TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Kalle Kaarne

Opinnäytetyö

Työmääräarvioinnin ja aikatauluttamisen merkitys ohjelmistoprojektissa

Työn ohjaaja Lehtori Jari Mikkolainen
Työn teettäjä Sasken Finland Oy, valvojana Jaakko Kuivanen
Tampere 5/2009

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka

Tekijä	Kalle Kaarne
Työn nimi	Työmääräarvioinnin ja aikatauluttamisen merkitys ohjelmistoprojektissa
Sivumäärä	35
Valmistumisaika	29.5.2009
Työn ohjaaja	Lehtori Jari Mikkolainen
Työn tilaaja	Sasken Finland Oy, valvojana Jaakko Kuivanen

TIIVISTELMÄ

Ohjelmistoprojektit ylittävät joskus aikataulutavoitteensa ennen valmistumistaan. Ratkaisevat virheet tehdään usein jo projektin alkuvaiheessa, työmääräarvioinnissa ja aikataulutuksessa. Tässä työssä paneudutaan näihin ohjelmistotuotannon osa-alueisiin. Lisäksi analysoidaan ohjelmistoprojektia, jossa työn kirjoittaja työskenteli kehittäjänä keväästä 2008 kevääseen 2009.

Työn alkuosassa kerrotaan yleisesti ohjelmistotuotannosta, työmääräarvioinnista ja aikataulutuksesta. Keskiosassa esitellään esimerkkiprojekti ja lopussa analysoidaan projektin kulku ja mietitään, mitä olisi voinut tehdä paremmin.

Avainsanat ohjelmistokehitys, työmääräarviointi, aikataulutus

TAMK UNIVERSITY OF APPLIED SCIENCES
Information Technology
Software Engineering

Writer	Kalle Kaarne
Thesis	The significance of estimating and scheduling in a software project
Pages	35
Graduation time	29.5.2009
Thesis supervisor	Jari Mikkolainen (lecturer)
Co-operating company	Sasken Finland Oy, supervisor Jaakko Kuivanen

---

ABSTRACT

Software projects sometimes exceed their original deadlines. The critical mistakes are usually made in the early phase of a project: workload estimation and scheduling. The purpose of this engineering thesis is to concentrate into these areas of software production. The writer of this thesis worked as a developer in a software project from spring 2008 to spring 2009. The problems that occurred during this project are analyzed and reconstructive suggestions are given.

At first this document goes through the fundamentals of software production, workload estimation and scheduling. The example project is introduced in the middle part, and analysis is found in the last part of this thesis.

Keywords software production, estimation, scheduling

Sisällysluettelo

Lyhenteet ja termit

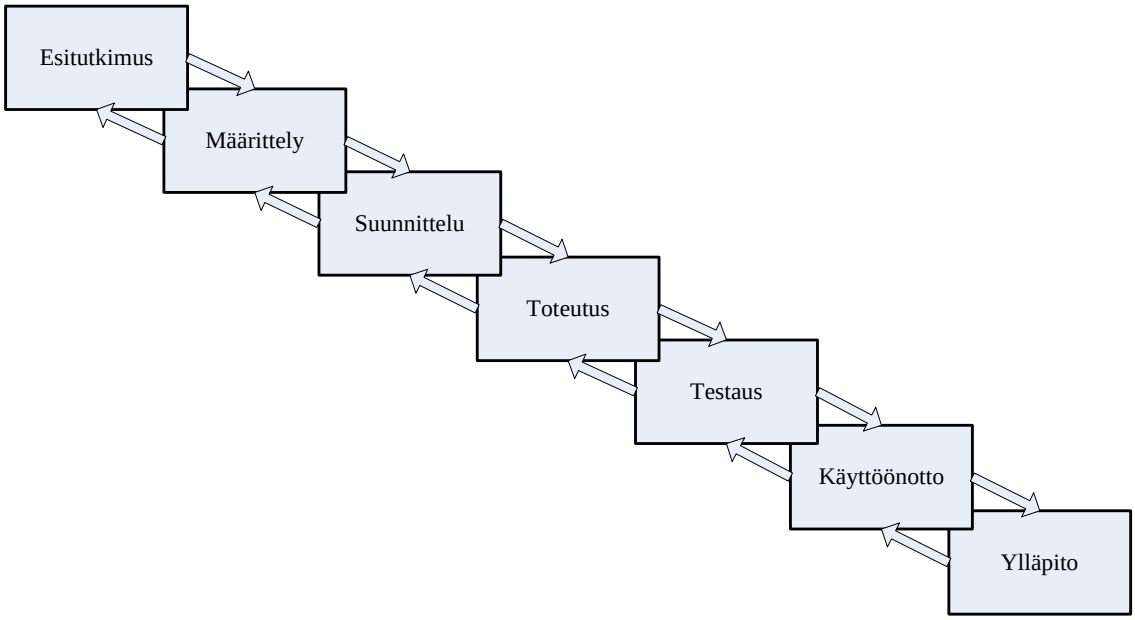
HTTP	HyperText Transfer Protocol, tiedonsiirtoprotokolla.
Iterointi	Toistaminen, toistetaan prosessia siten, että kunkin kierroksen tulos on seuraavan lähtökohta.
Käyttöliittymä	Ohjelmiston osa, jonka kautta käyttäjä käyttää ohjelmistoa.
PERT	Program Evaluation and Review Technique, tekniikka työmääräarvioinnin tekemiseen.
S60	Matkapuhelimissa käytettävä, Nokian kehittämä käyttöliittymä ja sovellusalusta.
Scrum	Yksi ketteristä ohjelmistokehityksen menetelmistä.
Symbian	Käyttöjärjestelmä, jota käytetään osassa matkapuhelimista.

1 Johdanto

Aloitin keväällä 2008 työskentelyn Symbian S60 -projektissa Sasken Finland Oy:ssä. Alusta asti oli selvää, että tulisin tekemään myös opinnäytetyöni työnantajalleni, mieluiten liittyen omaan projektiini. Projekti oli haastava ja aikaa vievä, ja kuten ohjelmistoprojektit usein, ei täysin ongelmaton. Kun projekti läheni loppuaan myöhästyttyään suunnitellusta valmistumisajasta useita kuukausia, alkoi opinnäytetyöni aihe muotoutua. Päätin koota samojen kansien sisään analyysin siitä, mitä ongelmia projektin aikana ilmeni, sekä ehdotuksia siitä, miten samankaltaisia ongelmia kyettäisiin tulevaisuudessa välttämään. Lisäksi kokosin omien kokemusteni sekä alan kirjallisuuden avulla tiiviin teoriapaketin ohjelmistoprojektien työmääräarvioinnista ja aikatauluttamisesta.

2 Ohjelmistotuotanto pähkinänkuoressa

Uusia ohjelmistoja kehitetään useimmiten ohjelmistoprojekteissa. Projektit on jaettu useisiin osaprojekteihin. Perinteinen tapa kuvata ohjelmistoprojektia kokonaisuutena on vesiputousmalli (kuvio 1). Siinä edellinen vaihe tuottaa aina dokumentin tai tuotteen, jonka perusteella voidaan toteuttaa seuraava vaihe. Vaikka jokainen ohjelmistoprojekti sisältääkin kaikki vesiputousmallin vaiheet, eivät ne kuitenkaan aina toimi vesiputousmallin mukaisesti yhteen suuntaan. Vesiputouksen alemmista vaiheista joudutaan joskus palaamaan myös taaksepäin. Esimerkiksi jo toteutusvaiheessa olevassa projektissa joudutaan palaamaan suunnittelupöydän ääreen, jos suunnittelussa huomataan puutteita. Usein projektit toteutetaan sarjana toistuvia vesiputouksia, joissa jokaisen vesiputouksen tuloksena on aina uusilla ominaisuuksilla kasvatettu ohjelmisto. Näitä osatuotoksia voidaan esimerkiksi julkaista asiakkaalle testattavaksi. /3/

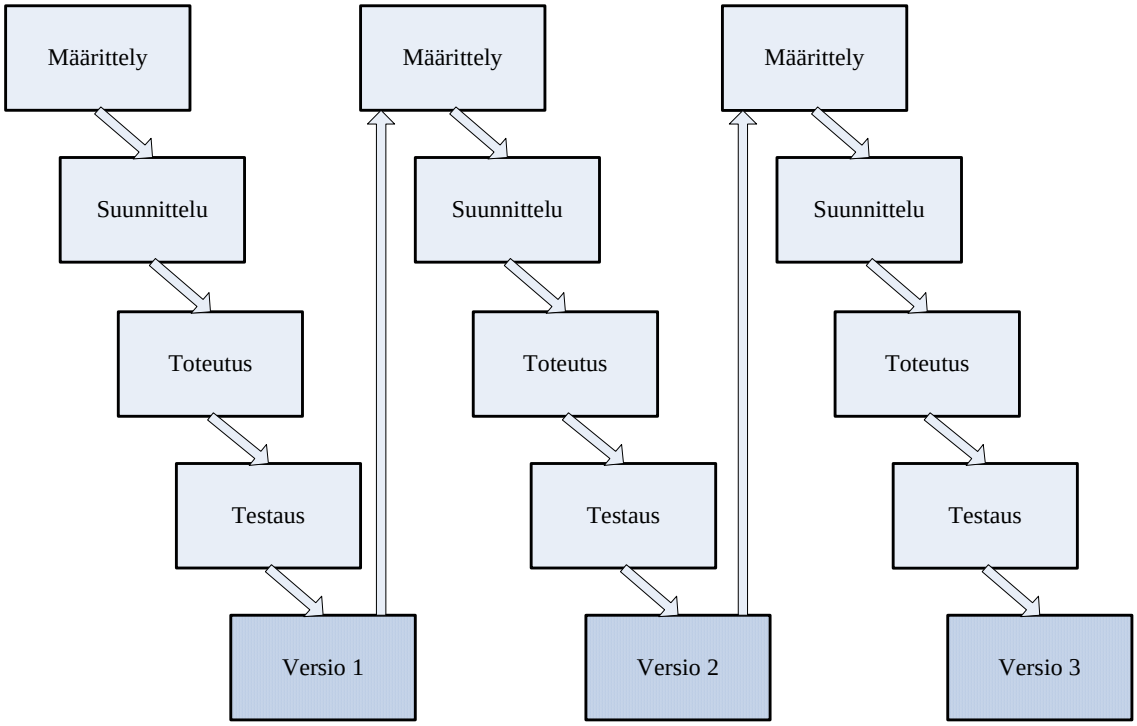


Kuvio 1. Vesiputousmalli /3/

Ohjelmistoprojektien pituus voi vaihdella muutamista kuukausista useisiin vuosiin. Kaikki ohjelmistoprojektit on päävaiheiden lisäksi syytä pilkkoa pienempiin tehtäviin, ja arvioida jokaisen tällaisen tehtävän työmäärä erikseen. Yleisesti voidaan sanoa, että mitä pienempiin tehtäviin päästään, sen luotettavampi koko projektisuunnitelmasta tulee. Kaikkia tehtäviä ei koskaan kuitenkaan huomata etukäteen, vaan aikatauluun on varattava myös aikaa ylimääräiselle, toteutusvaiheessa todettavalle työlle. /3/

