Ari-Antti Tervonen

Ohjelmistokehitysmenetelmän valinta tuotekehitysprojektissa

Kehittämistehtävä

Kevät 2017

05/2017

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä(t): Tervonen Ari-Antti

Työn nimi: Ohjelmistokehitysmenetelmän valinta tuotekehitysprojektissa

Tutkintonimike: Teknologiaosaamisen johtaminen

Asiasanat: Ohjelmointikehitysmenetelmät, pienmerkintä, tuotekehitys

Työn ensisijaisena tarkoituksena oli perehtyä erilaisiin ohjelmistokehitysmenetelmiin, joita voitaisiin käyttää kohdeprojektin kaltaisissa tuotekehitysprojekteissa. Opinnäytetyön taustaksi kartoitetaan kohdeprojektin asiakasvaatimuksia, ja pohditaan niiden erilaisia ratkaisuvaihtoehtoja asiakastarpeiden tyydyttämiseksi. Näin saadaan selville, millaisen pienmerkintälaitteiston asiakas haluaa, ja mitä tuotekehityksen tulisi ottaa huomioon laitteistoa rakentaessaan.

Työssä tutustuttiin yleisimpiin ohjelmistokehitysmenetelmiin ja projektinhallintamenetelmiin, sekä kohdeprojektin lähtökohtiin ja vaatimuksiin. Havaittiin, että kohdeprojektiin perehtyminen on ensisijaisen tärkeää, jotta valinta ohjelmistokehitysmenetelmälle voitaisiin tehdä. Kohdeprojekti on hyvä esimerkki kohdeyritykselle, sillä monet yrityksen tuotekehitysprojekteista ovat monilta avainalueilta samankaltaisia kohdeprojektin kanssa.

McConnell (2002) esitteli teoksessaan 11 pohdittavaa kysymystä, joihin kohdeprojektin näkökulmasta tulisi pyrkiä vastaamaan. Hän tarjosi perinteisiin ohjelmistokehitysmenetelmiin taulukon, josta voitiin testin jälkeen löytää kohdeprojektiin sopivin ohjelmistokehitysmenetelmä.

Testiä räätälöitiin työssä omiin tarpeisiin sopivammaksi, niin että jokaiselle kysymykselle annettiin prioriteettiarvo arvioidun tärkeyden mukaan. Tämän arvioitiin parantavan testin luotettavuutta, sillä jotkin kysymykset eivät olleet niin tärkeitä kohdeprojektin näkökulmasta kuin toiset. Tämän lisäksi testiin lisättiin kaksi ketterän kehityksen menetelmää, scrum ja eXtreme Programming (XP), joita alkuperäinen testi ei kattanut.

Testin avulla määritettiin kolme parasta menetelmää loppuanalyysiä varten, joka suoritettiin SWOT-menetelmällä. Soveliain ohjelmistokehitysmenetelmä kohdeprojektin kaltaisiin projekteihin oli spiraalimalli. Vähäriskisempiin ja suuremman tuotekehitystiimin omaaviin projekteihin suositeltiin scrum-menetelmän käyttöä, mahdollisesti hyödyntäen XP-menetelmän ohjelmiston laatua parantavia tekniikoita. XP-menetelmän tekniikoita tulisi testata pienessä mittakaavassa, ja tehdä jatkopäätöksiä tuotekehitystiimiltä saadun palautteen avulla.

ABSTRACT

Author(s): Tervonen Ari-Antti

Title of the Publication: Choosing Software Development Method in R&D Project

Degree Title: Master of Engineering, Technology Competence Management

Keywords: Software development methods, Road Marking, Product Development

Major goal for this thesis was to get familiar with various software development methods which could be used in projects similar than example project. For background of this thesis we gathered customer requirements, and ponder different kind of solutions to fulfill customers' needs. This way we could figure out, what kind of road marking device customer really wants, and what needs to be considered when making one.

In this thesis, we searched for most commonly used software development methods and project management methods, and example projects' requirements and basis. In this thesis we learned that we need a lot of information about project when choosing software development method for it. Selected target project is good choice for example, because it is very similar with many other complex projects in our company.

McConnell (2002) introduced 11 questions to ponder when choosing a software development method. After answering these 11 questions he offered a chart for traditional software development methods, which can be used to select optimal method for project.

Test was tailored to fulfill our needs better, we did add a priority value for every question pondered. That is because other questions seemed to be much more important than others, for our projects' perspective. With this priority value, we can prevent that from skewing the results. We also added two agile development methods (eXtreme Programming and scrum) to this test.

With this test we chose 3 best software development methods for final analysis which was executed with SWOT-method. Best software development for risky R&D project would be spiral model. In larger and less risky projects, we would suggest scrum-method to be used rather than spiral model. When using scrum-method it is suggested to add some techniques used in XP-method in order to get a better quality in the code. Selected methods should be added slowly and react to feedback by software development team, because many of them are abnormal and hard to internalize.

SISÄLLYS

SYMBOLILUETTELO

CAN-väylä

Controller Area Network, ajoneuvoissa yleisesti käytetty tiedonsiirto menetelmä

Esimerkintä

Ennen varsinaisten tiemerkintöjen tekemistä tulevat merkinnät esimerkitään. Esimerkinnällä kerrotaan tiemerkintöjä tekevälle kuljettajalle haluttujen merkintöjen paikka ja tyyppi.

Pienmerkinnät

Pienmerkinnät ovat teiden pintoihin tehtyjä kuvioita, nuolia, nopeusrajoitusmerkkejä sekä muita teillä käytettyjä ohjeistusmerkintöjä. Merkintöjen materiaalina toimii kuumamassa.

RMD Tieto-Oskari Oy:n tiemerkintäjärjestelmän nimi, joka tulee sanoista Road Marking Device

Tiemerkinnät

Tiemerkinnöiksi voidaan laskea kaikki tiehen tehtävät merkinnät. Tässä työssä termiä käytetään kuvaamaan tien pituussuuntaisia viivatyyppisiä merkintöjä: kaista-, reuna- sekä keskiviivoja.

TMA

Truck Mounted Attenuator. Ajoneuvon peräkoukkuun kytkettävä törmäysvaimennin.

XP

eXtreme Programming, ketterä ohjelmistokehitysmenetelmä.

Worst case -skenaario

Pahin mahdollinen realistinen tilanne.

1 JOHDANTO

Suomen tiemerkinnät poikkeavat muista Euroopan maista. Suomessa käytetään ainoana pohjoismaana värijärjestelmää, joka sisältää keltaisia ja valkoisia merkintöjä. Suomessa tiemerkinnät kuluvat talvikunnossapidon vuoksi myös keskimääräistä enemmän.

Tiemerkinnät jaetaan Suomessa kahteen osaan: Tiemerkintöihin ja pienmerkintöihin. Tiemerkinnät ovat tien suuntaisia viivatyyppisiä merkintöjä, kaistaviivoja, reunaviivoja sekä keskiviivoja. Pienmerkinnät ovat kuviomuotoisia opasteita maanteiden pinnassa, kuten kääntymisnuolet, varoitusmerkit, suojatiet yms.

Tiemerkinnöistä vastaa Suomessa Tiehallinto, ja tiemerkintöjä kunnostetaan aina kesäisin. Keväisin Tiehallinto kilpailuttaa teiden kunnossapidon urakkatyyppisesti tulevalle kesälle noin viiden Suomessa toimivan alan yrityksen kesken.

Jo vuodesta 1997 Tieto-Oskari Oy on ollut ainoa suomalainen yritys, joka toimittaa ohjausjärjestelmiä tiemerkintäajoneuvoihin. Koska yritys ei itse valmista tiemerkintäajoneuvojen mekaniikkaa, urakoitsijat ovat usein ostaneet tiemerkintäajoneuvoja kilpailevilta tuottajilta ilman ohjausjärjestelmiä, minkä jälkeen Tieto-Oskari Oy on asentanut ohjausjärjestelmänsä laitteeseen.

Yritys on saanut runsaasti markkinaosuutta Suomen markkinoilla erilaisissa tiemerkintöjen ohjausjärjestelmissä. Suurin osa laitteista on suunniteltu tekemään maanteiden pintoihin kaista-, reuna-, ja keskiviivoja. Tämän lisäksi yrityksen luoma RMD-tuoteperhe sisältää esimerkintäajoneuvoihin sekä jyrsimiin suunniteltuja ohjausjärjestelmiä.

Yrityksen suunnitelmissa on laajentaa RMD-tuoteperhettä pienmerkintöjä tekevällä laitteistolla kilpailukyvyn ylläpitämisen vuoksi. Tämän työn tarkoituksena on selvittää, minkälaisia ohjelmistokehitysmenetelmiä kohdeyrityksen tulisi käyttää kohdeprojektin kaltaisissa hankkeissa, vai kannattaako käytäntöjä muuttaa lainkaan.

Tutkimuskysymys ei kohdistu suoraan kohdeprojektiin, sillä aikataulullisesti on mahdotonta ottaa täysin uudenlainen ohjelmistokehitysmenetelmä suoraan käyttöön, vaan se tulee tehdä asteittain. Kohdeprojektin kaltaisia tutkivia tuotekehitysprojekteja tulee yrityksen tuotekehitykseen runsaasti, joten kohdeprojektin vaatimuksia voidaan käyttää hyvänä esimerkkinä kehitysmenetelmän valinnassa yrityksen muihin samankaltaisiin projekteihin.

Opinnäytetyön taustaksi kartoitetaan kohdeprojektin asiakasvaatimuksia ja pohditaan niiden erilaisia ratkaisuvaihtoehtoja asiakastarpeiden tyydyttämiseksi. Näin pyritään selvittämään, minkälaisen pienmerkintälaitteiston asiakas haluaa, ja mihin asioihin tuotekehityksen tulisi kohdentaa erityistä huomiota laitteistoa rakentaessaan.

Kohdeprojektissa on kolme osapuolta: laitteiston ensimmäinen asiakas, joka on tilannut tuotteen, yrityskumppani, joka valmistaa laitteistoon mekaniikan sekä me, jotka vastaamme elektroniikan ja ohjelmiston toimivuudesta.

Yrityksen tuotekehitysprojektit ovat usein monilta osin samankaltaisia. Kehitystiimin koostumus, projektin koko, aikataulu sekä uutta luova projektintyyppi ovat samankaltaisia monissa yrityksen projekteissa. Tällöin kyseessä on tutkiva tuotekehitys, jossa tehdään täysin uudentyyppinen laitteisto. Kun aiheesta ei ole aikaisempaa kokemusta, asiakasvaatimusten täydellinen kartoittaminen on vaikeaa.

Ohjelmistokehitysmenetelmiä on kehitetty lukuisia, ja jokainen menetelmä soveltuu tietyn tyyppisiin kehitysprojekteihin käytettäväksi. Kehitysmenetelmän käyttäminen epäsopivassa projektissa pienentää projektin onnistumisen todennäköisyyttä merkittävästi. Parhaimmillaan sopiva ohjelmistokehitysmenetelmä lyhentää tuotekehitykseen kuluvaa aikaa, parantaa laatua ja edistymisen näkyvyyttä, sekä auttaa projektia pysymään aikataulussa. Tällöin myös projektin johtaminen helpottuu.

Lukuisten hyvien puolien vuoksi on syytä panostaa optimaalisen ohjelmistokehitysmenetelmän löytämiseen kohdeprojektin kaltaisissa haastavissa hankkeissa. Tehtävä ei ole helppo, sillä todennäköisesti optimaalisin menetelmä olisi useiden eri menetelmien yhdistelmä, joka olisi räätälöity juuri kohdeyrityksen toimintatapoihin ja projekteihin.

Tässä työssä tutustutaan erilaisiin ohjelmistokehitysmenetelmiin sekä kohdeprojektin taustoihin ja asiakastarpeisiin. Kohdeprojektista pyritään löytämään avainkohdat, joiden perusteella voidaan tehdä valinta sopivalle tuotekehitysmenetelmälle.

Kuten kohdeprojektissakin, ohjelmistoprojekteilla on lähes aina haastava aikataulu. Tämän vuoksi menetelmän nopeudelle ja tehokkuudelle voidaan antaa paljon arvoa. Varsinainen menetelmien omaksuminen ja perehtyminen tulee tehdä asteittain, niin että ajanhetkellä käynnissä olevat projektit eivät häiriinny liikaa.

Tässä työssä valitaan menetelmä kohdeprojektin näkökulmasta. Esimerkkiprojektiin syvällisemmin perehtyminen tulisi auttaa havaitsemaan projektin avainkohtia, joihin sopivan ohjelmistokehitysmenetelmän valinta perustuu. Tämän vuoksi työssä ei tutustuta kohdeyrityksen tyypillisimpiin tutkiviin tuotekehitysprojekteihin, koska näin laaja otanta aiheuttaisi pinnallisemman tuloksen, eivätkä projektien avainkohdat välttämättä tulisi tällöin esiin.

Tieto-Oskari Oy:n ohjelmistokehityksessä ei nykyisin käytetä standardoituja ohjelmistokehitysmenetelmiä, vaan menetelmät ovat muovautuneet yrityksen tarpeisiin sopiviksi lukuisten kehitysprojektien aikana. Ohjelmistokehitysprojektit ovat yrityksessä onnistuneet erittäin suurella todennäköisyydellä. Kuitenkin toiminnan laajentuessa projektien kootkin kasvavat, tällöin myös tuotekehitystiiminkin koko kasvaa. Laajempien kokonaisuuksien hallitseminen kehitysmenetelmien avulla on helpompaa.

Standardoitujen ohjelmistokehitysmenetelmien avulla voitaisiin saavuttaa yhteisiä tapoja kehittää ohjelmistoja. Tämä auttaa tilanteissa, joissa henkilöstö vaihtuu tiimin sisällä. Yhteisten toimintatapojen ansiosta uusi henkilö perehtyy nopeammin, sillä toimintatavat ovat tuttuja.

Tässä työssä otettiin vertailuun mukaan useita ohjelmistokehitysmenetelmiä. Osa käsitellyistä menetelmistä ei ota kantaa siihen, kuinka itse ohjelmointityö tehdään, vaan keskittyvät projektinhallintaan. Tällaisten menetelmien kanssa voidaan aina käyttää lisämenetelmiä varsinaisen ohjelmiston laadun ja tehokkuuden parantamiseen. Tämän vuoksi kaikkia menetelmiä kohdellaan arvioinnissa samanlaisina.

2 OHJELMISTOKEHITYSPROJEKTIT NYKYISIN

2.1 Ohjelmistokehitysprojektit maailmalla

Ohjelmiston kehitysprojektit ovat suorastaan sekasortoisessa tilassa onnistumisprosenttien mukaan. Tutkimuksen mukaan jopa 31,1 % alan projekteista keskeytetään ennen projektin valmistumista. Tulokset myös osoittavat, että 52,7 % projekteista tulee maksamaan 189 % alkuperäisestä kustannusarviosta. (The Standish Group, 2014, s.3)

Ohjelmistoprojektien epäonnistuminen voi tulla yritykselle hyvin kalliiksi. Esimerkiksi Denverin lentokentällä epäonnistuttiin laukkujen käsittelyyn tarkoitetussa ohjelmistossa, ja tämä maksoi arvioiden mukaan kaupungille $1,1 miljoonaa joka päivä. Kokonaisarviot maailman epäonnistuneista ohjelmistoprojekteista ovat vuosittain triljoonia dollareita. Tarkkoja laskelmia ei voida kuitenkaan tehdä, sillä seurauksien kustannuksia on vaikea mitata. (The Standish Group, 2014, s.3)

Vuonna 1988 Westpac Banking Corporation halusi uudistaa tietojärjestelmänsä, he laativat 5 vuoden projektisuunnitelman 85 miljoonan dollarin budjetilla. 3 vuotta myöhemmin ja 150 miljoonaa dollaria kuluttaneena yritys keskeytti projektin epäonnistuneena ja irtisanoi 500 kehittäjää. (McConnell, 2002, s.82) Ohjelmistoprojekteista löytyy monia huonoja esimerkkejä. Edes parhaat ohjelmistoalan yritykset eivät näytä välttyvän ajoittaisilta suurilta epäonnistumisilta. IBM, Microsoft sekä Apple ovat kukin epäonnistuneet isoissa projekteissa vähintään kerran.

Esimerkiksi FBI aloitti vuonna 2000 ohjelmistoprojektin prosessoimaan automaattisesti organisaation lomakkeita ja hoidettavia tapauksia (Virtual Case File). Kongressi myönsi projektille 379,8 miljoonan dollarin rahoituksen ja 22 kuukautta aikaa projektille. Kaksi vuotta myöhemmin FBI anoi 123,2 miljoonaa dollaria lisärahoitusta projektille. Vuonna 2005 ohjelmiston toimittaja (Science Applications International Corp) toimitti 700 000 riviä niin virheellistä koodia, että FBI päätti lopettaa koko projektin. Ohjelmistotoimittaja syytti projektia liian kunnianhimoisesta aikataulusta, sekä mm. huonosti määritettyjä vaatimuksia. (Kerzner, 2014, s. 221)

Keskimäärin laskettuna 16,2 % projekteista on valmistuessaan alkuperäisessä aikataulussaan ja budjetissaan. Projekteissa epäonnistutaan myös vaatimusten toteuttamisessa. Amerikan suurimpien yritysten ohjelmistoprojekteissa (jotka ovat valmistuneet) on arviolta vain 42 % alkuperäisistä ominaisuuksista, joita ohjelmiston oli tarkoitus hallita. Pienillä yrityksillä menee hiukan paremmin - 78,4 % valmiista ohjelmistoista sisältää 74,2 % alkuperäisistä ominaisuuksista. (The Standish Group, 2014, s.3)

The Standish Group julkaisee vuosittain tilastotietoja maailman IT-projektien onnistumisesta. Raporttia kutsutaan nimellä Chaos Report, ja siinä jaetaan ohjelmistoprojektit kolmeen eri luokkaan, ”*Successful*”, ”*Challenged*”, ”*Failed*”. Raportti määrittelee onnistuneen projektin (*Successful*) pysyvän budjetissaan ja aikataulussaan. Projektilla tulee olla myös kaikki vaatimusmäärittelyyn kirjatut ominaisuudet. (The Standish Group, 2014, s.4)

Projektia pidetään osittain onnistuneena (*Challenged*), mikäli projekti valmistuu, mutta kustannukset ovat menneet yli budjetin tai suunniteltu aikataulu on ylittynyt. Lisäksi projektilla on oltava vähemmän ominaisuuksia ja toimintoja kuin alun perin on ollut tarkoitus, jotta projekti luokitellaan tähän ryhmään. Kolmas ryhmä koostuu projekteista, jotka keskeytettiin jossain projektin vaiheessa (*Failed*). (The Standish Group, 2014, s.4)

The Standish Group:n ”Chaos Report” -julkaisussa kysyttiin onnistuneiden projektien johtajilta, miksi heidän mielestään projekti onnistui. Raportissa julkaistiin kymmenen yleisintä syytä ohjelmistoprojektien onnistumiseen (Taulukko 1). Tärkein syy nähtiin olevan asiakkaan osallistuminen projektiin. Toiseksi tärkein onnistumisen edellytys oli toiminnallisen johdon tuki projektille. Selkeät vaatimukset projektille oli kolmanneksi tärkein onnistumisen edellytys.

Taulukko 1. Onnistumisen edellytykset. (The Standish Group, 2014, s.8)

Projektin onnistumisen tekijät	% vastauksista
1. Käyttäjän osallistuminen	15,9%
2. Toiminnallisen johdon tuki	13,9%
3. Selkeät vaatimukset	13,0%
4. Hyvä suunnittelu	9,6%

5. Realistiset odotukset	8,2%
6. Pienet etappipisteet projektissa	7,7%
7. Kilpailuhenkinen henkilöstö	7,2%
8. Projektin omistaminen	5,3%
9. Selkeät visiot ja tavoitteet	2,9%
10. Kovasti työskentelevä ja keskittynyt henkilöstö	2,4%
Muut	13,9%

The Standish Group:n ”Chaos Report” -julkaisussa on esitelty yleisimmät syyt siihen, ettei projektissa onnistuttu suunnitellusti (*Challenged*) (taulukko 2). Yleisimpänä syynä pidettiin sitä, ettei asiakas ollut riittävästi mukana projektissa. Seuraavaksi yleisimmät syyt olivat puutteelliset vaatimukset ja määritykset, sekä niiden muuttaminen projektin aikana.

Taulukko 2. Osittain epäonnistuneet projektit (*Challenged*). (The Standish Group, 2014, s.9)

Projektin osittaisen epäonnistumisen tekijät	% vastauksista
1. Käyttäjän osallistumisen puute	12,8%
2. Vaillinaiset vaatimukset ja määrittelyt	12,3%
3. Vaatimusten ja määrittelyjen muuttaminen	11,8%
4. Toiminnallisen johdon tuki	7,5%
5. Teknologian kykenemättömyys	7,0%
6. Resurssien puute	6,4%
7. Epärealistiset odotukset	5,9%
8. Epäselvät tavoitteet	5,3%
9. Epärealistiset aikataulut	4,3%
10. Uusi teknologia	3,7%
Muut	23,0%

Taulukossa 3 on esitetty raportin peruutettujen ohjelmistoprojektien yleisimpiä syitä (*Failed*). Yleisin syy projektin peruuntumiselle oli puutteelliset vaatimukset. Toiseksi yleisin syy oli loppukäyttäjän läsnäolon puute projektin aikana. Kolmanneksi yleisimpänä syynä pidettiin resurssien puutetta.

Taulukko 3. Lopetettujen projektien syyt (Failed). (The Standish Group, 2014, s.9)

Syyt projektien lopettamiseen	% vastauksista
1. Vaillinaiset vaatimukset	13,1%
2. Käyttäjän osallistumisen puute	12,4%
3. Resurssien puute	10,6%
4. Epärealistiset odotukset	9,9%
5. Toiminnallisen johdon puute	9,3%
6. Vaatimusten ja määritysten muuttaminen	8,7%
7. Vaillinaiset suunnitelmat	8,1%
8. Projektille ei ollut enää tarvetta	7,5%
9. IT-osaston johdon puute	6,2%
10. Teknologian osaamattomuus	4,3%
Muut	9,9%

The Standish Group:n julkaisemassa raportissa havaitaan myös, että projektien johtajien mielestä projektit epäonnistuvat nykyisin useammin kuin ennen (taulukko 4). Kyselyyn vastanneista johtajista 48 % uskoi, että nykyään projektit epäonnistuvat useammin kuin viisi vuotta sitten. Vastaajista 41 % puolestaan uskoi, että nykyään projektit epäonnistuvat useammin kuin viisi vuotta sitten. Ero ei ole kuitenkaan kovin suuri. (The Standish Group, 2014)

Taulukko 4. Projektien epäonnistuminen. (The Standish Group, 2014, s.9)

	Kuin 5 vuotta sitten	Kuin 10 vuotta sitten
Merkittävästi enemmän epäonnistumisia	27%	17%
Jonkin verran enemmän epäonnistumisia	21%	29%
Ei muutosta	11%	23%
Jonkin verran vähemmän epäonnistumisia	19%	23%
Huomattavasti vähemmän epäonnistumia	22%	8%

Raportti antaa ymmärtää, että ohjelmistoprojekteiden hallinnassa olisi paljon opittavaa. Kenties projektit ovat monimutkaistuneet tekniikan kehittymisen myötä niin paljon, että kokonaisuuksien hallitseminen tuottaa useammin ongelmia.

On vaikea arvioida, onko ilmiössä syytä myös siinä, että joissain projekteissa yritys voi tahallaan tehdä ylioptimaalisen aikataulutuksen voittaakseen kilpailutuksen. Raportti antoi kuitenkin ymmärtää, että projektin luokitteleminen ”*Challenged*”-ryhmään vaati muutakin kuin pelkän aikataulun pettämisen. Myös asiakastyytyväisyys ja vaatimusten täyttyminen täytyi olla vaillinaista.

Kritiikkiä Standish Group saa myös siitä, ettei se ole julkaissut projektien riskipitoisuuksia, eikä asiakkaiden tyytyväisyysprosentteja, jotka voivat hyvinkin vaikuttaa tuloksiin (Opelt & Gloger & Pfarl, 2013, s.4). Alan kirjallisuuden näkemys on kuitenkin hyvin yhtenäinen: Ohjelmistoprojektien läpiviemisessä olisi yrityksillä paljon parantamisen varaa. (Beck, 2005; Haikala & Mikkonen, 2011; Koch, 2004; Haikala & Märijärvi, 2004)

Savolainen (2011) analysoi väitöskirjassaan viittä epäonnistunutta ohjelmistokehitysprojektia. Hän pyrki työssään löytämään nimenomaan syitä ohjelmistoprojektien epäonnistumiseen. (Savolainen, 2011, s. 18)

Savolaisen tutkimuksissa nousi esiin viisi syytä epäonnistumiselle:

- Asiakastarpeiden väärinymmärtäminen.
- Kokeneiden kehittäjien puute kehitystiimissä.
- Riskit yhdistettynä tiukkoihin aikatauluihin.
- Ongelmat valitussa arkkitehtuurissa.
- Ei kyetty ymmärtämään teknisiä ongelmia. (Savolainen, 2011, s. 55)

Savolainen (2011) huomasi tutkimuksessaan, että ohjelmistoprojektien johtajat eivät useinkaan kyenneet ennen projektia määrittelemään, millainen on onnistunut projekti. (Savolainen, 2011, s. 55)

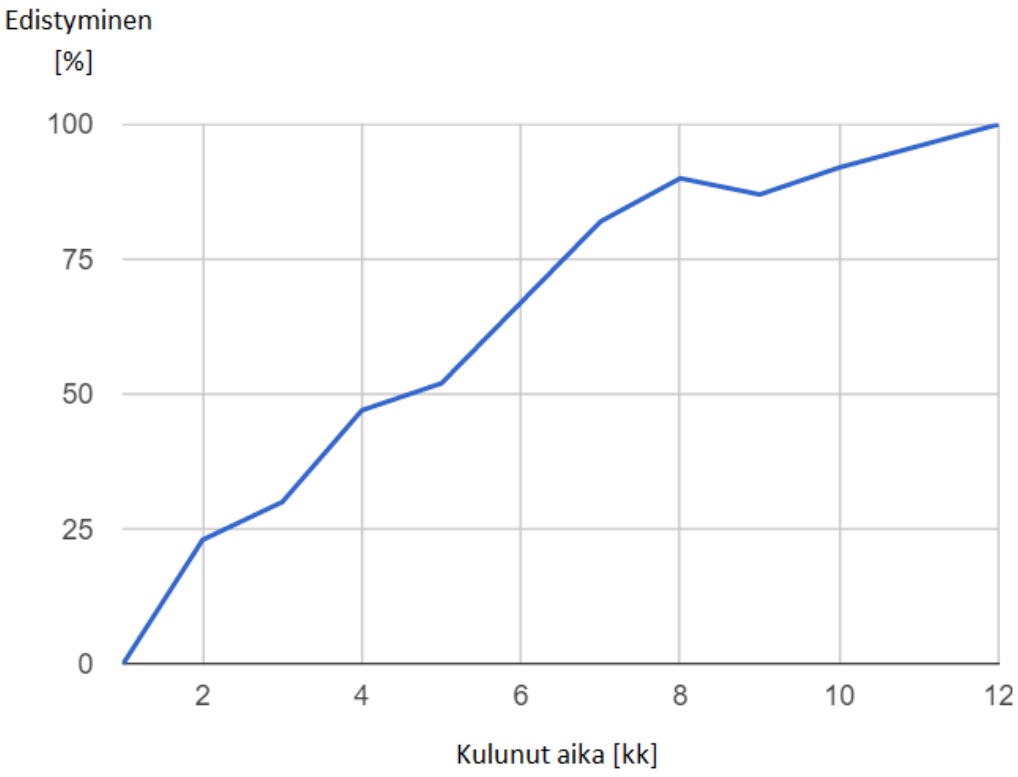
Ohjelmiston toimittaja toimii usein paineen alla ja hänellä on kiire aloittaa projekti mahdollisimman nopeasti projektin aloitusvaiheessa. Tutkimus siitä, millaiseen projektiin ollaan ryhtymässä jää usein vaillinaiseksi, ja tämä vaihe on erittäin suotuisa virheille. (Savolainen, 2011, s. 55)

McConnell (2002) huomauttaa, että monien ohjelmistokehitysprojektien epäonnistumisen taustalla on ohjelmistojen ymmärtämättömyys. Ohjelmistot ovat usein projektien johtajille ja myyntihenkilöstölle abstrakteja asioita ja mahdottomia hahmottaa. Kuitenkin myyntihenkilöstö ja yrityksen johto ovat yleensä ne, ketkä luovat aikataulun ja vaatimukset projektille. (McConnell, 2002)

Myyntihenkilöstö myy asioita työkseen, he ovat hyviä siinä. On yleistä, että projektin edetessä ohjelmiston toimittaja saa asiakkaalta uusia toivomuksia, ja on ohjelmistotoimittajan tehtävä ”myydä” uudet ajatukset tuotekehitykselle. Usein myyntihenkilöt onnistuvat tässä, sillä usein tuotekehityksenkin on vaikea perustellusti kyetä osoittamaan tavoitteet liian haastaviksi, vaikka epäilyksiä olisikin. (McConnell, 2002)

Asiakkaan toivomukset ovat tietysti ensisijaisen tärkeitä. Kuitenkin tässä yhtälössä piilee vaara. Yrityksen johdolle ja myyntihenkilöstölle voivat uudet muutokset kuulostaa helpoilta toteuttaa, ja ne myös asiakkaille luvataan. Joskus ne ovat helppoja toteuttaa, joskus eivät. Tätä on mahdoton hahmottaa, ellei ole ohjelmoinnin ammattilainen, ja perehtynyt nimenomaan kyseessä olevaan ohjelmistoon. (McConnell, 2002)

Tomayko & Hazzan (2004) puolestaan arvioivat, että projektit epäonnistuvat sen vuoksi, että tehty dokumentointi piilottaa todellisen tiedon projektin edistymisestä. Ohjelmistoprojektin edistyminen ei ole lineaarinen, vaan valmistuminen hidastuu, kun projekti alkaa lähestyä loppuaan (kuva 1). Sanotaan, että projekti on puolet ajastaan yli 90 % valmis. (Tomayko & Hazzan, 2004, s. 9)



Kuva 1. Tyypillinen ohjelmistoprojektin edistyminen. (Tomayko & Hazzan, 2004, s. 10)

Tomayko & Hazzan (2004) uskovat, että monet aikataulujen ylitykset johtuvat tästä ilmiöstä. Ilmiötä ei oteta huomioon riittävästi ohjelmistoprojektien aikatauluja luotaessa. (Tomayko & Hazzan, 2004, s. 10) McConnell (2002) uskoo, että ilmiö johtuu siitä, että mitä lähempänä projekti on valmistumistaan, sitä enemmän tuotekehitys joutuu tekemään muita asioita kuin varsinaista ohjelmointia. Täten saman tehtävän suorittaminen projektin alussa on huomattavasti nopeampaa kuin projektin loppuvaiheessa. (McConnell, 2002)

Kerzner (2014) listaa ohjelmistokehitysprojektien epäonnistumisten yleisimpiä syitä teoksessaan. Taulukkoon 5 on kerätty osa teoksessa mainituista syistä. Syyt kohdentuvat ensisijaisesti suuriin ohjelmistoprojekteihin. (Kerzner, 2014, s. 224)

Kerzner (2014) mainitsee, että järjestelmän vaatimuksia ei koskaan pystytä täysin määrittelemään etukäteen, sillä käyttäjät eivät pysty ennustamaan niitä. Tämän vuoksi sekä budjetti- että aikatauluarviot perustuvat aina epätäydelliseen informaatioon. (Kerzner, 2014, s. 225)

Taulukko 5. Epäonnistumisen syitä ohjelmistoprojekteissa. (Kerzner, 2014, s. 224)

Ohjelmistoprojektin epäonnistumisen yleisimmät syyt
Epäonnistunut liiketoimintamallin valmistelu
Ei yhteisymmärrystä tavoitteissa
Hallinnon epäonnistuminen
Ei selkeää määritelmää onnistumiselle
Toiveikas ajattelu, epärealistiset tavoitteet
Ei haluta kuulla huonoja uutisia
Ei opita edellisistä projekteista
Ei etsitä erilaisia toimintavaihtoehtoja
Heikko testaus
Ei elpymissuunnitelmaa epäonnistumisten varalle
Huono myynnin johtaminen
Ei riskienhallintaa
Organisaation muutosten johtamisen puutteet
Ei uskota, että vaatimukset eivät voi olla etukäteen määritettyjä
Heikot mittarit
Heikko sidosryhmien sitoutuminen

Taulukkoon 5 on kerätty syitä projektien epäonnistumiseen. Kerzner ei maininnut, että taulukon syyt olisivat yleisyysjärjestyksessä.

2.2 Nykytilanteeseen sopeutuminen

Jotta ohjelmistokehitysprojekteja voidaan suorittaa onnistuneesti, täytyy pyrkiä ymmärtämään syitä nykyisiin ongelmiin. Ongelmiin ei voida paneutua ennen kuin ne tunnistetaan. Ohjelmistoprojektien läpivienti suunnitelman mukaisesti on vaikeaa. Alan kirjallisuuden mukaan tilannetta kuitenkin voidaan helpottaa käyttämällä ohjelmistokehitysmenetelmiä tehokkaasti. Oikein käytettynä menetelmät pakottavat projektin tiettyyn raamiin. Tällöin osa ongelmista, kuten projektin alun kiirehtiminen, poistuu, sillä hyvät toimintamenetelmät pakottavat suorittamaan mallin mukaiset suunnitelmat projektin alustusvaiheessa. Menetelmien oikeaoppinen

käyttö luo rajoitteita myös uusien ominaisuuksien lisäilylle projektin myöhemmissä vaiheissa. (McConnell, 2002; Tomayko & Hazzan, 2004; Hamlet & Maybee, 2003; Haikala & Märijärvi, 2004; Sommerville, 2001; Beck & Andres, 2005)

Ohjelmistokehitysmenetelmiä on kuitenkin erittäin paljon, ja kohdeprojektiin huonosti sopivan menetelmän käytöstä voi olla enemmän haittaa kuin hyötyä. Yleisimpiä standardoituja ohjelmistokehitysmenetelmiä esitellään seuraavassa luvussa.

