Prosessimallit ja ohjelmistoprojektin onnistuminen toimittajan näkokulmasta

Sampo Karjalainen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Toukokuu 2008

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Sampo Karjalainen: Prosessimallit ja ohjelmistoprojektin onnistuminen toimittajan näkokulmasta
Pro gradu -tutkielma, 59 sivua
Toukokuu 2008

Tiivistelmä

Ohjelmistoprojektien (kehitys- ja käyttöönottoprojektit) onnistunut läpivieminen vaatii ennustettavaa, systemaattista ja mitattavaa työtä. Tässä tutkielmassa sovitetaan useita tietojärjetelmähankkeissa käytettyjä prosessimalleja saman viitekehyksen alle. Tämän jälkeen esitellään tietojärjestelmän onnistumiseen liittyvät käsitteet ja luodaan mittaristo kenttätukimuksessa vertailtaviin projekteihin. Kenttätutkimuksessa esitellään kolme projektia (luonnollista koetta) ja arvioidaan projektien onnistumista aikaisemmin esitetyn mittariston mukaan. Tutkimuksen tärkeimmät tulokset ovat ohjelmistoprojektien prosessimallien sovittaminen samaan viitekehykseen, menetelmä projektien työmäärien ennustettavuuden parantamiseen ja kolme yksinkertaista periaatetta, joilla tietojärjestelmäprojektin onnistumisen todennäköisyyttä voidaan parantaa.

Sisällys

Kuvat

Kiitokset

Kiitos perheelle ja sukulaisille tuesta ja painostamisesta tutkielman loppuun saattamiseksi. Täydellistä ohjelmistokehityksen prosessimallia odotellessa muistelkaamme, mitä Platon aikanaan sanoi: *Sellainen mielikuva minulla siis on, että ajatuksella tavoitettavassa maailmassa on viimeisenä ja vain vaivoin nähtävissä hyvä idea.*

1 Johdanto

Ohjelmistoprojekteissa luodaan uusia ohjelmistoja ja muokataan valmiita sovelluksia sopimaan asiakkaiden tarpeisiin. Molemmissa lähetysmistavoissa syntyy uutta ohjelmakoodia ja sitä kautta uusia ja erilaisia ohjelmia. Koska yritysten toiminta on yhä enemmän sidoksissa ohjelmistoihin, pitää ohjelmistojen olla laadukkaita ja toimintavarmoja. Tämän takaamiseksi ohjelmistoja valmistettaessa ja käyttöönotettaessa tulisi käyttää ennakoitavia hyväksihavaittuja toimintatapoja eli prosessimalleja.

Ohjelmistomarkkinat muuttuvat koko ajan, ja markkinoilla pysyäkseen täytyy toimittajayrityksen pystyä toimimaan nopealla toimitusaikataululla ja tuottamaan asiakkaan tarpeita vastaavia tuotteita. Markkinoilla pysyminen edellyttää myös kokonaisvaltaista panostamista laatuun, joka voidaan jakaa kolmeen osatekijään: tuotteen, prosessin ja resurssin laatuun. Tuotteen laadulla tarkoitetaan tuotetun ohjelman käytettävyyttä, ylläpidettävyyttä ja toimintavarmuutta. Ohjelmiston teossa käytettyä toimintatapaa eli prosessia mitataan vastaavasti prosessin laadulla. Kolmantena laadun mittarina ovat ohjelmiston resurssit. Henkilökunnalla tulee olla ajanmukaiset työvälineet ja työhön tarvittava koulutus. [Nevalainen, 1999]

Ohjelmistoprojekti voi olla joko talon sisäinen kehitysprojekti tai se voidaan ulkoistaa erilliselle yritykselle, jonka toimialana on ohjelmistoteollisuus. Molemmissa lähestymistavoissa prosessimallin valinta ja ylipäätänsä sen käyttäminen ovat tärkeitä asioita. Ulkoistetussa mallissa roolit ja tehtävät jaetaan asiakas- ja toimittajayrityksen kesken. Prosessimallilla viitekehyksessä kuvataan karkeasti ottaen seuraavat aktiviteetit: kommunikaatio, suunnittelu, mallintaminen, rakentaminen ja käyttöönotto. [Pressman, 2005]

Oikean prosessimallin valitseminen on tärkeää, ja sen tulee perustua syntyvän ohjelmistotuotteen vaatimuksiin. Prosessimallin valintaan vaikuttavat myös asiakkaan vaatimukset, organisaatio, käytettävissä olevat resurssit ja aikataulu. Yleisesti on tiedossa, että sovelluksia voidaan varmasti rakentaa millä tahansa olemassa olevalla prosessimallilla tai jopa ilman mitään toimintatapaa. On tärkeää selvittää, minkä tekijöiden avulla on mahdollista vertailla ja valita oikea lähestymistapa.

Tietojärjestelmähankkeen onnistumiseen vaikuttavat tekijät ovat moniulotteisia ja riippuvat toisistaan. Jotta ohjelmistoprojekti voidaan viedä onnistuneesti läpi, on tärkeää tuntea sekä mittaamisen käytettävissä olevat menetelmät että projektin tavoitteiden kartoittaminen. Onnistumismalli antaa hyvän kuvan

tietojärjestelmän onnistumisen mittaamisesta [Delone & McLean, 1992; Delone & McLean, 2003]. Tietojärjestelmän vaikuttavuuden tunteminen helpottaa tavoitteiden asettamisessa [Grover *et al.*, 1996]. Lisäksi analysoimalla tietojärjestelmän mahdollisuuksia voidaan luoda järkeviä tavoitteita ohjelmistoprojekteille [www.lawson.com, 2008].

Tässä tutkielmassa on tarkoitus testata olemassa olevaa teoriaa kenttäkokeiden avulla ja tehdä vertailevaa tutkimusta eri prosessimallien hyvistä ja huonoista puolista. Lisäksi tutkimuksessa hyödynnetään tutkijan omaa kokemusta tietojärjestelmätoimittajan näkökulmasta. Tutkimuksessa pyritään myös luomaan yksinkertaiset periaatteet, joilla tietojärjestelmähankkeen onnistumisen todennäköisyys paranee.

2 Tutkimuksen tausta ja menetelmä

2.1 Johdanto

Olen erittäin kiinnostunut suoraviivaistamaan toimintapoja ja hyödyntämään uudelleenkäytettäviä prosessimalleja työssäni. Yliopistourani alkuajoista lähtien olen työskennellyt osa-aikaisesti ja hieman myöhemmin päätoimisesti saman yrityksen palveluksessa. Lawson (Lawson Software, Inc.) on toiminnanohjausjärjestelmiä toimittava niin sanottu täyden palvelun ohjelmistotalo, joka toimittaa tuotteen lisäksi konsultoinnin (palvelu), ylläpidon (tuki) ja asiakkaiden tarpeiden mukaan tehtävät ohjelmistolaajennukset sekä jatkokehityshankkeet.

Vuonna 2004 sain lisää vastuuta ottamalla haltuuni integraatio-tiimin työnohjauksen. Viimeistään silloin silmäni aukesivat ja huomasin, että toimeksiantoja voisi tehdä suoraviivaisemmin ja tehokkaammin. Alussa suurin ongelma oli se, että konsulttien ei ollut mahdollista keskittyä riittävästi meneillään olevaan projektiin, vaan samaan aikaan piti hoitaa useiden asiakkaiden toimeksiantoja. Lisäksi käytössä olevat prosessit eivät täysin tukeneet konsulttien jokapäiväistä työskentelyä. Prosessien parannus on vaativaa jatkuvaa työtä, jota teemme koko ajan luodaksemme hyvän ympäristön laadukkaalle ja tehokkaalle palvelutyölle. Kuluneiden vuosien aikana olemme päässeet ison askeleen eteenpäin ja oppineet paljon.

Vuonna 2005 siirryin Lawsonin Suomen toimintayksikön johtoryhmään ja vastaamaan Suomen teknisestä liiketoiminnasta. Tästä seuraavat vuodet ovat menneet Lawsonin globaalien prosessien soveltamisessa suomalaiseen liiketoimintaan ja tietysti jokapäiväisen liiketoiminnan pyörittämiseen. Työni on antanut minulle aitiopaikan parametroitavan ohjelmiston myymiseen ja käyttöönottamiseen liittyviin haasteisiin.

Teknisessä konsultoinnissa on kaksi tärkeää asiaa: keskity yhteen asiaan kerrallaan ja pyri tekemään tehtävät aina samalla tavalla. On huomattavasti helpompi parantaa olemassa olevaa huonoakin toimintatapaa kuin todeta, että mitään yhtenäistä tapaa ei ole. Siksi prosessien ja toistettavien käytäntöjen hyödyntäminen on elintärkeää.

Tässä tutkimuksessa haluan esittää lukijalle, minkälaisia ohjelmistokehityksen prosessimalleja on olemassa, miten ne liityvät ohjelmistoprojekteihin ja miten tietojärjestelmäprojektin onnistumista voidaan ennustaa ja mitata. Lisäksi tarkoitukseni on osoittaa, että ohjelmistokehityksen prosessimallit eivät juuri eroa parametroitavan tuotteen (esim. toiminnanohjausjärjetelmät) käyttöönottomal-

leista.

2.2 Tutkimustavoite

Tämän tutkimuksen tavoite on selvittää, miten arvioida **toimittajan** palvelutyön onnistumista ohjelmiston käyttöönotto- tai jatkokehitysprojektissa. Tutkielmassa ei oteta kantaa projektin myötä käyttöönotettavan tuotteen laatuun ja sen tuomaan käyttäjäkokemukseen. Palvelutyön laatu on yksi tärkeimmistä tietojärjestelmähankkeen onnistumisen kriteereistä [Delone & McLean, 2003, 18]. Oman kokemukseni ja lähdekirjallisuuden perusteella arvioin työssäni seuraavia tutkimuskysymyksiä toimittajan näkökulmasta:

1. Mitä eri ohjelmiston kehitysmalleja voidaan hyödyntää ja voidaanko ne asettaa yhteisen viitekehyksen alle?

2. Miten projektin onnistumista voi mitata?

3. Millä yksinkertaisilla lauseilla voidaan kiteyttää toimittajan edellytykset onnistuneelle tietojärjestelmäprojektille?

Pääsääntöisesti liiketoiminta rahoittaa tietojärjestelmähankkeet, ja siitä johtuen myös hankkeiden onnistumista arvioidaan usein liiketoiminnan näkökulmasta. Tämä onkin suurelta osin vaikuttanut siihen, että suosittuja onnistumisen mittareita ovat sijoituksen takaisinmaksuaika (*Return on Investment*), liikevaihdon lisääntyminen, kustannusten vähentäminen ja liiketoiminnan strategiset muutokset [Delone & McLean, 2003, 19–20]. Vähintään yhtä tärkeä mittari ohjelmiston toimittajalle on asiakastyytyväisyys. Sen parantaminen on mukana usean tietojärjestelmätoimittajan strategiassa. Lawson teettää vuosittain asiakastyytyväisyystutkimuksen, mutta tässä tutkielmassa sitä ei huomioida empiirisesti, koska käytettävissä oleva aineisto ei yksilöi asiakkaita eikä projekteja.

Tässä tutkielmassa toimittajan näkokulmalla ei tarkoiteta sitä, että kyseessä olisi toimittajan kannattavuuden tai toiminnan tehostamisen edistämistä tukeva tutkimus. Tarkoitus on hyödyntää toimittajan roolissa kerättyä käytännön tietoa ja tutkia siihen pohjautuen projektien onnistumista projektimallin, työmäärä-budjetin ja tavoitteen kannalta

2.3 Tutkimusmenetelmä ja -aineisto

Tutkimukseni testaa ohjelmistokehityksen prosessimalleja ja projekteja kenttämenetelmien avulla. Tutkimuksessani testattava muuttuja on tietojärjestelmän käyttöönottomalli. Tätä muuttujaa testataan kolmella luonnollisella kokeella, joissa kohde ei ole tietoinen koetilanteesta, eikä tutkijalla ole täydellistä kontrollia tutkimusympäristöön. [Järvinen & Järvinen, 2004, 55-58]

Työstän samaan aikaan tutkielmaani tutkijana ja päätoimisesti olen osaltani johtamassa tutkittavaa organisaatiota. Lallé [2003] esittää toimija-tutkija -käsitteen (*actor-researcher*), jonka mukaisesti työskentelyn olen aloittanut vuonna 2004, kun mielenkiintoni projekteissa käytettäviä prosessimalleja ja niiden parantamista kohtaan heräsi. Toimija-tutkijan rooli on selvittää organisaatiossa syntyvää hiljaista tietoa, toimintatapoja ja käytäntöjä. Tämän tiedon pohjalta pystytään päivittämään olemassa olevia teorioita ja mahdollisesti luomaan uutta teoriaa.

Colquitt ja Zapata-Phelan [2007, 1282-1286] esittelevät luokitusmallin tutkimuksille, jotka testaavat teorioita ja luovat uutta teoriaa. Heidän luokittelunsa mukaan tämä tutkimus kuuluu testaaja-luokkaan (*tester*). Viitekehyksen avulla olemassa olevaa teoriaa yhteismitallistetaan ja vertaillaan. Lisäksi tulosten pohjalta luodaan uutta teoriaa kolmen ohjaavaan lauseen avulla.

Tutkielmani alkaa teoriaosuudella, jossa kartoitan ensin ohjelmistokehityksessä käytettäviä prosesseja ja sitä kautta siirryn laajempiin prosessimalleihin, koska ne sopivat paremmin toiminnanohjausjärjestelmien käyttöönotto- ja jatkokehitysprojektien malleiksi. Prosessimallien jälkeen kartoitan tietojärjestelmähankkeiden onnistumisen arviointimenetelmiä ja periaatteita.

Käytännön osuudessa esittelen kolme projektia (luonnollista koetta) ja arvioin niitä hyödyntäen sitä hiljaista tietoa, jota olen kerännyt työpaikallani seitsemän vuoden ajan toimiessani eri tehtävissä. Projekteista esitän laskutusaineistoon perustuvia lukuja, joiden avulla on mahdollista analysoida tehtyjen tuntien määrää verrattuna projektin budjettiin, tehtyjen ilmaisten tuntien määrää ja mahdollisten reklamaatoiden määrää. Projektien onnistumista arvioin kolmen osa-alueen perusteella: tutkielmassa esittämäni teorian, tavoitteeseen pääsemisen ja laskutusaineiston pohjalta.

2.4 Tutkimuksen rakenne

Luvussa 3 esitetään ohjelmistoprojekteihin liittyvät peruskäsitteet ja luku päättyy yleiseen ohjelmistoprosessimalliin. Tämä malli toimii tutkielmassa viitekehyksenä, kun luvuissa 4 ja 5 yhteismitallistetaan lähdekirjallisuudesta tutkielmaan mukaan otetut prosessimallit. Luvussa 6 keskitytään tietojärjestelmähankkeiden onnistumisen mittaamiseen ja keinoihin myös etukäteen vaikuttaa hankkeen onnistuneeseen läpivientiin. Seuraavat luvut 7, 8 ja 9 havainnollistavat teoriaa käytännössä. Niissä esitellään tutkielman luonnollisina kokeina toimineet käyttöönottoprojektit. Viimeinen 10. luku yhdistää projekteista tehdyt havainnot ja teorian tutkielman päätelmiksi.

3 Ohjelmisto, projekti ja viitekehys

Ohjelmistoprosessimallien ymmärtäminen edellyttää, että lukija tuntee taustalla olevat käsitteet. Tässä luvussa käydään läpi tutkielman kannalta olennaiset ohjelmistoteollisuuden yleiset käsitteet ja yleinen ohjelmistoprosessimalli. Tätä mallia hyödynnetään viitekehyksenä läpi koko tutkielman, kun perehdytään eri prosessimalleihin ja niiden eroihin.

3.1 Ohjelmistotuote ja -teollisuus

Ohjelmistoteollisuus vaikuttaa, ainakin välillisesti, useimpien ihmisten arkeen teollistuneissa maissa. Aikaisemmin ohjelmat olivat hakkereiden ja ”taitelijoiden” luomuksia, mutta nykyään tilalle ovat tulleet ohjelmistoasiantuntijatiimit. Jokainen tiimi keskittyy omaan osuuteensa suuresta monimutkaisesta ohjelmistotuotteesta. Jotkut asiat ovat kuitenkin pysyneet ennallaan. ”Taiteilijalta” kysyttiin samat kysymykset kuin nykypäivän yrityksiltä:

- Miksi ohjelmistojen tekeminen kestää niin kauan ja kehityskulut ovat valtavan suuret?
- Miksi valmiit tuotteet sisältävät vielä virheitä?
- Miksi ohjelmien ylläpitäminen maksaa niin paljon?
- Miksi ohjelmistokehitystä ei pystytä mittaamaan?

Nykyään moni osaa määritellä ja ymmärtää, mitä ohjelmistolla tarkoitetaan. Pressman [2005, 36] kirjoittaa, että sanakirjan määritelmä ohjelmistolle voisi olla: *”Ohjelmisto koostuu (1) toiminnallisuuksista (tietokoneohjelmista), jotka ajettaessa tarjoavat haluttuja ominaisuuksia, toimintoja ja suorituskykyä; (2) informaatioita käsittelevistä tietorakenteista; ja (3) dokumenteista, joilla kuvataan ohjelmiston toimintaa ja käyttöä.”* [Pressman, 2005, 35-37]

Ohjelmistolle voisi varmasti antaa monimutkaisemmankin muodollisen määritelmän, mutta se ei ole tärkein asia ymmärtääksemme ohjelmistokehitystä. Kehitystyön ymmärtäminen vaatii tutustumista ohjelmiston ominaispiirteisiin, koska ne poikkeavat muista ihmisen rakentamista tai valmistamista asioista. Silta, laiva ja avaruusalus ovat kaikki käsin kosketeltavia ihmisen tuottamia valmisteita. Ohjelmisto on looginen sovellus ja eroaa tämän takia perinteisistä tuotteista.

Ohjelmistoa ei valmisteta, vaan se kehitetään. Ohjelmistosta ei löydy sellaisia laatuongelmia kuin valmistettavissa tuotteissa voi olla. Näitä voivat olla esimerkiksi virheellinen raaka-aine. [Pressman, 2005, 37]

Laitteistoissa ongelmia aiheuttaa niiden kuluminen. Tätä ongelmaa ei ole ohjelmistopuolella, koska siinä ei ole kuluvia osia [Curtis *et al.*, 1987]. Ohjelmistosta löytyy toki virheitä, mutta ne eivät lisäänny iän myötä. Tosin jotkut virheet paljastuvat vasta käyttöympäristön muuttuessa. Ohjelmistojen korjaaminen aiheuttaa monesti uusia virheitä, ja tästä johtuen ohjelmisto saattaa hajota käsiin (virheiden korjaaminen aiheuttaa uusia virheitä).

Valmistavassa teollisuudessa tuotteet tehdään suurelle joukolle samalla muotilla, eikä yksittäisen tuotteen räätälöintiä tapahdu. Ohjelmistokehityksessä on melkein aina kyse räätälöidystä sovelluksesta. Ohjelmistokehitys on pikku hiljaa siirtynyt kohti uudelleenkäytettäviä komponentteja, mutta hyvin harvoin pystytään rakentamaan uusi tietojärjestelmä varastossa olevista ohjelmistokomponenteista. [Pressman, 2005, 39]

Ohjelmistotuotteet ovat usein monimutkaisempia kuin perinteiset tuotteet. Lisäksi ohjelmistoyrityksiin on tarjolla niukasti taitavia johtajia, koska kyseessä on suhteellisen uusi toimiala. Ohjelmistoa ei voi rakentaa vakaiden luonnonlakien varaan, vaan se on ihmisen kekseliäisyyden (ja sen tuotosten) varassa. Ennen kaikkea ohjelmiston on pystyttävä muuttumaan sen elinkaaren aikana. [Humphrey, 1989]

Edellä oleva Humphreyn näkemys ohjelmistotuotteista on lähes 20 vuotta vanha, mutta se kuvaa edelleen hyvin ohjelmistoteollisuuden haasteita. Laadukas ohjelmistotuote syntyy ennen kaikkea ihmisten hyvän yhteistoiminnan tuloksena. Lisäksi mahdollisuus mukautua nopeasti liiketoiminnan aiheuttamiin muutoksiin on nykypäivänä hyvän ohjelmistotuotteen edellytys.

3.2 Ohjelmistoprojekti

Projektipäällikön on otettava huomioon, että ohjelmistokehitystyö (eli systeemityö) vaatii ihmisiltä kovaa ponnistelua eikä ole ainoastaan tekninen ratkaisu. Hyvä projektinhallinta takaa paremman laadun ohjelmistolle, mutta onnistunutta ohjelmistoa se ei kuitenkaan voi taata. Ohjelmistoprojektin läpivieminen edellyttää keskittymistä neljään eri asiaan: ihmisiin, tuotteeseen, prosessiin ja itse projektiin. [Pressman, 2005]

3.2.1 Ihmiset

Ohjelmistoteollisuudessa henkilökuntaa pidetään tärkeimpänä edellytyksenä onnistuneisiin projekteihin. Tämän takia monet toimialan yritysjohtajat pitävät tärkeänä neljää asiaa: rekrytoidaan oikeat henkilöt, koulutetaan osaavia henkilöitä, mahdollistetaan osaamisen ylläpitäminen ja annetaan osaaville henkilöille toimiva tuotantoympäristö.

Jokaisessa ohjelmistoprojektissa (ja samalla ohjelmistoprosessissa) on mukana henkilöitä viidestä eri sidosryhmästä. Ylemmän tason johtajat määrittelevät liiketoimintakysymykset, joilla saattaa olla suuri merkitys projektin kannalta. Projektipäälliköt organisoivat ja jakavat tehtävät. Tekniset osaajat toimittavat tarvittavan tietämyksen tuotteen kehittämiseksi. Lopuksi tulevat asiakkaat ja loppukäyttäjät. Onnistuneessa projektissa myös kahden viimeisen ryhmän edustajat ovat omistautuneita ja innostuneita meneillään olevasta hankkeesta.

