisemiseksi, tai aikakin miten ongelmaa voitaisiin alkaa ratkaista.

2 OHJELMISTOPROJEKTI

Projekti on kertaluontoinen suunnitelmallinen tehtävä jonkin tavoitteen toteuttamiseksi (Graham & Portny 2013). Samat projektinhallinnan periaatteet toimivat kaikilla toimialoilla, olipa sitten kyse teollisuushallin rakentamisesta, tai ohjelmiston valmistamisesta. Nimensä mukaisesti ohjelmistoprojekti on siis kertaluontoinen ja uniikki tehtävä jonkinlaisen ohjelmiston valmistamiseksi. Ohjelmistoprojektin tarkoitus on tuottaa pääomaa ohjelmiston tuottajille myymällä valmistettua ohjelmistoa asiakkaalle, joka voi olla yritys, valtio, tai tavallinen kuluttaja.

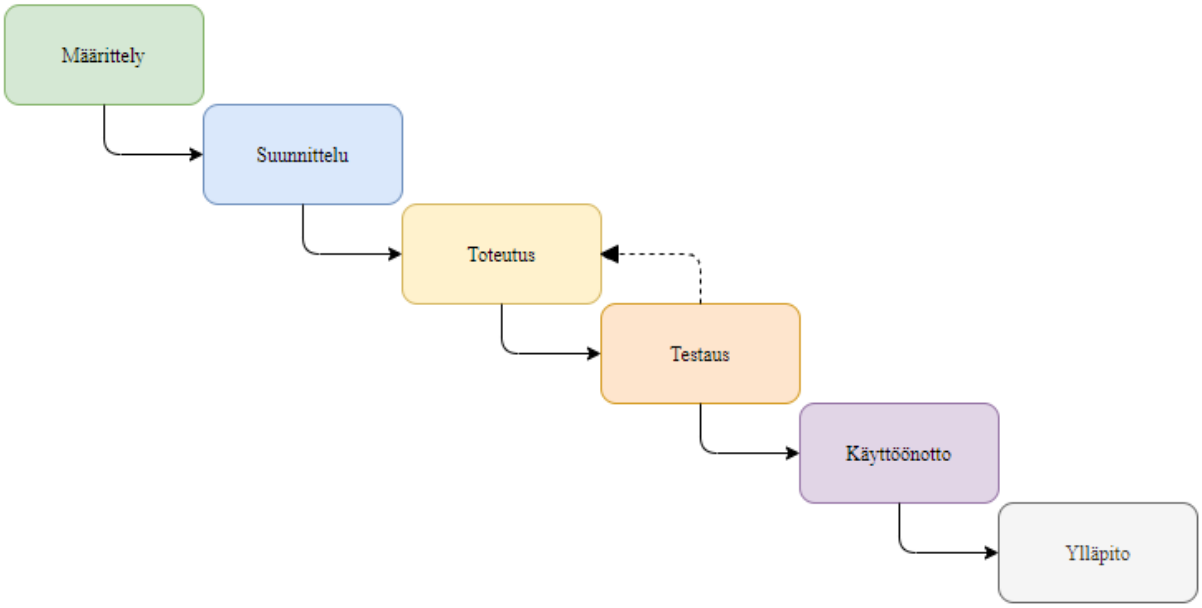
Yritysmarkkinoilla, tai valtiolle tehtävät ohjelmistoprojektit lähtevät aina liikkeelle asiakkaan tarpeesta. Asiakkaan tarve liittyy usein esimerkiksi toiminnan tehostamiseen, informaation saamiseen, tuottamiseen ja kulkuun, tai toiminnan automatisoimiseen (Armstrong 2018). Kuluttajamarkkinoilla taas asiakkaan tarve voi perustua esimerkiksi viihdyttämiseen, ihmissuhteiden ylläpitämiseen, tai tiedon saantiin ja jakamiseen. Asiakkaan ongelma ratkaistaan kehittämällä ohjelmisto, joka poistaa asiakkaan ongelman.

Ohjelmistoprojektien läpivientiin käytetään yleensä kolmea erilaista mallia: vesiputousmalli (waterfall), ketteriä menetelmiä, kuten esimerkiksi scrum- menetelmää, sekä protoilumallia. Monesti kuitenkin käytetään näiden mallien erilaisia yhdistelmiä (Kuhrmann et al. 2016, s. 49).

2.1 Vesiputousmalli

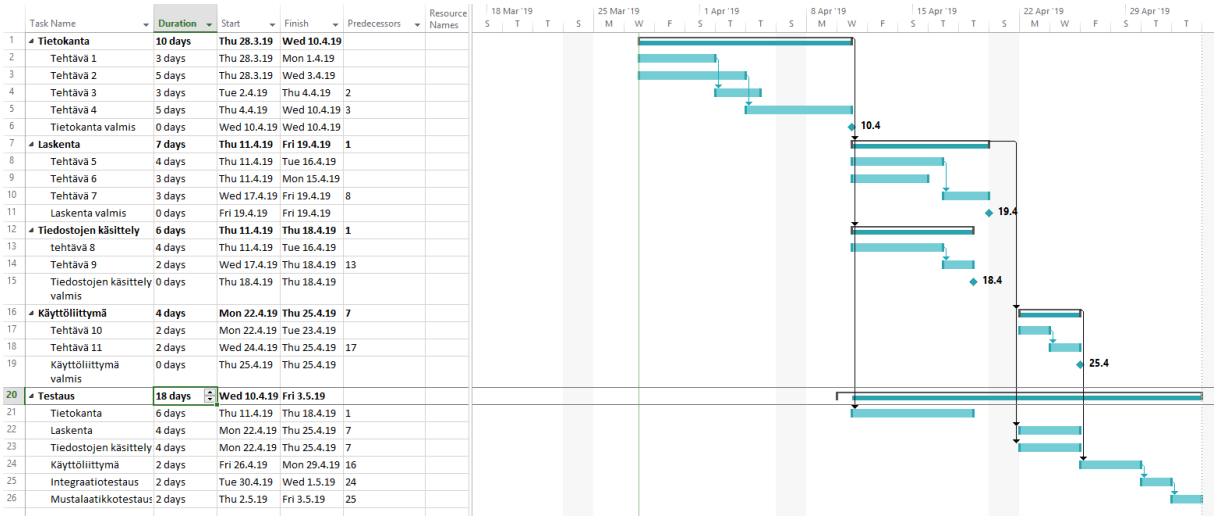
Vesiputousmallia (kuva 2) käytettäessä projekti alkaa huolellisella vaatimusmäärittelyllä usein asiakkaan kanssa yhteistyössä. Vaatimusmäärittelyssä pyritään saamaan asiakkaalta mahdollisimman paljon tietoa asiakkaan liiketoiminnasta, sekä varsinaisista tarpeista. Vaatimusmäärityksen perusteella tehdään hyvin yksityiskohtainen suunnitelma valmistettavan ohjelmiston rakenteesta, sekä projektin läpiviennistä. Yksityiskohtaisen suunnittelun jälkeen siirrytään toteutusvaiheeseen, jolloin kirjoitetaan varsinainen ohjelmakoodi. Koska ohjelmakoodiin tulee väkisinkin virheitä, tulee ohjelmisto testata mahdollisimman tarkasti. Näin varmistutaan ohjelmiston laadusta, sekä näin ollen asiakastyytyväisyydestä. Virheiden

löydyttyä siirrytään takaisin toteutusvaiheeseen, jolloin havaitut virheet korjataan. Toki, jos havaitaan puutteita, tai virheitä ohjelmiston määrittelyyn, tai suunnitteluun liittyen, palataan takaisin kohtaan, jossa virhe on tehty, oltiinpa sitten menossa missä projektin vaiheessa tahansa. Kun on varmistuttu siitä, että valmistettu ohjelmisto on asiakkaan vaatimukset täyttävä, tai erityisesti kuluttajamarkkinoilla projektiin käytettävän ajan loppuessa, luovutetaan ohjelmisto asiakkaalle. Käyttöönoton jälkeen siirrytään ohjelmiston ylläpitovaiheeseen, jolloin voidaan vielä korjata asiakkaan havaitsemia virheitä, parantaa ohjelmiston suoritustehoa, tai tietoturvaa yms. (Haikala & Mikkonen 2011, s. 36-38)