3 OHJELMISTOKEHITYSMENETELMÄT

Erilaiset ohjelmistokehitysmenetelmät toimivat joiltain osin samalla tavalla, ja kaikkien perusajatus on seuraava: Jos jokin asia on liian monimutkainen hallittavaksi, hajota se pienempiin osiin ja keskity yhteen osaan kerrallaan. (Hamlet & Maybee, 2003, s. 3)

Lähempää tarkasteltuna ohjelmiston kehitysmenetelmät ovat hyvinkin erilaisia. Optimaalinen ohjelmistokehitysmenetelmä riippuu täysin kohdeprojektin luonteesta. Tämän vuoksi mikään menetelmä ei yleisesti ole toista parempi ja päätös menetelmien käyttämisestä riippuu tapauskohtaisesti kohdeprojektista. (McConnell, 2002, s. 154)

3.1 Suunnitelmaohjautuvat ohjelmistokehitysmenetelmät

Ohjelmistokehitysmenetelmät jaetaan usein kahteen eri luokkaan: suunnitelmaohjautuviin sekä ketteriin ohjelmistokehitysmenetelmiin. Perinteisissä eli suunnitelmaohjautuvissa kehitysmenetelmissä on tyypillistä, että kaikki käytännön toiminta suunnitellaan tarkasti ennen aloittamista. (Tapio, 2010, s. 42)

Kaikki ohjelmistokehityksen menetelmät ovat kehitetty tarpeesta, menetelmien avulla on nähty olevan ohjelmistoprojektin läpivientiin positiivinen vaikutus. Koska ohjelmiston tuotantoprojektit ovat aina jollain tapaa toisistaan poikkeavia, on menetelmienkin oltava toisistaan poikkeavia. Yksikään tietty menetelmä ei ole kaikissa projekteissa paras, vaan sopivuus riippuu projektin tyypistä. (Haikala & Mikkonen, 2011; Tapio, 2010; Haikala & Märijärvi, 2004)

3.1.1 Vesiputousmalli

Vesiputousmalli on saanut alkunsa jo vuonna 1956, kun Beningtonin artikkelissa kuvattiin vesiputousmallin toimintamenetelmät. Suurempaa huomiota malli sai vuonna 1970, kun Winston W. Royce julkaisi artikkelin vesiputousmallin ollessa

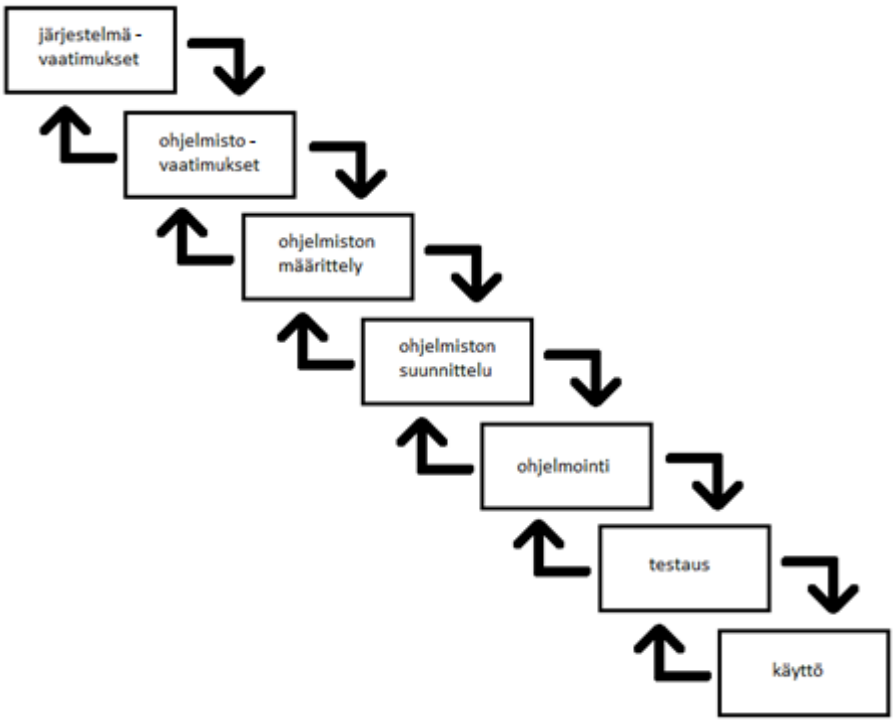
sen keskeisenä sisältönä. Mallia, ja sen erilaisia muunnelmia, käytetään runsaasti vielä tänäkin päivänä. (Haikala & Mikkonen, 2011, s. 36-37; Tapio, 2010; Steinberg & Palmer, 2004, s. 19; Sommerville, 2001, s. 45; Tomayko & Hazzan, 2004, s. 131)

Alkuperäisessä ideassa vesiputousmallissa on taaksepäin iteroinnit eri vaiheiden välissä, ja Roycen mukaan iteroinnit taaksepäin ovat tärkeitä tässä mallissa. Tapion (2010) mukaan taaksepäin palaaminen eri vaiheiden välillä käytännössä on kuitenkin harvinaista ja vaikeaa. (Tapio, 2010, s. 38)

Royce oli myös sitä mieltä, että jos vesiputousmallia käytetään uuden tyyppisessä projektissa, malli tulisi suorittaa kahdesti. Vesiputousmallin (kuva 2) katsotaan lähteneen liikkeelle Roycen artikkelista, vaikka sanaa vesiputousmalli ei koskaan artikkelissa mainittukaan. (Haikala & Mikkonen, 2011, s. 36-38; Steinberg & Palmer, 2004, s. 19)

Vesiputousmalli on saanut nimensä sen etenemistyylinsä vuoksi. Ohjelmistokehitys suoritetaan lineaarisesti vaihe vaiheelta, kunnes viimeinen vaihe päättyy. Seuraavaan vaiheeseen siirrytään vasta kun edellinen vaihe on täysin toimiva, ja täysin dokumentoitu. (Tapio, 2010, s. 45)

Uudemmat projektinhallintamenetelmät perustuvat usein jollain tapaa vesiputousmalliin. Jopa ketterä Scrum-kehitysmenetelmä voidaan nähdä sarjana lyhyitä vesiputousmalleja. (Haikala & Mikkonen, 2011, s. 37)



Kuva 2. Iteroituva vesiputousmalli. (Haikala & Mikkonen, 2011, s. 37)

Teollisuudessa malli yleistyi muunnelmana, josta taaksepäin iterointi puuttui. Tällöin jokainen vaihe on suoritettava kerralla loppuun asti. Menetelmä on hyvin yksinkertainen, ja se sopii sellaisiin projekteihin joissa kaikki vaatimukset ovat tiedossa jo projektin alkuvaiheessa. (Haikala & Mikkonen, 2011, s. 37; Sommerville, 2001, s. 46)

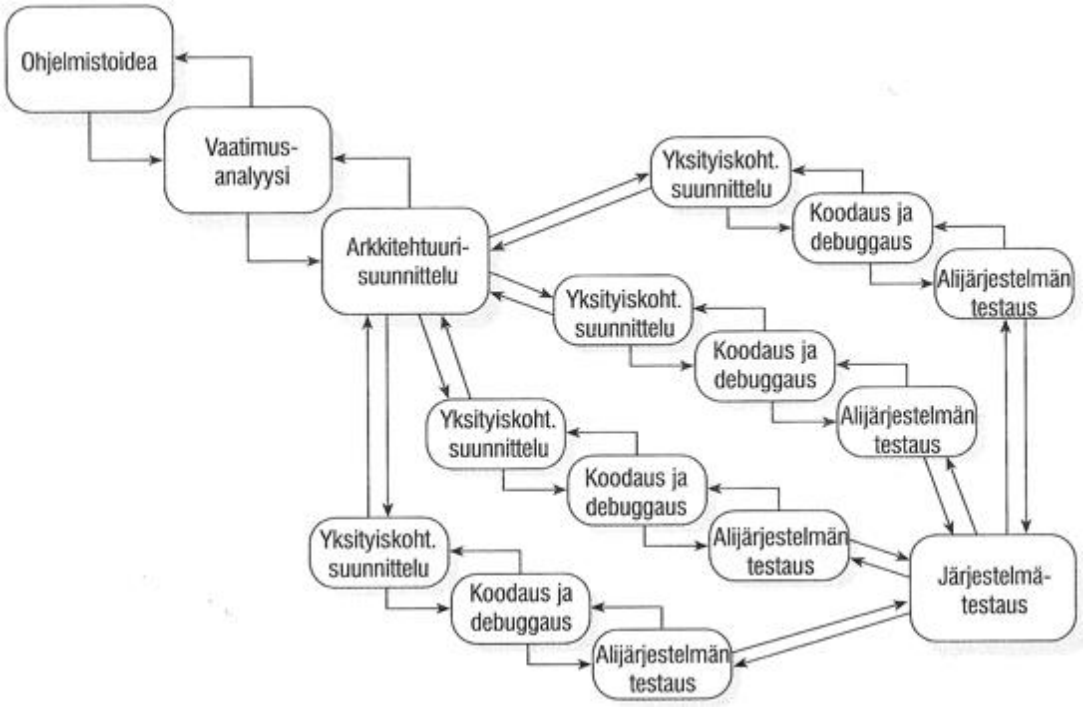
Vesiputousmalli on dokumenttiohjautuva. Tällä tarkoitetaan sitä, että vaiheesta toiseen kulkevat tärkeimmät työtulokset ovat dokumentteja. Projekti pitää katselmuksen jokaisen vaiheen loputtua, ja seuraavaan vaiheeseen siirrytään vasta kun edellinen on täysin valmis. (McConnell, 2002, s. 136)

Menetelmä on saanut osakseen paljon kritiikkiä siitä, että kaikki vaatimukset eivät ole lähes koskaan tiedossa projektin alkuvaiheessa. Dokumentoinnin osuus nähdään myös raskaaksi muihin toimintoihin nähden. Mallia noudattamalla kestää myös todella kauan, ennen kuin ohjelmistoa päästään testaamaan. (Haikala & Mikkonen, 2011, s. 37; McConnell, 2002, s. 137; Steinberg & Palmer, 2004, s. 19-21; Sommerville, 2001, s. 46)

Tapion (2010) mukaan menetelmän heikkous monimutkaisissa projekteissa on asiakkaan vaillinainen osallistuminen, jonka vuoksi on haastavaa päästä haluttuun lopputulokseen. (Tapio, 2010, s. 38-41) Menetelmä on kuitenkin usein käyttökelpoinen, jos taaksepäin iterointi sallitaan, ja eri vaiheita voidaan käynnistää jo ennen edellisen päättymistä. (Haikala & Mikkonen, 2011, s. 37; McConnell, 2002, s. 137; Steinberg & Palmer, 2004, s. 19-21)

Tätä menetelmää voisi pitää ohjelmistosuunnittelussa jonkinlaisena tukijalkana. Uudemmista menetelmistä voidaan huomata, että ne perustuvat vesiputousmalliin tavalla tai toisella. (Haikala & Mikkonen, 2011, s. 37)

Puhtaassa vesiputousmallissa nähdään nopean ja tehokkaan kehittämisen esteenä usein se, että yksityiskohtainen suunnittelu on saatava valmiiksi ennen koodauksen aloittamista. Tätä heikkoutta on pyritty korjaamaan aliprojekteihin jaetulla vesiputousmallilla, joka on puhtaan vesiputousmallin yksi muunnelmista (kuva 3). Tässä muunnelmassa arkkitehtuurisuunnittelun jälkeen projekti jaetaan itsenäisiin osiin, jolloin nopeammat ja helpommat osat projektista voi edistyä nopeammin kuin muut osat. (McConnell, 2002, s. 145)

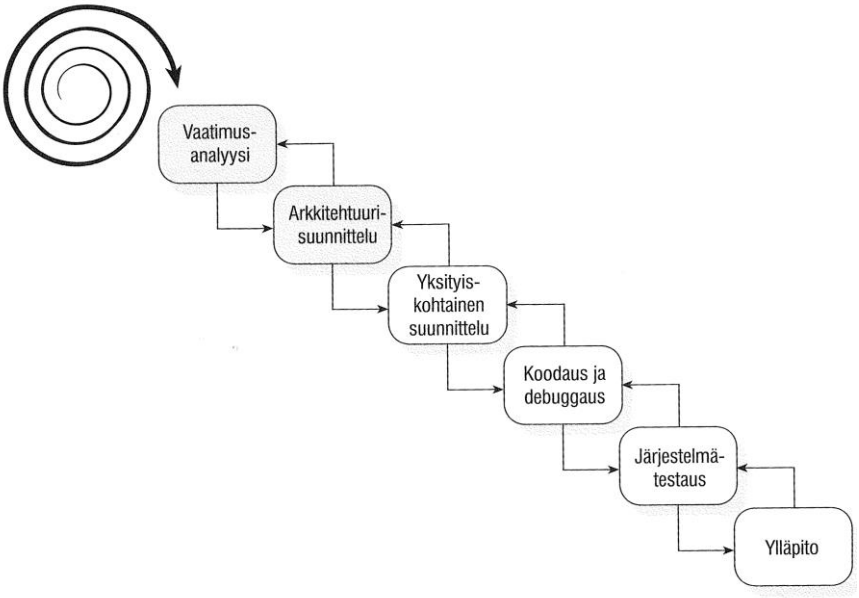


Kuva 3. Vesiputousmalli aliprojekteilla. (McConnell, 2002, s. 145)

Miksi viivästyttää yksinkertaisempien toimintojen valmistumista muutamien vaikeampien osien takia? Aliprojekteihin jaettu vesiputousmalli sallii aliprojektien edetä omaa vauhtiaan arkkitehtuurisuunnittelun jälkeen. Tämä tehostaa projektin läpivientiä (McConnell, 2002, s. 145). Mallin huonona puolena nähdään ennalta-arvaamattomat riippuvuussuhteet. Tätä voi osittain ratkaista eliminoimalla riippuvuussuhteet arkkitehtuurivaiheessa tai odottamalla yksityiskohtaisen suunnittelun valmistumista, ennen projektin jakamista aliprojekteihin. (McConnell, 2002, s. 145)

Monesti projektin aloitusvaiheessa ei tiedetä kaikkia ohjelmistovaatimuksia. Muitakin asioita voi olla pimennossa, kuten suoritustehon riittävyys. Tällöin puhtaalla vesiputousmallilla voi olla vaikea lähteä etenemään. Tätä ongelmallista tilannetta on pyritty helpottamaan riskejä vähentävällä vesiputousmallilla, jossa vesiputouksen eteen on lisätty spriraalimalli (kuva 4). (McConnell, 2002, s. 145-147)

Spriraalimallia kuvataan tarkemmin luvussa 3.2.2. Tässä yhteydessä tarkoitus on tehdä yksi prototyyppikierros spiraalimallilla ennen varsinaisen projektin suorittamista vesiputousmallilla. Prototyyppikierroksen avulla pyritään selvittämään jokin suuri riski ennen vaatimusanalyysin aloittamista. Riski voisi olla esimerkiksi epätietoisuus suunnitellun prosessorin tehon riittävyydestä kyseisessä sovelluksessa. Spiraalimallilla luotua protoa voidaan myös esitellä asiakkaalle, ja saada näin täydellisemmät vaatimusmäärittelyt ennen varsinaisen projektin aloittamista. (McConnell, 2002, s. 145-147)



Kuva 4. Riskejä vähentävä vesiputousmalli (McConnell, 2002, s. 146)

Vesiputousmallissa heikkoutena nähdään vaatimusmäärittelyn täydellistä valmistumista ennen muiden vaiheiden aloittamista. Tilanne on looginen, mutta usein epärealistinen, vaatimusten täydellinen määrittäminen projektin alkuvaiheessa on usein mahdotonta. (McConnell, 2002, s. 146)

Spiraalissa voidaan kehittää esimerkiksi käyttöliittymäprototyyppi, jolla voidaan selvittää esimerkiksi suorituskykyriskiä. Riskiä ehkäisevässä spiraalissa voidaan käyttää myös järjestelmäkuvausta, haastatella käyttäjiä tai käyttää mitä tahansa vaatimustenkeräystekniikoita, jotka tuntuvat sopivilta. Tämän avulla pyritään helpottamaan vaatimustenmäärittelyä prototyypin avulla ennen projektin aloittamista vesiputousmallin avulla. (McConnell, 2002, s. 146)

3.1.2 Spiraalimalli

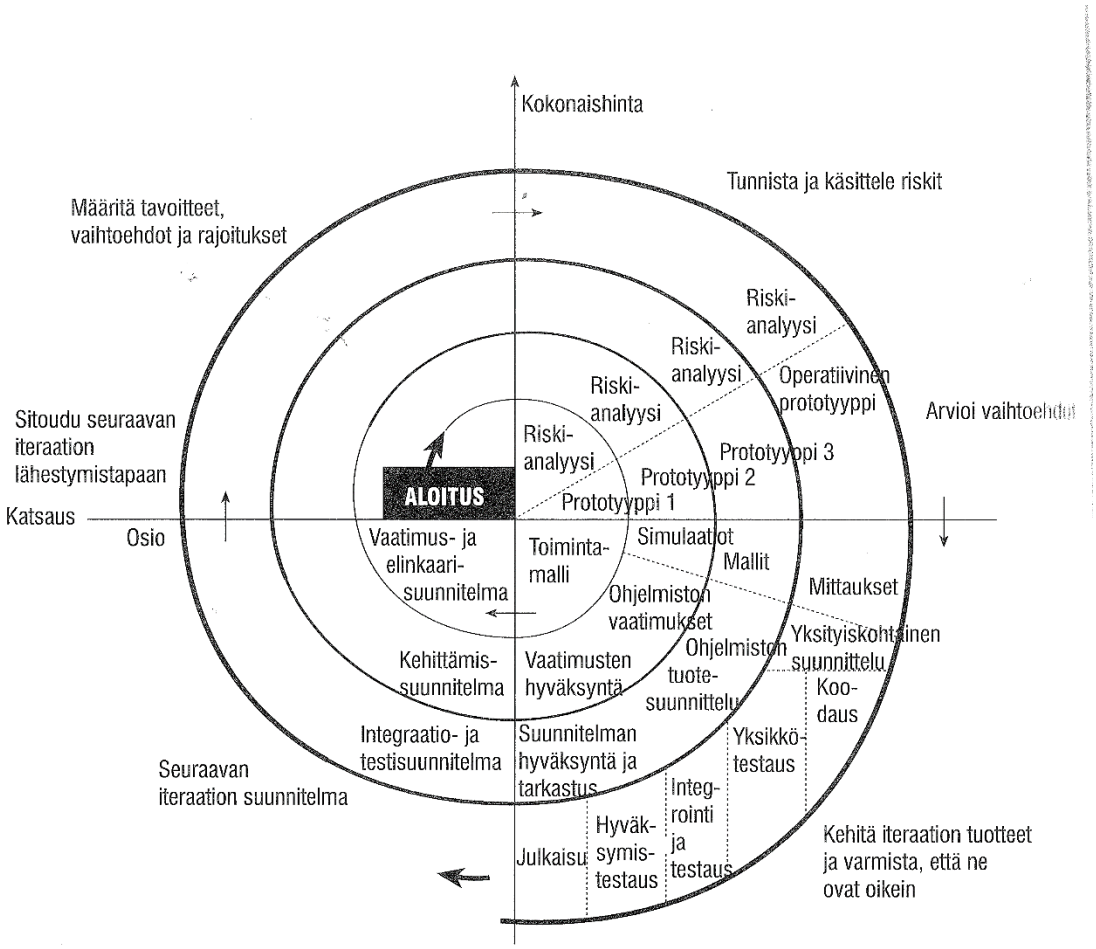
Spiraalimallin esitteli alun perin Boehm vuonna 1988, ja se on laajasti tunnettu ohjelmiston kehitysmenetelmä vielä tänäkin päivänä. Spriraalimalli kuvaa hyvin sitä, että jokainen kierros spiraalissa on edellistä suurempi, sillä ohjelmisto laajenee ajan funktiona. (Sommerville, 2001, s. 53; Tomayko & Hazzan, 2004, s. 15; Pfleeger, 1998, s. 57)

Spiraalimalli (kuva 5) on riskeihin suuntautunut elinkaarimalli, joka pilkkoo ohjelmistoprojektin pienempiin miniprojekteihin. Jokainen miniprojekti keskittyy yhteen tai useampaan pääriskiin. (McConnell, 2002, s. 141; Sommerville, 2001, s. 54)

Spiraalimallissa ideana on aloittaa ohjelmiston ydinprosesseista, jotka ovat vähiten tunnettuja. Tarkoituksena on siis aloittaa niistä osista, joiden uskotaan tarvitsevan eniten tutkimusta ennen toteutusta. Nämä kohdat ovat riskipitoisimpia. (Tomayko & Hazzan, 2004, s. 15)

Näin pyritään kartoittamaan kaikki projektin pääriskit. Suurimmat riskit suoritetaan aina ensin. Tätä toistetaan, kunnes ohjelmisto on valmis. (Tomayko & Hazzan, 2004, s. 15) Menetelmän hyvänä puolena on se, että ensimmäiset iteraatiot ovat halvimpia, eli projektin edetessä jäljellä on aina vain pienempiriskisiä tehtäviä jäljellä. Tilanne on suotuisa, sillä riskien muodostamat ongelmat ovat sitä kalliimpia, mitä myöhemmin ne havaitaan. (McConnell, 2002, s. 141)

”Riskin” käsite tässä yhteydessä on laaja, se voi tarkoittaa epäselvää asiakasvaatimusta, huonosti ymmärrettyä arkkitehtuuria, tai esimerkiksi suorituskykyongelmia. (McConnell, 2002, s. 141-143)



Kuva 5. Spriraalimallinen ohjelmistokehitys. (McConnell, 2002, s. 142)

Spiraalimallin jokainen iteraatio sisältää seuraavat askeleet:

1. Selvitä tavoitteet, vaihtoehdot ja rajoitukset.
2. Tunnista ja löydä riskit
3. Arvioi vaihtoehdot
4. Kehitä iteraatioon kuuluvat toimitukset ja varmista, että ne ovat oikein.
5. Suunnittele seuraavaa iteraatiota.
6. Sitoudu lähestymistapaan seuraavaa iteraatiota varten. (McConnell, 2002, s. 141)

Spiraalimallin iteraatioiden määrä ei ole kiinteä, vaan projektikohtainen. Mallin aikaisemmat iteraatiot ovat halvempia ja nopeampia kuin myöhemmät iteraatiot. Toimintaperiaatteiden kehittäminen on halvempaa kuin vaatimusten määrittely, joka

puolestaan on halvempi kuin suunnitelma, tuotteen toteuttaminen ja testaus. Spiraalimallia käytetään usein yhdessä muiden toimintamallien kanssa. Kun riskit ovat alennettu sopivalle tasolle vaikkapa prototyyppien avulla, projekti voidaan viedä loppuun esimerkiksi vesiputousmallilla. (McConnell, 2002, s. 141; Sommerville, 2001, s. 55)

Myös muissa malleissa voidaan hyödyntää väliaikaisesti spiraalimallia. Jos projektin edetessä ajatellaan esimerkiksi, että suorituskyky on suuri ja todennäköinen riski, voidaan suorittaa prototyyppi-iteraatio spiraalimallilla, ja selvittää riski tarkemmin. Spiraalimallin yhtenä suurimmista eduista nähdään siinä, että kun kustannukset nousevat projektin edetessä, riskit vähenevät. Suuret ohjelmistomuutokset projektin loppupuolella ovat hyvin kalliita siihen verrattuna, että ne korjattaisiin jo projektin alkuvaiheessa. (McConnell, 2002, s. 142-143; Sommerville, 2001, s. 54)

Taulukossa 6 on listattu spiraalimallin yleisimpiä hyötyjä ja haittoja. Monet menetelmän heikkoudet heijastuvat siitä, että menetelmä on raskas ja monimutkainen hallittavaksi.

Taulukko 6. Spiraalimallin hyvät ja huonot puolet (Tutorialspoint, 2017)

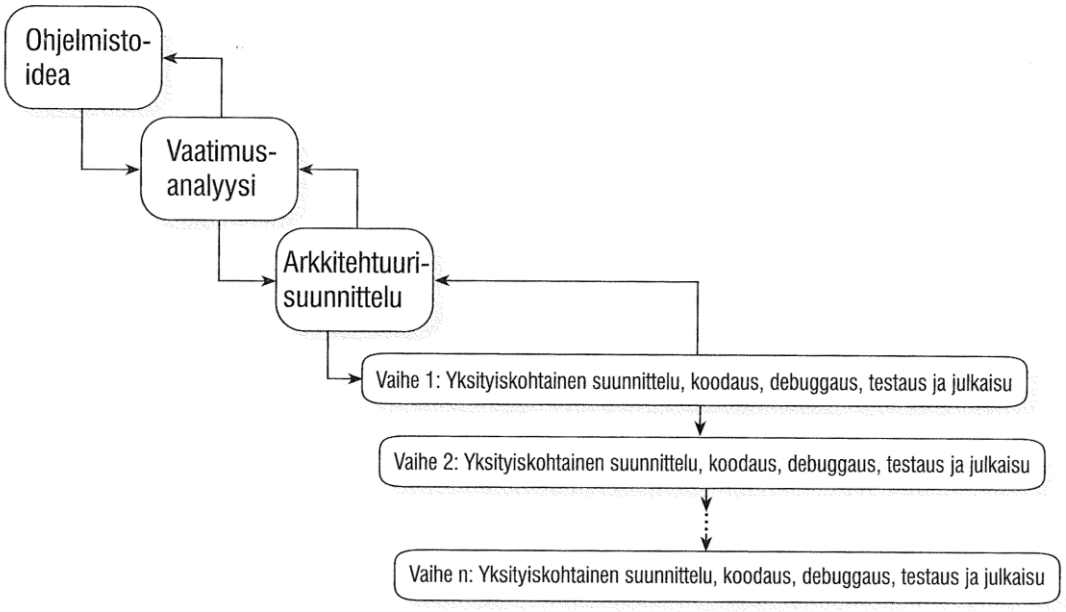
Hyödyt	Haitat
Muutokset vaatimuksiin onnistuvat sujuvasti	Johtaminen on monimutkaista
Sallii prototyyppien laajan käytön	Projektin aikataulua on vaikea arvioida
Vaatimukset voidaan kerätä tehokkaammin	Ei sovellu pieniin projekteihin
Loppukäyttäjät näkevät järjestelmän aikaisessa vaiheessa	Ei sovellu vähäriskisiin projekteihin
Kehitys voidaan jakaa pieniin osiin	Prosessi on monimutkainen
Riskipitoiset osa-alueet tehdään ensin	Spiraali voi jatkua loputtomiin
Parantaa riskienhallintaa	Suuri määrä välivaiheita vaatii paljon dokumentointia

Spiraalimallin haittapuolena nähdään sen monimutkaisuus. Menetelmän noudattaminen vaatii tunnollisen, valppaan ja asiantuntevan hallinnon. On usein vaikeaa

määritellä objektiivisia ja varmennettavia tarkastuspisteitä, jotka ilmaisevat, onko tiimi valmis siirtymään seuraavaan iteraatioon. Spiraalimallin hyvänä puolena on se, että se sallii ominaisuuksien lisäämisen tuotteeseen sen jälkeen, kun ne tulevat saataville (asiakaspalaute edelliseltä iteraatiolta). Tämä varmistaa sen, ettei vaatimus luo ristiriitoja edellisten vaatimusten ja suunnittelun kanssa. Menetelmän käyttö myös varmistaa, että jokaisen iteraation päätteeksi saadaan palautetta käyttäjiltä. (McConnell, 2002, s. 143; Tutorialspoint, 2017)

3.1.3 Vaiheistettu toimitus

Vaiheistetun toimituksen (kuva 6) kehitysmallia käytetään projekteissa, joissa tiedetään, mitä ollaan tekemässä. Vaiheen 1 jälkeen tuotetta esitellään asiakkaalle. Projekti jatkuu niin, että aina seuraavan vaiheen jälkeen ohjelmistosta julkaistaan asiakkaalle versio käytettäväksi. (McConnell, 2002, s. 148)



Kuva 6. Vaiheistetun toimituksen elinkaarimalli. (McConnell, 2002, s. 148)

Malli ei poikkea paljon vesiputousmallista projektin alkuosassa. Vaiheistetun toimituksen pääetuna on se, että asiakkaat saavat tuotteen käyttöön jo ennen projektin loppumista. Malli tarjoaa asiakkaalle myös selviä edistyksen merkkejä sen sijaan, että ohjelmisto julkaistaisiin vasta projektin päätyttyä. Edistymisen merkit

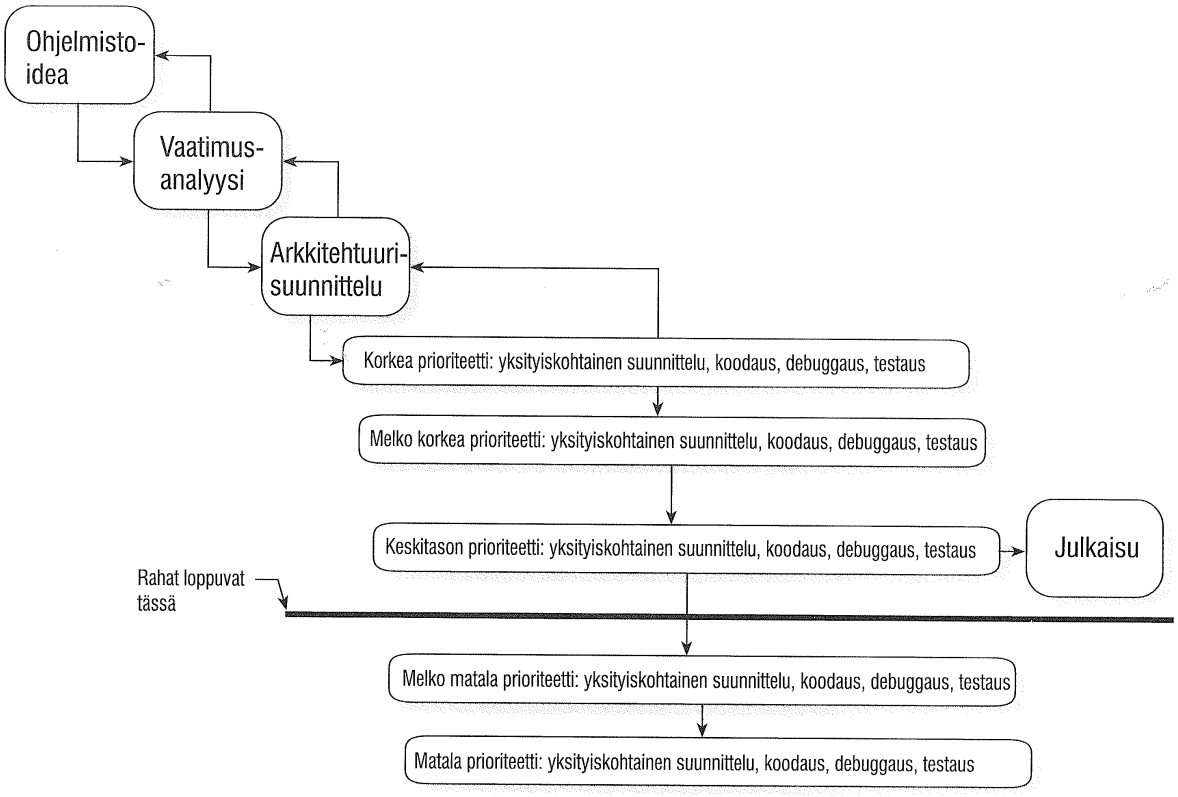
voivat olla erittäin tärkeitä aikataulupaineiden pitämiseksi hallitulla tasolla. (McConnell, 2002, s. 148)

Tyypillisesti ensimmäisen vaiheen julkaisuun pyritään saamaan kaikkein tärkeimmät toiminnot, jotta asiakas saisi todellista hyötyä jokaisesta julkaistusta versiosta. (McConnell, 2002, s. 148)

”Suurin vaiheistetun toimituksen haittapuoli on, ettei se toimi ilman tarkkaa suunnittelua sekä hallinnollisella että teknisellä tasolla. Hallinnollisella tasolla pitää varmistaa, että suunnitellut vaiheet ovat merkityksellisiä asiakkaalle ja että työ jaetaan projektin henkilöstön kesken siten, että kaikki saavat työnsä tehtyä vaiheen aikarajaan mennessä. Teknisellä tasolla on varmistettava, että kaikki tekniset riippuvuussuhteet tuotteen eri komponenttien välillä on selvitetty. Yleinen virhe on viivästyttää jonkin komponentin kehittäminen vaiheeseen 4 ja sitten havaita, että vaiheeseen 2 tarkoitettu komponentti ei voi toimia ilman sitä” (McConnell, 2002, s. 149)

3.1.4 Aikataulun mukainen suunnittelu

Tämä malli on hyvin samankaltainen vaiheistetun toimituksen mallin kanssa. Ohjelmistoa kehitetään vaiheissa, ja jokaisen vaiheen päätyttyä asiakkaalle toimitetaan versio. Tässä mallissa (kuva 7) ei kuitenkaan tiedetä, valmistuuko ohjelmisto annettuun aikatauluun mennessä. Menetelmää käytetään aikataulullisesti peräänantamattomissa projekteissa. (McConnell, 2002, s. 149)



Kuva 7. Aikataulun mukainen suunnittelu (McConnell, 2002, s. 150)

Tällä mallilla voidaan varmistaa, että kun peräänantamaton aikaraja tulee vastaan, tuotekehitystiimillä on jonkinlainen toimituskelpoinen versio olemassa. Mallia käytetään myös tiukkojen budjettien projekteissa. Projekti keskeytetään, kun rahat tai aika loppuvat; tällöin edellinen julkaistu versio on projektin lopputuotos. Jos projekti kohtaa loppunsa ennen valmistumista, silloin todellakin halutaan, että julkaistu versio sisältää kaikki tärkeimmät ominaisuudet. Ei haluta tilannetta jossa jokin tärkeä ominaisuus on jäänyt valmistumatta vain siksi, että aikaa on tuhlattu epäolennaisten asioiden hiomiseen. (McConnell, 2002, s. 150)

Tätä menetelmää käytettäessä kriittistä on vaiheiden suunnittelu siten, että edellisessä vaiheessa on aina tärkeämmät ominaisuudet kuin myöhemmässä vaiheessa. Kuten edellisessäkin menetelmässä, tässäkin toimintojen riippuvuussuhteiden määrittäminen on teknisesti haastavaa projektin johtamisen kannalta. (McConnell, 2002, s. 150)

Jos projekti on luonteeltaan sellainen, että aikataulun uskotaan pitävän, tämä on tehoton malli suorittaa tällainen projekti. Kuitenkin tiukan aikataulun tai budjetin projekteissa tämä voi olla hyvin käyttökelpoinen malli. (McConnell, 2002, s. 150)

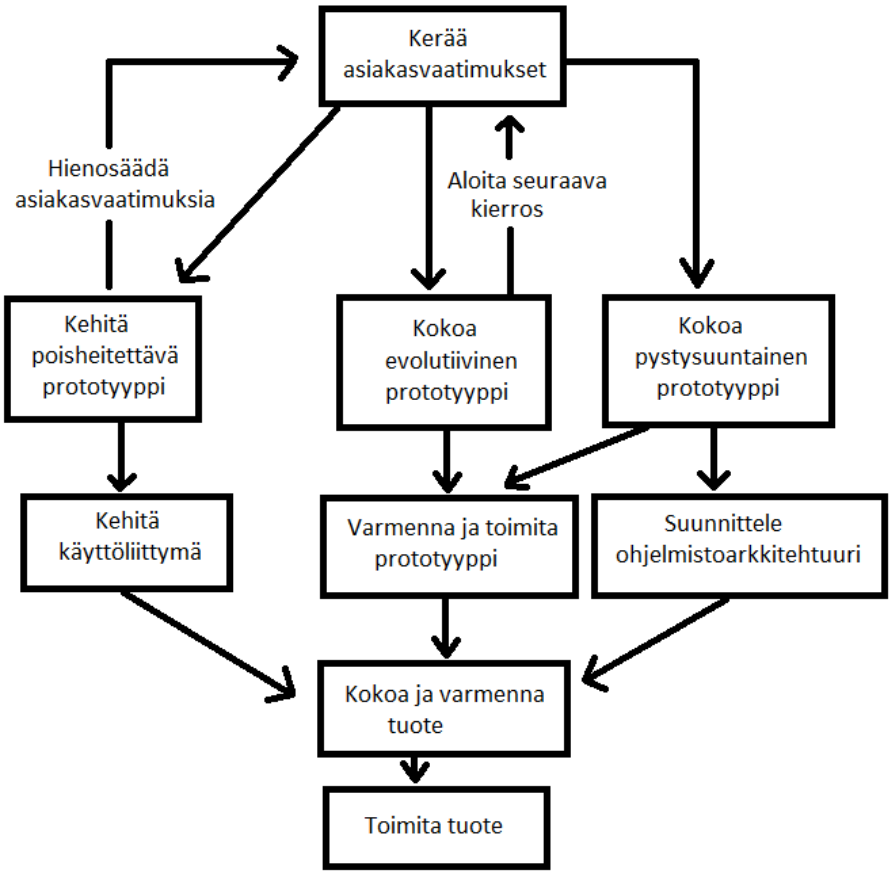
3.1.5 Prototyypit

Prototyyppien käyttämisessä on kyse siitä, että tehdään suunnitellusti vaillinainen prototyyppi, jolla voidaan testata jotain kriittistä osaa. Tällaisia osia voivat olla esimerkiksi käyttöliittymälogiikka, suorituskyky, muistin kulutus taikka ajoitukseen liittyvien ongelmakohtien tarkempi tutkiminen. (Haikala & Mikkonen, 2011, s. 38; Sommerville, 2001, s. 46; Wiegers, 2003, s. 234)

Prototyyppejä käytetään ohjelmistoissa kolmesta eri syystä. Ensimmäisenä syynä projektissa voidaan haluta selvittää ja täydentää asiakasvaatimuksia. Prototyyppien käyttäminen ja esitteleminen paljastavat usein puutteita asiakasvaatimuksissa. Toinen hyvä syy hyödyntää prototyyppien käyttämistä on selvittää erilaisten toteutusvaihtoehtojen eroja sovelluksessa. Erilaisista toteutusvaihtoehdoista tehdään tarvittavilta osin toimivia prototyyppejä, joita voidaan demonstroida erojen löytämiseksi. (Wiegers, 2003, s. 234)

Kolmas syy on kasvattaa prototyypistä asteittain valmis tuote. Prototyyppiä kehitetään tällöin useissa pienissä kehityssykleissä, haetaan palautetta asiakkaalta ja kehitetään taas lisää. (Wiegers, 2003, s. 234)