Systeemityössä kehitystiimin rakenne ja toimintamalli vaikuttavat projektin etenemiseen, mukana olevaan henkilökuntaan ja käytettävään prosessimalliin. Käytössä on yleisesti paljon perinteisiä tiimirakenteita, jotka tukeutuvat organisaatiorakenteisiin. Viime aikoina pienet ja erittäin motivoituneet ohjelmistotiimit (*agile teams*) ovat kasvattaneet suosiotaan, koska niiden uskotaan pystyvän vastaamaan systeemityössä toistuviin ongelmiin. Tiimin rakenteen tulisi tukea ja pitää käytössä hyviä kommunikointikanavia koko projektin ajan. [Pressman, 2005]

3.2.2 Tuote

Projektin alkaessa pitää määritellä, mitä tuotetta ollaan tekemässä. Määritys sisältää tiedon tuotteen vaikutusalueesta: Onko tuote osa isompaa kokonaisuutta? Mitä informaatiovaatimuksia tuotteella on? Mitä toimintoja tuotteessa on, ja miten suorituskykyisiä niiden tulee olla? Tuotteen vaikutusalueen tulisi olla yksiselitteinen ja ymmärrettävissä sekä asiakkaan, projektin johdon että teknisen osaston puolella. Tuotteen määritteleminen on tärkeää, koska ilman kunnollista määritystä ei voida rakentaa realistista projektisuunnitelmaa ja antaa järkeviä työmääräarvioita.

Kehitettävän tuotteen projektisuunnitelma on helpompi rakentaa pilkkomalla vaatimusmääritykset pienempiin osiin. Tämä ”hajoita ja hallitse” -taktiikka antaa hallitumman näkymän projektin osatekijöihin. Kun projektin palaset ovat selvillä, voidaan työmääräarviot esittää erikseen jokaista palasta kohti. Vaikka tuote olisi kokonaan uusi, voi sen osatekijöistä löytyä entuudestaan tuttuja kokonaisuuksia. Näiden arvioiminen on paljon helpompaa historiatiedon ansiosta. [Press-

man, 2005]

3.2.3 Prosessi

Ohjelmistoprosessimalleja on useita erilaisia, ja ne soveltuvat parhaiten eri tarkoituksiin. Projektista vastaavan onkin osattava valita oikea prosessimalli oikeaan paikkaan. Tähän vaikuttavat tietenkin ohjelmistoyrityksen historia ja yrityksen sisäiset prosessit. Jos aloitetaan puhtaalta pöydältä, yritys on vaihtamassa toimintamalliaan tai uusi projekti edellyttää vanhasta poikkeavaa mallia, vaikuttavat ainakin seuraavat asiat prosessimallin sopivuuteen:

1. asiakkaan esittämät vaatimusmääritykset
2. systeemityön tekevät henkilöt
3. kehitettävän tuotteen ominaispiirteet
4. projektin toimintaympäristö ja siinä toimiva ohjelmistotiimi.

Valittu prosessi määrittelee projektisuunnitelman rungon ja rakenteen. Jokainen ohjelmistotiimin luoma komponentti (ohjelman osa) luodaan tietyn toimintamallin mukaan. Tässä vaiheessa projektia sulautetaan *tuote* ja *prosessi* yhteen ja luodaan prosessin mukainen projektisuunnitelma.

3.2.4 Projekti

Ohjelmistoprojektin johtaminen hallitusti edellyttää, että projektipäällikkö ymmärtää, mitä asioita tulee välttää [Pressman, 2005]. Toisin sanoen projektipäällikön olisi hyvä tietää, mitkä asiat voivat mennä pieleen. John Reel [1999] esittää seuraavat väitteet tietojärjestelmäprojektien epäonnistumisesta:

1. Ohjelmiston tuottajat eivät ymmärrä loppukäyttäjän tarpeita.
2. Tuotteen vaikutusalue ei ole määritelty oikein.
3. Projektin aikana tapahtuvat muutokset hallitaan huonosti.
4. Teknologia-alusta muuttuu kesken projektin.
5. Liiketoiminnan tarpeet muuttuvat.

6. Tavoiteaikataulu on mahdoton toteuttaa.

7. Loppukäyttäjät eivät ole halukkaita hyväksymään uutta tietojärjestelmää.

8. Projektin rahoittaja keskeyttää tukemisen.

9. Projektin työntekijöillä ei ole tarvittavia taitoja.

10. Projektin johto ja osalliset eivät opi aikaisemmista kokemuksista.

Käytännössä nykypäivän kova kilpailu aiheuttaa lisäjännitettä ohjelmistoprojekteille. Useissa ohjelmistoyrityksissä myynti ja kehitys (tai palveluyksikkö käyttöönottoprojekteissa) ovat lähes kokonaan toisistaan irrallisia osastoja. Yrityksen pysyminen markkinoilla edellyttää uusien kauppojen syntymistä, ja tämän takia myynti ei pysty esittämään asiakkaalle totuudenmukaisia tarjouksia, vaan annetuista työmääräarvioista joudutaan usein karsimaan todenmukaista siistimpi versio asiakkaalle. Tämä aiheuttaa valitettavan usein projektin vääristymisen heti alkuvaiheessa, koska projektin toimittajapuolen johdolle saattaa jäädä harhaluulo, että myydyt työpäivät vastaavat alunperin sisäisesti arvioituja. Prosessimallin pitäisi pystyä tukemaan ohjelmistoyrityksen koko ketjua myynnistä ylläpitoon.

Reel [1999] esittää epäonnistumisen aiheuttavien syiden lisäksi ohjenuoria projektin johdolle: aloita oikealla jalalla, pidä liike käynnissä, seuraa edistymistä, tee järkeviä päätöksiä ja tee Mitä opimme tästä -analyysi. Ensimmäinen kohta tarkoittaa sitä, että heti projektin alusta alkaen luodaan realistinen käsitys tilanteesta ja varataan riittävästi resursseja. Lisäksi resursseille annetaan riittävästi liikkumavaraa ja aikaa projektin toteuttamiseksi. Toinen kohta viittaa siihen, että projektin aikana ei pidä sortua oikopolkuihin, vaan tulee ylläpitää laatu koko projektin ajan. Jos projektin edistymistä ei seurata ennalta sovituin mittarein, johdolla ei voi olla käsitystä, miten hyvin projekti on onnistunut. Neljäs ohje neuvoo tekemään järkeviä päätöksiä eli valitsemaan vähemmän riskialttiin vaihtoehdon aina, kun se on mahdollista. Jos on pakko toteuttaa korkean riskin vaihtoehtoja, lisätään niihin resursseja ja aikaa. Viides ja viimeinen ohje on ehkä kaikkein tärkein. Tutkitaan, miten onnistuttiin suhteessa projektissuunnitelmaan ja aikaisemmin tehtyihin vastaaviin projekteihin.

Liike-elämässä näistä ohjeista usein karsiutuvat pois mielestäni tärkeimmät kohdat, projektin seuraaminen ja jälkipyykin hoitaminen, koska ne eivät ole pakollisia tehtäviä yksittäisen projektin läpiviemisessä ja ne lisäävät kustannuksia, mutta pitkällä tähtäimellä niistä saadaan eniten hyötyä. Prosessien paran-

taminen lähtee liikkeelle juuri siitä, että tiedetään, missä mennään. Jokaisen ohjelmistoyrityksen tulisi pyrkiä parantamaan prosessejaan, jos se aikoo pysyä markkinoilla pitkään ja tuottaa tulosta. Prosessien parantumiseen voidaan päästä vain mittaamalla toimintaa ja vertaamalla sitä aikaisemmin kerättyyn historiatietoon [Humphrey, 1989]. Täytyy myös muistaa, että projektin seuranta antaa mahdollisuuden ennustaa uusien projektien työmääriä muun muassa tilastollisen analyysin avulla.

3.3 Yleinen ohjelmistoprosessimalli

Yleinen ohjelmistoprosessimalli voidaan jakaa viiteen eri aktiviteettiin. *Kommunikointi* (K) tarkoittaa projektiin liittyvien *ihmisten* yhteistoimintaa ja tiedonvaihtoa. *Suunnittelu* (S) sisältää resurssien varaamisen, riskien arvioimisen, tehtävien jaon ja aikataulun. *Mallintamista* (M) käytetään ohjelmistosuunnittelijoiden ja asiakkaiden välillä ohjelmiston suunnittelun ja vaatimusmääritysten tekemiseen. *Rakentaminen* (R) tarkoittaa varsinaista ohjelmointia ja koodin testaamista. *Käyttöönotto* (KO) tarkoittaa sovelluksen tai osan hyväksymistestiä ja siirtämistä asiakkaan ympäristöön tuotantokäyttöä varten. Kaikki viisi aktiviteettia löytyvät ohjelmistoprosesseista, ja niiden avulla eri prosessien vertaaminen on mahdollista. [Pressman, 2005, 52–57]

Ohjelmistoprosessien parantamiseen on olemassa ISO-standardi nimeltä SPICE (*Software Process Improvement and Capability dEtermination*). Tämä standardi on tarkoitettu prosessien laadun mittaamiseen ja sitä kautta parempien prosessien kehittämiseen. Tässä työssä ei keskitytä prosessien parantamiseen, mutta eri prosessimallien ymmärtämisen kannalta on hyvä tietää, että on jo olemassa maailmanlaajuisesti käytetty ohjelmistoprosessin viitekehys. [ISO, 1998; Nevalainen, 1999]

Prosessimallit voidaan jakaa kahteen pääryhmään: perusprosessimallit ja laajat prosessimallit. Perusprosessimallit keskittyvät vain tehtäviin ja niiden järjestykseen. Laajoissa prosessimalleissa käsitellään näiden lisäksi organisaation rakennetta, dokumentaatioita ja laadunvarmistamista. [Ludewig, 2004]

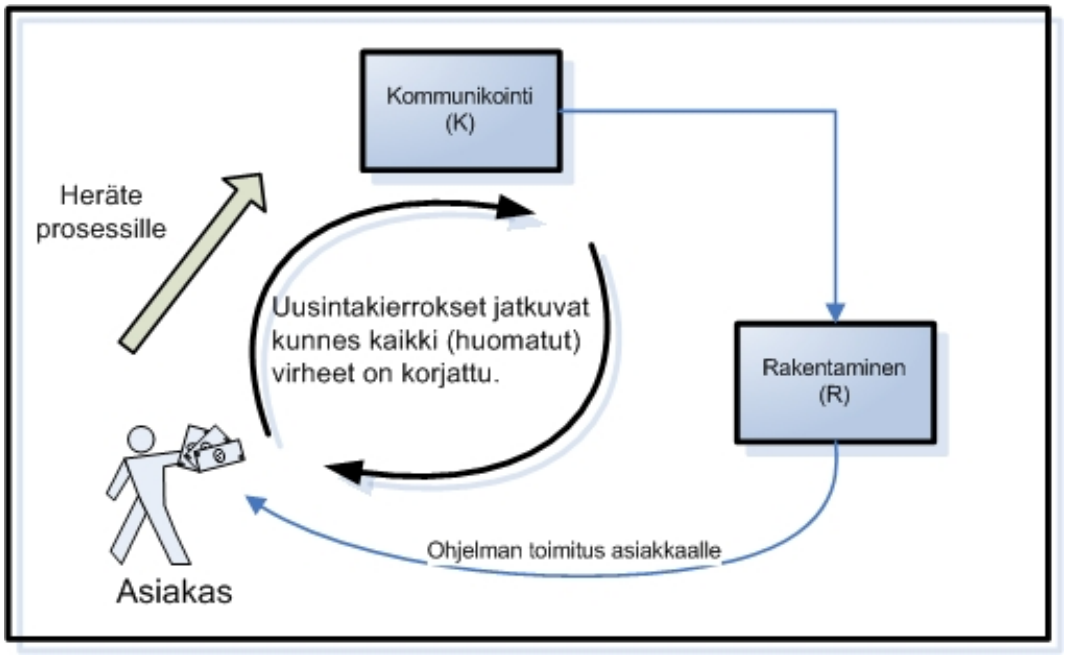
Tässä tutkielmassa käydään läpi perusprosessimallit, osa laajoista prosessimalleista ja tutkitaan, mitä ominaisuuksia eri prosessimalleilla on. Kenttätutkimuksessa mukana olleet projektit esitetään omissa luvuissaan ja analyysissa tutkitaan onnistumisen lisäksi, mitä prosessimallia projekteissa käytettiin ja olisiko eri prosessimallin valinta mahdollisesti parantanut projektia.

4 Perusprosessimallit

Neljännessä luvussa käyn läpi olennaisimmat perusprosessimallit ja sovitan ne kolmannessa luvussa esittelemääni viitekehykseen. Jokainen prosessimalli sisältää osajoukon viitekehyksen aktiviteeteista ja sitoo ne toisiinsa prosessin työnkulkukaavion avulla.

4.1 Koodaa ja korjaa -malli

Perinteinen ohjelmistokehitys tapahtuu koodaa ja korjaa -mallin mukaan. Sovelluskehitys aloitetaan suoraan ohjelmoinnista eikä suoriteta minkäänlaista systeemityötä. Kun tätä mallia verrataan viitekehykseen, huomataan, että *suunnittelu* (S), *mallinnus* (M) ja *käyttöönotto* (KO) -aktiviteetteja ei huomioida lainkaan. *Kommunikointi* (K) on mukana, mutta ainoastaan pakollisena työvälineenä ilman ennalta määrättyjä sääntöjä. Näillä säännöillä tarkoitetaan nimenomaan työnkulkukaaviota, joka ohjaa prosessin toimintaa. Koodaa ja korjaa -lähestymistapa on havainnollistettu kuvassa 4.1.



Kuva 4.1 Koodaa ja korjaa

Rakennusvaiheessa (R) koodi ohjelmoidaan suoraan ohjelmoijan ajatuksista, minkä jälkeen se käännetään lähdekoodista suoritettavaksi sovellukseksi. Samalla korjataan kääntäjän ilmoittamat syntaksivirheet. Ohjelmaa ajetaan muutaman

kerran ja havaitut virheet korjataan palaamalla koodausvaiheeseen. Kun virheitä ei enää ilmene, todetaan sovelluksen valmistuneen.

Tässä prosessimallissa on useita epäkohtia: Vaatimusmäärittelyjä ei ole olemassa, koska niitä ei ole tehty. Arkkitehtuurivaihtoehtoja ei ole mietitty. Koska vaatimusmäärittely puuttuu, sovellusta ei voi testata, eikä sitä voida ylläpitää (ohjelmoidusta koodista ei ole dokumentaatioita). Näistä syistä johtuen tämä prosessimalli on liian kallis yritykselle, jonka liiketoiminta perustuu ohjelmistokehitykseen. [Ludewig, 2004]

4.2 Lineaariset elinkaarimallit

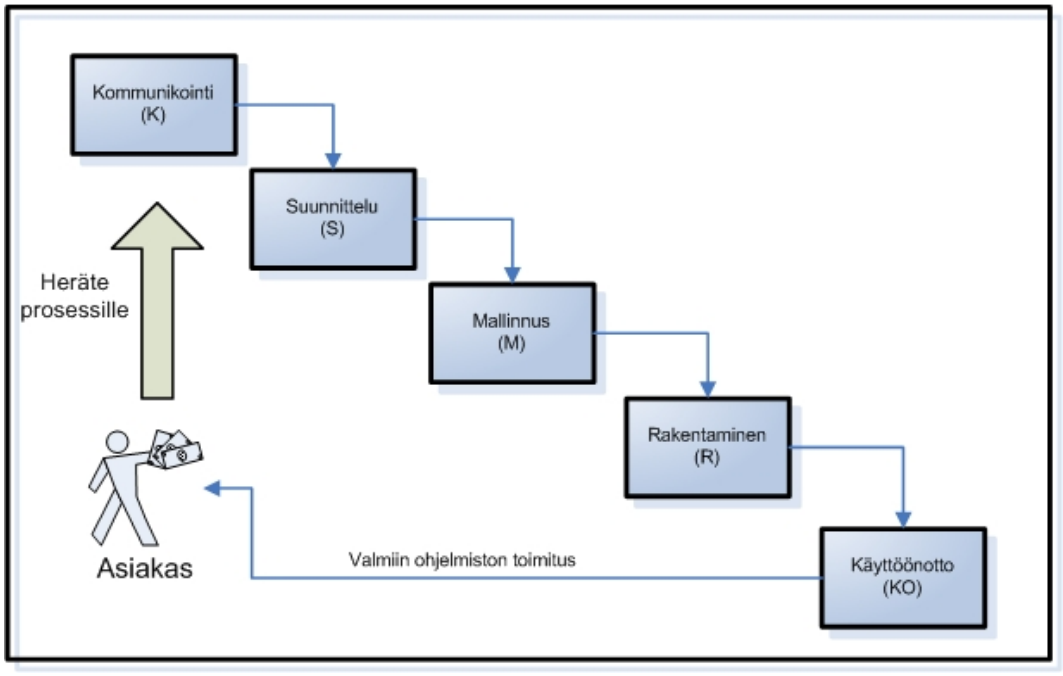
Ohjelmistoja kehitettäessä on järkevää käyttää kehitystyöhön suunniteltua toimintamallia. Ohjelmistokehityksen prosessimallit on rakennettu pääsääntöisesti ohjelmiston elinkaari -ajattelun varaan. Tässä tutkielmassa elinkaarella tarkoitetaan aikaa ohjelmiston tarpeen syntymisen ja sen ylläpitoon siirron välillä. Laajemmassa katselmuksessa ohjelmiston elinkaareen kuuluu myös ohjelmiston ylläpito, kunnes sen käyttö lopetetaan. Elinkaaren osavaiheet ovat analysointi, vaatimusmäärittelyjen tekeminen, arkkitehtuurin suunnittelu, ohjelmiston suunnittelu, ohjelmointi, käyttöönotto ja ylläpito [Ludewig, 2004]. Nämä osavaiheet ovat samoja tai osia viitekehyksen aktiviteeteista.

Tässä kohdassa esitellään lineaariset (yksisuuntaiset) prosessimallit. Seuraavassa aliluvussa käsitellään toistuvat prosessimallit. Näiden kahden tyypin selkein ero on siinä, että lineaarisia malleja ei ole suunniteltu tukemaan siirtymistä takaisin prosessin edellisiin vaiheisiin.

4.2.1 Vesiputousmalli

Vesiputousmalli on yksi perinteisimmistä prosessimalleista, ja se on pysynyt pitkän aikaa käytössä ohjelmistoteollisuudessa. Tämä on sopiva malli silloin, kun vaatimusmäärittelyt ovat erittäin hyvin tiedossa eikä uuteen ohjelmaan tai moduliin ole tulossa muutoksia kehitystyön aikana. Toisin sanoen vaatimusmäärittelyt ovat muuttumattomat [Pressman, 2005]. Kuten kuvasta 4.2 näkyy, noudattelee vesiputousmalli lineaarisesti viitekehyksen mukaisia aktiviteetteja.

Ohjelmistoteollisuuteen on kehittynyt uusia malleja, koska vesiputousmalli ei ole pystynyt vastaamaan nykypäivän ongelmiin. Ongelmat ovat monimutkaisia: vaatimusmäärittelyjä ei tunneta hyvin kehitystyön alussa, vaatimusmäärittelyt muuttuvat koko ajan, ja käyttöönoton jälkeenkin järjestelmää joudutaan jatkuvasti muuttamaan. [Davis & Bersoff, 1991]



Kuva 4.2 Vesiputousmalli

Hannan [1995] mukaan vesiputousmallin ongelmia ovat muun muassa:

1. Oikeissa projekteissa työnkulku ei ole lineaarista. Vesiputousmalli tukee käytännössä taaksepäin menemistä, mutta projektiryhmälle sen toteuttaminen on käytännössä hankalaa, koska prosessimallin soveltaminen vaikeutuu.

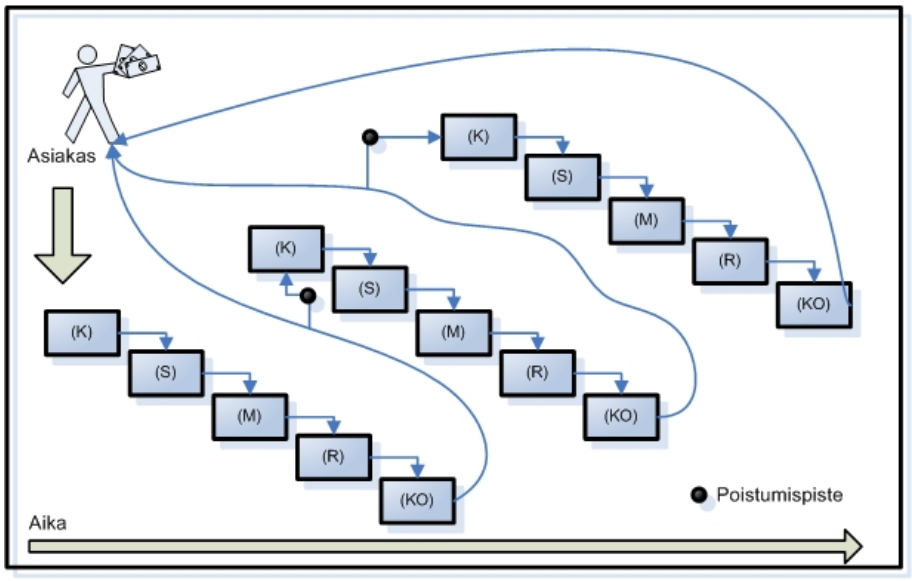
2. Vesiputousmalli vaatii, että sovelluksen vaatimusmäärittelyt ovat aukottomat. Tämä on usein asiakkaalle vaikeaa tai mahdotonta toteuttaa.

3. Asiakas ei näe tilaamaansa tuotetta ennen *käyttöönottoa* (KO). Usein asiakas huomaa todelliset käyttötarpeet vasta, kun hän pääsee käsiksi ohjelmaan.

4.2.2 Laajentuva malli

Vesiputousmallissa ongelmana oli se, että muutoksia ei voida käsitellä ja asiakas näkee lopputuotteen vasta projektin viime metreillä. Näitä ongelmia silmällä pitäen voidaan alkuperäinen vesiputousmalli pilkkoa osiin (inkrementteihin).