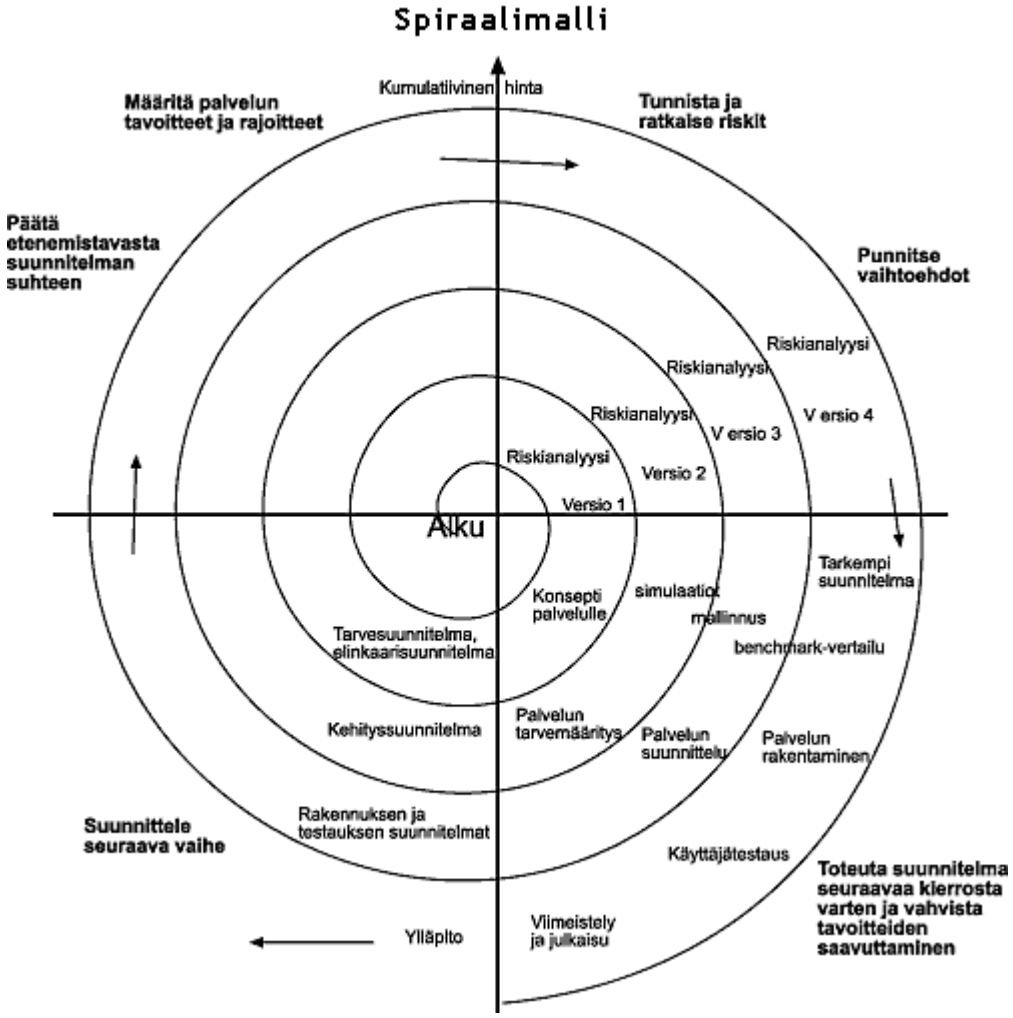
2.1 Vaihejakomallit

Edellä kuvattu vesiputous on yksi vaihejakomalleista. Useimpien ohjelmistoprojektien läpivienti tapahtuu vesiputouksesta johdetulla Evo-mallilla (evolutionary delivery model). Ohjelmistosta julkaistaan uusilla ominaisuuksilla täydennetty versio tietyin aikavälein. Jokaista julkaisua edeltää täydellisen vesiputousmallin mukaisen projektin läpivienti kuvion 2 mukaisesti. /3/



Kuvio 2. Evo-malli /3/

Spiraalimalli muistuttaa jonkin verran Evo-mallia. Siinä korostetaan erityisesti riskien hallintaa ottamalla riskit joka kierroksella omaksi vaiheekseen. Riski voi tarkoittaa esimerkiksi huonosti ymmärrettyjä vaatimuksia, huonosti ymmärrettyä arkkitehtuuria, suorituskykyongelmia, aikatauluongelmia tai teknologiaan liittyviä ongelmia. Kuvion 3 spiraalissa kehitys aloitetaan keskustasta ja jokainen kierros päättyy uuteen toimitukseen ja siirtää projektia suurempaan mittakaavaan. /1/



Kuvio 3. Spiraalimalli /6/

Joskus tilanne voi olla myös se, että projekti ei toteuta minkäänlaista varsinaista vaihejakomallia. Tällöin voidaan puhua koodaa ja korjaa -mallista. Koodaa ja korjaa -malliin päädytään, jos kunnollista projektisuunnitelmaa ei ole tehty. On vain olemassa yleinen näkemys siitä, mitä pitää tehdä. Yhtään aikaa ei käytetä muuhun kuin puhtaaseen ohjelmointiin (esimerkiksi suunnitteluun, dokumentointiin tai laadunvalvontaan). Malli on tehokas, koska tuloksia saadaan heti, mutta koodaa ja korjaa -mallilla tuskin koskaan päästään haluttuun lopputulokseen. /1/

2.2 Ketterä ohjelmistokehitys /5/

Ketterä ohjelmistokehitys tarkoittaa menetelmää, jossa tärkeimmät asiat ovat nopea kehitys, suora viestintä ja muutoksiin reagointi. Ketterät menetelmät minimoivat riskejä jakamalla ohjelmistokehityksen lyhyisiin kokonaisuuksiin, iteraatioihin. Yksi iteraatio

voisi olla esimerkiksi kuvion 2 ensimmäisen version toteutus. Iteraatio lisää vain vähän uutta toiminnallisuutta, mutta kun projektia viedään eteenpäin iteratiivisesti, pysyy projektin kokonaiskuva paremmin hallussa. Kuvaavaa ketterälle ohjelmistokehitykselle on, että muutoksiin reagoidaan nopeasti ja suunnitelmia muutetaan vastaamaan todellista tilannetta. Yhden iteraation lopussa arvioidaan uudelleen projektin tärkeysjärjestykset ja suunnitellaan seuraavan iteraation sisältö. Ketterissä menetelmissä arviointimenetelmänä käytetään usein suunnittelupokeria, joka on kuvattu luvussa 3.2.5.

Scrum on ehkä suosituin ketterä menetelmä. Scrumissa kehitystyö tapahtuu 1–4 viikon iteraatioissa. Jokaisen iteraation päätteeksi julkaistaan toimiva tuotos. Päivittäisissä palavereissa jokainen ryhmän jäsen kertoo, mitä on tehnyt ja mitä aikoo tehdä, sekä arvioi, miten paljon aikaa kuluu jäljellä olevien ominaisuuksien toteuttamiseen. Näin tietämys leviää ja ongelmiin voidaan puuttua heti.

Vesiputousmallin mukaisissa projekteissa on yleensä ainakin seuraavat roolit: määrittelijä, suunnittelija, ohjelmoija, testaaja ja projektipäällikkö. Scrum-projektissa esiintyy vain kolme eri roolia: Tuotteen omistaja, Scrum-mestari ja tiimi. Tuotteen omistaja vastaa tuotteen ominaisuuksista. Omistaja on tyypillisesti tuotepäällikkö tai asiakkaan edustaja. Omistajan tehtävänä on tehdä kaikki päätökset tuotteen ominaisuuksista ja toiminnallisuuksiin vaikuttavista seikoista. Scrum-mestarin tehtävänä on huolehtia siitä, että tiimi voi tehdä työtään häiriöittä. Projektiryhmän jäsenet raportoivat päivittäin ongelmista, jotka vaikeuttavat töiden etenemistä. Scrum-mestari ratkoo nämä ongelmat, jonka lisäksi hän johtaa päivittäiset aamupalaverit. Tiimiin kuuluvat kaikki projektissa mukana olevat henkilöt. Scrumissa korostetaan, että kukin projektiryhmän jäsen on projektin kannalta yhtä tärkeä ja että ryhmä yhdessä vastaa tuotteen kaikista puolista, ei koskaan yksittäinen henkilö.