Kuvassa 8 on esitetty useita eri tapoja tehdä prototyyppi. Kuva sisältää myös pystysuuntaisen prototyypin, jota ei tässä työssä tarkemmin käsitellä. Pystysuuntainen prototyyppi sisältää arkkitehtuurisesti kaikki ohjelmiston toiminnan tasot. Tällaista prototyyppiä käytetään esimerkiksi silloin, kun halutaan testata ohjelmiston ajoitusvaatimuksia. Ohjelmiston eri tasot vaikuttavat kokonaisuuden toimivuuteen, niinpä prototyyppi tulee kattaa kaikki ohjelmiston tasot mielekästä testausta varten. (Wiegers, 2003, s. 236)



Kuva 8. Ohjelmiston kehitysprosessi prototyyppien avulla. (Wiegers, 2003, s. 239)

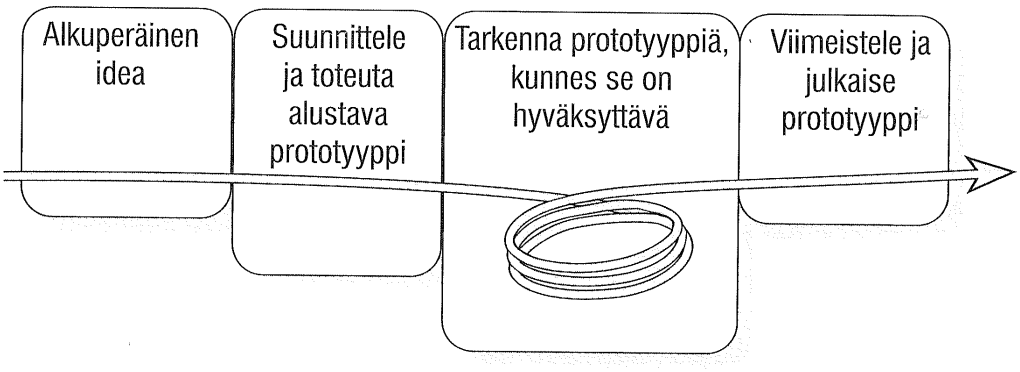
Protoilussa käytetään yleensä kahta eri vaihtoehtoa. Evoluutioprototyyppi on sellainen, että kyseisestä prototyypistä lopuksi tulee varsinainen tuote. Malli muistuttaa iteratiivista vesiputousmallia. Projekti jaetaan osiin ja kukin osa määritellään, suunnitellaan, toteutetaan, testataan ja lopuksi hyväksytään. Osia yhdistelemällä lähestytään lopullista tuotetta. (Haikala & Mikkonen, 2011, s. 38-40; Sommerville, 2001, s. 46)

Toinen vaihtoehto on käyttää kertakäyttöistä prototyyppiä, tässä mallissa tehdään useita erilaisia toimintaratkaisuja ja valitaan paras ratkaisu. Menetelmää kutsutaan myös poisheitettäväksi prototyypiksi. Tyypillisesti varsinainen tuote tehdään uudestaan, eikä edellistä prototyyppiä hyödynnetä enää. Yksinkertainen esimerkki poisheitettävistä prototyypeistä voisi olla käyttöliittymien kuvia paperiarkeille, joita esitellään asiakkaalle. (Haikala & Mikkonen, 2011, s. 38-40)

Poisheitettävää prototyyppi-menetelmää käytetään usein silloin, kun asiakasvaatimukset ovat huonosti ymmärrettyjä. Prototyyppikokeiluilla kehitetään parempaa

ymmärrystä asiakasvaatimuksista. Nimensä mukaisesti poisheitettävä prototyyppi jää käyttämättömäksi sen jälkeen, kun se on suorittanut tehtävänsä. Niinpä prototyyppi tulisi suunnitellusti rakentaa mahdollisimman kustannustehokkaasti ja nopeasti. Prototyyppi rakennetaan selvittämään jotakin tiettyä ongelmaa, ja tehtävän suorittamiseen tulee käyttää minimaalinen, mutta riittävä määrä resursseja. (Wiegers, 2003, s. 236-237; Sommerville, 2001, s. 46)

Evoluutioprototyypissä (kuva 9) on vaarana, että huonosti toteutettu prototyyppi jää elämään osaksi varsinaista järjestelmää. Näin voi käydä varsinkin, jos projektin aikataulu luo paljon painetta. Evolutiivisessa prototyypissä aloitetaan usein kehittämällä näkyvimmät piirteet ohjelmistosta. Järjestelmän osaa esitellään asiakkaalle ja kehitystä jatketaan saatu palaute huomioiden. Jossain vaiheessa asiakas ja kehittäjät ovat samaa mieltä siitä, että tuote on riittävän hyvä, jolloin prototyyppi julkaistaan lopullisena tuotteena. (McConnell, 2002, s. 147; Wiegers, 2003, s. 236-237; Haikala & Mikkonen, 2011, s. 39)



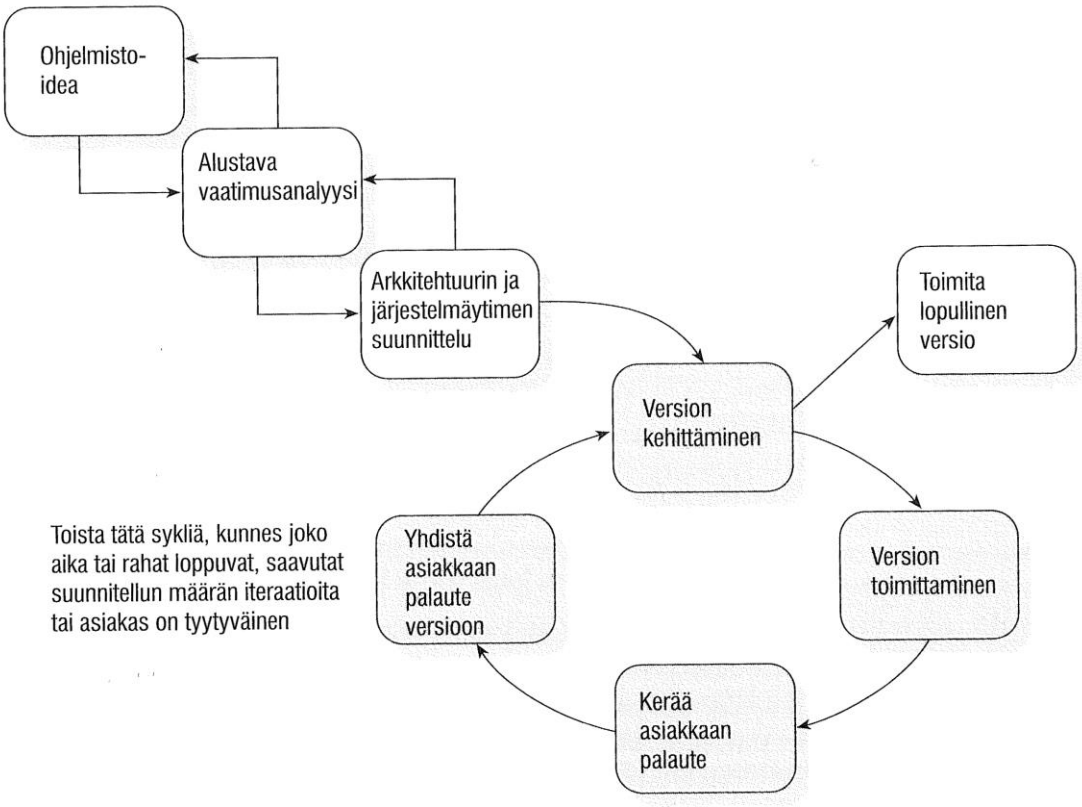
Kuva 9. Evolutiivinen protoilu. (McConnell, 2002, s. 147)

Evolutiivisen prototyypin kehittämismenetelmä on erityisen käyttökelpoinen, kun vaatimukset muuttuvat nopeasti. Kun asiakkaat, eivätkä kehittäjät, kummatkaan tunne sovellusaluetta kunnolla, on evolutiivinen protoilu hyvä lähtökohta projektin läpiviemiseksi. (McConnell, 2002, s. 147)

Tämän menetelmän huonona puolena pidetään sitä, että projektin alkuvaiheessa on mahdotonta arvioida projektin kestoa. Ei voida tietää, paljonko hyväksyttävän

prototyypin valmistumiseen kuluu aikaa. Ei myöskään voida tietää montako iteraatiota se vaatii. Toisaalta evolutiivinen protoilu on asiakkaan suuntaan erittäin näkyvä kehittämismalli. Asiakas kykenee hahmottamaan projektin edistymisen erittäin hyvin. (McConnell, 2002, s. 149)

Evolutiivinen toimitus (kuva 10) on elinkaarimalli, joka asettuu evolutiivisen protoilun ja vaiheistetun toimituksen välimaastoon. Ohjelmistosta kehitetään versio, esitellään se asiakkaalle, ja parannetaan ohjelmistoa saadun palautteen mukaan. Kyseinen menetelmä muistuttaa evolutiivista protoilua, mikäli suuri osa asiakkaan haluamista muutoksista toteutetaan. Mikäli vain harva asiakkaan muutospyyntö lopulta hyväksytään, malli muistuttaa tällöin vaiheistettua toimitusta. (McConnell, 2002, s. 149)



Kuva 10. Evolutiivinen toimitusmalli. (McConnell, 2002, s. 151)

Evolutiivisessa prototyypissä keskitytään aluksi enemmän ohjelmiston visuaalisiin piirteisiin. Evolutiivisessa toimituksessa sen sijaan alkupainotus on järjestelmän ytimessä. Tällainen koostuu matalan tason järjestelmäfunktioista, joita asiakkaan palaute ei tyypillisesti tule muuttamaan. (McConnell, 2002, s. 152)

Evolutiivinen prototyyppi on vesiputousmallia parempi menetelmä silloin, kun asiakas voi hyötyä myös osittain toimivasta sovelluksesta. Tässä menetelmässä asiakas saa useita osittain toimivia versioita ennen lopullista toimitusta. (Sommerville, 2001, s. 46-47)

3.2 Ketterät ohjelmistomenetelmät

1990-luvun loppupuolella huomattiin tarve myös toisenlaisille ohjelmiston kehitysmenetelmille, tekniikan nopean kehityksen vuoksi. Joukko ohjelmistokehittäjiä ja konsultteja kokoontui Utahin hiihtokeskuksessa kaksi päivää kestävään kokoukseen. (Haikala & Mikkonen, 2011, s.43; Tapio, 2010, s. 52; Tomayko & Hazzan, 2004, s. 18-19; Hänninen, 2010, s. 9)

Ryhmän yhdistelemät uudet teoriat uudenlaisista toimintatavoista loivat joukon uusia ohjelmiston kehitysmenetelmiä, joita alettiin kutsua ketteriksi ohjelmiston kehitysmenetelmiksi (Agile Methods). Ryhmä perusti ketterien menetelmien edistämistä ajavan Agile Alliance – järjestön, jonka ”peruskirjana” julkaistiin ”Agile Manifesto”. (Haikala & Mikkonen, 2011, s.44; Tapio, 2010, s. 52; Tomayko & Hazzan, 2004, s. 18-19; Hänninen, 2010, s. 9)

AgileManifesto (2001) korostaa neljää perusarvoa ohjelmistokehityksessä:

1. **Yksilöitä ja vuorovaikutusta**, enemmän kuin prosesseja ja työkaluja
2. **Toimivaa sovellusta**, enemmän kuin kattavaa dokumentaatiota.
3. **Asiakasyhteistyötä**, enemmän kuin sopimusneuvotteluja
4. **Muutokseen reagoimista**, enemmän kuin alkuperäisen suunnitelman noudattamista.

”*Vaikka oikeanpuoleisilla asioilla onkin arvoa, me arvostamme enemmän vasemman puolesia*.” (AgileManifesto, 2001; Hänninen, 2010, s. 9)

Manifestissa tuotiin esille 12 periaatetta, joita ketterät menetelmät arvostavat. Haikala & Mikkonen (2011, s.45) esittävät periaatteet vapaasti suomennettuna taulukossa 7.

Taulukko 7. Ketterien menetelmien periaatteet (Haikala & Mikkonen, 2011, s.45)

1	Asiakas on pidettävä tyytyväisenä toimittamalla tälle projektin alusta asti tasaisella tahdilla toimivia ohjelmistoversioita.
2	Vaatimusten muuttuminen tulee hyväksyä projektin aikana, myös myöhäisissä kehitysvaiheissa.
3	Toimivien asiakasversioiden julkaisuväli voi vaihdella muutamasta viikosta muutamaan kuukauteen.
4	Liiketoiminta- ja kehitysrooleissa olevien työntekijöiden tulee työskennellä yhdessä päivittäin koko projektin ajan.
5	Projektit tulee rakentaa motivoituneiden yksilöiden ympärille. Heille tulee antaa ympäristö ja tuki, jota he tarvitsevat työn tekemiseksi. Luota siihen, että he saavat työn tehtyä.
6	Kaikkein tehokkain kommunikointikeino sekä kehitystiimin ja ulkomaailman välillä, että itse kehitystiimissä, on keskustelu kasvokkain.
7	Toimiva ohjelmisto on projektin edistymisen tärkein mittari.
8	Ketterät menetelmät edistävät tasaista kehitystahtia. Projektin parissa työskentelevien henkilöiden tulisi kyetä pitämään työtahti ja -kuorma mahdollisimman tasaisena, eikä ylitöitä kuulu tehdä.
9	Huomion jatkuva kiinnittäminen tekniseen erinomaisuuteen ja hyvään suunnitteluun tehostaa ketteryyttä.
10	Asiat pitäisi aina pyrkiä tekemään mahdollisimman yksinkertaisella tavalla: Minimoidaan vähemmän tärkeät tehtävät.
11	Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseohjautuvasti organisoituneissa tiimeissä.
12	Tiimin tulee selvittää säännöllisin väliajoin, miten se voisi tulla aikaisempaa tehokkaammaksi ja hienosäätää käytettävää työtapaa ja prosessia sen mukaisesti.

Uusissa menetelmissä asiakkaan rooli oli huomattavasti merkittävämmässä roolissa kuin perinteissä menetelmissä. Uusien toimintatapojen uskottiin kykenevän

vastaamaan paremmin nykyaikaisten ohjelmistoprojektien tarpeeseen. (Beck & Andres, 2005; Tapio, 2010; Tomayko & Hazzan, 2004, s. 18-19)

Ennen vuotta 2000 oli julkaistu jo kymmenkunta kevyttä iteratiivista ohjelmistonkehitysmenetelmää, joista eniten huomiota on saanut Kent Beckin kehittämä eXtreme Programming (XP). (Haikala & Mikkonen, 2011)

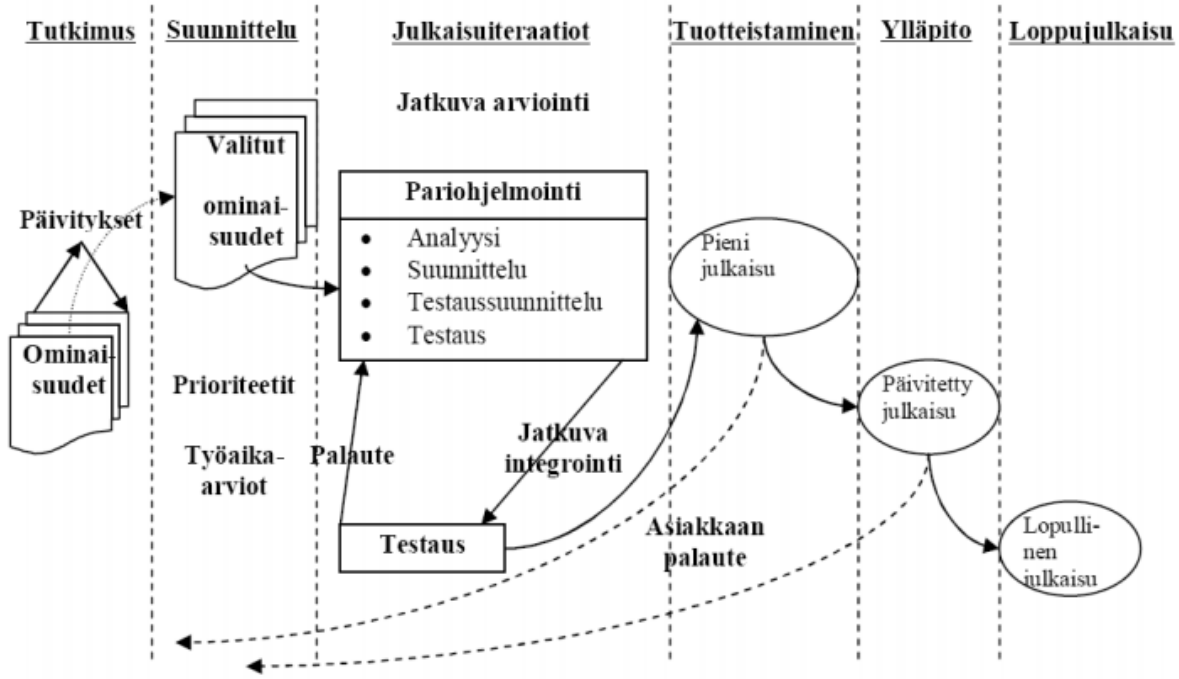
XP-menetelmää pidetään kaikkein tunnetuimpana ja jopa tärkeimpänä kaikista ketteristä menetelmistä, vaikka Beck kokosikin vain jo olemassa olevia ideoita ja malleja yhteen omaksi teoriakseen. (Tapio, 2010, s. 55)

3.2.1 Extreme programming (XP)

XP-menetelmä on kehitetty nimenomaan nopeasti muutoksiin reagoivaksi ohjelmistokehitysmenetelmäksi. Ohjelmistoissa kaikki asiat muuttuvat: vaatimukset, suunnittelu, markkinat, ohjelmoijat, teknologia sekä tiimit muuttuvat. Ongelma ei ole se, että muutoksia tapahtuu, sillä niitä tulee tapahtumaan. Ongelma on se, että emme kykene mukautumaan muutoksiin. (Beck & Andres, 2005, s. 11)

Menetelmän kehittäjänä pidetään Kent Beckiä, joka kuvaa XP-menetelmän kevyeksi, tehokkaaksi, joustavaksi ja pieniriskiseksi menetelmäksi. XP-menetelmä sopii käytettäväksi pieniin ja keskisuuriin ohjelmistoprojekteihin. Menetelmässä pyritään karsimaan kaikkia ylimääräisiä resursseja vieviä toimintoja, ja keskittymään varsinaisen koodin luomiseen. (Hänninen, 2010, s. 19)

Sana ”extreme” (äärimmäinen) tulee menetelmän nimeen Beckin mukaan siitä, että hyväksi havaitut käytännöt on menetelmässä viety äärimmäisyyksiin. Menetelmän käyttämiä tekniikoita kuvataan tarkemmin seuraavissa luvuissa. XP-menetelmää käyttävän projektin vaiheet on esitetty kuvassa 11. (Beck & Andres, 2005; Hänninen, 2010, s. 19)



Kuva 11. XP-menetelmän prosessikuvaus. (Tapio, 2010, s. 49)

XP-menetelmä ottaa tarkasti kantaa siihen, kuinka ohjelmointityö tehdään. Myös asiakkaan rooli poikkeaa selkeästi perinteisistä menetelmistä siinä, että palaute on annettava epätavallisessa muodossa - käyttäjätarinoiden avulla.

3.2.1.1 Käyttäjätarinat

XP-menetelmällä tehdyissä tuotekehitysprojekteissa kaikki tuotevaatimukset esitellään käyttäjätarinoilla (user stories). Tarinat kirjoitetaan muutamalla virkkeellä irrallisille paperinpaloille, jotka ovat kooltaan noin kämmenen kokoisia. Tarinat kirjoittaa asiakas. Asiakas voi olla myös yrityksen henkilö, joka on läheisessä kanssakäymisessä varsinaisen asiakkaan kanssa. (Jeffries & Anderson & Hendrickson, 2001, s. 23-30)

Kukin tarina on tehtävä, joka valmiilla tuotteella pitää kyetä suorittamaan. Kun tuote kykenee toteuttamaan kaikki tarinat, se on valmis. Tarinat ovat tuotekehityksen näkökulmasta ainoa vaatimusten lähde tuotteelle. Tämä tarkoittaa, että yhdessä projektissa näitä vaatimustarinoita tulee olla mahdollisesti satoja kappaleita. (Jeffries ym., 2001, s. 23-30)

Ohjelmoijat eivät itse kirjoita käyttäjätarinoita, vaan toteuttavat niiden sisältämiä vaatimuksia, ja tarvittaessa pyytävät asiakkaalta lisätarkennuksia. Tarinoiden sisältämät vaatimukset tulee olla suuruudeltaan sellaisia, ettei tuotekehityksellä mene vaatimuksen toteuttamiseen kuin maksimissaan kaksi viikkoa. Mikäli vaatimus on tätä suurempi, asiakkaan tulisi jakaa vaatimus pienempiin osiin. (Jeffries ym., 2001, s. 23-30)

Asiakkaan on tietysti mahdoton kirjoittaa kaikki tuotevaatimukset yhdellä kertaa usealle sadalle paperilapulle. Kaikkia vaatimuksia ei voida tietää ennalta, ja kaikkia toimintoja ei asiakas voi huomata pyytää. Asia korjaantuu sillä, että tuotekehityksen aikana asiakkaalle toimitetaan tuotteesta aina iteraatioiden päätyttyä uusi versio. Asiakkaan tulisi käyttää versiota, ja tässä yhteydessä lisävaatimuksia ilmenee. (Jeffries ym., 2001, s. 23-30)

Ohjelmoijat jaetaan kahteen ryhmään, joissa molemmat ryhmät arvioivat, kuinka kauan kunkin tarinan kehittäminen kestää yhdeltä ohjelmointiyksiköltä. Vaikeus ilmaistaan pisteinä, jossa esimerkiksi helpoimmaksi valittu tehtävä vastaa yhtä pistettä. Vaihtoehtoisesti yhden viikon työ vastaa yhtä pistettä. Kun kaksi ryhmää on antanut vaikeuspisteet kaikille käyttäjätarinoille, valitaan yleensä pienempi pistemäärä. (Jeffries ym., 2001, s. 37-44)

Kun ohjelmistokehittäjät tottuvat arviointiin, he ovat yllättävän tarkkoja ja nopeita arvioimaan kunkin tehtävän vaatiman pistemäärän. Kun tehtävät ovat suoritettu, niiden toteutuneet kestot kirjataan ylös, näin projektin vetäjä saa tarkan kuvan siitä, paljonko yksi piste vastaa oikeaa työaikaa. (Jeffries ym., 2001, s. 37-44)

Pisteytyksen mukaan projektin vetäjä valitsee kullekin viikolle sopivan määrän tehtäviä. Projekteissa, joissa vaatimukset (käyttäjätarinat) ovat jo alkuvaiheessa hyvin kattavasti tiedossa, voidaan helposti arvioida hyvin tarkasti projektin vaatima työaika. (Jeffries ym., 2001, s. 23-30)

Projektin edetessä asiakkaan on helppo tehdä uusia vaatimuksia tuotteelle - tarvitaan vain muutama lause paperinpalalle. Menetelmä on huomattavasti kevyempi kuin ”virallisten” laitteistovaatimusten päivittäminen ajan tasalle. Menetelmä luo myös ohjelmistokehittäjille pieniä tehtäviä, joiden valmistuminen tuottaa onnistumisen iloa. (Jeffries ym., 2001; Beck & Andres, 2005)

XP-tuotekehitys etenee viikon sykleissä. Viikon ensimmäisenä päivänä pidetään aamulla tiimipalaveri, jossa valitaan mm. tärkeimmät tarinat, jotka tulisi saada valmiiksi viikon aikana. Viikon viimeisenä päivänä julkaistaan uusi versio, jossa on mukana viikon käyttäjätarinat. (Beck & Andres, 2005, s. 44-47)

3.2.1.2 Pariohjelmointi

Kaikki ohjelmakoodi tehdään pariohjelmointina. Kaksi ohjelmoijaa istuu vierekkäin yhden tietokoneen edessä, ja ohjelmoi yhtä käyttäjätarinaa kerrallaan. Ohjelmoijista toinen suorittaa varsinaisen ohjelmoinnin (kuljettaja), ja toinen seuraa aktiivisesti vierestä (partneri). Osia vaihdetaan tasaisin väliajoin, esimerkiksi kahden tunnin välein. (Jeffries ym., 2001, s. 87-91; Beck & Andres, 2005, s. 42-43)

Ennen aloittamista pari keskustelee keskenään siitä, kuinka ohjelmointi tulisi suorittaa. Kun sopiva näkökulma löytyy, ja molemmat ymmärtävät mitä ollaan tekemässä, voidaan ohjelmointi aloittaa. Jo tämä kahden ohjelmoijan välinen keskustelu aiheuttaa sen, että valittu ohjelmointitapa on todennäköisesti optimaalisempi kuin muuten olisi. (Jeffries ym., 2001, s. 87-91; Beck & Andres, 2005, s. 42-43)

Ohjelmoinnissa kuljettajalla on idea mielessään ohjelmoinnin aikana. Ajatus siitä mitä ollaan tekemässä, tulee pitää tiukasti mielessä. Partneri voi sen sijaan keskittyä muihin asioihin. Kirjoitusvirheet, ulkoasu, rakenne, vaihtoehtoiset toteutustavat sekä potentiaaliset virhetilanteet ovat partnerille helpommin hahmotettavissa. Partnerin ei tule keskeyttää tai kommentoida vähäpätöisten syiden takia, ettei kuljettajan ajatus toteutettavasta asiasta katoa. Esille tulleet asiat käydään läpi tilanteen jälkeen. Mikäli kuljettaja ajautuu sivuraiteille, partnerin tulee tietysti puuttua asiaan välittömästi. (Jeffries ym., 2001, s. 87-91; Beck & Andres, 2005, s. 42-43)

Pariohjelmointi tuottaa laadukkaampaa ohjelmakoodia kuin yksin tehtynä. Pari tuottaa myös enemmän ohjelmakoodia kuin samat ohjelmoijat olisivat tuottaneet samassa ajassa erikseen tehtynä. Lisäksi ohjelmoinnin jälkeen kaksi henkilöä ymmärtää kaiken tehdyn, sen sijaan että kaksi ihmistä ymmärtäisi puolet tehdystä. (Jeffries ym., 2001, s. 88)

Parhaimmillaan pariohjelmointi on hyvin tehokasta. Kokenut pari voi olla lähes pysäyttämätön täydentäessään toistensa osaamista. Kun kuljettaja kirjoittaa ohjelmaa, partneri pystyy usein osoittamaan kuljettajalle mistä hänen hakemansa asia löytyy, vaikka mitään ei ehditty edes kysyä. (Jeffries ym., 2001, s. 87-91; Beck & Andres, 2005, s. 42-43)

Jos projektissa on esimerkiksi 10 ohjelmoijaa, on suositeltavaa kierrättää pareja vakiomittaisin väliajoin. Tavoitteena on, että kaikki 10 ohjelmoijaa hallitsevat koko ohjelmiston, ja että he pystyvät tarvittaessa muuttamaan mitä tahansa osaa ohjelmassa yksinään. (Beck & Andres, 2005)

Männistö (2008) teki opinnäytetyön XP-menetelmän pariohjelmoinnista. Työssä testattiin pariohjelmointia yrityksessä viikon ajan. Lopuksi suunnittelijoita haastateltiin ja menetelmästä pyrittiin löytämään hyviä ja huonoja puolia.

Lyhyen kokeilun jälkeen positiivista palautetta tuli enimmäkseen virheiden nopeasta huomaamisesta sekä toteutustavan valintakeskustelusta ennen ohjelmointia. Ajankäyttö oli myös mielekkäämpää, sillä sähköpostin katselu, sekä ”nettisurffailu” jäi vähemmälle.

Enimmäkseen palaute oli negatiivista. Toisen henkilön välitön läsnäolo ja herkeämätön tuijotus näyttivät vaikuttavan negatiivisesti usean kuljettajan keskittymiskykyyn. Koettiin että kommunikointi ja kritisointi olivat vaikeaa. Partnerina olo tuntui olevan tylsää ja väsyttävää. (Männistö, 2008, s. 38-42)

Pariohjelmointi poikkeaa huomattavasti normaaleista ohjelmointikäytännöistä. Sopeutuminen pariohjelmointiin vie aikaa. (Beck & Andres, 2005) Tämän vuoksi viikon mittainen testijakso oli todennäköisesti liian lyhyt, eikä menetelmän edut päässeet täysin näkyviin Männistön testissä.

Pariohjelmoinnin hyvänä puolena nähdään ohjelmiston laadun parannus, tehokkuus sekä tiedon jako tiimin kesken. Menetelmä myös pakottaa tiimin suosimaan yhtenäisiä tapoja ohjelmaa kirjoittaessaan.

Huonona puolena nähdään menetelmän uuvuttavuus. Harva kehittäjä jaksaa tehdä pariohjelmointia yli 6 tuntia päivässä. Joidenkin kehittäjien luonne ei sovi

pariohjelmointiin, vaikka sopeutumiseen annettaisiinkin aikaa. (Beck & Andres, 2005)

3.2.1.3 Yksikkötestaus

XP-menetelmässä testaus on erittäin merkittävässä roolissa. Jokainen ohjelmistolohko, jolla on mahdollisuus vikaantua (break), tulisi testata. Testien tulee olla automaattisia, ja niitä olisi suoritettava usein. Kaikki ohjelmisto tulisi kirjoittaa niin, että ensin kirjoitetaan testaus, sitten vasta varsinainen ohjelmisto. (Jeffries ym., 2001, s. 93-103; Beck & Andres, 2005, s. 97-102)

XP-kehityksessä on kehitystilanne usein sellainen, että samaa ohjelmistoa kehittää esimerkiksi viisi paria, eli kymmenen ohjelmistokehittäjää. Menetelmän perussääntönä on, että kaikki koodi kuuluu kaikille. Usein käy niin, että ohjelmoija tarvitsee jonkinlaista funktiota, ja sellainen on jo tehty, muttei ihan sellainen kuin ohjelmoija haluaisi. Tällöin ohjelmoijan tulisi muokata tätä toisen tekemää ohjelmakoodia mitään kysymättä, ja laajentaa sitä omiin tarpeisiin sopivaksi. (Jeffries ym., 2001, s. 93-103; Beck & Andres, 2005, s. 97-102)

Mikäli muutosten jälkeen alkuperäiset testit menevät läpi, ohjelmoija voi olla luottavainen siihen, ettei rikkonut toisen ohjelmoijan tekemää rakennetta. Näin toimitaan, sillä jos toisen tekemää ohjelmaa ei uskallettaisi muuttaa, tulisi useita lähes samankaltaisia koodirakenteita. Tämä ei käy päinsä, sillä XP-menetelmien tavoitteena on yksinkertainen koodi, jossa ei saa olla samoja asioita tehtynä useampaan kertaan. (Jeffries ym., 2001, s. 93-103; Beck & Andres, 2005, s. 97-102)

Ohjelmoijien tulisi tehdä kaikki testit niin kattaviksi, että jos varsinaista ohjelmakoodia muutetaan liikaa, niin testien tulisi hälyttää. Tällöin muutoksen tekijä voi luottaa koodin toimivuuteen, kunhan testit menevät lävitse. Toisaalta raskaita automaattisia testejä ei suositella. (Jeffries ym., 2001, s. 93-103; Beck & Andres, 2005, s. 97-102)

Testejä tehdään luottamuksen takia. Asiakkaan on kyettävä luottamaan ohjelmistoon. Projektin vetäjien on kyettävä luottamaan projektin etenemisraportteihin. Ohjelmoijien tulee pystyä itseensä sekä toisiinsa. Ohjelmistoviat nakertavat tätä luottamusta. (Beck & Andres, 2005, s. 97)

3.2.1.4 Jaettu ohjelmisto

Kaikki ohjelmisto on yhteistä. Tämä tarkoittaa sitä, että jos ohjelmoija näkee ohjelmakoodia, jonka voisi tehdä siistimmin tai paremmin, hänen velvollisuutensa olisi tehdä se. Kuka tahansa ohjelmakoodin olikin tehnyt, häneltä ei tarvitse pyytää lupaa, eikä häntä tarvitse tiedottaa muutoksista. (Beck & Andres, 2005, s. 66)

Kukaan yksittäinen ohjelmoija ei siis ole vastuussa henkilökohtaisesti mistään osasta koodia, ainoastaan tiimillä on kokonaisvastuu. On siis mahdollista, että ohjelmoija tekee huolimatonta ohjelmakoodia, ja jättää epäsiistin ohjelmakoodin toisten siivottavaksi. Näin voisi käyttää hyväksi sitä, ettei kukaan ole henkilökohtaisesti vastuussa mistään osasta koodia. (Beck & Andres, 2005, s. 66)

Tällaisten asioiden vuoksi XP-tiimissä tulisikin olla hyvä ”me”-henki, ja kunkin tiimin jäsenen tulisi ottaa tiimin vastuut vakavasti. Pariohjelmointi auttaa myös tähän ongelmaan. (Beck & Andres, 2005, s. 66)

XP-menetelmissä käytetään usein versionhallintajärjestelmää. Nykyisin hyviä versionhallintajärjestelmiä löytyy useita, joista monet ovat täysin ilmaisia. Versionhallintajärjestelmä on ehdottoman tärkeä varsinkin useamman ohjelmoijan projekteissa. Tällöin useampi kuin yksi ohjelmoija muokkaa samaa ohjelmaa. Kohdeyrityksessä käytetään TortoiseSVN versionhallintaohjelmaa, ja se on toiminut luotettavasti.

Versionhallintaohjelmistoa voi käyttää ohjelmiston jakamiseen ohjelmiston kehittäjien kesken. Uusin versio on aina ladattavissa versionhallintaohjelmasta omalle työkoneelle muokattavaksi. Muokkausten jälkeen ohjelmoija voi päivittää version uusimmaksi versioksi. Tällöin versionhallintaohjelma päivittää ainoastaan muutokset. Jos joku toinen työpari on ehtinyt päivittää muutoksia toisiin osiin ohjelmistoa, nämä muutokset säilyvät.