Kuva 2. Vesiputousmallin vaiheet

Vesiputousmallia pyritään myös hahmottamaan erilaisten Gantt- kaavioiden avulla (kuva 3). Gantt- kaavion ideana on pilkkoa projekti pieniin tehtäviin ja kuvata koko projektin kulkua visuaalisesti. Kun koko projekti on pilkottu riittävän pieniksi palasiksi, määritetään tehtävien tekemiseen järjestys, eli mikä tehtävä tulee tehdä ennen mitäkin tehtävää, mitkä tehtävät voidaan tehdä samaan aikaan, miten tehtävät ovat toisistaan riippuvaisia jne. Järjestyksen määrittämisen jälkeen tehtävien työmäärä arvioidaan ajassa mitattuna, jonka jälkeen tehtävät laitetaan tiettyjen henkilöiden, tai ryhmien vastuulle. Näin saadaan aikaiseksi hyvin visuaalinen hahmotelma projektin kulusta. (Kukhnavets 2018)



	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Tietokanta**	**10 days**	**Thu 28.3.19**	**Wed 10.4.19**		
2	Tehtävä 1	3 days	Thu 28.3.19	Mon 1.4.19		
3	Tehtävä 2	5 days	Thu 28.3.19	Wed 3.4.19		
4	Tehtävä 3	3 days	Tue 2.4.19	Thu 4.4.19	2	
5	Tehtävä 4	5 days	Thu 4.4.19	Wed 10.4.19	3	
6	Tietokanta valmis	0 days	Wed 10.4.19	Wed 10.4.19		
7	**Laskenta**	**7 days**	**Thu 11.4.19**	**Fri 19.4.19**	**1**	
8	Tehtävä 5	4 days	Thu 11.4.19	Tue 16.4.19		
9	Tehtävä 6	3 days	Thu 11.4.19	Mon 15.4.19		
10	Tehtävä 7	3 days	Wed 17.4.19	Fri 19.4.19	8	
11	Laskenta valmis	0 days	Fri 19.4.19	Fri 19.4.19		
12	**Tiedostojen käsittely**	**6 days**	**Thu 11.4.19**	**Thu 18.4.19**	**1**	
13	tehtävä 8	4 days	Thu 11.4.19	Tue 16.4.19		
14	Tehtävä 9	2 days	Wed 17.4.19	Thu 18.4.19	13	
15	Tiedostojen käsittely valmis	0 days	Thu 18.4.19	Thu 18.4.19		
16	**Käyttöliittymä**	**4 days**	**Mon 22.4.19**	**Thu 25.4.19**	**7**	
17	Tehtävä 10	2 days	Mon 22.4.19	Tue 23.4.19		
18	Tehtävä 11	2 days	Wed 24.4.19	Thu 25.4.19	17	
19	Käyttöliittymä valmis	0 days	Thu 25.4.19	Thu 25.4.19		
20	**Testaus**	**18 days**	**Wed 10.4.19**	**Fri 3.5.19**		
21	Tietokanta	6 days	Thu 11.4.19	Thu 18.4.19	1	
22	Laskenta	4 days	Mon 22.4.19	Thu 25.4.19	7	
23	Tiedostojen käsittely	4 days	Mon 22.4.19	Thu 25.4.19	7	
24	Käyttöliittymä	2 days	Fri 26.4.19	Mon 29.4.19	16	
25	Integraatiotestaus	2 days	Tue 30.4.19	Wed 1.5.19	24	
26	Mustalaatikkotestaus	2 days	Thu 2.5.19	Fri 3.5.19	25	

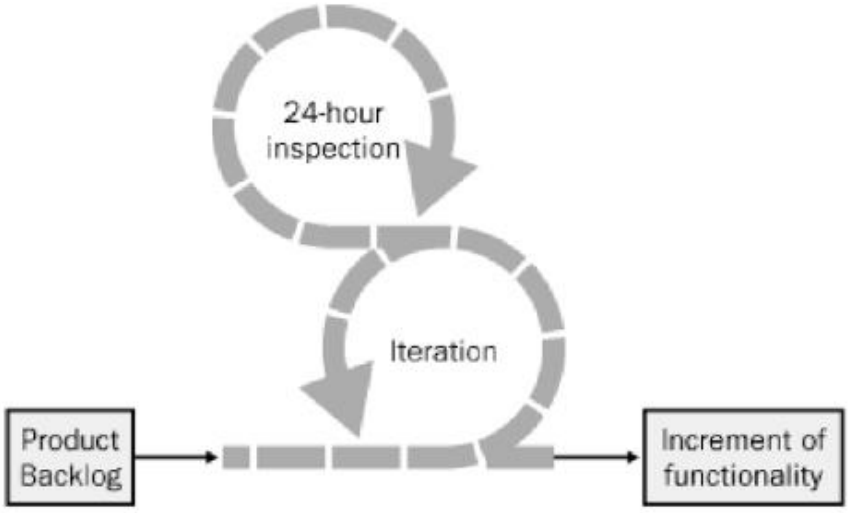
Kuva 3. Esimerkki Gantt- kaaviosta

Vesiputousmallia käytetään tyypillisesti silloin, kun projektin lopputulos on tarkasti tiedossa jo projektin alkuvaiheessa, eikä asiakkaan tarpeet tule todennäköisesti muuttumaan projektin edetessä. Tällöin myös asiakasta ei varsinaisesti tarvita projektin läpiviennin aikana (Castillo 2016.) Vesiputousmallia käytettäessä työmäärää, ja sitä kautta valmiin ohjelmiston kustannukset ovat jokseenkin mahdollista arvioida, sillä koko ohjelmiston vaatimukset ovat tiedossa. Varsinkin suurissa projekteissa työmäärän arvioiminen on hyvin hankalaa. Bannik (2014) toteaa, että koska nykyisin maailma muuttuu kovaa vauhtia, asiakkaan tarpeet saattavat hyvinkin muuttua ohjelmiston valmistuksen aikana. Näin ollen monesti ohjelmistoprojektit pyritään viemään muilla tavoilla läpi, kuten esimerkiksi käyttämällä ns. ketteriä menetelmiä. (Bannik 2014.)

Vesiputousmallia käytettäessä projektit eivät juuri koskaan valmistu etuajassa. Syynä tähän on se, että projektitiimi sovittaa usein työtahtinsa siten, että työvaihe on valmis juuri ennen ennalta määriteltyä ajankohtaa. Joskus vesiputousmallia käytettäessä projekti valmistuu ajallaan. Yleensä projekti kuitenkin myöhästyy, koska projektin toteutus vie enemmän aikaa, kun osataan arvioida. Lisäksi usein viimehetken töiden vaatimaa työmäärää ei osata ottaa huomioon. Myöhästyneellä projektilla on myös usein negatiivisia vaikutusta yrityksen tuleviin projekteihin, koska työvoimaa ei saada siirrettyä seuraaviin projekteihin ennen edellisen valmistumista. (Juvonen 2018, s. 103)

2.2 Scrum ja ketterät menetelmät

Toisin, kuin vesiputousmallissa, scrum- mallissa (kuva 4) koko ohjelmistoa ei suunnitella kerralla, vaan ohjelmistoa suunnitellaan vähän kerrallaan yhteistyössä asiakkaan kanssa. Tyypillisesti asiakkaan kanssa määritellään kerralla jokin ohjelmiston ominaisuus (Product Backlog). Ominaisuuden määrittelyn jälkeen suunnitellaan ominaisuuden toteutus (Sprint Backlog), minkä jälkeen toteutetaan ja testataan ominaisuus yhtenä pyrähdyksenä (Sprint). Yhden pyrähdyksen pituus vaihtelee ajassa mitattuna tyypillisesti noin kahden ja neljän viikon välillä. Kun ominaisuus on onnistuneesti toteutettu, integroidaan se jo olemassa olevaan ohjelmistoon. Kun integraatiotestauksen jälkeen on varmistuttu ohjelmiston toimivuudesta, julkaistaan ohjelmiston uusin versio. (Castillo 2016)



Kuva 4. Hahmotelma scrum- mallista. (Schwaber 2004)

Ketteriä menetelmiä käytetään tyypillisesti silloin, kun asiakas ei vielä itsekään tiedä, millainen on projektin lopputulos, tai asiakkaan vaatimukset voivat muuttua ohjelmiston valmistuksen edetessä. Ketteriä menetelmiä käytettäessä asiakas on siis hyvin vahvasti mukana projektin etenemisessä (Castillo 2016.) Koska ketteriä menetelmiä käytettäessä ei usein voida varmasti arvioida projektin laajuutta etukäteen, on projektin työmäärää, ja sitä kautta valmiin ohjelmiston kustannuksia on lähes mahdotonta arvioida etukäteen. Kuitenkin yhden pyrähdyksen vaatima työmäärä on jokseenkin helposti arvioitavissa.