Jokainen osa (ohjelmiston komponentti) tekee tietyn osajoukon asiakkaan tarvitsemista toiminnallisuuksista. Kuva 4.3 havainnolistaa, että jokainen osa tehdään omassa aliprosessissaan, joka on malliltaan sama kuin vesiputousmallissa. [Pressman, 2005]



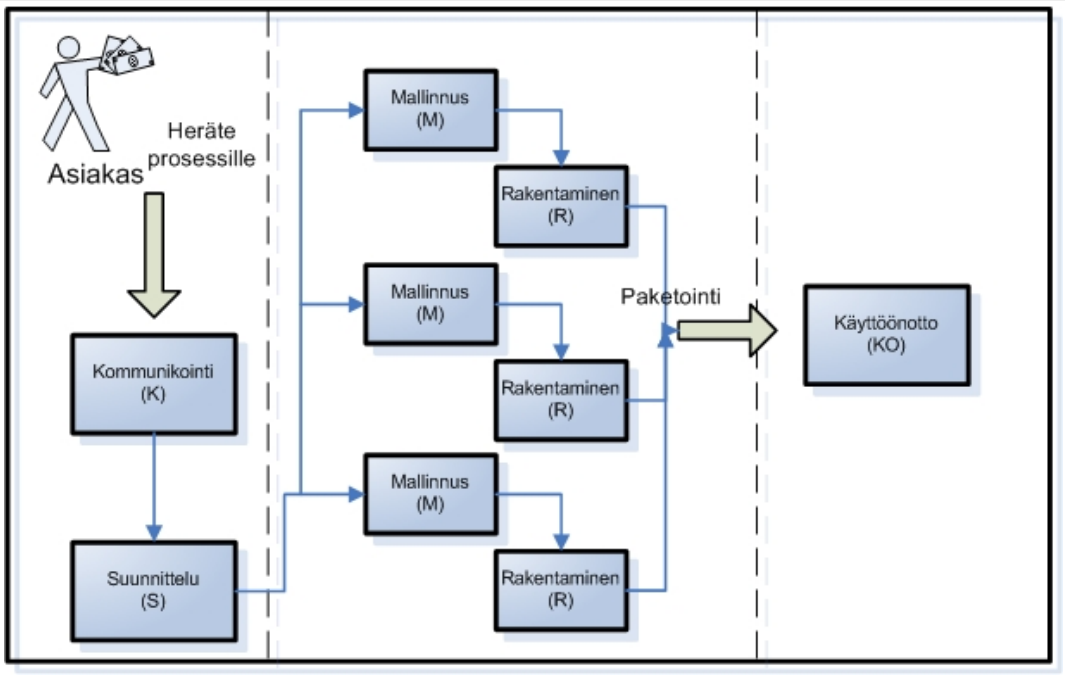
Kuva 4.3 Laajentuva malli

Tämän mallin avulla asiakas saa tarvittaessa nopeasti tärkeimmät toiminnot tuotantokäyttöön. Näin vältetään osittain riski perusliiketoiminnan vaikeutumisesta uuden ohjelmiston *käyttöönoton* (KO) yhteydessä. Projektinhallinnalle tämä malli tarjoaa poistumispisteet. Poistumispisteellä tarkoitetaan ajankohtaa, jossa asiakas tai toimittaja voi keskeyttää projektin.

4.2.3 Ripeän kehityksen malli

Ripeä sovelluskehitys (Rapid Application Development) perustuu komponenttiajatteluun. Jos rakennettava ohjelmisto on jaettavissa selkeästi toisistaan eroteltaviin komponentteihin, niiden väliset rajapinnat pystytään toteuttamaan ja resursseina on useita kehitystiimejä, voidaan RAD-mallia pitää vaihtoehtona projektin toteuttamiseen. RAD-mallissa *kommunikointi* (K) ja *suunnittelu* (S) tehdään lineaarisen mallin mukaan, mutta suunnittelussa luodaan sopiva määrä komponentteja, jotka jaetaan kehitystiimien kesken. Jokainen tiimi suorittaa *mallintamisen* (M) ja *rakentamisen* (R) rinnakkain eristetysti muista ryhmistä. Lopuksi *käyttöönoton* (KO) yhteydessä komponentit paketoidaan yhteen ja toimitetaan

asiakkaalle. [Pressman, 2005]

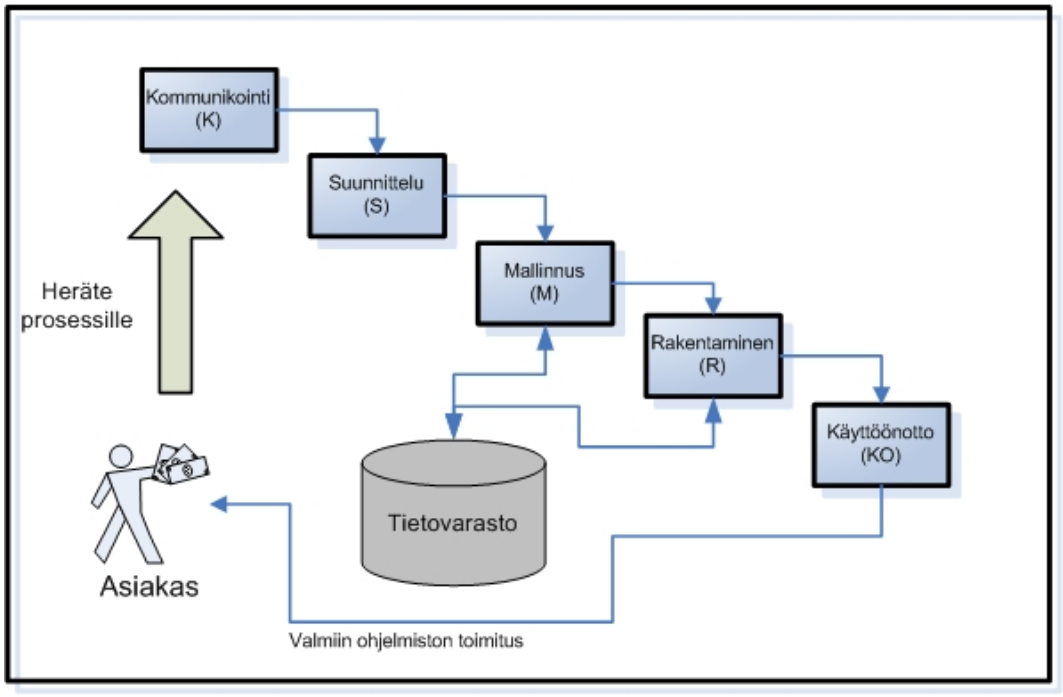


Kuva 4.4 Ripeän kehityksen malli

Kuvasta 4.4 erottuu kolme vaihetta. Ensimmäisessä vaiheessa tehdään ohjelman suunnittelutyö yhdessä asiakkaan kanssa. Toinen vaihe jakaa tehtävät useille ryhmille ja mahdollistaa nopean kehittämisen, koska töitä tehdään rinnakkain. Viimeisessä vaiheessa tehdään komponenttien paketointi, kokonaisuuden testaus ja toimitetaan ratkaisu asiakkaalle.

4.2.4 Ohjelmiston uudelleenkäyttö -malli

Laitteistovalmistajat valmistavat harvoin uusia laitteita kokonaan uusista osista. He käyttävät komponentteja, joita on käytetty ennen ja joiden tiedetään olevan luotettavia. Tietämys komponenteista on varastoitu tietovarastoihin, joista niitä voidaan aina tarvittaessa hakea ja ottaa käyttöön. Ohjelmiston uudelleenkäyttö -mallin idea oli samanlainen: pilkotaan ohjelmisto komponentteihin ja luodaan kirjasto uudelleenkäytettävistä ohjelmistokomponenteista. Tietty ohjelmakoodi on näin aina käytettävissä, kun samaa algoritmia ja toiminnallisuutta tarvitaan sovelluksen eri kohdissa. Malli takaa myös sen, että sama toiminto tekee saman asian. Ilman uudelleenkäyttöä on mahdollista, että samassa ohjelmassa tehdään sama asia usealla eri tavalla. [Davis & Bersoff, 1991]



Kuva 4.5 Ohjelmiston uudelleenkäyttö -malli

Ohjelmiston uudelleenkäyttö -malli ei ole juurikaan erilainen perinteiseen vesiputousmalliin verrattuna. Uudelleenkäyttö -mallissa tietovarastoa hyödynnetään prosessin tekniseen ohjelmistokehitykseen liittyvissä vaiheissa: *mallintamisessa* (M) ja *rakentamisessa* (R). Kuvaan 4.5 on tuotu mukaan tietovarasto, jota hyödynnetään näissä kahdessa vaiheessa.

Ohjelmistoteollisuudessa uudellenkäyttö hankaloittaa ohjelmiston hallintaa. Samaa komponenttia voidaan käyttää useassa eri projektissa ja jokainen projekti saattaa räätälöidä komponenttia omiin tarpeisiinsa sopivaksi. Jo pelkästään eri versioiden seuraaminen on hankalaa, mutta vielä vaikeampaa on hallita virhekorjausten viemistä kaikkiin versioihin. Jos peruskomponentissa huomataan virhe, ei sitä voida viedä suoraan kaikkiin räätälöityihin versioihin. Käytännössä kaikki eri versiot joudutaan testaamaan, ja tämä aiheuttaa ylimääräisiä kuluja ohjelmiston ylläpitoon. [Davis & Bersoff, 1991]

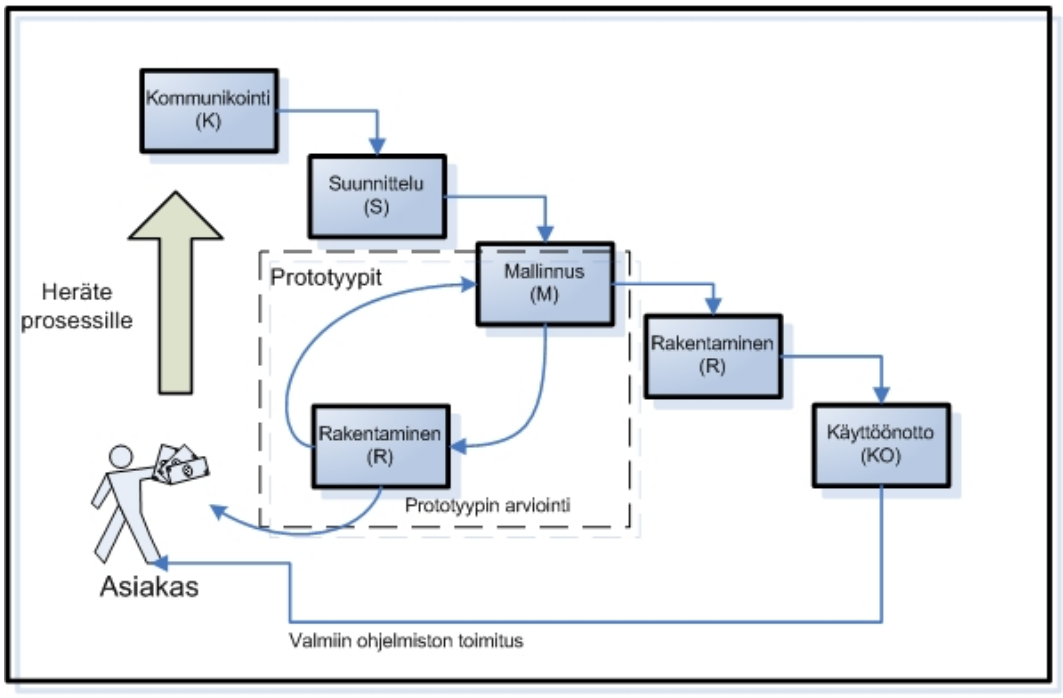
4.3 Toistuvat elinkaarimallit

Toistuville (iteroiville) prosessimalleille on tyypillistä, että niistä löytyy useita kertoja toistettavia osioita. Lineaarisista malleista poiketen niissä on mahdollista

korjata aikaisemmissa vaiheissa tehtyjä päätöksiä. Tämä ominaisuus vähentää riskiä, että asiakkaan todellisia tarpeita ei ymmärretä oikein. Tässä kohdassa yhdistetään viitekehys iteroiviin prosessimalleihin.

4.3.1 Protoilu

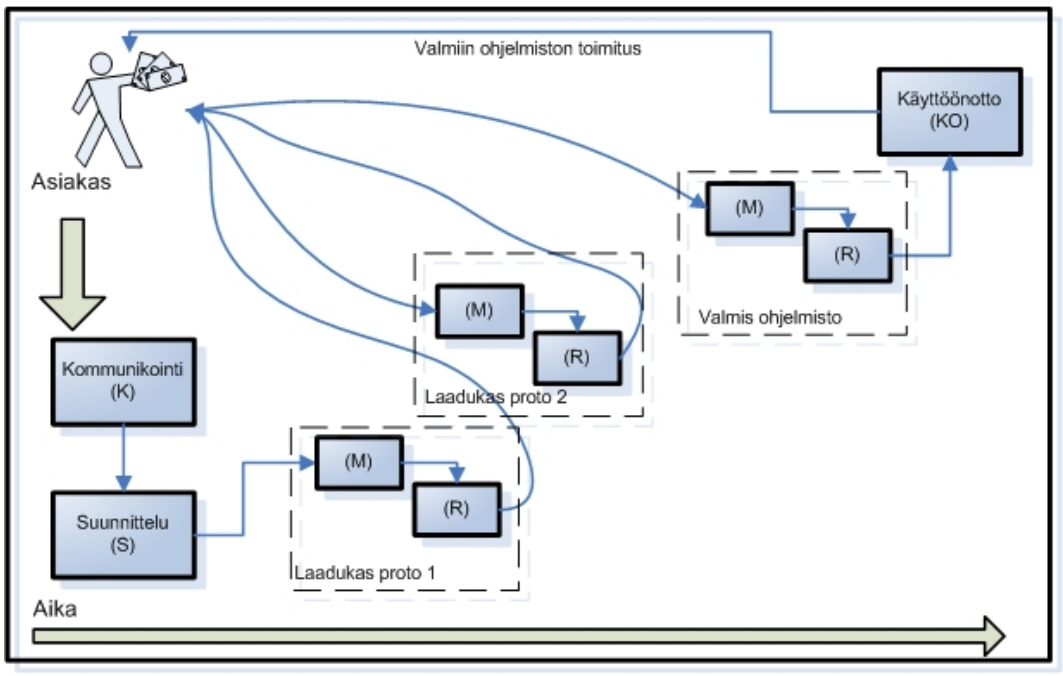
Usein asiakkaalta saatavan informaation avulla ei ole mahdollista määritellä ohjelmistoa riittävän tarkasti. Lineaarista prosessimallia käytettäessä on vaarana, että syntyvä ohjelmistotuote ei vastaa asiakkaan todellisia tarpeita. Valmistavasta teollisuudesta tuttu prototyyppien rakentaminen on saanut kannatusta myös ohjelmistoteollisuuden puolella. Ohjelmasta luodaan oikean näköisiä ja tuntuisia prototyyppejä nopealla vauhdilla. Prototyypit *rakennetaan* (R) näyttämään, mitä asiakas on saamassa. Todelliseen toiminnallisuuteen ja hyvään ohjelmointitapaan ei niiden tekemisessä kiinnitetä huomiota. [Davis & Bersoff, 1991]



Kuva 4.6 Protoilu

Kuvasta 4.6 huomataan paljon yhtäläisyyksiä vesiputousmallin kanssa. Prototyypeillä näytetään asiakkaalle käytännössä, mitä *mallinnuksessa* (M) on havaittu ja minkälainen järjestelmä aiotaan *rakentaa* (R). Protojen tekemistä varten on oma aliprosessi, joka sisältää vain *rakennusaktiviteetin* (R).

Protoilu on hyödyllistä ohjelmistokehityksessä, koska se vähentää huomattavasti riskiä siitä, että rakennettava sovellus ei tyydytä asiakkaan vaatimuksia. Se vähentää kehityskuluja, koska projektin aikana tapahtuvat muutokset vaatimusmäärittelyihin vähenevät. Protoilu parantaa projektin onnistumismahdollisuuksia, koska kehittäjän ja käyttäjän välinen kommunikaatio paranee. Suurimmat ongelmat protoilulle ovat prototyypin laajuuden (tekoon käytettävän ajan) rajaaminen ja valmistuvan lopputuotteen laadun takaaminen. Asiakkaalle on usein vaikeata selittää, miksi protoa ei voida käyttää tuotantokäytössä, vaan joudutaan ohjelmoimaan kokonaan uusi sovellus. [Davis & Bersoff, 1991]



Kuva 4.7 Evoluutiokehitys

4.3.2 Evoluutiokehitys

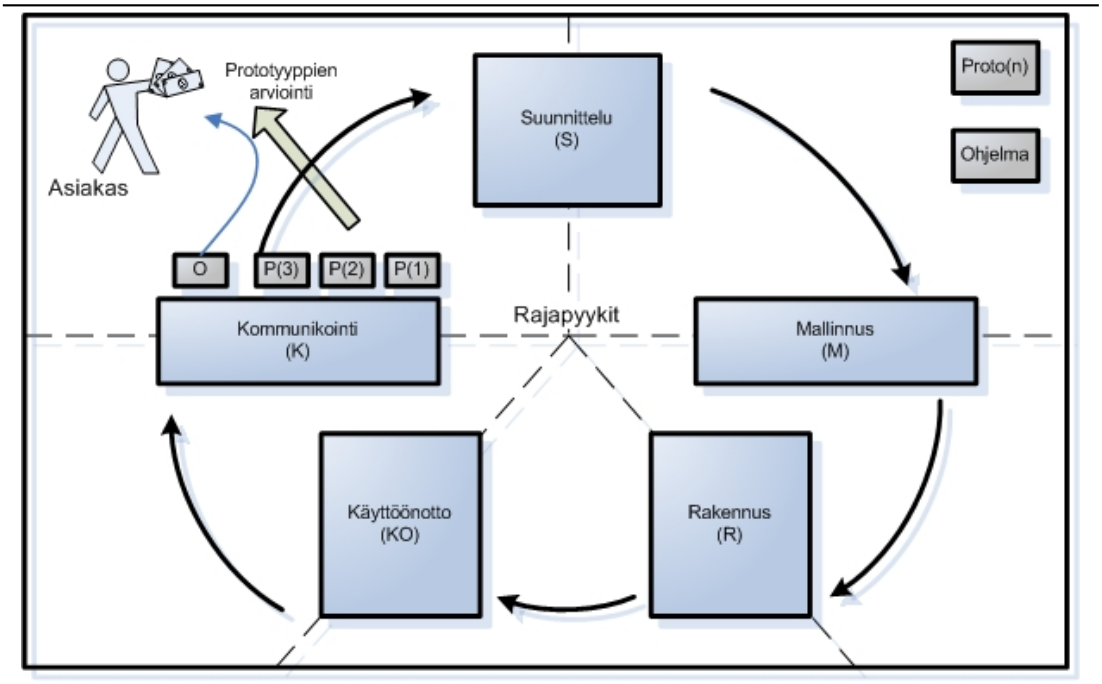
Protoilua on kritisoitu, koska se ei tuota dokumentaatiota eikä kehitystyö ole systemaattista. Poisheitettävien prototyyppien tarkoitus on parantaa kehitettävän järjestelmän laatua loppukäyttäjän näkökulmasta. Evoluutiokehitys on protoilun systemaattisempi versio, jossa protot *mallinnetaan* (M) ja *rakennetaan* (R) huolella eikä niitä heitetä pois, vaan niistä tulee lopullisen ohjelmistotuotteen osia. [Luqi *et al.*, 1998]

Laajentuvassa mallissa ohjelmisto on koko ajan käytettävissä (valmiit osat),

ja uusien osien kehitys aloitetaan aina alusta ja viedään loppuun asti. Evoluutiokehityksessä sen sijaan kehitetään ensin kaikista selkeimmät ja riskittömimmät osat ja laajennetaan niitä kierros kerrallaan kohti valmista ohjelmistotuotetta. Kuvia 4.3 ja 4.7 vertaamalla havaitaan, että evoluutiokehityksessä asiakas saa käytettävän tuotteen vasta prosessin lopussa. Laajentuvassa mallissa osio on heti valmistuttuaan asiakkaan käytettävissä. [Davis & Bersoff, 1991]

4.3.3 Spiraalimalli

Spiraalimalli on perinteisen lineaarisen vesiputousmallin ja iteroivan protoilun yhdistävä prosessimalli. Vesiputousmallista on otettu mukaan systemaattisuus, vaiheet ja ankkuripisteet. Spiraalimallissa kehitys tapahtuu kierroksittain. Jokainen kierros tuottaa prototyypin, joka alussa saattaa olla paperiversio tai poisheitettävä sovellus, mutta loppua kohti protot ovat evoluutiokehityksen mukaisia ohjelmiston osia. [Boehm, 1988]



Kuva 4.8 Spiraalimalli

Spiraalimalli tukee projektinhallintaa ja toisaalta nopeaa kehitystä. Projektille pystytään suunnittelemaan selkeät aikajanat kuvassa 4.8 näkyvien rajapyykkien avulla. Toisaalta malli minimoi riskejä, koska kehitys tehdään yksi sykli kerrallaan ja joka syklin jälkeen on mahdollista arvioida uudelleen, kannattaa-

ko ohjelmistokehitystä jatkaa. Spiraalimalli on kuitenkin melko raskas toteuttaa, koska se vaatii monia muita malleja enemmän aikaa ja resursseja projektinhallintaan. Tämän lisäksi prototyyppien kehittäminen vaatii oman aikansa ja kasvattaa projektin kustannuksia. [Pressman, 2005]

5 Laajat prosessimallit

Perusprosessimalleissa keskityttiin siihen, mitä seuraavaksi pitää tehdä. Laajoissa ohjelmiston kehitys- ja käyttöönottoprojekteissa suuri osa työstä on muutoksenhallintaa. Muutos pitää sisällään ihmisten kouluttamisen ja totuttamisen, tehtyjen päätösten ja ratkaisujen dokumentoinnin ja tehtyjen muutosten testaamisen organisaatiossa.