XP-menetelmissä versionhallintaohjelmaa tulisi hyödyntää niin, että uusin versio on aina sellainen, jossa kaikki testit läpäistyvät. Tällöin uusin versio on käytännössä aina mahdollista päivittää asiakkaalle testattavaksi. (Jeffries ym., 2001, s. 122)

Jaettua ohjelmistoa tulisi päivittää aina kun mahdollista. Ohjelmoija tekee muutoksia ohjelmaan, ja heti kun testit läpäistyvät muutokset tulisi päivittää versionhallintajärjestelmään muiden ohjelmoijien saataville. Näin tehdään, jotta muutosten ristiriitojen määrä voidaan minimoida, jos kaksi tiimiä on muokannut samaa ohjelmaosaa. Näin minimoidaan myös riski, että kaksi ohjelmoijaa tekisi samaa asiaa kovin pitkään tietämättä toisistaan. (Jeffries ym., 2001, s. 122-123; Beck & Andres, 2005, s. 66)

Ohjelmoijien tulisi päivittää ohjelmistoa versionhallintaan ainakin kahden tunnin välein. Jos ohjelmoija muokkaa ohjelmaa laaja-alaisesti, aikaa tulisi pienentää. (Beck & Andres, 2005, s. 66)

XP-menetelmä sopii parhaiten pieniin projekteihin, joissa on alle 30 kehittäjää. XP-menetelmä vaatii paljon kommunikointia kasvotusten, ja tämä onnistuu parhaiten pienissä kehitystiimeissä, jotka toimivat samassa yrityksessä. Tyypillinen suositeltu tiimin koko on 6 – 10 kehittäjää. (Tomayko & Hazzan, 2004, s. 22-23)

3.2.2 Scrum

Scrum on projektinhallintamenetelmä, joka luokitellaan ketteriin menetelmiin. Sana ”scrum” tulee rugby pelistä, ja viittaa ”scrummage”-pelitilanteeseen, jossa ratkaistaan, kenen hallintaan pallo päätyy. (Koch, 2004, s. 257)

Menetelmä on saanut paljon suosiota viime vuosina. Yksi suosion syistä voi olla menetelmän keveys ja selkeys. Menetelmä on yksinkertainen ja se on helppo selittää nopeasti. Usein ensinäkemältä vaikuttaa, että menetelmä tarjoaa ratkaisuja moniin ohjelmistotuotannon ongelmiin. (Haikala & Mikkonen, 2011, s. 38-40)

Perinteisten tuotekehitysmenetelmien heikkous on siinä, että ne erottavat asiakkaat ja kehittäjät toisistaan. Tämä aiheuttaa kuilua myös strategisilla sekä toiminnallisilla tasoilla. Tuloksena tiimi tietää mitä on tehtävä, muttei tiedä miksi. Taustatiedon tietämistä vaaditaan, jos halutaan saavuttaa innovatiivisia lähestymistapoja ongelmanratkaisuun. (Opelt ym., 2013, s.15)

Ohjelmistonkehittäjät perinteisissä menetelmissä ovat usein tietämättömiä keskipitkän ja pitkän tähtäimen liiketaloudellisista suunnitelmista projektissa. Scrum-menetelmässä sen sijaan kehittäjät ovat mukana strategisessa suunnittelussa. Tällöin kehittäjät alkavat ymmärtämään kuinka tärkeitä eri tehtävät ovat koko järjestelmässä. (Opelt ym., 2013, s.16)

Kohtuullisen yleinen harhakäsityksen mukaan Scrum olisi ohjelmistokehitystä ohjaava menetelmä, ja tällöin se antaisi ohjelmistokehittäjille rajattoman vapauden työssään. Scum kuitenkin menetelmänä keskittyy projektin ohjaukseen, tämän vuoksi menetelmä ei ota kantaa varsinaisiin ohjelmointitapoihin. (Opelt ym., 2013, s.12)

3.2.2.1 Scrum-roolit

Scrum on tunnettu projektihallinnan menetelmä, joka tarjoaa kehyksen projektin edistymisen hallinnalle. Alla on esitelty scrum-projektin eri roolien tarkoitusta projektissa. Scrum-menetelmää voidaan käyttää toimialasta riippumatta, mutta tässä työssä keskitytään asiaan ohjelmistoprojektin näkökulmasta. (Hänninen, 2010, s. 13)

Tuotekehitystiimi kehittää tuotteen. Tiimin optimaalinen koko on 7 henkilöä. Tiimillä on valta suorittaa tarpeelliset asiat, jotta haluttu lopputulos saavutetaan. Tämä on tehtävä yrityksen sääntöjä ja prosesseja noudattavalla tavalla. Tiimi itsessään hallinnoi omaa työkuormitustaan ja tätä kautta hyväksyy vastuun tuotteen laadusta. (Opelt ym., 2013, s.16; Hänninen, 2010, s. 14)

Tiimi jakaa itse pyrähdyksen aikana tehtävät keskenään. Tiimi myös sitoutuu pyrähdyksen alussa tuottamaan toimivan version pyrähdyksen loppuun mennessä,

tähän versioon tulisi olla tehtynä yhdessä sovitut toiminnot (tehtävät). (Koch, 2004, s. 259; Hänninen, 2010, s. 14)

Projektin omistaja (visionääri) ohjaa projektin kehittymistä, ja on vastuussa siitä, että tiimi kehittää oikeita asioita, halutussa järjestyksessä. Hän on vastuussa tuotteen taloudellisesta menestymisestä. Hän on mukana tiimin kehityksessä päivittäin, ja työstää projektin raportteja sekä jatkosuunnitelmia. (Opelt ym., 2013, s.16; Hänninen, 2010, s. 14)

Perinteisiin projektinohjausmenetelmiin verrattuna omistajaa voi verrata projektijohtajaan. Ero perinteiseen projektijohtajaan on se, ettei omistajan tarvitse ymmärtää syvällisesti asiakkaan tarkoituksia ja asiakasvaatimuksia. (Opelt ym., 2013, s.13)

Scrum-mestari (muutosagentti) auttaa tiimiä saavuttamaan tavoitteet. Hänen vastuullaan on, että kaikki vaikeudet ja esteet tiimin menestymisessä voidaan voittaa nopeasti. Hänellä ei ole oikeutta muuttaa ohjeistuksia, mutta hänen vastuullaan on seurata että scrum-menetelmää noudatetaan. Hänen tehtävänään on perehdyttää kaikki tiimin jäsenet ymmärtämään roolinsa. Hänellä on oikeus poistaa henkilö tiimistä. (Opelt ym., 2013, s.17)

Scrum-mestari seuraa, että scrumin periaatteita noudatetaan. Hän työskentelee johdon ja asiakkaan kanssa ja tunnistaa henkilön, josta tulee tuotteen omistaja (Product Owner). (Koch, 2004, s. 257)

Johtaja (toimittaja) tuottaa tietoa ja ohjeita organisaatioon. Tämä tuottaa scrum-mestarille, tuotteen omistajalle ja tiimille kehyksen, jossa he voivat liikkua. Johtaja usein ratkoo ongelmia, joita scrum-mestari on esittänyt. (Opelt ym., 2013, s.17)

Asiakas (rahoittaja) on projektin vaatija, joka tilaa tuotteen. Tyypillisesti asiakas on yrityksen ulkopuolinen toimija. Yrityksen sisäisessä projektissa asiakas on se, jonka on vastuussa projektin budjetista. (Opelt ym., 2013, s.17)

Loppukäyttäjä (expertti) on scrum-tiimin tiedolle erittäin tärkeä lähde. Hän on se, joka lopuksi tulee käyttämään ohjelmistoa. Tämän vuoksi tiimi lisää usein käyttäjän jollain tasolla projektin kehittämisen prosessiin. Pyrähdyksen suunnittelun aikana

loppukäyttäjä tekee yhteistyötä tuotteen omistajan kanssa vaatimusten määrityksen vuoksi. Myöhemmin hän työskentelee tiimin kanssa varmistuakseen sovelluksen käytettävyydestä. (Opelt ym., 2013, s.17)

3.2.2.2 Scrum-projektin käsitteet

Projektilista (Product backlog) on lista vaatimukista käyttäjätarinoiden muodossa kuten XP-menetelmässäkin. Käyttäjätarinat pilkotaan tehtäviksi. Tehtävät priorisoidaan ja listataan projektilistaan. Projektin omistajan tulisi järjestellä lista tärkeysjärjestykseen. Sopiva määrä tärkeimpiä tehtäviä otetaan projektin pyrähdykseen tehtäviksi. (Opelt ym., 2013, s.17; Koch, 2004, s. 258)

Päivittäiset scrum-palaverit (Daily Scrum) ovat noin 15 minuutin mittaisia pystypalavereja, jotka pidetään esimerkiksi tehtävätaulun edessä. Palaverissa on läsnä scrum-mestari sekä tiimi. Tässä palaverissa haetaan tiimin jäsenille tehtävät jokaiselle päivälle. Palaveria saa tulla kuuntelemaan myös muut jäsenet, mutta puhevuoro on vain tiimin jäsenillä sekä scrum-mestarilla (Koch, 2004, s. 259).

Jokainen vastaa myös kolmeen kysymykseen:

1. Mitä olet tehnyt edellisen palaverin jälkeen?
2. Mitä aiot tehdä seuraavaan palaveriin asti?
3. Onko tiedossasi asioita, jotka estävät/hidastavat sinua pääsemästä tavoitteeseesi?

(Opelt ym., 2013, s.18; Koch, 2004, s. 259)

Pyrähdyksen suunnittelupalaveri (Sprint Planning Meeting) pidetään jokaisen pyrähdyksen alussa, ja siihen osallistuvat kaikki asianosaiset. Tässä palaverissa valitaan tulevan pyrähdyksen tavoitteet. Palaverissa siis valitaan tehtävät tehtävälistasta. Pyrähdys (Sprint) on noin 30-päivän mittainen jakso, jossa pyritään suorittamaan osa projektia valmiiksi asti. (Koch, 2004, s. 259)

Pyrähdyksen päättyessä tiimillä on oltava sovituilta osin toimiva versio saatavilla. Siihen laitetaan paljon painoarvoa. Pyrähdysten keston katsotaan olevan mieluummin muutamia viikkoja kuin kuukausia. (Opelt ym., 2013, s.28)

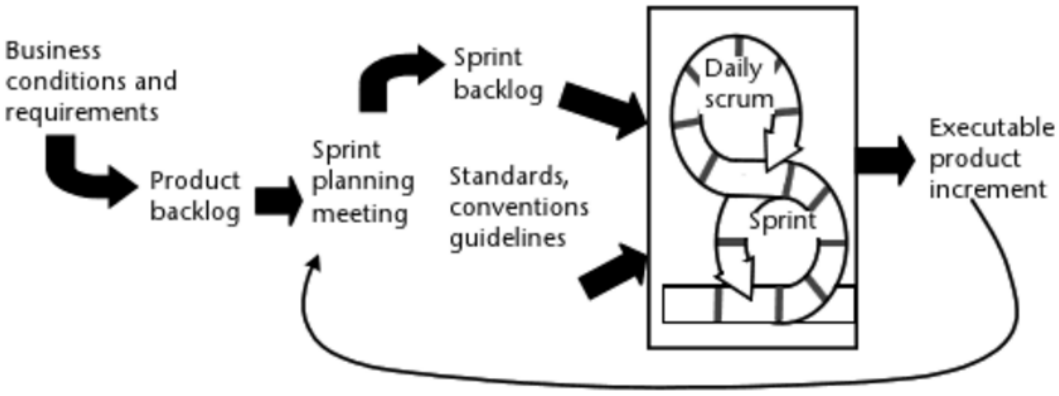
Projektin omistaja selittää käyttäjätarinat ja yhdessä määritellään päämäärät tulevalle pyrähdykselle. Tämän jälkeen tiimillä on tiedossa mitkä tehtävät kuuluvat kyseiseen pyrähdykseen. Pyrähdyksen katselmoinnissa (Sprint Retrospective) arvioidaan juuri päättynyttä pyrähdystä, ja pyritään löytämään parannuskeinoja seuraavaa pyrähdystä varten. Palaverin tulokset tallennetaan erilliseen listaan, joista muodostetaan ehdotuksia pyrähdyksen suunnittelupalaveriin. (Opelt ym., 2013, s.19)

Pyrähdyksen arviointi (Sprint Review Meeting) suoritetaan jokaisen pyrähdyksen päätteeksi, jossa tiimi esittelee tarinat, joiden ohjelmointi valmiina liitettäväksi seuraavaan versioon. Keskeneräisiä asioita ei esitellä tässä arvioinnissa lainkaan. (Koch, 2004, s. 260)

3.2.2.3 Scrum-projektin eteneminen

Kuvassa 12 on esitettu scrum-projektin eteneminen. Projekti alkaa ehtojen ja vaatimusten määrittelyllä.

Usein tiimin kehittäjälle projektin idea tulee asiakkaalta. Asiakas kehittää ajatusta niin kauan, kunnes tuotteesta on tuotevisio. Projektivisio sisältää tuotteen perusidean sekä tarvittavat rajoitukset, jotka on ajanhetkellä nähtävissä. (Opelt ym., 2013, s.17)



Kuva 12. Scrum-projektin eteneminen. (Koch, 2004, s. 258)

Yhteen pyrähdykseen kuuluu kaksi suunnittelupalaveria, päivittäinen scrum-pystypalaveri, tehtävien arviointipalaveri, pyrähdyksen loppuarviointipalaveri sekä pyrähdyksen katselmointi. (Opelt ym., 2013, s.18)

Toinen pyrähdyspalaveri on tiimin ja scrum-mestarin yhteinen palaveri, jossa on tarkoitus sopia tarkemmin menetelmistä, joiden avulla ensimmäisen palaverin tavoitteisiin päästään. Tämä palaveri järjestetään heti ensimmäisen jälkeen, esimerkiksi seuraavana päivänä. Palaverissa käydään läpi käytännönläheisesti asioista, joita tulee ottaa huomioon pyrähdyksen aikana ja asioiden tekemisjärjestyksessä. (Opelt ym., 2013, s.19)

Monissa tuotekehitystiimeissä tehtävät arvioidaan vaivannäön perusteella. Kuitenkin arviointi tulisi tapahtua tehtävän koon mukaan. On hyvin loogista, että projektin johtaja haluaa pitää kirjaa projektin tehtävien suorittamiseen vaadittavista ponnistuksista. Tämä on helposti muokattavissa tiedoksi projektin kustannuksista. (Opelt ym., 2013, s.20)

Ohjelmistotuotannossa on kuitenkin mahdollista, että tehokkaampi ohjelmoija on 25 kertaa tehokkaampi kuin heikompi yksilö. On myös mahdollista, että koodi ei koskaan valmistu heikommalla yksilöllä. Eri ohjelmoijien nopeudella ja ohjelmiston laadulla ei ole riippuvuussuhdetta. Arvioinnilla tarkoitetaan tässä menetelmässä koon arviointia, ei suorituskeston arviointia. (Opelt ym., 2013, s.20)

Scrumissa tiimin tehokkuutta mitataan nopeudella (velocity). Kuinka paljon toimintaa voidaan tuottaa aikayksikössä, tai pyrähdyksen aikana. Eli tehtävät määritellään koon mukaan ja tiimin suorituskyky lasketaan suoritettujen tehtävien (ja niiden kokojen) sekä nopeuden avulla. (Opelt ym., 2013, s.20)

Tehtäviä arvioidaan niin, että jokainen tiimin jäsen antaa tehtäville koon arviointinumeron. Numerot katsotaan läpi, ja poikkeavat näkemykset saavat perustella omaa kantaansa. Tämän jälkeen tehdään uusi kierros tehtävän koon arvioinnista, kunnes kompromissi on löytynyt. Tämä prosessi auttaa kaikkia tiimin jäseniä myös hahmottamaan, mitä tehtävät pitävät sisällään. Kokemus ja historia opettavat scrum-johtajalle paljonko tiimin arvioimat kokopisteet tarkoittavat keskimäärin työtunteina. (Opelt ym., 2013, s.23)

Scrum-menetelmä noudattaa luvussa 3.2 esiteltyjä ketterien menetelmien periaatteita.

3.3 Ohjelmistokehitysmenetelmien vertailu

3.3.1 Vertailu testikysymysten avulla

Kaikki yllä kuvatuista menetelmistä voivat olla tehokkaimpia menetelmiä jossain tietyssä projektissa. Eri projekteilla on erilaiset tarpeet, vaikka ne kaikki pitäisikin saada valmiiksi mahdollisimman nopeasti. Jotta voisimme valita optimaalisen kehitysmallin projektille, meidän tulisi vastata ensisijaisesti seuraaviin kysymyksiin. (McConnell, 2002, s.154)

1. Kuinka hyvin asiakas ja me tunnemme vaatimukset projektin alussa? Tuleeko ymmärryksemme todennäköisesti muuttumaan merkittävästi projektin edetessä?

Ketterät ohjelmointimenetelmät kehitettiin myös siksi, että ne toimisivat perinteisiä malleja paremmin projekteissa, joissa vaatimukset eivät ole täysin tiedossa projektin alkuvaiheessa. Tämä on tilanne useimmissa projekteissa.

2. Kuinka hyvin ymmärrämme järjestelmän arkkitehtuurin?

Käsitettä ”ohjelmiston arkkitehtuuri” ei ole aikaisemmin tässä työssä määritelty. Koskimies & Mikkonen (2005, s. 18-19) määrittelevät sen olevan järjestelmän perustuslaki. Arkkitehtuuri määrittelee järjestelmän ytimen, joka ei yleensä muutu kehityksen aikana. (Koskimies & Mikkonen, 2005, s. 18-19)

Arkkitehtuuri tarjoaa ohjelmistoprojekteille infrastruktuurin. Usein niin, että yrityksen eri projekteissa hyödynnetään samoja infrastruktuureita. Voidaan kuvailla, että arkkitehtuuri tarjoaa väylän esimerkiksi eri teknologioiden käyttöön nopeasti, mutta kuitenkin arkkitehtuuri asettaa rajoitteet sille, mitä sen avulla voidaan tehdä. (Koskimies & Mikkonen, 2005)

Arkkitehtuurien muuttaminen ja vaihtaminen on aina käyttämistä työläämpää. Tämän vuoksi pyritään aina valitsemaan arkkitehtuurit, jotka kykenevät tarjoamaan ohjelmistoprojektin tarpeet nykyisellään.

IEEE:n (2017) arkkitehtuurien kuvaamista koskeva standardi määrittelee ohjelmistoarkkitehtuurin järjestelmän perusorganisaatioksi, joka sisältää järjestelmän osat, niiden keskeiset suhteet ja niiden suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota. (IEEE, 2017)

Arkkitehtuuri siis ei koske ainoastaan ohjelmiston rakenteita, vaan myös ohjelman suorituksen aikana tapahtuvia rakenteita ja ohjelmiston yleistä käyttäytymistä. Arkkitehtuuri on siis tuttu, jos ohjelmiston kehitysympäristö sekä ohjelmiston kehitysmenetelmät ovat tuttuja. (Koskimies & Mikkonen, 2005, s. 18)

3. Kuinka paljon luotettavuutta tarvitsemme?

Kertoo, kuinka monta virhettä tuotteessa on käyttöönottovaiheessa, kun on käytetty kyseistä ohjelmistokehitysmenetelmää. (McConnell, 2002, s. 158)

4. Tuottaako järjestelmän, jossa on hyvin kasvunvaraa?

Tämä kuvaa, kuinka helposti ohjelmistoa voidaan muuttaa monipuolisuuden ja laajuuden suhteen projektin aikana. Tähän kuuluvat myös sellaiset muutokset, joita ei osattu odottaa. (McConnell, 2002, s. 158)

5. Kuinka paljon riskiä tähän projektiin sisältyy?

Tämä kysymys vastaa siihen, kuinka hyvin ohjelmiston kehitysmallilla voidaan tunnistaa ja hallita projektiin liittyviä riskejä. (McConnell, 2002, s. 158)

6. Olemmeko rajoitettu ennalta määrättyyn aikatauluun?

Tällä kysymyksellä haetaan vastausta siihen, kuinka hyvin ohjelmistokehitysmenetelmä soveltuu siihen, että ohjelmisto toimitetaan ennalta sovittuna ajankohtana. (McConnell, 2002, s. 158)

7. Paljonko ohjelmistokehitysmenetelmä saa tuoda projektiin lisätyötä?

Kuinka raskas ohjelmistokehitysmenetelmä on? Paljonko hallinnollista ja teknistä lisätyötä tarvitaan, jotta menetelmää voidaan tehokkaasti käyttää? Lisätyöksi lasketaan projektin suunnittelu ja seuranta, dokumentointi sekä kaikki toiminnot, jotka eivät liity itse ohjelmiston kehittämiseen. (McConnell, 2002, s. 158)

8. Täytyykö meidän pystyä tekemään kesken kaiken suunnanmuutoksia?

”Kuinka helposti tuotteen merkittäviä piirteitä voidaan muuttaa aikataulun puolivälissä. Tähän ei sisälly tuotteen perustarkoituksen muuttaminen, mutta sen merkittävä laajentaminen sisältyy.” (McConnell, 2002, s. 158)

9. Onko meidän saatava asiakkaille näkyviä edistyksen merkkejä projektin aikana?

Mallien läpinäkyvyydet asiakkaalle poikkeavat runsaasti toisistaan. Mikäli asiakkaan olisi tärkeää päästä näkemään projektin kehittymistä, kannattaa valita kehitysmenetelmä, joka automaattisesti tarjoaa edistymisen merkkejä asiakkaalle. (McConnell, 2002)

10. Onko meidän saatava hallinnolle näkyviä edistyksen merkkejä projektin aikana?

Kuinka hyvin malli tarjoaa automaattisesti menetelmiä, joiden avulla hallinto voi seurata projektin edistymistä. (McConnell, 2002, s. 158)

11. Kuinka paljon erikoistumista tarvitsemme käyttääksemme tätä elinkaarimallia onnistuneesti?

”Viittaa koulutuksen ja kokemuksen tasoon, joka vaaditaan mallin onnistuneeseen käyttöön. Tähän kuuluu erikoistumistaso, joka vaaditaan edistymisen seuraamiseksi mallin avulla, mallin sisältyvien riskien välttämiseen, ajan haaskaamisen välttämiseen ja mallin etujen hyödyntämiseen, joiden takia malli alun perin on valittu.” (McConnell, 2002, s. 158)

McConnell (2002) esittää, että kun tiedämme projektin luonteesta riittävästi, voimme vastata näihin kysymyksiin käyttämällä karkeaa arviointitapaa kuvastamaan ominaisuuden tarpeellisuutta kyseisessä projektissa. Vaihtoehdot ovat: *Heikko*, *Kohtuullinen*, *Erinomainen*. (McConnell, 2002, s. 156)

McConnell (2002) mukaan tarkempi arviointi ei projektin alkuvaiheessa olisi mielekästä. Arvioinnin jälkeen tuloksia voidaan verrata eri toimintamallien tarjoamiin ominaisuuksiin, ja pyrkiä näin löytämään paras mahdollinen toimintamalli projektikohtaisesti. (McConnell, 2002, s. 157) Kysymyksiin haetaan vastauksia luvussa 6.1.

Tomayko & Hazzan (2004) pitää ihmisten näkökulmasta katsottuna tärkeimpänä ominaisuutena kommunikointia, kun valitaan ohjelmistokehitysmenetelmää. Ketterissä menetelmissä kommunikoidaan kasvotusten enemmän kuin muissa menetelmissä. Tämä tulisi huomioida menetelmää valittaessa. Pienet projektit, joissa kehittäjät ovat lähellä toisiaan, ovat otollisia projekteja ketterille menetelmille, joissa vaaditaan paljon kommunikointia kasvotusten. (Tomayko & Hazzan, 2004, s. 22-23)

Ohjelmistokehitysmenetelmän valinta riippuu siis täysin projektista. Jotta valinta perustuisi järkeviin valintoihin, on ymmärrettävä projektin tausta ja luonne. Seuraavissa luvuissa tutustutaan kohdeprojektin taustoihin ja projektin tärkeimpiin kohtiin.

Tavoitteena on löytää pienmerkintälaitteen ohjelmiston kehittämiseen optimaalisesti sopivat menetelmät ja käytännöt. Yrityksen tuotekehityksen projektit ovat

usein myös samankaltaisia, jolloin hyväksi havaittuja käytäntöjä voitaisiin harjoittaa myös muissa projekteissa tulevaisuudessa.

Näihin kysymyksiin perehtymällä pyritään tässä työssä löytämään sopivimmat kehitysmenetelmät yrityksen ohjelmiston tuotantoon. Sopivimpiin menetelmiin kohdennetaan vielä lopuksi SWOT-analyysi, jossa parhaita ohjelmistokehitysmenetelmiä tutkitaan vielä tarkemmin.

3.3.2 SWOT-analyysi

SWOT-lyhenne tulee englannin kielen sanoista: Strengths (Vahvuudet), Weaknesses (Heikkoudet), Opportunities (Mahdollisuudet), sekä Threats (Uhat). SWOT-analyysissä (kuva 13) eritellään vahvuudet, heikkoudet, mahdollisuudet ja uhat omiin lohkoihinsa. Menetelmää käytetään moniin eri tarkoituksiin, kuten strategian laatimiseen. (Lindroos & Lohivesi, 2010, s. 219; Ambrosini & Johnson & Scholes, 1998, s. 122-123)

Menetelmässä pyritään löytämään kuhunkin kategoriaan useita eri asioita eri näkökulmista, jotka sitten analysoidaan. Menetelmää käytetään usein ryhmätyönä, jolloin eri näkemykset pääsevät hyvin esiin myöhempää analysointia varten. Toinen henkilö voi hyvinkin nähdä saman asian uhkana, jonka toinen näkee mahdollisuutena. (Ambrosini ym., 1998, s. 122-133)

Menetelmää voi käyttää moniin yrityksen analysointia kaipaaviin kohteisiin, tuotteiden tai palveluiden asemaan, kilpailijoihin, vaihtoehtoihin sekä omiin toimintamalleihin ja kilpailukykyyn. Menetelmä on suosittu ja suoraviivainen. (Lindroos & Lohivesi, 2010, s. 219-220; Ambrosini ym., 1998, s. 122)

Kuva 13. SWOT-analyysialusta (Lindroos & Lohivesi, 2010, s. 220)

Analyysin pohjalta voidaan tehdä päätöksiä esimerkiksi siitä, miten vahvuuksia voidaan käyttää parhaiten hyväksi, miten heikkoudet voitaisiin kääntää vahvuuksiksi, miten mahdollisuuksia voitaisiin hyödyntää, ja kuinka uhkia voitaisiin välttää. (Lindroos & Lohivesi, 2010, s. 220)

SWOT-analyysia tehtäessä ei kannata sekoittaa nykyhetkeä ja tulevaisuutta, vaan tarvittaessa tulisi tehdä kaksi SWOT-ruudukkoa rinnakkain, jossa toisessa pohditaan nykyhetkeä, toisessa tulevaisuutta. Näin vältetään turhaa sekavuutta, koska usein nykyhetken ja tulevaisuuden tärkeimmät asiat poikkeavat toisistaan. (Lindroos & Lohivesi, 2010, s. 219)

SWOT–menetelmä jää usein puutteelliseksi siitä syystä, että näkökulma muutoksiin on usein negatiivinen. Tällöin monet asiat nähdään mieluummin uhkina kuin mahdollisuuksina. (Ambrosini ym., 1998, s. 122-133)

4 CASE: PIENMERKINTÄLAITTEEN TUOTEKEHITYS: TARVEKARTOITUS

4.1 Pienmerkinnät nykyisin

Pohjoismaiden tienmerkintäurakoissa viime vuosien trendinä on ollut, että turvallisuussäädökset kovenevat vähitellen. TMA-törmäysvaimenninta (kuva 14) vaaditaan yhä useammissa urakoissa. Törmäysvaimennin painaa niin paljon, ettei sitä voi käyttää kuin vain kokonaismassaltaan kuorma-autoa vastaavissa ajoneuvoissa. Massaa tarvitaan lähes 10 000 kg.

Kuva 14. TMA-törmäysvaimennin (RMD truck – Tieto-Oskari, Youtube)

Suomessa on käytössä säädös, jonka mukaan useilla teillä vaaditaan käyttämään törmäysvaimenninta. Vaatimus perustuu nopeusrajoitusalueeseen, sekä vuoro-

kausittaisiin liikennemääriin kyseisellä tieosuudella. Lisäksi rinnakkaisten kaistojen lukumäärä vaikuttaa asiaan. (Liikennevirasto, 2013) On todennäköistä, että tulevaisuudessa turvallisuussäädökset tulevat kiristymään myös jatkossa.

TMA-törmäysvaimentimen käyttö tiemerkinnöissä tarkoittaa käytännössä sitä, että TMA on oltava asennettuna ensimmäisessä ajoneuvossa liikenteen kulkusuunnassa. Tätä ajoneuvoa kutsutaan suoja-autoksi.

TMA voi olla kiinnitettynä myös suoraan tiemerkintöjä tekevään työkoneeseen, jos työkone on vain sopiva sen vetämiseen.

Nykyisistä menetelmistä tiedusteltiin alan urakoitsijalta, ja selvisi, että nykyisin pienmerkintöjä tehdään kuorma-autolla. Kuumamassalla täytetyin pannuin kuorma-auto ajaa pienmerkittävän kohdan luokse. Työntekijät jalkautuvat ja ottavat käyttöön pienet työnnettävät kärryt, joihin laitetaan pieni määrä kuumamassaa, jolla esimerkiksi suojatie tehdään. Kärryihin voidaan laittaa myös muuta pienmerkintöjen tekoon käytettyä materiaalia kuten hiekkaa tai lasihelmeä. (Puhelinhaastattelu 23.12.2016)

Nämä kuorma-autot on suunniteltu niin, että kärryjä täytetään kuorma-auton takaosasta. Muutkin toiminnot pienmerkinnän tekohetkellä vaativat pääsyä kuorma-auton takaosaan useita kertoja työn aikana. (Puhelinhaastattelu 23.12.2016)

Tämä estää TMA-törmäyssuojan käytön nykyisessä ajoneuvossa. Ajoneuvo tulisi suunnitella eritavoin toimivaksi, jotta törmäyssuojan käyttö olisi mahdollista suoraan työkoneessa. Nykyiset pienmerkintöjä tekevät ajoneuvot ovat myös huomattavan takapainoisia, sillä niiden kuumamassasäiliöt sijaitsevat auton takaosassa. TMA painaa jopa 800 kilogrammaa ja ulottuu pitkälle auton taakse ilman tukirattaita. TMA:n käyttö aiheuttaisi huomattavan takapainoisuuden pienmerkintäautoihin. (Puhelinhaastattelu 23.12.2016)

Edellisistä syistä johtuen pienmerkintöjä tehtäessä tarvitaan erillinen TMA-törmäysvaimentimella varustettu suoja-auto varsinaisen työkoneen takapuolelle. Kaiken lisäksi varsinaisessa työssä urakoitsijoiden on jalkauduttava tekemään työ käsikäyttöisillä työkaluilla.

”Tiellä tehtävät työt on luokiteltu työturvallisuuslainsäädännössä töiksi, joihin liittyy erityisiä vaaroja työntekijöiden turvallisuudelle tai terveydelle. Tiellä työtä tekevät joutuvat jatkuvasti alttiiksi liikenteen aiheuttamille vaaroille ja aiheuttavat samalla vaaraa yleiselle liikenteelle. Liikenteen aiheuttamien vaarojen lisäksi tiellä tehtävään työhön liittyy monenlaisia työturvallisuuteen liittyviä riskejä.” (Liikennevirasto 2014)

4.2 Kilpailevat laitteistot

Pienmerkintään on tarjolla kilpailevia laitteita hyvin niukasti. Ruotsalainen RME-yritys on tehnyt yhden laitteen, jolla pystytään tekemään pienmerkintöjä (kuva 15).

Kuva 15. Tienmaalausrobotti (Lähde. Youtube, RME Channel)

Laite osaa maalata ohjelmallisesti pienmerkintöjä teiden pintaan, mutta siinä on urakoitsijoiden mukaan joitakin huonoja puolia.

Laite maalaa kuviota kuumamassalla ja sirottelee lasihelmeä suuttimista. Jos laitteella tehtäisiin suora viiva, laatu olisi hyvä. Massa tulee ensin tien pintaan, ja lasihelmi massan päälle niin kuin kuuluukin. Kuvioita tehdessä maalaava kärki kuitenkin pyörii piirron aikana. Tämä johtaa siihen, ettei lasihelmi osu massan päälle kaikissa tilanteissa. Lasihelmeä tulee runsaasti tienpintaan myös sille osuudelle,

johon kuumamassaa tulee kuvion myöhemmässä vaiheessa. Kun massaa laitetaan lasihelmen päälle, massa ei tartu tien pintaan kovin hyvin. (Puhelinhaastattelu, 23.12.2016)

Laite on myös todella hidas, kuvion tekemiseen menee noin kymmenen minuuttia. Ajoneuvoon on asennettava myös tukijalat maalauksen ajaksi, ja laite ei ulotu tekemään useita kuvioita yhdestä paikasta. Laitteiston siirtäminen tukijalkojen kanssa on työlästä. (Puhelinhaastattelu, 23.12.2016)

Normaalit pienmerkinnät tehdään kuumamassasta, jonka päälle ripotellaan lasihelmeä, joka antaa kuviolle paluuheijastuvuuden. Suojateillä lasihelmeä ei käytetä, vaan suojatie koostuu kuumamassasta ja lisätystä hiekasta. Kuumamassalla ja lasihelmellä tehty suojatie olisi liian liukas, eikä täytä Suomessa suojatien laatuvaatimusta. Tämän vuoksi laitteella ei pystytä tekemään suojateitä. Laitteita on tehty vain yksi kappale Ruotsissa, eivätkä urakoitsijat pidä todennäköisenä, että nämä laitteistot yleistyisivät Suomessa. (Puhelinhaastattelu, 23.12.2016)

Norjalainen yritys nimeltä Trysil Maskin valmisti yhden pienmerkintöjä tekevän tulostimen vuonna 2012. Toisen laitteen yritys valmisti vuonna 2015. Laitteet ovat käytössä pohjoismaissa, mutta Suomeen laitteet eivät vielä ole rantautuneet. Laitteen toimintaperiaate on erilainen kuin kilpailevassa RME-yrityksen pienmerkintätulostimessa. (Puhelinhaastattelu, 23.12.2016)

Laite asennetaan kuorma-auton takaosaan, jossa kuvio tulostetaan ajoneuvon liikkuessa eteenpäin (kuva 16). Laitteen hyvä puoli on se, että tulostus tapahtuu nopeasti koko tulostimen leveydeltä. Kuumamassaa lasketaan tulostuksen mukaan, jonka jälkeen laitteisto sirottelee lasihelmen massan päälle. Tällöin lasihelmeä ei tule tienpinnalle ennen kuumamassaa, ja toisaalta massaa ei jää ilman lasihelmi päällystettä. (Puhelinhaastattelu, 23.12.2016)

Kuva 16. Trysil Maskin Trafficprinter. (Youtube, Trysil Maskin channel)

Laitteisto kuitenkin tulostaa kuvion tipoittain, jolloin kuvion väleihin jää massatonta pinta-alaa. Kuvasta 16 näemme, ettei tulostunut hirven kuva ole täysin valkoinen, vaan tulostusjälki on rakeinen. Tämä automaattisesti tarkoittaa sitä, ettei laitteella voida tehdä suojateitä. Laite on myös huomattavasti kapeampi kuin tulostava ajoneuvo, jolloin leveämmät kuviot tulee tehdä osissa.

Kilpailevia laitteistoja, jotka automaattisesti tekevät pienmerkintöjä teiden pintaan on vain muutamia. Jokaisella laitteistolla on suuria puutteita Suomen urakoitsijoiden tarpeisiin.

4.3 Asiakasvaatimukset

Pienmerkintöjen tekeminen nykyisillä menetelmillä on yhä vaikeampaa ja tehottomampaa kiristyvien säädösten vuoksi. Erillisen törmäysvaimentimella varustetun suoja-auton käyttäminen on yhä useammin tarpeen, eikä vaihtoehtoja ole. Urakoitsijoiden on pakko jalkautua liikenteen sekaan tehdäkseen pienmerkintöjä, eikä tämä edistä työturvallisuutta eikä työtehokkuutta.

Yrityksessämme halutaan täyttää tämä asiakastarve, ja tehdä asiakkaita tyydyttävä pienmerkintätulostin Suomen olosuhteisiin. Tienmerkintä alan yritys on tilannut kumppaniltamme pienmerkintöjä tulostavan laitteiston. Yrityskumppanimme on vastuussa laitteiston mekaanisesta toteutuksesta. Meidän tehtävämme on luoda laitteelle elektroninen toimintalogiikka sekä auttaa mekaniikan suunnittelussa. Ensisijainen tehtävämme on tuotekehityksessä pyrkiä täyttämään tämän asiakkaan vaatimukset. Laitteistovaatimuksia on tässä työssä pyritty kartoittamaan yritysasiakkaan toimitusjohtajalta, sekä yrityksen työntekijöiltä.

Asiakastarpeiden kartoitukseen on järkevää panostaa, sillä koko tuotekehitysprojekti voi epäonnistua, jos kaikki tärkeimmät asiakasvaatimukset eivät ole tiedossa laitteen suunnitteluvaiheessa. Mitä pidemmälle laitteen suunnittelu etenee, sitä vaikeampaa on tehdä suuria muutoksia laitteiston toimintaan. Se, kuinka hyvin asiakastarpeet ovat kartoitettavissa ennen projektin alkua, vaikuttaa optimaalisen ohjelmistokehitysmenetelmän valintaan merkittävästi.

4.3.1 Laitteen käyttökohteet

Asiakas haluaa laitteiston, jolla voidaan tehdä suojateitä ja muita pienmerkintöjä. Tämä tarkoittaa sitä, että laitteiston tulee kyetä tekemään normaalit pienmerkintäkuviot kuumamassasta ja lasihelmestä, sekä suojatiet kuumamassasta ja lisätystä hiekasta. Tätä vaatimusta varten mekaniikasta vastaava kumppanimme on suunnitellut suutinjärjestelmän, jossa voidaan samoilla suuttimilla syöttää kuumamassan päälle joko hiekkaa, tai lasihelmeä. Suomessa käytetyissä pienmerkinnöissä ei koskaan tarvita yhtä aikaa lisättyä hiekkaa ja lasihelmeä. Optimaalinen käyttötapa hiekalle ja lasihelmelle on sirotella aine kuumamassan päälle välittömästi massan tienpintaan laskemisen jälkeen.

Tuotekehityksessä tulee ottaa huomioon poikkeaako lasihelmen ja hiekan määrä toisistaan optimaalisessa työn jäljessä, ja täten suunnitella oikea syöttöteho molemmille lisäaineille.