2.3 Protoilu

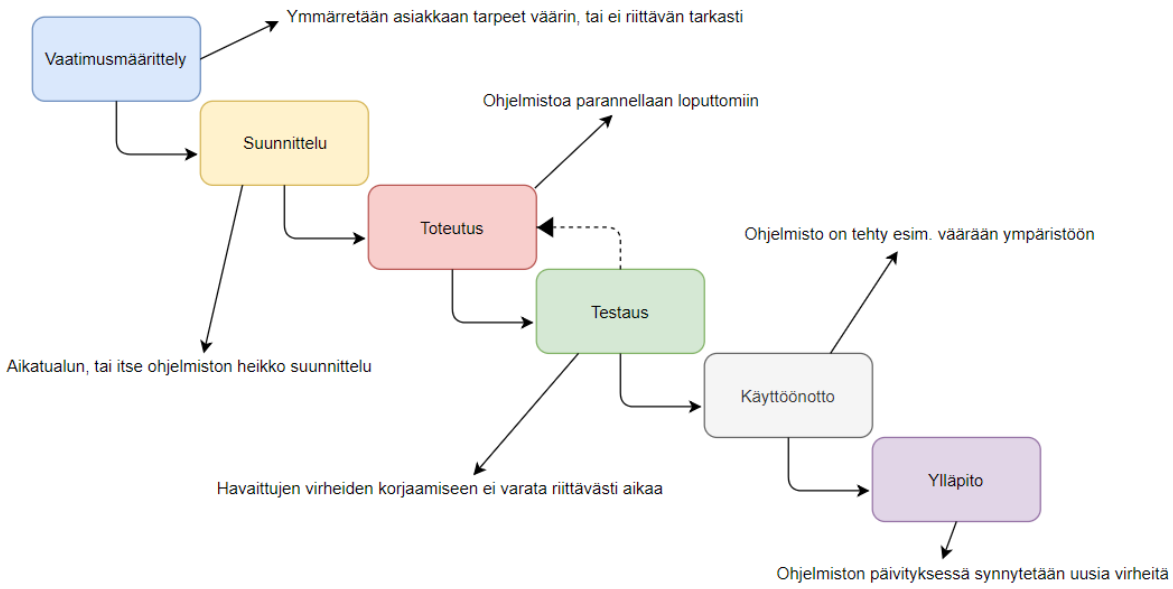
Protoilulla tarkoitetaan sitä, että valmistetaan vaillinainen prototyyppi ohjelmiston uudesta ominaisuudesta ja tutkitaan ominaisuuden toimimista. Protoilun kohteita voivat olla esimerkiksi uudenlainen käyttöliittymä, uuden toteutusteknologian käyttö, tai vaikka suorituskyvyn, tai uudenlaisen muistinkulutuksen tarkempi tutkiminen. (Haikala & Mikkonen 2011, s. 38)

Protoilussa on kaksi päävaihtoehtoa. evoluutioprototyyppi (Evolutionary prototype), jossa prototyypistä iteroidaan pikkuhiljaa valmis tuote. Toinen malli on poisheitettävä prototyyppi, tai kertakäyttöinen prototyyppi (throwaway prototype) jota käytetään lopullisen tuotteen mallintamiseen. Mallintamisen ideana on kysyä asiakkaan mielipide prototyypistä. Kertakäyttöisessä prototyypissä prototyyppi unohdetaan, jonka jälkeen suunnittelu aloitetaan uudelleen asiakaspalautteen perusteella. Monesti käytetään myös näiden välimuotoa, jolloin osaa ohjelmiston toimivia osia kehitetään entistä paremmiksi, kun taas ei- toimivat osat poistetaan ja korvataan uusilla paremmin toimivalla toteutuksella. (Storrs & Windsor 1992)

Protoilua käytetään monesti kuluttajamarkkinoilla esimerkiksi videopelien valmistuksessa. Monesti julkaistaan pelistä Beta- versio, jonka tarkoituksena on määrittää, mikä pelin ominaisuus toimii ja vastaavasti mikä ominaisuus mahdollisesti ärsyttää pelin käyttäjää. Kun on saatu tarpeeksi informaatiota käyttäjien mielipiteistä, korjataan, tai poistetaan ärsyttävät ominaisuudet ja kehitetään niitä ominaisuuksia, joista käyttäjät pitävät. Nykyaikainen teknologia on mahdollistanut sen, että varsinaisen tuotteen lanseerauksen jälkeen kehitystoimintaa voidaan jatkaa paremman käyttäjäkokemuksen turvaamiseksi erilaisten verkosta ladattavien päivitysten muodossa.

3 TYYPILLISET ONGELMAT OHJELMISTOPROJEKTISSA

Ohjelmistoprojektin epäonnistumisella tarkoitetaan sitä, että projektin aikataulu viivästyy, projektin budjetti ylittyy, tai valmistettu ohjelmisto ei täytä asiakkaan vaatimuksia. Viivästyminen, budjetin ylittyminen, tai vaatimusten täyttämättömyys on usein vain jäävuoren huippu, eli projektissa on saatettu tehdä useitakin eri asioita väärin (kuva 5).



Kuva 5. Virheiden mahdollisuudet ohjelmistoprojektin eri vaiheissa

Ohjelmistoprojektin epäonnistuminen maksaa monesti asiakkaalle paljon. Ohjelmistohankkeiden kustannuksista reilusti yli valtaosa syntyy kaiken kehitystyön ja käyttöönoton jälkeen (Järvenpää & Kankare 2013, s. 36). Näin ollen suuri osa pelkästään heikon laadun takia syntyvät ohjelmiston käytön kustannukset heikentävät ohjelmiston tilanneen yrityksen tulosta merkittävästi. Vastaavasti valtion tilaaman ohjelmiston suuret käytön kustannukset näkyvät tavallisten veronmaksajien lompakoissa.

Ohjelmistoprojektin epäonnistuminen on sitä kalliimpaa, mitä aiemmassa vaiheessa epäonnistuminen tapahtuu, varsinkin jos virhe havaitaan vasta, kun ohjelmisto on luovutettu asiakkaalle. Jos projektin suunnitteluvaiheessa tehdään suuri virhe, ovat virheestä johtuneet

korjauskustannukset todella suuret. Syynä siihen on se, että virheen korjaamiseen kuluu todella paljon resursseja. Resursseja kuluu siihen, että koko ohjelmisto voidaan joutua ensin suunnittelemaan uudelleen, jonka jälkeen kirjoitetaan ohjelmakoodi ja ohjelmisto testataan uudelleen. Kuitenkin jos virhe tapahtuu projektin myöhemmässä vaiheessa, on kustannukset huomattavasti pienemmät, koska virheen korjaaminen on verrattain helppoa, eikä se vie kovinkaan paljoa resursseja. Vastaavasti mitä nopeammin virhe huomataan projektin aikana, sitä edullisempaa on virheen korjaaminen. Syy tähän on se, että ns. “turhaa työtä” virheen tekemisen jälkeen ei olla ehditty tekemään.

3.1 Vaatimusmäärittely

Usein ajatellaan, että vaatimusmäärittely on ohjelmistoprojektin kriittisin vaihe. Monesti ohjelmistoprojektin epäonnistuminen johtuukin huonosta vaatimusmäärittelystä. Huono vaatimusmäärittely tarkoittaa sitä, että asiakkaan tarpeita ei olla osattu määritellä riittävän tarkasti, tai asiakas ei aina itsekään tiedä, millainen ohjelmisto vastaa hänen omia tarpeitaan. Tällaisissa tapauksissa asiakkaalle toimitetaan ohjelmisto, joka ei vastaa hänen tarpeitaan, tai ohjelmisto ei yksinkertaisesti toimi asiakkaan vaatimalla tavalla. (Browne & Ramesh 2002, s. 625)

Heikko vaatimusmäärittely johtaa usein siihen, että vaatimuksia tulee lisää projektin edetessä. Tällaisissa tilanteissa käy usein niin, että ohjelmisto on jo keretty suunnitella kokonaisuudessaan ennen uusien vaatimusten tullessa. Tällöin käy helposti niin, että ohjelmointi, tai integrointivaiheessa huomataan vaatimusten välillä olevan ristiriitoja. Lisäksi joskus voidaan ohjelmointivaiheessa huomata, että moduulitason suunnittelu on hyvin epäselvä, tai jopa jäänyt tekemättä. (Writer 2010)