Lisäämällä näitä elementtejä prosessimalleihin pystymme luomaan projektisuunnitelmia, jotka mahdollistavat projektin aikana syntyvien organisaatio- ja liiketoimintamuutosten huomioimisen. Ohjelmiston toimittajan näkökulmasta erittäin tärkeää on pystyä rajaamaan hankkeen alkuperäinen tavoite ja pystyä hallitsemaan muutos projektin edetessä. [Ludewig, 2004]

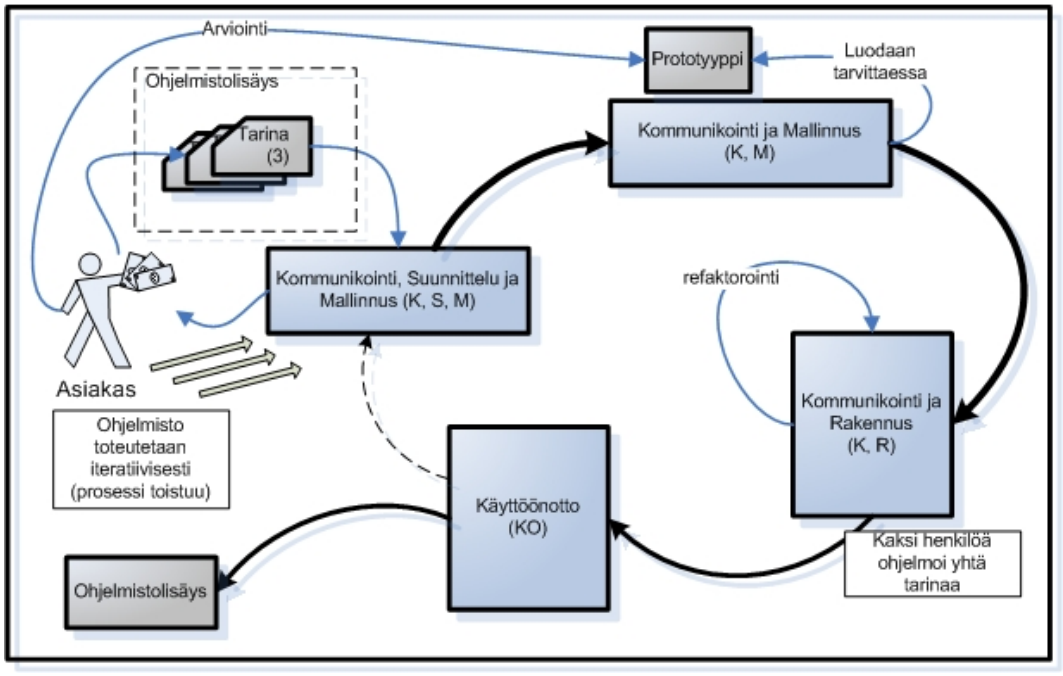
Tässä luvussa esitellään kaksi ketterää (*agile*) ohjelmiston kehitysmallia ja kaksi vaihejakomalliin perustuvaa ohjelmiston käyttöönottomallia. Samalla mallit sovitetaan aikaisemmin esiteltyyn viitekehykseen.

5.1 Extreme Programming

Extreme Programming (XP) on laaja prosessimalli, joka on suunniteltu nimenomaan välttämään perinteisten prosessimallien pitkiä ohjelmiston kehityskierroksia [Beck, 1999]. XP-mallista löytyy samoja piirteitä kuin evoluutiokehityksestä, mutta erityisesti se keskittyy luomaan ohjelmointiin sopivan työympäristön. Ohjelmointi tehdään pieninä lisäyksinä (*software increment*) ja palasia testataan koko ajan. Osat integroidaan toisiinsa tiheällä syklillä, jotta pienennetään kokoonpanovaiheen riskejä. Yksi tunnetuimmista piirteistä on XP-mallin edellyttämä pariohjelmointi. Aina kun koodia kirjoitetaan, toinen ohjelmoi ja toinen tarkkailee.

XP:ssä ohjelmiston lisäykset muodostuvat tarinoista. Tarinat ovat asiakkaan tekemiä, ja ne sisältävät ohjelmistolta vaaditut ominaisuudet ja toiminnot. XP-prosessissa jokainen tarina saa tärkeysasteen ja arvion kehitystyön vaativuudesta (hinnasta). Tarinoiden koko on rajoitettu siten, että yhden kehittämiseen ei saa mennä yli kolmea kehitystyöviikkoa. Jos arvio ylittää tämän rajapyykin, palautuu tarina sen suunnittelijalle, ja se pyritään jakamaan pienempiin osiin. [Pressman, 2005, 110-113]

Kehitys tapahtuu iteroimalla prosessia uudelleen ja uudelleen. Kuvassa 5.1 siirtymistä uudelle iteraatiokierrokselle kuvaa katkoviiva, joka lähtee *käyttöönotosta* (KO). Jokaisen kierroksen on tarkoitus tuottaa uusi lisäyspaketti tuotanto-



Kuva 5.1 XP-malli

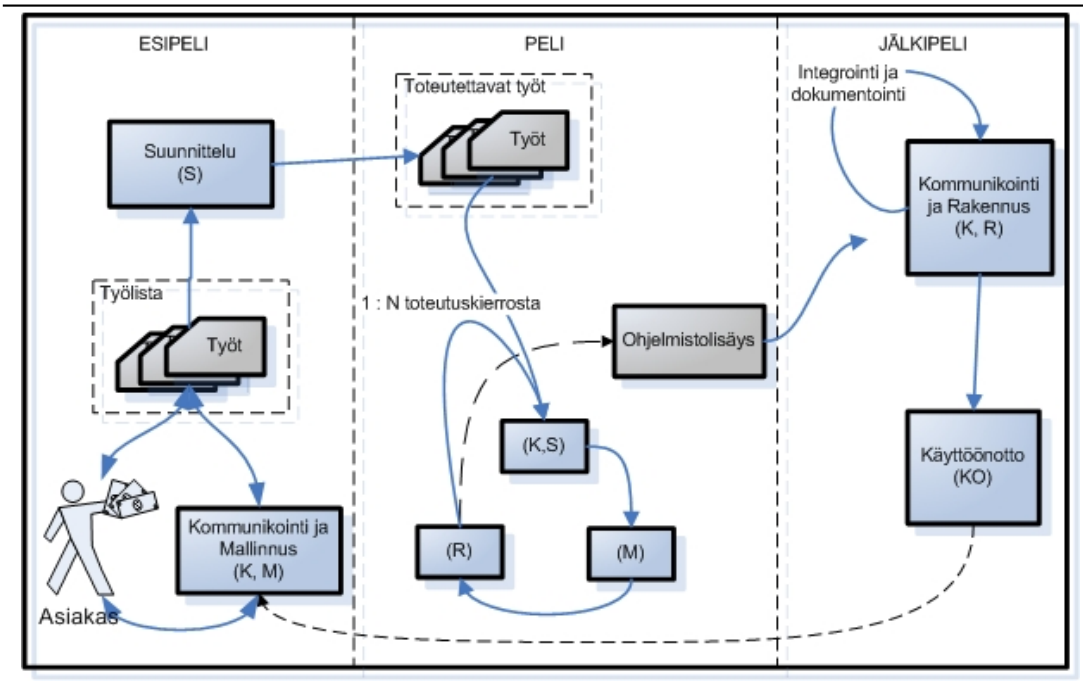
käyttöön. XP-mallissa *kommunikointi* (K) on vahvassa asemassa. Sitä tapahtuu lähes joka vaiheessa ja varsinkin pariohjelmoinnin käyttäminen *rakentamisessa* (R) vähentää huomattavasti virheiden määrää ja nopeuttaa ongelmien ratkaisua. Lisäksi XP-mallissa *rakentaminen* (R) aloitetaan tekemällä yksikkötestit jokaista tarinaa vasten. Vasta sen jälkeen siirrytään ohjelmoimaan. [Beck, 1999]

XP-mallin huono puoli on se, että se keskittyy nimenomaan uuden ohjelmiston tekemiseen ja tuo vähän keinoja laajojen projektien hallintaan. Mielestäni XP-malli sopii hyvin pienten ohjelmistotiimien tehokkaaseen toimintaan, kun asiakas on joko talon sisäinen tai muuten erittäin lähellä kehitystyötä ja näin pystyy tiiviisti tukemaan ohjelmointia. Ottaen huomioon liike-elämän vaatimukset pariohjelmointia tulisi käyttää harkiten tiettyihin osiin sovellusta, koska se on kallista, vaikka sen tuomaa hyötyä ei voidakaan kiistää.

5.2 Scrum

Scrum on yksi laajoista ketterän ohjelmistokehityksen prosessimalleista. Nimi juontaa juurensa rugby-ottelussa käytetystä aloituskokoonpanoa kuvaavasta termistä [Schwaber & Beedle, 2002]. Seuraavia voidaan pitää Scrum-prosessimallin periaatteina [www.controlchaos.com, 2008]:

- Se on ketterä prosessi, jolla johdetaan ja hallitaan kehitystyötä.
- Se laajentaa olemassa olevia ohjelmistokehityksen toimintatapoja.
- Se on tiimityöskentelyyn perustuva lähestymistapa, jossa ohjelmisto luodaan pieninä lisäyksinä useissa kehityskierroksissa ja vaatimusmäärittelyt muuttuvat useasti.
- Prosessilla kontrolloidaan eriävien toiveiden ja tarpeiden kaaosta.
- Se parantaa kommunikaatiota ja maksimoi yhteistyön.
- Prosessi pyrkii havaitsemaan ja poistamaan kaikki ohjelmiston kehittämisen ja valmistumisen esteet.
- Se on prosessi, jolla voidaan hallita yhtä pienen tiimin projektia tai tarvittaessa laajan organisaation ohjelmistonkehitystä.
- Se keskittyy ihmisiin tarjoamalla jokaiselle oman paikan, selkeän etenemisen tunteen ja mahdollisuuden tehdä parhaansa.



Kuva 5.2 Scrum-malli

Scrum-prosessi on kokonaisuudessaan esitetty kuvassa 5.2 piirrettynä viitekehyksen muotoon. Kysessä on vaihejakomalli, jossa on kolme vaihetta: esipeli, peli (kehitysvaihe) ja jälkipeli. Seuraavaksi käydään läpi Scrumin toimintaperiaate vaihe vaiheelta.

Esipeli lähtee liikkeelle työlistan (*backlog*) muodostamisesta. Työlista muodostuu tehtävän ohjelmiston vaatimuksista tai edelleenkehitettävän ohjelmiston lisävaatimuksista. Se tehdään yhdessä asiakkaan ja Scrum-tiimin kanssa *kommunikoimalla* (K) ja *mallintamalla* (M) alustava arkkitehtuuri. Työlistaa voidaan täydentää missä tahansa vaiheessa. Esipelivaiheen lopussa *suunnitellaan* (S) pelivaiheeseen mukaan otettavat toteutettavat työt.

Peli muodostuu toteutuskierroksista (*sprint*). Jokaiseen kierrokseen kuuluu yksi tai useampia töitä toteutettavien jonosta, ja ne kehitetään perinteisesti *kommunikoimalla* (K) ja *suunnittelemalla* (S) ensin. Sen jälkeen siirrytään *mallintamaan* (M) ja lopulta *rakennetaan* (R) haluttu lopputuote. Yhden tai useamman kierroksen lopputuotteista syntyy ohjelmistolisäys. Yhden kierroksen kesto on rajattu 30 päivään.

Jälkipelissä *kommunikoimalla* (K) ja *rakentamalla* (R) integroidaan ja dokumentoidaan edellisessä vaiheessa tehty ohjelmistolisäys. Tarvittaessa ohjelmistolisäys voi toimia eräänlaisena protona, jos edeltävään pelivaiheeseen oli otettu vain rajallinen määrä töitä tehtäväksi. Lopulta ohjelmistolisäys *käyttöönotetaan* (KO) ja tämän jälkeen voidaan jatkaa ohjelmiston edelleenkehitystä. [Schwaber & Beedle, 2002]

Scrumin hieno piirre on se, että se keskittyy koko hankkeen läpi *ihmisiin* ja kehitettävään *tuotteeseen*. Toisin sanoen se pitää huolen siitä, että ohjelmiston kehitys on osa asiakkaan ja toimittajan arkipäivää. Palavereiden ja raportoinnin avulla myös projektin johto pysyy riittävän nopealla syklillä (alle yksi kuukausi) ajan tasalla. Sekä XP-mallissa että Scrumissa ajatus työlistasta ja lyhyistä kehityskierroksista mahdollistaa *tuotteen* valmistumisen markkinoille silloin, kun niin halutaan. Tekemättömät työlistan työt jäävät vain odottamaan seuraavaa versiota. Tämä on mielenkiintoinen ja erittäin hyödyllinen ominaisuus nykypäivän liike-elämälle.

Käyttöönottoprojekteissa otetaan usein kerralla kaikki asiakkaan liiketoimintaprosessit käyttöön uudessa ohjelmistossa. Tämä luonnostaan tekee projektista ison ja riskit kasvavat. Scrum-mallista voisi hyvinkin ottaa ideoita myös toiminnanohjausjärjestelmän käyttöönottomalliin. Kuvitellaan tilanne, jossa asiakkaalla on nykyjärjestelmässä tuotantokäytössä myynti-, hankinta- ja talousprosessit. Normaalissa käyttöönottoprojektissa kaikki kolme prosessia määriteltäisiin uu-

teen ohjelmistoon, testattaisiin ja vanha ohjelmisto korvattaisiin kerralla. Vaihtoehtoisessa mallissa kolme prosessia laitettaisiin työjonoon ja yhden käyttöönottokierroksen aikana otettaisiin käyttöön aina vain yksi prosessi. Tämä pienentäisi huomattavasti projektin riskejä ja helpottaisi uuden ohjelmiston oppimista.

Uusi malli toisi myös haasteita. Uudella ohjelmistolla käyttöönotettava prosessi pitää pystyä integroimaan vanhan ohjelmiston prosesseihin, ja loppukäyttäjä joutuu käyttämään kahta ohjelmistoa samanaikaisesti eri prosessien hoitamiseen. Kolmantena haasteena ovat kustannukset, koska kaksi tuotantokäytössä olevaa järjestelmää lisää lisenssi- ja laitteistokustannuksia, jotka ovat usein merkittävä osa ohjelmistohankkeen kokonaiskustannuksista.

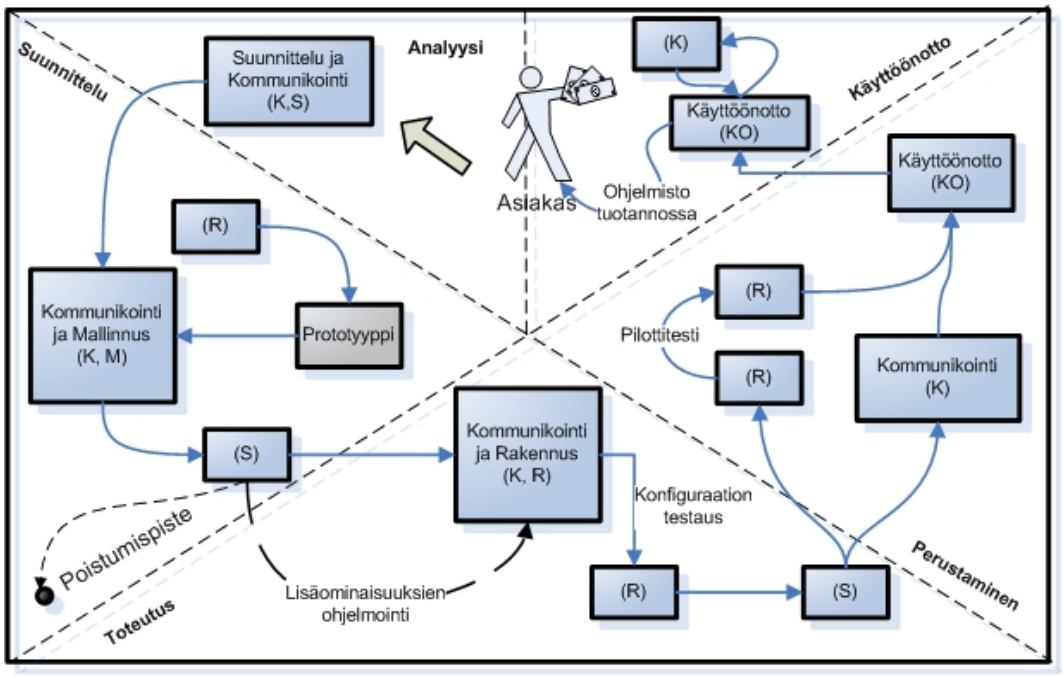
Integrointiominaisuudet ovat nykypäivän ohjelmistoissa kehittyneet huomattavasti. Tämän vuoksi välivaiheen (kaksi rinnakaista ohjelmistoa samanaikaisessa käytössä) prosessi-integraatiot ovat varteenotettavia vaihtoehtoja projekteissa, joissa integroinnin edellytykset on otettu huomioon. Toinen haaste on kaksi erillistä käyttöliittymää. Nykypäivän järjestelmissä käyttöliittymä pyritään eriyttämään sovelluksen toiminnoista. Tästä johtuen on mahdollista rakentaa loppukäyttäjille yksi yhteinen käyttöliittymä avaintoimintojen osalta sekä vanhaan että uuteen ohjelmistoon. Näiden molempien mahdollisuuksien taustalla on palvelukeskeinen arkkitehtuuri (*Service Oriented Architechture*, SOA), jonka päälle monet nykypäivän sovellukset rakennetaan [Papazoglou & van den Heuvel, 2007].

Kolmantena haasteena ovat kustannukset. Niihin vastaamisen täytyy lähteä kokonaishankkeen kustannuksista. Yksittäisen prosessin siirtäminen uuteen ohjelmistoon voi olla niin merkittävä tekijä, että on mielekästä lisätä projektin kustannuksiin ylimääräinen integrointityö, lisenssi- ja laitteistokustannukset. Toisaalta täysmittaisen käyttöönottoprojektin riskien mahdolliset kustannukset voivat olla riittävä perustelu kalliimmalle projektille.

5.3 Implex

Implex on perinteiseen vaihejakomalliin perustuva parametroitavan ohjelmiston käyttöönottomalli, jonka Lawson on kehittänyt M3 (entiseltä nimeltään Movex) toiminnanohjausjärjestelmän käyttöönottomenetelmäksi [Lawson, 2008a]. Malli jakaantuu viiteen vaiheeseen: analyysi (*position*), suunnittelu (*design*), toteutus (*configure*), perustaminen (*implement*) ja käyttöönotto (*start up*). Tämän ja muiden käyttöönottomallien kohdalla *rakentaminen* (R) sisältää sekä ohjelmointia että parametrointia, koska niitä voidaan pitää lopputuloksen kannalta samanarvoisina tehtävinä. Lisäksi parametroinnin testaus on *rakentamista* (R), kuten

ohjelmiston palasen testaus.



Kuva 5.3 Implex-malli

Kuvassa 5.3 Implex-malli on sovitettu yleisen ohjelmistoprosessimallin mukaiseksi. Ensimmäisessä analyysivaiheessa *suunnitellaan* (S) projektin tavoitteet, aikataulu ja organisaatio. Näin syntynyt projektisuunnitelma ja -organisaatio *kommunikoidaan* (K) sekä asiakkaan että toimittajan hankkeeseen osallistuville henkilöille.

Seuraavaksi suunnitteluvaiheessa *kommunikoinnin* (K) avulla *mallinnetaan* (M) asiakkaan liiketoimintaprosessit sovitettuna toiminnanohjausjärjestelmän prosesseihin ja parhaisiin käytäntöihin. Rinnalle *rakennetaan* (R) prototyyppi, jonka avulla asiakkalle näytetään, miten ohjelmistossa halutut prosessit toimivat. Vaiheen aikana asiakkaan prosessit dokumentoidaan ja mahdolliset puuttuvat ominaisuudet ja toiminnot analysoidaan. Vaiheen lopussa *suunnitellaan* (S) aikataulu ja budjetti jatkovaiheille. Suunnitelma huomioi asiakaskohtaisten laajennusten *rakentamiseen* (R) kuluvan ajan, jonka mahdollisten poikkeamien paikkaaminen aiheuttaa. Joissain projekteissa on sovittu, että juuri tämän vaiheen lopussa on olemassa poistumispiste, jota sopimuksesta riippuen joko vain asiakas tai asiakas ja toimittaja voivat hyödyntää.

Kolmannessa eli toteutusvaiheessa *kommunikoimalla* (K) asiakkaan kanssa *ra-*

kennetaan (R) järjestelmä tukemaan projektin tavoitteiden mukaisia liiketoimintaprosesseja. Kaikki suunnitteluvaiheessa sovitut asiakaskohtaiset laajennukset (sisältäen sekä muutokset järjestelmän prosesseihin, tulosteisiin ja käyttöliittymään että liittymät ja konversiot muiden järjestelmien välillä) *rakennetaan* (R) ja lopuksi ennen vaiheen hyväksymistä tehdään konfiguraatiotesti. Seuraavaan vaiheeseen siirrytään *suunnittelemalla* (S) projektin kaksi viimeistä vaihetta ja toteamalla konfiguraatio hyväksytyksi.

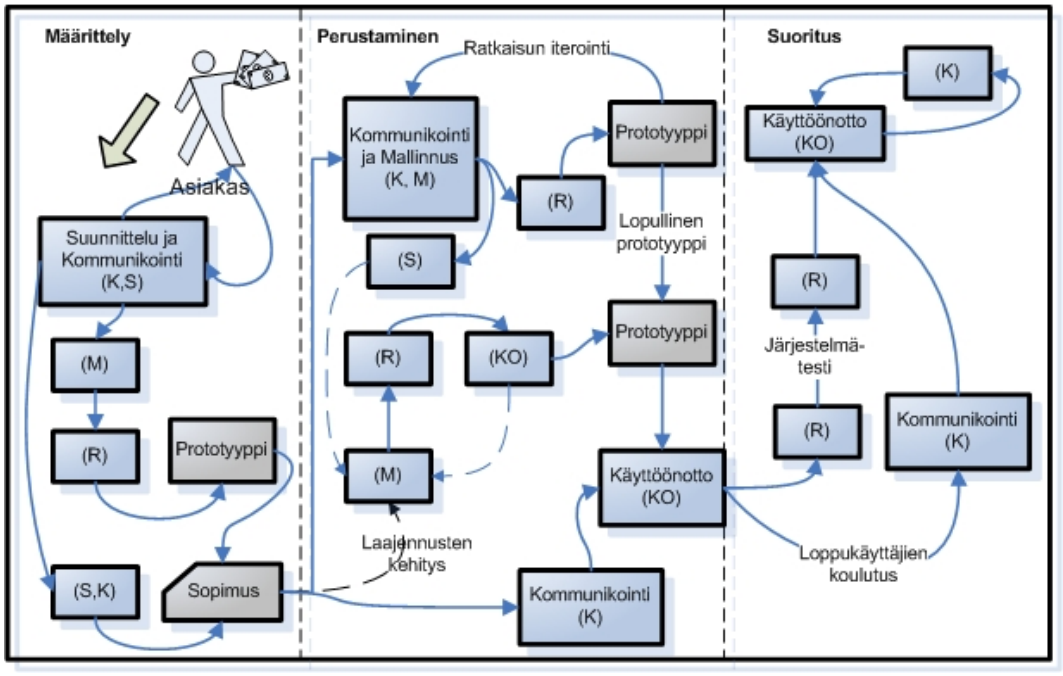
Perustamisvaiheella on kolme tehtävää: varmistaa pilottitestillä, että *rakennettu* (R) ratkaisu täyttää asiakkaan liiketoimintaprosessien vaatimukset, kouluttaa loppukäyttäjät ja suorittaa laajamittainen testi, jolla varmistetaan järjestelmän valmius tuotantokäyttöön. Pilottitestissä hyödynnetään asiakkaan omaa aineistoa, joka on tuotu uuteen järjestelmään *rakennetuilla* (R) konversioilla ja manuaalisesti syöttämällä. Implex-malli hyödyntää kouluta kouluttajaa -lähestymistapaa, jossa edeltävissä vaiheissa konsultit kouluttavat *kommunikoimalla* (K) asiakkaan pääkäyttäjät. Pääkäyttäjien tehtäväksi jää *kommunikoida* (K) tieto eteenpäin asiakasorganisaation loppukäyttäjille. Laajamittainen testi pyrkii simuloimaan tuotantotilannetta mahdollisimman tarkasti, ja sen tulos hyväksymällä siirrytään viimeiseen vaiheeseen. Perustamisvaiheen *käyttöönotto* (KO) prosessikaaviossa kuvaa hyväksymistestiä.