Jotta välttyisimme RME-yrityksen valmistaman laitteiston huonolta puolelta, lasihelmen suuttimet tulee sijoittaa heti kuumamassaluukkujen jälkeen, jotta lasihelmeä, tai hiekkaa, ei tulisi tienpintaan ennen kuumamassaa.

Laitteistolla halutaan tehdä normaalikokoiset pienmerkinnät yhdellä tulostuskerralla. Suomessa käytetyt pienmerkinnät ovat alle 2,5 metriä leveitä ja alle 7,5 metriä pitkiä. (Liikennevirasto, 2015, s. 39 - 66)

Kuorma-auto on 2,6 metriä leveä. Jos tällaisen ajoneuvon takaosassa olisi koko ajoneuvon levyinen tulostin, sillä voisi tehdä Suomessa käytetyt pienmerkinnät yhdellä ajokerralla. Tällöin tulostus tapahtuisi koko ajoneuvon leveydeltä samanaikaisesti liikkuipa ajoneuvo eteen-, tai taaksepäin. Jos tulostin asennetaan kuorma-auton takaosaan, tuotekehityksessä on otettava huomioon, että työkoneen takaosassa on kyettävä käyttämään samanaikaisesti myös TMA törmäysvaimenninta, jotta erillistä suoja-autoa ei tarvita. (Sähköpostikysely 18.11.2016)

Asiakas haluaisi, että laitteistoon olisi esiladattu kaikki Suomessa käytetyt pienmerkinnät oikeassa mittakaavassaan. Saimme pienmerkinnät CAD-kuvina sähköpostikyselyn avulla, ja kuvat sisältävät oikean mittakaavan. Suomessa käytetään usein kahden kokoisia kuvioita tieliikenteessä, pienemmät alle 60 km/h nopeusrajoitusalueella, ja suuremmat kuviot vastaavasti yli 60 km/h nopeusrajoitusalueella.

4.3.2 Turvallisuus

Sekä kuljettajan että muiden tien käyttäjien turvallisuuden kannalta on parasta, että työkoneeseen voidaan asentaa törmäysvaimennin. Tällöin yksi urakoitsija yhdellä ajoneuvolla voisi suorittaa pienmerkintöjen maalaamista.

Työturvallisuus parantuu huomattavasti, jos tiemerkintää tehdessään työntekijän ei tarvitse poistua kuorma-auton hytistä lainkaan. Tällöin työturvallisuus pienmerkinnöissä ei poikkea millään tavalla normaalista tienmerkintöjen tekemisestä.

Tuotekehityksessä on huomioitava, että laitteen saattaminen kuljetustilasta käyttötilaan on kyettävä suorittamaan hallintalaitteilla ajoneuvon hytistä. Turvallisuu-

den nimissä meidän on asetettava tavoite korkealle ja luotava sellainen järjestelmä, ettei kuljettajan tarvitse minkään syyn takia käydä fyysisesti tulostimen luona ennen kuvion maalaamista.

Mikäli saamme järjestelmästä helppokäyttöisen ja tehokkaan, turvallisuus paranee jo sillä perusteella, ettei kuljettajan tarvitse ajallisesti viettää yhdessä kohteessa ylimääräistä aikaa. Helppokäyttöisyys antaisi kuljettajalle myös stressittömämmän työkokemuksen ja näin kuljettajalla jäisi enemmän huomiokykyä myös muulle liikenteelle.

Voimme varmasti kaikki jollain tasolla samaistua siihen visioon, että kuorma-auton siirtäminen 3 senttimetriä vasemmalle säilyttäen ajoneuvon tien suuntaisuuden voisi olla hyvin turhauttava kokemus ilman kunnollisia havainnointiin tarkoitettuja laitteita. Ilman kunnollisia havainnointiin tarkoitettuja laitteita kuljettajan olisi myös mahdotonta tiedostaa, että oikea siirtymän tarve oli esimerkissämme 3 senttimetriä.

Tuotekehityksessä on huomioitava hyvin tarkkaan, että kohdistimia on laitteistossa oltava useita. Kameroita, lasertähtäimiä tai viivalasereita voisi hyvinkin käyttää. Kohdistamista pohdimme työssä tarkemmin myöhemmin, mutta myös turvallisuuden näkökulmasta asia on tärkeä. Jatkuva tarpeeton turhautuminen liikenteessä ei ole turvallisuuden kannalta kestävä vaihtoehto.

Normaali tiemerkintäjärjestelmämme sisältää ajoneuvon takaosassa suuren leditaulun, jonka avulla on helppo ohjeistaa muita autoilijoita asettamalla taululle tilanteeseen sopivat liikennemerkit ja tekstiselitteet. Tilanteet vaihtuvat nopeasti, ja kuskin on sekä oman että muiden autoilijoiden turvallisuuden kannalta järkevää kyetä vaihtamaan leditaulussa olevaa kuvaa nopeasti ajoneuvon hytistä. Samaa leditaulujärjestelmää suosittelen asennettavaksi myös pienmerkintäajoneuvoihin.

Tiehallinnossa on pohdittu kunkin pienmerkintäkuvion oikeat koot senttimetrin tarkkuudella vallitseviin ajonopeuksiin suhteutettuna. Epäilemättä jokainen kuvio on myös muodollisesti tarkoin harkittu edistämään liikenteen sujuvuutta ja turvallisuutta. Meidän on tehtävä laitteisto, joka annetuista virallisista pienmerkintöjen CAD-kuvista maalataan mahdollisimman tarkasti tienpintaan alkuperäisessä muodossa ja koossaan.

Käsin tehdyissä pienmerkinnöissä voi olla poikkeuksia ja poikkeamat lisäävät aina vähintään hieman väärintulkitsemisen riskiä. Pienetkin mahdollisuudet kuvion väärintulkintaan heikentävät tienkäyttäjien turvallisuutta. Automaattisella pienmerkintätulostimella voisi tulostaa kuvion juuri sellaisena kuin se on syötetty ohjelmaan ja tulostuksesta tulisi joka kerta samanlainen. Parantaisi yleistä turvallisuutta, jos kuviot olisivat juuri sellaisia kuin ne ovat suunniteltu ja keskenään samanlaisia.

Pohjoismaat ovat jo pitkään olleet maailman huippuja tieliikenteeseen liittyvän työturvallisuuden tason ylläpitämisessä. Ei ole kovinkaan harvinaista, että uusia turvallisuutta lisääviä määräyksiä tulee käyttöön vuosittain. Hyväksi havaittuja määräyksiä kopioidaan usein myös muissa pohjoismaissa.

Kysyin tienmerkintöjä tekevän yrityksen toimitusjohtajalta, uskooko hän Suomen kieltävän lähitulevaisuudessa pienmerkintöjen tekemisen käsikäyttöisillä laitteilla ja painottuvan koneellisiin tuotantoratkaisuihin. Vastauksessaan toimitusjohtaja uskoi, että määräykset tulevat muuttumaan Ruotsin mallin suuntaisiksi, jossa tällaista muutosta pyritään ajamaan läpi. (Sähköpostikysely 18.11.2016).

4.3.3 Laitteen kohdistaminen

On hyvin yleistä kaupunkialueella, että näemme liikennevaloristeyksessä suuntanuolia, jotka ovat kuluneet vuosien varrella. Pienmerkinnöistä vastaavat urakoitsijat joutuvat usein korjaamaan merkintöjä. Kuvassa 17 näemme esimerkin erittäin kuluneista pienmerkinnöistä, jotka tulisi ehdottomasti korjata.

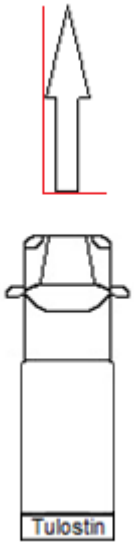
Kuva 17. Kuluneet pienmerkinnät (Tiehallinto, 2004)

Asiakas vaatii, että laitteella pystyy tulostamaan pienmerkintöjä myös vanhojen merkintöjen päälle. Asiakkaan mukaan kohtuullisen suuri osa laitteen käytöstä on vanhojen merkintöjen korjausta.

Suomen lain (39 b §. (SK: 164/1992)) mukaan tiemerkinnät tulee pitää sellaisessa kunnossa, ettei sitä voida erheellisesti käsittää muuksi tiemerkinnäksi. Tämän vaatimuksen pohjalta tehtyjä pienmerkintöjä joudutaan korjaamaan mm. talvikunnossapidon aiheuttaman kulumisen vuoksi.

Merkintöjen korjaaminen asettaa laitteistolle tiukan käyttövaatimuksen laitteen kohdistuksessa. Kuinka kuljettaja voisi maalata saman kuvion kaksi kertaa päällekkäin hallitusti? Suunnitelmien mukaan pienmerkintätulostin olisi ajoneuvon levyinen, ja sijaitsisi ajoneuvon peräosassa. Kuorma-auton ollessa noin 12 metriä pitkä, kuljettajan on hyvin vaikea ajaa tulostin juuri vanhan kuvion päälle tulostuksen toistamista varten. Kuljettajan tulisi kyetä kohdistamaan laite kohtuullisen helposti muutaman senttimetrin tarkkuudella.

Yksi ratkaisu tähtäinongelmaan voisi olla viivalasereista koostuva tähtäin ajoneuvon etuosassa (kuva 18). Kuvan mukaisessa ratkaisuehdotuksessa kohde olisi hyvin valaistu, ja selkeästi kuljettajan näkyvillä kaikissa tilanteissa.



Kuva 18. Nuolen uudelleen tulostaminen.

Kuvassa 18 on esitetty esimerkki kohdistimesta ajoneuvon etupuolella. Kuljettajan tulisi ajaa ajoneuvo kohtisuoraan korjattavan pienmerkinnän eteen ja säätää viivalasereista koostuva tähtäin leveyssuunnassa kuvion vasempaan laitaan. Lasertähtäintä voisi olla tarpeen säätää myös pituussuunnassa, sillä ajoneuvon jouset elävät ajoneuvon jarruttaessa ja liikkuessa.

Aikaisempiin tiemerkintäautoihin olemme tehneet lasertähtäimen ajoneuvon katolle osoittamaan paikkaa ajoneuvon etuosassa. Tässä tapauksessa kyse on pistemäisestä tähtäimestä, jonka avulla kuljettajat ovat saaneet ajoneuvoa kohdistettua sivusuunnassa oikealle kohdalle. Kuljettaja on voinut säätää lasertähtäintä ajoneuvossa olevan tietokoneen kautta tarkasti.

Lasertähtäimen ajoon olemme käyttäneet laadukkaita ja tarkkoja askelmoottoreita, jolloin tähtäimen asento on ohjelmiston tiedossa kaikissa tilanteissa. Jos pistemäisen lasertähtäimen korvaisi viivalasereilla kuvan 18 mukaisesti, kuljettaja voisi hyvin helposti osoittaa ohjelmistolle paikan johon tulostus tulee suorittaa. Merkitsemisen jälkeen kuljettajan tarvitsisi vain ajaa eteenpäin ja ohjelmisto suorittaisi tulostuksen oikeassa kohdassa. Alkukalibroinnin jälkeen ohjelmisto tietäisi kyllä lasersäteen ja tulostimen välisen etäisyyden tarkasti. Ajoneuvoon etenemisen mittaamiseen tarvitsemme tiedon ohjelmistolle. Aiemmissa maalikoneissa mittaus tehdään akselille asennettavan hammasrattaan ja tätä mittaavan pulssianturin avulla. Menetelmä on toiminut hyvin ja sitä voisi hyvinkin käyttää myös näissä ajoneuvoissa.

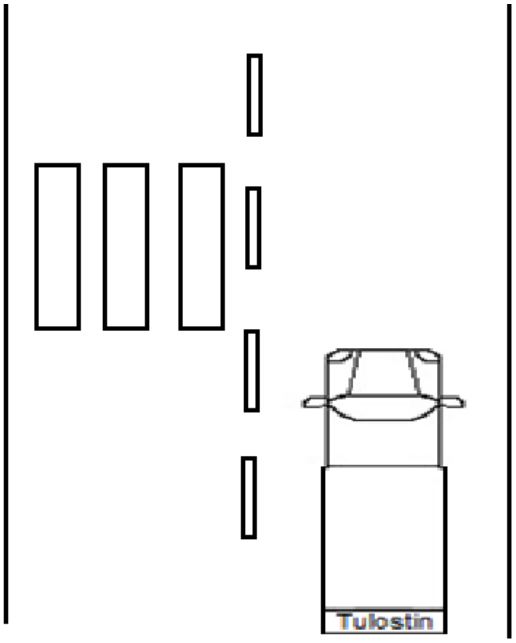
Kyseinen kohdistamismenetelmä ei ole saanut laajalti kannatusta tuotekehitystiimissämme. Yleinen käsitys on, että tähtäin olisi liian kaukana. Merkitsemisen jälkeen kuljettajan tulisi ajaa yli 12 metriä kohtisuoraan eteenpäin, pienikin mutka matkan aikana voi johtaa useiden senttimetrien virheeseen tulostuksessa. Samaa mieltä olivat urakoitsijat, joiden mielipidettä kysyin asiasta.

Ajoneuvon etupuolella kohde olisi myös hyvin valaistu ajovalojen ansiosta. Runsasliikenteisissä tieosissa urakoitsijat saavat usein työskennellä vain 21:00 - 06:00 välisenä aikana, jolloin työkohteen valaistus olisi tarpeen. Vaatimus tulee työn tilaajalta ja on yleisesti käytössä esimerkiksi Uudellamaalla.

Jos menetelmä olisi sama kuin kuvassa 18, mutta kohdistin olisi lähempänä tulostinta, olisi sen oltava ajoneuvon alapuolella tulostimen edessä. Kuljettaja ei näe auton alapuolelle suoraan hytistä, joten menetelmä tarvitsee viivalaserien lisäksi useita kameroita, ja valoja, ajoneuvon alapuolelle.

Tulostimen ja kohdistimen välisen välimatkan pieneneminen auttaisi minimoimaan ajovirheet suorituksen aikana, mutta menetelmä tuo myös monimutkaisuutta järjestelmään.

Kameroiden linssit tulevat kohtaamaan paljon tiepölyä auton alla. Kameroita tulisi kenties olla useita, jotta koko tulostimen leveys (vähintään 2,6 metriä) olisi kameroiden kautta nähtävissä. Ongelmaksi kenties muodostuu myös kohdistaminen tilanteessa, jossa esimerkiksi suojatie tehdään kahdella ajokerralla (kuva 19). Kuvan esimerkissä ajetaan tiehen ensin kolme suojatien palkkia, sitten kuljettaja kääntää ajoneuvon vastaantulevien kaistalle, ja haluaa kohdistaa toiset kolme suojatien palkkia edelliseen tulostukseen. Tilanne ei ole kuitenkaan ongelmallinen, jos suojatien paikka on hyvin esimerkitty. Tällöin myös oikean puoleisen kaistan suojatien paikka näkyisi kameroissa.



Kuva 19. Kohdistaminen viereiselle kaistalle.

Kuvan 19 kaltaisissa kohteissa pienmerkintä tulee olla aina hyvin esimerkitty, jotta oikea paikka näkyisi auton alapuolisissa kameroissa. Uudet tienpinnat esimerkitään aina erillisellä laitteella. Esimerkinnät suorittaa saman yrityksen työntekijä, ja esimerkintämenetelmät ovat yhdessä sovittuja. Esimerkinnän tarkoituksena on antaa maalikoneen kuljettajalle nopeasti tieto siitä, minkälaista tiemerkintää kohteeseen tulee asettaa. Lisäksi esimerkinnät toimivat kohdistimina tien leveyssuunnassa.

Kolmas tapa kohdistaa pienmerkintä olisi samanlainen kuin kuvassa 19, mutta lasertähtäimen sijaan kamera toimisi tähtäimenä. Auton alla olevien kameroiden sijainti ja näkökenttä olisivat ohjelmiston tiedossa. Kun kuljettaja näkee vanhan pienmerkinnän kamerassa, hän valitsisi kaksi pistettä kuvasta, joita hän kosketusnäytöltä painaa. Tämän jälkeen hänen tulisi valita samanlainen kuva ohjelman kuvakirjastosta, ja valittava samat pisteet myös kuvasta. Tämän jälkeen ohjelmalla on kaikki tarvittava tieto tulostuksen aloittamiseen.

Menetelmän hyvänä puolena on, ettei erillistä lasertähtäintä tarvita lainkaan. Menetelmällä havaittaisiin myös tilanne, jossa ajoneuvo on hieman vinossa tulostettavaan kuvaan nähden. Tällöin ohjelmisto voisi korjata tilanteen kiertämällä tulostettavaa kuvaa tarvittavan määrän.

Huonona puolena menetelmässä näkisin tilanteen, jossa merkintä tehdään uudelle pinnoitteelle. Käyttäjän tulisi valita pisteet kamerasta, jossa näkyy vain puhdasta asfalttia. Tämän jälkeen kuljettajan tulisi valita vastaavat pisteet tulostettavasta kuvasta. Tämä voisi olla hyvin hämmentävää. Kenties kuvan voisi merkitä myös yhdellä pisteellä, jolloin kuvan vinouden korjausta ei suoritettaisi. Merkintätavan voisi kenties suorittaa tapauskohtaisesti halutulla tavalla.

Kohdistus tällä menetelmällä voi olla haastavaa toteuttaa ohjelmallisesti. Kamerat todennäköisesti asennetaan vinoon, jotta niiden näkemäalue ajoneuvon alla olisi riittävän suuri. Jotkin kuvat, kuten nopeusrajoitus 80 on myös haastava käyttäjälle merkitä. Pyöreistä muodoista voi olla vaikea merkitä juuri samaa pistettä sekä kamerasta, että tulostettavasta kuvasta. Pahimmillaan pisteet sattuvat hieman si-

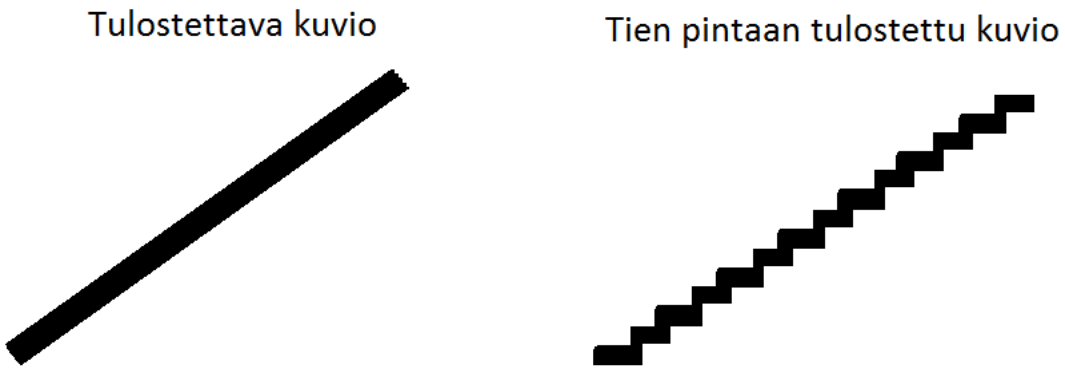
vuun, jolloin ohjelma voi tehdä kuvan vinoon. Menetelmän ongelmat eivät kuitenkaan ole ylitsepääsemättömiä, hyvällä suunnittelulla kohdistamistavasta voisi tulla loistava.

4.3.4 Tulostuksen laatuvaatimukset

Asiakas vaatii, että pienmerkintätulostimella pitää pystyä tekemään suojateitä ja erilaisia symboleja. Suojatiet tehdään menetelmällä, jossa yksi suojatiepalkki koostuu yhtenäisestä kuumamassa-alueesta. Pienmerkintäsymboleissa on sen sijaan useita kaarevia rajoja. Pyrin kartoittamaan sähköpostikyselyssä asiakkaalta vastausta laitteen resoluution vaatimuksiin. (Sähköpostikysely 18.11.2016)

Laitteiston leveyssuuntainen resoluutio on kaikkein kriittisin. Asiakasvaatimuksella on tässä tapauksessa suuri vaikutus koko järjestelmän toimintatavan suunnitteluun, sillä tätä asiaa on myöhemmässä vaiheessa hyvin vaikea korjata.

Asiakkaan mukaan leveyssuuntainen pikselin koko voi olla noin 2 senttimetriä. Kuvassa 20 on kuvattu tulostettavan vinoviivan avulla mitä pikselin leveyssuuntainen koko käytännössä tarkoittaa loppukuvassa.



Kuva 20. Tulostimen resoluution merkitys.

Kuvassa 20 pyritään havainnollistamaan sivusuuntaisen pikselin koon merkitystä vinoilla viivoilla. Liian suuri pikselin koko johtaa tökerön näköisiin reunoihin. Asiakas vaatii laitteelta, että pikselin koko sivusuunnassa tulisi olla pienempi kuin 25

mm. Yhteisymmärryksessä olemme arvioineet, että 25 mm levyiset särmät eivät juurikaan näy katseluetäisyydeltä useiden metrien kokoisissa symboleissa.

Jotta tarkkuusvaatimus varmasti täyttyisi, olemme suunnitelleet laitteiston sivusuuntaisen pikselin olevan 16,6 mm. Käytännön laitteisto koostuisi 16,6 mm levyisistä massaluukuista, joita aukomalla laite voi tulostaa symboleja ajoneuvon edetessä. Tämä tarkoittaisi sitä, että tulostimen ollessa ajoneuvon levyinen (2,6 m) luukkuja tarvittaisiin vieretysten 156 kappaletta.

4.4 Asiakkaat

Suomessa toimii neljä merkittävää tiemerkintöjä tekevää yritystä. Monet näistä yrityksistä tekevät tiemerkintäurakoita myös Ruotsissa ja Norjassa. Suomessa kilpailutetaan vuosittain tiemerkintöjen tekeminen alueittain näiden yritysten kesken. Kun yritys on saanut kilpailutuksessa tietyn alueen hoitaakseen, sen vastuulle jää tiemerkintöjen kunnossapito tällä alueella. Alueet pilkotaan vielä pienemmiksi työmaiksi, joille annetaan erilliset aikarajat, joihin mennessä urakat on hoidettava.

Urakat voivat olla vanhentuneiden maalausten uudistamista, tai uusien uusitun pinnoitteen merkitsemistä. Kunnossapidossa urakoitsijat maalaavat vanhojen merkintöjen päälle uusia merkintöjä. Tässä tapauksessa merkintöjen laatuvaatimusten lisäksi maalaukset tulee tehdä tarkasti vanhojen merkintöjen päälle.

Tiehallinto (2015, s.22) asettaa dokumentissaan Tiemerkintöjen laatuvaatimukset tiukkoja rajoja tiemerkintöjen laadulle. Merkintöjen pituus ja leveys on oltava muutamien senttimetrien tarkkuudella oikean kokoisia. Paksuuden on oltava suurempi tai yhtä suuri kuin tilattu paksuus, yleensä noin 3 mm kuumamassatöissä. Vanhojen merkintöjen päälle tehtäessä uusi merkintä ei saisi poiketa enempää kuin 2 senttimetriä vanhasta merkinnästä, jos vanha merkintä on oikeassa paikassaan.

Samat säännöt koskevat viivatyyppisiä merkintöjä ja pienmerkintöjä. Alueellinen työurakka sisältää tyypillisesti kaikki alueelle tehtävät tiemerkinnät ja pienmerkinnät. Tämän vuoksi kaikki Suomessa toimivat alan yritykset ovat pienmerkintätulostimen potentiaalisia asiakkaita.

Nykyisin tiemerkintäalan yrityksissä on eri työntekijät pienmerkinnöille ja normaaleille pituussuuntaisille tiemerkinnöille. Nykyiset menetelmät ovat hyvin erilaisia, ja työt jaetaan eri työntekijöiden kesken heidän omien vahvuusalueidensa mukaan.

Tiemerkintätulostin voi vaikuttaa merkittävästi tähän. Suunnitelmissamme on tehdä pienmerkintäajoneuvosta käyttöliittymältään hyvin samanlainen kuin aiemmista tiemerkintäjärjestelmistämme. Ensimmäinen ajoneuvo tulee olemaan kuorma-auto, joka osaa tehdä vain erilaisia pienmerkintöjä, sillä tämä on asiakkaan tilaama tuote. Kuitenkin jatkossa on hyvin todennäköistä, että sama ajoneuvo osaa tehdä tiemerkintöjä ja pienmerkintöjä. Sama kuljettaja voi operoida molempia tehtäviä.

4.4.3 Käyttö Suomessa

Tiemerkintäkuljettajilla on erittäin vaativa ammatti. Heidän tulee osata ajaa tarkasti, ja samaan aikaan heidän on käytettävä maalikoneen käyttöliittymää oikeanlaisten viivojen ja merkintöjen tekemiseksi oikealle paikalleen. Heidän on huomioitava normaalia tarkemmin muu liikenne, ja ohjeistettava autoilijoita LED-taulun avulla. Heidän tulee myös suojella vastatehtyjä merkintöjä muulta liikenteeltä. Kaiken lisäksi kuljettajat ovat usein itse vastuussa oman ajoneuvonsa toimivuudesta, jolloin mekaanista huoltamista riittää yllin kyllin.

Kuljettajat ja heidän ajoneuvonsa ovat usein toisistaan paljon poikkeavia. Normaalissa maalikoneessa järjestelmämme ensisijainen käyttöliittymä on ajoneuvoon asennettava kosketusnäytöllinen ajoneuvoihin tarkoitettu tietokone. Tällaisia tietokoneita valmistaa ammattikäyttöön Sunit Oy. Yritys toimittaa ajoneuvoon asennettavia tietokoneita mm. poliisiautoihin, ambulansseihin, metsäkoneisiin sekä takseihin. Tietokoneet ovat erittäin laadukkaita kosketusnäytöltään ja toiminnaltaan. Tietokoneessa ajetaan meidän valmistamamme tiemerkintäkoneen käyttöliittymäohjelmaa, jonka avulla järjestelmää hallitaan.

Ottaen huomion projektien samankaltaisuuden ja myöhemmin mahdollisen yhdistämisen on meidän järkevää tehdä pienmerkinnän käyttöliittymä samanlaiseksi

kuin aiemmissa maalikoneissa. Jo tehtyyn PC-ohjelmaan tämä tarkoittaa lisälaajennusten tekemistä.

On hyvin haastavaa tehdä selkeä käyttöliittymä, josta voi hallita näin laajaa järjestelmää ottaen lisäksi huomioon, että ajoneuvot ja asiakkaiden tarpeet ovat aina erilaisia. Tämän vuoksi kuljettajat ovat usein mukana tuotekehityksessä antamalla ideoitaan ja näkemyksiään. Tämä olisi järkevää myös pienmerkintäajoneuvossa.

Tuotekehityksessä tulisi pyrkiä projektin edetessä siihen tilanteeseen, että asiakasyritys on valinnut ensimmäiselle pienmerkintäajoneuvolle kuljettajan ja pystyisimme tekemään käyttöliittymäasioissa yhteistyötä tämän henkilön kanssa. Parhaassa tapauksessa henkilö olisi kokenut pienmerkintöjen tekijä, sekä innokas edistämään työskentelytapoja elektronisten järjestelmien avulla. Tällöin henkilöllä olisi kaikki tarvittava tieto, minkälaisiin kohteisiin pienmerkintöjä käytännössä tehdään, ja mitä kaikkea tulee ottaa huomioon. Tällaisen kokemuksen saaminen tuotekehitykseen mukaan olisi järjestelmän toimivuuden ja kattavuuden kannalta ensisijaisen tärkeää.

4.4.4 Käyttö ulkomailla

Tähän mennessä tiemerkintäajoneuvoja on toimitettu vain Suomeen, mutta tilanne voi muuttua tulevaisuudessa. Muutamissa maissa ollaan oltu kiinnostuneita laitteistamme, ja kiinnostus varmasti vain lisääntyy, kunhan laitteisto kykenee tekemään myös pienmerkintöjä. Pienmerkintöjen käyttö on yleistä myös muissa maissa.

Kun kyseessä on uuden laitteiston tuotekehitys, olisi järkevää tehdä laitteistoja kotimaiseen käyttöön, kunnes laitteistoon löydetään hyvä ja vakaa toimintatapa. Kun uusista laitteista tulee samankaltaisia kuin vanhoista, ja vanhat laitteet ovat toimiviksi todettuja, voisi uusia laitteistoja valmistaa myös ulkomaille. Potentiaalisia asiakkaita löytyy ulkomailta runsaasti.

Keskeneräisiä ja käytännössä testaamattomia menetelmiä ei kannata testata ulkomailla. Huoltokustannukset voisivat nousta liian korkeiksi, jos laitteisto kohtaa odottamattomia ongelmia.

Markkinanäkymät ovat kuitenkin ulkomailla meille suotuisia, sillä kilpailevia tuotteita ei tällä hetkellä ole. Yleisen trendin mukaan myös ulkomailla pyritään edistämään työturvallisuutta ja tehokkuutta pienmerkintöjen tekemisessä.

4.5 Yrityksen tavoitteet

Yrityksellämme on Suomessa tiemerkintälaitteissa vahva markkina-asema. Tätä asemaa vahvistaisi entisestään tilanne, jossa laitteistolla olisi tehokkaalla ja luotettavalla tavalla mahdollista tehdä myös pienmerkintöjä.

Onnistunut tuotekehitys auttaisi meitä laajentumaan alalla ylivoimaiseksi järjestelmätoimittajaksi pohjoismaissa. Yrityksen tavoitteena on lisätä tuotteiden myyntiä tällä alalla Suomessa ja ulkomailla.

Meidän mielestämme on hyvin todennäköistä, että jos projekti tuottaa asiakkaallemme suunnitelmien mukaisen tavan tehdä pienmerkintöjä, tavasta tulee hyvin suosittu. Kun on varteenotettava mahdollisuus tehdä pienmerkintöjä tehokkaasti ja turvallisesti, tilaajat alkavat vaatia urakoita suoritettavaksi tällä tavalla.

Turvallisuutta parantavien työtapojen käyttäminen otetaan avosylin vastaan sekä urakoita tilaavilta että niitä tekeviltä tahoilta. Tämän vuoksi menetelmän suosio todennäköisesti kasvaa erittäin nopeasti. Kun näin käy, on meidän asemamme laitteiston toimittajana kilpailijoihin nähden erittäin hyvä.

Ensimmäinen asiakkaamme ostaa tuotteen ensisijaisesti sen vuoksi, että työskentelytapa parantaa yrityksen työntekijöiden turvallisuutta. Myös tässä yrityksessä uskotaan, että menetelmä saa suosiota tilaajien vaatimusten kautta lähitulevaisuudessa. Vaatimukset kasvavat, jos vaihtoehtoja on olemassa.

Asiakkaamme arvioi heidän yrityksensä tarpeen olevan mahdollisesti kaksi kappaletta tienmerkintäprinttereitä, joiden avulla heidän urakoiden sisältämät pienmerkintätyöt saataisiin tehtyä. (Sähköpostikysely 18.11.2016)

Suomessa toimii neljä merkittävää tiemerkintäyritystä, joiden työpanoksella Suomen tiemerkintäurakat valtaosin suoritetaan. Yritysten tiemerkintää tekevät osat ovat arviolta yhtä suuria. Tämän perusteella voisi päätellä, että jos uusi tapa tehdä pienmerkintöjä yleistyy, tarve Suomen markkinoilla olisi noin kahdeksan toimivaa ajoneuvoa samanaikaisesti. Yleistymisen jälkeen potentiaalinen tarve ulkomailla on erittäin merkittävä.

Sen lisäksi, että tavoitteemme on laajentaa toimintaamme tiemerkintäalalla, yrityksemme on aina saanut parhaat tuloksensa laitteistoissa, jotka ovat suunnattu yritysasiakkaille. Laitteistot ovat tyypillisesti arvokkaita, monimutkaisia, asiakkaiden tarpeet tyydyttäviä, eikä laitteistoilla ole varteenotettavia kilpailijoita.

Laitteesta, joka olisi suunniteltu kuluttajille kilpailluilla markkinoilla, mahdollinen tuotto tulee ainoastaan valtavan myyntimäärän ansiosta. Tällöin yrityksen tulisi mukauttaa tuotantolinjojansa nimenomaan kyseistä tuotetta varten. Tuotantokustannuksia tulisi minimoida kaikin mahdollisin keinoin. Tällainen menettelytapa ei olisi yritykselle järkevää strategista toimintaa, sillä yrityksen muut tuotteet ovat hyvin paljon tiemerkintäalasta poikkeavia.

Eräs menestyvän yrityksen toimitusjohtaja totesi, että yrityksen tuotannon tehokkuus ja joustavuus kulkevat aina päinvastaisiin suuntiin. Jos yrityksen tuotantolinja on optimoitu tekemään nuppineuloja mahdollisimman kustannustehokkaasti ja nopeasti, ei samalla linjalla voida tehdä hakaneuloja.

Yritys ei ole rakenteeltaan sopiva tällaiseen liiketoimintaan ilman valtavia muutoksia. Sen vuoksi yrityksen olisi järkevää lisätä toimintaansa laitteistoissa, jotka ovat asiakkaille tärkeitä, ja niin monimutkaisia ja laajoja hankkeita, että jopa 10 laitteiston myynti 5 vuoden aikaikkunassa olisi hyvä saavutus. Yksi laitteisto tulee loppuasiakkaalle maksamaan arvioiden mukaan noin 500 000 €. Hinta-arvio on kohtuullisen vähäisen informaation avulla arvioitu ja tarkentuu projektin edetessä.

Tuotekehitystyö tiemerkintätulostimen parissa vaikuttaa lupaavalta. Projekti on linjassa yrityksen yleisen liiketoimintatavoitteen kanssa ja pyrkii laajentamaan yrityksen toimintaa suunnitellulla markkina-alueella. Valmis tuote olisi ominaisuuksiltaan sellainen, jonka tyyppisissä projekteissa yritys on vuosien aikana menestynyt parhaiten.

Tuotekehitysprojektin koko on myös sopivaa kokoluokkaa. Yritys on tehnyt vastaavia projekteja usein aikaisemminkin eikä kyseinen projekti aiheuta yrityksen toiminnalle poikkeuksellisen suuria riskejä. Vaikka uudenlaisen ja monimutkaisen laitteiston valmistaminen tiukassa aikataulussa sisältääkin aina riskejä.

4.6 Tuotekehitystiimin lähtökohdat

Ajallisesti tuotekehitysprojekti on myös hyvä, sillä juuri tällä hetkellä RMD-10 järjestelmän kehittäjillä ei ole juuri tällä hetkellä liian suuria projekteja, etteikö tiimin työtakkaa voisi kuormittaa hiukan lisää.

RMD-10 järjestelmä kehitettiin noin kaksi vuotta sitten, joten tiemerkintälaitteistojen kanssa toimiminen on tuotekehitystiimillä hyvin tuoreessa muistissa. Aiempi järjestelmä oli suunnattu kaistaviivojen, keskiviivojen sekä reunaviivojen tekemiseen. Pienmerkintälaitteistossa on kuitenkin aiemman järjestelmän kanssa hyvin paljon samanlaisuutta.

Tuotekehitystiimi, joka valmisti RMD-10 järjestelmän, pysyy täysin samana pienmerkintälaitteiston rakentamisessa. Mekaniikkaosien suunnitteluun tuotekehitystiimiin otetaan kuitenkin lisävahvistusta.

Ohjelmistot ovat jaettu kolmeen osaan, jokaisessa on yksi ohjelmiston kehittäjä. PC-ohjelmaa kehittää yksi henkilö, ja ohjelma on pohjimmiltaan sama, kuin RMD-10 laitteistossa. Tulostimen haasteellinen käyttöliittymän tarve aiheuttaa kuitenkin uutta ohjelmiston kehitystä tällä osa-alueella.

RMD-10 järjestelmään verrattuna laitteistoon on suunniteltu tulevaksi uudenlaisia CAN-yksiköitä, jotka ovat suunniteltu nimenomaan pienmerkintätulostinta varten.

Uudet yksiköt suunnitellaan ohjaamaan tulostimen lukuisia lähtöjä mahdollisimman tehokkaasti. On suunniteltu, että yksi laite ohjaisi 64 lähtöä, joten näitä laitteita tarvitaan noin kuusi kappaletta. Näiden laitteiden ohjelmoinnista vastaa toinen ohjelmistosuunnittelija.

Kolmannen ohjelmistosuunnittelijan tehtävänä on kehittää keskusyksikköön tulostinta ohjaava ohjelmisto. Lähtökohtaisesti paras tilanne olisi, jos keskusyksikön RMD-10 ohjelmaan voisi tehdä tulostinta tukevan laajennuksen. Näin ohjelmisto pysyisi samana molemmissa laitteissa.