Monesti tapauksissa, joissa ohjelmistoprojektin viivästymisen syynä on heikko vaatimusmäärittely. Heikon vaatimusmäärittelyn perimmäinen syy voi olla se, että ohjelmiston toimittajalla ei ole riittävää ymmärrystä asiakkaan liiketoiminnasta ja näin ollen ohjelmiston varsinaisesta tarpeesta. (Browne & Ramesh 2002, s. 625)

Joskus on kuitenkin tilanteita, joissa vaatimusmäärittely tehdään jopa liian tarkasti. Näissä tapauksissa kaikilla on omanlaisensa käsitykset projektista, jolloin niin asiakkailta, kuin myös toimittajilta tulee projektin mittaan erilaisia lisä- ja muutosvaatimuksia. Monesti lisä- ja muutosvaatimukset johtuvat siitä, että asiakkaan liiketoimintaympäristössä tapahtuu jonkinlaisia merkittäviä muutoksia. Tämän vuoksi alkuperäisten vaatimusten mukainen ohjelmisto on asiakkaan varsinaisten tarpeiden kanssa ristiriidassa. Eli siis vaatimusmäärittely ja projektin suunnittelu on hyvä tehdä alussa sellaisella tasolla, että ohjelmiston suunnittelu ja toteuttaminen voidaan aloittaa ja tarvittaessa tehdä muutostyöt järkevällä työmäärällä (Juvonen 2018, s. 59.) Jos lisä- ja muutosvaatimukset tehdään liian myöhään, tulee ohjelmiston muutokset ja lisäykset tehtyä varsin kestämättömillä ratkaisuilla. Kestämättömien ratkaisujen ansiosta ohjelmiston käytön kustannukset voivat kasvaa todella suuriksi, sekä vakavan virheen riski kasvaa. Vakava virhe voi pahimmillaan johtaa fyysisiin loukkaantumisiin, tai jopa kuolemantapauksiin.

3.2 Aikatauluttaminen

Ohjelmistoprojektissa, kuten kaikissa muissakin projekteissa on suunniteltava, kuinka kauan mihinkin projektiin liittyvän tehtävän toteuttamiseen kuluu aikaa. Todella usein esimerkiksi kustannusten minimoimiseksi projektin vaatimiin tehtäviin annetaan tietoisesti vähemmän aikaa, kun tehtävän suorittaminen oikeasti vaatii (McDonnell 2002, s. 29-42). Usein on tilanne se, että projektin muut tehtävät on aloitettava heti, kun edellinen on saatu valmiiksi. Näin ollen monet pienet viivästymiset aiheuttavat dominoefektin, jolloin koko projekti viivästyy merkittävästi. Tilanne voi myös johtaa siihen, että aikataulun kiinnipitämiseksi aikaa otetaan kiinni projektin myöhemmissä vaiheissa, jolloin ohjelmiston laatu heikkenee.

Jos projektiin varataan käyttöön suuria määriä resursseja, kasvavat projektin läpiviennin arvioidut kustannukset. Liian suuret arvioidut kustannukset johtavat kovassa kilpailussa siihen, että asiakas valitsee kilpailevan yrityksen projektin toteuttamiseen. Kuitenkin on muistettava, että projektin aikataulun venyminen maksaa usein huomattavasti enemmän, kuin pieni resurssien lisääminen projektin läpiviemiseen. (Hardee-Vallee 2012)

3.3 Loppumaton projekti

Projektin alkuvaiheessa, kuten esimerkiksi vaatimusmäärittelyssä, tai suunnitteluvaiheessa tehtyjä virheitä ei voida enää korjata projektin loppuvaiheessa palaamatta takaisin projektin alkuvaiheeseen. Joskus projektin alkuvaiheessa tapahtunutta aikataulun venymistä yritetään kompensoida karsimalla projektin sisältöä. Vaikka ohjelmistoprojektin viivästymisen syynä on hyvin usein projektin alkuvaiheessa tapahtuneet virheet, on projekti mahdollista saada epäonnistumaan myös projektin loppuvaiheessa, vaikkakin projektin alkuvaihe onkin onnistunut ilman suurempia epäonnistumisia.

Aina silloin tällöin ohjelmistokehittäjät haluavat parannella, optimoida ja kehittää luomaansa ohjelmistoa loputtomiin. Monesti jos ohjelmiston ylläpitoon ei olla varattu erikseen tarpeeksi rahaa, virheenkorjauksia raportoidaan vielä pitkään varsinaisen ohjelmistokehityksen päättymisen jälkeen. Esimerkiksi seuraavissa tilanteissa ohjelmistoprojekti voi ylittää budjettinsa ja aikataulunsa projektin loppumetreillä:

- Moduulien integrointi aloitetaan liian myöhään
- Testaus aloitetaan liian myöhään
- Virheenkorjauksiin ja uudelleentestaukseen ei olla varattu riittävästi resursseja
- Dokumentointia ei ole otettu tarpeeksi vakavasti (esimerkiksi tarvittavia ohjeita aletaan kirjoittaa aivan liian myöhään)
- Työvaihe unohtuu suunnitella, tai sen tarve ei ole hahmottunut
- Työvoimaa otetaan projektiin mukaan aivan liian myöhään

(Juvonen 2018, s. 101-102)

3.4 Hankkeiden seuranta

Yksi merkittävä syy ohjelmistoprojektien epäonnistumiseen on se, että ohjelmistohankkeiden onnistumista ei edes seurata. Esimerkiksi vuonna 2013 vain noin 28 % yrityksistä seuraa

ohjelmistohankkeidensa onnistumista. Äkkiseltään voisi kuvitella ohjelmistoprojektien kiinnostavan yrityksiä, sillä projektin epäonnistuminen on yritykselle todella kallista, ja voi pahimmillaan johtaa yrityksen kalliisiin oikeusjuttuihin, tai jopa konkurssiin (Järvenpää & Kankare 2013, s. 39-40.) Pelkästään projektien onnistumisia seuraamalla yrityksillä olisi mahdollisuus parantaa toimintaansa. Jos tiedetään, miten paljon, tai miksi projektit epäonnistuvat, voidaan epäonnistumisille keksiä syyt. Kun tiedetään epäonnistumisten syyt, voidaan niistä oppia. Kun ohjelmistoprojektit saadaan onnistumaan entistä paremmin ja useammin, on yrityksen liiketoiminta entistä kannattavampaa.

3.5 Ohjelmistoprojektien riskienhallinta

Ohjelmistoprojekteissa, kuten kaikessa muussakin liiketoiminnassa on olemassa erilaisia riskejä. Jos toimintaan liittyviä riskejä ei osata ottaa huomioon, voi ne toteutuessaan aiheuttaa huomattavia negatiivisia vaikutuksia yrityksen talouteen, tai liiketoimintaan. Suuri osa ohjelmistoprojekteihin liittyvistä riskeistä on varsin helposti määriteltävissä, sekä niihin on jokseenkin helppo varautua. Kuitenkin joitain riskejä ei voida ennustaa. Koska joitakin riskejä on mahdotonta ennustaa, on niihin myös mahdotonta varautua etukäteen.

Yksi merkittävä ohjelmistoprojekteihin liittyvä riski on se, että projektin läpiviennin kannalta merkittävä henkilö joutuu sairaslomalle, tai siirtyy muiden organisaatioiden palvelukseen. Tällaisissa tapauksissa työntekijän mukana lähtee olennainen osa projektin vaatimasta tiedosta, mikä voi jo ykin viivästyttää projektin valmistusta, tai jopa saada projektin suistumaan pois raiteiltaan. Ongelmaa voidaan esimerkiksi yrittää korjata siten, että kaikki tieto on kaikkien projektiin valmistukseen osallistuvien tahojen saatavilla ja tekemällä työnteko yrityksessä mahdollisimman kannattavammaksi. (Writer 2010)

Monesti projektin yksityiskohtaista suunnittelua ei mielletä tarpeeksi tärkeänä tekemisenä, jolloin suunnitteluvaihe pyritään tekemään mahdollisimman nopeasti alta pois (Writer 2010). Syynä siihen, miksi projektin suunnittelu tehdään joskus vähän turhankin karkealla tasolla voisi olla myös se, että ajatellaan ohjelmoinnista vastaavien henkilöiden tietävän, mitä milläkin karkean tason suunnitelmalla tarkoitetaan matalammalla tasolla. Jos projektin suunnitelma

tehdään liian karkealla tasolla, menee arvokasta ohjelmointiin varattavaa aikaa hukkaan. (Writer 2010)

Ohjelmistoprojekti voidaan siis helposti saada epäonnistumaan monesta eri syystä kaikissa projektin vaiheissa. Kuitenkin useimmiten projekti saadaan onnistumaan, kunhan vain tehdään riittävän huolellinen vaatimusmäärittely ja perusteellinen, sekä realistinen projektisuunnitelma. Lisäksi on hyvä tiedostaa, että kaikki sidosryhmät, kuten esimerkiksi omat työntekijät ja asiakkaat pidetään tyytyväisinä. Tämä tarkoittaa monesti sitä, että pidetään hinta riittävän alhaisena, maksetaan työntekijöille riittävää palkkaa, sekä tehdään realistiset projektisuunnitelmat.