3 Ohjelmistoprojektin arviointi

Jotta ohjelmistoprojekti voisi onnistua, se pitää suunnitella hyvin. Yksi tärkeimmistä suunnittelun kohteista on projektin aikataulutus ja työmääräarviointi. Karkeasti sanottuna: mitä enemmän aikaa käyttää aikataulujen suunnitteluun, sitä varmemmin projekti onnistuu. Jos projektille ei ole tehty riittävän tarkkaa aikataulua, tai aikataulu on arvioitu väärin, seuraa erilaisia ongelmia. Usein ongelmat kasaantuvat sitä enemmän, mitä pidemmälle projekti etenee. Jos jo projektin varhaisessa vaiheessa ei ryhdytä korjaaviin toimenpiteisiin, ollaan pian tilassa, jossa projektin eteneminen pysähtyy ja kaikki aika menee erilaisen hukkatyön tekemiseen.

Kun projektia määritellään, arvioidaan projektin koko ja projektiin kuluva aika saatavilla olevan informaation avulla. Arvion perusteella saadaan päivämäärä tai aikaväli, jolloin projektin arvioidaan olevan valmis. Projektin valmistumisella saattaa olla kiinteä tavoiteaika. Sovelluksen pitää esimerkiksi olla valmis, ennen kuin sitä käyttävä laite julkaistaan. Arviota ei missään tapauksessa pidä sekoittaa tavoitteeseen, vaan arvioiden perusteella projekti resursoidaan siten, että tavoitteeseen päästään. /1/

3.1 Ohjelmistoprosessin tarkentuminen

Ohjelmiston kehittäminen on prosessi, joka tarkentuu vähän kerrallaan. Kun saadaan ensimmäinen mielikuva valmistettavasta ohjelmistotuotteesta, ei vielä osata sanoa juuri mitään siitä, kuinka kauan tuotteen valmistaminen saattaisi viedä aikaa. Kun tuotteen kaikki ominaisuudet on määritelty, työmääräarvio tarkentuu, mutta on edelleen hyvin epätarkka. Kun määrittelyjen pohjalta tehdään tekninen suunnittelu ja aloitetaan toteutus, työmääräarvion pitäisi olla jo melko tarkka. Jos tilannetta käsitellään kuvion 1 vesiputouksen kautta, voidaan todeta, että mitä ylemmällä tasolla vesiputouksessa ollaan, sitä epätarkempi on työmääräarvio. Mutta kuten luvussa 2 mainittiin, vesiputouksen alemmilta tasoilta joudutaan joskus palaamaan myös ylöspäin. Jos vasta testausvaiheessa huomataan jokin puute, joka vaatii paluuta suunnittelu-, tai jopa määrittelytasolle, alkuperäiset työmääräarviot eivät enää päde. /1/

3.2 Arvioiden tekeminen

Arvioissa pitäisi pyrkiä mahdollisimman suureen tarkkuuteen. Ilman riittäviä perusteluja tehtyjä arvioita tulisi välttää. Joskus niihin täytyy kuitenkin turvautua. Esimerkiksi suunniteltaessa projektia, jossa tarkoitus on rakentaa jonkin olemassa olevan komponentin päälle, mutta lähdekoodeja tähän komponenttiin ei ole tarjolla, täytyy turvautua epämääräisempään arvioon. Tällöin täytyy pitää huoli siitä, että arvio ei missään vaiheessa prosessia muutu lopulliseksi, vaan arviota tarkennetaan heti kun se on mahdollista. /1/

Hyvä keino arvion tarkentamiseksi on hyödyntää edellisten, vastaavanlaisten projektien tietoja. Jokaisesta projektista pitäisi jäädä dokumentti, josta selviää esimerkiksi kuinka projektin alkuperäiset aikataulut pitivät ja mitä ongelmia projektin aikana ilmeni. Usean eri henkilön voi antaa arvioida projektin osia itsenäisesti ja vertailla näistä arvioista saatuja tuloksia. /1/

3.2.1 Ohjelmiston koon arviointi /1/

Ohjelmiston kokoarvio voidaan tehdä esimerkiksi käyttämällä toimintopistemallia. Määrittelydokumentin perusteella ohjelmisto voidaan jakaa toimintopisteisiin, joita ovat:

1. Syötteet: lomakkeet, kyselyt tai kontrollit, joiden kautta käyttäjä lisää, poistaa tai muuttaa ohjelman tietoja.
2. Tulosteet: näytöt, raportit tai viestit, joita ohjelma tuottaa käyttäjälle tai toiselle ohjelmalle.
3. Kyselyt: syöte/tulostusyhdistelmät, joissa syöte tuottaa välittömästi tulosteen.
4. Loogiset sisäiset tiedostot: esimerkiksi relaatiotietokannassa olevat yksittäiset tiedostot tai taulut, jotka ovat täysin ohjelman hallinnassa.
5. Rajapinnat: esimerkiksi tiedostot, joita ohjaavat muut ohjelmat, joiden kanssa toteutettava ohjelma on vuorovaikutuksessa.

Toimintopisteiden ja erilaisten kertoimien (kompleksisuuskerroin, ohjelmointikielen vaikutuskerroin) avulla voidaan arvioida lopullinen koodirivimäärä. Toimintopisteitä

voidaan verrata myös aikaisempiin vastaavanlaisiin projekteihin ja tehdä tämän perusteella päätelmät ohjelmiston koosta.

3.2.2 Algoritmiset lähestymistavat

Varsinaisen työmäärän arviointiin on useita eri menetelmiä. Yksi tapa on käyttää valmista arviointiohjelmistoa, jolle voidaan syöttää esimerkiksi ohjelmiston koko toimintopisteinä ja rivimääränä. Voidaan myös käyttää algoritmisia lähestymistapoja, joita ovat esimerkiksi COCOMO-malli. COCOMO-mallin syötteitä ovat mm. ohjelman koko sekä joukko kustannuskertoimia. Kustannuskertoimet kuvaavat projektin ajankäyttöön vaikuttavia tekijöitä, esimerkiksi projektin vaikeusastetta, osallistujien kokemusta ja luotettavuusvaatimuksia. Tuloksena mallista saadaan työmäärä henkilötyökuukausina ja projektin vaatima kalenteriaika. Erilaiset mallit antavat hyviä tuloksia, jos aiemmin on tehty useita samankaltaisia projekteja, joista on lisäksi olemassa luotettavia mittaustietoja. Käytännössä COCOMO-mallin hyödyntäminen ohjelmistotekniikassa on suhteellisen vähäistä. COCOMO-malli on kuvattu yksityiskohtaisesti alan kirjallisuudessa eikä tässä opinnäytetyössä paneuduta malliin tämän enempää. Luvussa 3.2.4 käydään läpi erästä algoritmista suunnittelutapaa. /3/

3.2.3 Yksityiskohtien arviointi

Onnistuneen työmääräarvion aikaansaamiseksi on erittäin tärkeää arvioida pieniä yksityiskohtia. Tämä tarkoittaa projektin pilkkomista mahdollisimman useaan osaan ja osien arvioimista erikseen. Esimerkkinä voitaisiin ajatella ohjelmiston (jokseenkin monimutkaista) käyttöliittymän näkymää. Koko näkymän ohjelmoimisesta ei kannata tehdä yhtä tehtävää ja arvioida sille tiettyä kestoa. Sen sijaan kannattaa miettiä esimerkiksi näkymässä olevien syötteiden ja tulostuksien lukumäärää, onko näkymän graafisten komponenttien toteuttamiseen jo olemassa valmiit luokat vai täytyykö ne mahdollisesti tehdä itse, sekä esimerkiksi kuinka näkymän ja muiden ohjelmiston osien välinen tiedonsiirto toteutetaan. Näin arviosta saattaa muodostua huomattavasti suurempi. Vaikka virhearviointeja tehtäisiinkin, teettävät ne pienempien kokonaisuuksien osana vähemmän lisätyötä. Tämä perustuu suurten lukujen lakiin, jonka mukaan summien virhe on suurempi kuin virheiden summa. Tämä tarkoittaa sitä, että 10 %:n virhe yhdessä isossa tehtävässä voi olla 10 % yli tai 10 % alle

kokonaisarvion. Toisaalta 10 % virhe 50 pienessä tehtävässä on vaihtelevasti sekä yli että alle arvion ja yleensä virheet kumoavat toisensa. /1/

3.2.4 Tapauspohjainen arviointi

Otetaan esimerkki projektista, jolle projektiryhmä on laatinut yksityiskohtaisen tehtävälistan. Kehittäjä toteuttaa ensimmäisen tehtävän ajoissa. Toisen tehtävän aikana ilmaantuu yllättäviä ongelmia, mutta kehittäjä tekee ylitöitä saaden tehtävän valmistumaan ajallaan. Kolmannen tehtävän aikana ongelmia tulee lisää ja pian kehittäjä huomaa olevansa päivän myöhässä aikataulusta. Viikon kuluttua kehittäjä on myöhässä kokonaisen tehtävän verran. Miksi näin tapahtui?

Jos kehittäjää pyydetään antamaan kiinteä yhden pisteen työmääräarvio jollekin ominaisuudelle, tulos voi olla esimerkiksi 1,5 päivää. Jos samaa kehittäjää pyydetään antamaan arvio toteutuvasta aikataulusta parhaassa ja huonoimmassa tapauksessa, tulos on useimmiten seuraavan kaltainen: parhaassa tapauksessa 1,25 päivää, huonoimmassa 2,0 päivää. Kiinteät työmääräarviot ovat usein lähempänä tapauspohjaisen arvion parasta tapausta. Toisin sanoen kun kehittäjää pyydetään arvioimaan työmäärä antamalla yksi arvo vaihteluvälin sijaan, on tämä arvio todennäköisesti optimistinen. Taulukko 1 kuvaa erään projektin kiinteät arviot sekä arviot parhaassa ja huonoimmassa tapauksessa. /2/

Taulukko ei vielä kerro, mitä arviota tulisi lopulta käyttää. Yksi vaihtoehto olisi laskea parhaimman ja huonoimman tapauksen keskiarvo. Monessa tapauksessa huonoin tapaus on kuitenkin paljon huonompi kuin odotettavissa oleva tulos. Jos arviot perustuisivat keskiarvoon, tuloksena saattaisi olla liian pessimistinen aikataulu.