Laitteistot ovat kuitenkin luonteeltaan niin paljon eroavia, että voisi olla paras tehdä tulostimen keskusyksikkö täysin omaksi ohjelmakseen, jotta ohjelmistot säilyisivät mahdollisimman selkeinä ja tehokkaina. Asiakkaan arvion mukaan on kuitenkin epätodennäköistä, että asiakkaat haluaisivat RMD-10 järjestelmän toiminnot sekä tulostimen toiminnot samaan ajoneuvoon. Mikäli tätä myöhemmin yritettäisiin, olisi hyvä, jos sama keskusyksikkö pystyisi molempiin toimintoihin.

5 PIENMERKINTÄLAITTEEN OHJELMISTOKEHITYS

Tässä projektissa tulisi ohjelmiston kehityksen aikana muistaa projektin luonne. Projekti tulee etenemään niin, että meillä kehitetään ohjelmistoa ja suunnitellaan tiettyjä mekaanisia osia. Yrityskumppanimme valmistaa samaan aikaan Ruotsissa laitteen mekaniikkaa. Laitteisto rakennetaan keväällä ja alkukesästä laitteen tulisi olla toimintakuntoinen. Pyrimme pitämään toisemme ajan tasalla projektin etenemisestä säännöllisin projektikatsauksin, ja pyrimme läpiviemään projektin hyvän ja tarkan suunnittelun avulla.

Ohjelmiston näkökulmasta projektin kriittisin aika on mekaniikan valmistumisen jälkeen, kun ajoneuvo on täysin rakennettu. Vasta kun mekaniikka on täysin valmis ja ajoneuvo olisi ajokuntoinen, voimme viimein kokeilla järjestelmän ohjelmistoa. On mahdollista, että projekti on jo tässä vaiheessa viivästynyt alkuperäisestä aikataulustaan.

Laitteen ohjelmistoa voidaan kokeilla käytännössä vasta kun ajoneuvo on valmis, ja usein asiakkaat haluavat laitteiston mahdollisimman nopeasti tuottavaan työhön. Mekaniikan valmistava taho haluaisi nähdä mekaniikan toimivuuden käytännössä. Ilmapiiri tässä vaiheessa on hiukan kärsimätön, ja tämä on täysin ymmärrettävää.

Vasta tässä vaiheessa pystymme aloittamaan täysipainoisen ohjelmiston testauksen. Yllättäviä asioita tulee tässä vaiheessa aina, ja ohjelmistoa joudutaan muokkaamaan. Laitteet ovat yleensä monimutkaisia kokonaisuuksia niin mekaanisesti kuin ohjelmallisestikin ja hyvästä suunnittelusta huolimatta yllätyksiä voi testivaiheessa tulla. Muutokset olisi tällöin myös hyvä tehdä ripeästi.

5.1 Ohjelmiston toiminta

Oikeassa ympäristössä kuljettaja käyttää PC-ohjelmaa, jonka avulla hän määrittelee laitteistolle tehtävän. Valinnat kuvan tyypistä, paikasta ja koosta tulee valita PC-ohjelman käyttöliittymästä.

PC-ohjelma kertoo tämän informaation keskusyksikölle, jonka tehtävä on suorittaa tulostus sitä mukaa kuin ajoneuvo etenee. Keskusyksikkö saa pulssitietoa ajoneuvon etenemisestä, ja voi pitää yllä myös ajoneuvon nopeutta tämän tiedon pohjalta.

PC-ohjelmalla määritetään tähtäimen avulla paikka jossa tulostus tulisi tapahtua. PC-ohjelma kertoo keskusyksikölle etäisyyden tähtäimeen ja keskusyksikkö ohjaa 250 – 400 lähtöä CAN-väylän kautta. Ohjattavien lähtöjen lukumäärää ei vielä tiedetä, mutta se on todennäköisesti hieman alle 400.

Lähtöjen määrään vaikuttaa tulostimen leveys, yhden lähdön ohjaama leveys kuumamassassa, sekä oheisohjausten määrä. Oheisohjauksiksi laskisin paineilmasuuttimet, joilla puhdistetaan tulostettava alue hiekasta, lasihelmisuuttimet, hiekkasuuttimet suojateille, sekä vesisuuttimet. Lopuksi kuumamassan päälle asetettu vesikalvo jäähdyttää pienmerkintää ja suojaa ajoneuvojen renkaita tarttuvalta massalta. Laitteistossa tulee olemaan useita pumppujen ohjauksia tulostuksen aikana. Pumpuilla esimerkiksi paineistetaan sopiva määrä massa- ja helmilinjoja.

5.2 Ohjelmiston rakenne

Kun katsomme yrityksen aikaisempia projekteja ja niiden kehitysvaiheita, on erittäin yleistä, etteivät ohjelmistot kaikilta osin täytä asiakkaan vaatimuksia heti ensimmäisten asennusten jälkeen. Asiakkaat eivät osaa eivätkä muista antaa täydellistä vaatimuslistaa ohjelmistojen toimivuuden suhteen. On täysin ymmärrettävää ettei kaikkea voida osata ennakoida. Ohjelmistoja joudutaan aina muokkaamaan useita kertoja asiakkaan tarpeisiin sopiviksi ennen kuin oikea toimintatila löytyy kaikissa olosuhteissa. Voimme olettaa, että jokaisen asennetun tuotteen jälkeen meidän tulee tehdä useita kertoja kyseiseen laitteen ohjelmistoon muutoksia.

Kuten aiemmassa luvussa todettiin, tässä projektissa voimme odottaa lyhyttä ja intensiivistä testausaikaa ohjelmistolle. Tällöin ohjelmistoon joudutaan tekemään ripeästi muutoksia kun muutostarpeita kohdataan.

5.3 Mahdolliset riskit

Ohjelmiston kehittäjän näkökulmasta ohjelmistossa voi olla useita erityyppisiä vikoja. Valtaosa vioista on helposti paikannettavia ja helposti korjattavia. Vian oireet kertovat täsmälleen missä ongelma piilee ja tarvittava korjaus on pieni.

Seuraavaksi yleisin vian tyyppi on puuttuvien ominaisuuksien havaitseminen. Ominaisuuksia ei joko ole tai niistä puuttuu jokin tarvittava osa. Ohjelmisto kaipaa tällöin pientä laajennusta. Oireet kuullessaan tuotekehittäjät tietävät heti, mistä on kyse, mutta ohjelmistomuutokset vievät hiukan kauemmin kuin ensimmäisessä vikatyypissä. Vaadittava laajennus on kuitenkin tyypillisesti erittäin pieni, muuten muutostarve olisi huomattu jo suunnittelussa.

Aiempia vikoja harvinaisemmassa, mutta silti kohtuullisen yleisessä vikatyypissä ohjelmistossa näyttäisi olevan vikaa, mutta vika onkin jossain muualla. Jokin osa voi olla väärää tyyppiä eikä toimi suunnitellulla tavalla. Voi olla, ettei laitteen mekaniikassa ole otettu jotakin asiaa huomioon. Tällöin ohjelmistoon ei välttämättä tarvitse tehdä muutoksia ollenkaan, mutta vian paikantaminen ja todentaminen voi olla joskus työlästä ja pitkäkestoista.

Toiseksi pahin tilanne on yleensä sellainen, että jokin ominaisuus ei toimi, vaikka sitä on testattu simulaattoreissa tuotekehityksessä useita kertoja. Laitteen ohjelmisto esimerkiksi kaatuu odottamattomasti, tai toimii täysin epäloogisesti. Kun ominaisuus toimii simulaattoreissa ja vian oireet ovat täysin epäloogisia, on tuotekehityksessä hyvin vaikea löytää vikaa.

Vika voi olla jossain ihan muualla, kuin mihin oireet viittaavat. Joskus vikoja voi olla useampia, joiden yhdistetyt oireet näyttävät epäloogisilta. Pahimmillaan vian paikantaminen voi kestää viikkoja. Ohjelmaviat voivat joskus olla erittäin vaikeasti paikannettavia. Tällaiset vaikeasti löydettävät ohjelmaviat ovat kuitenkin yleensä helppoja korjata sen jälkeen, kun vika on paikallistettu. Jos korjaus vaatisi paljon ohjelmamuutosta, se olisi varmasti löydetty jo aiemmin.

Pahin vika on helppo löytää. Se löydetään yleensä heti testien alettua oikeassa ympäristössä. Huomataan ettei jokin osa ohjelmasta toimi kunnolla. Hyvin harvoin

vika voi olla sellainen, että sille ei ole nopeaa korjausratkaisua. Jokin on jäänyt suunnittelussa huomiotta, ja sen vuoksi ohjelma on rakennettu noudattamaan sellaista toimintalogiikkaa jota ei voidakaan käyttää. Tällöin ainoa ratkaisu on tehdä ohjelmistoon rajuja muutoksia, ja rakentaa viallinen toimintalogiikka eri tavalla.

Tällaisessa tilanteessa aika on ongelma. Tällaiseen ongelmaan ei olla koskaan varauduttu, ja aikataulut viivästyvät. Ongelmaa lähdetään välittömästi ratkaisemaan ohjelmistomuutoksilla. Kiireessä ja paineen alla tehdyt ohjelmamuutokset ovat usein huonosti rakennettuja. Vaikka muutos toimisikin, heikko suunnittelu uudelle toimintalogiikalle tulisi vastaan tuotekehityksessä vielä vuosien päästäkin.

On tavallista, että nopeasti tehty koodimuutos koskettaa muuta ohjelmaa epäloogisesti ja kosketuspinta on liian laaja. Tämä johtuu siitä, että olisi kestänyt kauemmin tehdä muutos paremmin. Paremmin tehty ohjelmamuutos olisi vaatinut myös hieman suunnittelua ja pohdiskelua siitä, kuinka kosketuspinnat muuhun ohjelmakoodiin kannattaisi tehdä. Tähän ongelmaan voidaan varautua ohjelman rakenteessa ja suunnittelussa.

Rakennesuunnittelussa tulee ottaa huomioon, että yhdellä ohjelmamoduulilla (funktiolla) on vain yksi tehtävä. Ohjelman muissa osissa ei saa tehdä tämän funktion toimintoja, vaan toiminnot on suoritettava tämän funktion kautta. Moduulin on oltava mahdollisimman itsenäinen ja yksinkertainen. Sen kosketuspinta muuhun ohjelmaan on oltava mahdollisimman ohut.

Asia voi kuulostaa itsestäänselvyydeltä, mutta todellisuudessa on vaikea tehdä ohjelmaa, joka noudattaisi aina näitä ohjeita. Funktioille annetaan helposti myös lisätehtäviä, joita se voi suorittaa päätoimensa ohessa. Ajatus voi olla houkutteleva, kun lisätehtävälle etsitään suorittajaa. On huomattavasti raskaampaa luoda tälle tehtävälle oma itsenäinen moduuli, kuin käyttää jo olemassa olevaa moduulia tehtävän suorittamiseen. Lisätehtävän myötä funktion kosketuspinta muuhun ohjelmaan aina kasvaa, ja juuri sitä tulisi välttää.

On myös erittäin yleistä luoda itsenäinen moduuli hoitamaan jotakin tiettyä tehtävää oikeaoppisesti. Kuitenkin ohjelmiston kehityksen yhteydessä huomataan jokin hieman poikkeava tilanne. Poikkeavan tilanteen hoitaminen ei suoraan onnistu-

kaan kyseiseltä moduulilta. Kehittäjä voi helposti erehtyä tällöin tekemään tehtävän muilla tavoin. Tämän jälkeen ohjelmistossa on tilanne, jossa selkeästi mennään tehtävää suorittavan funktion ”tontille”, vaikka ei pitäisi.

Tämä voi aiheuttaa useita erilaisia ongelmia, kun kyseisen tehtävän suoritukseen tarvitsee tehdä nopeasti muutoksia. Tällaisia epäloogisia sidonnaisuuksia on vaikea, ellei mahdoton huomata silloin kun johonkin ohjelmamoduulin toimintaan tarvitaan nopeasti muutoksia.

Sen sijaan muutosten tekeminen on helppoa, jos ohjelma on rakennettu oikein. Tiedetään välittömästi mihin moduuliin tarvitaan muutoksia, kun muutoksia tarvitaan. Kehittäjä saa muokata moduulia niin paljon kuin parhaaksi näkee, eivätkä muutokset kosketa mihinkään muuhun kuin alkuperäisen tehtävän suorittamiseen. Kehittäjä voi olla varma, ettei tehtävän suorittamiseen puututa ulkopuolelta millään tavoin. Kehittäjä voi luottaa siihen, että ohjelmisto toimii, vaikka tehtävän suoritus rakennettaisiin kokonaan uudestaan, vaikka täysin eri toimintaperiaatteella.

Ohjelmistoa voidaan muokata huomattavasti helpommin, kun ohjelmisto koostuu itsenäisistä ja irrallisista moduuleista. Pikaisetkaan muutokset eivät tällöin aiheuta yllättäviä ongelmia ohjelmiston toimintaan. Ohjelmistojen kehittäjien tulisi aina pyrkiä tähän määränpäähän, mutta tällaisessa projektissa se on ensiarvoisen tärkeää. Yllätyksiä tulee hyvin todennäköisesti uuden laitteiston asennus- ja testivaiheessa, jolloin ohjelmiston tulee mukautua muutoksiin.

5.4 Ohjelmiston vaatimukset

5.4.1 Laskentaprosessorin suorituskyky

Tulostustilanteessa laitteiston on tulostettava pienmerkintää sitä mukaa kuin tulostin etenee maantiellä. Tulostuksen on tapahduttava koko tulostimen leveydeltä samalla ajokerralla. Tulostustoimenpiteet on tarkoitus suorittaa aina kun ajoneuvo saa uuden matkapulssin. Tämä tarkoittaa sitä, että ajoneuvo on edennyt yhden senttimetrin.

Ajoneuvolla tullaan ajamaan alle 5 km/h tulostuksen aikana, joten laskentatehoa pohtiessamme käytämme tätä lukua esimerkkinä. Jos ajoneuvo ajaa 5 km/h, yhden sekunnin aikana laitteistolle tulee noin 140 pulssia, jonka aikana ajoneuvo on edennyt senttimetrin. Tämä tarkoittaa sitä, että laitteisto saa tällaisen sentin pulssin 7,2 millisekunnin välein. Tässä ajassa laitteen tulisi analysoida kuvaa, ja suorittaa tarvittavat tehtävät riittävän nopeasti

Ohjelmistolle pahin mahdollinen tilanne olisi se, että jokainen ohjattava suutin vaihtaa tilaansa jokaisella senttimetrillä. Tällaista kuviota urakoitsijoiden eivät varmaankaan tarvitse tulostaa, mutta tilannetta on hyvä käyttää worst case -skenaariossa. Ajoneuvossa tulee olemaan noin 350 ohjattavaa lähtöä ja jos kukin lähtö vaihtaisi tilaa jokaisella senttimetrillä, tilanne olisi ohjelmistolle vaativin mahdollinen.

Jos ohjelmisto kykenee tulkitsemaan kuvasta yhden suuttimen muutostarpeen, ja toteuttamaan sen alle 20 mikrosekuntiin, ohjelmistolla pystyisi tekemään vaikeinta mahdollista kuviota 5 km/h nopeudella. Ohjelmisto tarvitsisi myös hiukan aikaa muihin toimintoihin tulostamisen aikana.

Suoritustehovaatimus on todella tiukka useille sulautettujen järjestelmien prosessoreille. Alustavien testien jälkeen uskomme, että ohjelmisto voisi kyetä tähän, jos algoritmi, joka muuntaa tulostettavan kuvan suutinohjauksiksi pysyy yksinkertaisena. Vaatimusten tarkentuessa, ohjelman loogisen päättelyn ketju usein monimutkaistuu projektin edetessä. Projektin ollessa vielä alkuvaiheessa, monimutkaisuutta todennäköisesti tulee lisää.

5.4.2 Ohjelmistojen päivittäminen

Järjestelmämme PC sisältää 3G-yhteyden ja PC:n sekä tulostimen välillä on tiedonsiirtoyhteys. Tulostinta ohjaavaan yksikköön on tehtävä oma ohjelmointiohjelma, jolla laite voidaan päivittää etänä 3G-yhteyden kautta. Mikäli näin ei tehdä, on jokaisen päivityksen yhteydessä mentävä laitteen luokse ja kytkettävä tietokone ohjelmointiadapterilla piirikorttiin, jolloin uusi ohjelmaversio voidaan päivittää

prosessorille. Tämä olisi täysin kestämätön tilanne projektin kannattavuuden kannalta. Kun järjestelmässä on internetyhteys, on kuvatun kaltaisessa projektissa etäpäivityksen tekeminen pakollinen vaatimus.

PC-ohjelman päivittäminen on helppoa - käyttäjän tarvitsee sulkea ohjelma ja korvata ohjelmatiedosto uudella versiolla. Tämän jälkeen käyttäjän tulee käynnistää ohjelma uudelleen. Uusi versio voidaan ladata verkkosivuilta, tai välittää esimerkiksi sähköpostitse.

Keskusyksikkö tulee myös pystyä päivittämään helposti. Ohjelman päivittäminen tulisi tapahtua PC-ohjelman kautta. PC-ohjelmasta voitaisiin valita päivitettävä tiedosto, ja PC-ohjelma lähettäisi tiedoston ethernet-yhteyden kautta keskusyksikölle. Keskusyksikkö kerää ohjelman muistiinsa ja tarkistaa että se on saapunut kokonaisena.

Ohjelmistoon on tehtävä erillinen ohjelma joka osaa päivittää uuden version muistista vanhan ohjelman päälle. Tällaista ohjelmaa kutsutaan bootloader-ohjelmaksi. Bootloader suoritetaan aina käynnistyksen yhteydessä ja se antaa suoritusvastuun heti varsinaiselle ohjelmalle tai päivittää tarvittaessa ensin uuden version varsinaisen ohjelman tilalle. Versio päivitetään ainoastaan silloin kun uusi versio on ladattu laitteen muistiin varsinaisen ohjelman toimesta. Bootloader-ohjelmaa ei pystytä päivittämään etänä, sillä päivitysohjelma ei kykene päivittämään itseään. Bootloader-ohjelman päivittäminen onnistuu ainoastaan kytkemällä ohjelmointiadapteri laitteeseen ja tällöin päivitys onnistuu esimerkiksi kannettavalla tietokoneella.

Kun bootloader-ohjelma tehdään toimivaksi ja sen tehtävät ovat hyvin yksinkertaisia, sitä ei ole tarvetta päivittää valmistamisen jälkeen. Kaikki äly on varsinaisessa ohjelmistossa, ja sitä tarvitsee päivittää todennäköisesti usein.

Ohjelmiston päivittäminen etänä on kohtuullisen luotettavaa. Kokemuksieni mukaan kenties 1:3000 päivityskerrasta kohtaa ongelmia. Kuitenkin joskus ohjelman päivittäminen menee pieleen, eikä varsinainen versio käynnisty, eikä keskusyksikkö näin ollen saa yhteyttä PC-ohjelmaan. Tällöin PC-ohjelmalla ei myöskään voida päivittää ohjelmaa toiseen kertaan. Sama tilanne kävisi, jos laitteesta häviäisi käyttösähköt juuri kun bootloader on päivittämässä uutta versiota.

Tapaukset ovat harvinaisia mutta kuitenkin niin todennäköisiä, että niiden varalle tulisi keksiä jokin keino varmistaa laitteiston toimivuus kaikissa tilanteissa. Ensimmäinen vaihtoehto olisi, että bootloader rakennettaisiin niin, että se osaisi itse yhdistää itsensä PC-ohjelmaan ja ladata uuden version. Tällöin ongelmatilanteessa bootloader voisi vain ladata uuden version uudelleen. Bootloaderille varattu ohjelmamuisti on kuitenkin todella pieni, ja ethernet-yhteyden muodostaminen vaatii melko suurta ohjelmamuistia.

Toinen vaihtoehto olisi tehdä varsinaisesta applikaatiosta karsittu versio, joka kykenee ainoastaan ottamaan yhteyttä PC-ohjelmaan, ja kommunikoimaan sen kanssa. Tämä ohjelma voitaisiin päivittää esimerkiksi ohjelmamuistin loppuosaan, jonne bootloaderin valtuudet eivät yllä. Tällöin tämä ohjelma löytyisi aina ohjelmamuistin lopusta.

Tällöin bootloader-ohjelmaan voisi tehdä ominaisuuden, että jos uusi versio ei käynnisty oikein, bootloader voisi polttaa uuden version tilalle tämän karsitun version ohjelmamuistin lopusta. Kun tämä ohjelma sitten käynnistyy, se yhdistyy PC-ohjelmaan, jonka kautta varsinainen versio voidaan ladata uudestaan.

5.4.3 Väylien hyödyntäminen tiedonsiirrossa

Järjestelmässämme on PC, joka toimii urakoitsijan käyttöliittymänä laitteiston asetuksien asettamiseen ja anturien sekä tilojen lukemiseen. PC on laitteistossa sijaitsevaan keskusyksikön kanssa yhteydessä ethernet-väylän kautta.

Tulostimen operointi tulisi ehdottomasti tehdä jo olemassa olevan PC:n kautta. Kosketusnäytöllinen PC on optimaalisesti asetettu jo nykyisessä järjestelmässä urakoitsijan ulottuville kuorma-auton hyttiin ja tilavasta näytöstä olisi helppo operoida tulostinta kaikin tavoin. Kaikki tulostimeen asennettavat anturit ja tilatiedot tulisi olla PC:ltä helposti nähtävissä ja säädettävissä.

Keskusyksikkö hallinnoi ajoneuvossa olevaa CAN-väylää. Ajoneuvoissa on myös tehdasvalmisteisena oma CAN-väylänsä ajoneuvon toimintaan, mutta emme

käytä sitä, vaan teemme oman CAN-väylän keskusyksikön ja ohjauslaitteiden välille. CAN-väylä on toimintavarman ja nopean tiedonsiirtokapasiteettinsa ansiosta loistava valinta eri yksiköiden välille.

Olisi luonnollista, jos tulostin voitaisiin liittää RMD-10 järjestelmän olemassa olevaan CAN-väylään, jolloin keskusyksikkö voi hallinnoida laitetta toimintavarmasti ja nopeasti. Kaikki tulostimessa olevat anturit ja tilatiedot siirtyisivät CAN-väylän avulla keskusyksikköön, josta ne välitetään myös PC ohjelmalle.

CAN-väylän hyödyntäminen tiedonsiirrossa olisi optimaalista myös laitteiston asennuksessa, sillä verkko on jo muutenkin olemassa järjestelmässämme. Näin toteutettuna voidaan minimoida kaapelointi eri yksiköiden välillä. CAN-väylään voidaan kytkeä 127 eri yksikköä ja väylän pituus saa olla maksimissaan 100 metriä 500 kbit/s tiedonsiirtonopeudella (CiA, 2016). Nykyisissä järjestelmissä CAN-väylän pituus on ollut noin 30 metriä, ja laitteita on ollut maksimissaan yhdeksän. CAN-väylää voisi järjestelmän vuoksi näiden tietojen pohjalta vielä kuormittaa.

5.4.4 Kuvion kohdistaminen

Kuvion kohdistaminen tiessä on suuri haaste ohjelmistolle. Tällä on suuri vaikutus laitteen käytettävyyteen, ja siihen tulisi tuotekehityksessä panostaa sen mukaisesti. Kohdistamisen kaltaiset haasteet tulisi huomioida myös ohjelmistokehitysmenetelmää valitessa.

Luvussa 4.3 esiteltiin erilaisia kohdistusmenetelmiä. Kohdistusmenetelmistä pidettiin tuotekehitystiimin kesken palaveri, jossa erilaisten menetelmien hyviä ja huonoja puolia vertailtiin keskenään. Tultiin siihen tulokseen, että tähtäin on liian kaukana, jos tähtäin olisi ajoneuvon etupuolella. Tämän vuoksi tähtäin on oltava tulostimen edessä, ajoneuvon alapuolella. Tulostin on tarkoitus olla ajoneuvon takaosassa.

Suunniteltiin, että ajoneuvon pohjassa voisi olla suuria reikiä, joista kamerat kuvaavat alaspäin. Tällöin kamerat olisivat paremmassa suojassa tiepölyltä ja lialta, sekä kameroiden näkemäalue olisi suurempi kuin ajoneuvon pohjasta katsottuna.

Kameroita tarvittaisiin ainakin kaksi kappaletta, jotta koko ajoneuvon leveyssuunta saataisiin kameroiden näkökenttään. Kameroiden kuvat olisivat erillisellä kosketusnäytöllä, jota klikkaamalla kuljettaja voisi merkata tulostuksen aloituspisteet. Aloituspisteitä kuljettaja voi valita yhden tai useamman, mutta saman verran pisteitä on valittava myös tulostettavasta kuvasta, jotta ohjelma osaa kohdistaa toiminnon oikein.

5.4.5 Mekaanisen toteutuksen vaatimukset

Mekaanisesti laitteisto on erittäin haastava. Laitteisto on massiivinen, siinä tulee olemaan noin 300 paineilmalla ohjattua luukkua kuumamassalle. Lisäksi helmisuuttimia on tarkoitus tulla noin 70 kappaletta. Ohjattavia venttiililähtöjä tulee olemaan runsaasti, ja se näkyy myös laitteiston kokonaisvirrankulutuksessa.

Voiko samoista suuttimista todella ajaa oikean määrän hiekkaa tai lasihelmeä kuumamassan päälle? Pitäisikö näihin ulostuloihin saada myös pumppu, jotta tehoa voisi lisätä, jos esimerkiksi hiekkaa ei tulekaan suojateihin tarpeeksi? Tuotekehityksen tulisi myös laskea, ja arvioida, riittääkö paineilma ohjaamaan yhtäaikaisesti näin monia lähtöjä, ja vaikuttaako tämä suoritukseen millään tavalla.

5.4.6 Projektin aikataulu

Ensimmäinen tuote tulisi valmistua kesällä, eli noin 5-6 kuukauden kuluttua. Aikataulu on tiukka, mutta saavutettavissa, jos emme kohtaa suuria ongelmia. Ongelmallista on, että ajoneuvo valmistuu mekaanisesti samaan aikaan kuin ohjelmistokin. Koko kokonaisuuden testaaminen onnistuu vasta kun kaikki on valmista. Tämän vuoksi projekti voi kohdata suuriakin ongelmia vielä projektin loppuvaiheessa, jossa aikataulu näytti vielä siihen asti pitävän hyvin. Testeille tulisi vielä projektin loppuvaiheessa varata riittävästi aikaa.

5.5 Ohjelmiston testaus

Kuinka voimme testata tällaista toimintaa ilman varsinaista laitetta? Laite valmistuu vasta myöhäisessä vaiheessa, eikä varsinaiseen testaukseen jää paljon aikaa, joten meidän tulee keksiä hyviä tapoja testata laitteiston toimintaa ennen laitteiston valmistumista.

PC-ohjelmaa on helppo testata ilman kokonaista laitteistoakin. PC-ohjelman pääasiallinen tehtävä on toimia käyttöliittymänä kuljettajalle. Kuljettajan on kyettävä näkemään ohjelman kautta antureiden ja ohjauksien tilatiedot. Kuljettajan on myös kyettävä helposti ja loogisesti muokkaamaan laitteiston asetuksia.

Käyttöliittymäasiat eivät ole riippuvaisia varsinaisesta laitteistosta, joten testaamiseen ei tarvita mekaanista laitteistoa tai monimutkaisia simulaattoreita. PC-ohjelman tehtäviä voidaan testata tuotekehityksessä ja luottaa siihen, että testatut moduulit toimivat myös kuorma-autossa.

Keskusyksikön toiminnan testaaminen on vaativampaa. Yksittäisiä ohjelmamoduuleita voidaan testata monissa eri olosuhteissa, mutta laitteisto on testattava myös täydellisessä järjestelmässä. Kun järjestelmä sisältää kaikki laitteistot, anturit, lähdöt kuin todellisuudessakin, keskusyksikkö kuormittuu kaikkein eniten.

Täydellisessä testiympäristössä keskusyksikköön tulisi kaikki CAN-verkon laitteet, jotka tulevat oikeastikin olemaan laitteistossa. Keskusyksikkö tulisi olla ethernet-verkon kautta yhteydessä PC-ohjelmaan, sekä keskusyksikölle tulisi olla säädettävä matkapulssia luova generaattori, jonka avulla keskusyksikölle voidaan syöttää erilaisia ajonopeuksia.

CAN-verkkoon tullaan kytkemään kenties 7-9 päätelaitetta. Näihin laitteisiin tullaan kytkemään pumppujen sekä venttiilien ohjauksia. Kaikki anturit tullaan myös kytkemään näihin päätelaitteisiin. Päätelaitteiden tehtävänä on päivittää antureiden tietoja keskusyksikölle CAN-väylän avulla halutulla nopeudella. Vielä tärkeämpi tehtävä on ottaa CAN-väylästä vastaan laitteelle tarkoitettuja viestejä ja päivittää omien lähtöjen tilat mahdollisimman nopeasti komennon tullessa.

Yhteen päätelaitteeseen kytkettävien lähtöjen ja anturien määrä olisi järkevää rajoittaa 64 kappaleeseen. Rajoitus tulee siitä, että yhdessä CAN-väylän viestissä voi päivittää kerralla 64 lähdön tilaa. Seuraavat lähdöt voisi päivittää toisessa viestissä, mutta myös muut asiat puoltavat 64 lähdön rajoitusta.

Jos lähtöjä olisi tätä enemmän, ne veisivät huomattavan paljon virtaa, ja tämä tulisi ottaa suunnittelussa huomioon järein teholähtein ja jäähdytyksin. Jos laitteeseen on kytketty 64 lähtöä, on myös johdotuksen kannalta järkevää, että seuraavat lähdöt kytketään toiseen laitteeseen, joka voidaan sijoittaa fyysisesti eri paikkaan.

Vaikka tarvittavat laitteet saisi kytkettyä testiympäristössä samaan CAN-väylään, mitä laitteet voisivat ohjata? Kuinka ajoituksen saisi tallennettua ymmärrettävään muotoon testin aikana? Testissä venttiilit eivät oikeasti liiku, eikä niitä ohjaamalla tapahdu mitään, ellei testiympäristö luo jotain tapahtuvaksi.

Jos laitteistolle annetaan testissä esimerkiksi 10 km/h nopeus, ja laitteistossa on useita satoja lähtöjä, joita ohjailemalla ohjelmisto kuvittelee tulostavansa jotain tiettyä pienmerkintää, on todella vaikea kehittää ympäristö, jonka avulla voisimme oikeasti nähdä mitä laite teki käytännössä. Ilman testiympäristöä voimme vain havaita, että laite ohjasi satoja venttiileitä tulostuksen aikana ja ohjatut venttiilit vaikuttivat olevan oikeita.

Testiympäristön tulisi jollain tapaa tallentaa tapahtumat ajallisesti ymmärrettävään muotoon jälkianalysointia varten. Jos kuvassa piirrettäisiin 3 senttimetriä leveä poikittaisviiva, laitteiston tulisi jollain tapaa näyttää visuaalisesti mitä tapahtui. Jos jälkianalyysistä havaittaisiin, että ajanhetkellä x laitteisto ohjasi esimerkiksi 200 venttiiliä päälle, ja 10,8 millisekunnin kuluttua sulki kaikki venttiilit. Voisi ajatella, että laitteisto toimi. Koska laskennallisesti esimerkin 10 km/h nopeudella kolmen senttimetrin etenemiseen kuluisi 10,8 millisekuntia.

Esimerkin 200 ohjattavaa lähtöä olisivat myös eri laitteissa. Kuinka voimme testiympäristössä huomata, että laitteisto ohjasi esimerkin 200 lähtöä päälle, mutta tietyt lähdöt päivittyivät todellisuudessa 400 mikrosekuntia muita myöhemmin. Tällaiset eroavaisuudet tulee testata jollain tapaa. Nimenomaan ajoitukset ovat tässä projektissa ohjelmiston kannalta suurin riski.

Ohjelmisto kuormittuu kovasti, jos laitteistolla ajetaan esimerkiksi 10 km/h ja ohjelmiston on alati analysoitava mitä lähtöjä tulee ohjata, sekä suorittaa tarvittavat toimenpiteet. Lähtöjä voi olla noin 400, ja ajoneuvolla voidaan ajaa esimerkiksi 10 km/h. Laitteiston tulee vaatimusten mukaan kyetä ohjaamaan yhden sentin tarkkuudella kaikkia lähtöjä.

Tämä tarkoittaa, että yhden senttimerin tapahtumia tulisi laitteistolle 3,6 millisekunnin välein. Jokaisen tapahtuman aikana ohjelmiston tulisi kyetä analysoimaan, minkälaisia muutoksia lähtöjen tiloihin tarvitaan ja tarvittaessa ohjaamaan kaikki 400 lähtöä eri tilaan.

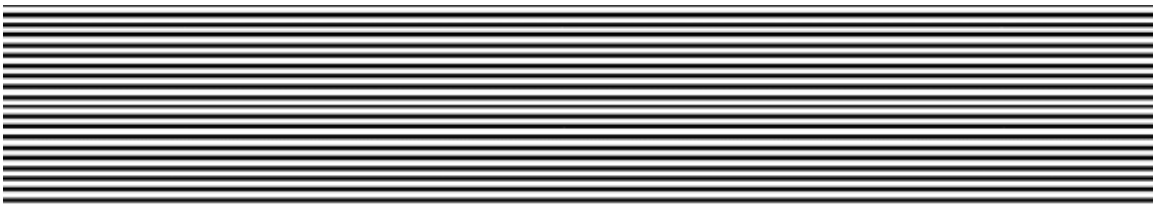
5.5.1 CAN-väylän ajoitustestaus

Laajan ohjelmiston testaaminen tulee tehdä osissa, sillä kaikkia ominaisuuksia ei voi testata yhdellä kertaa. Keskusyksikköön olisi suositeltavaa kytkeä adapterin avulla PC-ohjelma ja mallintaa väylän kaikkia päätelaitteita yhdellä kertaa.

PC-ohjelman tulisi tallentaa kaikki väylään lähetetyt viestit hyvin tarkan aikaleiman kanssa. Tallennetun ajan tulisi olla ainakin 100 mikrosekunnin tarkkuudella todellisesta ajasta, jotta ohjelmiston suorituskyvystä voisi testin avulla tehdä päätelmiä.

CAN-väylään lähetetyistä viesteistä on hieman hankala tulkita tarkalleen mitä on tapahtunut. Olisi kuitenkin resurssien haaskausta tehdä datan tulkitsemiseen erillinen analysointiohjelma, joka näyttäisi ajoitukset jollain tapaa selkeämmässä muodossa. Tässä testityypissä halutaan vain nähdä keskusyksikön suorituskyky, jolla keskusyksikkö kykenee tulkitsemaan tulostettavaa kuvaa ja ohjaamaan kuvan mukaisia lähtöjä oikeisiin tiloihin CAN-väylän kautta.

Kuvassa 21 on suositeltu kuva tähän testaukseen. Kuva on suunniteltu niin, että pienmerkintälaitteisto joutuu ohjaamaan kaikki lähdöt ohjaavaan tilaan samalla senttimetrillä, ja sulkemaan kaikki seuraavalla. Jokainen tila vaihtuu jokaisen senttimetrin jälkeen. Tämä on ohjelmistolle kaikkein raskain mahdollinen kuvio suorittaa.



Kuva 21. Kuva suorituskyvyn testaukseen.

Ohjelmistossa on aina hyvä testata pahin mahdollinen tilanne ohjelmiston kannalta. Tämän avulla voimme tehdä rajoituksia esimerkiksi tulostuksen suurimpaan sallittuun ajonopeuteen.

Testikuvan hyvänä puolena on myös se, että CAN-väylästä on kohtuullisen helppo lukea, että milloin ohjaukset tapahtuvat CAN-väylässä todellisuudessa. Erot ovat selkeitä, sillä tiedetään kuvan perusteella, että ohjaukset tulisivat tapahtua yhtä aikaa, ja ohjaukset koskevat jokaista lähtöä. Tämän vuoksi kaikki poikkeamat ohjausten ajoituksessa, tai datassa, ovat virheitä.

Testaus tulisi suorittaa ensiksi hyvin hitaalla nopeudella, jotta näemme testiympäristön toimivuuden ja samalla laitteiston suorittamat vähimmäisvirheet. Teoriassa virhettä ei saisi tulla muualle kuin CAN-väylän kautta lähtöjen ohjauksessa. Ohjauskomennot menevät väylän eri päätelaitteisiin, ja jonkinlaista eroa ajoituksessa on tällöin oltava.

