Tampereen yliopisto

Juuso Haapanen

KETTERYYS KONEOPPIMISPROJEKTEISSA

Diplomityö
Johtamisen ja talouden tiedekunta
Tarkastajat: Prof. Kari Systä
Lehtori Harri Keto
Lokakuu 2021

TIIVISTELMÄ

Juuso Haapanen: Ketteryys koneoppimisprojekteissa
Diplomityö
Tampereen yliopisto
Johtamisen ja tietotekniikan tutkinto-ohjelma
Lokakuu 2021

Koneoppimisen hyödyntäminen osana liiketoimintaa lisääntyy jatkuvasti ja organisaatiot kehittävät koneoppimisratkaisuja osana ohjelmistokehitystiimien toimintaa. Tämän työn tarkoituksena oli ymmärtää hyödynnetäänkö koneoppimisprojekteissa ohjelmistokehityksen kaltaisia menetelmiä kuten prosessimalleja tai ketteriä menetelmiä sekä miten näiden menetelmien käyttö soveltuu koneoppimisprojekteihin.

Työ toteutettiin puolistrukturoituna haastattelututkimuksena, jossa haastateltiin koneoppimisen parissa työskenteleviä henkilöitä.

Haastattelujen perusteella koneoppimisprojektit toteutettiin joko osana laajempaa ohjelmistoprojektia tai erillisenä kokeiluna, jonka tarkoituksena oli selvittää onko jokin ongelma ratkaistavissa koneoppimisen avulla.

Haastatteluista nousseiden tuloksien perusteella esitetään keinoja, joiden avulla organisaatiot voisivat paremmin organisoida koneoppimista kehittäviä projekteja.

Avainsanat: koneoppiminen, ketterät menetelmät, agile, lean

ABSTRACT

Juuso Haapanen: Agile and Lean Methods in Machine Learning projects
Master's Thesis
Tampere University
Master's Programme in Manamagement and Information Technology
October 2021

The utilization of machine learning methods as part of business operations is constantly increasing and organizations are developing machine learning solutions as part of software development projects. The purpose of this study was to understand whether machine learning projects utilize common software development methods, such as process models or agile methods, and how the use of these practices is suitable for developing machine learning projects.

The study was carried out as a semi-structured interview study in which informants working with machine learning solutions were interviewed.

Based on the interviews, the machine learning projects were implemented either as part of a larger software development project or as a separate proof of concept, where the purpose was to find out if a problem could be solved by using machine learning methods.

Based on the results of the interviews, organisations are presented with better ways of organizing machine learning projects.

Keywords: machine learning, agile methods, agile, lean

ALKUSANAT

Opiskelu täysipäiväisen työn ja lapsiperhe-elämän ohessa voi kuulostaa raskaalta ja sitä se varmaankin on välillä ollut. Opintojen ja lopulta diplomityön eteneminen on saanut kärsiä siitä kun muut, tärkeämmät asiat ovat menneet edelle. Vaikka tätä työtä ei olekaan kirjoitettu millekään organisaatiolle, niin siitä huolimatta olen kiitollinen siitä että minulla on ollut mahdollisuus työskennellä sellaisissa organisaatioissa, joissa opiskelua sekä itse

Haluan kiittää haastateltaviani siitä, että antoivat arvokasta työaikaansa ja mahdollistivat työn kannalta keskeiset haastattelut.

Lisäksi haluan kiittää professori Kari Systää ohjauksesta sekä motivoimisesta eteenpäin.

Lopuksi haluan vielä kiittää puolisoani sekä tytärtäni siitä, että ovat jaksaneet välillä hyvinkin raskasta sohvan nokassa nököttämistä sekä opiskelua.

Helsingissä, 12. lokakuuta 2021

Juuso Haapanen

SISÄLLYSLUETTELO

1 Johdanto 1
2 Koneoppiminen 3
2.1 Mitä koneoppiminen on? 3
2.2 Koneoppimismenetelmät 3
2.3 Neuroverkot 5
2.4 Koneoppimismenetelmien metriikat ja testaaminen 5
2.5 Koneoppimisen sovelluskohteet 6
2.6 Koneoppimisprojektien tyypit 7
3 Ohjelmistokehityksen ja koneoppimisen prosessimallit 8
3.1 Ohjelmistokehityksen prosessimalli 8
3.2 Koneoppimisprosessi 10
3.3 Ohjelmistokehityksen ja koneoppimisen prosessimallien vertailu 13
4 Ketterät menetelmät 16
4.1 Lean yritysten johtamisessa 16
4.2 Ketterät menetelmät ohjelmistokehityksessä 17
4.3 Ketterät menetelmät UX-kehityksessä 21
4.4 DevOps, DataOps ja MlOps 22
4.4.1 Devops 22
4.4.2 DataOps ja MlOps 23
4.5 Ohjelmistokehitysmenetelmän valinta 25
5 Haastattelu ketterien menetelmien hyödyntämisestä 27
5.1 Haastattelu tutkimusmenetelmänä 27
5.1.1 Haastateltavien valinta ja haastattelujen toteutus 28
5.2 Haastattelun runko ja kysymykset 29
6 Tutkimuksen tulokset ja analyysi 30
6.1 Koneoppimistekemisen iteratiivinen luonne 30
6.2 Kommunikaatio asiakkaan kanssa 31
6.3 Projektin työryhmien kokoonpanot 31
6.4 TK1: Koneoppimisen prosessimallien hyödyntäminen 33
6.5 TK2: Ketterien menetelmien hyödyntäminen 34
7 Ehdotus ketterien menetelmien käytöstä koneoppimisprojekteissa 36
7.1 Organisaation rakenne ja toiminta 36
7.1.1 Oppiminen 36
7.1.2 Prosessien läpinäkyvyys 37
7.1.3 Työn läpinäkyvyys ja kommunikaatio 37

7.1.4 Iteratiivisuus . . . 37
7.1.5 Koneoppimisprojektin kokoonpano . . . 37
7.2 Taksonomian ymmärtäminen . . . 38
7.3 Työkalut koneoppimisprojektissa . . . 39
7.3.1 Versionhallinta . . . 39
7.3.2 Rajapinnat ja integraatiot . . . 40
7.3.3 Koneoppimisen ja data-analyysi . . . 40
7.4 Koneoppimisen prosessimalli ketteriin koneoppimisprojekteihin . . . 41
7.5 Vertailu olemassa oleviin malleihin . . . 44
8 Yhteenveto . . . 45
8.1 Johtopäätelmät . . . 45
8.2 Mahdolliset jatkotutkimukset . . . 47
Lähteet . . . 48

KUVALUETTELO

3.1 Ohjelmistokehityksen prosessimalli, CRISP-DM ja Amershin malli 14

4.1 Visuaalinen hahmotelma organisaation toiminnan, ketterien menetelmien ja DevOpsin yhteydestä 24
4.2 CI/CD ja automatisoitu koneoppimisprosessi [53] 25

7.1 Kaavio tilastollisen menetelmän valinnasta [63] 41
7.2 Esitetty koneoppimisprosessi 42

TAULUKKOLUETTELO

3.1 Koneoppimisen prosessimallit 13

5.1 Haastateltavien taustat 29

6.1 Roolit koneoppimisprojekteissa 32
6.2 Haastateltavien esittämät prosessit 34

LYHENTEET JA MERKINNÄT

CD	Jatkuva tuotantoonvienti (eng. Continuous Development)
CI	Jatkuva integraatio (eng. Continuous Integration)
GPU	Graphics Processing Unit, grafiikkalaskentayksikkö
ML	Koneoppiminen (eng. Machine Learning)

1 JOHDANTO

Ketterät menetelmät ovat ohjelmistokehitysmetodologioita, joiden tarkoituksena on mahdollistaa ohjelmistojen kehittäminen mahdollisimman nopeasti ja mahdollisimman hyvin muutoksiin reagoiden. Ketterät menetelmät ovat syntyneet siksi, että perinteisemmät ohjelmistokehitysmenetelmät kuten vesiputousmalli sekä Rational Unified Process koettiin raskaiksi alan ammattilaisten keskuudessa.

Data-analyysin sekä koneoppimisen yleistyessä sekä tullessa osaksi ohjelmistoprojekteja, myös niitä varten on kehitetty prosessimalleja joiden avulla projektien eri vaiheita voidaan kuvata.

Tämän työn tarkoituksena on ymmärtää miten ohjelmistokehityksestä peräisin olevat menetelmät soveltuvat koneoppimisprojekteihin sekä miten käytännön koneoppimisprojekteissa sovelletaan ketteriä menetelmiä ja miten erilaiset prosessilähtöiset mallit näkyvät käytännön työssä. Oletuksena tässä työssä on se, että koneoppimista hyödyntävien sovellusten kehittämisessä on mukana aina epävarmuutta esimerkiksi sen vuoksi, että mallit perustuvat tilastollisiin menetelmiin eikä projektien alussa välttämättä voida tietää pystytäänkö haluttua ohjelmistoa tai ominaisuutta toteuttamaan. Toisena työn oletuksena on se, että organisaatiot kehittävät ohjelmistoja - riippumatta siitä, onko kyseessä koneoppimista hyödyntävä sovellus tai ei, käyttäen joitakin yleisesti tunnettuja ohjelmistokehitysmenetelmiä. Työn kannalta ei ole oleellista, onko kyseessä Scrum, Kanban vai vesiputousmalli. Koneoppimisen prosessimalli, johon perehdytään luvussa 3 nähdään iteratiivisena kuvauksena siitä mitä koneoppimisprojektissa pitäisi tehdä.

Tutkimuskysymyksiä ovat:

- **TK1** Sovelletaan koneoppimisen prosessimalleja ohjelmistoprojekteissa, joissa tehdään koneoppimista hyödyntäviä ohjelmistoja?
- **TK2** Miten ketteriä menetelmiä hyödynnetään koneoppimisprojekteissa?

Ensimmäisen tutkimuskysymyksen tarkoituksena on ymmärtää sovelletaanko koneoppimiseen kehitettyjä prosessimalleja ohjelmistokehitysprojekteissa, joiden lopputuloksena tavoitellaan koneoppimista hyödyntävää sovellusta.

Toisen tutkimuskysymyksen tarkoituksena on ymmärtää miten koneoppimisprojekteissa näkyy ketterien menetelmien ominaispiirteet sekä miten ketteriä menetelmiä hyödynnetään projekteissa. Tavoitteena on ymmärtää projekteja etenkin niiden etenemisen, projektissa käytössä olevan osaamisen sekä ketterien menetelmien erityispiirteiden näkö-

kulmasta.

Työn rakenne on seuraava: luvussa 2 käsitellään koneoppimisen perusteita tämän työn ymmärtämisen tarvitsemassa laajuudessa. Luvussa 3 käsitellään ohjelmistokehityksen sekä koneoppimisen prosessimalleja eli menetelmiä, jotka määrittelevät minkälaisia vaiheita ohjelmistojen kehittämiseen sekä koneoppimiseen liittyy. Luvussa 4 perehdytään ketterien menetelmien perusperiaatteisiin organisaatioiden sekä ohjelmistoprojektien näkökulmasta. Luvussa 5 käsitellään haastattelua tutkimusmenetelmänä sekä käydään läpi haastattelun tulokset.

2 KONEOPPIMINEN

Tässä luvussa perehdytään siihen mitä koneoppiminen on, minkälaisia ongelmia sillä voidaan ratkaista ja miten koneoppimista hyödyntävät ohjelmistot eroavat perinteisistä ohjelmistoista.