Käyttöönottovaiheessa vanha järjestelmä ajetaan alas ja uusi *käyttöönotetaan* (KO). Vaiheen ajan asiakkaan pääkäyttäjät ja toimittajan konsultit *kommunikoivat* (K) loppukäyttäjien kanssa ja näin tukevat uuden järjestelmän käyttöä. Vaihe päättyy, kun järjestelmä on käytössä ja molemmat osapuolet hyväksyvät projektin.

5.4 StepWise

StepWise on kolmivaiheinen parametroitavan ohjelmiston käyttöönottomalli [Lawson, 2008d]. Implex-mallin haaste on pitkä suunnitteluvaihe, jonka aikana ratkaisu dokumentoidaan, mutta lopputulosta ei päästä kunnolla näkemään. StepWise käyttää protoilua, jonka avulla lopputulos on koko projektin ajan nähtävissä. Mallin vaiheet ovat: määrittely (*define*), perustaminen (*establish*) ja suoritus (*execute*). Kuvassa 5.4 on kunkin vaiheen prosessikaaviot sovitettuna viitekehykseen.

Ensimmäisen määrittelyvaiheen tärkein lopputulos on toimeksiantosopimus asiakkaan ja toimittajan välille. Sopimuksesta tulee selvitä projektin tavoite ja siihen pääsemiseksi tehtävä työ. Määrittelyvaihe on käytännössä myyntityötä, mutta joissain tapauksissa se voi sisältää asiakkaan toimeksiannosta tehtävän



Kuva 5.4 StepWise-malli

tarkemman esiselvityksen. *Kommunikoimalla* (K) ja *suunnittelemalla* (S) tehdään alustava projektisuunnitelma ja aloitetaan ensimmäisen prototyypin *mallinnus* (M). *Rakennettavalla* (R) prototyypillä esitellään asiakkaalle tärkeimmät prosessit, joita toiminnanohjausjärjestelmässä tullaan käyttämään. Lopputuloksena syntyvään sopimukseen pyritään prototyypin ja *kommunikoinnin* (K) avulla kirjaamaan projektin tavoite mahdollisimman tarkasti. Tavoitteessa huomioidaan kaikki lisäohjelmointia vaativat laajennukset toiminnanohjausjärjestelmään.

Kun sopimus on allekirjoitettu, siirrytään perustamisvaiheeseen. Vaihe aloitetaan *kommunikoimalla* (K) projektin tavoitteet ja tehty prototyyppi projektiryhmälle. Tämän jälkeen toimittaja ja asiakas *mallintavat* (M) yhdessä ratkaisua ja prototyypistä *rakennetaan* (R) uusi versio. Ratkaisun *mallintaminen* (M) ja prototyypin *rakennus* (R) tehdään vähintään kaksi kertaa. Projektin laajuudesta riippuen iterointia jatketaan, kunnes ratkaisu sisältää kaikki tarvittavat toiminnot. Iteroinnissa mukana oleva *kommunikointi* (K) varmistaa, että ratkaisun kuvaus pysyy ajan tasalla ja projektiryhmän tiedossa. Ratkaisua läpikäytäessä saatetaan havaita puuttuvia toimintoja tai ominaisuuksia. Nämä *suunnitellaan* (S) sopimuksessa määritellyn muutoksenhallintaprosessin mukaan ja tarvittaessa sisällytetään projektin tavoitteeseen. Lisäominaisuudet kehitetään *mallintamalla* (M) ja *rakentamalla* (R). Ne *käyttöönottetaan* (KO) asiakkaan hyväksynnäl-

lä lopulliseen prototyyppiin. Vaihe päättyy *käyttöönottoon* (KO), jossa suoritetaan ratkaisun hyväksymistesti. Ennen testiä projektiryhmälle *kommunikoidaan* (K) ohjelmiston ja ratkaisun käyttämiseen tarvittava osaaminen erillisen oppimissuunnitelman mukaan.

Viimeinen suoritusvaihe alkaa, kun ratkaisu on hyväksytty. Suoritusvaiheessa ratkaisua ei enää muuteta, vaan tehdään tarvittavat toimenpiteet *käyttöönoton* (KO) varmistamiseksi. Kaaviossa kaksi peräkkäistä *rakennus* (R) -aktiviteettia kuvaavat tehtyjen konversio-ohjelmien koeajoa ja järjestelmätestiä, jolla ohjelmiston suorituskyky ja toimivuus varmistetaan. Tämän rinnalla asiakkaan pääkäyttäjät *kommunikoimalla* (K) kouluttavat loppukäyttäjät. *Käyttöönottoon* (KO) valmistaudutaan tekemällä laajamittainen testi. Kun testi on hyväksytty, otetaan uusi järjestelmä käyttöön. Alkuvaikeuksien välttämiseksi projektiryhmä *kommunikoimalla* (K) tukee loppukäyttäjiä.

StepWise-mallin perusidea on varmistaa, että toimittaja tietää koko ajan, mitä asiakas on tilaamassa. Toisaalta prototyyppien avulla asiakas tottuu uuteen järjestelmään pienissä paloissa ja näkee mitä on saamassa. Näin pidetään huoli, että poikkeamiin pystytään puuttumaan ajoissa ja sitä kautta varmistamaan, että projekti tulee onnistumaan.

6 Tietojärjestelmäprojektin onnistuminen

Tässä luvussa käsitellään eri mittareita, joita voidaan käyttää tietojärjestelmäprojektin onnistumisen mittaamiseen. Ensin käsitellään tutkijoiden W. Delone ja E. McLean tutkimuksia vuosilta 1992 ja 2003. Näissä tutkimuksissa luodaan ja jatkokehitetään tietojärjestelmän onnistumismittaristoa (*I/S Success*) ja siihen liittyvää D&M-onnistumismallia (*D&M IS Success Model*).

Onnistumismallin jälkeen käydään läpi tietojärjestelmän vaikuttavuuden mittaamista ja sen hyödyntämistä onnistumisen mittarina. Kolmantena perehdytään mahdollisuuksien analysointiin, mikä edesauttaa projektien onnistumista. Viimeisenä käsitellään onnistumisen mittaamista tutkielman kenttätutkimusosiossa.

Tässä luvussa esitellään tietojärjestelmäprojektien onnistumisen periaatteita ja menetelmiä, joiden avulla pystytään sekä mittaamaan onnistumista että toteuttamaan onnistuneita projekteja. Näihin periaatteisiin ja menetelmiin viitataan kenttätutkimuksessa, kun projektien onnistumista arvioidaan. Toisaalta tämän tutkielman keskeisessä osassa ovat prosessimallit. Jotta prosssimalleja olisi mahdollista kehittää eteenpäin tai ylipäätänsä vertailla keskenään, on tärkeää tuntea onnistuneen projektin edellytykset.

6.1 Tietojärjestelmän onnistumismalli

Tietojärjestelmäprojektin läpivienti on usein pitkä ja työläs prosessi. Se vaatii sitoutumista sekä toteuttavalta että tulevaa ratkaisua käyttävältä osapuolelta. Mittaamiseen ei ole yhtä oikeaksi todistettua vaihtoehtoa, vaan onnistumisen mittaaminen tapahtuu hyödyntämällä useita toisistaan riippuvaisia muuttujia.

Delone ja McLean ovat tutkineet tietojärjestelmien mittaamista laajasti, ja heidän työnsä tuloksena on syntynyt tietojärjestelmän onnistumismittaristo. Mittaristoa hyödynnetään D&M-onnistumismallissa, jossa on käytössä kuusi kategoriaa muuttujille:

1. Järjestelmän laatu (*System Quality*): Mitataan tietojärjestelmän toimivuutta ja prosessointikykyä. Laitteiston ja ohjelmiston vasteaikoja, paikkansapitävyyttä ja vakautta.

2. Informaation laatu (*Information Quality*): Tällä tarkoitetaan tietojärjestelmän tuottamaa informaatiota (näytölle, paperille, raportteihin jne). Kuinka tuoretta ja paikkansapitävää järjestelmän tuottama tieto on? Onko tieto riittävän tarkkaa?

3. Käyttö (*Use*): Mitataan, kuinka laajasti tietojärjestelmä on käytössä ja kuinka hyvin sen tuottamaa tietoa voidaan käyttää.

4. Käyttäjätyytyväisyys (*User Satisfaction*): Kategoria kattaa käyttäjän kokemuksen järjestelmästä, sen käytettävyydestä ja järjestelmän tuottaman sisällön hyödyllisyyden loppukäyttäjälle.

5. Henkilökohtainen vaikutus (*Invidual Impact*): Kuinka tietojärjestelmä vaikuttaa loppukäyttäjän käyttäytymiseen ja mitä hyötyjä se tuo?

6. Organisatorinen vaikutus (*Organizational Impact*): Kuinka tietojärjestelmä vaikuttaa organisaation tehokkuuteen ja käyttäytymiseen?

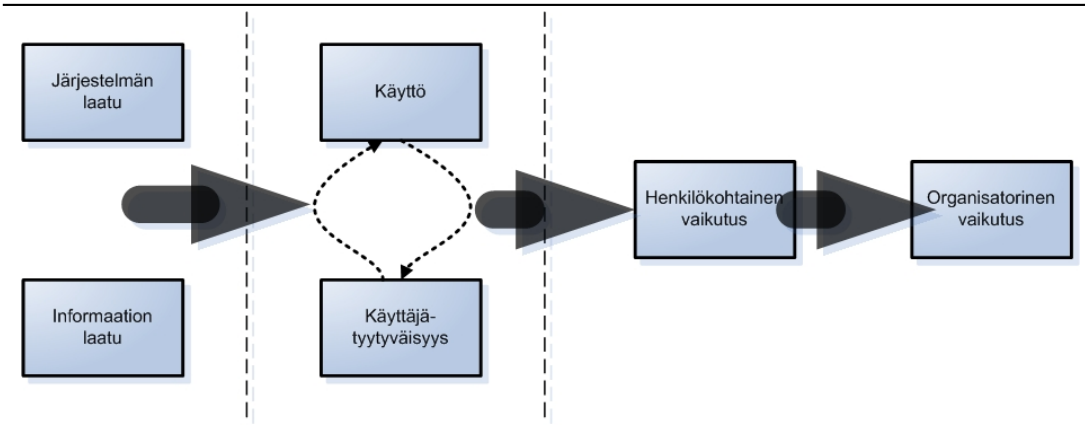
Nämä kategoriat voidaan jakaa kolmeen ylätasoon: tekniseen, semanttiseen ja vaikuttavuuteen. Tekninen sisältää järjestelmän laadun. Tässä tasossa mitataan käytännössä tietojärjestelmän suorituskykyä tuotantokäytössä. Semanttinen taso mittaa tuotteen tietosisällön oikeellisuutta ja sen hyödyllisyyttä loppukäyttäjille. Yllä olevista kategorioista informaation laatu kuuluu tähän tasoon. Kuinka hyvin tietojärjestelmän tuottama tieto vastaanotetaan ja miten helppoa ja mukavaa järjestelmän käyttäminen on? Näitä asioita mitataan vaikuttavuustasolla, johon kuuluvat käyttö, käyttäjätyytyväisyys, henkilökohtainen vaikutus ja organisatorinen vaikutus.

Käyttäjätyytyväisyys on yksi käytetyimmistä tietojärjestelmän onnistumismittareista. Suosion takana on kolme perussyytä. Ensinnäkin tietojärjestelmää voidaan pitää hyvänä, jos sen käyttäjä pitää siitä. Toiseksi käyttäjätyytyväisyyden mittaamiseen on olemassa hyviä ja luotettavia menetelmiä, kuten Baileyn ja Pearsonin [1983] menetelmä. Kolmantena syynä on se, että muut käytettävissä olevat mittarit ovat hankalasti hahmotettavissa ja vaikeita mitata.

Henkilökohtainen vaikutus keskittyy arvioimaan ihmisten käyttäytymistä. Sillä pyritään selvittämään, kuinka paljon tietojärjestelmä on vaikuttanut loppukäyttäjien päätöksentekoon. Olennaista on myös selvittää, olisivatko päätökset olleet erilaisia ilman tietojärjestelmää ja toisaalta, ovatko päätökset syntyneet nopeammin.

Organisatorinen vaikutus mittaa tietojärjestelmän hyötyjä ogranisaatiolle. Perinteisiä liiketoiminnan mittareita ovat myynnin kasvu, tuotannon tehostus ja tietenkin tietojärjestelmähankkeen näkökulmasta sijoituksen takaisinmaksuaika. Nykypäivän nopeatempoisessa liike-elämässä strategisia muutoksia tulee koko ajan. Sen takia on elintärkeää, että tietojärjestelmä pystyy mukautumaan, kun

liiketoimintaprosessit muuttuvat, yritysostot muuttavat organisaation rakennetta ja tuotantolaitoksia siirretään maantieteellisesti eri paikkaan.



Kuva 6.1 D&M-onnistumismalli

Kuten kuvasta 6.1 käy ilmi, D&M-onnistumismalli jakautuu kolmeen lohkoon. Vasemmalta oikealle ensin tulee järjestelmän tekninen suorituskyky ja semanttinen tietosisällön laatu. Nämä ovat perinteisiä tietojärjestelmän mittauskohteita. Ajan myötä tietojärjestelmistä on kuitenkin tullut luonnollinen osa ihmisten elämää ja työtä. Sen takia nykypäivän järjestelmissä pelkkä toimivuus ei ole hyvän järjestelmän perusta. Ihmisten pitää pystyä oppimaan järjestelmä helposti ja sen on oltava saatavilla heti, kun sitä tarvitaan. Käytön kautta käyttäjätyytyväisyyden voidaan olettaa lisääntyvän ja sen taas lisäävän käyttöä. D&M-onnistumismalli on tehty 1990-luvun alkupuolella, mutta se toimii hyvin edelleenkin. Kolmas lohko kuvaa tietojärjestelmän vaikutusta ihmisen ja organisaation toimintaan. Tämän merkitys on kasvanut tähän päivään asti, ja sen merkitys kasvaa koko ajan. [Delone & McLean, 1992]

Nykypäivän tietojärjestelmähankkeet käynnistetään perustuen siihen olettamukseen, että organisatoriset vaikutukset ovat järjestelmän hankkijalle positiiviset tai elinehto. Mietitäänpä hetki tietojärjestelmien vaikutusta nykypäivän yrityksiin. Tavarantoimittajaksi suurelle ostajayritykselle ei välttämättä pääse, jos toimittaja ei tue oikeita sähköisiä rajapintoja (esimerkiksi EDI, EbXML, RosettaNet). Näin ollen tavaratoimittaja voi hankkia kalliin huonosti toimivan järjestelmän pelkästään pysyäkseen markkinoilla ja pystyäkseen jatkamaan liiketoimintaansa. Monessa osto-organisaatiossa on esimerkiksi siirrytty ennusteiden laatimisesta niiden analysointiin. Käytännössä toiminnanohjausjärjestelmä luo automaattiset ostoehdotukset, ja työntekijöiden tehtäväksi jää arvioida ja tehdä

johtopäätökset.

Vaikka tässä kohdassa puhutaan tietojärjestelmän onnistumisen mittaamiseta, löytyy tästä kaksi avainta onnistuneeseen projektiin. Järjestelmän hankkijan tulee ymmärtää, miten laaja-alaisesti se vaikuttaa hankkijan toimintaan. Päätökset tulisikin tehdä kokonaisvaikutuksen eikä yksittäisten tarpeiden perusteella. Hyvässä projektissa asiakas tietää, mitä on tilaamassa. Näin ei kuitenkaan aina ole, ja siksi toinen vähintään yhtä tärkeä avain on tietojärjestelmän toteuttajan käytettävissä. Toteuttajan on tiedettävä tai otettava selvää, mitä tietojärjestelmän hankkija oikeasti tarvitsee ja varmistettava, että hankkija myös ymmärtää sen.

6.2 Onnistumismallin toinen vaihe

Delone & McLean tekivät uuden tutkimuksen kymmenen vuotta ensimmäisen artikkelin ilmestymisen jälkeen. Tässä artikkelissa ensin tiivistettiin alkuperäisen raportin päähavainnot, sen jälkeen kartoitettiin tehty jatkotutkimus ja lopuksi päivitettiin D&M-onnistumismallia lähemmäs nykypäivän tilannetta. Alkuperäisen raportin havainnot olivat:

1. Informaatiojärjestelmän käyttöönoton onnistumisen mittaamiseen käytettävä I/S Success -mittaristo on moniulotteinen ja käytettävät mittarit ovat riippuvaisia toisistaan.

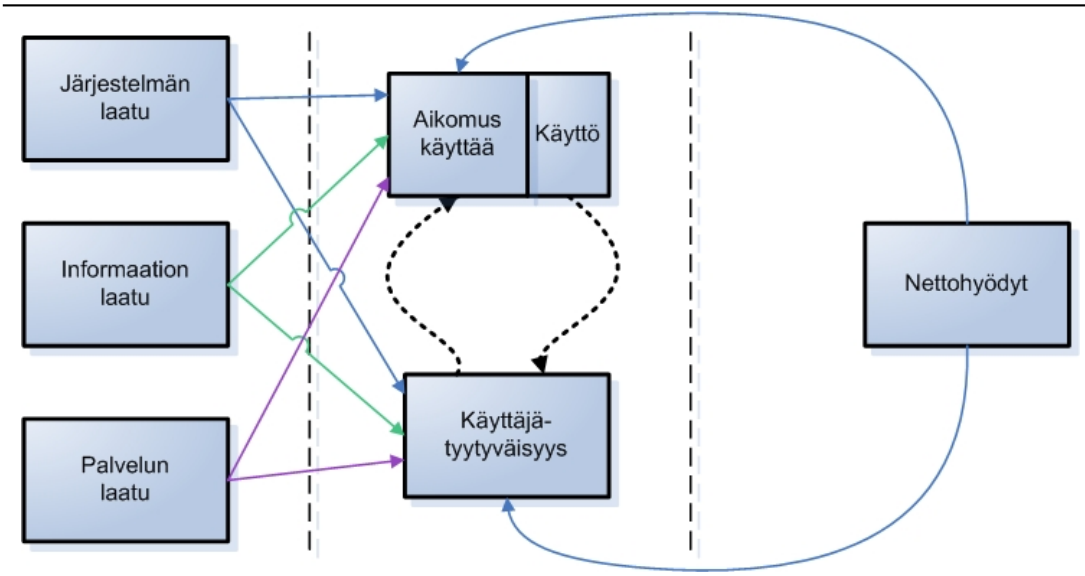
2. Tutkimukseen valittujen mittareiden ja niiden laajuuden tulee olla riippuvaisia empiirisen tutkimuksen kohteesta ja tavoitteista, mutta mahdollisuuksien mukaan on hyvä käyttää testattuja ja hyväksi todettuja mittareita.

3. Vaikka ohjelmistoprojektin onnistuminen käsitteenä perustuu useaan toisistaan riippuvaan muuttujaan, tulisi yrittää vähentää mittareita ja päästä kohti yksinkertaisempia mittausmenetelmiä.

4. Tarvitaan lisää kenttätutkimusta, ja organisaation vaikutus tulisi huomioida paremmin.

5. Tämä onnistumismalli tarvitsee lisää kehittämistä, ennen kuin sitä voi täysin hyödyntää ohjelmistoprojektin onnistumisen mittaamiseen.

Perustuen kymmenen vuoden aikana tehtyyn tutkimukseen ja omiin havaintoihinsa tutkijat ovat tehneet muutamia korjauksia D&M-onnistumismalliin. Ensinnäkin malliin on lisätty palvelun laatu uutena ulottuvuutena alkuperäisten järjestelmän ja informaation laadun rinnalle. Alunperin onnistumismallin lopputuloksena olivat vaikutukset henkilökohtaisella ja organisaation tasolla. Päivitetyssä mallissa tätä on yksinkertaistettu ja näiden tilalle on otettu uusi muuttuja: nettohyödyt (*net benefits*). Tutkijat ovat päätyneet tähän termiin, koska se kuvaa selkeämmin positiivisia vaikutuksia eli hyötyjä.



Kuva 6.2 Päivitetty D&M-onnistumismalli

Alkuperäisessä mallissa kuvassa 6.1 asiayhteys oli avoin. Tietojärjestelmähankkeen onnistumista ei voi mitata, jos ei oteta huomioon haluttua asiayhteyttä. Nettohyödyt -muuttuja voidaan helpommin määritellä kattamaan muun muassa toimittajan, tilaajan, loppukäyttäjän tai omistajan vaatimukset. Uusi muuttuja ei poista alkuperäistä ongelmaa, mutta se tekee itsestään selväksi tarpeen määritellä näkökulman, josta onnistumista tarkastellaan.