Laitteistolla tullaan ajamaan todennäköisesti maksimissaan noin 5 km/h ajonopeudella. Laitteistoa on suunniteltu käytettäväksi niin, että ajoneuvo pysähtyy kohdistamaan tulostuksen pienmerkinnän eteen. Tällöin laitteistolla ei ole paljon aikaa eikä tarvetta kiihdyttää kovin rajusti ajoneuvoa ennen tulostuksen alkua.

Kun testi on ajettu hitailla ajonopeuksilla riittävän monta kertaa, ajonopeutta tulisi kasvattaa asteittain seuraaviin testeihin. Vähimmäisvirheiden tiedostamisen avulla voisi tässä vaiheessa myös arvioida, kuinka suurella ajonopeudella laitteiston suorituskyky ei enää riitä, vaan virheiden suuruus alkaa kasvaa.

Ajonopeuksia tulisi kasvattaa asteittain ja katsoa, missä kohtaa ajoitusvirheet alkavat kasvamaan, ja millä tavalla ne kasvavat. Syntyykö tilanne, jossa lähtöjä ohjataan samalla taajuudella, vaikka nopeus kasvaisi? Vai jääkö osa tehtävistä tekemättä?

Tällä testillä pystytään hakemaan laitteiston suorituskyvyn äärirajat, ja vertaamaan niitä laitteiston käyttövaatimuksiin. Mikäli suoritustaso on vaatimuksia selkeästi parempi, voidaan CAN-rajapinnan testaus lopettaa ja luottaa siihen, että ilman uusia ohjelmamuutoksia päivitys kuvasta päätelaitteiden lähtöihin on riittävän tehokas.

5.5.2 Kuvan tulostumisen testaaminen

Ajoneuvon keulasta taaksepäin katsottuna tulostimen osat ovat seuraavassa järjestyksessä: tähtäin, puhdistusventtiilit, massaluukut, helmisuuttimet, vesisuuttimet.

Lähtöjen etäisyys poikkeaa fyysisesti toisistaan niin paljon, että jokaiselle lähdölle on erikseen kyettävä määrittämään fyysinen etäisyys. Jokaisella lähdöllä on myös lähtökohtaisesti yksilöllinen avaus- ja sulkumekanismi, joiden kestot voivat ajallisesti poiketa toisistaan. Jokaiselle lähdölle pitää kyetä säätämään myös oma avausviiveensä/ennakkonsa. Samoin myös sulkuviive/sulkuennakko on kyettävä määrittämään jokaiselle lähdölle erikseen.

Ajallista eroa lähdön toimintaan voisi kutsua viiveeksi, jos ajallinen ero myöhäistäisi tapahtumaa. Mikäli tapahtumaa nopeutetaan, olisi kyseessä ennakko. Jatkossa käytämme tässä työssä termiä viive, vaikka kaikkia tapahtumia voi ennakoida tai myöhäistää tapauskohtaisesti.

Viiveiden toiminta sekä lähtöjen fyysisen paikan poikkeavuuksien testaaminen on kyettävä toteamaan monin eri asetuskokoonpanoin. Tätä varten on tehtävä erillinen testaustila, jotta ominaisuuksien toiminnan voisi todeta.

Testissä on hyvä huomioida, ettei CAN-rajapintaa tarvitse enää testata. Se testataan muiden testien avulla. Tämän vuoksi keskusyksikkö voisi lähettää ethernet-väylään jokaisen lähdön tilan jokaisella senttimetrillä. Lähtöjen tilan voisi lähettää bittitietoina. Viestiliikenteessä voisi sopia, että ensin lähetetään kuumamassaluukkujen tilatiedot, sitten lasihelmisuutinten tilatiedot, ja lopuksi pumppujen, vesisuutinten ja paineilmasuutinten tilatiedot.

PC-ohjelma tietää kunkin suuttimen etäisyyden toisistaan, ja voisi ottaa sen huomioon visuaalisessa esitystavassaan. PC-ohjelma voisi piirtää ohjelman piirtoalueelle rivin kuvaamaan jokaista senttimetriä. Rivin sisällöksi piirrettäisiin suutinten tila.

Tällä tavalla voitaisiin testata erilaisten pienmerkintöjen tulostumista. Ohjelmiston käyttäytymistä voitaisiin seurata visuaalisesti, ja poikkeamat tulostamisessa olisivat selkeästi nähtävissä.

PC-ohjelma voisi tulostaa senttimetrin välein noin 250 pikseliä, jonka tulisi olla joko tyhjä, tai täysi, riippuen kyseisen suuttimen tilasta. Lopuksi päällä olevat lasihelmisuuttimet voisivat muuttaa merkinnän väriä, jotta voitaisiin visuaalisesti havaita mihin ohjelma on laskenut lasihelmiä.

5.5.3 Testitilat kenttätestauksessa

Aiemmissa luvuissa on tullut esille, että koko järjestelmän kattavaa testaamista voidaan suorittaa vasta kun koko ajoneuvo on mekaanisesti ja elektronisesti valmistettu. Tällöin tarvitaan testitiloja, jotta ohjelmiston kehittäjät voivat ymmärtää mahdollisten vikatilanteiden perimmäisiä syitä.

Laitteiston ohjelman tulee toimia kohtuullisen hyvin jo ennen kuin varsinaista tulostusta voidaan valmiilla laitteella kokeilla. Automatiikan on kyettävä lämmittämään kuumamassa ~230 °C lämpötilaan ja pitämään lämpötila vakiona. Kuumamassaa on sekoitettava jatkuvasti sopivalla teholla. Ohjelmiston tulee ohjata monia eri tekijöitä, jotta varsinaista toimintaa on mahdollista testata.

Kun laitteella kokeillaan tulostaa pienmerkintöjä tien pintaan, joitain asioita menee todennäköisesti pieleen. Tuloksesta on monissa tilanteissa vaikea suoraan sanoa, mikä meni pieleen. Lopputulos on yleensä lukuisten toimintojen summa, jossa toimintoja tapahtuu mekaniikassa, ohjelmissa, venttiileissä, pumpuissa sekä monissa muissa asioissa. Vaikka kaikki toiminnot olisivat oikeita, myös toimintojen ajoitukset ovat ensisijaisen tärkeitä.

Ohjelmiston kehittäjien tulee pystyä kenttätestauksen aikana ymmärtämään ohjelmiston toiminta, jotta vikojen jäljille päästäisiin mahdollisimman nopeasti. Ohjelmiston toimintaan voidaan päästä käsiksi helpoiten kytkemällä kannettava tietokone keskusyksikköön. Toinen vaihtoehto on ajoneuvossa olevan PC:n käyttäminen testaamiseen. Testitiloja kannattaa varmasti luoda molemmille tavoille.

Valmiin laitteiston saattaminen testaustilaan on pitkäkestoinen prosessi, laitteiston lämmittäminen suorituslämpötilaan vie useita tunteja. Tämän vuoksi laitteiston tulostuskokeiluista tulisi kerätä kaikki mahdollinen informaatio talteen, jotta testien lukumäärä voitaisiin minimoida.

Paras tilanne olisi, että laitteistoon laitettaisiin tallennustila päälle, tehtäisiin testit, ja ladattaisiin tallennetut tiedot kannettavaan tietokoneeseen jälkikäteen. Hyvin tehdystä testitilasta olisi mahdollista nähdä myös tapahtumien ajoitukset, jotka ovat hyvin kriittisiä, millisekunnin tarkkuudella.

On kuitenkin huomioitavaa, ettei sulautetun järjestelmän ohjelmistoon haluta yleensä tehdä hyvin laajaa muistialuetta, johon voisi tallentaa väliaikaisesti testien tuloksia. Toisin kuin PC-ohjelmissa, sulautetun järjestelmän ohjelmissa täytyy muistin käyttöön ja ohjelmakoodin tilaan kiinnittää huomattavasti enemmän huomiota niiden rajallisuuden vuoksi.

Tämä ei tarkoita, etteikö testitiloja voisi tehdä, vaan että ne tulisi tehdä mielekkäällä ja tehokkaalla tavalla. Jos laitteistolla tulostettaisiin kuvaa 21 nopeudella 5 km/h, kaikkien lähtöjen tiloja vaihdettaisiin jokaisella senttimetrillä. Lähtöjen määrän ollessa esimerkiksi 350 tulisi laitteiston suorittaa 48611 lähdön ohjausta sekunnissa. Tietomäärä on suuri, ja jokaisen tapahtuman tarkka ajoituksen jäljitys olisi myös erittäin tärkeää.

Kaikki tieto voi olla jossain tilanteessa tärkeää nähdä laitteen kokeilun yhteydessä. Testitilassa tulisi olla asetuksia, joiden avulla voidaan nopeasti suodattaa testattavia asioita. Näin tietomäärä saataisiin pidettyä maltillisena ja ohjelmoijat voivat testata asioita, joita pitävät kyseisellä hetkellä tärkeinä. Testaustilat on saatava myös pois päältä nopeasti, sillä testien suorittaminen vaatii keskusyksikön prosessorilta resursseja, jotka muutoin voisivat olla varsinaisen tulostamisen käytössä.

6 OHJELMISTOKEHITYSMENETELMÄN VALINTA

McConnell (2002) erittelee jokaisen ohjelmointimallin kyvyn vastata kuhunkin tarpeeseen. Hän ei kuitenkaan käsittele ketteriä menetelmiä kirjassaan lainkaan. Tämän vuoksi scrum-, ja eXtreme Programming-menetelmien toimivuuksien arviointi on lisätty testiin jälkikäteen.

Seuraavaksi käydään jokainen McConnellin esittämä ominaisuus läpi kyseessä olevan projektin näkökulmasta. Vastausten avulla pyritään löytämään muutamia sopivilta vaikuttavia ohjelmistomenetelmiä tarkempaa vertailua varten.

6.1 McConnell (2002) menetelmän valintatesti

McConnell (2002) kehottaa vastaamaan luvussa 3.3 esitettyihin 11 asiaan kohdeprojektin näkökulmasta. Testin avulla pyritään hahmottamaan tarkemmin, mitkä ohjelmistokehitysmenetelmät olisivat hyviä vaihtoehtoja käytettäväksi kohdeprojektissa. Tarpeita arvostellaan viisiportaisella asteikolla: *Heikko*, *Heikosta kohtuulliseen*, *Kohtuullinen*, *Kohtuullisesta erinomaiseen* sekä *Erinomainen*.

McConnell (2002) menetelmän valintakysymyksillä on kuitenkin keskenään eritasoisia tärkeyseroja. Toiset kysymykset ovat selkeästi tärkeämpiä kohdeprojektin näkökulmasta kuin toiset. Jotta tämä otettaisiin huomioon lopullisessa pisteytyksessä, jokaiselle kohdalle arvioidaan prioriteetti asteikolla 1-5. Luku 5 tarkoittaa tärkeintä prioriteettia, ja luku 1 vastaavasti tarkoittaa, että asia arvioidaan vähiten tärkeäksi. Jokaisen kohdan prioriteetti kerrotaan suoraan lopulliseen pisteytykseen luvussa 6.1.11.

6.1.1 Toimivuus tutkimuksellisessa ohjelmistokehittämisessä

Kuinka hyvin kehitysmalli toimii sellaisissa projekteissa, joissa kukaan ei ole täysin tietoinen laitteiston vaatimuksista. Tämä ennakoi sitä, että vaatimukset tulevat pro-

jektin etenemisen yhteydessä todennäköisemmin muuttumaan. Tällaista ohjelmistoprojektia kutsutaan usein tutkimukselliseksi ohjelmistokehittämiseksi, jolloin projektissa on paljon uutta. (McConnell, 2002, s.157)

Kohdeprojektissa tehdään pienmerkintöjä valmistavaa tulostinta ammattikäyttöön. Laitteen käyttäjät ovat tiemerkintäalan urakoitsijoita, jotka ovat aiemmin tehneet pienmerkinnät käsikäyttöisillä laitteilla.

Kokeneilla pienmerkitsijöillä on hyvin tiedossa, mitä laitteella tulisi pystyä tekemään. Käyttötapa on kuitenkin heillekin täysin uusi, kaikki operoinnit haluttaisiin tehdä suoraan kuorma-auton hytistä. Kilpailevat laitteistot, jotka tekevät myöskin pienmerkintöjä kuorma-auton hytistä, operoivat toistaiseksi vain ulkomailla ja suomalaisilla urakoitsijoilla ei ole näistä laitteista kokemusta.

Urakoitsijoilla on siis hyvä käsitys siitä, minkälaisia lopputuloksia laitteistolla tulisi saada aikaiseksi, mutta kaiken oleellisen tiedon siirtäminen tuotekehitykselle on usein vaikeaa. Asioita voidaan väärinymmärtää, unohtaa mainita, tai jotkin asiat voivat tuntua olevan liian itsestäänselviä mainittaviksi. Tilannetta pahentaa se, ettei tuotekehityksellä ole käytännön kokemusta pienmerkintöjen tekemisestä.

Kohdeprojektissa tehdään täysin uusi laitteisto, jolloin eri käyttötapojen hyvät ja huonot puolet ovat kaikille vaikeita hahmottaa. Tutkivassa tuotekehityksessä myös ohjelmistokehitykseen käytetty menetelmä tulisi olla sellainen, että menetelmä toimii hyvin ”huonosti ymmärrettyjen vaatimusten” kanssa.

Kun kyseinen vaatimus on näin tärkeässä roolissa tässä projektissa, voidaan vaatimukselle antaa korkeimman luokituksen McConnell (2002) testissä *erinomainen.* (Taulukko 8)

Taulukossa 8 on listattu eri menetelmien kykenevyys toimia huonosti ymmärrettyjen vaatimusten projektissa. Mikäli vaatimus vastaa projektin odotuksia, se on merkitty vihreällä. Kun menetelmä ei vastaa tavoiteltua tasoa, sävy on punainen. Mitä vaaleampi punainen sävy on, sitä vähemmän tavoitellusta tasosta poiketaan. Eli jos vaatimus on korkealla, mutta menetelmä ei kykene palvelemaan tätä ominaisuutta kovinkaan hyvin, merkitään ristiriita tumman punaisella.

Taulukko 8. Toimivuus muuttuvien vaatimusten kanssa.

Kehitysmenetelmä	Toimii huonosti ymmärrettyjen vaatimusten kanssa
Puhdas vesiputous	Heikko
Spiraali	Erinomainen
Muunnetut vesiputoukset	Kohtuullisesta erinomaiseen
Evolutiivinen protoilu	Erinomainen
Vaiheistettu toimitus	Heikko
Evolutiivinen toimitus	Kohtuullisesta erinomaiseen
SCRUM	Erinomainen
eXtreme programming (XP)	Erinomainen

Scrum ja XP luokitellaan ketteriin menetelmiin. Ketterät menetelmät kehitettiin nimenomaan vastaamaan tarpeita projekteissa, joissa vaatimukset muuttuvat usein (Beck & Andres, 2005; Koch, 2004). Näissä menetelmissä myös asiakkaan tiivis läsnäolo projektin edetessä auttaa muuttuvien vaatimusten huomaamista mahdollisimman nopeasti. Tämän vuoksi luokittelin näiden menetelmien toimivuuden huonosti ymmärrettyjen vaatimusten kanssa luokkaan *Erinomainen*.

McConnell (2002) arvioi evolutiivisen protoilun, ja spiraali-menetelmän sopivan parhaiten projekteihin, joissa vaatimukset ovat alttiita muutoksille. (McConnell, 2002, s.156 - 157)

Kohdeprojekti voidaan laskea tutkimukselliseksi tuotekehitysprojektiksi, jossa ohjelmistovaatimukset eivät tule olemaan täysin selvillä. Tämän vuoksi tämän kohdan prioriteetti on luku 5. On erittäin tärkeää, että ohjelmistokehitysmenetelmä tukee projektin tarpeita tällä osa-alueella.

6.1.2 Arkkitehtuurisesti uusia asioita

Kohdeprojektissa tehdään kokonaan uusi laitteisto, joka tulee olemaan osa RMD-tuoteperhettä, ja monet tuoteperheessä käytetyt teknologiat vaikuttavat olevan

hyödynnettävissä myös kohdeprojektissa. Prosessorit ja ohjelmointiympäristöt pyritään valitsemaan niin, että ne ovat tehokkuudeltaan riittäviä kohdeprojektiin, mutta kuitenkin tuotekehitykselle tuttuja.

Kun projekti jaetaan pieniin osiin, ovat ohjelmistotehtävät tuttuja aiemmista projekteista yrityksen tuotekehitykselle. Ohjelmistotehtävien muodostama järjestelmä on täysin erilainen kuin aiemmin tehdyt järjestelmät, mutta kuitenkin koostuu tutuista toteutuksista.

Ohjelmointiin valitut ohjelmointikielet (C ja C++) ovat yrityksen tuotekehitykselle tuttuja aiemmista projekteista. Kaikkiin suunniteltuihin prosessoreihin on tehty ainakin yksi aikaisempi projekti, eli ohjelmointiympäristö ja prosessorien arkkitehtuurit ovat tuotekehitykselle tuttuja.

Luvussa *4.3 Asiakasvaatimukset* havaittiin, että tulostuksen kohdistaminen tulisi todennäköisesti tehdä kameroiden avulla. Vaatimus on kokonaisuuden kannalta merkitsevä, ja sen käytettävyys tulee olla hyvä. Kameroiden kuvien analysointi paikkatiedoksi, ja tulostuskuvien esikatselu kameroiden näkökentässä ovat kriittisiä ominaisuuksia ja meille täysin uutta. Ohjelmistokehitysmenetelmän vaatimukseksi annetaan tämän vuoksi *Kohtalainen*. (Taulukko 9)

Taulukko 9. Uudenlaista arkkitehtuuria.

Kehitysmenetelmä	Toimii huonosti ymmärretyn arkkitehtuurin kanssa
Puhdas vesiputous	Heikko
Spiraali	Erinomainen
Muunnetut vesiputoukset	Kohtuullisesta erinomaiseen
Evolutiivinen protoilu	Heikosta kohtuulliseen
Vaiheistettu toimitus	Heikko
Evolutiivinen toimitus	Heikko
SCRUM	Heikko
eXtreme programming (XP)	Heikko

Spiraalimalli ja muunnetut vesiputoukset toimivat vaadittua paremmin heikosti tunnetun arkkitehtuurin näkökulmasta. Ketterille menetelmille (scrum ja XP) arvioin samanlaisen toimivuuden heikosti ymmärrettyjen arkkitehtuurien näkökulmasta kuin vaiheistetussa toimituksessa. Ketterät menetelmät eivät mielestäni sisällä toimintoja, jotka auttaisivat kohdeprojektia toimimaan heikosti ymmärretyn arkkitehtuurin projekteissa vaiheistettua toimitusta paremmin.

Koska kohdeprojektissa on vain vähän arkkitehtuurisesti uusia asioita, luokitellaan osa-alueen prioriteetiksi 2. Ei ole ensisijaisen tärkeää, että kohdeprojektissa käytettäisiin menetelmää, jolla on hyvät edellytykset toimia heikosti tunnettujen arkkitehtuurien parissa.

6.1.3 Luotettavan järjestelmän tuottaminen

Ohjelmiston luotettavuus on erittäin tärkeää kaikissa projekteissa. Luotettavuuden tarve toimitushetkellä riippuu esimerkiksi toimituksen laajuudesta, tällöin luotettavuuden tarve voi olla hyvinkin poikkeava. Toisessa projektissa voidaan käyttää ohjelmistoa tuhansissa kuluttajille myytävissä laitteissa. Ohjelmiston käyttöönoton jälkeen virheiden korjaus tulee kalliiksi, ohjelmisto voi olla levinnyt jo tuhansille kuluttajille. Tässä tapauksessa ohjelmiston luotettavuus käyttöönottohetkellä on tärkeää.

Esimerkiksi avaruusrakettien ohjelmistoissa maksetaan hurjia summia siitä luotettavuudesta, että ohjelmistossa ei varmasti ole virheitä käyttöönottovaiheessa. Tämä on hyvin ymmärrettävää, sillä viat voisivat olla hyvin kohtalokkaita.

Kohdeprojektissa on tarkoitus ottaa laitteisto koekäyttöön yhdellä asiakkaalla. Koekäytössä asiakkaan ja tuotekehityksen välinen kanssakäyminen on jokapäiväistä, ja kehitystä tehdään nopeasti saadun palautteen mukaan.

Ohjelmistosta pitää saada luotettava, mutta käyttöönottovaiheen luotettavuus ei ole projektin tärkeimpiä ominaisuuksia. Annan kohdeprojektin prioriteetiksi 2 ja

vaatimukseksi arvosanan *Heikko*. Taulukossa 10 on verrattu kohdeprojektin tarvetta eri ohjelmistokehitysmenetelmien tarjoamiin ominaisuuksiin. Koska vaatimus on alin mahdollinen (*Heikko*), niin kaikki menetelmät täyttävät tarpeen.

Taulukko 10. Luotettavuus eri ohjelmistokehitysmalleilla.

Kehitysmenetelmä	Tuottaa luotettavan järjestelmän
Puhdas vesiputous	Erinomainen
Spiraali	Erinomainen
Muunnetut vesiputoukset	Erinomainen
Evolutiivinen protoilu	Kohtuullinen
Vaiheistettu toimitus	Erinomainen
Evolutiivinen toimitus	Kohtuullisesta erinomaiseen
SCRUM	Erinomainen
eXtreme programming (XP)	Erinomainen

Jokaisen ohjelmistokehitysmenetelmän tarjoama luotettavuus on hyvää tasoa. Evolutiivinen toimitus ja evolutiivinen protoilu ovat ainoita menetelmiä, jotka eivät saaneet parasta arvosanaa. Kuitenkin nämä menetelmät ovat sellaisia, joiden avulla tuotekehitys varmistaa, että jonkinlainen toimiva versio on varmasti toimitettavissa, kun versiota tarvitaan.

Esimerkiksi puhtaalla vesiputousmallilla voi helposti käydä niin, että projektin kohdatessa ongelmia tuotekehityksellä ei ole minkäänlaista toimivaa versiota valmiina, kun toimituspäivämäärä koittaa. Mutta onnistuessaan tämä menetelmä takaa, että version luotettavuus olisi kohdallaan.

6.1.4 Järjestelmän kasvunvara

Kaikki testissä mukana olevat menetelmät saivat kyseisestä vaatimuksesta parhaan mahdollisen arvosanan. Ketterät menetelmät on kehitetty sopeutumaan

muuttuviin vaatimuksiin ja tilanteisiin. Ketterät menetelmät ovat hyvin varautuneet muutoksiin, joissa tuotteen elinkaaren aikana esimerkiksi tuotteen laajuuden tarve muuttuu. Muiden menetelmien ohella myös ketterät menetelmät sopeutuvat uusiin muuttuviin tilanteisiin erinomaisesti, vaikka tuotekehitystiimi ei osaisikaan odottaa muutoksia.

Kohdeprojektissa vaaditaan menetelmää, jota käyttämällä saadaan luotua kasvunvaraa omaava järjestelmä. Kun kyseessä on uusi laitteisto, asiakastarpeiden muuttuminen on hyvin todennäköistä. Tämä vaikuttaa helposti myös projektin kokoon ja monipuolisuuteen. Prioriteetiksi asetetaan tämän vuoksi korkein mahdollinen, 5.

Luvun 4. case-osuudessa pystyimme havaitsemaan, että tiemerkintäalalla vaatimukset ja säännökset muuttuvat usein myös töiden tilaajien taholta. Tämä vaikuttaa suoraan urakoitsijoiden laitevaatimuksiin laitetoimittajalta, eli meiltä. Kohdeprojektin kasvunvaran tarve kuuluu luokkaan *Erinomainen* (Taulukko 11)

Taulukko 11. Projektin kasvunvara eri menetelmillä.

Kehitysmenetelmä	Tuottaa järjestelmän, jossa on hyvin kasvuvaraa
Puhdas vesiputous	Erinomainen
Spiraali	Erinomainen
Muunnetut vesiputoukset	Erinomainen
Evolutiivinen protoilu	Erinomainen
Vaiheistettu toimitus	Erinomainen
Evolutiivinen toimitus	Erinomainen
SCRUM	Erinomainen
eXtreme programming (XP)	Erinomainen

Kaikki menetelmät täyttävät kohdeprojektin tarpeen taulukossa 11, tällöin pohdittava kysymys ei vaikuta ohjelmistokehitysmenetelmän valintaan.

6.1.5 Riskien hallitseminen

Kuinka hyvin menetelmä tarjoaa projektille mahdollisuuden tunnistaa ja hallita erilaisia riskejä projektin kehityksen aikana? Tuotekehitys on hyvin riskialtista, pahimpien riskien toteutuminen voi olla yritykselle jopa kohtalokasta ja seuraukset pitkäkestoisia.

Alustavan laskelmani mukaan projektiin menee noin 30 työkuukautta. Projekti on siis laajuudeltaan kohtuullisen suuri alle 20 hengen yritykselle. Asiakas on jo ostanut projektiin kuorma-auton, johon mekaniikkaa työstävä yrityskumppanimme on jo tehnyt paljon mekaanista työtä.

Pahimmillaan elektronisten toteutusten suunnittelussa tulisi perustavanlaatuinen virhe, esimerkiksi riittämätön suorituskyky ja liian hidas tiedonsiirto eri yksiköiden välillä. Tällaisten syiden vuoksi koko elektroniikka tulisi suunnitella uudestaan ja ohjelmistoihin tulisi tehdä suuria muutoksia.

Tutkiva tuotekehitys on hyvin riskialtista. Kun järjestelmä sisältää samassa verkostossa esimerkiksi yli 10 älyllistä sulautettua elektroniikkayksikköä, kokonaisuuden toimintaa on mahdotonta testata täydellisesti samoin kuin oikeassa toimintaympäristössä.

On mahdollista, että yksikkötestausvaiheessa kaikki näyttää vielä hyvältä, mutta kokonaisuutta testatessa huomataan, että uuden tiedon valossa jokin ohjelmisto onkin huonosti suunniteltu ja toteutettu, ja on rakennettava kokonaan uudestaan.

Kun tehdään laitteistoja, joista ei ole aikaisempaa kokemusta, on kyseessä tutkiva tuotekehitys. Kun tuotekehityksellä ei ole käytännön kokemusta varsinaisesta tiemerkintöjen tekemisestä, kaikki informaatio on tultava asiakkaalta, tavalla tai toisella. Kohdeprojektin kaltaiset hankkeet ovat kohdeyritykselle kaikkein riskialtteimpia, jolloin riskienhallinta on erittäin merkittävässä roolissa.

Luvussa *5.3 Mahdolliset riskit* havaitsimme kuinka vakavia seurauksia riskien toteutumisella voisi olla. Tämän vuoksi kohdeprojektin tarve ohjelmistokehitysmenetelmälle, jonka avulla voidaan minimoida ja välttää riskejä, on luokassaan *Erinomainen* (taulukko 12). Prioriteetiksi asetetaan 5, tärkein mahdollinen.

Taulukko 12. Riskienhallinta eri menetelmillä.

Kehitysmenetelmä	Riskienhallinta
Puhdas vesiputous	Heikko
Spiraali	Erinomainen
Muunnetut vesiputoukset	Kohtuullinen
Evolutiivinen protoilu	Kohtuullinen
Vaiheistettu toimitus	Kohtuullinen
Evolutiivinen toimitus	Kohtuullinen
SCRUM	Kohtuullisesta erinomaiseen
eXtreme programming (XP)	Kohtuullisesta erinomaiseen

McConnell (2002) arvioi, että ainoastaan spiraalimallilla on erinomaiset riskienhallinta ominaisuudet. Muunnettuja vesiputousmalleja on useita erilaisia. Niistä yksi on kuitenkin spiraalimallilla alkava vesiputous, jonka riskienhallintaominaisuudet pitäisi olla muita vesiputousmalleja parempi.

McConnell (2002) ei sisällyttänyt testiin scrum-, ja eXtreme Programming-menetelmiä, ja arvioin niiden riskienhallintaominaisuuksien olevan luokkaa *Kohtuullinen*.

Ketterien menetelmien luonteeseen ja rakenteeseen on integroitu riskienhallintaelementtejä. Riskienhallinta ketterissä menetelmissä perustuu siihen, että muutoksiin pystytään reagoimaan nopeasti. Iteratiivinen kehitysmalli tarjoaa myös mahdollisuudet esimerkiksi riskianalyyseille jonkin iteraation alussa. (Tapio, 2010, s. 62-63)

Koska tässä testissä pyritään vertailemaan eri ohjelmistokehitysmalleja objektiivisesti, ketterät menetelmät eivät tarjoa parempaa riskienhallintaa kuin testin muut menetelmät. Kuitenkin ketterien menetelmien riskienhallinta on huomattavasti parempi kuin puhtaalla vesiputousmallilla. Ketterät menetelmät eivät ota kantaa projektin riskienhallintaan, kuitenkin ketterillä menetelmillä voi suorittaa projektin läpi hyvällä (tai huonolla) riskienhallinnalla.

Vaikka menetelmää voisi käyttää riskienhallinnan kannalta tehokkaasti osaavissa käsissä, ei se tee itse menetelmän riskienhallintaominaisuuksista hyvää. Tällöin menetelmän käyttö on riskienhallinnallisesti hyvää ja sama pätee kaikkiin ohjelmistokehitysmalleihin.

Antaessani ketterille menetelmille toiseksi parhaan arvosanan riskienhallinnassa *Kohtuullisesta erinomaiseen*, tein sen sillä oletuksella, että tuotekehitys ymmärtää projektin riskien merkityksen ja käyttäisi menetelmää riskipainotteisesti. Tällöin menetelmän riskienhallinta olisi hyvällä tasolla. Monissa menetelmissä tällaista joustavuutta ei ole, jolloin arviontikin on helpompaa.

6.1.6 Sopivuus joustamattomissa aikatauluissa

Kohdeprojektin aikataulu ei ole sidottu tiettyyn päivään. Asiakas toivoo, että laitteistoa voitaisiin käyttää tulevana kesänä, kun tiemerkintöjä tehdään. Jos kesän kuukausista esimerkiksi puolet menisi hukkaan sen vuoksi, ettei laitteistolla pystytä tekemään vaadittuja töitä, olisi tilanne kaikille osapuolille pettymys.

Tavoitteena olisi saada laitteisto mahdollisimman nopeasti sellaiseen kuntoon, että sillä voidaan tehdä töitä, ja tällöin asiakas saa laitteiston koekäytöstä täysimääräistä hyötyä. Tämän jälkeen kehitystyötä jatketaan, kunnes asiakas on laitteiston toimintaan tyytyväinen ja alkuperäiset asiakasvaatimukset täyttyvät.

Käytännössä tämä tarkoittaa sitä, että kaikki ominaisuudet, jotka eivät estä laitteiston peruskäyttöä, tehdään matalammalla prioriteetilla kuin muut ominaisuudet. Näin varmistetaan se, että laitteiston perusominaisuudet varmasti toimivat myös alkukesästä.

Aikataulu on moniin ohjelmistoprojekteihin verrattuna joustava päivämäärien sekä toimintojen ominaisuuksien suhteen. Tämän vuoksi kohdeprojektissa ei tulisi kohdentaa ylimääräisiä resursseja varmistamaan sitä, että projekti pysyy päivälleen oikeassa aikataulussa, sillä tähän ei ole tarvetta. Kohdeprojektin tarve sellaiselle

ohjelmistokehitysmenetelmälle, joka tuottaa ohjelmiston joustamattomassa aikataulussa on *Heikko* (taulukko 13). Prioriteetti asetetaan kohdeprojektin vähäisen tarpeen vuoksi alimpaan tasoon, 1.

Taulukko 13. Sopivuus joustamattomissa aikatauluissa

Kehitysmenetelmä	Toimivuus joustamattomissa aikatauluissa
Puhdas vesiputous	Kohtuullinen
Spiraali	Kohtuullinen
Muunnetut vesiputoukset	Kohtuullinen
Evolutiivinen protoilu	Heikko
Vaiheistettu toimitus	Kohtuullinen
Evolutiivinen toimitus	Kohtuullinen
SCRUM	Erinomainen
eXtreme programming (XP)	Erinomainen

Ketterät menetelmät (scrum ja XP) luokiteltiin luokkaan *Erinomainen,* kun kyse on aikataulujen pitävyydestä. Ketterissä menetelmissä kaikki tehtävät jaetaan pieniin palasiin, joiden vaatima työmäärä arvioidaan jokaiselle tehtävälle erikseen. Pienten tehtävien arvioiminen on helpompaa, ja kokenut kehitystiimi on siinä erittäin tarkka. Näitä menetelmiä käyttämällä projektin kokonaiskeston arviointi on tarkkaa ja helppoa.

Ketterien menetelmien aikataulujen tarkoissa laskuissa tarvitaan kuitenkin kokemusta aiemmista projekteista. Tuotekehitystiimi on tarkka arvioimaan pienten tehtävien vaatiman työmäärän. Kokemuksen tulisi antaa tälle kertoimen käytettäväksi oikeana työaikana. Kerroin muuttuu myös projektin edistyessä, työmäärät tehtävissä pysyvät samoina, mutta todellinen kesto työaikana pitenee.

6.1.7 Minimaalinen lisätyön määrä

Kohdeprojektin näkökulmasta menetelmän aiheuttama lisätyö tulisi minimoida. Tässä luvussa vertaillaan menetelmiä sillä oletuksella, että menetelmän käyttäminen on jo ennestään tuotekehitykselle tuttua.

Tämän vuoksi arvioitiin, että ketterät menetelmät (scrum ja XP) sisältävät minimaalisen määrän ylimääräistä työtä. Menetelmiä kehitettäessä on pyritty minimoimaan kaikki toiminnot, joilla itsessään ei ole asiakkaalle arvoa. Kohdeprojektin tarve minimaaliselle lisätyölle on *Erinomainen*, sillä aikaa ei ole tuhlattavaksi ylimääräiseen työhön (taulukko 14). Koska tehokas menetelmä auttaa tehokkaampaan suoritukseen, ja tätä kautta supistaa itsessään aikataulupaineita, ei prioriteetiksi voida tämän perusteella antaa korkeinta mahdollista arvoa. Lisätyön määrän minimointi on kuitenkin tärkeää, ja sen prioriteetiksi asetetaan 3.

Taulukko 14. Sisältää vain vähän lisätyötä

Kehitysmenetelmä	Sisältää vain vähän lisätyötä
Puhdas vesiputous	Heikko
Spiraali	Kohtuullinen
Muunnetut vesiputoukset	Erinomainen
Evolutiivinen protoilu	Kohtuullinen
Vaiheistettu toimitus	Kohtuullinen
Evolutiivinen toimitus	Kohtuullinen
SCRUM	Erinomainen
eXtreme programming (XP)	Erinomainen

Ohjelmistokehitysmenetelmistä ainoastaan muunnetut vesiputoukset -malli, scrum- sekä XP -menetelmät täyttävät kohdeprojektin asettaman vaatimustason. Puhdas vesiputous -malli vastasi tarpeeseen huonoiten.

6.1.8 Suunnanvaihdosten salliminen

Kuinka hyvin menetelmä sallii tehdä merkittäviä muutoksia tuotteeseen projektin puolivälissä? Tällä kysymyksellä pyritään löytämään menetelmien erot, kun tuotteessa on tehtävä merkittäviä muutoksia kesken projektin. Merkittävillä muutoksilla tarkoitetaan yleensä projektin laajentamista, ei varsinaisen perustarkoituksen muuttamista.

Kohdeprojektissa voi tulla projektin kokoon merkittäviä muutoksia. Kun kyseessä on tutkiva tuotekehitys, asiakaskaan ei välttämättä heti hahmota miten laitteiston tulisi optimaalisesti toimia. Uusia ideoita tulee varmasti projektin edetessä. Tähän olisi hyvä varautua, jotta voisimme joustaa uusien ideoiden edessä mahdollisimman tehokkaasti. Taulukossa 15 on esitetty eri menetelmien sopeutuminen projektin suunnanmuutoksiin. Taulukossa on arvioitu kohdeprojektin tarpeen suunnanmuutoksille olevan luokkaa *Erinomainen*.

Prioriteetiksi arvioidaan kohdeprojektissa 4, sillä suunnanvaihdoksia on todennäköisesti tulossa ja tämän vuoksi osa-alue on tärkeä. Kuitenkin osa-aluetta ei voida nähdä kohdeprojektin kannalta yhtä tärkeänä kuin esimerkiksi riskienhallintaa.