4 TYÖMÄÄRÄN ARVIOINNIN MENETELMIÄ

Ohjelmistoprojektin vaatima työmäärä on lähes poikkeuksetta hyvin hankalaa ennustaa. Kuitenkin ajan mittaan on laadittu useita erilaisia menetelmiä, joiden on tarkoitus tuoda helpotusta ongelmaan. Kehitetyt mallit lähestyvät ongelman ratkaisua kolmesta näkökulmasta. Jotkut kehitetyt mallit, kuten valistunut arvaus ja kolmen pisteen malli (3- point model) perustuvat enemmän, tai vähemmän aiempaan kokemukseen vastaavan kaltaisista projekteista. Toiset mallit, kuten COCOMO (Constructive Cost model) ja toimintopisteanalyysi (Function Point Analysis) perustuvat ohjelmakoodin pituuteen koodiriveissä. Jotkut mallit, kuten suunnittelupokeri (Planning Poker) perustuvat projektitiimin jäsenten väliseen avoimeen keskusteluun. MacDonell:in ja Shepperd:in (2003) mukaan monesti työmäärän arvioimiseen käytetään vain yhtä keinoa. Koska millään keinolla ei saada riittävän tarkkaa arviota, tulee arviointiin käyttää ainakin kahta eri mallia, ja tehdä tarvittavat johtopäätökset niiden perusteella (MacDollenn & Shepperd 2003, s. 92)

4.1 Valistunut arvaus

Ohjelmistoprojektin vaatiman työmäärän arviointiin on vuosien saatossa keksitty useita varsin formaaleja menetelmiä. Kuitenkin hyvin usein käytetään niin sanottua valistuneen arvauksen menetelmää. Valistuneen arvauksen menetelmässä työmäärän arvioiminen perustuu aikaisempaan kokemukseen vastaavan kaltaisista projekteista. Tämä tarkoittaa sitä, että kun on aiempaa kokemusta tietyntyyppisistä projekteista, oletetaan vastaavanlaisen projektin vaativan suunnilleen saman työmäärän. Menetelmää käytettäessä arvioinnissa tulisi olla mukana vähintään kolme arvioijaa, jotka keskustelemalla pyrkivät pääsemään yhteisymmärrykseen projektin vaatimasta työmäärästä. Usein ongelmana valistuneessa arvauksessa on tarvittavan historiatiedon puuttuminen ja, että työn alla olevia projekteja vertaillaan vääriin aiempiin projekteihin. (Haikala & Mikkonen 2011, s. 161-162)

Valistuneen arvauksen yksi merkittävimmistä ongelmista on se, että valistunutta arvausta tekevät ne henkilöt, jotka eivät juuri ymmärrä asiasta, tai ne, jotka asiasta ymmärtävät eivät saa ääntään kuuluviin (Ahonen & Savolainen 2010, s. 2185). Vaikka asiasta ymmärtävät henkilöt

pääsisivätkin ääneen, on usein tilanne se, että resursseja annetaan projektin läpivientiin liian vähän, monesti kustannussyistä. Esimerkiksi työmääräarviota voidaan kysyä ohjelmoinnista vastaavilta henkilöiltä. Kuitenkin aikaa ja resursseja annetaan käyttöön ohjelmoinnista vastuussa olevan henkilön arvioita vähemmän (McDonnell 2002, s. 29-42.) Kannusteena tälle on esimerkiksi se, että yritys menestyy tarjouskilpailussa ja saa projektin itselleen hoidettavaksi. Tällainen johtaa väkisinkin siihen, että projekti ei valmistu vaaditussa ajassa, tai valmistunut ohjelmisto ei täytä vaatimuksia. Tällainen kustannusten välttely voi myös johtaa siihen, että ohjelmoinnista vastuussa olevat henkilöt kertovat tahallaan liian suuren työmääräarvion. Motiivina työmäärän liioitteluun on esimerkiksi se, että resursseja saadaan kuitenkin vaadittua vähemmän, jolloin aikataulu venyy, tai laatu heikkenee. Pitkässä juoksussa tällainen vääristely johtaa yrityksen sisäiseen luottamuspulaan, jolloin projektien läpivientiin ei saada enää koskaan sopivaa määrää aikaa, tai resursseja.

4.2 Constructive Cost Model

COCOMOn, ideana on määrittää ohjelmiston valmistuksen tarvittava työmäärä arvioimalla ensin varsimaisen ohjelmakoodin pituus, jonka perusteella lasketaan projektin vaatima työmäärä kuukausissa mitattuna. Työkuukaudet lasketaan kaavalla 1.

$$MM = 2{,}4(KDSI)^{1{,}05} \quad (1)$$

Kaavassa MM kuvaa työkuukausia ja KDSI kuvaa arvioitua ohjelmakoodin pituutta tuhansissa riveissä. Kun kuukaudet on saatu laskettua, lasketaan seuraavaksi ohjelmiston valmistukseen kuluva aika kalenterikuukausina kaavalla 2.

$$TDEV = 2{,}5MM^{0{,}38} \quad (2)$$

COCOMOa käytettäessä malli ei itsessään ota huomioon minkäänlaisia rajoituksia, kuten esimerkiksi työvoiman kokemusta, projektin haastavuutta, tai käytettävissä olevan laitteiston tuomia rajoituksia. Rajoituksia on pyritty ottamaan huomioon laskennassa kertomalla työkuukaudet eri suuruisilla kertoimilla. (Boehm 1981, s. 57 - 61)

Vaikka erilaisia rajoituksia voidaankin ottaa huomioon käyttämällä erilaisia kertoimia, on malli alan ammattilaisten mielestä varsin järjetön malli, sillä sen perustana on yksinomaan ohjelmakoodin pituus (Juvonen 2018, s. 75). Trendowicz:in ja Ross:in (2014) mukaan tähän syynä voisi olla esimerkiksi se, että samat asiat toteuttava ohjelma voidaan kirjoittaa hyvinkin eripituisilla koodeilla. Lisäksi yksinkertaisella toimintalogiikalla toimiva pitkäkin ohjelmakoodi voidaan kirjoittaa hyvinkin nopeasti, kun taas monimutkaisella toimintalogiikalla lyhyt toimiva ohjelmakoodi on huomattavasti hitaampaa kirjoittaa. Myös kaikenlainen koodin uudelleenkäyttö, valmiiden koodikirjastojen, tai avoimen lähdekoodin käyttö nopeuttaa ohjelmiston valmistumista, vaikka koodin pituus riveissä mitattuna voisi olla verrattain pitkä. (Trendowicz & Ross 2014, s. 291-292)

4.3 Toimintopisteanalyysi

Toisin, kuin COCOMO, toimintopisteanalyysissä työmäärää arvioidaan koodirivien sijaan ohjelmiston toiminnallisuuden perusteella. Toimintopisteanalyysissä lasketaan toimintopisteiden lukumäärä projektissa ja työmäärä arvioidaan toimintopisteiden määrän perusteella. Toimintopisteiden määrää kasvattavat esimerkiksi ohjelmiston käyttämien tiedostojen, tulosteiden ja liiketoimintasääntöjen määrä. Kun tiedetään toimintopisteiden lukumäärä ja kuinka monta toimintopistettä tiimi pystyy tekemään esimerkiksi kuukaudessa, voidaan arvioida projektin vaatima työmäärä kalenterikuukausissa mitattuna. (Haikala & Mikkonen 2011, s. 162)