Kuvassa 6.2 nähdään, kuinka nettohyödyn, käytön ja käyttäjätyytyväisyyden välille syntyy kehävaikutussuhde. Kun käyttäjä kokee hyötyvänsä järjestelmästä, halu käyttää järjestelmää kasvaa. Tämä lisää käyttöä ja sen lisäys näkyy käyttäjätyytyväisyydessä. Tietysti voidaan todeta, että on myös järjestelmiä, joita on hankala käyttää. Tämä ei kuitenkaan sulje pois sitä tosiasiaa, että riittävät nettohyödyt lisäävät käyttäjätyytyväisyyttä. [Delone & McLean, 2003, 22-24]

6.3 Tietojärjestelmän vaikuttavuus

Vaikuttava tietojärjestelmä voi olla onnistuneen projektin lopputulos. Toisaalta vaikuttava järjestelmä voi syntyä pitkän ja monta kertaa uudelleensuunnitellun toteutusprojektin aikana. Vaikuttavuus ei siis suoraan kerro onnistuneesta tietojärjestelmähankkeesta, mutta vaikuttavuuden mittaus sisältää samoja muuttujia ja käsitteitä.

Käyttäjien uskomukset ja asenteet uutta tai olemassaolevaa tietojärjestelmää kohtaan vaikuttavat suuresti tietojärjestelmän vaikuttavuuteen [Grover *et al.*, 1996, 181]. Täysin samalla tavalla ne vaikuttavat siihen, kuinka ohjelmiston toimittajan tekemä työ otetaan vastaan. Juuri tämän takia toimittajan yksi tärkeimmistä tehtävistä jokaisessa tietojärjestelmähankkeessa on varmistaa, että palvelun tilaaja ymmärtää, mitä on saamassa. Kun uskomukset ja asenteet kohtaavat toimittajan tarjoamat palvelut ja tuotteet, edellytykset vaikuttavalle tietojärjestelmälle ja onnistuneelle projektille täyttyvät.

Vaikuttavuutta voidaan tutkia mittaamalla tietojärjestelmän vaikutusta organisaation toimintaan. Liike-elämässä organisaatioita mitataan muun muassa tuottavuudella, kannattavuudella ja sitä kautta organisaatioiden tehokkuudella. Myös tietojärjestelmän on pystyttävä näyttämään vaikutuksensa näihin muuttujiin [Grover *et al.*, 1996, 181]. Kun yhdistetään yksittäisten henkilöiden uskomukset ja asenteet organisaation liiketoiminnan vaatimuksiin, voidaan todeta, että onnistuneessa projektissa asiakas tietää mitä on tilaamassa.

Jotta ohjelmistoprojektien menetelmiä ja toimintapoja voitaisiin arvoida ja kehittää, on pystyttävä ymmärtämään ja mittaamaan tietojärjestelmän vaikuttavuutta [Grover *et al.*, 1996, 187–188]. Tähän perustuen voidaan olettaa, että hyvässä ohjelmistokehityksen prosessimallissa on jo etukäteen mietitty, miten projektin vaikuttavuutta voidaan mahdollisesti arvioida yksilön ja organisaation kannalta. Tässä tutkielmassa esitetyistä prosessimalleista Scrum- ja StepWise-mallit luovat hyvän pohjan tälle, koska molemmissa tavoite tarkentuu projektin aikana pienissä palasissa ja projektiorganisaatio pidetään hyvin ajan tasalla. Pienet palaset (prototyypit ja ohjelmistolisäykset) mahdollistavat projektin edetessä sen, että projektin osapuolet oppivat, mitä ja miten pitää mitata. Tämä on tärkeä ominaisuus, sillä kaikissa ohjelmistoprojekteissa ei ole mahdollista etukäteen tietää, mihin kaikkialle valmis ohjelmisto tulee vaikuttamaan.

6.4 Mahdollisuuksien analysointi

Mahdollisuuksien analysointi (Opportunity Analyzer ™) on Lawsonin kehittämä prosessi ja siihen liittyvä työkalu. Kyseessä on jatkuva prosessi, joka muodostuu kolmesta vaiheesta. Ensin kartoitetaan liiketoiminta ja siihen liittyvät prosessit. Toisessa vaiheessa prosesseista etsitään kehitettävät osa-alueet, ja mietitään tarvittavat parannustoimenpiteet. Kolmannessa vaiheessa listataan kaikki mahdollisuudet (kehitystoimenpiteet) tärkeysjärjestyksessä ja luodaan tiekartta (*roadmap*), jossa toimenpiteet on laitettu projektisuunnitelman muotoon. [www.lawson.com, 2008]

Liiketoimintakartoituksessa selvitetään ylhäältä-alas -menetelmällä yrityksen strategiaan ja tavoitteisiin liittyvät prosessit ja valitaan niiden kannattavuutta kuvaavat avainmittarit (KPI, *key performance indicator*). Mittarit voidaan jakaa kahteen ryhmään: tuotto ja kustannus.

Tämän jälkeen etsitään mahdollisuudet alhaalta-ylös -menetelmällä, jossa kukin liiketoimintaprosessi käydään läpi ja peilataan olemassa oleviin parhaisiin käytäntöihin. Jokaiselle näin löydetylle mahdollisuudelle lasketaan tuotto ja kustannus huomioiden parannuksen vaatima käyttöönottoaika. Esimerkiksi laskujen tulostuksen ulkoistaminen voisi olla yksi mahdollisuus. Sen toteuttaminen vaatisi uuden rajapinnan luomista ulkoistuskumppanille ja toisi säästöjä yrityksen sisäisissä kustannuksissa, kuten postituskuluissa, tulostustarvikkeissa ja henkilökuntamenoissa.

Lopuksi luodaan vaihtoehtoisia suunnitelmia, joista jokainen sisältää yhden tai useamman mahdollisuuden totetushankkeen. Laittamalla näitä suunnitelmia aikajanalle pystytään luomaan tiekartta, joka sisältää kolme asiaa. Ensinnäkin kehitystä kaipaavat liiketoimintaprosessit ja tehtäväksi suunnitellut parannukset. Toiseksi arvion muutoksista syntyvistä kuluista ja nettohyödyistä. Tässä tapauksessa nettohyödyillä tarkoitetaan yrityksen kannattavuuden parantumista. Viimeisenä ja ehkä tärkeimpänä asiana ovat mittarit. Mahdollisuuksien analysoinnin lopputuloksena syntyvät kehityshankkeiden mittaamiseen sopivat mittarit ja menetelmät.

Mahdollisuuksien analysointi sopii käytännössä kaikkiin ohjelmistojen kehityshankkeisiin. Kun yritys investoi uuteen toiminnanohjausjärjestelmään tai räätälöityyn ohjelmistokomponenttiin, on luonnollista, että liiketoiminnan johto on kiinnostunut hankkeen takaisinmaksuajasta ja siitä, kuinka järjestelmän vaikuttavuutta voidaan mitata. Mahdollisuuksien analysointi luo pohjan onnistuneelle projektille, koska se auttaa yritystä hahmottamaan, mitä pitää mitata ja miten

se tehdään. Lisäksi syntyy projektisuunnitelma, jonka avulla laajamittainen kehityshanke voidaan pilkkoa pienempiin helpommin perusteltaviin toimenpiteisiin.

6.5 Onnistuminen kenttätutkimuksessa

Ohjelmistoprojektin onnistumisen mittaamiseen on olemassa monia eri tapoja, joista edellä on esitelty muutamia. Karvinen ja muut [1994, 212] toteavat, että tietojärjestelmähankkeelle on olemassa kuusi investointilajia: korvausinvestointi, rationalisointi, välttämättömyys, laajennus, tutkimus- ja kehitys tai strateginen investointi. Investoinnilla on aina jokin tarkoitus ja investoinnin voidaan olettaa onnistuneen, jos alkuperäinen tarkoitus tuli täytettyä. Onnistumismalli, mahdollisuuksien analysointi ja tietojärjestelmän vaikuttavuus kaikki viittaavat siihen, että ei ole olemassa yhtä helppoa mittaria, jolla onnistumista voitaisiin empiirisesti mitata. Kaikki kuitenkin viittaavat siihen suuntaan, että ohjelmistokehityksessä projektin tavoite asettaa onnistumisen edellytykset.

Tämän tutkimuksen luonnollisissa kokeissa (tarkasteltavissa projekteissa) onnistumista mitataan kolmella eri tavalla. Ensinnäkin mietitään, onko projektissa käytetty prosessimalli ollut oikea valinta vai olisiko toisenlainen lähestymismalli tuonut paremman tuloksen. Toisena mittarina käytetään projekteista kerättyä laskutusaineistoa, jonka avulla on mahdollista tarkastella budjettia, ilmaisia tunteja ja mahdollisia reklamaatioita. Nämä kertovat omalta osaltaan projektin onnistumisesta. Kolmantena arvioidaan, kuinka hyvin projektin tavoite saavutettiin.

7 Projekti A: Ohjelmiston laajennuksen käyttöönotto

7.1 Tausta ja projektin tavoite

Syksyllä 2006 aloitettiin käyttöönottoprojekti projektivalmistusta tekevälle asiakasyritykselle. Tätä edelsi asiakkaan ja Lawsonin yhdessä tekemä esiselvitysprojekti, jossa tutustuttiin asiakkaan tarpeisiin ja M3-toiminnanohjausjärjestelmän käyttömahdollisuuksiin. Järjestelmä oli jo entuudestaan tuttu asiakkaalle, koska muutama vuosi aikaisemmin järjestelmä oli otettu käyttöön muutamissa muissa yksiköissä hankinnan ja projektitoiminnan prosesseihin.

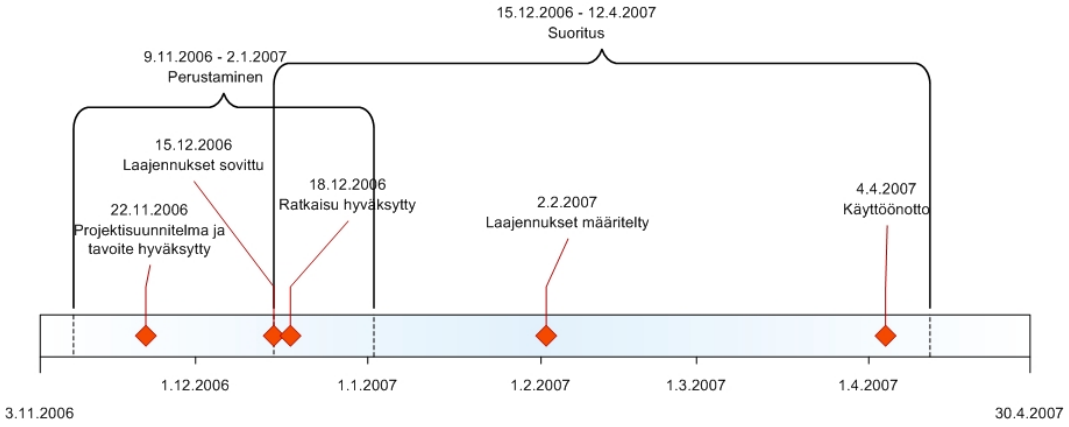
Esiselvitysprojektissa määriteltiin, että M3-järjestelmän käyttöä laajennetaan kattamaan uuden yksikön toiminta. Uuden yksikön tehtävänä oli toimia keskitettynä projektin- ja tilauksenhallintayksikkönä. Tämän käyttöönottoprojektin rinnalla asiakas rakensi itse oman ohjelmiston hoitamaan valmistusprosessia, ja sen integrointi oli olennainen osa käyttöönottoprojektia. Esiselvityksen aikana sovittiin, että käyttöönottoprojekti alkaisi lokakuussa 2006 ja päättyisi helmikuun 2007 lopussa. Alustavat työmääräarviot muodostivat noin 80 päivän kokonaisbudjetin. Tässä vaiheessa arviosta puuttui vielä integraation rakentaminen asiakkaan valmistusjärjestelmään.

Asiakkaan tavoitteena oli ottaa kaksi toisiinsa integroitua uutta järjestelmää käyttöön samaan aikaan ja näin parantaa projektitoiminnan läpinäkyvyyttä ja tehostaa toimintaa. Toimittajan näkökulmasta tavoitteena oli parametroida M3-järjestelmä asiakkaan tarpeiden mukaan, luoda tarvittavat liittymät valmistusjärjestelmään ja toimittaa projekti aikataulussa sovitun budjetin mukaan.

7.2 Projektin eteneminen

Toimin itse projektipäällikkönä tässä projektissa, ja se pääsi alkamaan noin kuukauden myöhässä esiselvityksessä ehdotetusta aikataulusta – marraskuussa 2006. Ensimmäisenä tehtävänäni oli tehdä projektisuunnitelma ja -budjetti. Valitsin projektin prosessimalliksi StepWise-mallin, koska kyseessä oli uuden osa-alueen käyttöönotto jo tuotannossa olevaan järjestelmään. Lisäksi lopullinen ratkaisu oli vielä aika avoin ja halusin, että ratkaisu on olennaisessa osassa projektia. StepWise-mallista poiketen ohjelmistoon tehtävät laajennukset toteutettiin vasta suoritusvaiheessa.

Projektin aloituspalaverissa korjasin sekä budjettia että aikataulua. Sovittu ai-

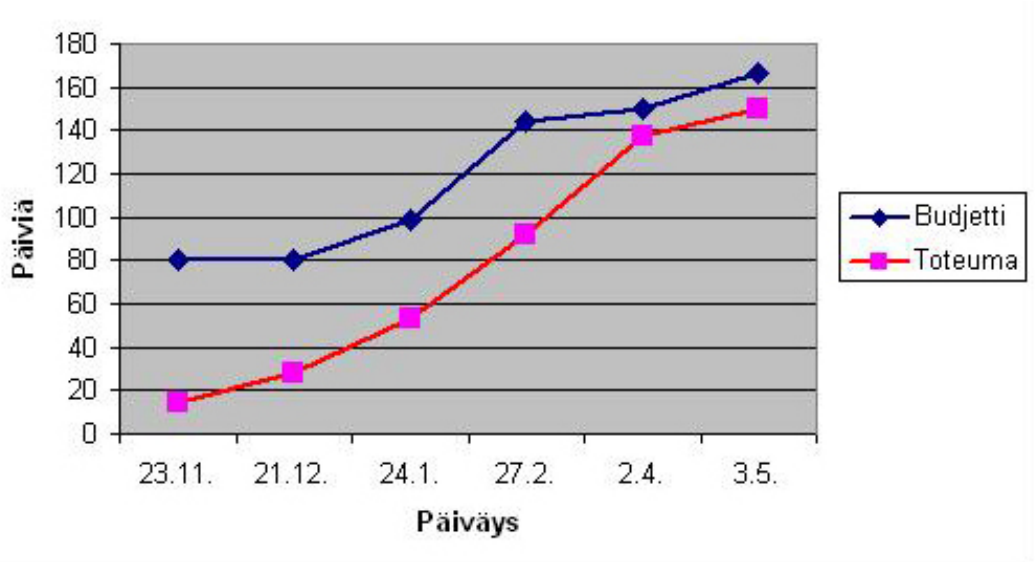


Kuva 7.1 Projekti A:n aikataulusuunnitelma vaiheittain

kataulu on esitetty kuvassa 7.1. Ensinnäkin työmääräarvioista puuttui integroinnin vaatimien liittymien toteutustyö. Korjattu kokonaisbudjetti oli 111 päivää sisältäen tässä vaiheessa tiedossa olleet lisätyötä vaativat ohjelmistolaajennukset. StepWise-mallin mukaan perustamisvaiheen budjetti oli 36 päivää ja suoritusvaiheen 75 päivää. Tämä tietenkin herätti keskustelua ja epävarmuutta asiakkaan edustajissa, mutta halusin asiakkaan tietävän alusta lähtien niin oikean tilanteen kuin oli mahdollista. Projektin aikataulua venytin noin kuukaudella eteenpäin eli käyttöönottopäiväksi valittiin 4. huhtikuuta 2007.

Perustamisvaiheen saimme valmiiksi tammikuussa 2007. Alkuperäisen aikataulun mukaan olin suunnitellut, että tämä vaihe olisi päättynyt jo joulukuussa 2006, mutta projektin aikana huomasimme, että ratkaisun lukkoon lyöminen vaatii enemmän aikaa. Lisäksi projektin aikana sovimme useista lisätöistä. Tästä johtuen korjasin budjettia projektin edetessä siten, että lopulta perustamisvaiheen osuus oli 48 päivää ja suoritusvaiheen osuus 118 päivää. Näin muodostui projektin 166 päivää sisältävä kokonaisbudjetti, joka sisälsi 50 päivää lisätöitä. Kun tätä verrataan projektin aloituspalaverissa esitettyyn budjettiin (111 päivää), huomataan, että alkuperäisen tavoitteen mukainen budjetti oli nyt hieman korkeampi, 116 päivää.

M3-järjestelmä oli valmis tuotantoon huhtikuun alussa. Tosin osa lisätöistä oli vielä tässä vaiheessa tekemättä, mutta niiden valmistuminen oli sovittu myöhemmäksi. Käyttöönotto viivästyi kuitenkin viikolla johtuen valmistuksen käyttämästä järjestelmästä, jonka käyttöönotto siirtyi viikolla. Käyttöönottokuukauden jälkeen projektin alkuperäisen tavoitteen mukaisiin töihin oli käytetty 112 päivää ja 38 päivää lisätöihin. Projektin budjetin ja toteuman vaiheet on esitetty kuvassa



Kuva 7.2 Projekti A:n budjetin kehitys verrattuna toteumaan

7.2.

7.3 Projektin onnistuminen

Käytetty StepWise-malli sopi mielestäni erittäin hyvin tähän projektiin. Ratkaisukuvaus on mallin tärkeimpiä tuotoksia, ja se ohjasi projektin kulkua alusta loppuun. Vaikka projektin aikana oli kova paine aloittaa sovittujen liittymien tekeminen mahdollisimman aikaisin, mallin ansiosta saimme ratkaisun riittävän pitkälle, ennen kuin lähdimme toteuttamaan tarvittavia ohjelmistolaajennuksia. Tämä on tärkeää, koska muuten olisimme tehneet turhaa työtä, jos lopullinen ratkaisu olisikin aiheuttanut muutoksia jo tehtyihin laajennuksiin.

Projektissa mallin vaihejako ei mennyt täysin StepWise:n mukaan. Oikean mallin mukaan kaikki käyttöönotettavat ohjelmistolaajennukset olisi pitänyt tehdä valmiiksi perustamisvaiheen viimeiseen prototyyppiin. Tästä huolimatta projekti valmistui mallin kannalta onnistuneesti aikataulussa, eikä prosessimallin aiheuttamia ongelmia ollut havaittavissa.

Projektin budjetti oli huomattavasti suurempi kuin alunperin oli suunniteltu. Tämä johtui siitä, että esiselvityksessä osa asioista oli jätetty auki ja ratkaisun tarkentuessa huomasimme enemmän lisätyötarpeita kuin projektin tavoitteessa oli mukana. Budjetin kasvaminen ei kuitenkaan aiheuttanut yhtään reklamaatiota, koska kaikki muutokset olivat perusteltuja ja hyvissä ajoin tuotu projektiryh-

män tietoon. Projektiin tehtiin ilmaista työtä 6,5 päivää, koska tämä oli yhdelle konsultille ensimmäinen projekti ja hän joutui opettelemaan työnteon ohessa M3-järjestelmän parametrointia.

Toimittajan näkokulmasta budjetti pysyi hyvin hallussa, eikä yllätyksiä tullut. Asiakkaan kannalta projektin kustannukset kasvoivat, mutta ne olivat hyvin tiedossa ja projekti pysyi aikataulussa. Koska yhtään reklamaatiota ei tehty, voidaan asiakkaan olettaa olleen tyytyväinen projektin toteutustapaan ja etenemiseen.

Projektin tavoitteena oli ottaa kaksi uutta järjestelmää käyttöön samaan aikaan ja näin tehostaa asiakkaan projektien seurantaa ja läpimenoa. Molemmat järjestelmät otettiin käyttöön lähes sovitussa aikataulussa ja niiden välinen integrointi toimi. M3-järjestelmän lopullinen ratkaisu olisi kuitenkin voinut olla yksinkertaisempi ja osittain sen takia kaikkia haluttuja toimintoja ja ominaisuuksia ei päästy täysin hyödyntämään. Tavoitteen kannalta projekti onnistui osittain, koska kaikkia toimintoja ei voitu ottaa käyttöön, vaan ne jäivät mahdollisiksi jatkokehityshankkeiksi.

7.4 Projektin haasteet ja opit

Projektin olennaisin haaste toimittajan kannalta oli tiukka aikataulu ja budjetti. Lisäksi esiselvityksessä suunniteltu ratkaisumalli oli erittäin monimutkainen. Projekti vahvisti sitä tietoa, että kaikki muutokset tulee kommunikoida mahdollisimman aikaisin. Vaikka projektin läpivienti vaati paljon enemmän työtä kuin oli oletettu, asiakas tiesi koko ajan, mitä hänelle oltiin toimittamassa.

Toinen asia, minkä projekti opetti, oli se, että kaikkia olettamuksia vaatimusmäärittelystä ja tehtävästä ratkaisusta tulee kyseenalaistaa. Jos olisimme heti projektin alussa arvioineet suunnitellun ratkaisun lähestymistavan ja miettineet vaihtoehtoisia toteutustapoja, lopullinen ratkaisu olisi todennäköisesti ollut erilainen ja ennen kaikkea yksinkertaisempi. Hieman eri lähestymistavasta tehty ratkaisu olisi myös kattanut projektin tavoitteet paremmin. Tämä olisi voitu tehdä lisäämällä StepWise-mallin perustamisvaiheen alkuun kaksi rinnakkaista prototyyppiä, jotka olisivat kuvanneet vaihtoehtoisia toteutustapoja.

8 Projekti B: Käyttöönotto

8.1 Tausta ja projektin tavoite

Keväällä 2007 aloitettiin käyttöönottoprojekti LVI-tuotteita, kunnallistekniikkaa ja teollisuuden putkituotteita myyvälle tukkuliikkeelle. Asiakasyritys kuuluu kansainväliseen konserniin. Tässä projektissa tarkoituksena oli toteuttaa Suomen yhtiön käyttöön sopiva taloushallinnon tietojärjestelmäratkaisu. Asiakas valitsi M3-järjestelmän, koska sen talousmoduulin standardiprosessit sopivat heidän toimintatapoihinsa ja sen järjestelmäarkkitehtuuri oli nykyaikainen.

Lawsonin projektipäällikkö ja asiakas määrittelivät projektin keskeisimmiksi tavoitteiksi raportoinnin parantamisen konsernin vaatimusten mukaan, manuaalisen työn vähentämisen, tytäryhtiöiden liittämisen mahdollistamisen ja järjestelmäarkkitehtuurin nykyaikaistamisen. Uudelta talousjärjestelmältä odotettiin saavutettavan sekä nopea ja helppokäyttöinen toiminta että tarkka, analysoitavissa oleva tietosisältö. Lisäksi projektin tavoitteisiin kuului toimittajan ja asiakkaan jatkuva yhteistyö ja avoimuus.