Taulukko 15. Suunnanmuutosten sallinta.

Kehitysmenetelmä	Sallii suunnanmuutoksia
Puhdas vesiputous	Heikko
Spiraali	Kohtuullinen
Muunnetut vesiputoukset	Kohtuullinen
Evolutiivinen protoilu	Erinomainen
Vaiheistettu toimitus	Heikko
Evolutiivinen toimitus	Kohtuullisesta erinomaiseen
SCRUM	Kohtuullisesta erinomaiseen
eXtreme programming (XP)	Kohtuullisesta erinomaiseen

Scrum ja XP toimivat mielestäni yhtä hyvin kuin evolutiivinen toimitus, kun kyse on suunnanmuutoksista projektin aikana. Evolutiivisen toimituksen toiseksi paras arvosana *Kohtuullisesta erinomaiseen* tulee siitä, että projektissa pyöritetään pientä kehityskierrosta: kehitä versiota, toimita versio asiakkaalle, kerää palaute, yhdistä palaute versioon. Kuvatun kaltainen sykli on aika joustava muutoksiin.

Ketterät menetelmät toimivat hyvin samalla tavalla tässä mielessä. Menetelmissä luodaan versio aina lyhyiden iteraatioiden päätteeksi ja toimitetaan se asiakkaalle. Tämän jälkeen reagoidaan palautteeseen ja lähdetään kehittämään uutta iteraatiota. Suunnanmuutosten näkökulmasta ketterien menetelmien toiminta vastaa evolutiivista toimitusta, joten ne saavat saman arvosanan.

6.1.9 Projektin edistymisen näkyvyys

Kappaleessa yhdistetään kaksi lähes samanlaista kehitysmenetelmän ominaisuutta. Kyseessä on kysymys siitä, kuinka hyvin kehitysmenetelmä luo automaattisesti näkyvyyttä projektin edistymisestä asiakkaalle ja hallinnolle.

Kohdeprojektissa asiakas on eniten kiinnostunut projektin edistymisestä vasta käyttöönottovaiheen lähestyessä. Kun kyseessä on pieni projekti (3 tuotekehittäjää), hallinnon on kohtuullisen helppo saada kuva projektin tilasta eri vaiheissa. Samalla tavoin pienen tuotekehitystiimin on helppo hahmottaa projektin edistyminen eri vaiheissa. Vastuu aikataulussa pysymiseen on tuotekehityksellä ja mahdollisista muutoksista tuotekehitystiimi ilmoittaa hallinnolle ja asiakkaalle. Kohdeprojektin tarve menetelmän tarjoamille edistymisen näkyvyydelle on *Heikko*.

Suurempi näkyvyys ei toisi kohdeprojektiin lisäarvoa. Tämä pitää paikkansa, mikäli tuotekehitys päättää uusien ominaisuuksien lisäämisestä tuotteen eri versioihin, kuten kohdeprojektissa on suunniteltu tehtävän. Taulukossa 16 on esitelty eri menetelmien luoma näkyvyys hallinnolle ja asiakkaalle. Kohdeprojektin tarve molempiin kysymyksiin on *Heikko*. Sekä asiakkaalle, että hallinnolle kohdistunut projektin etenemisen näkyvyys alkuvaiheessa voidaan arvioida prioriteetiksi 2. Prioriteettiarvolla ei kuitenkaan ole merkitystä menetelmän valintaan, sillä kaikki menetelmät ylittävät kohdeprojektin tarpeen taulukossa 16.

Taulukko 16. Projektin etenemisen näkyvyys eri menetelmillä.

Kehitysmenetelmä	Tarjoaa asiakkaalle edistymismerkkejä	Tarjoaa hallinnolle edistymismerkkejä
Puhdas vesiputous	Heikko	Kohtuullinen
Spiraali	Erinomainen	Erinomainen
Muunnetut vesiputoukset	Kohtuullinen	Kohtuullisesta erinomaiseen
Evolutiivinen protoilu	Erinomainen	Kohtuullinen
Vaiheistettu toimitus	Kohtuullinen	Erinomainen
Evolutiivinen toimitus	Erinomainen	Erinomainen
SCRUM	Erinomainen	Kohtuullisesta erinomaiseen
eXtreme programming (XP)	Erinomainen	Kohtuullisesta erinomaiseen

Scrum ja XP menetelmät luovat hyvät puitteet näkyvyydelle. Menetelmien tarjoamat versiot jokaisen iteraation päätteeksi luovat näkyvyyttä asiakkaalle. Menetelmien asiakaslähtöisyys ja asiakkaan kanssa tiiviissä yhteydessä olo koko projektin kehityksen ajan lisää myös omalta osaltaan näkyvyyttä.

Menetelmien tarjoamien versioiden ja palaverien tulisi tarjota myös hallinnolle näkyvyyttä projektin etenemisestä. Ketterät menetelmät kuitenkin keskittyvät enemmän näkyvyyteen asiakkaalle kuin hallinnolle.

6.1.10 Menetelmien vaatima erikoisosaaminen

Kuinka paljon erikoistumista vaaditaan hallinnolta ja tuotekehitykseltä ohjelmistokehitysmallin tehokkaaseen käyttöön? Erikoistumisella tarkoitetaan koulutusta ja kokemusta kyseisen kehitysmallin käyttämiseen.

XP-menetelmä poikkeaa runsaasti perinteisiin ohjelmointitapoihin tottuneista menetelmistä. Menetelmä ottaa kantaa siihen, kuinka varsinainen ohjelmointi tulisi suorittaa. ”Testaus ensin” -ohjelmointi, pariohjelmointi, sekä käyttäjätarinoiden käyttö vaativat runsaasti totuttelua. Beck & Anders (2005) arvelevat menetelmän

vaativan tyypillisesti noin kaksi vuotta totuttelua ennen kuin tuotekehitys saa menetelmästä kaiken tehon käyttöön.

Scrum-menetelmän tehokkaaseen käyttöön suositellaan koulutusta tietyille avainhenkilöille. Scrum Master sekä Scrum Product Owner rooleihin tarjotaan koulutusta. Tuotekehitystiimille scrum-menetelmän käyttö on huomattavasti XP-menetelmää helpompaa omaksua, sillä scrum ei ota kantaa siihen, kuinka varsinainen ohjelmointityö tehdään. Scrum onkin projektinhallintamenetelmä, ja ohjelmistoprojektissa scrumin lisäksi voidaan tarvita joitain varsinaiseen ohjelmointiin suunnattuja menetelmiä ja käytäntöjä varmistumaan ohjelmiston laadusta.

Näiden asioiden perusteella arvioin XP-menetelmän vaatiman erikoisosaamisen luokkaan *Heikko*. Huomattavasti kevyemmän scrum-mallin vaatima erikoisosaaminen on asetettu luokkaan *Kohtuullinen*.

Menetelmän tehokkuus ja toimivuus perehtymisvaiheen jälkeen on kuitenkin huomattavan paljon tärkeämpää, kuin varsinaiseen perehtymiseen käytetty aika ja vaiva. Tähän kysymykseen ei voida vastata siis kohdeprojektin kannalta, vaan tilannetta on katsottava kaukonäköisemmin. Arvioin tarpeen olevan luokkaa *Kohtuullinen* (Taulukko 17). Prioriteetiksi asetetaan 2.

Taulukko 17. Menetelmien vaatima osaaminen

Kehitysmenetelmä	Vaatii vain vähän hallinnon tai kehittäjien erikoistumista
Puhdas vesiputous	Kohtuullinen
Spiraali	Heikko
Muunnetut vesiputoukset	Heikosta kohtuulliseen
Evolutiivinen protoilu	Heikko
Vaiheistettu toimitus	Kohtuullinen
Evolutiivinen toimitus	Kohtuullinen
SCRUM	Kohtuullinen
eXtreme programming (XP)	Heikko

Koska verrattavia ohjelmistokehitysmenetelmiä on realistisesti mahdollista käyttää vasta tulevaisuuden projekteissa, jotka ovat samankaltaisia kohdeprojektin kanssa, tuotekehityksellä on mahdollista perehtyä monimutkaisempaankin menetelmään. Monimutkaiseen menetelmään perehtyminen vie kuitenkin tuotekehitystiimin resursseja, eikä sitä voida pitää hyvänä asiana.

6.1.11 McConnell (2002) testin yhteenveto

Menetelmien ominaisuuksia verrataan kohdeprojektin vaatimiin tarpeisiin niin, että mikäli menetelmä ei vastaa projektin tarpeisiin, se saa yhdestä neljään miinuspistettä. Mikäli menetelmä tarjoaa riittävän (tai paremman) ominaisuuden kohdeprojektin tarpeisiin, menetelmä saa nolla pistettä.

Pluspisteitä ei jaeta, sillä jos menetelmä ylittää kohdeprojektin tarpeet, ei kohdeprojektin kannalta saavuteta lisäarvoa siitä kuinka selkeästi tarve ylittyy. Mikäli kohdeprojektin tarve on *Heikko*, kaikki menetelmät saavat nolla pistettä.

Miinuspisteitä menetelmä saa yhtä monta kuin luokkien erojen lukumäärä on kohdeprojektin tarpeen ja menetelmän tarjoaman ominaisuuden välillä. Esimerkiksi jos kohdeprojektin tarve on luokkaa *Erinomainen*, ja menetelmä tarjoaa luokan *Heikko*, saa menetelmä -4 pistettä, sillä tarpeen ja ominaisuuden eroon jää neljä luokkaa *Kohtuullisesta erinomaiseen*, *Kohtuullinen*, *Heikosta kohtuulliseen*, sekä *Heikko*.

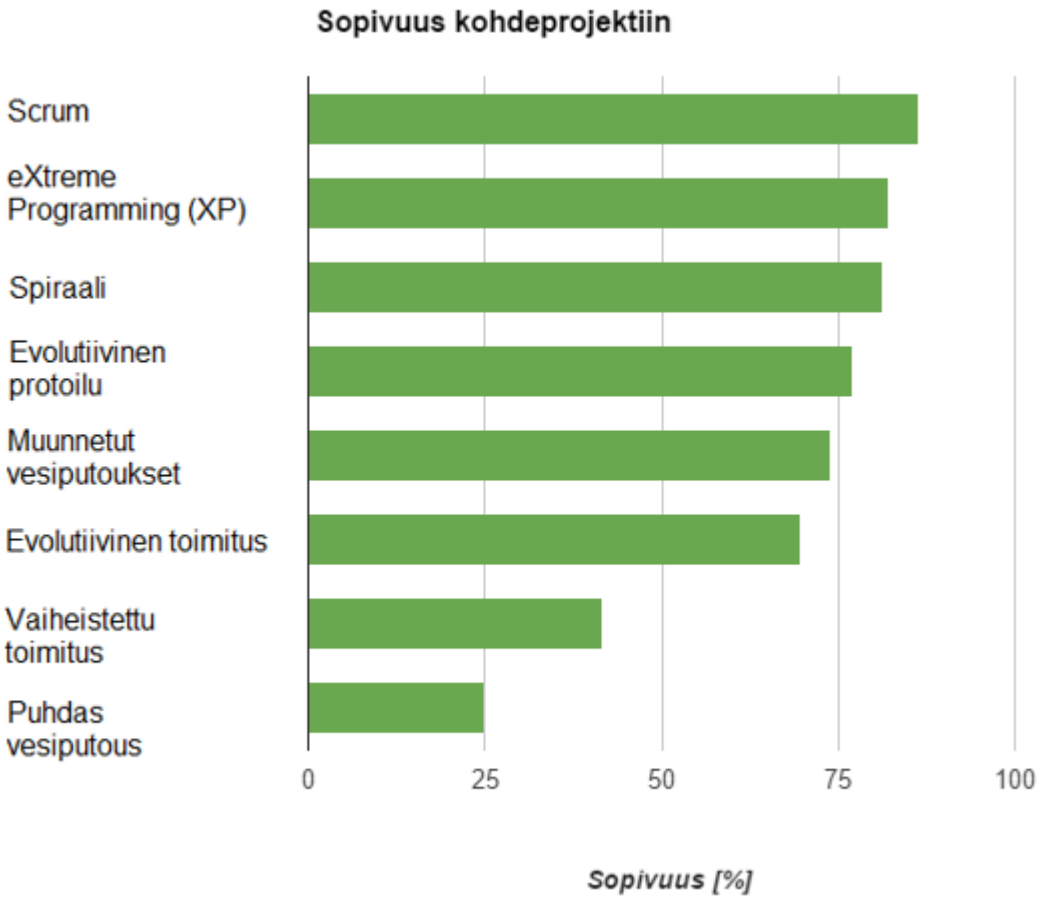
Jokaisen kohdan pisteet kerrotaan osa-alueelle annetulla prioriteetilla. Vähiten pisteitä saanut menetelmä on sopivin kohdeprojektiin (Taulukko 18).

Taulukko 18. Menetelmien sopivuudet kohdeprojektiin

Kehitysmenetelmä	Poikkeavuus-pisteet	Poikkeavuus-pisteet prioriteetti huomioituna	Sopivuus kohdeprojektiin [%]
SCRUM	−4	−13	**86**
eXtreme Programming (XP)	−4	−17	**82**
Spiraali	−6	−18	**81**

Evolutiivinen protoilu	−5	−22	**77**
Muunnetut vesiputoukset	−8	−25	**74**
Evolutiivinen toimitus	−8	−29	**70**
Vaiheistettu toimitus	−14	−56	**42**
Puhdas vesiputous	−18	−72	**25**

Paras sopivuus kohdeprojektiin saavutetaan scrum-menetelmällä. XP-menetelmä on toiseksi sopivin, ja lähes samoilla pisteillä kolmantena on Spiraali-menetelmä (Kuva 22).



Kuva 22. Kehitysmenetelmien sopivuudet kohdeprojektiin

Neljä sopivinta menetelmää saavuttavat yli 75 % sopivuuden kohdeprojektin vaatimuksiin. Heikoiten kohdeprojektiin sopisi puhdas vesiputous 25 % sopivuudella.

6.2 SWOT-analyysi

McConnell (2002) testausmenetelmän kolme parhaiten sopivaa menetelmää olivat Scrum, XP, sekä Spiraali. SWOT-analyysissä pyritään tarkastelemaan jokaista näitä tarkemmin.

6.2.1 Scrum

Scrum menetelmän perusajatus on yksinkertainen ja selkeä. Siinä pyritään pitämään jatkuvasti tarkka tilannetieto siitä, missä kohdassa projektia ollaan, mitä ollaan saatu tehtyä, mitä on vielä tekemättä, ja kuinka kauan se tulee kestämään. Menetelmä luo prosesseja, jotka varmistavat, että näin tapahtuu ilman ylimääräistä työtä. Taulukossa 19 on analysoitu scrum-menetelmä SWOT:in avulla.

Taulukko 19. Scrum-menetelmän SWOT-analyysi

Scurm SWOT	
Vahvuudet	Heikkoudet
Asiakasläheisyys	Tehokkaaseen käyttöön tarvitaan työhistoriaa
Helpottaa projektin johtamista	Ei ota kantaa ohjelmiston kehitystapoihin
Parantaa näkyvyyttä	Luo rajoituksia ohjelmistokehitykseen
Tarkentaa aikatauluarvioita	Paljon tehtävien aika-arviointeja
Paljon kommunikointia tiimissä	Paljon palavereja
Ei ota kantaa ohjelmointitapoihin	Tarvitsee erillisen alamenetelmän ohjelmointiin
Nopea reagointi ongelmiin	Paljon omaa terminologiaa
Aikataulutetut versiojulkaisut	Ei markkinayhteyttä
Mahdollisuus käyttää muita menetelmiä ohjelmoinnissa	
Mahdollisuudet	Uhat
Tarkkojen aikataulujen luonti	Kestojen arviointivaikeus tutkivassa kehityksessä
Projektin johtaminen helpottuu	Koodien laatu ei parane
Projektin ongelmat huomataan	Useat palaverit vaativat täsmällistä läsnäoloa
	Sprinttien sisällön suunnittelu haastavaa
	Dokumentit jäävät vähemmälle

Scrum-menetelmä voi olla hyvä kehys ohjelmistokehitysprojekteille, mutta tarvitsee mahdollisesti vielä alamenetelmän, jota käytettäisiin varsinaisen ohjelmiston tekemiseen.

Scrum-menetelmää käytetään tyypillisesti 5-9 kehittäjän tiimeissä, joka olisi sopiva useimmissa kohdeyrityksen projekteissa.

6.2.2 EXtreme Programming

Taulukossa 20 tutkitaan XP-menetelmää SWOT-menetelmän avulla. XP-menetelmä keskittyy ohjelmiston laatuun, mutta on kuitenkin raskas menetelmä. Vaikka XP menetelmää suositaan käytettäväksi myös kohtuullisen pienissä projekteissa, menetelmän parhaat edut tulevat esiin, kun samaa ohjelmistoa kehittää 6-30 ohjelmistokehittäjää. Tällöin jaettu ohjelmisto, testi ensin -ohjelmointitapa sekä pariohjelmointi auttavat tuotekehitysryhmää pitämään ohjelmiston hallinnassa. Koodin laatu paranee ja ohjelmistokehittäjät saavat menetelmästä kaiken hyödyn irti.

Taulukko 20. EXtreme Programming SWOT-analyysi

eXtreme Programming SWOT	
Vahvuudet	Heikkoudet
Ohjelmiston laatu paranee	Vaikea ottaa käyttöön
Asiakasläheisyys	Tarkoitettu suuremmille tiimeille
Testausmäärä tuo luotettavuutta	Pariohjelmointi on raskasta
Helpottaa projektin johtamista	Rajoittaa ohjelmoinnin tiettyihin tehtäviin
Parantaa näkyvyyttä	Luo rajoituksia ohjelmistokehitykseen
Tarkentaa aikatauluarvioita	Paljon tehtävien aika-arviointeja
Paljon kommunikointia tiimissä	Paljon palavereja
Ei päällekkäistä työtä	Tehokkaaseen käyttöön tarvitaan työhistoriaa
Vikojen minimointi	Ohjelmakoodi sisältää paljon testiohjelmaa
Koko tiimi tuntee ohjelmiston	
Nopea reagointi ongelmiin	
Nopea versioiden toimitus	
Aikataulutetut versiojulkaisut	

Mahdollisuudet	Uhat
Tarkkojen aikataulujen luonti	Kestojen arviointivaikeus tutkivassa kehityksessä
Projektin johtaminen helpottuu	Muutosvastarinta poikkeavien menetelmien takia
Projektin ongelmat huomataan	Useat palaverit vaativat täsmällistä läsnäoloa
Yhtenäiset ohjelmointitavat	Iteraatioiden suunnittelu haastavaa
Uudet tiimin jäsenet sopeutuvat nopeasti	Pienten prosessorien käyttö hankaloituu
Kaiken ohjelmakoodin ymmärtää useampi kehittäjä	Dokumentointi voi jäädä vähäiseksi
Laatu ja tehokkuus parantuvat	Testi ensin -ohjelmointi vaikeaa sulautetuissa järjestelmissä
	Tehtävien kestojen arviointi epätarkkaa aluksi

Kohdeyritys työskentelee sulautettujen järjestelmien parissa ja projektit koostuvat usein monien sulautettujen laitteiden verkostosta yhdistettynä PC-ohjelmaan. Tämä on tilanne myös kohdeprojektissa.

Vaikka tuotekehittäjien määrän puolesta menetelmää voisikin käyttää, tuotekehittäjät on kuitenkin jaettu kehittämään järjestelmän eri osia. Tämän vuoksi XP-menetelmän täysimääräinen käyttö olisi kohdeyrityksen projekteissa liian raskas etuihin nähden.

6.2.3 Spiraali

Spiraalimalli yhdistää idean iteratiivisesta kehityksestä hallitusti rajattuihin osiin vesiputousmallia. Menetelmä on yhdistelmä iteratiivista kehitysprosessia ja lineaarista kehitysmallia (vesiputousmalli), yhdistettynä hyvään riskienhallintaan. Taulukossa 21 on analysoitu spiraalimallia SWOT-menetelmän avulla.

Kohdeyrityksessä tehdään paljon tuotekehitysprojekteja, jotka voidaan luokitella tutkivan tuotekehityksen luokkaan. Tutkiva tuotekehitys on riskialtista, eikä koskaan voida olla varmoja siitä, millainen laitteistosta loppujen lopuksi tulee. Vaatimusmuutokset ovat todennäköisiä, sekä laitteistot ovat kokonaisuuksina monimutkaisia.

Spiraalimalli toimii erinomaisesti juuri tällaisissa projekteissa. Menetelmä on raskas ja kallis, mutta suurissa tutkivan tuotekehityksen projekteissa myös riskit ovat suuret. Riskien toteutuminen suuressa tuotekehitysprojektissa voi pahimmillaan kaataa koko yrityksen. Seuraukset voivat olla erittäin kauaskantoiset, kuten luvun 2 esimerkeistä havaittiin.

Kuten luvussa *3.1.2 Spiraalimalli* havaittiin, spiraalimallia käytetään yleisesti suurissa riskialtteissa projekteissa. Spiraalimallia voidaan käyttää niin ohjelmoinnissa, kuin elektronisessa suunnittelussakin.

Taulukko 21. Spiraalimallin SWOT-analyysi

Spiraalimalli SWOT	
Vahvuudet	**Heikkoudet**
Riskien minimointi	Haastavaa käyttää
Hyvä suurissa projekteissa	Kallis ja raskas menetelmä
Hyvä dokumentointi	Riskianalyysit vaativat paljon tietotaitoa
Sallii joustavasti muutoksia	Ei sovi hyvin pieniin projekteihin
Aikainen ohjelmointi ja testaus	Ei sovi vähäriskisiin projekteihin
Sallii versioiden julkaisun kesken kehityksen	
Palautteen keräys iteraation päätteeksi	
Asiakas näkee ohjelmiston jo aikaisessa vaiheessa	
Mahdollisuudet	**Uhat**
Riskienhallinta riskialtteissa projekteissa	Kustannusten nousu raskaan menetelmän vuoksi
Sopii hyvin tutkivaan tuotekehitykseen	Epätarkat riskianalyysit
Sopii hyvin kun käyttäjät eivät ole varmoja vaatimuksista	Liian haastava käytettäväksi
Sopii monimutkaisiin järjestelmiin	Tehoton epäsopivissa projekteissa
	Riskianalyysit vaikuttavat suuresti projektiin
	Aikataulutus vaikeaa
	Spiraali voi jatkua loputtomiin

Muut menetelmän negatiiviset puolet kohdentuvat menetelmän haastavaan käyttöön. Riskianalyysit ovat haastavia, ja menetelmän käyttö on kaikilta osin vaikeaa.

7 JOHTOPÄÄTÖKSET

Kohdeyrityksen tuotekehitysprojektit ovat usein joko pieniä, yksinkertaisia, sulautettuja järjestelmiä, tai tutkivan tuotekehityksen kautta luotuja laajoja ja riskialttiita sulautettujen järjestelmien ja PC-ohjelmien verkostoja.

Kohdeprojekti, tiemerkintöjä tekevä tulostin, voidaan luokitella jälkimmäiseen luokkaan. Projekti on riskialtis, sillä riskit voivat pahimmillaan johtaa siihen, että koko järjestelmä on suunniteltava uudestaan, ja se voi paljastua vasta käyttöönottovaiheessa.

Riskialttiisiin tutkivan tuotekehityksen projekteihin suosittelen spiraalimallin täysimääräistä käyttöä, jos projekti on riittävän laaja (yli 20 työkuukautta). Menetelmän vaikeakäyttöisyyden ja monimutkaisuuden vuoksi menetelmää ei voida käyttää kohdeprojektissa. Menetelmän tehokas käyttö vaatii harjoittelua ja koko tiimin perehtymistä menetelmän käyttöön.

Uskon, että mallin hyvät puolet riskipitoisissa projekteissa selkeästi voittaisivat menetelmän monimutkaisuudesta johtuvat huonot puolet. Menetelmän monimutkaisuudesta johtuvat huonot puolet tulevat myös pienenemään, kunhan menetelmään on tutustuttu, ja menetelmää on käytetty muutamissa projekteissa.

Vähäriskiset, mutta laajat tuotekehitysprojektit olisivat tehottomia suorittaa spiraalimallin avulla, tähän tarkoitukseen scrum-menetelmän käyttö olisi huomattavasti sopivampi, jos projektissa olisi vähintään 5 kehittäjää. Tätä menetelmää voisi käyttää myös omien tuotteiden tuotekehityksessä, jossa asiakkaana toimivat kohdeyrityksen myynti-, ja johto-osastot. Tällöin menetelmän vaatima asiakkaan läsnäolo voidaan varmistaa.

Scrum-menetelmän kanssa voi olla hyödyllistä pyrkiä käyttämään eXtreme Programming -menetelmää. Scrum-menetelmän avulla johdetaan ja hallitaan projektia. XP-menetelmä keskittyy ohjelmiston tuotannon tehokkuuteen ja laatuun. Jos kohdeprojektissa useat kehittäjät tekevät samaa ohjelmistoa, Scrum & XP-hybridin käyttö olisi tehokasta.

XP-menetelmä on täysimääräisesti käytettynä perinteisestä tuotekehityksestä erityisen poikkeava. Täysimääräinen käyttö vaatii ajallisesti paljon sopeutumista. Menetelmän tekniikat ovat kuitenkin toisistaan riippumattomia eli niitä voisi pyrkiä hyödyntämään vain osittain.

Testi ensin -ohjelmointitapa vaikuttaa olevan ensisijaisesti suunniteltu PC-ohjelmointiin. Sulautetuissa järjestelmissä kyseinen tekniikka on haasteellista, vaikka testaus sulautetuissa järjestelmissä onkin erittäin tärkeää. Tekniikka luo rajoitteita myös pienten mikrokontrollerien käyttöön, sillä niiden kooditila on aina hyvin rajallinen. Testi ensin -menetelmä loisi paljon sellaista koodia, jota ei tarvita ilman testi ensin -ohjelmointitapaa.

XP-menetelmän käyttämässä pariohjelmoinnissa voisi olla paljon potentiaalia ilman XP-menetelmän käyttöäkin. Pariohjelmointi on riskialtis, sillä se on intensiivinen ja raskas tapa ohjelmoida. Jotkut ohjelmoijat eivät sopeudu menetelmään koskaan.

Tuotekehityksessä olisi hyvä huomioda pariohjelmoinnin potentiaali, sitä olisi hyvä käyttää tietyissä tilanteissa, jos kehittäjät ovat tähän halukkaita. Kohdeprojektin tulisi sisältää suuri ohjelmisto, jota useat eri ohjelmoijat kehittävät esimerkiksi yli vuoden ajan. Tällöin pariohjelmoinnin avulla varmistettaisiin laadun ja tehokkuuden paraneminen, sekä koko tiimin koko ohjelmiston hallinta.

Tuotekehityksessä tulisi aina välttää tilanteita, jossa yksi ihminen tekee ohjelmiston käyttäen tapoja jotka poikkeavat kohdeyrityksen muiden kehittäjien toimintatavoista. Tällöin toisen kehittäjän on erittäin vaikea perehtyä tehtyyn koodiin, jos tilanne niin tulevaisuudessa vaatii.

Pariohjelmointi ja jaettu ohjelmisto -tekniikat auttaisivat hallitsemaan tätä ongelmaa useissa eri projekteissa. Pariohjelmointia voisi hyödyttää myös uuden ohjelmoijan perehdyttämiseen kohdeprojektiin, sillä tällä tekniikalla perehtyminen on nopeaa.

Kuten luvun 3 teoriaosuudessa havaittiin, tehokas ohjelmistokehitysmenetelmä riippuu täysin kohdeprojektista. Kohdeyrityksen erilaisiin projekteihin tyypillisesti sopisivat spiraalimalli tai scrum. Molempien menetelmien lisäksi suosittelen

käytettäväksi ohjelmiston laatua ja tehokkuutta parantavia menetelmiä, joita eXtreme Programming menetelmä tarjoaa runsaasti.

LÄHTEET

AgileManifesto, 2001. Luettu 18.2.2017. http://agilemanifesto.org/

Ambrosini, V., Johnson, G., & Scholes, K. (1998). Exploring techniques of analysis and evaluation in strategic management. Englewood Cliffs, N.J.: Prentice-Hall.

Beck, K., & Andres, C. (2005). Extreme programming explained : Embrace change. Boston: Addison Wesley.

Haikala, I., & Mikkonen, T. (2011). Ohjelmistotuotannon käytännöt (12. uud. p. ed.). Helsinki: Talentum.

Haikala, I., & Märijärvi, J. (2004). Ohjelmistotuotanto (10. uud. p., 11. p. 2006. ed.). Helsinki: Talentum.

Hamlet, D., & Maybee, J. (2003). The engineering of software : Technical foundations for the individual. Boston: Addison Wesley.

Hänninen, Eeva, 2010, Extreme Programming ohjelmistokehityksessä. Opinnäytetyö, Jyväskylän ammattikorkeakoulu. Luettu 19.1.2017. https://www.theseus.fi/bitstream/handle/10024/15511/Hanninen_Eeva.pdf?sequence=1

IEEE, 2017, Systems and software engineering – Architecture description. Luettu 24.2.2017. http://www.iso-architecture.org/ieee-1471/defining-architecture.html

Jeffries, R., Anderson, A., & Hendrickson, C. (2001). Extreme programming installed. Boston: Addison-Wesley.

Kerzner, H. R. (2014). Project Recovery. Somerset: John Wiley & Sons, Incorporated.

Koch, A. (2004). Agile software development : Evaluating the methods for your organization. Norwood, US: Artech House Books.

Koskimies, K., & Mikkonen, T. (2005). Ohjelmistoarkkitehtuurit. Helsinki: Talentum.

Liikennevirasto (2013). Törmäysvaimentimen käytön hyödyllisyys liikenne- ja työturvallisuuden näkökulmasta. (luettu 22.12.2016). http://www2.liikennevirasto.fi/julkaisut/pdf3/lr_2013_tormaysvaimentimen_kayton_web.pdf

Liikennevirasto (2014). Tieturva 1. Tiellä työskentelyn turvallisuuskoulutus. Helsinki, FI, http://www2.liikennevirasto.fi/julkaisut/pdf8/lop_2014-03_tieturva_1_web.pdf , Luettu 24.12.2016.

Liikennevirasto (2015). Tiemerkintöjen suunnittelu, luettu 24.12.2016. http://www2.liikennevirasto.fi/julkaisut/pdf8/lo_2015-25_tiemerkintojen_suunnittelu_web.pdf

Lindroos, J., & Lohivesi, K. (2010). Onnistu strategiassa [talentum- verkkokirjahylly] (3. uud. p. ed.). Helsinki: WSOYpro.

Männistö, Sami, 2008, Extreme Programming (XP) –metodologian soveltaminen ohjelmistojen tuotantoon. Opinnäytetyö, Tampere. https://www.theseus.fi/bitstream/handle/10024/10227/M%C3%83%3fnnist%C3%83%C2%B6.Sami.pdf?sequence=2

McConnell, S., Toikkanen, T., & Arola, J. (2002). Ohjelmistotuotannon hallinta. Helsinki: Edita : IT Press.

Pfleeger, S. L. (1998). Software engineering : Theory and practice. Upper Saddle River (NJ): Prentice Hall.

Puhelinhaastattelu 23.12.2016. Kohde: Yrityskumppanin toimitusjohtaja.

Savolainen, Paula, 2011, Why Do Software Development Projects fail? Jyväskylän yliopisto, väitöskirja. Luettu 19.2.2017. https://jyx.jyu.fi/dspace/bitstream/handle/123456789/36970/9789513944681.pdf?sequence=1

Sommerville, I. (2001). Software engineering (6th ed ed.). Harlow, England: Pearson Education.

Steinberg, D. H., & Palmer, D. W. (2004). Extreme software engineering : A hands-on approach. Upper Saddle River: Pearson.

Sähköpostikysely 18.11.2016, Kohde: Asiakasyrityksen toimitusjohtaja.

The Standish Group, Chaos Report, 2014, luettu 24.1.2017. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

Tapio, Tero, 2010, Riskienhallinta perinteisessä ja ketterässä ohjelmistokehityksessä. Pro gradu –tutkielma, Tampereen yliopisto. Luettu 8.2.2017. https://tampub.uta.fi/bitstream/handle/10024/81663/gradu04362.pdf?sequence=1

Tiehallinto, 2004, Helsinki, Tiemerkintöjen kuntoluokitus, luettu 25.12.2016, http://alk.tiehallinto.fi/thohje/pdf/2200022-v-04tiemerkint_kuntoluok.pdf

Tiehallinto, 2015, Helsinki, Tiemerkintöjen laatuvaatimukset, Luettu 26.12.2016, http://www2.liikennevirasto.fi/julkaisut/pdf8/lo_2015-38_tiemerkintojen_laatuvaatimukset_web.pdf

Tomayko, J., & Hazzan, O. (2004). Human aspects of software engineering. Herndon, US: Charles River Media / Cengage Learning.

Trysil Maskin Trafficprinter, katsottu 24.12.2016. Youtube esittelyvideo. https://www.youtube.com/watch?v=dzIF9fNTdFY

Tutorialspoint, SDLC- Spiral Model. luettu 10.3.2017. https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

Wiegers, K. E. (2003). Software requirements (2nd ed ed.). Redmond (WA): Microsoft Press.

LIITE 1. Ohjelmistokehitysmenetelmän valintatestin tulokset

Elinkaarimallin ominaisuus	Puhdas vesiputous	Spiraali	Muunnetut vesiputoukset	Evolutiivinen protoilu	Vaiheistettu toimitus	Evolutiivinen toimitus	SCRUM	eXtreme programming	Arvioitu tarve projektissa	Prioriteetti 1-5
Toimii huonosti ymmärrettyjen vaatimusten kanssa	**Heikko**	Erinomainen	Kohtuullisesta erinomaiseen	Erinomainen	**Heikko**	Kohtuullisesta erinomaiseen	Erinomainen	Erinomainen	**Erinomainen**	**5**
Toimii huonosti ymmärretyn arkkitehtuurin kanssa	Heikko	Erinomainen	Kohtuullisesta erinomaiseen	Heikosta kohtuulliseen	Heikko	Heikko	Heikko	Heikko	**Kohtuullinen**	**2**
Tuottaa hyvin luotettavan järjestelmän	Erinomainen	Erinomainen	Erinomainen	Kohtuullinen	Erinomainen	Kohtuullisesta erinomaiseen	Erinomainen	Erinomainen	**Heikko**	**2**
Tuottaa järjestelmän, jossa on hyvin kasvuvaraa	Erinomainen	Erinomainen	Erinomainen	Erinomainen	Erinomainen	Erinomainen	Erinomainen	Erinomainen	**Erinomainen**	**5**
Riskienhallinta ominaisuudet	**Heikko**	Erinomainen	Kohtuullinen	Kohtuullinen	Kohtuullinen	Kohtuullinen	Kohtuullisesta erinomaiseen	Kohtuullisesta erinomaiseen	**Erinomainen**	**5**
Toimivuus joustamattomissa aikataululuissa	Kohtuullinen	Kohtuullinen	Kohtuullinen	Heikko	Kohtuullinen	Kohtuullinen	Erinomainen	Erinomainen	**Heikko**	**1**
Sisältää vain vähän lisätyötä	**Heikko**	Kohtuullinen	Erinomainen	Kohtuullinen	Kohtuullinen	Kohtuullinen	Erinomainen	Erinomainen	**Erinomainen**	**3**
Sallii suunnanvaihdoksia	**Heikko**	Kohtuullinen	Kohtuullinen	Erinomainen	**Heikko**	Kohtuullisesta erinomaiseen	Kohtuullisesta erinomaiseen	Kohtuullisesta erinomaiseen	**Erinomainen**	**4**
Tarjoaa asiakkaalle edistymismerkkejä	Heikko	Erinomainen	Kohtuullinen	Erinomainen	Kohtuullinen	Erinomainen	Erinomainen	Erinomainen	**Heikko**	**2**
Tarjoaa hallinnolle edistymismerkkejä	Kohtuullinen	Erinomainen	Kohtuullisesta erinomaiseen	Kohtuullinen	Erinomainen	Erinomainen	Kohtuullisesta erinomaiseen	Kohtuullisesta erinomaiseen	**Heikko**	**2**
Vaatii vain vähän hallinnon tai kehittäjien erikoistumista	Kohtuullinen	Heikko	Heikosta kohtuulliseen	Heikko	Kohtuullinen	Kohtuullinen	Kohtuullinen	Heikko	**Kohtuullinen**	**2**