2.1 Mitä koneoppiminen on?

Koneoppimisella tarkoitetaan algoritmeja ja menetelmiä, joiden avulla voidaan olemassa olevasta datasta eli esimerkiksi numeerisesta aineistosta, kuvasta, videosta tai vaikkapa äänestä oppia ominaisuuksia ja näiden perusteella esimerkiksi ennustaa, luokitella tai luoda uusi arvoja. Yleistäen voidaan sanoa, että on olemassa jokin tehtävä T, joka halutaan ratkaista koneella siten, että suoriutumista voidaan arvioida mittarilla P kun algoritmille annetaan syötteenä signaaleja eli tietynlaista kokemusta E [1]. Koneoppimisen ero ns. perinteisiin algoritmeihin on se, että koneoppimisalgoritmi pyrkii itsenäisesti annetun aineistoin perusteella oppimaan kun perinteisessä algoritmissa koneelle annetaan jotkin säännöt, joita se noudattaa.

Rajan vetäminen data-analyysin, koneoppimisen sekä tekoälyn välillä ei ole suoraviivainen. Data-analyysissä käytetään menetelmiä, joita voidaan käyttää myös koneoppimisessa. Tekoälymenetelminä käytetään usein koneoppimismenetelmiä. Luvussa 2.2 käsitellään koneoppimismenetelmiä, joista osa on menetelminä sellaisia joita voi hyödyntää myös data-analyysissä. Tämän työn puitteissa data-analyysillä tarkoitetaan sitä, että organisaation käytössä on jotakin dataa, jonka analysoinnista katsotaan olevan organisaatiolle jotakin hyötyä mutta prosessin lopputuotetta ei välttämättä automatisoida. Tekoälyllä tarkoitetaan tässä työssä sellaisia koneoppimisalgoritmien toteutuksia, jotka toimivat jossakin autonomisessa ympäristössä agentteina tehden itse päätöksiä.

2.2 Koneoppimismenetelmät

Koneoppiminen voidaan jakaa karkeasti ohjattuun (supervised), ohjaamattomaan (unsupervised) sekä vahvistusoppimiseen (reinforcement learning) [1]. Ohjattu oppiminen perustuu siihen, että algoritmilla tai mallilla on olemassa jotakin aiempaa dataa jonka perusteella pystytään tekemään joko luokittelu- tai regressiomalleja [2]. Luokittelussa koneoppimismalli pyrkii annetun aineiston perusteella luomaan sääntöjä, joilla aineistoa voidaan jakaa annettuihin luokkiin. Regressiomallien tarkoituksena on ennustaa annetun aineis-

ton perusteella jonkin muuttujan arvo.

Koneoppimisalgoritmeissa käytössä oleva data jaetaan tyypillisesti kehittämisen aikana kolmeen osaan: opetus-, testi- ja validointidataan. Opetusdatan avulla koneoppimisalgoritmi opetetaan, eli haluttuun malliin sovitetaan parametrit. Validointidatan avulla pyritään välttämään, ettei malli ylisovita dataa. Testidatan tarkoituksena on testata miten hyvin malli toimii verrattuna tunnettuihin arvoihin. Kappaleessa 2.4 käsitellään metriikoita, joiden avulla mallin toimivuutta voidaan arvioida.

Esimerkkinä yksinkertaisesta koneoppimismallista voidaan pitää lineaarinen regressiota, jossa saadaan syötteenä aineiston $X^T = (X_1, X_2, \ldots, X_k)$ perusteella halutaan ennustaa jokin reaaliarvo Y. Lineaarisella regressiomalli voidaan esittää matemaattisesti muodossa

$$f(X) = \beta_0 + \sum_{n=1}^{k} X_n \beta_n \tag{2.1}$$

Käytännön koneoppimissovelluksen näkökulmasta tässä kaavassa X_n voi tulla jostakin muuttujasta tai sen muunnoksesta. β_n on tuntemattomia parametreja, joita pyritään estimoimaan funktiolla $f(X)$. Tyypillisesti regressiomallin käyttö perustuu siihen, että on olemassa jokin opetusaineisto sekä testiaineisto, jossa on arvoja (y_i, x_i) joiden perusteella voidaan estimoida parametrit β_n. Eräs menetelmä parametrien $\beta = (\beta_0, \beta_1, \ldots, \beta_n)$ estimointiin on pienimmän neliösumman menetelmä jäännöksille (residual sum of squares), joka voidaan määritellä kaavan 2.1 avulla seuraavasti

$$RSS(\beta) = \sum_{i=1}^{n} (y_i - f(x_i))^2. \tag{2.2}$$

Jäännösneliösummat sekä lineaarisen mallin yhtälö ovat hyvin yksinkertaisia kaavoja, mutta niiden avulla voidaan havainnollistaa hyvin ohjatun oppimisen toimintaa: käytössä on jokin testiaineisto johon pyritään sovittamaan funktiota siten, että mahdollinen virhe on hyvin pieni. Lineaarinen malli soveltuu menetelmänä esimerkiksi tabulaariselle aineistolle eli sellaiselle, jossa on numeerisia muuttuja matriisimuodossa mutta perusperiaate on sama myös monimutkaisemmille menetelmille. Esimerkki lineaarisesta regressiosta voisi olla esimerkiksi asuntoja välittävän yrityksen tarve ennustaa asuntoja asuntojen hintoja asuntoihin liittyvien parametrien avulla. Asunnosta voidaan tietää esimerkiksi sijainti, huoneiden lukumäärä, neliömäärä sekä rakennusvuosi ja näiden perusteella kiinteistövälitysyrityksen palkkaama insinööri voisi rakentaa sovelluksen, jonka avulla asiakas tai työntekijä voisi arvioida tiettyjen asuntojen hintoja. Opetus- ja testiaineistona olisi jonkin tietyn alueen toteutuneet asuntokaupat tai verkosta löytyvä listaus myytävistä asunnoista. Käytännön sovelluksissa käytössä oleva aineisto sekä mallit voivat olla hyvin monimutkaisia. [1, 2, 3].

Ohjaamattoman koneoppimisen menetelmissä algoritmi jakaa aineiston luokkiin datan perusteella, ilman että syötteenä on annettu luokkia etukäteen. Esimerkiksi yritys voisi luokitella asiakkaat ostotapahtuman perusteella luokkiin, joita ei ole entuudestaan määritelty. Toinen esimerkki ohjaamattomasta opetustuksesta on assosisaatiosääntöihin pe-

rustuvat algoritmit, jotka pyrkivät päättelemään aiempien yhdistelmien perusteella todennäköisiä, uusia yhdistelmiä [2, 3].

Kolmas koneoppimisen menetelmä on vahvistusoppiminen, jossa ohjelma koostuu agenteista jotka tekevät jotain toimitoja tietyssä ympäristössä saaden siitä tietynlaisen palkkion sen mukaan, miten hyvin on tehtävässä onnistunut. Vahvistusoppimisen kehittämisen taustalla on idea siitä, miten eläimet oppivat luonnostaan. Esimerkiksi ihmislapsen oppiminen pyöräilemään voidaan nähdä eräänlaisena vahvistusoppimisena: lapsi kokeilee pyöräilemistä ilman apupyöriä ja kaatuu, jolloin hän oppii että jotain tuli tehtyä väärin. Lapsi voi kuitenkin saada vanhemmaltaan positiivista palautetta tai lohdutusta hyvästä yrityksestä, jolloin hän yrittää uudestaan pyöräillä ja tekee mahdollisesti jotain toiste kuitenkin aiemmin. Vahvistusoppimisalgoritmit ovat autonomisia, eli ne voivat tehdä valintansa itse ja sen perusteella oppia toimimaan oikein [4].

2.3 Neuroverkot

Neuroverkot ovat eräänlaisia graafeja, joissa voidaan suorittaa erittäin paljon funktiokutsuja niin että funktiokutsujen ketjutuksella voidaan approksimoida hyvin jonkin funktion arvoja tai tehdä monimutkaisia laskutoimituksia.

Neuroverkot voidaan jakaa eteenpäinmeneviin (feedforward) sekä palaaviin (recurrent) verkkoihin. Eteenpäinmenevien neuroverkkojen avulla yritetään estimoida jotakin funktiota f^*, joka voi olla esimerkiksi luokittelija. Eteenpäinkutsuvat neuroverkot saavat nimensä siitä, että funktion approksimaatio perustuu valissä tapahtuviin laskutoimituksiin ja verkossa ei ole takaisinkytkentöjä takaisin.

Palaavat neuroverkot on kehitetty sellaiselle datalle, jossa on joko ajallinen ulottuvuus tai arvot riippuvat muuten edellisistä arvioista. Tällaista dataa on esimerkiksi aikasarjat tai teksti. Aikasarjoissa datapiste on aina sidottu johonkin ajanhetkeen ja niiden avulla voidaan mallintaa hyvin esimerkiksi pörssikursseja, sääilmiöitä tai vaikkapa sateellitin ottamia kuvia samasta kohdasta maapalloa eri aikoina. Tekstiä käsiteltäessä tietokoneella yksittäisen merkin tai sanan käsittely voi riippua siitä, mitä merkkejä tai sanoja kyseistä sanaa edeltää. Tällaista dataa käsiteltäessä paluupolkuja voi olla hyvin paljon. [1, 5]

Neuroverkkojen yleistyminen sekä laskentatehon kasvu ovat mahdollistaneet sen, että koneoppimista hyödyntävät sovellukset ovat yleistyneet. Neuroverkkojen käytön yleistymistä on tukenut se, että laskentatehoa on entistä enemmän saatavilla ja GPU eli grafiikkaprosessoreja voidaan hyödyntää neuroverkoissa tehokkaasti [1].

2.4 Koneoppimismenetelmien metriikat ja testaaminen

Koneoppimismenetelmien toimivuuden arviointi perustuu metriikoihin, joiden avulla arvioidaan miten hyvin malli suoriutuu sille annetuista tehtävistä. Thakur [6] esittää kirjassaan *Approaching (almost) any machine learning problem* listan metriikoita, joita voidaan hyödyntää koneoppimismallien hyvyyden arvioimiseen. Metriikat voidaan jakaa luokitte-

lun sekä regression metriikoihin. Luokittelun metriikoita on esimerkiksi seuraavat:

- Oikeellisuus (eng. Accuracy)
- Tarkkuus (eng. Precision)
- Muistuttavuus (eng. Recall)
- F1-arvo (eng. F1 score)
- AUC eli *Area Under ROC Curve*
- Logaritminen häviö (eng. log loss)

Luokittelun metriikoiden käyttö perustuu siihen, että tunnetaan osasta aineistoa todelliset luokittelijan arvot sekä mallin ennustamat arvot. Näiden pohjalta lasketaan haluttu metriikka. Esimerkiksi lintujen havainnointiin kehityn sovelluksen kehittämisessä voi käytössä olla kuvia linnuista, joita halutaan tunnistaa ja niiden pohjalta kehitetään malli. Tämän jälkeen mallia testataan tunnetuiden lintujen kuvilla ja arvioidaan miten hyvin mallin luokittelu toimii.

Regressio-ongelmien metriikat perustuvat siihen, että lasketaan regressiomallin aineiston sekä mallilla ennustetun arvon välinen virhe. Näiden metriikoiden käyttö sekä tarkat määritelmät löytyy esimerkiksi [1, 3, 5]. Malleihin liittyvien metriikoiden ymmärtäminen on kuitenkin tärkeää koneoppimisprojektien ketteryyden analysoinnissa, koska mallien toimivuus sekä hyvyys määrittelevät miten onnistuneita projektit ovat ja miten niissä voidaan edetä.