Toimintopisteanalyysi voidaan myös yhdistää COCOMOn kanssa. Kun toimintopisteet on saatu laskettua toimintopisteanalyysiä käyttäen, muutetaan pisteet koodiriveiksi käyttäen muutostaulukkoa (taulukko 1). Koodiriveistä voidaan laskea COCOMO- mallia käyttäen työmääräarviot kalenterikuukausissa. (Haikala & Mikkonen 2011, s. 162)

Taulukko 1. Esimerkki muutostaulukosta

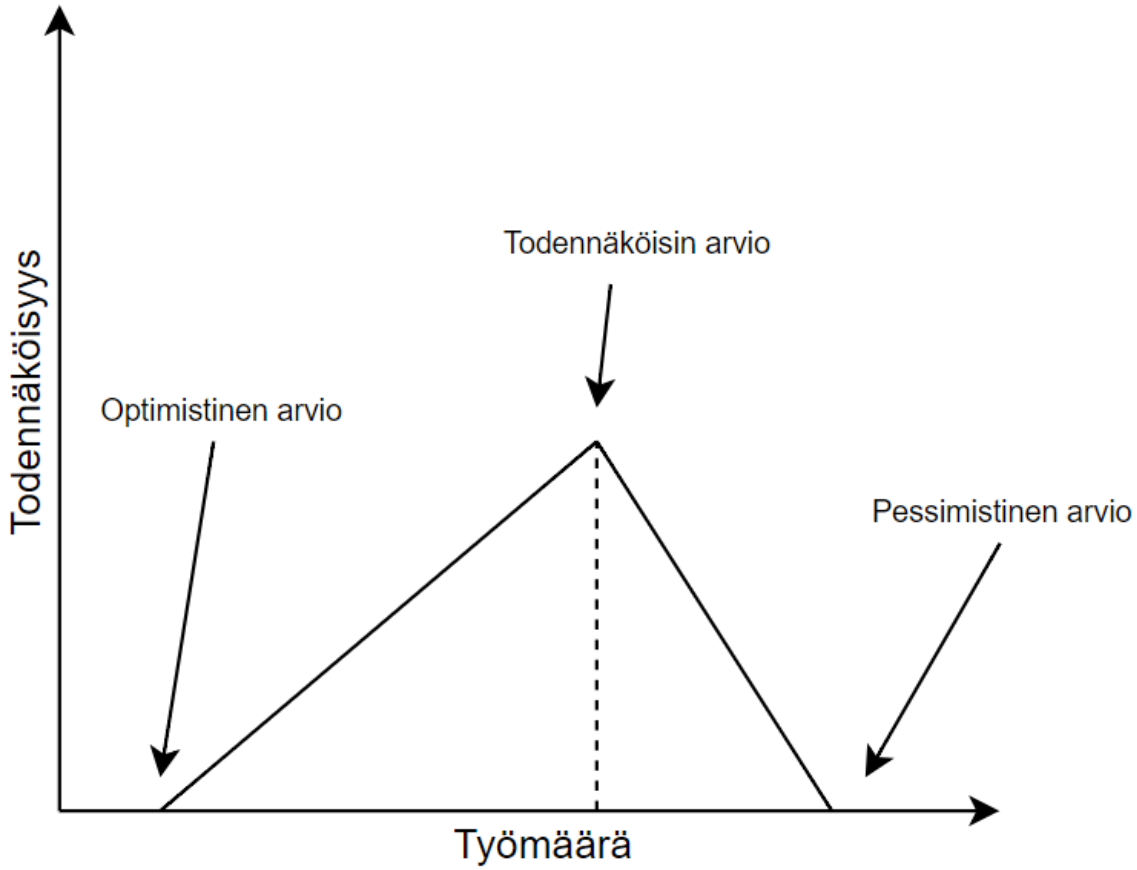
Toimintopisteet	**Koodirivit**
7	500
14	1000
21	1500
28	2000
35	2500
42	3000
49	3500

Toimintopisteanalyysin hyödyntämisen vaikeutena on se, että toiminnon toteuttamiseen vaadittava työmäärä riippuu hyvin pitkälti käytössä olevasta toteutusympäristöstä. Tämä tarkoittaa esimerkiksi sitä, että käyttämällä tehokkaita ohjelmointiympäristöjä, valmiita koodikirjastoja ja uudelleenkäytettävää koodia työmäärä pienenee huomattavasti siitä, kun kaikki koodi kirjoitettaisiin itse alusta lähtien. Myös käyttämällä oikeita ohjelmointikieliä voidaan työmäärän tarvetta joissain tapauksissa pienentää (Juvonen 2018, s. 75.) Kuitenkin se, kuinka paljon valmiita koodikirjastoja, tai koodia voidaan käyttää uudelleen, on kuitenkin varsin helposti määriteltävissä, eikä se sinällään estä toimintopistemallin käyttämistä.

4.4 Kolmen pisteen arviointimalli

Kolmen pisteen arviointimallissa pyritään määrittämään alhaisin mahdollinen järkevä ohjelmiston valmistukseen tarvittava työmäärä (optimistinen arvio), korkein mahdollinen järkevä työmäärä (pessimistinen arvio), sekä kaikkein todennäköisin työmäärä (todennäköisin työmäärä). Lisäksi tulee arvioida todennäköisimmän työmääräarvion todennäköisyys. Koska mallia käytettäessä ajatellaan optimistisen ja pessimistisen arvion todennäköisyyden olevan joko nolla, tai sitä hyvin lähellä, ei niille tarvitse erikseen määrittää todennäköisyyttä. Työmääräarviot voidaan tehdä niin koodiriveissä, kuin myös työtunneissa, tai -kuukausissa. Kun kyseiset työmääräarviot on saatu määritettyä, piirretään mallista kuva, jossa vaaka-akselilla on työmäärä ja pystyakselilla on todennäköisyys (kuva 6). Kuvasta huomataan,

lopputuloksena saadaan kolmion muotoinen kuvaaja. Jos oletetaan, että kolmion kulmapisteet on laadittu oikein, sijaitsee projektiin tarvittava työmäärä jossain muodostuneen kolmion rajaaman alueen sisällä. (Ruhe & Wohlin 2014, s. 53)



Kuva 6. Hahmotelma kolmen pisteen mallin kuvaajasta

Joskus ollaan tilanteessa, jossa mallista saatu kuvaaja ei ole kallellaan kummallekaan puolelle. Tällaisissa tapauksissa todellinen projektin vaatima työmäärä on verrattain lähellä todennäköisimmän työmäärän kohdalla olevaa arvoa. Kuitenkin projektissa tulee varautua siihen, että projekti syystä, tai toisesta vaatii enemmän, tai vähemmän työtä, kun kuvaaja antaa ymmärtää.

Monesti kuitenkin kolmen pisteen mallista saatava kuvaaja on kallellaan jommallekummalle puolelle. Tällöin todellinen työmäärä voi hyvinkin olla ”painottomalla” puolella, jolle kolmio ei ole kallellaan.

Kolmen pisteen arviointimallia voidaan myös soveltaa siten, että kuvaajan piirtämisen sijaan lasketaan työmäärän tarve kaavalla 3.

$$\frac{Pessimistinen\ arvio+4*Todennäköisin\ arvio+Optimistinen\ arvio}{6} \quad 3$$

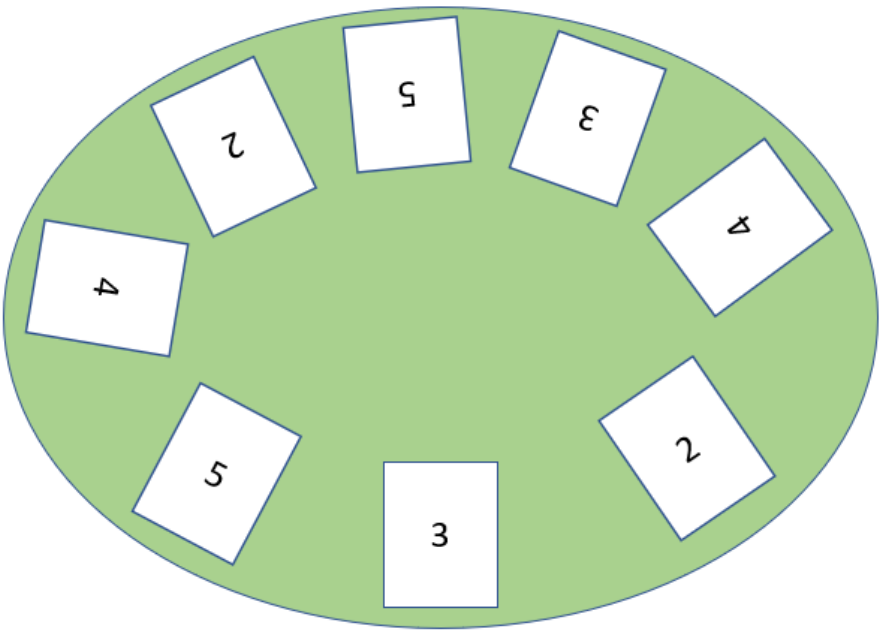
Kaavaa lähemmin tarkastelemalla huomataan, että kyseessä on vain keskiarvon laskemista. Kaavassa pessimistinen arvio arvioidaan olevan yhden yksikön arvoinen arvio, todennäköisin arvio arvioidaan olevan suurin, neljän yksikön arvoinen arvio, kun taas optimistinen arvio arvioidaan olevan pessimistisen arvion tapaan yhden yksikön arvoinen arvio. Kun arviot lasketaan yhteen, saadaan aikaiseksi kuusi yksikköä. Jotta saataisiin aikaan keskiarvo, jaetaan summa kuudella. Näin saadaan laskettua ohjelmiston valmistukseen vaadittava työmäärä. (Haikala & Märijärvi 2004)