Taulukko 1. Kiinteät arviot sekä arviot parhaassa ja huonoimmassa tapauksessa /2/

Ominaisuudet	**Alkuperäinen kiinteä arvio (d)**	**Paras tapaus (d)**	**Huonoin tapaus (d)**
Ominaisuus 1	1,5	1,25	2,0
Ominaisuus 2	1,5	1,5	2,5
Ominaisuus 3	2,0	2,0	3,0
Ominaisuus 4	0,5	0,75	2,0
Ominaisuus 5	0,5	0,5	1,25
Ominaisuus 6	0,25	0,25	0,5
Ominaisuus 7	2,0	1,5	2,5
Ominaisuus 8	1,0	1,0	1,5
Ominaisuus 9	0,75	0,5	1,0
Ominaisuus 10	1,25	1,25	2,0
Yhteensä	**11,25**	**10,5**	**18,25**

Program Evaluation and Review Technique (PERT) on tekniikka, jolla voi kätevästi laskea odotettavissa olevan aikataulun. Tekniikka vaatii uuden käsitteen käyttöönoton. Tätä kutsutaan *todennäköisimmäksi tapaukseksi*. Todennäköisimmän tapauksen, eli todennäköisimmän toteutuvan aikataulun, voi arvioida esimerkiksi käyttämällä apuna asiantuntijoita. Tämän jälkeen odotettavissa oleva aikataulu lasketaan kaavalla:

$$\text{odotettavissa oleva} = [\text{paras} + \text{huonoin} + (4 \cdot \text{todennäköisin})] / 6$$

Kyseessä on siis keskiarvo, jossa painotetaan todennäköisintä tapausta. Tulokset sovellettuna esimerkkiin on esitetty taulukossa 2. Keskimääräinen arvio olisi ollut 14,4 päivää. Taulukosta 2 nähdään, että nyt odotettavissa oleva tulos on lähempänä arvoalueen alapäätä. /2/

Taulukko 2. Arviointi käyttäen parasta, huonointa ja todennäköisintä tapausta /2/

Ominaisuudet	Paras tapaus (d)	Todennäköisin tapaus (d)	Huonoin tapaus (d)	Odotettavissa oleva tulos (d)
Ominaisuus 1	1,25	1,5	2,0	1,54
Ominaisuus 2	1,5	1,75	2,5	1,83
Ominaisuus 3	2,0	2,25	3,0	2,33
Ominaisuus 4	0,75	1	2,0	1,13
Ominaisuus 5	0,5	0,75	1,25	0,79
Ominaisuus 6	0,25	0,5	0,5	0,46
Ominaisuus 7	1,5	2	2,5	2,00
Ominaisuus 8	1,0	1,25	1,5	1,25
Ominaisuus 9	0,5	0,75	1,0	0,75
Ominaisuus 10	1,25	1,5	2,0	1,54
Yhteensä	**10,5**	**13,25**	**18,25**	**13,62**

PERT on yksi algoritminen lähestymistapa muiden joukossa. Oikein käytettynä erilaiset kaavat voivat antaa parempia tuloksia kuin nopeasti tehdyt kiinteät yhden pisteen arviot. Kokeneimmatkaan arvioijat eivät aina osaa ottaa huomioon kaikkea. Arvioijan on syytä kehittää ajattelutapaansa hahmottamaan parhaimpia ja huonoimpia tapauksia. Tällöin arvioijan kiinteätkin työmääräarviot tulevat luotettavimmiksi. /2/

3.2.5 Suunnittelupokeri /4/

Ketterissä menetelmissä suosittu tapa on käyttää suunnittelupokeria (planning poker). Suunnittelupokeriin osallistuvat kaikki projektin kehittäjät. Ketterällä menetelmällä toteutettavassa projektissa kehittäjien lukumäärä ei yleensä ylitä kymmentä. Jos ylittää, on ryhmä parempi jakaa osiin.

Suunnittelupokeri-istunnon alussa jokaiselle arvioijalle annetaan korttipakka. Jokaisella kortilla on oma lukuarvo, joka kuvaa arviota. Jokaista arvioitavaa vaatimusta kohden luetaan vaatimuksen kuvaus. Lukijana toimii tuotteen omistaja (katso luku 2.2), joka myös vastaa kaikkiin mahdollisiin arvioijien esittämiin kysymyksiin. Kun kaikkiin kysymyksiin on vastattu, jokainen arvioija valitsee henkilökohtaisesti kortin kuvaamaan

arviota. Kortteja ei näytetä toisille ennen kuin kaikki arvioijat ovat tehneet päätöksensä arviosta. Kun päätökset on tehty, kaikki kortit käännetään ympäri samanaikaisesti.

Tässä vaiheessa on hyvin todennäköistä, että arviot ovat erikokoisia. Mikäli näin on, korkeimmat ja alhaisimmat arviot antaneet kehittäjät perustelevat syyn arviolleen. Eri arvion antaneilla kehittäjällä voi olla eri näkemys siitä, miten vaatimus tulisi teknisesti toteuttaa ja mitä riskejä toteutukseen liittyy. Keskustelua arvioista on hyvä käydä joitakin minuutteja, mutta liikaa aikaa ei kannata käyttää. Usein pienistä yksityiskohdista voi saada aikaiseksi pitkiä keskusteluja, mikä kuitenkin lopulta voi olla ajan tuhlausta. Keskustelut kannattaa siis käydä lyhyehköinä ja tehokkaina.

Keskustelun jälkeen tehdään uudelleenarviointi, joka tapahtuu samalla tavalla kuin ensimmäinen arviointi. Uudelleenarvioinnin jälkeen arviot saattavat olla jo yhtenevät tai lähempänä toisiaan. Jos arvioissa on vielä suuria eroja, prosessi toistetaan. Tavoitteena on päästä ryhmän sisällä yhteisymmärrykseen ja luoda vaatimuksen toteuttamiselle yksi arvio. Yleensä tähän päästään vähemmällä kuin kolmella kierroksella.

Suunnittelupokeria voidaan käyttää esimerkiksi suunniteltaessa uutta projektia. Tällöin projekti on jaettu useisiin tehtäviin ja jokainen tehtävä arvioidaan erikseen. Pokeria voi käyttää myös kesken projektin. Esimerkiksi Scrumissa pelataan suunnittelupokeria jokaisessa iteraatiossa, kun suunnitellaan iteraation julkaisua. Suunnittelupokerin vahvuus on, että se yhdistää monien eri ammattilaisten perustellut mielipiteet.

3.4 Liian optimistiset aikataulut

Aikatauluarvioiden tekemiselle pitäisi varata riittävästi aikaa. Arvioiden tekeminen saatetaan kuitenkin nähdä tuottamattomana työnä, joka aiheuttaa paineita saada arvio nopeasti tehtyä. Ohjelmistoprojektin arviointivaihe voidaan jaotella kolmeen sidosryhmään: hallinto, arvioija, asiakas. Usein hallinnolla ja arvioijilla on erilaiset näkemykset siitä, kuinka kauan projektin tulisi kestää. Hallinto haluaisi pitää kulut alhaalla ja aikataulun pienenä, jotta projekti tuottaisi mahdollisimman hyvin voittoa (tai jotta kilpailutustilanteessa projekti voitettaisiin ja päästäisiin tekemään). Arvioijat eivät välttämättä halua leimautua hankaliksi ja saattavat tehdä liian optimistisen arvion hallinnon paineen alla. Asiakas hyväksyy tietysti suunnitelman, jonka perusteella

ohjelmisto tultaisiin toteuttamaan nopeimmin ja halvimmin. Samalla asiakas tulee hyväksyneeksi suunnitelman, joka ehkä on kaikista vaihtoehdoista epärealistisin. /1/

Liian optimistisiin arvioihin voi toki olla muitakin syitä. Asiakas voi esimerkiksi vaatia tuotetta esittelyvalmiiksi tiettyyn päivämäärään mennessä esimerkiksi jo sovitun tuotteen esittelytilaisuuden takia. Jos arvioijat antavat arviovälin, jonka mukaan lopullinen tuote on valmis esimerkiksi 10–14 kuukauden päästä, hallinto tai asiakas saattaa automaattisesti käyttää nopeinta vaihtoehtoa. Projekti saattaa alkaa realistisella aikataululla, mutta projektin edetessä ohjelmistoon määritellään uusia ominaisuuksia, jonka seurauksena arvio muuttuu liian optimistiseksi. Yleisin syy liian optimistisille aikatauluille on kuitenkin se, että arvioinnissa ei ole osattu ottaa huomioon kaikkia mahdollisia ongelmia ja riskejä, eli projekti on vain arvioitu huonosti. /1/

Liian optimistisen aikataulun vuoksi projektin alkupään toimissa saatetaan käyttää liian vähän aikaa vaatimusanalyysiin ja suunnitteluun. Huonosti tehdyn suunnittelun jälkeen projektin aikana ilmenee ylimääräistä testausta, puutteiden korjaamista, uudelleensuunnittelua ja muuta lisätyötä. Tämä lisätyö saattaa kasvaa niin suureksi, että varsinainen ohjelmiston kehitys hidastuu huomattavasti tai pahimmassa tapauksessa pysähtyy kokonaan. Aina kun projekti myöhästyy varsinaisesta valmistumispäivästään, tehdään uusi aikatauluarvio (tai ainakin olisi syytä tehdä), joka vie taas kehittäjien aikaa. /1/

3.5 Uudelleenkalibrointi

Joskus tulee eteen tilanteita, joissa projektin jo ollessa käynnissä huomataan, että kaikki arviot eivät ole realistisia. Jos tehtävän pituudeksi on määritelty esimerkiksi 4 viikkoa, mutta tehtävä tulee valmiiksi vasta viiden viikon kuluttua, on mahdollista reagoida myöhästymiseen kolmella tavalla.