Koneoppimisjärjestelmiä kehittäessä voidaan päätyä tilanteeseen, jossa samaan ongelmaan voi olla useampia erilaisia ratkaisuvaihtoja ja datatietelijän on päätettävä mikä niistä on soveltuvin omaan ratkaisuun. Tällöin voidaan käyttää menetelmää nimeltä hyperparametrien viritys (eng. *hyperparameter tuning*), jonka tarkoituksena on löytää sellaiset mallin hyperparametrit eli malliin liittyvät parametrit. Tällöin on tarpeen jakaa opetusaineisto vielä yhteen joukkoon opetus-, testi- ja validointidatan lisäksi, jotta voidaan testata mallin toimivuutta sen avulla [7].

2.5 Koneoppimisen sovelluskohteet

Koneoppimismenetelmiä voidaan hyödyntää lähes millä tahansa teollisuuden tai tieteen alalla. Lääketieteessä koneoppimisen menetelmiä voidaan hyödyntää erilaisten sairauksien tunnistamiseen aineistoista, esimerkiksi syöpien tunnistaminen kuvista voidaan antaa koneoppimista hyödyntävän sovelluksen tehtäväksi [8]. Voisi ajatella, että kone joka oppii melko hyvin kuvista erilaiset syövät olisi tehokkaampi kuin ihminen jolla menee useampia vuosia tunnistaa kuvista erilaisia syöpiä.

Verkkokauppojen ja markkinoinnin koneoppimismenetelmiä käytetään esimerkiksi suosittelujärjestelmien toteuttamisessa. Suosittelujärjestelmällä tarkoitetaan järjestelmää, joka tarjoaa jonkin palvelun käyttäjälle suosituksia seuraavista toiminpiteistä tai ostettavista tuotteista sen perusteella mitä käyttäjä on aiemmin tehnyt tai ostanut. Tavoitteena näillä

järjestelmillä on saada luotua lisämyyntiä.

Nykyaikaiset matkapuhelimet tarjoavat koneoppimiseen perustuvia algoritmeja osana käyttöjärjestelmää, jolloin käyttäjä voi ohjata laitetta puheen avulla ja laitteen käyttöjärjestelmä ryhmittelee esimerkiksi kuvia niissä esiintyvien henkilöiden naaman perusteella.

Tekstin kääntäminen eri kielien välillä ja luonnollisen kielen tunnistaminen ja ymmärtäminen on myös koneoppimisen ongelma. Google Translate käyttää koneoppimismalleja luodakseen käännöksiä erilaisten testiaineistojen välille ja erilaiset asiakaspalvelurobotit pyrkivät koneoppimisen avulla tunnistamaan asiakkaan tarpeita tekstistä.

Uusimpia koneoppimisen sovelluskohteita on uusien kuvien sekä myös videoiden luominen aiemmasta datasta generoiden.

2.6 Koneoppimisprojektien tyypit

Tämän työn kyselytutkimuksessa, jota käsitellään luvussa 5, haastateltavat kuvailivat hyvin eri tyyppisiä koneoppimisprojekteja: osa projekteista on selkeästi kokeiluita joiden avulla pyritään arvioimaan teknologian soveltuvuutta käyttöön kun taas toiset projektit ovat enemmän ratkaisuja, jotka tukevat järjestelmäkokonaisuutta.

Lwakatare et al. [9] jakaa koneoppimista hyödyntävät kaupalliset sovellukset viiteen kategoriaan sen mukaan mitä sovelluksessa tehdään. Ensimmäisen tason sovellukset ovat kokeilut ja prototyypit, joiden tarkoituksena on arvioida miten hyvin jokin ongelma soveltuvuu ratkaistavaksi koneoppimismenetelmillä tai haluttujen tuloksien kokeiluja. Tavoitteena on selvittää miten hyvin olemassa olevat järjestelmät sekä esimerkiksi niiden käyttämä data soveltuu koneoppimisalgoritmien kehittämiseen.

Seuraavalla tasolla on ei-kriittinen tuotantoonvienti, jonka tarkoituksena on soveltaa prototyyppien ja kokeiluiden avulla luotuja malleja ei-kriittisessä tuotantoympäristössä sekä oppia miten tuotannossa oleva data tukee kehitettyjä malleja.

Kolmas taso on kriittinen tuotantoonvienti, jossa malleja sovelletaan myös kriittisemmissä järjestelmissä. Tämän tason sovelluksissa oleellista on se, että kehitettyä sovellusta voidaan tarkastella ja arvioida automaattisesti aina datan tuomisesta tuloksiin.

Neljännellä tasolla on ketjutetut järjestelmät, joissa koneoppimiskomponentti hyödyntää jonkin toisen koneoppimiskomponentin tuottamaa dataa omassa mallissaan. Tämä voi tarkoittaa käytännössä sitä, että koneoppimismalli hyödyntää vaikkapa sellaista dataa jota on siistitty jossakin toisessa järjestelmässä.

Viidennen tason järjestelmät ovat itsenäisiä koneoppimiskomponentteja, jotka toimivat hyvin muuttuvalla datalla prosesseissa jotka vaativat melko vähän ihmisen osallistumista. Luvussa 4.4.2 esitetyt MlOps ja Dataops-menetelmät mahdollistavat projektien autonomian.

3 OHJELMISTOKEHITYKSEN JA KONEOPPIMISEN PROSESSIMALLIT

Tässä luvussa perehdytään ohjelmistokehityksen sekä koneoppimisen prosessimalleihin.

3.1 Ohjelmistokehityksen prosessimalli

Sommerville [10] määrittelee ohjelmistokehityksen prosessimallin kokoelmana aktiviteetteja, joiden avulla toteutetaan toimiva ohjelmistotuote. Ohjelmistotuotteen kehittäminen voi tarkoittaa yksinkertaisimmillaan sitä, että ohjelmistokehittäjät kehittävät ohjelmistoja käyttämällä yleisesti käytössä olevia ohjelmointikieliä tai sitä, että integroivat sovellukseen valmiita komponentteja tai konfiguroivat valmista ohjelmistoa toimimaan halutulla tavalla. Joidenkin komponenttien tuottaminen itse ei ole aina tarkoituksen mukaista ja yritykset tarjoavat rajapintoja näiden tehtävien toteuttamiseen verkon yli joko ilmaiseksi tai maksua vastaan [11].

Ohjelmistoprosesseja voi olla erilaisia, mutta niissä kaikissa määritellä neljä vaihetta: ohjelmiston määrittely, ohjelmiston suunnittelu ja toteutus, ohjelmiston validointi sekä ohjelmiston muuntautuminen.

Ohjelmiston määrittelyn tarkoituksena on selvittää mitä ongelmaa tai tehtävää suunniteltavalla ohjelmistolla halutaan ratkaista, miten sen on toimittava sekä minkälaisia rajoituksia ohjelmistoon liittyy. Ohjelmiston määrittely voidaan toteuttaa vaatimusmäärittelyn avulla, jossa vaatimukset voidaan jakaa toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnallisten vaatimuksien tarkoituksena on kuvata mitä ohjelmiston on tehtävä kun taas ei-toiminnalliset kuvaavat muita ohjelmistoon liittyviä rajoituksia tai toiveita. Esimerkiksi koneoppimissovelluksen toiminnallinen vaatimus voisi olla "tunnistaa kuvasta kissanpennut"ja ei-toiminnallinen vaatimus "tunnistaminen tapahtuu nopeasti".

Ohjelmiston suunnittelu ja toteutus tarkoittaa sitä, että varsinainen ohjelmisto toteutetaan. Ohjelmiston suunnittelulla tarkoitetaan ohjelmiston rakenteen suunnittelemista siten, että ohjelmistotuotteen rakenne sekä erilaiset komponentit voidaan tunnistaa. Eri komponentit sekä niiden väliset suhteet voidaan mallintaa ja suunnitella hyödyntämällä erilaisia mallinnusmenetelmiä, kuten vaikkapa UML:n tarjoamia visuaalisia malleja tai prosessien mallintamista. Ohjelmiston toteutuksessa ohjelmistokehittäjät ohjelmoivat halutut toiminnallisuudet käyttämällä jotakin ohjelmointikieltä.

Ohjelmiston validointi tarkoittaa sitä, että ohjelmisto vastaa sille annettuja vaatimuksia. Tämä voidaan todeta esimerkiksi testauksen avulla. Testaus voidaan jakaa yksikkötestaukseen sekä integraatiotestaukseen. Yksikkötestauksen tarkoituksena on varmentaa yksittäisen funktion tai luokan oikeellisuus, kun taas integraatiotestauksen tarkoituksena tutkia toimiiko ohjelmiston komponentit keskenään oikein.

Koneoppimissovelluksen testaaminen perustuu luvussa 2 kuvattuun malliin, jossa malli opetetaan opetusaineistolla ja testataan testiaineistolla ja tämän hyvyyttä testataan.

Ohjelmistojen muuntautuminen tarkoittaa sitä, että ohjelmiston käyttötapaukset ja vaatimukset voivat muuttua sen elinkaaren aikana. Tämä vaihe ohjelmistoprosessissa vaatii ylläpitoa sekä ohjelmiston operointia eli esimerkiksi tuotantoonvientiä palvelimelle. Asiakas voi haluta uusia ominaisuuksia tai vanhoja ominaisuuksia voidaan poistaa, jos niitä ei tarvita. Ohjelmiston vaatimukset voivat myös muuttua kehityksen aikana.

Sommerville [10] jakaa ohjelmistoprosessit kolmeen kategoriaan: vesiputousmalli, inkrementaaliset prosessit sekä uudelleenkäytettävät ohjelmistot.

Vesiputousmalli on Roycen [12] kehittämä malli, jossa ohjelmistot kehitetään siten, että suunnittelu, toteutus, testaus ja tuotantoonvienti tehdään peräkkäisin vaihein. Roycen alkuperäisessä artikkelissa vesiputousmallia ei mainita nimeltä vaan termi on tullut jostain myöhemmin [13]. Tässä tutkielmassa Roycen kehittämää, vaiheittaista mallia kutsutaan kuitenkin vesiputousmalliksi. Vesiputousmalli on saanut paljon kritiikkiä julkaisunsa jälkeen raskautensa vuoksi, mutta sitä on saatettu myös käyttää väärin. Royce kuvaa alkuperäisessä vesiputousmallin artikkelissa, että mallin toteutus on riskialtis eli on riskinä, että sovelluskehityksessä sitä hyödynnetään väärin. Vesiputousmallia on myös hyödynnetty väärin, koska on malli on ymmärretty väärin niin että se mahdollistaisi ohjelmistojen tekemisen oikein yhdellä kerralla, ilman iteraatioita. Alkuperäisessä artikkelissa Royce [12] kehottaa tekemään kaiken kahdesti: ensimmäiseksi tehtävän sovelluksen pohjalta on kerättävä palautetta varsinaisilta käyttäjiltä ja muutettava sovellusta sen mukaan.

Inkrementaalisessa kehityksessä ohjelmistoa kehitetään pienissä palasissa saadun palautteen perusteella. Yksittäisen ohjelmiston osasen suunnittelu, kehitys ja validointi tapahtuu samanaikaisesti mikä mahdollistaa nopean palautteen saamisen toteutuista toiminnallisuuksista sekä nopean reagoinnin virheisiin. Nopean palautteen saamisen lisäksi inkrementaalisen kehittämisen hyötynä on edullisemmat kehityskustannukset sekä nopeampi ohjelmiston kehitys. Osiossa 4.2 käsiteltävät ketterät menetelmät perustuvat inkrementaaliseen prosessimalliin.