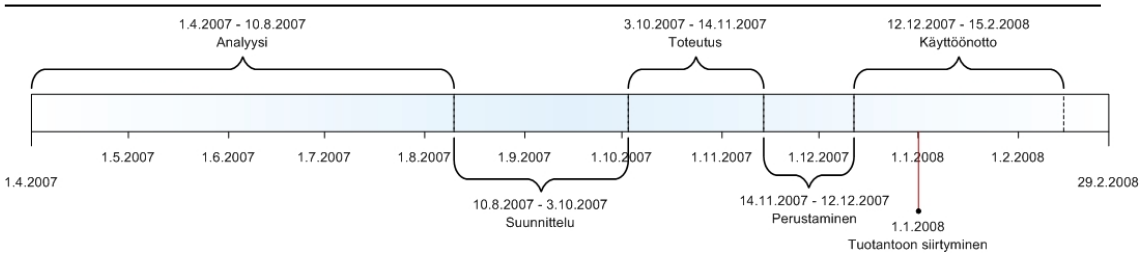
Tässä projektissa myös onnistumistekijät kirjattiin projektisuunnitelmaan. Ensinnäkin käyttöönotettavan ratkaisun tulee kattaa asiakasyrityksen liiketoimintaprosessit ja henkilökunnan on opittava käyttämään järjestelmää. Toiseksi järjestelmän tulee toimia vaadituista tehtävistä riittävän nopeasti ja sen käyttämisen tulee olla helppoa. Viimeisenä onnistumistekijänä suunnitelmaan kirjattiin, että projektin aikataulun ja budjetin pitää pysyä hallinnassa.

Tämän luvun tiedot perustuvat vuonna 2007 kirjoitettuun projektisuunnitelmaan, projektin aikana syntyneeseen raportointiaineistoon [2008b], asiakkaan projektista antamaan palautteeseen ja tutkijan omaan kokemukseen hankkeesta.

8.2 Projektin eteneminen

M3-talousmoduulin käyttöönottomenetelmäksi valittiin Implex-malli. Kuten kuvaan 8.1 on havainnollistettu, projektisuunnitelmaan kirjattiin käyttöönottopäiväksi 1.1.2008. Tätä edeltävät vaiheet suunniteltiin siten, että analyysivaihe loppuisi elokuun alussa, suunnitteluvaihe lokakuun alussa, toteutusvaihe marraskuun puolivälissä, perustamisvaihe joulukuun puolivälissä ja käyttöönottovaihe helmikuun 2008 puolivälissä.

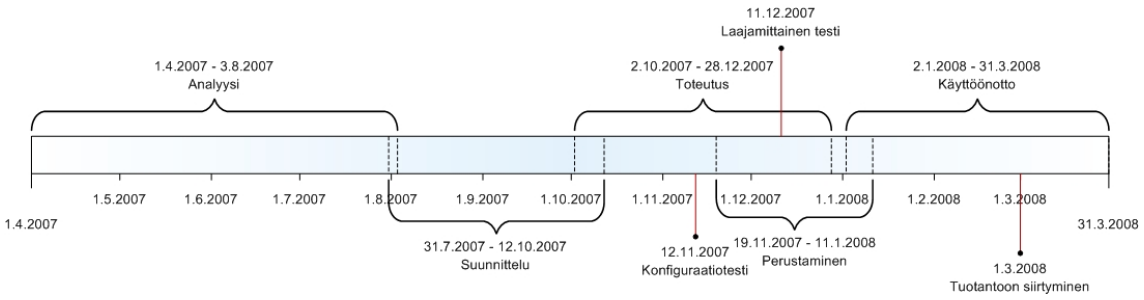
Toimitussopimuksessa projektin kokonaisbudjetti oli 104 päivää, johon sisältyi neljä konversiota ja 7 liittymää. Heti projektin alussa projektipäällikkö perusteli 5 päivän lisäbudjetin alkuperäiseen 104 päivän budjettiin, koska projektin



Kuva 8.1 Projekti B:n aikataulusuunnitelma vaiheittain

suunnittelun ja hallinnoinin työmääräarviot olivat liian alhaiset. Konversioiden ja liittymien lisäksi projektiin ei kuulunut yhtään asiakaskohtaista muutostyötä. Projektin onnistumisen kannalta palkka- ja pankkiliittymät olivat keskeisessä osassa.

Projektin suunnitteluvaihe eteni aikataulun mukaan, mutta M3-järjestelmän asennukset veivät enemmän aikaa kuin oli ennustettu. Tämä johtui siitä, että M3:sta oli juuri tullut uusi versio ja sen asentamisesta oli vähän kokemusta. Tästä johtuen asennustöille varattu budjetti ylittyi, mutta projektin kokonaisbudjetti pysyi hallussa, koska esimerkiksi liittymiin ja konversioihin kului paljon vähemmän aikaa kuin oli odotettu.



Kuva 8.2 Projekti B:n toteutunut aikataulu vaiheittain

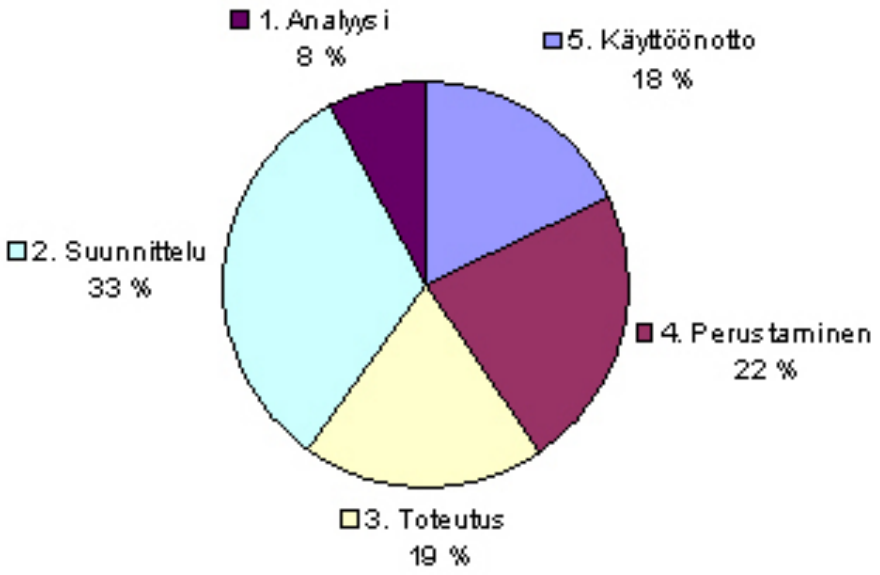
Kuvasta 8.2 nähdään vaiheittainen aikataulu, joka lopulta toteutui. Mielenkiintoista on se, että Implex-mallin mukaiset toteutus- ja perustamisvaiheen testit tehtiin alkuperäisen aikataulun mukaan, vaikka molemmat vaiheet venyivät ja menivät jopa päällekkäin. Tämä johtui siitä, että M3:n perusratkaisu oli valmis alkuperäisen aikataulun mukaan, mutta liittymien lopullinen testaus venyi. Toteutusvaihe jatkui pidempään, koska liittymien testaus- ja kehitystyö jatkui aina vuodenvaihteeseen asti. Käyttöönotto olisi voitu kiirehtimällä tehdä vuoden alussa, mutta lopullinen syy siirtoon löytyi infrastruktuurin aiheuttamista haasteista

liittymäliikenteen avaamiseen.

Käyttöönotto siis siirtyi kahdella kuukaudella eteenpäin. Kun projekti jatkuu pidempään, on selvää, että siihen kuluu enemmän työaikaa. Tästä huolimatta projekti päättyi 116 toteutuneeseen päivään.

8.3 Projektin onnistuminen

Tämä projekti on malliesimerkki siitä, kuinka hyvin suunniteltu on puoliksi tehty. Vaikka projektin aikana kohdattiin useita ongelmia, kuten M3-järjestelmän uuden version asennukset ja liittymien tarvitsemien yhteyksien rakentaminen, projekti pysyi budjetissa ja korjatussa aikataulussa. Implex-mallin parhaita puolia on sen suunnitelmallisuus. Samalla se on sen huono puoli, koska pitkä suunnitteluvaihe aiheuttaa projektille liian helposti laajat ja vaikeat tavoitteet. Tässä projektissa näin ei kuitenkaan käynyt.



Kuva 8.3 Projekti B:n toteuma vaiheittain

Työpäivien jakautuminen on esitetty Implex-mallin mukaisesti kuvassa 8.3. Toteumassa kannattaa kiinnittää huomiota seuraaviin kohtiin. Suunnitteluvaihe on selkeästi työläin osuus, ja se kuluttikin kolmasosan koko budjetista. Tähän vaiheeseen kuuluvat myös ohjelmiston perusasennukset. Toteutusvaihe sen sijaan on suhteellisen lyhyt. Viimeisenä asiana havaitaan, että käyttöönottovaiheeseen on kulunut melkein yksi viidesosa kokonaisbudjetista.

Budjetin kannalta projekti onnistui, kun todetaan, että budjetti ylittyi 6.4%:lla ja se jakautui lähes odotetusti valitun prosessimallin mukaan. Lisäksi projektiin ei tehty ilmaista työtä, eikä asiakas esittänyt yhtään reklamaatiota. Projektin suunnittelusta voidaan kuitenkin todeta, että toteutusvaiheelle oli varattu liian vähän aikaa. Koska liittymät olivat tärkeässä osassa projektia, vaikuttavat myös projektin ulkopuolisten henkilöiden ja järjestelmien aikataulut toteutustyöhön.

Projektille esitetyt tavoitteet täyttyivät yhtä lukuunottamatta. Tietosisällön tarkkuudessa ja luotettavuudessa oli vielä käyttöönottovaiheen jälkeen avoimia asioita. Lähinnä pankkiliittymien käytön suoraviivaistaminen ja hienosäätö jäivät ylläpidon ja jatkokehityksen tehtäviksi.

Onnistumistekijöiksi määritelty projektin aikataulu ei mennyt alkuperäisen suunnitelman mukaan, mutta muilta osin onnistumiskriteerit täyttyivät ja näin ollen projektia voidaan pitää kohtuullisen hyvin onnistuneena.

8.4 Projektin haasteet ja opit

Toimittajan kannalta projektin selkeimmät haasteet olivat suhteellisen lyhyt käyttöönottoprojekti ja uusi versio M3-järjestelmästä. Molemmista kohdista jäi opittavaa. Ensinnäkin aikataulun suunnitteluun pitää huomioida riskitekijät paremmin. Tiukkaa aikataulua ei voi rakentaa, jos siihen vaikuttavat tekijät eivät ole tiedossa. Tämä havaitaan vertailemalla toteutusvaiheen suunniteltua ja todellista kestoa. Todellisuudessa vaihe kesti paljon pidempään, vaikka siihen kului vähemmän aikaa kuin budjettiin oli varattu.

Toinen oppimista vaativa kehityskohde on projektin eri osa-alueiden työmääräarvioiden tarkempi ennustaminen. Tässä projektissa kokonaistyömäärä pysyi hallinnassa, mutta eri aliprojekteissa (konversiot, liittymät, asennukset jne.) arviot heittivät lähes kymmenen prosenttia. Projekteista pitäisi pystyä oppimaan ja sitä kautta ennustamaan työmääräarvioita paremmin. Ennustamisen avulla toimittaja tietää, mitä on toimittamassa. Tämä taas mahdollistaa onnistuneen projektin, koska budjetissa ja aikataulussa pysyminen on monen projektin onnistumiskriteeri.

9 Projekti C: Käyttöönotto

9.1 Tausta ja projektin tavoite

Vuoden 2004 alussa projektivalmistusta tekevä asiakas ja Lawson suorittivat yhteisen pilottiprojektin, jonka tarkoituksena oli tehdä esiselvitys M3-järjestelmän käyttöönottoprojektia varten. Asiakkaalla oli jo käytössään M3-järjestelmän vanhempi versio yhdessä toimintayksikössään. Esiselvitysprojektissa kartoitettiin asiakkaan kokonaishanke, johon kuuluivat vanhan jo käytössä olevan version päivitys, uuden toimintayksikön siirtäminen M3-järjestelmään ja myöhemmin tehtävät käyttöönotot asiakasyrityksen ulkomailla toimiviin tytäryhtiöihin. Toimitussopimuksessa sovittiin kolmesta erillisestä projektista.

Asiakas valitsi M3-järjestelmän, koska jo käytössä ollut M3-järjestelmä haluttiin päivittää uudempaan ja integroida uuden käyttöönotettavan yksikön toimintaan. Integrointi oli asiakkaalle strategisesti tärkeää, koska kolmea eri saman konsernin yhtiötä oltiin yhdistämässä yhdeksi yhtiöksi. Lisäksi onnistunut esiselvitysprojekti tuki järjestelmätoimittajan valintaa.

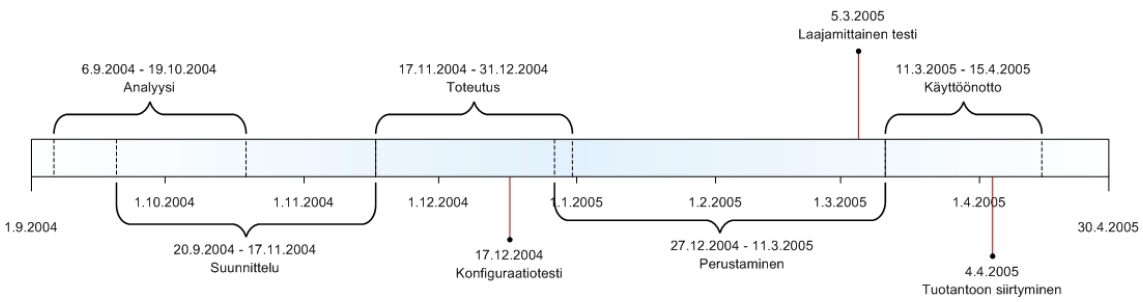
Tämä luku käsittelee M3-järjestelmän käyttöönottoprojektia uudelle toimintayksikölle. Projektin tärkeimmät tavoitteet olivat: yksi yhteinen järjestelmä, reaaliaikaiset tarkat raportit, asiakkaan prosesseja hyvin tukeva järjestelmä ja asiakkaan liiketoiminnan tavoitteiden mukaan ajallaan tehty käyttöönotto. Lisäksi projektilta vaadittiin hyvää yhteistyötä asiakkaan ja toimittajan välillä, ajantasaista dokumentaatiota ja M3-järjestelmän vakioprosessien hyödyntämistä, jotta vältyttäisiin asiakaskohtaisilta ohjelmistolaajennuksilta.

Projektisuunnitelmaan kirjattiin laajat ja kohtalaisen avoimet onnistumisen kriteerit, jotka koskivat sekä asiakasta että Lawsonia. Lawsonin kannalta tärkeimpiä onnistumisen ehtoja olivat: M3-järjestelmä ei huononna asiakkaan liiketoiminnan prosessien toimivuutta, projekti etenee aikataulussa, budjetissa ja hyvässä yhteistyössä, asiakaskohtaisten ohjelmistolaajennusten määrä ja asiakkaan ja Lawsonin yhteistyön jatkuvuus vielä projektin jälkeen.

Tämän luvun tiedot perustuvat vuonna 2004 tehtyyn toimitussopimukseen ja projektisuunnitelmaan. Lisäksi projektin etenemistä kuvataan pohjautuen Lawsonin tuntiraportointiaineistoon [2008c] ja tutkijan omaan kokemukseen projektista.

9.2 Projektin eteneminen

Projekti eteni Implex-mallin mukaan, joka oli tuolloin ainoa Lawsonilla käytössä ollut käyttöönottomalli. Projektisuunnitelma oli liitettynä toimitussopimukseen, joten projektin alkaessa tai sen aikana sitä ei enää juurikaan muutettu. Suunnitelman mukaan järjestelmän käyttöönoton oli tarkoitus tapahtua 4.4.2005. Kuvassa 9.1 havainnolistetaan, kuinka Implex-mallin eri vaiheet oli jaksotettu kalenteriin.

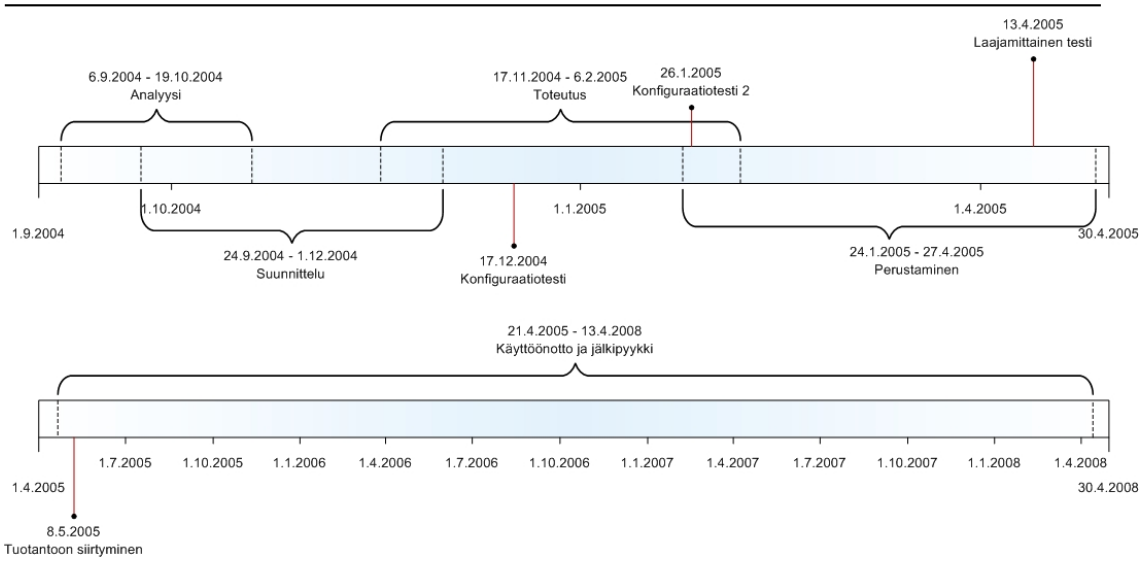


Kuva 9.1 Projekti C:n aikataulusuunnitelma vaiheittain

Projektin toimitussopimuksessa oli tarkkaan määritelty kiinteähintainen budjetti, joka sisälsi käyttöönottoprojektin lisäksi muutamia asiakaskohtaisia laajennuksia. Lisäksi projektin edetessä sovittiin muista lisätöistä, joista osalla oli oma kiinteähintainen budjetti. Koska kaikkea projektin aikana syntynyttä kirjeenvaihtoa ei ole käytettävissä tätä tutkimusta varten, lisätöiden budjetti on johdettu toteutuneista kustannuksista. Projektin toimitussopimuksessa sovittu kiinteä budjetti oli 335 päivää, joka tässä tutkimuksessa on jaettu tasan eri Implex-mallin vaihdein välille (67 päivää jokaista vaihetta kohti). Tämän lisäksi erikseen veloitettavia töitä oli 220 päivää.

Projektin analyysivaihe eteni aikataulun mukaan, mutta jo seuraava suunnitteluvaihe kesti pidempään kuin alunperin oli aiottu. Tässä vaiheessa havaittiin uusia puutteita ratkaisussa, ja tarvittavien laajennusten alustava suunnittelu vaati oman aikansa. Toteusvaihe aloitettiin kuitenkin ajallaan, mutta konsulteille työskenteleminen kahden päällekkäisen vaiheen töiden kanssa oli hankalaa. Tästä johtuen työskentely hidastui ja viimeistään konfiguraatiotestissä huomattiin, että suunniteltu ratkaisu oli vielä liian kaukana tavoitteesta. Toteutusvaihetta jatkettiin kuukaudella ja konfiguraatiotesti järjestettiin uudelleen tammikuun 2005 lopussa. Toinen konfiguraatiotesti meni riittävän hyvin läpi ja siirryttiin perustamisvaiheeseen.

Suurin osa asiakaskohtaisista laajennuksista oli vielä tässä vaiheessa tekemättä ja itse asiassa osittain suunnittelematta. Suunnittelun viivästyminen on varsin yleistä ja varsinkin, kun kyseessä on kiinteähintainen projekti, asiakas pyrkii Implex-mallin suunnitteluvaiheessa sisällytämään kaiken mukaan projektiin. Implex-mallin mukaan perustamisvaiheessa ei enää pitäisi tehdä ohjelmointia tai parametrointia.



Kuva 9.2 Projekti C:n toteutunut aikataulu vaiheittain

Kuvassa 9.2 on esitetty toteutunut aikataulu. Perustamisvaihetta ja käyttöönottoa siirrettiin noin kuukaudella eteenpäin. Järjestelmä saatiin kuin saatiinkin tässä muuttuneessa aikataulussa tuotantoon, mutta projektin aikana sovitut lisätyöt venyttivät projektia aina vuoteen 2008 asti. Järjestelmän käyttöönoton jälkeen jäi selvitettäväksi useita avoimia asioita. Lisäksi muutama toteuttamaton asiakaskohtainen laajennus tehtiin vasta käyttöönoton jälkeen. Tässä kohtaa on hyvä muistuttaa, että lähes kaikki lisätyötkin oli myyty asiakkaalle kiinteään hintaan.

Projektiin liittyvistä viimeisistä avoimista asioista sovittiin joulukuussa 2007. Huhtikuussa 2008 vietiin viimeinen asiakaskohtainen laajennus tuotantoon. Tuolloin projektin toteutunut budjetti oli 1205 päivää. Budjetin tarkempi analysointi on seuraavassa projektin onnistumista käsittelevässä kohdassa.

9.3 Projektin onnistuminen

Tässä projektissa yhdistyi monta toimittajan kannalta epäonnistumiseen johtavaa tekijää. Ensinnäkin projekti myytiin kiinteähintaisena, mutta sopimus jätti kuitenkin auki, mitä oikeasti oltiin toimittamassa. Kun tähän lisätään huonosti hoidettu muutoksenhallinta ja ostamisen osaava asiakas, on projekti valmis luisumaan sivuraiteelle.