1. Olettaa, että menetetty viikko saadaan kiinni myöhemmin.
2. Lisätä menetetty viikko kokonaisaikatauluun.
3. Kertoa koko aikataulu myöhästymisen suuruudella, tässä tapauksessa 25 prosentilla. /1/

Hyvin usein toimitaan kohdan 1 mukaisesti. Varsinkin jos ollaan projektin alkupäässä, ajatellaan, että myöhästyminen johtuu yleisistä alkuhankaluuksista ja menetetty aika varmasti saadaan kiinni, kun projekti etenee. Saatetaan myös ajatella, että projektin loppupäähän on laskettu tarpeeksi ”tyhjää” tilaa, joka eliminoi pienet aikatauluongelmat. Tämä on varsin usein kuitenkin toiveajattelua, eikä projekti koskaan tule saamaan kiinni menetettyä aikaa. /1/

Realistisempaa on lisätä kohdan 2 mukaisesti menetetty aika kokonaisaikatauluun. Tässä tapauksessa tehdään kuitenkin oletus, että aikataulu on oikein lukuun ottamatta kyseistä tehtävää, jossa tapahtui viikon myöhästyminen. Syyt, joiden takia aikataulut ovat epätarkkoja, saattavat kuitenkin olla systemaattisia (esimerkiksi hallinnon painostus käyttää optimistisia oletuksia, koodialustan epävarma toiminta). Ne ovat voimassa koko projektin ajan, jonka vuoksi olisi parempi soveltaa kohtaa 3. Usein näin radikaaliin vaihtoehtoon ei kuitenkaan haluta turvautua. Tällöin jatketaan alkuperäisten suunnitelmien pohjalta ja tarkkaillaan, kuinka seuraavan välietapin toteuttaminen sujuu. Jos silloinkin ollaan saman verran tai vielä enemmän myöhässä, voi tarvittavien korjaustoimenpiteiden tekeminen olla jo huomattavasti vaikeampaa. /1/

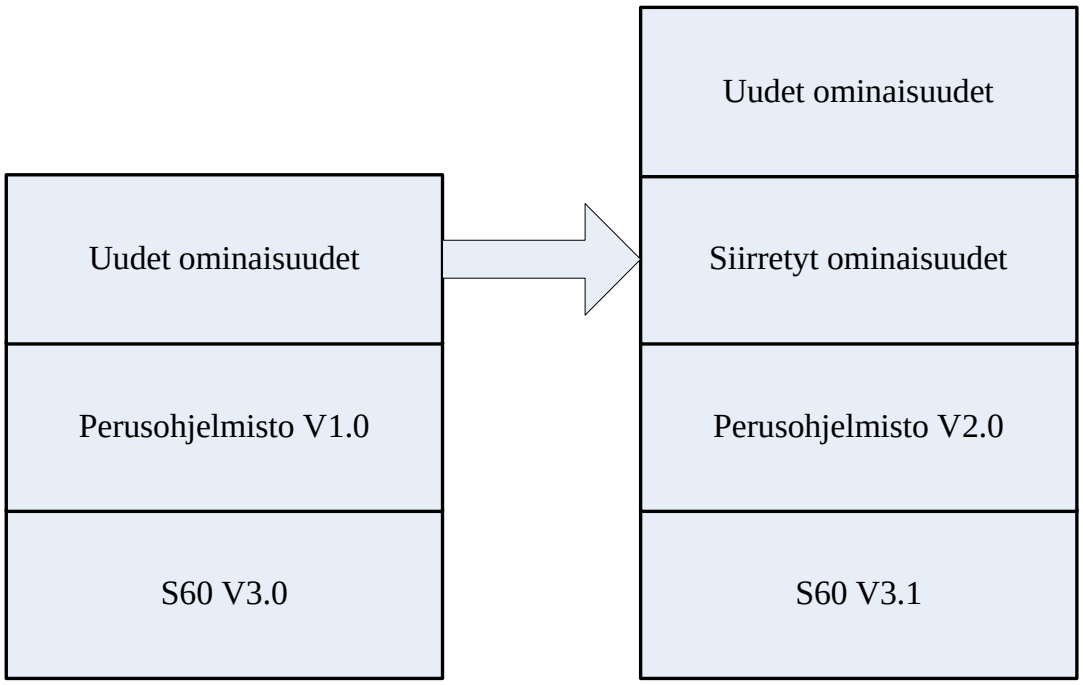
4 Projektin kuvaus

Tässä luvussa kuvataan esimerkkinä toimiva Sasken Finland Oy:n ohjelmistoprojekti. Projektin tavoitteena oli toteuttaa ohjelmisto matkapuhelimeen asiakkaan tekemien määrittelyjen pohjalta. Asiakas oli aikaisemmin teettänyt Sasken Finland Oy:llä ohjelmiston ensimmäisen version. Nyt kyseessä oli jatkoprojekti, jonka aikana oli tehtävänä toteuttaa ohjelmistosta uusi versio. Ohjelmisto tuli toteuttaa uudelle alustalle, siirtää siihen ominaisuuksia vanhasta ohjelmistosta ja toteuttaa tämän kokonaisuuden päälle määritellyt uudet ominaisuudet.

Projektin oli arvioitu olevan suhteellisen pienitöinen. Tarkoituksena oli, että projektiin otettaisiin kehittäjiksi harjoittelija ja kokeneempi tekijä, jolloin projekti toimisi harjoittelijalle hyvänä ponnahduslautana ohjelmistoalalle.

Sasken Finland Oy ei rakentanut ensimmäistä versiota tyhjästä, vaan ohjelmisto rakennettiin jo olemassa olevan (asiakkaan valmistaman) ohjelmiston päälle. Nyt tehtävä uusi ohjelmisto tuli rakentaa asiakkaan ohjelmiston uuden version päälle. Samalla projektin pohjana oleva sovellusalusta päivittyi S60 versiosta 3.0 versioon 3.1.

Kuviossa 4 havainnollistetaan ohjelmiston ensimmäisen ja toisen version suhdetta. Projektin tavoitteena oli toteuttaa oikealla puolella oleva ohjelmisto. S60-alustan muutos ei sinänsä tuonut paljon muutoksia uuden version toteutukseen, vaan version muutoksen seurauksena oli lähinnä lisätty S60-alustan käyttöliittymän muokattavuutta ja nopeutta sekä lisätty alustaan joitakin uusia ominaisuuksia. Myös perusohjelmistojen erot arvioitiin alkuperäisten selvitysten perusteella pieniksi.



Kuvio 4. Alkuperäisen ja jatkoprojektin kuvaus

Ohjelmisto pohjautui asiakas/palvelin-arkkitehtuuriin, jossa tiedonsiirto asiakasohjelmiston ja palvelimen välillä toteutettiin HTTP:n yli. Asiakkaan vastuulla oli tehdä palvelimen toteutus ja määritellä asiakasohjelmiston ja palvelimen välinen rajapinta. Toimittajan vastuulla oli rakentaa asiakaspään ohjelmisto, joka osaisi keskustella palvelimen kanssa asiakkaan määrittelemän rajapinnan kautta.

4.1 Arviot

Alkuperäisessä projektisuunnitelmassa varsinainen ohjelmointityö oli jaettu noin 90 tehtävään. Koko projekti sisälsi 3 virstanpylvästä, jotka on esitetty korkealla tasolla seuraavassa.

Virstanpylväs 1	Ohjelmiston v1.0 ominaisuudet siirrettynä perusohjelmiston v2.0 päälle. Kaikki perusohjelmiston v1.0 ominaisuudet toimivat. Toimitetaan lähdekoodi ja binäärit. Aikaa noin kuusi viikkoa projektin alusta.
Virstanpylväs 2	Keskeisimmät uudet ominaisuudet toteutettuna. Toimitetaan lähdekoodi ja binäärit. Aikaa noin seitsemän viikkoa edellisestä virstanpylväästä.
Virstanpylväs 3	Loput ominaisuudet toteutettuna. Valmis ohjelmisto toimitettavaksi. Toimitetaan lähdekoodi, binäärit ja testausraportti. Aikaa noin neljä viikkoa edellisestä virstanpylväästä.

4.2 Projektihenkilöstö

Projektiin kiinnitettiin projektipäällikkö, kaksi kehittäjää (joista toinen tämän opinnäytetyön tekijä) ja yksi tekninen neuvonantaja (mukana projektin ensimmäisen kuukauden ajan). Asiakkaan puolelta projektissa oli mukana kaksi palvelimen kehittäjää sekä kaksi käyttöliittymän suunnittelusta vastuussa olevaa henkilöä.

4.3 Asiakkaan toimittamat dokumentit

Asiakas toimitti käyttöliittymämäärittelyn, joka sisälsi käyttöliittymän tasolla kuvauksen ohjelmiston toiminnasta, sekä rajapintamäärittelyn, jossa kuvattiin palvelimen ja asiakasohjelmiston välinen tiedonsiirto. Lisäksi asiakas toimitti vaatimuslistan, joka sisälsi jokaisesta vaatimuksesta pienen kuvauksen ja tärkeysasteen.

5 Projektin aikana ilmenneet ongelmat

Tässä luvussa käydään läpi projektin aikana ilmenneitä ongelmia ja kuinka niihin reagoitiin. Varsinaisia parannusehdotuksia tässä luvussa ei käydä läpi, vaan tarkemmat analyysit ovat luvussa 6.