Uudelleenkäytetttävyyslähtöinen ohjelmistokehitysprosessimalli tarkoittaa sitä, että ohjelmistoa kehitetään käyttäen valmiita komponentteja tai ohjelmistoja, jotka konfiguroidaan ja muokataan omia tarpeita vastaavaksi. Esimerkiksi Salesforcen CRM-järjestelmät ovat valmiita sovelluksia, joita käyttäjät voivat konfiguroida omaan tarpeeseen vastaavaksi joko ohjelmakoodia kirjoittamalla tai käyttämällä visuaalisia työnkulkusovelluksia.

3.2 Koneoppimisprosessi

Koneoppimisen käyttöä varten on kehitettyä ohjelmistoprosessin kaltaisia prosessimalleja, joita voidaan hyödyntää kehitysvaiheiden kuvaamisessa. Koneoppimisessa käytettävät prosessimallit on kehitetty tiedon louhinnan ja data-analyysin malleista, koska laskentatehon ja saatavilla olevan datan määrä on kasvanut vasta viimeisten vuosien aikana GPU-laskennan yleistymisen vuoksi [1]. Koneoppimisen menetelmät ovat kuitenkin pohjimmiltaan samoja kuin vaikkapa tiedon louhinnassa, joten prosessien näkökulmasta niitä voidaan käsitellä samoina. Tässä osiossa käsitellyt mallit ovat sellaisia, että ne esiintyvät sekä tiedon louhintaa käsittelevässä kirjallisuudessa kuin koneoppimisen prosesseja käsittelevissä, uudemmissa tutkimuksissa.

Knowledge Data Mining eli KDD-prosessi oli ensimmäisiä prosesseja, jotka kehitettiin tiedonlouhintaa varten. Prosessin lähtökohtana on ollut kehittää menetelmä, jonka avulla voidaan systemaattisesti hakea tietoa tietokannoista sekä muodostaa niiden perusteella sellaista tietämystä, mitä liiketoiminnassa tarvitaan. KDD-prosessin vaiheita on tiedon hakeminen, esiprosessointi, muuntaminen käytettävään muotoon, tiedon louhinta sekä tulosten tulkinta. Menetelmä on iältään suhteellisen vanha, koska se on kehitetty jo vuonna 1996. Menetelmän iästä huolimatta siihen liittyy myös nykyään koneoppimisprojekteissa käytössä olevia menetelmiä kuten regressio tai luokittelu [14].

CRISP-DM eli Cross Industry Process for Data Mining -mallin [15] tarkoituksena on parantaa tiedonlouhintaprosessin kustannustehokkuutta sekä lisätä ymmärrystä prosessin eri vaiheista. CRISP-DM jakautuu kuuteen eri osaan:

- Liiketoiminnan ymmärtäminen
- Datan ymmärtäminen
- Datan valmistelu
- Mallinnus
- Mallin arviointi
- Tuotantoonvienti

Liiketoiminnan ymmärtämisellä tarkoitetaan sitä, että dataa käsittelevä henkilö ymmärtää liiketoiminnan tarpeet sekä sen miksi jotain mallia ollaan tekemässä. Liiketoimintatarpeisiin kuuluu ymmärtää mitä kustannuksia jostakin mallintamisesta tulee sekä minkälaisia oletuksia ja riskejä projektiin liittyy. Myös projektin suunnittelu kuuluu liiketoiminnan ymmärtämiseen.

Datan ymmärtämisen tarkoituksena on varmistua siitä, että mallintaja ymmärtää mistä datasta on kyse. Tämä voi tarkoittaa esimerkiksi sitä, että tiedetään tietokannan eri taulujen merkitykset sekä miten niihin kerätään tietoa. Myös datan laatu kuuluu datan ymmärtämiseen: mallinnuksessa täytyy ottaa huomioon, jos data on esimerkiksi puutteellistä tai siellä on puuttuvia arvoja. Puuttuviin arvoihin voidaan puuttuu esimeriksi imputoinnin avulla [1]. Datan ymmärtämiseen voidaan käyttää esimerkiksi kuvailevan tilastotie-

teen menetelmiä kuten erilaisten graafien piirtämistä tai lukujen taulukointia. Datan tilastollisen ymmärtämisen kannalta oleellista on esimerkiksi aineistossa olevat poikkeavat havainnot, koska ne voivat antaa osviittaa siitä onko data kerätty väärin tai onko siellä muunlaisia laadullisia ongelmia. Numeerisen aineiston muuttujien jakaumien ymmärtäminen voi vaikuttaa siihen minkälaisia malleja aineistoon voidaan soveltaa.

Datan valmistelulla tarkoitetaan sitä, että analyysiin tarvittava data on siinä muodossa että sitä voidaan analysoida ja että mukana on tarvittavat muuttujat. Datan hakeminen tietokannoista sekä muokkaaminen mallinnusta varten voi olla työlästä ja viedä yllättävän paljon aikaa. Joidenkin arvioiden mukaan jopa 80% datan kanssa käytetystä ajasta voi kulua siihen, että tehdään valmisteluja [16]. Datan valmistelussa on tärkeää myös se, että prosessiin otetaan mukaan vain tarpeellinen data: esimerkiksi EU:n yleinen tietosuoja-asetus sääntelee sitä miten henkilökohtaista dataa voidaan käyttää [17].

Mallintamisen tarkoituksena on se, että mallinnetaan ratkaisu haluttuun ongelmaan. Tämä tarkoittaa sitä, että valitaan oikea menetelmä tai algoritmi valittuun ongelmaan sekä kalibroidaan mallia ja sen parametreja tarpeen mukaan. Neuroverkkojen tapauksessa mallin kalibrointi voi tarkoittaa esimerkiksi mallin hyperparametrien optimointia.

Mallin arvioinnin tarkoituksena on arvioida miten hyvin mallinnuksen avulla voidaan ratkaista haluttu ongelma sekä arvioidaan sen hyvyys. Mallin arviointi tapahtuu arvioimalla ennustettuja tai luokiteltuja arvoja suhteessa todellisiin arvoihin kappaleen 2.4 menetelmien avulla.

Mallin tuotantoonviennin tarkoituksena on viedä mallin pohjalta tehtävä sovellus johonkin ympäristöön, jossa sitä voidaan hyödyntää eli käytännössä kehitetään jonkinlainen sovellus, joka hakee mallin tarvitsemaa dataa ja suorittaa koneoppimismallin toteuttaneen sovelluksen.

CRISP-DM -malli on hyvin dokumentoitu malli tiedonlouhinnan prosessista ja se on yhtenevä esimerkiksi Aurélien [18] kirjassa esitetyn mallin kanssa sekä Tensorflown verkkosivuilla esitettyihin tuotantomalleihin [19]. CRISP-DM-mallin ongelmana on kuitenkin se, että se ei huomioi lainkaan koneoppimissovellusta ohjelmistotuotteena, johon liittyy ohjelmistojen ongelmat.

Studer [20] on kehittänyt laajennetun mallin CRISP-DM-mallin pohjalta, jonka tarkoituksena on vastata myös laadullisiin ongelmiin tiedonlouhinnan prosessissa sekä huomioida koneoppimissovelluksen hallintaan liittyviä vaiheita, joita on sovelluksen ylläpito sekä monitorointi. Ylläpidon ja monitoroinnin tarkoituksena on huolehtia siitä koneoppimismallin vaiheesta, jossa mallinnuksen tuote on viety käytettäväksi sovellukseksi. Sovelluksen myötä ongelmaksi voi muodostua esimerkiksi muuttuva data, laitteiston vanheneminen sekä ohjelmistojen päivitykset.

Amershi [21] esittää koneoppimismallien kehittämisen yhdeksän vaiheen mallin:

- Mallin vaatimukset
- Datan kerääminen

- Datan puhdistaminen
- Datan merkitseminen
- Piirteiden mallintaminen "feature engineering"
- Mallin opettaminen
- Mallin arvioiminen
- Mallin tuotantoonvienti
- Mallin monitorointi

Amershin [21] mallin loppupään tuotantoonvienti sekä monitorointi voidaan nähdä ohjelmistokehityksen ongelmina. Tuotantoonvienti voi tarkoittaa esimerkiksi jonkin rajapinnan tai käyttöliittymän kehittämistä, jonka kautta kehitettyä mallia voidaan hyödyntää. Monitorointi tarkoittaa sitä, että mallin toimivuutta seurataan tuotantoympäristössä.

Microsoft on kehittänyt Team Data Science Process-mallin (lyh. TDSP) [22], jota kuvataan iteratiiviseksi ja ketteräksi malliksi älykkäiden sovelluksien tekemiseen. TDSP-mallin ajatuksena on luoda standardoitu rakenne analytiikkaa ja koneoppimista hyödyntäville sovelluksille sekä määritellä tarkasti mallin elinkaari, infrastruktuuri sekä tarvittava tiimi projektien luomiseen. Mallia kutsutaan ketteräksi, mutta ainakaan sitä esittelevillä verkkosivuilla ei ole kerrottu miten mallissa esiintyy ketterien menetelmien piirteita, joita on esitelty luvussa 4. Aiemmin esiteltyihin malleihin verrattuna TDSP kuvaa tarkemmin käytännön vaiheita, joita älykkäiden sovelluksien kehittämiseen liittyy sekä ottaa vahvasti kantaa esimerkiksi hakemistorakenteeseen sekä siihen minkälaisia rooleja projekteissa tarvitaan. CRISP-DM sekä Amershin mallin mukaisia kutsutaan TDSP-mallissa mallin elinkaareksi, joka on jaettu neljään osaan: liiketoiminnan ymmärtäminen, datan kerääminen ja ymmärtäminen, mallinnus sekä tuotantoonvienti. Nämä on jaettu pienempiin osiin, jolloin kokonaisuus vastaa CRISP-DM-mallia.

Aiemmin esiteltyjen mallien lisäksi on tutkittu myös sitä, miten käytännössä koneoppimisprojekteja tehdään pienissä yrityksissä. de Souca Nascimento et al. [23] on tutkinut minkälaisella prosessilla koneoppimisprojekteja tehdään pienissä yrityksissä. Tämän tutkimuksen tarkoituksena oli selvittää haasteita ja vaiheita, joita projekteihin liittyy ja tutkimus oli tehty haastattelemalla pieniä yrityksiä. Tämän tutkimuksen pohjalta vaiheet oli jaettu nelivaiheiseen malliin joka koostui ongelman ymmärtämisestä, käytettävän datan hallinnasta ja käsittelystä, mallin kehittämisestä sekä monitoroinnista. Yksittäiset vaiheet saattoivat sisältää tarkemmin määriteltyjä vaiheita aivan kuten TDSP-mallissa. Vaikka tässä tutkimuksessa oli vaiheita vähemmän, siinä nähtiin myös yhtäläisyyksiä esimerkiksi Amershin tutkimukseen ohjelmistokehitysmenetelmien käytöstä koneoppimisprojekteissa.