Vaikka kolmen pisteen arvio antaa varsin tarkan tuloksen projektin vaatimasta työmäärästä, on kuitenkin aina muistettava, että mallin käyttäminen perustuu puhtaaseen arviointiin, eikä näin ollen mallista saatava työmääräarvio ole mitenkään eksakti. Näin ollen mallista saatavaan työmääräarvioon tulee suhtautua tietyllä lailla skeptisesti, eli tulee varautua siihen, että projekti vaatii joko enemmän, tai vähemmän työtä, kuin mitä mallista saatava tulos antaa ymmärtää.

4.5 Suunnittelupokeri

Suunnittelupokeri on varsin usein käytetty menetelmä työmäärän arvioimisessa erityisesti ketteriä menetelmiä käytettäessä. Suunnittelupokerissa jokaiselle ryhmän jäsenelle annetaan korttipakka, jossa on tyypillisesti Fibonaccin lukujonon lukuja (1,2,3,5,8,13,21,34,55). Lukujonon luvut edustavat työmääräarvioita halutussa mittayksikössä, kuten esimerkiksi työpäivissä, työtunneissa, tai jopa koodiriveissä. Kun ryhmälle esitetään sille annettava tehtävä, asettaa jokainen ryhmän jäsen sellaisen kortin eteensä, joka hänen mielestään kuvaa parhaiten tehtävän suorittamisen vaatimaa työmäärää (kuva 7). Jos jotkut ryhmän jäsenet ovat toistensa kanssa eri mieltä tehtävän vaatimasta työmäärästä, keskustellaan arvioiden erojen syistä. Perusteellisen keskustelun jälkeen toistetaan äänestys. Tätä jatketaan niin pitkään, kunnes

kaikki ryhmän jäsenet ovat asettaneet samansuuruisen luvun eteensä, tai kunnes työmäärästä päästään yhteisymmärrykseen muilla keinoin. (Grenning 2002)



Kuva 7. Kuva suunnittelupokerista

Suunnittelupokerin hyviä puolia on esimerkiksi se, että kaikki tiimin jäsenet joutuvat tuomaan omat näkemyksensä kaikkien tietoisuuteen. Lisäksi menetelmää käytettäessä kaikkien ryhmän jäsenten arviot ovat yhtä tärkeitä, eikä lopputulokseen päästä ennen, kuin kaikkien näkemykset on otettu huomioon ja asiasta on päästy yhteisymmärrykseen. Myös ryhmän työmääräarvio saadaan usein varsin nopeasti kasaan ilman suurempia erimielisyyksiä. Kuitenkin menetelmän haittapuolena mainittakoon, että ryhmä koostuu usein vain ohjelmistosuunnittelijoista, tai ohjelmistotestaajista, jolloin ryhmän työmääräarvio on puhtaasti tekninen. Tällöin projektin liiketaloudelliset ja sosiaaliset tekijät tulevat usein kokonaan unohdettua. (Juvonen 2018, s. 79-83)

4.6 Sosiaalinen päätös

Sosiaalisen päätöksen keinossa (Social choise) ohjelmiston valmistuksesta vastaavat henkilöt valitsevat heidän mielestään parhaat arviot projektiin vaadittavasta työmäärästä. Tämän jälkeen he äänestävät antamalla annetuista työmääräarvioista omat mielipiteensä laittamalla annetut työmääräarviot paremmuusjärjestykseen. Esimerkiksi annetuista työmääräarvioista a, b ja c joku tiimin jäsen voisi äänestää muodossa a>c>b. Tämä tarkoittaa sitä, että henkilön mielestä vaihtoehto a on järkevin työmäärän ennuste, seuraavaksi b ja vähiten järkevänä hän pitää arviota c. Kun kaikki tiimin jäsenet ovat äänestäneet, voidaan määrittää, mikä arvio on tiimin mielestä kaikkein järkevin vaihtoehto. Lisäksi voidaan myös päätellä, mihin suuntaan suosituimmasta arviosta projektin todellinen työmäärä voisi asettua. (Koch & Mitlöhner 2009, s. 896)

5 JOHTOPÄÄTÖKSET

5.1 Keskeiset havainnot

Ohjelmistoprojektit tapaavat epäonnistua todella usein, joskus jopa karmein seurauksin. Erityisesti vesiputousmallia käytettäessä projekti ei lähes koskaan valmistu etuaikaisesti. Kuitenkin joissain tapauksissa vesiputousmallia käytettäessä projekti saattaa valmistua aikataulussa budjettia ylittämättä, tai jopa asiakkaan vaatimukset täyttäen.

Sille, miksi ohjelmistoprojektit tapaavat epäonnistua, on olemassa useita syitä. Syinä voi olla vaatimusmäärittelyn huono toteutus, aikataulutuksen ongelmat, projektin seuraamisen laiminlyönti, henkilöstöön liittyvät ongelmat, tai liiketoiminnalliset syyt.

Todella usein ohjelmistoprojektin epäonnistumisen syynä on se, että projektin läpivientiin vaadittavaa työmäärää ei osata tai haluta ennustaa tarpeeksi tarkasti. Varmaankin yksi keskeisimmistä ongelman syistä on se, että ongelmaan ei olla keksitty oikeasti käyttökelpoisia menetelmiä. Monet ongelman ratkaisemiseksi tehdyt menetelmät perustuvat ohjelmakoodin pituuteen. Ongelmia tässä mallissa on runsaasti. Ohjelmakoodin pituus voi vaihdella hyvinkin paljon esimerkiksi sen perusteella, kuka koodin on kirjoittanut. Lisäksi ongelmana näissä malleissa on se, että eri toimintalogiikoilla toimivan koodin kirjoittaminen kuluttaa eri määrän resursseja. Yksinkertainen koodi, kuten esimerkiksi valikko, on hyvin nopeaa kirjoittaa, vaikka koodi voi olla verrattain pitkä. Vastaavasti monimutkaisella logiikalla toimiva koodi voi olla huomattavasti hitaampaa kirjoittaa, vaikka koodin pituus voi olla hyvinkin lyhyt. Vaikkakin malleissa pyritään ottamaan huomioon kyseiset asiat huomioon esimerkiksi käyttämällä erisuuruisia kertoimia, on arviointi siltikin hyvin epätarkkaa, sillä pienikin kertoimen muuttaminen voi muuttaa työmääräarviota paljonkin suuntaan, jos toiseen.

On myös kehitetty malleja, joissa koodirivien määrä ei ole pääasiallinen syöte. Koodirivien määrän sijaan käytetään yleensä historiatietoa vastaavan kaltaisista aiemmista projekteista. Monesti kuitenkin ongelmana näissä tilanteissa on se, että projektia verrataan sellaisiin

projekteihin, joihin ei aloitettavaa projektia missään nimessä pitäisi verrata, koska valmisteella oleva projekti on ollut niinkin paljon erilainen yrityksen historiatietoon verrattuna.

Koska vielä ei ole keksitty sellaista menetelmää työmäärän arviointiin, jota yritykset voisivat oikeasti soveltaa omissa projekteissaan, ovat yritykset joutuneet turvautumaan ns. valistuneen arvauksen menetelmään. Valistunutta arvausta olisi hyvä olla tekemässä useampi arvioija, mutta monesti kuitenkin valistunutta arvausta on tekemässä entistä vähemmän arvioijia, vieläpä niin, että arvioijien pääasiallinen tarkoitus työmäärän arvioinnissa ei ole riittävän laadun takaaminen, vaan projektiin käytettävien kustannusten minimointi.