5.1 Koodin siirtämisen ongelmat

Heti projektin alussa kävi ilmi, että asiakkaan toimittama perusohjelmiston versio 2.0 erosikin täydellisesti arkkitehtuuriltaan ensimmäisestä versiosta. Itse asiassa kyseessä ei varsinaisesti edes ollut uudempi versio, vaan kokonaan uusi ohjelmisto. Ohjelmiston siirtotehtävä oli suunniteltu olettaen, että muutos ohjelmiston S60 versio 3.0:sta 3.1:een ei olisi merkittävä.

Ohjelmistojen eroista jotain kertoo se, että aiemman version varsinaista ohjelmaa ajettiin yhdessä prosessissa, kun uudessa prosesseja oli kolme. Vanhassa ohjelmistossa oli 60000 koodiriviä, uudessa 150000. Siirtotehtävä (virstanpylväs 1) osoittautui käytännössä mahdottomaksi toteuttaa. Suuri osa alkuperäisesti siirrettäviksi suunnitelluista ohjelmiston osista jouduttiin toteuttamaan alusta uudelleen, ja samalla alkuperäiset aikatauluarviot menettivät merkityksensä.

Alkuperäisistä työmääräarvioista luovuttiin. Asiakkaan kanssa päätettiin, että virstanpylväs 1 jätetään toteuttamatta. Sen sijaan tavoitteeksi otettiin julkaista mahdollisimman nopeasti versio, joka sisältäisi perustoiminnot ja jo osan uusista ominaisuuksista. Kun ensimmäinen julkaisukelpoinen versio ohjelmistosta saatiin valmiiksi, oltiin aikataulusta jo selvästi myöhässä. Tällöin siirryttiin viikoittaisiin julkaisuihin. Syy tähän oli, että projekti haluttiin saada etenemään järjestyksessä, ja asiakkaalle haluttiin antaa välitoimituksia testattavaksi mahdollisimman usein. Näin projektin eteneminen näkyisi ja asiakkaalta saataisiin kommentteja. Viikon alussa suunniteltiin viikon aikana toteutettavat tehtävät ja viikon loppupuolella julkaistiin tuotos asiakkaalle.

5.2 Ohjelmakoodin laajuus

Projektin pohjana olevaa perusohjelmiston koodia oli paljon enemmän kuin alunperin oli ajateltu. Tämä pakotti kehittäjät käyttämään paljon aikaa koodin tutkimiseen ja ymmärtämiseen. Asiakkaalle tehtiin pyyntö selvittää, kuka ohjelmiston oli tehnyt ja olisiko mahdollisesti saatavilla asiantuntija-apua. Tällaista apua ei koskaan saatu.

5.3 Määrittelydokumenttien laatu

Asiakkaan toimittamat määrittelydokumentit aiheuttivat yllättävän paljon päänvaivaa kehittäjille. Ensimmäinen vaikutelma määrittelydokumenteista oli tosin myönteinen, mutta kun projekti eteni, dokumenttien epätarkkuudet alkoivat valjeta. On toki selvää, että määrittelyt eivät koskaan ole täydellisiä, mutta muutama asia on syytä tässä mainita. Rajapintamäärittely oli tehty ohjelmiston edellisen version dokumentin pohjalle. Osa viestinvälityksestä oli kuitenkin kokonaan muutettu uudessa versiossa. Tästä syystä dokumentissa oli määrittelyjä, jotka eivät enää päteneet uudessa versiossa. Häiritsevästi osa näistä ominaisuuksista oli varustettu ”deprecated”, eli poistunut käytöstä -merkinnällä, mutta osan kohdalla merkintää ei ollut. Tutkimalla vaatimuslistaa ja käyttöliittymämäärittelyä (ja viime kädessä kysymällä asiakkaalta) oli selvitettävissä, mitkä osat lopulta piti toteuttaa. Tämä oli kuitenkin melko työlästä.

Käyttöliittymämäärittelyn heikoin osa oli kuvitus. Käytännössä yksikään dokumentin kuvista ei tarkalleen kuvannut sitä, miltä lopputuloksen olisi pitänyt näyttää. Kuvat olivat vain suuntaa antavia. Kuvien perusteella oli esimerkiksi vaikea hahmottaa, oliko tarkoitus käyttää S60-alustan valmiita graafisia komponentteja vai ohjelmoida komponentit itse. Nämäkin asiat selkenivät keskustelemalla asiakkaan kanssa. Tästä huolimatta ohjelmiston näkymiin jouduttiin tekemään muutoksia jälkikäteen, koska kehittäjien ja asiakkaan tulkinnat käyttöliittymämäärittelyn eri osista vaihtelivat.

5.4 Kanssakäyminen asiakkaan kanssa

Sasken Finlandin vastuulla oli toteuttaa asiakaspään ohjelmisto, ja asiakkaan vastuulla oli palvelimen toteutus. Täyttä selvyyttä siitä, mitkä ominaisuudet palvelimelle oli jo toteutettu, ei aina ollut. Toiminta selvisi viimeistään silloin, kun palvelimelle lähetettiin

jokin rajapintamäärittelyssä kuvattu viesti ja tutkittiin, mitä palvelin vastaa. Apuna tässä oli palvelimen päivittämä lokitiedosto. Jos palvelin ei toiminut toiveiden mukaisesti, piti siitä ilmoittaa asiakkaalle, joka yleensä sai korjauksen tehdyksi 1-2 päivän kuluessa.

5.4.1 Uudet vaatimukset

Projektin kuluessa ilmeni monia epätarkkuuksia käyttöliittymämäärittelyssä, jonka seurauksena määrittelyä tarkennettiin tai lisättiin kokonaan uusia ominaisuuksia. Yleensä uudet ominaisuudet olivat hyvin pienitöisiä toteuttaa, joten joissain tapauksissa sovittiin niiden toteutuksesta ilman erillistä korvausta.

5.4.2 Vaatimusten tärkeysjärjestys

Alkuperäisessä vaatimuslistassa kaikki vaatimukset oli varustettu merkinnällä ”mandatory”, eli pakollinen. Asiakas ei ollut halunnut asettaa vaatimuksia mihinkään tärkeysjärjestykseen, eikä valinnaisia vaatimuksia ollut. Tämä tarkoitti sitä, että kun projektin aikatauluja jouduttiin uusimaan, vaatimuksista ei voinut hakea joustoa.

5.5 Kehittäminen puhelimen tuotantoversiolla

Usein ohjelmistoprojekteissa, joissa kehitetään matkapuhelinsovelluksia, käytetään puhelimen tuotekehitysversiota testaamiseen. Tällaisella puhelimella on mahdollista saada ajonaikaista dataa ohjelman kulusta. Jos käytössä ei ole tällaisia ominaisuuksia, virheiden jäljittäminen on hidasta ja vaikeaa. Tässä projektissa tuotekehityspuhelimesta olisi ollut suuri hyöty.

5.6 Projektihenkilöstön vaihtuvuus

Koska projekti myöhästyi, jouduttiin osa kehittäjistä vaihtamaan. Tämä johtui siitä, että esimerkiksi pääsuunnittelija oli luvattu toiseen projektiin tietystä ajankohdasta lähtien eikä nykyistä projektia saatu valmiiksi ajoissa. Projektissa oli myös mukana kehittäjiä noin yhden kuukauden pituisissa jaksoissa auttamassa helpoimmin lähestyttävien tehtävien toteutuksessa. Yhteensä projektissa oli eri aikoina mukana kuusi kehittäjää. Yhtaikaisesti kehittäjiä työskenteli maksimissaan kolme.

Tämän opinnäytetyön kirjoittaja oli ainoa, joka työskenteli projektin alusta loppuun. Suuri henkilöstön vaihtuvuus vaati, että kaikki tarvittava tieto siirrettiin aina lähtevältä henkilöltä jäljelle jäävälle projektiryhmälle. Toisaalta kun uusi henkilö aloitti projektissa, aikaa kului henkilön perehdyttämiseen. Tätä helpotettiin antamalla uusille kehittäjille tehtäviä, jotka oli helppo sisäistää nopeasti. Myös asiakkaan puolella tapahtui muutoksia projektiryhmässä. Käytännössä asiakkaan alkuperäinen projektiryhmä hajosi projektin aikana kokonaan, jolloin asiakkaan puolelta mukana oli enää kaksi henkilöä. Tämä tapahtui sen jälkeen, kun alkuperäisestä valmistumistavoitteesta oli jo jääty jälkeen.

5.7 Testaaminen ja virheiden raportointi

Alkuperäisten suunnitelmien mukaan projektin aikana asiakkaalle tuli toimittaa kolme julkaisua, joista viimeinen olisi valmis ohjelmisto. Tarkoitus oli, että ennen jokaista julkaisua varmistettaisiin testaamalla, että julkaisu täyttää vaatimukset. Testausta tuli myös suorittaa jokaisen tehtävän kohdalla erikseen. Näin varmistuttaisiin siitä, että tehtävän vaatimukset täyttyvät ja että ominaisuudet toimivat määrittelyjen mukaisesti.

Ongelmaksi muodostui kiire. Tehtävien tekemiseen kului lähes poikkeuksetta enemmän aikaa kuin työmääräarvioihin oli kirjattu. Tästä syystä testaaminen jäi paikoittain liian vähäiseksi. Kun siirryttiin viikoittaisiin julkaisuihin, testausvastuuta siirrettiin asiakkaalle. Asiakas lupasi testata jokaisen julkaisun ja antaa palautetta ennen seuraavan julkaisun ajankohtaa. Projektisopimuksessa ei tällaisesta toiminnasta ollut sovittu, joten asiakkaan testaaminen ja virheiden raportointi oli hyvin vapaamuotoista. Liian usein asiakkaan raportit olivat enemmänkin käyttäjäkokemuksia kuin kunnollisia virheraportteja.