Prosessin eri vaiheet voidaan nähdä myös sen suhteen, minkälaisia kompetensseja tarvitaan kehitystiimiltä. Lakshmanan et al. esittävät kirjassaan *Machine Learning Design Patterns* [24] miten eri prosessin vaiheet riippuvat eri kompetensseista. Prosessin alkupään, eli mallien vaatimuksien kuin liiketoimintavaatimuksien määrittely vaatii datatieteili-

Koneoppimisen prosessimalli	Ylätason vaiheet	Erityispiirre
KDD	5	
CRISP-DM	6	
CRISP-DMQ	7	Mukana laadunvarmistus
Amershi	10	
Team Data Science Process	4	Määrittelee tarkasti roolit, vaiheet jakautuvat pienempiin osiin

Taulukko 3.1. *Koneoppimisen prosessimallit*

jältä enemmän liiketoiminnan ymmärtämistä kun taas prosessissa edetessä teknisemmät kompetenssit kuten ymmärrys erilaisien dataputkien, rajapintojen ja käyttöliittymien kehittämisestä korostuvat. TDSP-mallissa erilaiset roolit on määritelty erittäin tarkasti. Taulukossa 3.1 on vertailtu koneoppimisprosessien vaiheet ylätasolla.

Lwakataren et al. [9] taksonomiassa koneoppimisprojektin vaiheet on jaettu vain neljään kategoriaan: datan kokoamiseen, mallin luomiseen, mallin opettamiseen ja testaukseen sekä tuotantoonvientiin neljällä eri taksonomiatasolla, joita kuvataan kappaleessa 2.6 luvussa. Mitä monimutkaisempaan koneoppimiseen mennään esitetyllä taksonomiatasolla, sen teknisemmiksi eri vaiheisiin liittyvät haasteet muuttuvat. Alhaisella taksonomiatasolla, jossa kehitetään vain prototyyppejä tai testataan erilaisten menetelmien soveltuvuutta ongelmien ratkaisuun ongelmat liittyvät enemmän organisaatioihin sekä osaamiseen. Korkeilla taksonomiatasoilla ongelmia liittyy myös ohjelmistoteknisiin asioihin, kuten vaikkapa järjestelmän skaalautumiseen.

3.3 Ohjelmistokehityksen ja koneoppimisen prosessimallien vertailu

Kuvassa 3.1 esitetään yksinkertaistetusti miten ohjelmistotuotannon yleinen prosessimalli, CRISP-DM-malli sekä Amershin [21] malli suhtautuvat karkeasti toisiinsa. Yksinkertaistuksen vuoksi kuvasta puuttuu paluu aikaisempiin vaiheisiin. Eri prosesseja rinnakkain tarkastellen voi havaita eri vaiheiden välillä yhtäläisyyksiä, jolloin koneoppimissovelluksien kehitystä voidaan hahmottaa ohjelmistotuotannon näkökulmasta sekä hahmottaa käytännön kehittämisessä tarvittavia vaiheita sekä niissä käytettäviä työkaluja. CRISP-DM on prosessin vaiheiltaan karkeammin jaettu kuin Amershin malli koneoppimissovellusten luomisesta. Esimerkiksi datan ymmärtäminen ja valmistelu analyysiä varten eivät vastaa yksi yhteen toisiaan.

Vaatimusmäärittelyn rinnastaminen vaatimusmäärittelyyn Amershin mallissa sekä liiketoiminnan ymmärtämiseen CRISP-DM-mallissa voidaan rinnastaa siksi, että kaikessa on kyse siitä kehittäjä pyrkii ymmärtämään ongelman johon on joko rakentamassa ohjelmistoa tai kehittämässä koneoppimisratkaisua. CRISP-DM-mallin datan ymmärtäminen voidaan nähdä myös osana liiketoiminnan ymmärtämistä, koska käytettävissä olevan datan avulla koneoppimisongelma voidaan ratkaista tai sopiva menetelmä saadaan valittua. Myös ohjelmistokehityksen prosessimallin järjestelmän ja sovelluksen suunnittelu liittyy osin siihen mitä dataa on saatavilla, mistä sitä on saatavilla ja miten sitä täytyy mahdolli-

Kuva 3.1. *Ohjelmistokehityksen prosessimalli, CRISP-DM ja Amershin malli*

sesti muokata.

Ohjelmistoprosessin toteutusvaihe voidaan rinnastaa mallinnukseen sekä testaamiseen, koska näiden tehtävien tarkoitus koneoppimisprosessissa on ongelman varsinainen ratkaiseminen, aivan kuten ohjelmistokehityksessäkin toteutus vastaa varsinaisen ongelman toteuttamista. Koneoppimisprosessissa kuvataan mallin testaaminen, joka tapahtuu luvussa 2.4 esitetyillä menetelmillä. Koneoppimissovelluksissa hyödynnettävät algoritmit voidaan ajatella ohjelmistoiksi, joiden toiminta on aina oikein koska koneoppimismallit voidaan aina osoittaa matemaattisesti oikeiksi [25, 26]. Mallien formaalit matemaattiset todistukset eivät kuitenkaan ole tae siitä, että koneoppimista hyödyntävä sovellus käyttää algoritmia oikein [27]. Murphyn et al. [27] esittämän artikkelin mukaan koneoppimismalleja voi testata erilaisilla datajoukoilla, kuten sellaisilla joissa on puuttuvia arvoja tai muita ääriarvoja. Tämän testaustavan käyttö koneoppimisprojekteissa sopii yhteen luvussa 5 esitettyihin haastatteluihin, koska niissä nousi esiin tapauksia joissa koneoppimismallit eivät toimi välttämättä oikein kun ne viedään tuotantoon.

Integraatio- ja järjestelmätestausta ei ole esitetty CRIPS-DM-mallissa eikä Amershin malleissa, mutta sen voisi ajatella mallin monitorointiin sekä tuotantoonvientiin liittyvänä vaiheena. Luvun 5 haastatteluissa kävi ilmi, että koneoppimisen projekteja tehdään usein osana ohjelmistokehitysprojektia, tällöin integraatio- ja järjestelmätestauksen roolin voi nähdä siinä että niiden avulla testataa se, miten hyvin järjestelmä voisi tarjota koneoppimismallin tuloksia toiselle osalle sovellusta.

Koneoppimisen sekä ohjelmistokehityksen prosessimallit nähdään yleisesti koneoppimissovelluksien kehittämisen kontekstissa keinoina hahmottaa erilaisia työvaiheita projektissa. Esimerkiksi Martin Fowler [13] korostaa verkkosivuillaan olevassa kirjoituksessa vesiputousmallin merkitystä välineenä, jonka avulla isoja kokonaisuuksia voi pilkkoa pienempiin osiin.

Haastattelujen perusteella projektimalleja ei käytetä, mutta esimerkiksi kappaleen 4.4.2 kuvissa esitetään vaiheittain miten näiden järjestelmien tulisi toimia, joten järjestelmän eri komponenttien ja toiminnallisuuksien hahmottamisen kannalta voi olla hyödyllistä kuvata myös työn vaiheita. Luvussa 4 käsitellään Lean-metodologiaa niin organisaation kuin ohjelmistoprojektien näkökulmasta ja voisi myös ajatella, että organisaation tehostaessa toimintaansa myös tiettyjen prosessien vaiheiden sekä erityisesti niihin kuluvan ajan sekä mahdollisten pullonkaulojen näkökulmasta olisi hyödyllistä hahmottaa työn eri vaiheet.

4 KETTERÄT MENETELMÄT

Ketterien menetelmien teoriaa lähestytään kirjallisuudessa usein useasta eri näkökulmasta: ketteriin ohjelmistokehitysmenetelmiin, organisaation ketteryyteen sekä ketterien menetelmien soveltamiseen muihin aloihin kuten vaikkapa käyttöliittymäsuunnitteluun. Lisäksi perehdytään DevOps-ajatteluun, koska jatkuva kehitys sekä tuotannossa olevien järjestelmiene monitorointi nousi haastatteluissa esiin.

4.1 Lean yritysten johtamisessa

Organisaation ketteryyteen liittyvät menetelmät pohjautuvat ainakin kahteen lähtökohtaan. Ensimmäinen, perinteinen lähtökohta on Toyotan tuotantolaitoksilta lähtenyt ajattelu siitä, että tuotteiden teollisessa tuotannossa on pyrittävät vähentämään mahdollisimman paljon hukkaa prosesseissa sekä pyritään parantamaan laatua jatkuvasti [28, 29]. Toinen, hieman modernimpi lähtökohta on *Lean Startup*-ajattelu [30], jossa ajatuksena on se, että hyvin pitkälle vietyjen suunnitelmien sijaan organisaatio kehittää toimintaansa inkrementaalisesti, asiakkaan toiveita ja vaatimuksia kuunnellen sekä ohjelmistokehityksessä ketteriä menetelmiä hyödyntäen.

Toytotan tuotantolinjoilta alkunsa saanut tehokkuusajattelu korostaa sitä, että tuotantolinjat on kehitettävä mahdollisimman tehokkaiksi vähentämällä hukkaa tavoitteena hyödyntää työntekijöiden kapasiteettia mahdollisimman hyvin. Autoteollisuuden erityispiirre on siinä, että massatuotannossa tuotetaan tuotteita joissa on tuhansia osia ja asiakkaalla on mahdollisuus konfiguroida tuotetta haluamallaan tavalla. Autojen tuotantolinjoissa on useita aliprosesseja, joiden on toimittava hyvin yhteen ja siten oltava hyvin varautunut epätyypillisiin tilanteisiin. Toyotan tuotantolinjoilta on peräisin termi *Kanban*, joka tarkoittaa reaaliaikaista (eng. *Just-In-Time*) järjestelmää joka hyödyntää täysin henkilökunnan osaamisen. Kanban-järjestelmän on riippumaton mistään tietojärjestelmästä, jolloin tiedon prosessointiin liittyvät kustannukset vähenevät. Kanban-järjestelmän avulla voidaan reagoida myös nopeasti muuttuviin faktoihin, kuten tuotantoaikataulun muutoksiin tai tavarantoimitusten ongelmiin. Kolmas periaate on pitää huoli siitä, että kokoamisprosessin vaiheissa on vain sellainen määrä tarvittavia osia, ettei synny ylijäämää prosessissa. [28]

Toyotan tuotantolinjojen ketteryyden tai Lean-periaatteiden soveltaminen ei ole rajoittunut pelkästään fyysisiä tuotteita valmistavien yritysten työkaluksi, vaan sitä voidaan hyödyntää myös asiantuntijaorganisaatioiden työn ja prosessien kehittämisessä. Sari Torkkola kuvaa kirjassaan *Lean asiantuntijatyön johtamisessa* [31], että perinteinen tapa johtaa

organisaatioita ei välttämättä toimi asiantuntijayritysten johtamisessa, koska työn luonne voi olla hyvin monimutkainen sekä vaatimukset esimerkiksi johdolta ja asiakkaalta voivat olla vaativia. Asiantuntijatyössä vaihtelu - esimerkiksi työmäärien tai tehtävien - aiheuttaa prosessille eli asiantuntijayrityksen tapauksessa ihmisen ajattelulle, ylikuormitusta josta seuraa hukkaa. Torkkolan kirjassa tätä kuvataan syklisenä prosessina, jolloin hukka tuottaa lisää vaihtelua ja sitä myötä ylikuormitusta ja edelleen hukkaa. Asiantuntijatyössä korostuu myös työn läpinäkyväksi tekeminen sekä virtaustehokkuuden korostaminen. Työn läpinäkyväksi tekeminen tarkoittaa sitä, että on olemassa jokin visuaalinen keino havainnollistaa koko organisaatiolle mitä ollaan tekemässä tällä hetkellä ja mitä on jo tehty. Virtaustehokkuudella tarkoitetaan sitä, että organisaatio pystyy tekemään tehtäviä mahdollisimman nopeasti: tavoitteena on se, että organisaatiossa on mahdollisimman vähän tehtäviä työn alla samanaikaisesti. Tehtävien tekemistä organisaatiossa voidaan ajatella jonoteorian mallina, jossa on useita tekijöitä ja tehtäviä saapuu tietty määrä tietyssä ajassa [32].