Nykyisin vallalla olevan trendin mukaan liiketoimintajohto kokee ymmärtävänsä ohjelmistoalaa entistä paremmin, kun taas vastaavasti tietohallinto kokee, että heidän ymmärryksensä omaa toimintaansa kohtaan on huonontunut merkittävästi (Järvenpää & Kankare 2013, s. 40). Monesti kuitenkin liiketoiminnan johto ei ymmärrä ohjelmiston teknisestä toteutuksesta juuri mitään. Vastaavasti teknisestä toteutuksesta vastuussa olevat henkilöt eivät ymmärrä liiketaloudesta juuri mitään (Wallin et al. 2002, s. 29). Tämä on ainakin osaltaan johtanut siihen, että vaikka liiketoimintajohto pyrkiikin ottamaan ohjelmiston toteutuksesta vastuussa olevilta henkilöiltä heidän mielipidettään projektiin läpivientiin vaadittavasta työmäärästä, antavat he kuitenkin aikaa ja resursseja liian vähän käyttöön. Tästä on seurannut se, että ohjelmiston toteutuksesta vastuussa olevat henkilöt kertovat tietoisesti liian suuren arvion työmäärästä, jolloin taas liiketoimintajohto pyrkii antamaan taas vähemmän aikaa ja resursseja käyttöön. Tämä voi johtaa pitkässä juoksussa siihen, että ajan mittaan yrityksessä vallitsee suuri luottamuspula, jolloin työmääräarviointi ei voi mitenkään mennä oikein. Näin ollen ohjelmistoprojektit epäonnistuvat yhä useammin (mikäli se edes on mahdollista), ja tuo yritykselle lisää ylimääräisiä kustannuksia, mikä taas suoraan yrityksen tuloksessa ja kannattavuudessa.

5.2 Pohdinta

Ongelmaa voitaisiin tulevaisuudessa ratkaista esimerkiksi kehittämällä sellaisia malleja työmäärän arviointiin, jotka ottavat huomioon niin koodirivien ja historiatiedon lisäksi niin,

henkilöstön määrän ja osaamisen, valmisteella olevan projektin haastavuuden, käytettävän laitteiston tuomat rajoitteet ja edut jne. nykyistä huomattavasti järkevämmällä tavalla oikeassa mittasuhteessa. Suurissa projekteissa työmäärän arvioiminen on huomattavasti hankalampaa, kuin pienissä projekteissa. Näin ollen yksi ratkaisu ongelmaan voisi olla ketterien menetelmien käyttö, sillä ketterissä menetelmissä ohjelmistoa suunnitellaan pieni pala kerrallaan, jolloin yhden pyrähdyksen vaatiman työmäärän arviointi on jokseenkin helppoa. Ketteriin menetelmiin on myös tehty paremmin toimivia malleja (esimerkiksi suunnittelupokeri) työmäärän arviointiin, kuin vesiputousmalliin. Lisäksi jos ongelma pystyttäisiin ratkaisemaan jotenkin koneellisesti, poistuu ongelmasta inhimilliset tekijät, kuten erilaiset kannustimet, tai ihmisen tekemät ajatteluvirheet.

Uusien ja entistä parempien arviointimallien lisäksi ongelmaa voidaan yrittää ratkaista esimerkiksi siten, että työmäärää ovat arvioimassa oikeat henkilöt. Oikeat henkilöt ovat sellaisia henkilöitä, joilla on riittävästi ymmärrystä niin liiketoiminnan johtamisesta, kuin myös ohjelmistojen teknisestä toteutuksesta. Tällä tavoin voitaisiin välttyä tilanteilta, joissa liiketoiminnan johdolla on tietyt vaatimukset ohjelmiston toteutukseen käytettävien resurssien määrästä kannattavan liiketoiminnan takaamiseksi, ja ohjelmiston teknisestä toteutuksesta vastaavilla henkilöillä on oma käsityksensä resurssien tarpeesta valmiin ohjelmiston tarvittavan laadun turvaamiseksi.

6 LÄHTEET

Ahonen. J. J., Savolainen, P. Software porojects may fail before they are started: Post modern analysis of five cancelled projects. 2010. The Journal of Systems and Software. Vol. 83, Nro. 11. Sivut 2175-2187.

Armstrong, K. 2018. Do I Need Software? Why and When it's Time to Invest in Business Software. [WWW-artikkeli]. [viitattu 4.4.2019]. Saatavissa: https://www.activecampaign.com/blog/do-i-need-software-why-and-when-its-time-to-invest-in-business-software/

Bannik, S. 2014. Challenges in the Transition from Waterfall to Scrum – a Casestudy at Portbase. University of Twente. [WWW-artikkeli]. [viitattu 6.2.2019]. Saatavissa: https://pdfs.semanticscholar.org/62c6/d552e6a4aaa3c8f7a1baf65070671d038d53.pdf

Browne, G. J., Ramesh, V. 2002. Improving requirements determination: a cognitive perspective. Information Management. Vol. 39, Nro. 8. Sivut 625-645.

Castillo, F. 2016. Managing Information Technology. Springer International Publishing.

Colucci, L. Calculatin Cost of Delay for software projects. Plataformatec. [WWW-artikkeli]. [viitattu 26.3.2019]. Saatavissa: http://blog.plataformatec.com.br/2016/11/calculating-cost-of-delay-for-software-projects/

Graham, N., Protny, S. 2013. Project management for dummies. Chichester: John Wiley & Sons 2013. 2. Painos.

Grenning, J. 2002. Planning Poker or How to avoid analysis paralysis while release planning. [WWW-artikkeli]. [viitattu 22.3.2019]. Saatavissa: http://www.renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf

Haikala, I., Mikkonen, T. 2011 Ohjelmistotuotannon käytännöt. Helsinki: Talentum. 12. Painos.

Haikala, I,. Märijärvi, I. 2004. Ohjelmistotuotanto. Helsinki: Talentum. 10. Painos.

Hardy-Vallee, B. 2012. The Cost of Bad Project Management. Business Journal. [WWW-artikkeli]. [viitattu 7.4.2019]. Saatavissa: https://news.gallup.com/businessjournal/152429/cost-bad-project-management.aspx

Juvonen, R. 2018. Ohjelmistoprojektin sudenkuopat ja miten ne vältetään. Helsinki: Books on Demand.

Järvenpää, T., Kankare, I. 2013. Veikö Moolok vallan? Vapauta projektisi tuhlaajakuluista. Talentum Media Oy.

Koch, S., Mitlöhner, J. 2009. Software project effort estimation with voting rules. Decision Support Systems. Vol. 46, Nro. 4. Sivut 895-901.

Kuhrmann, M., Ternité, T., Friedrich, J., Rausch, A., Broy, M. 2016. Flexible software process lines in practice: A metamodel-based approach to effectively construct and manage families of software process models. Journal of Systems and Software. Vol. 121. Sivut 49-71.

Kukhnavets, P. 2018. How Do Gantt Charts Make Project Managers' Life Easier?. Hygger [WWW-artikkeli]. [viitattu 21.4.2019]. Saatavissa: https://hygger.io/blog/what-is-a-gantt-chart-in-project-management/

McConnell. 2002. Ohjelmistotuotannon hallinta. Helsinki: Edita.

MacDonnell, S. G., Shepperd, M. J. Combining techniques to optimize effort prodictions in software project management. Journal of Systems and Software. Vol. 66, Nro. 2. Sivut 91-98

Ruhe, G., Wohlin, C. 2014. Software Project Management in a Changing World. SpringerLink Berlin Heidenberg.

Storrs, G., Windsor, P. 1992. Prototyping user interfaces.

Schwaber, K. 2004. Agile Project Management with Scrum. SpringerLink Books.

Trendowicz, A., Ross, J. 2014. Software Project Effort Estimation : Foundations and Best Practice Guidelines for Success. Cham : Springer International Publishing 2014.

Wallin, C., Ekdahl, F., Larsson, S. 2002. Integrating Business and Software Development Models. IEEE Software. Vol. 9, Nro 6. Sivut 28-33.

Writer, S. 2010. Top 10 Software Development Risks. ITProPortal. [WWW-artikkeli]. [Viitattu 29.3.2019]. Saatavissa: https://www.itproportal.com/2010/06/14/top-ten-software-development-risk