6 Miten olisi voitu toimia

Tässä luvussa analysoidaan, tunnistettiinko kaikki projektin aikana ilmenneet ongelmat, miten ongelmiin reagoitiin ja mitä ohjelmointi- ja arviointivaiheessa olisi ehkä voinut tehdä paremmin.

6.1 Reagointi virheellisiin työmääräarvioihin

Jos todetaan, että työmääräarviot eivät pidä paikkaansa, on projektin onnistumisen kannalta jopa suotavaa että työmääräarviot hylätään. On kuitenkin virhe jättää uudet arviot tekemättä. Viikoittainen suunnittelu ja julkaiseminen oli toki toimiva keino viedä projektia eteenpäin. Ohjelmoimiseen kuluva kokonaisaika olisi kuitenkin pitänyt arvioida mahdollisimman nopeasti alusta loppuun uudelleen. Uusien arvioiden tekoa hankaloitti se, että perusohjelmiston erojen takia tehtävälistalle oli tullut useita alkuperäisistä arvioista puuttuneita tehtäviä. Näistä osa oli teknisesti niin haastavia, ettei niitä osattu varmuudella arvioida ennen perinpohjaista tutustumista.

Tämä ei kuitenkaan olisi saanut olla syy jättää arvioita tekemättä. Jos olisi heti tehty edes suuntaa antava arvio (ja painotettu että kyseessä on vain suuntaa antava arvio), olisi heti huomattu, että projekti ei tule valmistumaan läheskään alkuperäisessä aikataulussa. Koska projektia vietiin eteenpäin viikon pituisissa iteraatioissa ja lopullinen takaraja oli vielä jokseenkin kaukana, syntyi mielikuva, että projekti etenee toivotusti. Tämä oli kuitenkin toiveajattelua.

Projektin myöhemmässä vaiheessa uudet työmääräarviot lopulta tehtiin. Arvioiden perusteella todettiin, että projekti tulee myöhästymään yli puolella vuodella. Tämä tieto olisi pitänyt olla käytettävissä jo paljon aiemmin. Tällöin tarvittavia korjaustoimenpiteitä (uusia kehittäjiä projektiin, ominaisuuksien karsimisia) olisi voitu suunnitella aiemmin, ja projekti ei ehkä olisi myöhästynyt yhtä paljon.

6.2 Työmäärien arviointi

Edellisessä luvussa todettiin, että työmääräarviot olivat virheelliset. Tässä luvussa tarkastellaan tarkemmin työmääräarvioinnin toteutusta. Työmääräarvion ensimmäisen

version laati ohjelmistosuunnittelija asiakkaan vaatimuslistan ja määrittelydokumenttien perusteella. Arvioinnissa käytettiin apuna Sasken Finlandin valmistaman ensimmäisen ohjelmistoversion dokumentaatioita, ja keskusteluissa oli mukana myös ensimmäistä versiota valmistamassa ollut pääsuunnittelija. Arvioiden tuloksena syntyi noin 90 tehtävän lista, jossa jokaiselle tehtävälle oli arvioitu aika päivissä.

Projekti ei kuitenkaan alkanut heti ensimmäisen työmääräarvioinnin jälkeen, vaan asiaan palattiin noin puolen vuoden kuluttua, jolloin vaatimuksiin oli tullut joitakin muutoksia. Projektin työmäärä annettiin toiselle ohjelmistosuunnittelijalle arvioitavaksi. Asiakasdokumenttien lisäksi käytettävissä oli nyt arvioiden ensimmäinen versio. Ohjelmistosuunnittelija kävi arviot läpi, lisäsi joitain puuttuvia tehtäviä ja kasvatti joidenkin tehtävien työmääräarviota. Projektipäällikkö aikataulutti projektin myöhemmin perustuen näihin arvioihin.

6.2.1 Arvioiden vertaileminen

Kaikkia luotettavia arviointimenetelmiä yhdistää eräs seikka: se, että arvioita vertaillaan keskenään. Tästä puhuttiin esimerkiksi luvussa 3.2.5, jossa käytiin läpi suunnittelupokeria. Tämän projektin ongelma oli se, että työmääräarviointi oli liikaa yksittäisen ohjelmistosuunnittelijan harteilla. Tämän seurauksena kunnollista keskustelua työmäärien suuruuksista ei syntynyt. Aina ei voida olettaa, että yksi henkilö tunnistaisi kaikki mahdolliset tuntemattomat tekijät ja riskit. Jos olisi toimittu niin, että kaksi eri ohjelmistosuunnittelijaa olisi arvioinut työmäärän alusta loppuun itsenäisesti, eteen olisi saattanut tulla toisistaan poikkeavia työmääräarvioita. Koska kumpikin arvioija olisi joutunut puolustamaan näkemystään, arvioista olisi lopulta saattanut tulla realistisemmat. Nyt tilanne oli se, että myöhemmin tehdyt arviot perustuivat liikaa jo olemassa oleviin arvioihin.

6.2.2 Muutoksiin reagointi

Luvussa 5.1 kerrottiin siirtotehtävän epäonnistumisesta. Epäonnistuminen ei välttämättä johtunut siitä, että siirtotehtävä olisi ollut huonosti arvioitu. Ongelmaksi muodostui pikemminkin se, että arvio oli tehty arviointiaikaan saatavilla olevan informaation perusteella. Myöhemmin tapahtui muutos, jonka seurauksena kohdealusta vaihtui

toiseen. Tällöin tehtiin oletus, että muutoksella ei ole välitöntä vaikutusta työmääräarvioihin. Tätä perusteltiin sillä, että itse S60-alusta ei muuttunut, vaan ainoastaan alustan päällä toimivan ohjelmiston versionumero. Oletus oli virhe, jonka seurauksena alkuperäiset työmääräarviot menettivät merkityksensä. Tämä oletus oli itse asiassa suurin syy projektin myöhästymiseen. Projektin aikana kaikkiin muutoksiin olisi syytä suhtautua vakavasti. Aina kun puhutaan muutoksesta, pitäisi automaattisesti osata analysoida, miten muutos vaikuttaa tuotteen aikatauluun, kustannuksiin ja ominaisuuksiin. Muutosprosessi analyyseineen ja arviointeineen voi viedä aikaa, mutta se on välttämätöntä jotta projektin aikataulutus säilyy kestävällä pohjalla.

6.2.3 Vaatimusten ymmärtäminen

Suurin osa projektin ongelmista johtui siis muuttuneesta alustasta. On kuitenkin syytä myös tutkia projektin työmääräarvioita olettaen, että alusta ei olisi koskaan muuttunut ja siirtotehtävä olisi onnistunut. Tässä luvussa tarkastellaan, kuinka hyvin vaatimukset ymmärrettiin.

Arvioiden pohjana käytettiin asiakkaan toimittamaa vaatimuslistaa. Alkuperäinen vaatimuslista ei kuitenkaan ollut täysin yhteensopiva käyttöliittymämäärittelyn kanssa. Vaatimuslistalta puuttui useita käyttöliittymämäärittelyssä mainittuja uusia ominaisuuksia. Toisaalta osa vaatimuksista oli kuvattu niin yleisellä tasolla, että oli vaikeata arvioida, mitä kaikkia yksityiskohtia vaatimuksen toteuttaminen käytännössä vaatii. Arvioita tehtäessä toki kartoitettiin määrittelyjen ja vaatimuslistan ristiriitoja, mutta ei tarpeeksi.

Luvussa 5.3 puhuttiin määrittelydokumenttien laadusta ja siitä, kuinka käyttöliittymämäärittelyn kuvat olivat epämääräisiä. Kun tehdään työmääräarviota perustuen määrittelyyn, jossa kokonaisia ohjelmiston näkymiä on kuvattu ainoastaan suuntaa antavasti, olisi hyvä pyytää tällaisiin määrittelyjen osiin tarkennuksia, ennen kuin lopullinen arvio tehdään. Vaihtoehtoisesti työmääriin kannattaisi lisätä reilusti puskuria, jotta epäselvyyksien selvittämiseen riittäisi aikaa. Täytyy muistaa, että ohjelmoija ei ole graafinen suunnittelija. On paljon helpompaa ja nopeampaa työskennellä, jos vaatimukset ovat selkeät ja yksiselitteiset. Pahimmillaan tilanne oli se, että jonkin käyttöliittymäkokonaisuuden työmääräarvioksi oli laskettu 0,5 päivää, kun

lopulta sen valmiiksi saaminen kesti kaksi viikkoa. Arvio olisi voinut olla onnistuneempi, jos arvioinnit olisi tehty tapauspohjaisesti, jolloin huonoimman tapauksen perustaksi olisi voinut ottaa epäselvät vaatimukset ja valmiiden käyttöliittymäkomponenttien saatavuuden. Myöhästyminen voidaan osittain perustella kehittäjien kokemattomuudella: Symbian S60 -käyttöliittymäkehitys ei ole helppoa, varsinkaan jos ei voi käyttää valmiita käyttöliittymäkomponentteja. Onnistuneissa arvioissa tämäkin olisi otettu huomioon.

Koska ohjelmisto toteutettiin valmiin perusohjelmiston päälle, tehtiin perusohjelmiston ominaisuuksiin monia pieniä muutoksia. Usein perusohjelmiston ominaisuuksia käytettiin kuitenkin sellaisenaan. Käyttöliittymämäärittelyssä oli useita kohtia, joissa jonkin ominaisuuden oli kerrottu toimivan ”samoin kuin perusohjelmistossa”. Ohjelmistossa oli esimerkiksi käyttöliittymänäkymä, jonka ulkonäköön ei ollut tarkoitus tehdä mitään muutoksia. Tietosisältö tälle näkymälle jouduttiin kuitenkin tietyissä tapauksissa muodostamaan toisin kuin perusohjelmistossa. Tästä johtuen lauseen ”toimii samoin kuin perusohjelmistossa” taakse oli piilotettu ylimääräistä työtä, jota ei välttämättä osattu ottaa alkuperäisissä arvioissa millään lailla huomioon.