Organisaatiot voivat kehittää toimintaansa ketteräksi hyödyntäen arvovirtojen mallinnusta. Arvovirralla tarkoitetaan sellaista yrityksen prosessia, joka tuottaa jotain arvoa yritykselle. Arvovirtakuvauksien tarkoituksena on kuvata visuaalisesti mitä työtä organisaatiossa tehdään ja löytää sieltä keinoja organisaation prosessien tehostamiseen Esimerkiksi, teollisuusyritys voi kuvata arvovirtaprosessin avulla materiaalin toimituksen tehtaalle sekä tehtaalla tapahtuvat toimenpiteet, joilla tuote valmistetaan raaka-aineesta. Arvovirtamallinnuksessa määritellään aluksi organisaation prosessien nykytila, jossa näkyy kuinka kauan yksittäisiin tehtäviin menee aikaa ja pyritään ymmärtämään prosessien pullonkaulat. Alkutilassa määritellään prosessin eri vaiheet sekä niihin kuluva työaika (eng. *Process Time*) sekä odotusaika (eng. *Lead Time*), joka kuvaa sitä kuinka kauan jokin työ odottaa ennen kuin sen voi aloittaa [33]. Luvussa 3 esitetty ohjelmistokehityksen prosessimalli tai CRISP-DM-menetelmä voitaisiin kuvata myös arvovirtakaaviona, jolloin näitä menetelmiä hyödyntäville organisaatioille voisi hahmottua tarkemmin se kuinka paljon mihinkin vaiheeseen menee aikaa projektissa. Kirjan [33] esimerkissä arvovirran mallinnusta on esimerkinomaisesti sovellettu ohjelmistoprojektin muutospyyntöjen mallintamiseen Scrum-menetelmää käyttävälle tiimille.

4.2 Ketterät menetelmät ohjelmistokehityksessä

Edellisessä osiossa käsiteltiin lyhyesti sitä, miten Lean-ajattelun ja ketterien menetelmien käyttö hyödyntää asiantuntijaorganisaatioita. Tässä kappaleessa perehdytään ketteriin menetelmiin ohjelmistokehityksessä.

Ohjelmistokehitysprojektien ketterät menetelmät ratkovat samoja ongelmia kuin perinteisimmissä organisaatioissa: projekteissa voi olla ylimääräisiä askeleita tai hidastavia tekijöitä, joista olisi syytä päästä eroon ja sitä kautta tehostaa ohjelmistoprojekteja. Ohjelmistokehityksessä kehitettävät tuotteet eivät kuitenkaan ole samanlaisia kuin teollisessa tuotannossa, sillä kehitysprojektien lopputuotteet voivat vaihdella suurestikin. Tämän vuok-

si ohjelmistokehitysprojektien johtamista ei voi ajatella samaan tapaan kuin autojen tai jonkin muun fyysisen tuotteen kehittämistä. Cantor et al. [32] esittää artikkelissaan, että ohjelmistoprojekteja tulisi mallintaa tuotantolinjojen sijaan samaan tapaan kuin tietoverkkoja mallinnetaan: työtehtävät ovat kuin verkon yli kulkevia tietoliikennepaketteja, jotka saapuvat reittitimeen ja reititetään eteenpäin. Saapuvat työtehtävät priorisoidaan ja aikataulutetaan sovituilla kriteereillä ja ohjataan kehittäjille eteenpäin. Osa tehtävistä voidaan palauttaa takaisin liiketoiminnalle, jos niissä on esimerkiksi kysyttävää tai tarkennettavaa. Vaikka yhdenmukaisuus verkkoteknologian kanssa ei ole täydellinen, sen avulla voi pyrkiä hahmottamaan ohjelmistokehityksen eroavaisuutta teollisesta tuotannosta.

Poppendieckit [34] esittävät seitsemän hukkaa (eng. *waste*) eli hidastavaa tekijää, joita ohjelmistoprojekteissa esiintyy. Näitä on osittain tehty työ, ylimääräiset prosessit, ylimääräiset ominaisuudet, vaihtelu tehtävien välillä, odottelu, liike sekä ongelmat. Osittain tehty työ tarkoittaa sitä, että jokin tehtävä on tehty vain osittain eli se ei ole vielä valmis tai kukaan ei tiedä miten se pitäisi saada tehtyä valmiiksi. Ylimääräiset prosessit tarkoittavat kaikkea sellaista paperityötä sekä vaiheita, jotka vaativat ylimääräistä työtä.

Ylimääräisillä ominaisuuksilla tarkoitetaan ominaisuuksia, jotka eivät ole toiminnallisuuden kannalta keskeisiä mutta halutaan kuitenkin mukaan projektiin. Toiminnallisuus voi olla esimerkiksi sellainen, että sen toiminnallisuutta halutaan testata tai ominaisuus on sellainen, että se on ollut mukana esimerkiksi vanhemmassa järjestelmässä ja halutaan mukaan myös uuteen ilman oikeaa tarvetta.

Tehtävien vaihtelu voi johtua esimerkiksi siitä, että kehittäjä kehittää useampaa projektia samaan aikaan tai on useampia, saman projektin sisäisiä tehtäviä samaan aikaan. Työn vaihteleminen aiheuttaa ongelmia sen suhteen, ettei kehittäjä voi tietää mitä tehtävää milloinkin olisi tehtävä. Ratkaisuna työn vaihteluun on niin kutsuttu pull-ajattelu, joka mahdollistaa sen että työntekijä tietää täsmälleen mitä on tekemässä kun tulee töihin ja valitsee itselleen uuden tehtävän työjonosta. Poppendieckien kirjan [34] mukaan tehtävien vaihtelua voidaan ratkaista myös jonomaisella ajattelulla, jossa yksittäisen työntekijän työtä tarkastellaan jonoteorian avulla siten, että työtä tulee tehtäväksi ja sitä valmistuu tietyn ajan kuluttua jonka jälkeen voidaan ottaa uusi tehtävä tehtäväksi. Poppendieckit [34] väittävät kirjassaan myös, että nopein tapa saada kaksi projektia valmiiksi samalla tiimillä on tehdä ne erikseen eikä samaan aikaan.

Liikkeellä tarkoitetaan sitä aikaa, joka menee esimerkiksi siihen kun kehittäjä saa vastauksen kysymykseensä tai miten muut projektin tekijät liikkuvat. Ongelmilla tarkoitetaan havaittuja puutteita ohjelmistotuotteessa: ne voivat olla bugeja tai muita ongelmia. Tavoitteena on, että ongelmat löydetään mahdollisimman nopeasti. Liikkeellä voidaan tarkoittaa myös fyysistä, oikeassa maailmassa tapahtuvaa liikettä kun ohjelmistokehittäjä joutuu siirtymään vaikkapa huoneesta toiseen kysyäkseen asioita.

Odottelu on nimensä mukaisesti odottamista. Kehittäjä voi joutua odottamaan, että saa vaikkapa määrittelyt asiakkaalta tai koneoppimisalgoritmin kehittäjä voi joutua odottamaan että pääsee käsiksi oikeaan dataan, jota voi hyödyntää kehitystyössään. Käytän-

nön projekteissa voi olla myös tilanteita, että projektissa mukana olleet henkilöt voivat joutua odottamaan omaa vuoroaan, jos he tekevät jotain hyvin erikoistunutta tehtävää.

Ongelmat tarkoittavat mitä tahansa ongelmia, joita ohjelmakoodissa on ja jotka estävät etenemisen projektissa. Ongelmat pitäisi löytää ja korjata mahdollisimman pian, jotta päästään jatkamaan tekemistä. Odottelu ja ongelmat ovat kulkevat keskenään hyvin käsikädessä: ongelmat voivat aiheuttaa odottelua, jolloin ongelmien pikainen korjaaminen parantaa projektien läpivientiaikaa. Käytännössä ohjelmistokehityksessä ongelmat löydetään kehitysvaiheessa testaamisen avulla, mutta voi olla myös sellaisia ongelmia jotka ilmenevät vasta kun sovellus on todellisilla käyttäjillään.

Poppendieckien kirjassa tarjotaan monia keinoja ohjelmistoprojektien käytäntöjen parantamiseen sekä hukan minimointiin. Käytettävät menetelmät korostavat oppimista, ihmisten välistä kommunikointia sekä päätöksentekoa projekteissa. Monet menetelmät ovat samankaltaisia kuin Torkkolan [31] esittämät asiantuntijaorganisaatioiden toimintaan liittyvät Lean-ajattelumallit.

Vuonna 2001 joukko tunnettuja ohjelmistokehittäjiä julkaisivat teoksen nimeltä *Agile Manifesto* [35]. Manifestin tärkein sisältö on se, että ohjelmistoprojekteissa pitäisi korostaa ihmisiä ja kommunikaatio ennemmin kuin prosesseja tai työkaluja, toimivaa sovellusta ennemmin kuin kattavaa dokumentaatiota, yhteistyötä asiakkaiden kanssa sopimusten sijaan sekä muutoksiin reagointia valmiin suunnitelman seuraamisen sijaan. Jälkimmäisten merkitys ymmärretään, mutta ensimmäisiä korostetaan.

Ohjelmistokehityksessä ketterät menetelmät sopii erityisen hyvin pienien ja keskisuurien organisaatioiden ohjelmistokehitykseen sekä sellaisiin projekteihin, joissa asiakasorganisaatio on valmis osallistumaan kehitykseen yhdessä kehittäjien kanssa [10]. Ketterää kehitystä varten on kehitetty useampia menetelmiä, jotka koostuvat prosessista, rooleista ja vastuista sekä käytännöistä [36]. Isojen organisaatioiden käyttöön on kehitetty ketteristä menetelmistä versioita, jotka tukevat ja asettuvat yhteen isojen organisaatioiden päätöksentekoprosessien kanssa. Tällaisia menetelmiä on esimerkiksi Scaled Agile Framework (SaFE). SaFE-menetelmä perustuu ajatukselle, jossa organisaatiota johdetaan Lean-periaatteiden mukaisesti jolloin sovelluskehitys saadaan toimimaan ketterästi myös isossa organisaatiossa [37].

Ketterien ohjelmistokehitysmenetelmien prosessit määrittelevät miten esimerkiksi ohjelmiston suunnittelu sekä toteuttaminen viedään eteenpäin tietyssä menetelmässä. Joissakin ketterissä menetelmissä, kuten Scrumissa, on määritelty rooleja joita projekteissa tarvitaan. Tuoteomistaja (eng. *Product owner*) vastaa kehitettävästä tuotteesta ja edustaa projektissa asiakasta tai liiketoiminnan vaatimuksia. *Scrum Master* pyrkii vastaamaan siitä, että projektissa työskentelevillä on mahdollisuus tehdä työnsä mahdollisimman hyvin.