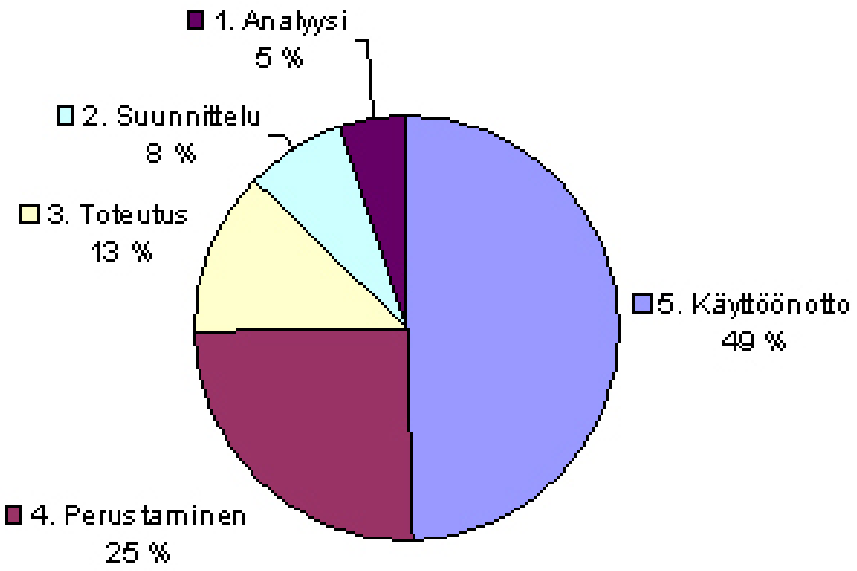
Implex-malli esitti huonot puolensa tässä projektissa. Se ei yksinkertaisesti sovi tiukkaan pakettihinnoiteltuun projektiin, koska suunnitteluvaihe käytännössä pistää asiakkaan kuvaamaan nykytilansa ja vasta sitä kautta tavoitetilan. Näin syntyvät vaatimukset, joiden pohjalta uuden järjestelmän pitää sisältää kaikki nykytilan hyvät asiat ja paikata kaikki vanhan puutteet. Tämä on ongelmallista toiminnanohjausjärjestelmien kannalta, koska niiden käyttöönottoprojekteissa on tarkoitus hyödyntää käyttöönotettavan järjestelmän standardiprosesseja. Nykytilan liian tarkka läpikäynti johtaa usein turhiin asiakaskohtaisiin laajennuksiin.

Jos olisimme käyttäneet StepWise-mallia, prototyypit olisivat lähentyneet asiakkaan toivomaa ratkaisua koko ajan. Protoilua tehdään yhdessä asiakkaan kanssa ja samalla projektin viikko-ohjelmaan kuuluu koulutuspäiviä, joissa opetellaan toiminnanohjausjärjestelmän standardiprosesseja. Samalla kun asiakas oppii käyttämään järjestelmää, vähentyy muutostoiveiden määrä ja tätä kautta asiakaskohtaisia laajennuksia tulee vähemmän. Tässä projektissa tiukempi ja selkeämpi lähestymistapa asiakaskohtaisiin laajennuksiin olisi auttanut paljon.

Projektin toteumasta puolet syntyi käyttöönottovaiheessa. Tähän vaiheeseen kuuluvat kaikki työt projektin sulkemiseen asti. Tämän lisäksi perustamisvaiheessa tehtiin yksi neljäsosa töistä. Prosessimalliin nähden jakauma on pahasti pielessä, koska mallin mukaan työläimpiä vaiheita ovat suunnittelu- ja toteusvaihe. Kuvassa 9.3 havainnollistetaan jokaisen vaiheen osuus työpäivistä.

Käyttöönottovaiheseen tehtiin kaiken kaikkiaan 594 päivää töitä. Mielenkiintoista on se, että tästä alkuperäisen kiinteähintaisen budjetin mukaisia päiviä oli 67 ja lisätöitä 130. Ilmaisia työpäiviä jouduttiin tekemään 397 päivää. Budjetoidun, ilmaisen ja lisätyön kehitys vaihe vaiheelta on esitetty kuvassa 9.4. Projekti epäonnistui budjetin kannalta pahasti. Tosin asiakkaan kannalta järjestelmä saatiin käyttöön lähes ajallaan, mutta käyttöönottovaiheessa sieltä vielä puuttui osa tavoitelluista ominaisuuksista. Asiakas esitti muutaman työn laatua koskevan reklamaation. Tämä puoltaa vahvasti sitä, että asiakaskaan ei ollut tyytyväinen projektiin.

Projektin päätavoitteista kahdessa jäätiin selkeästi jälkeen. Ensinnäkin pro-



Kuva 9.3 Projekti C:n toteuma vaiheittain

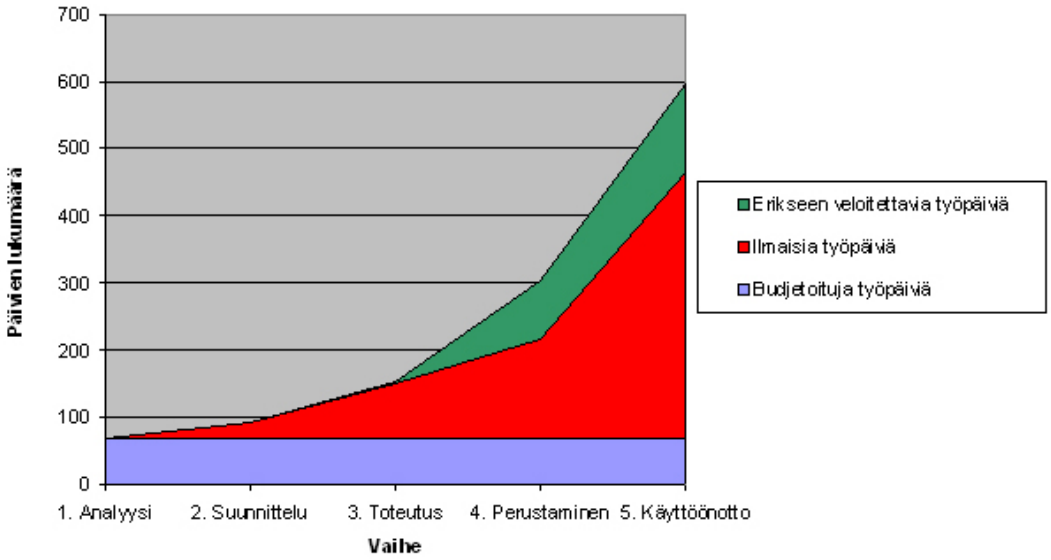
jekti viivästyi ja ajallaan toimittaminen oli yksi tavoitteista. Toiseksi asiakkaan prosessit eivät olleet täysin tuetut tuotantoonsiirron jälkeen, koska osa asiakaskohtaisista laajennuksista oli vielä tekemättä. Vaikka projekti muilta osin onnistui, voidaan projektia pitää epäonnistuneena tavoitteiden kannalta.

Projektille määritellyissä onnistumistekijöissä epäonnistuttiin pahasti. Tarkalleen ottaen ainoastaan yhteistyö on mennyt kohtalaisen hyvin, koska tämän projektin rinnalla on tehty muitakin projekteja samalle asiakkaalle. Yhteistyö siis toimi, mutta muilta osin tavoite jäi osittain saavuttamatta

9.4 Projektin haasteet ja opit

Usein projektit alkavat myyntivaiheen aikatauluun nähden myöhässä, koska sopimusneuvottelut venyttävät myyntivaiheen loppuun viemistä. Kuitenkaan harva projektipäällikkö uskaltaa heti projektin alussa kyseenalaistaa alkuperäistä suunnitelmaa. Tässä projektissa vaiheet olisi pitänyt työstää väkisin loppuun eikä aloittaa kesken kaiken seuraava. Koska asiakkaan tuleva fuusiotilanne oli edessä, olisi projektin tavoitetta pitänyt rajata ja sitä kautta saada projekti valmiiksi aikataulussa. Tavoitteesta poistetut ominaisuudet olisi voitu tehdä onnistuneemmin jatkokehityshankkeina.

Lisäksi tämä projekti opettaa, että koskaan ei pidä jättää mitään arvailujen



Kuva 9.4 Projekti C:n työpäivien laskutettavuus

varaan. Kaikki muutokset standarditoimintaan pitää ensin suunnitella toiminnallisesti ja teknisesti. Vasta sen jälkeen voidaan arvioida toteutuskustannuksia. Tiukka muutostenhallinta on olennainen osa onnistunutta projektia. Tämä oppi voidaan helposti tiivistää seuraavaan virkkeeseen: Toimittaja tietää, mitä on toimittamassa.

Asiakaskohtaisten laajennusten määrän vähentäminen onnistuu parhaiten, kun asiakas tietää mitä on saamassa. Tämä onnistuu parhaiten, jos asiakas pääsee alusta asti opettelemaan käyttöönotettavaa järjestelmää oikein. Protoilun ja koulutuksen yhdistäminen jatkuvaksi prosessiksi vähentää asiakkaan epävarmuutta ja sitä kautta muutosten tarvetta.

10 Yhteenveto

Ensimmäinen tutkimustavoite oli esitellä ohjelmistokehitykseen prosessimallit sovitettuna samaan viitekehykseen. Neljäs ja viides luku sovittivat lähdekirjallisuudesta tutkimukseen mukaan otetut prosessimallit yleiseen viitekehykseen. Näin prosessimallit saatiin esiteltyä yhteisen käsitteistön avulla ja niiden vertaileminen oli mahdollista.

Tämän tutkielman perusteella ei voida esittää yksiselitteisiä valintaehtoja, joiden avulla prosessimalleista pystyisi valitsemaan aina juuri sen oikean. Tutkielma antaa kuitenkin hyvän pohjan ymmärtää, mitä asioita tulee ottaa huomioon valittaessa sopivaa prosessimallia ja mihin asioihin valittu prosessimalli vaikuttaa.

Tutkielmassa esitettävän aineiston perusteella voidaan todeta, että suunnitelmallinen ja loppukäyttäjän hyvin huomioiva lähestymistapa kehitystyöhön takaa ohjelmistolle paremman laadun ja pidemmän käyttöiän. Käyttöikä pitenee, koska ohelmistossa on vähemmän virheitä ja puuttuvia ominaisuuksia. Sopivan prosessimallin valitsemiseen vaikuttavat myös käytettävissä olevat resurssit, aikataulu ja ohjelmiston laatuvaatimukset. Esimerkiksi lentokoneen autopilotin täytyy toimia jokaisessa mahdollisessa tilanteessa, mutta digitelevision tekstityksessä sallitaan (tosin ei toivota) pieniä viiveitä ja virheitä.

Laajoissa prosessimalleissa otetaan huomioon muutoksenhallinta ja ihmisten sopeuttaminen uuteen järjestelmään. Sekä Scrum-malli että StepWise-malli ottavat hyvin huomioon ohjelmistotuotteen ominaisuuden muuttua jatkuvasti ja ihmisten kyvyn oppia uusi asioita. Näiden prosessimallien avulla tietojärjestelmäprojekti pystytään pitämään tavoitteessa, vaikka tavoite muuttuukin projektin edetessä. Scrum-malli sopii mielestäni hyvin uuden ohjelmiston kehittämiseen ja StepWise-malli vastaavasti parametroitavan ohjelmiston käyttöönottoon. Molemmat hyödyntävät tehokkaasti ”hajoita ja hallitse” -taktiikkaa, jossa tavoite pilkotaan pieniin helposti toteutettaviin ja opittaviin palasiin.

Toisena tutkimustavoitteena oli tietojärjestelmäprojektin onnistumisen mittaaminen. Kuten kuudennesta luvusta käy ilmi, tietojärjestelmähankkeen onnistumisen mittaaminen tapahtuu usealla toisistaan riippuvaisella moniulotteisella mittarilla. Nämä mittarit voidaan jakaa kolmeen ryhmään: laatu, ihmisen ja tietojärjestelmän vuorovaikutus sekä nettohyödyt.

Yhtä yksiselitteistä onnistumisen mittaria ei siis ole olemassa. Tässä tutkimuksessa onnistumista mitattiin palvelun laadun ja nettohyötyjen näkökulmasta. Palvelun laatua arvioitiin käyttöönottoprojekteissa käytettyjen prosessimallien sopivuuden, toteutuneiden työmäärien ja aikataulujen avulla. Projektien ta-

voitteet sen sijaan havainnollistivat tietojärjestelmältä odotettavia nettohyötyjä.

Teoriasta ja kenttäkokeesta voidaan päätellä, että projektin onnistumista voidaan edesauttaa ja jopa ennustaa. Kohdassa 6.5 esitelty mahdollisuuksien analysointi on hyvä keino suunnitella tietojärjestelmäinvestointeja liiketoiminnan vaatimusten mukaisella tavalla, jonka myös yrityksen johto ymmärtää. Hyödyntämällä toistettavia prosessimalleja ja projektin aikana kerättyä mallin mukaista aineistoa voidaan parantaa sekä toimittajan palvelun laatua että projektin kokonaiskustannusten arviointia.

Mielestäni projektinhallinnan kehittämisen kannalta pitäisi luoda vakiomittaristo, joka sopii kohdeyrityksen käyttämään prosessimalliin. Jokainen projekti analysoitaisiin jälkikäteen projektista ulkopuolisen henkilön avulla vakiomittaristoon perustuen ja tulokset tallennettaisiin yhteiseen tietovarastoon. Tietovarastoon tallentuisi jokaisen projektin arviot ja toteutuneet työmäärät. Kun seuraavan kerran arvioitaisiin uuden projektin työmääriä, voitaisin virhemarginaali laskea regressioanalyysin avulla. Mitä enemmän projekteja tietovarastoon tallennettaisiin, sitä tarkemmiksi työmääräarviotkin tulisivat.

Viimeinen tutkimukseni tavoite oli kaikista haastavin. Tutkimukseni tavoitteena oli luoda yksinkertaiset projektin onnistumisperiaatteet pohjautuen lähdekirjallisuuteen ja kenttäkokeisiin. Tämän tutkimuksen perusteella voidaan todeta, että tietojärjestelmän toimittaja pystyy mahdollistamaan onnistuneen projektin, kun noudatetaan seuraavia periaatteita:

1. Asiakas tietää, mitä on tilaamassa ja mitä hänelle ollaan toimittamassa.

2. Toimittaja tietää, mitä asiakas olettaa saavuttavansa tietojärjestelmällä.

3. Toimittaja tietää, mitä on toimittamassa ja miten sen tekee.

Ensimmäinen ja toinen periaate voidaan perustella tietojärjestelmän onnistumismallin ja vaikuttavuuden avulla. Kuudennessa luvussa tuodaan esille, että onnistuneessa projektissa liiketoiminnan asettamien tavoitteiden (nettohyötyjen) tulee olla realistiset. Lisäksi asiakkaan täytyy huomioida uuden tietojärjestelmän vaikutukset yksittäisiin ihmisiin ja organisaatioon.

Myös kenttäkokeet puolustavat tätä väitettä. Projekti A:n tavoite jäi saavuttamatta, koska asiakas ei täysin ymmärtänyt, kuinka uusi järjestelmä tulee tukemaan liiketoiminnan prosesseja ja mitä puutteita toteutettavassa ratkaisussa

tulee olemaan. Projekti B onnistui hyvin, koska asiakkaalla oli selkeä kuva M3-järjestelmän talousmoduulista ja tulevasta projektista. Vaikka projektin aikataulu venyi, asiakas oli tyytyväinen, koska asetetut tavoitteet saavutettiin.

Projekti C on malliesimerkki tilanteesta, jossa toimittaja ei tiedä, mitä asiakas olettaa saavuttavansa uudella tietojärjestelmällä. Varsinkin asiakaskohtaiset ohjelmistolaajennukset toimittajan tulee arvioida tarkasti, jotta vältyttäisiin yllätyksiltä. StepWise-mallissa prototyyppin iterointi kasvattaa sekä asiakkaan että toimittajan tietämystä.

Kolmannella periaatteella tarkoitetaan yksinkertaisesti sitä, että toimittaja osaa ennustaa projektin työmäärät ja kustannukset oikein. Lisäksi projektia myytäessä ei saa olla avoimia tehtäviä, joiden laajuutta ja tavoitetta ei tiedetä, eikä ratkaisun toimittamiseen tarvittavia työkaluja ole olemassa.

Mielestäni kaikki kolme kenttäkoetta tukevat väitettä, että projekti voi onnistua, kun toimittaja osaa ennustaa työmäärät riittävän tarkasti ja perustella lisätyöt muutoksenhallinnan avulla. Projekti C:stä nähdään, mitä tapahtuu, kun projekti sisältää asiakaskohtaisia laajennuksia, joiden toteutustapaa ja sisältöä ei tiedetä etukäteen.

Tutkimukseni eteni suunnitellusti, mutta prosessimalleja vertaillessani huomasin, että käytetty viitekehys on liian suppea. Laajoja prosessimalleja pystyisi vertailemaan paremmin laajentamalla viitekehyksen käsitteistöä. Olisi mielenkiintoista tehdä jatkotutkimus, jossa ensin luotaisiin tarkempi viitekehys. Täytyy kuitenkin muistaa, että viitekehyksestä ei saa tehdä liian monimutkaista, vaan jokaisen vertailtavan mallin tulee mahtua yhteen kuvaan.

Tälle laajemmalle viitekehykselle voidaan luodaan onnistumismittaristo, jota voidaan hyödyntää kenttäkokeissa. Kenttäkokeista kerätään aineistoa luodun mittariston mukaan ja lopulta se analysoidaan. Prosessimallien vertailun ja kenttäkokeiden analysoinnin perusteella jatkotutkimuksen tavoitteena olisi luoda uusi ”ideaalimalli”, joka olisi kehitetty nimenomaan ohjelmistoprojektin onnistumisen näkökulmasta.

Viiteluettelo

[Bailey & Pearson, 1983] James E. Bailey & Sammy W. Pearson. Development of a tool for measuring and analyzing computer user satisfaction. *Management Science*, 29(5):530–545, 1983.

[Beck, 1999] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[Boehm, 1988] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[Colquitt & Zapata-Phelan, 2007] Jason A. Colquitt & Cindy P. Zapata-Phelan. Trends in theory building and theory testing: A five-decade study of the Academy of Management Journal. *Academy of Management Journal*, 50(6):1281–1303, 2007.

[Curtis *et al.*, 1987] W. Curtis, H. Krasner, V. Shen, & N. Iscoe. On building software process models under the lamppost. In *Proceedings of the 9th International Conference on Software Engineering*, pages 96–103. IEEE Computer Society Press, 1987.

[Davis & Bersoff, 1991] Alan M. Davis & Edward H. Bersoff. Impacts of life cycle models on software configuration management. *Commun. ACM*, 34(8):104–118, 1991.

[Delone & McLean, 1992] William H. Delone & Ephraim R. McLean. Information systems success: The quest for the dependent variable. In *Proceedings of the 9th International Conference on Software Engineering*, pages 59–95. Georgia State University, 1992.

[Delone & McLean, 2003] William H. Delone & Ephraim R. McLean. The Delone and McLean model of information systems success: Ten-year update. *Journal of Management Information Systems*, 19(4):9–30, 2003.

[Grover *et al.*, 1996] Varun Grover, Seung R. Jeong, & Albert H. Segars. Information systems effectiveness: The construct space and patterns of application. *Information & Management*, 31(4):177–191, 1996.

[Hanna, 1995] Mary Hanna. Farewell to waterfalls? *Softw. Mag.*, 15(5):38–46, 1995.

[Humphrey, 1989] Watts S. Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.

[ISO, 1998] ISO. *15504-5 Standardi: Software Process Improvement and Capability dEtermination*, 1998. URL: `http://www.sqi.gu.edu.au/spice/` [30.3.2008].

[Järvinen & Järvinen, 2004] Pertti Järvinen & Annikki Järvinen. *Tutkimustyön metodeista*. Opinpajan Kirja, Tampere, 2004.

[Karvinen *et al.*, 1994] M. Karvinen, T. Reponen, & R. Vehviläinen. *Tietotekniikkainvestoinnit: valmistelijan ja johdon opas*. Gummerus, 1994.

[Lallé, 2003] Béatrice Lallé. The management science researcher between theory and practice. *Organization Studies*, 24(7):1097–1114, 2003.

[Lawson, 2004] Lawson. Projekti c:n toimitussopimus ja projektisuunnitelma, Lokakuu 2004. Sisäinen raportti, 22.10.2004.

[Lawson, 2007a] Lawson. Projekti a:n raportointiaineisto, Toukokuu 2007. Sisäinen raportti, 2.5.2007.

[Lawson, 2007b] Lawson. Projekti b:n projektisuunnitelma, Elokuu 2007. Sisäinen raportti, 21.8.2007.

[Lawson, 2008a] Lawson. Implex methodology, March 2008. Internal report, 30.3.2008.

[Lawson, 2008b] Lawson. Projekti b:n raportointiaineisto, Huhtikuu 2008. Sisäinen raportti, 3.4.2008.

[Lawson, 2008c] Lawson. Projekti c:n raportointiaineisto, Maaliskuu 2008. Sisäinen raportti, 26.3.2008.

[Lawson, 2008d] Lawson. Stepwise methodology, March 2008. Internal report, 30.3.2008.

[Ludewig, 2004] Jochen Ludewig. Advanced software project management, June 2004. Laudatur -course at University of Helsinki. URL: `http://www.iste.uni-stuttgart.de/se/people/ludewig.html`.

[Luqi *et al.*, 1998] Luqi, Carl K. Chang, & Hong Chu. Specifications in software prototyping. *Systems and Software*, 42(2):125–140, August 1998.

[Nevalainen, 1999] Risto Nevalainen. Kokemuksia ohjelmistoprosessien arvioinnista SPICE:n avulla. *Systeemityö*, 1:2–8, 1999.

[Papazoglou & van den Heuvel, 2007] Mike P. Papazoglou & Willem-Jan van den Heuvel. Service oriented architectures: Approaches, technologies, reasearch issues. *The VLDB Journal*, 16(3), 2007.

[Pressman, 2005] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2005.

[Reel, 1999] J.S. Reel. Critical success factors in software projects. *IEEE Software*, 16(3):18–23, 1999.

[Schwaber & Beedle, 2002] Ken Schwaber & Mike Beedle. *Agile software development with Scrum*. Prentice Hall, 2002.

[www.controlchaos.com, 2008] www.controlchaos.com. What is Scrum, March 2008. URL: `http://www.controlchaos.com/about` [29.3.2008].

[www.lawson.com, 2008] www.lawson.com. Opportunity Analyzer ™, March 2008. URL: `http://www.lawson.com/wcw.nsf/pub/OA_FD357C` [30.3.2008].