Arvioiden epätarkkuudet selittyvät osittain myös sillä, että useita toisistaan riippumattomia arvioita ei tehty eikä arvioiden vertailuja siis suoritettu tarpeeksi. Suunnittelupokeri-istunto projektin alkuvaiheessa olisi voinut olla hyvä vaihtoehto. Jopa ilman koodin siirtotehtävää arviot sisälsivät nyt liikaa puutteita.

6.2.4 Arvioiden tekemiseen käytetty aika

Arvioiden tekemiseen käytettiin todennäköisesti liian vähän aikaa. Ensimmäiset arviot valmistuivat reilussa päivässä. Kun työmäärät myöhemmin tulivat uudelle suunnittelijalle arvioitavaksi, aikaa annettiin muutama päivä. Kyseisellä suunnittelijalla oli edellinen projekti vielä kesken, joten aikaa paneutua uuden projektin työmääräarviointiin ei ollut tarpeeksi. Kiire tarjouksen toimittamiseksi saattoi olla myös osasyynä siihen, että projektille kohtalokkaaksi osoittautunut koodialustan muutos jätettiin huomiotta.

6.2.5 Osaamisen huomiointi

Vaikka arvioinnin alussa oli tiedossa, että projektiin kiinnitetään toiseksi kehittäjäksi harjoittelija, ei asiaa otettu työmääräarvioissa riittävästi huomioon aikataulua pidentävänä tekijänä. Harjoittelijan mukanaolo ei vaikuttanut lainkaan ensimmäisen työmääräarvion aikatauluihin. Myöhemmin projektipäällikkö tosin lisäsi virstanpylväiden perään aikaa ylimääräiselle työlle. Projektin onnistumisen kannalta osaamisen huomiointi ei ollut kriittisin ongelma. Asia olisi kuitenkin voitu ottaa huomioon esimerkiksi lisäämällä harjoittelijalle osoitettuihin alkupään tehtäviin lisäaikaa. Tämä jo siksikin, että projektissa käytettävien työkalujen käytön oppiminen vei aikaa.

6.3 Uudet vaatimukset

Yksi tavallisimmista syistä ohjelmistoprojektien myöhästymisiin on kesken projektin määriteltävät uudet ominaisuudet. Tämänkin projektin aikana toteutettiin suuri joukko ominaisuuksia, joita ei alkuperäisissä dokumenteissa ollut määritelty. Nämä voidaan jakaa karkeasti kahteen osaan:

1. Ohjelman toimintaan tehdyt pienet lisäykset ja korjaukset, jotka luvattiin toteuttaa ilman erillistä korvausta.
2. Isommat kokonaisuudet, joille tehtiin työmääräarvio ja määriteltiin hinta.

Kohdan 2 mukaiset uudet ominaisuudet olivat pienempi ongelma, koska ne otettiin selkeästi esille uusina asiakastarpeina, päivitettiin projektisuunnitelmaan ja ennen kaikkea, niistä sovittiin ja maksettiin korvaus. Toki nämäkin ominaisuudet teettivät lisätyötä jo muutenkin kiireelliseen projektiin suunnitteluineen ja toteuttamisineen. Kohdan 1 mukaiset ominaisuudet olivat kuitenkin ongelmallisempia jo pelkästään siitä syystä että niitä oli lukumääräisesti enemmän. Yksittäisen pienen korjauksen tai muutoksen tekeminen ei tuota paljon työtä, ja monessa tapauksessa olikin järkevämpää luvata tehdä muutos ilman erillistä korvausta. Näin vältettiin turhaa byrokratiaa ja ylläpidettiin asiakastyytyväisyyttä (projektin aikana asiakas oli valmis joustamaan tietyissä asioissa, joten toimittaja osasi myös joustaa tarvittaessa). Joissakin tapauksissa kävi kuitenkin niin, että pienten ominaisuuksien työmäärä kasvoi oletettua

suuremmaksi. Asioita, joita alkuperäisissä määrittelyissä ei ollut otettu huomioon lainkaan tai tarpeeksi, oli lukuisia. Näistä esimerkkeinä voidaan mainita:

1. Erilaiset käyttäjälle näytettävät viestit tai dialogit, esimerkiksi jos käyttäjän hakemaa tietoa ei löydy tai jos käyttäjä joutuu odottamaan jonkin toiminnon tapahtumista.
2. Toiminta, kun puhelimessa on tai ei ole muistikorttia, tai jos muistikortti irrotetaan kesken käytön.
3. Laitteistoriippuvaiset asiat: asiakas testasi laitteilla, jotka eivät sisältäneet kaikkia ohjelmistossa käytettyjä laitteisto-ominaisuuksia. Näiden ominaisuuksien automaattisen poiskytkemisen toteutus.

Hyvä määrittelydokumentti määrittelee kaikki tarvittavat asiat selkeästi. Tästä voidaan siis päätellä, että huono dokumentti ei määrittele kaikkia asioita selkeästi tai lainkaan. Parhaiten huonon määrittelydokumentin voisi kuitenkin kuvata seuraavasti: Huono määrittelydokumentti antaa lukijalle tunteen, että kaikki tarvittava on määritelty, vaikka se todellisuudessa sisältääkin aukkoja. Asiakkaan toimittamissa määrittelydokumenteissa oli jonkin verran tällaisia aukkoja. Näiden aukkojen taakse piiloutui yllättävän paljon lisätyötä, joita työmääräarvioissa ei osattu ottaa huomioon. Tällaisten puutteiden tunnistaminen on hyvin vaikeaa. Usein ne huomataan vasta, kun ominaisuuksia ollaan jo toteuttamassa. Jos projekti aiotaan saada ajoissa valmiiksi, pitäisi määrittelydokumentteja kuitenkin tutkia kriittisemmin. Avain tähän lienee runsaampi ajankäyttö projektin työmääräarvioinnissa.

6.4 Johtopäätökset

Projekti olisi voinut onnistua paremmin, jos tietyt asiat olisi otettu paremmin huomioon. Tässä esitetään kertausluontoisesti neljä asiaa, joita noudattamalla olisi voitu päästä parempaan lopputulokseen.

1. Varaa tarpeeksi aikaa työmääräarvioiden tekemiseen.
2. Anna useamman kuin yhden kehittäjän tehdä itsenäiset työmääräarviot.
3. Etsi työmääräarviointivaiheessa systemaattisesti määrittelydokumenteista puutteita, jotka tulisivat projektin kuluessa kasvattamaan aikataulua.
4. Tutki kaikkien muutosten vaikutus kokonaisaikatauluun. Reagoi muutoksiin mahdollisimman nopeasti ja tee uudet työmääräarviot ja aikataulut.

7 Loppusanat

Ohjelmistoprojektin työmääräarviointi on vaikeaa, eikä ole mitenkään harvinaista, että ohjelmistoprojekti myöhästyy alkuperäisestä aikataulusta. Usein käy myös niin, että projekti ei koskaan tule valmiiksi. Yhdeksi syyksi on esitetty ohjelmistotekniikan nuoruutta teollisuudenalana, jonka vuoksi alalle ei ole määritelty tarkkoja toimintamalleja ja standardeja. Kovin kauaa tällä ei enää voida selittää epäonnistuneita projekteja, sillä ohjelmistoja on tehty jo suhteellisen kauan ja tällä hetkellä kesken olevien ohjelmistoprojektien määrä maailmassa on valtava. Epäonnistuneet ohjelmistoprojektit ovat luoneet tarpeen kehittää koko ajan uusia menetelmiä menestyksekkäiden projektien aikaansaamiseksi. Tärkeintä olisi, että kaikista projekteista, onnistuneista ja epäonnistuneista, opittaisiin ja että projektit toimivan ohjelmiston lisäksi tuottaisivat riittävän tarkat dokumentit myöhempää käyttöä varten.

On aina helppoa olla jälkiviisas. Toisaalta jälkiviisauteen on pakko syyllistyä analysoitaessa ongelmallista projektia. Esimerkkiprojektista on löydettävissä monia klassisia ohjelmistotuotannossa esiintyviä ongelmia. Jokainen niistä on kuvattu kirjallisuudessa moneen kertaan. Lienee kuitenkin tosiasia, että kirjoja lukemalla ei opita minkään alan ammattilaisiksi. Usein oppiminen tapahtuu kantapään kautta. Lisäksi ohjelmistotuotannossa sattumalla ja tuurilla on myös oma roolinsa. Esimerkkiprojekti olisi voinut onnistua. Näin luultavasti olisi käynyt, jos asiakas ei yllättäen olisi vaihtanut koodialustaa toiseksi. Jos tähän muutokseen olisi reagoitu ja tehty projektin uudelleenarviointi, kustannukset olisivat luultavasti kasvaneet niin suureksi, että projektia ei mahdollisesti koskaan olisi Sasken Finlandilla edes aloitettu, ainakaan samansisältöisenä.

Lähteet

1. McConnell, Steve 2002. Ohjelmistotuotannon hallinta. Edita Publishing Oy.

2. McConnell, Steve 2006. Software Estimation, Demystifying the Black Art. Microsoft Press.

3. Haikala, Ilkka, Märijärvi, Jukka 2002. Ohjelmistotuotanto.Talentum Media Oy.

4. Planning Poker in Detail [www-sivu]. [viitattu 4.4.2009]. Saatavissa: http://www.planningpoker.com/detail.html

5. Ketterät käytännöt [www-sivu]. [viitattu 12.4.2009]. Saatavissa: http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/

6. Web-palveluiden käytettävyys ja tuotanto [www-sivu]. [viitattu 12.4.2009]. Saatavissa: http://www.uiah.fi/mediastudio/survey4/24.html