Scrum-menetelmään kuuluu myös käytäntöjä, joista tärkein on sprintti eli tapauskohtaisesti joko kahden tai kolmen viikon jakso, jolloin tehdään sovitut tehtävät. Jokaista sprinttiä edeltää suunnittelu, jossa määritellään mitkä tehtävät tulevat kyseiseen sprinttiin ja

mitkä jäävät tulevaisuuteen. Suunnittelussa on aina koolla koko tiimi sekä liiketoiminnasta vastaavat henkilöt, erityisesti tuoteomistaja jonka tehtävänä on vastata siitä että suunnitteluun otetaan mukaan liiketoiminnan kannalta tärkeimpiä ominaisuuksia. Kehittäjien rooli suunnittelukokouksessa on arvioida kuinka paljon työtä yksittäisiin tehtäviin menee ja kuinka paljon pystyvät ottamaan seuraavaan sprinttiin mukaan. Scrumissa pidetään päivittäin *Daily Scrum*, jossa tiimin jäsenet kertovat mitä ovat tehneet eilen, mitä aikovat tehdä tänään sekä käyvät nopeasti läpi mitä ongelmia ovat kohdanneet. Sprintin loppupuolella järjestetään katselmointi (eng. *Scrum Review*), jossa käydään läpi mitä on saatu valmiiksi ja sprintin jälkeen käydään läpi retrospektiivi, jossa nimensä mukaisesti katsotaan taakse mitä olisi voinut tehdä toisin ja mikä meni hyvin. Tämän perusteella pyritään parantamaan käytäntöjä [38]. Näitä toimenpiteitä Scrumissa kutsutaan rituaaleiksi.

Toinen yleisesti käytössä oleva menetelmä ketterään ohjelmistokehitykseen on Kanban, joka perustuu pohjimmiltaan luvussa 4.1 esitettyyn saman nimiseen Kanban-menetelmään. Ohjelmistokehitysprojekteissa käyttö perustuu siihen, että yksittäisen kehittäjän työmäärää rajataan, työmäärät tehdään koko tiimille näkyväksi, prosessit tehdään selkeiksi ja kehitystä tehdään yhdessä hyödyntäen tieteellisistä menetelmää (eng. *scientific method*) [39]. Kanban-menetelmää käyttäessä ohjelmistotiimillä on usein näkyvissä koko tiimin työmäärä, joko jossakin sovelluksessa tai vaihtoehtoisesti ohjelmistokehitystiimin fyysisen tilan seinällä, riippuen siitä miten työ on organisoitu.

Ketterien menetelmien hyvien puolien lisäksi niiden käyttöön voi liittyä riskejä tai ongelmia. Millerin [40] mukaan ketterän kehityksen ongelmat voidaan jakaa neljään pääkategoriaan: viestintään ja muutosjohtamiseen, asiakkaiden ja johdon kykyyn hyödyntää menetelmiä, päivittäisiin ongelmiin sekä osaamattomuuteen. Ketterässä projektissa toimivat kehittäjät voivat olla haluttomia kommunikoimaan asiakkaan kanssa tarpeeksi hyvin, jotta projektin muutosvaatimukset sekä haasteet tulisi käsiteltyä ketterästi. Johdon ja asiakkaiden kyky hyödyntää ketteriä mentelmiä voi olla puutteellinen jos ketterään projektiin otetaan mukaan vain kehittäjiä tai siihen kohdistuu ulkoisia paineita. Päivittäiset ongelmat voivat liittyä esimerkiksi ongelmiin ohjelmistojen kehittämisessä, liian suureen työmäärään tai testaamisen puutteeseen. Organisaatio, joka hyödyntää ketteriä menetelmiä ei välttämättä hallitse menetelmien käyttöä tarpeeksi hyvin eikä osaa valita projektiin sopivaa menetelmää oikein. [40]

Tuoreempaa näkökulmaa ketterien menetelmien käyttöön edustaa niin kutsuttu *Modern Agile*-lähestymistapa, jossa yksittäisten ketterien menetelmien ja perinteisten käytäntöjen sijaan pyritään luomaan ohjelmistoyrityksistä kulttuureiltaan sellaisia, että autetaan ihmisiä loistamaan, kokeillaan ja opitaan nopeasti, toimitaan jatkuvasti arvoa sekä luodaan turvallisuudesta perusperiaate. Näiden tavoitteiden tarkoituksena on saada työryhmät toimimaan siten, että niissä työskentelevien on turvallista oppia [34] ja kokeilla sekä tuottaa mahdollisimman hyvin arvoa asiakkaille. [41] Nämä menetelmät ovat hyvin lähellä kappaleessa 4.4 esitettävää DevOps-kulttuuria ja Westrumin organisaatiomallia.

4.3 Ketterät menetelmät UX-kehityksessä

Vaikka ketterien menetelmien käyttö ja soveltaminen on hyvin tunnettua ohjelmistokehityksessä, on olemassa sellaisia projektien osa-alueita, joissa ketterien menetelmien käyttöä on tarkasteltu erikseen irrallaan ohjelmistokehityksestä. UX-suunnittelulla tarkoitetaan käyttäjäkokemuksen suunnittelua, jonka tarkoituksena on differentioida sovellusta tai palvelua eli tehdä tuotteesta sellainen, että se erottuu kilpailijoiden tuotteista [42]. Ketterien menetelmien käyttöä UX-suunnittelussa käsitellään tässä työssä koska se antaa erilaisen kuvan ketterien menetelmien käyttöön ohjelmistoprosesseissa kun tehtävä työ on luonteeltaan muunlaista kuin ohjelmistokehitystä. Tavoitteena on pyrkiä löytämään lainalaisuuksia sekä yhteyksiä koneoppimisprojekteihin.

Seiden ja Gothelf [43] määrittelevät kirjassaan *Lean UX* lean-periaatteita hyödyntävän UX-suunnittelun niin, että se pyrkii purkamaan niin kutsuttuja siiloja ohjelmistokehityksen sekä UX-suunnittelun väliltä, panostaa yhteistyöhön tuotteita kehittävän tiimin sisällä yksittäisten henkilöiden esiinnostamisen sijaan sekä pyritään mahdollistamaan jatkuva kehittyminen sekä oppiminen.

Lean UX-kirjassa [43] lean-menetelmien hyödyntämistä tiimeissä korostetaan monimuotoisten, pienien ja itseorganisoitujen tiimien kautta. Näiden tiimien tehtävänä on ymmärtää ongelmat, joita ovat ratkaisemassa ja jakaa tietoa mahdollisimman hyvin keskenään.

Kirjassa esitetään myös prosesseja sille, miten lean-ajattelua voidaan hyödyntää olemassa olevissa tiimeissä. Kaikki tekeminen lähtee siitä, että tiimillä on jokin hypoteesi siitä mitä ollaan tekemässä ja näitä hypoteeseka pyritään testaamaan.

Isomursu [42] on tutkinut haastatteluilla isossa telealan yrityksessä UX-työn yhdistämistä ketteriin menetelmiin. Keskeisinä tuloksena tutkimuksessa on ollut se, että UX-työ on saatu ketterämmäksi kun UX-suunnittelijat ovat lähteneet työskenetelemään osana ohjelmistokehityksen Scrum-tiimiä irrallisen, oman tiiminsä sijaan. Hyötynä muun ohjelmistokehitystiimin kanssa työskentelyssä oli se, että UX-tiimi oli lähellä tekijöitä, mutta ongelmana nähtiin se että tällöin käyttäjäkokemus saattoi muuttua isossa organisaatiossa sen mukaan missä tiimissä sitä kehitettiin. Toinen tapa yhdistää UX-työ sekä ohjelmistokehitys oli tehdä UX-työ omissa sprinteissään, mutta osallistua ohjelmistokehitystiimien suunnitteluihin. Toinen Isomursun havainto oli että suunnittelutyö ei pysynyt ohjelmistokehityksen aikataulussa: suunnittelu vaati aikaa visualisointiin, prototyyppien ja hahmotelmien tekemiseen ennen kuin tuloksia oli mahdollista hyödyntää ohjelmistokehityksen tarpeissa. Myös kommunikaatio ohjelmistotiimien kanssa, esimerkiksi maantieteellisen etäisyyden vuoksi. Haastattelun johtopäätelmänä oli se, että UX-työn saaminen ketteräksi oli haastavaa edellä mainittujen ongelmien vuoksi.

Isomursun [42] artikkelin tarkastelu Gothelfin [43] kirjan tarjoamien keinojen kautta vaikuttaa siltä, että ketterien menetelmien käyttö UX-suunnittelussa saattaa olla haastavaa mutta kirja pyrkii tarjoamaan systemaattisia menetelmiä siihen, miten ketteryyttä UX-suunnittelussa voisi lisätä. Organisaation ketteryyden, kuten arvovirtojen mallintamisen

näkökulmasta tarkasteltuna, UX-suunnittelun ketteryyden parantamiseksi pitäisi avata työhön liittyviä prosesseja tarkemmin sekä pyrkiä löytämään UX-työn sekä ohjelmistokehityksen väliltä pullonkaulat, jotka aiheuttavat ongelmia.

4.4 DevOps, DataOps ja MlOps

4.4.1 Devops

DevOpsin määritelmä ei ole aivan täysin vakiintunut [44], mutta yksinkertaisimmillaan se tarkoittaa ohjelmistokehityksen organisointia siten, että ohjelmistojen kehitystyö testaamisineen ja operointi eli tuotantoon vienti, palvelimien tai ympäristöjen ylläpito sekä monitorointi ovat lähellä ohjelmistokehitystiimin toimintaa siten, että ohjelmistoja voidaan kehittää mahdollisimman nopeasti. [45]

The Devops Handbook-kirjassa [45] esitetään ajatus kolmesta periaatteesta (eng. *Three ways*) [46], joihin DevOps perustuu. Ensimmäinen periaate perustuu systeemiajatteluun eli organisaation on ajateltava tekemistä kokonaisuutena, lukkiutumatta tiettyihin siiloihin. Tavoitteena on saada organisaatio ymmärtämään, miten yksittäisten käyttötapauksien toteuttaminen kulkee koko yrityksen arvovirran läpi alkaen liiketoiminnan vaatimuksista, päättyen järjestelmän operointiin. Tavoitteena on se, että vältytään yksittäisen tekijän optimoinnilta ja sen vaikutukselta järjestelmään. Ensimmäiseen periaatteeseen kuuluu vahvasti virran ymmärtäminen. Työhön liittyvän virran ymmärtäminen perustuu siihen, että työ on tehty näkyväksi, rajataan sitä miten paljon työtä on kesken kerrallaan, pienennetään erien kokoa, vähennetään ohjeistuksien määrää sekä pyritään ymmärtämään jatkuvasti prosessin rajoitteet. Nämä periaatteet vastaavat suoraan niitä periaatteita, joita esitettiin kappaleessa 4.2 ja näiden tavoitteena on vähentää kehitysprosessissa olevaa hukkaa.

Toinen periaate perustuu siihen, että vahvistetaan palautesyklejä eli pyritään ottamaan vastaan palautetta toiminnasta ja parantamaan tekemistä prosessin aikana. Nopean palautesyklin tavoitteena on se, että pyritään lisäämään tietoisuutta ja autetaan organisaation tai projektin jäseniä oppimisessa. Nopea palautteen saaminen sekä organisaation oppiminen ovat erityisen tärkeitä periaatteita silloin kun ollaan tekemässä monimutkaisia projekteja, koska usein ongelmat ovat nopeampia ratkaista silloin kun ne eivät ole vielä paisuneet suureksi.

Kolmas periaate kannustaa organisaatioita oppimiseen sekä kokeilukulttuuriin. Tämän periaatteen taustalla on idea siitä, että oppiakseen uusia asioita ihminen tarvitsee toistoja sekä myös virheitä, joista voidaan oppia. Organisaatio, joka kannustaa oppimaan ja on valmis kokeilemaan voi olla kehittäjille psykologisesti turvallisempi paikka työskennellä: tämä tarkoittaa sitä, että työntekijät uskaltavat reagoida virheisiin paremmin eivätkä he pelkää virheistä johtuvia seurauksia. Tämä saattaa parantaa ohjelmistojen laatua.

Näiden periaatteiden perusteella DevOps voidaan nähdä eräänlaisena kulttuurina, joka tukee ketteriä menetelmiä ja hyödyntää lean-ajattelun periaatteita. Toinen lähestymistapa

DevOpsiin perustuu työkaluihin, joiden avulla teknologinen arvovirta saadaan nopeammaksi automaation avulla. Näitä välineitä on esimerkiksi automaattinen testaus, versionhallinta sekä jatkuvan tuotantoonviennin ja integraation (CI/CD) työkalut. [47]

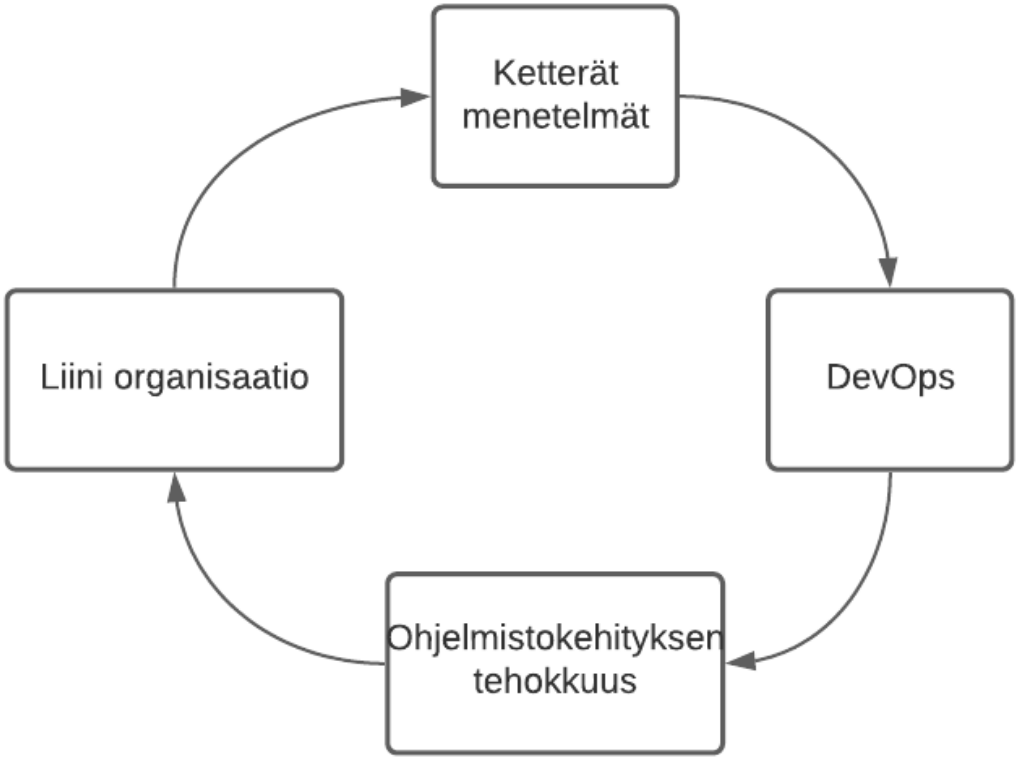
Teknologian näkökulmasta edellä mainitut kolme periaatetta tulee integroida päivittäiseen tekemiseen siten, että mittaamiseen, oppimiseen sekä nopeaan palautteen antamiseen on olemassa joko jokin työkalu mikä tukee tekemistä sekä työtavat, joilla näihin periaatteisiin päästään. Näiden rakentaminen on sekä teknologinen että organisaation kehittämisen ongelma [45, 47].

DevOps-menetelmien käytön hyötyä on tutkittu ja on todettu, että ohjelmistokehityksen tehokkuus vaikuttaa myös organisaation toimintaan. Mitä nopeammin organisaatio pystyy vastaamaan ohjelmistokehityksen tarpeisiin, sen tehokkaammin organisaatio toimii. Toisaalta, organisaation DevOps-käytäntöjen hyödyntäminen täysin edellyttää organisaatiolta oikeanlaista suhtautumista työntekijöihin sekä kulttuuriin. Yhdysvaltalaisen sosiologin Ron Westrumin organisaatiokulttuuriteorian mukaan organisaation kulttuuri vaikuttaa siihen, miten hyvin informaatio kulkee organisaatiossa. Tutkimuksessaan Westrum havaitsi, että hyvin toimivassa organisaatiossa kysymyksiin saadaan vastauksia tarvittaviin kysymyksiin ajallaan sekä siten, että informaatio on siinä muodossa että sitä tarvitseva henkilö pystyy informaatiota hyödyntämään parhaalla mahdollisella tavalla [48, 49, 50]. Tutkimuksen [50] mukaan Westrumin organisaatioteoria ennustaa ohjelmistokehityksen sekä organisaation tehokkuutta. Saman tutkimuksen mukaan monet organisaatioiden johtamiseen sekä ihmisiin liittyvät tekijät parantavat ohjelmistokehityksen tehokkuutta. Tämän perusteella on perusteltua ajatella, että organisaation johtamiseen sekä Lean-ajattelun hyödyntämiseen liittyvät asiat tukevat myös ohjelmistokehityksen sekä ketterien menetelmien käyttöä. Kuvassa 4.1 on hahmoteltu organisaation ketteryyden (Lean-menetelmien käytön), ketterien menetelmien käytön sekä ohjelmistokehityksen tehokkuuden yhteyttä syklisessä muodossa.

4.4.2 DataOps ja MlOps

DataOpsin määritelmä ei ole ollut vakiintunut [51], mutta sen voidaan katsoa olevan ketterien menetelmien sekä edellisessä kappaleessa käsiteltyjen DevOps-menetelmien soveltamista erityisesti tietokantoihin, analytiikkaan sekä tiedonlouhintaan liittyviin projekteihin ja liiketoimintatarpeisiin [52]. Atwalin [52] kirjan *Practical Dataops* perusteella tämän nimityksen alle kuuluu organisaation useat, erilaiset dataan liittyvät prosessit kuten esimerkiksi tiedon hakeminen tietokannoista, tietoon liittyvä oikeuksien ja pääsynhallinta sekä datan puhdistaminen ja muut hyödyntämiseen liittyvät prosessit.

MlOps tarkoittaa sitä, että koneoppimisprojekteissa hyödynnetään DevOpsin periaatteita kuten jatkuvaa integraatiota ja tuotantoonvientiä sekä muita DevOps-periaatteita [53]. Blogikirjoituksen [53] mukaan MlOps-menetelmillä pyritään vastaamaan koneoppimissovellusten kehittämisen erityispiirteisiin. Koneoppimisprojektin erityispiirteet liittyvät siihen, että malleja on usein kehittämässä datatieteilijät sekä datainsinöörit, joilta voi puuttua

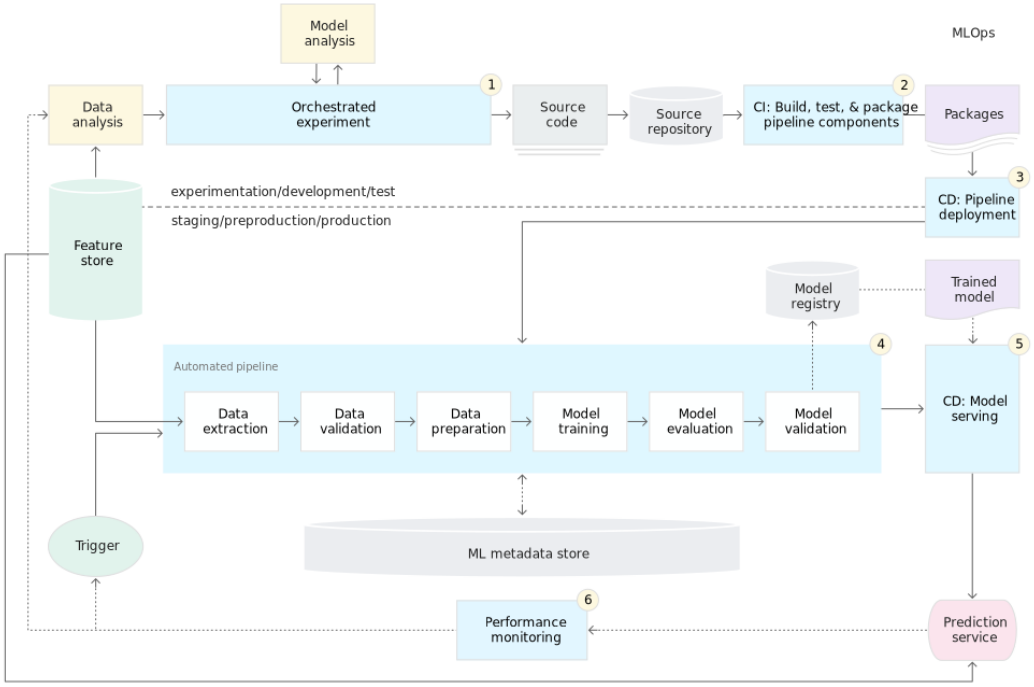


Kuva 4.1. *Visuaalinen hahmotelma organisaation toiminnan, ketterien menetelmien ja DevOpsin yhteydestä*

osaaminen liittyen perinteiseen ohjelmistokehitykseen. Lisäksi koneoppimisen iteratiivisen sekä kokeilevan luonteen vuoksi on tärkeää ylläpitää tietoa siitä minkälaiset mallit toimivat ja minkälaisen datan kanssa. Koneoppimismallien on myös toimittava tuotannossa mikä edellyttää sitä, että mallien toimintaa on monitoroitava jatkuvasti.

MlOps- sekä DataOps-periaatteissa on jonkin verran yhtäläisyyksiä kappaleessa 3.2 kuvattujen prosessien kanssa. Kuvassa 4.2 esitetään näkemys siitä, miltä MlOps-periaatteen mukaan kehitetty automaatiojärjestelmä koneoppimisprojektille voisi näyttää. Kuvan perusteella voidaan tunnistaa myös koneoppimissovellusten kehittämiseen liittyviä kompetensseja sekä järjestelmiä, joista sovellukset ovat riippuvaisia. Kuvan perusteella rooleina voisi nähdä ainakin datatieteilijän tai analytiikon tehtävät, koneoppimiskehittäjän sekä ohjelmistokehittäjän tehtävät. Datatieteilijän ja analytiikon roolina voidaan nähdä itse mallin kehittäminen sekä analysointi, koneoppimis- tai datainsinöörin tehtäviksi mallin tuotantoonvienti sekä dataan liittyvät tehtävät ja ohjelmistokehittäjän tehtäviin palvelun tarjoaminen sovelluksen muille kehittäjille.

Koneoppimisprojektien näkökulmasta DataOps ja MlOps vaikuttavat hyvin samankaltaisilta periaatteilta, mutta niiden eroavaisuudet ovat siitä mitä ongelmaa ollaan ratkaisemassa. Ohjelmistojen osana olevien koneoppimisratkaisujen näkökulmasta DataOps voidaan nähdä enemmän järjestelmässä käytettävän tiedon operatiivista käsittelyä organi-



Kuva 4.2. *CI/CD ja automatisoitu koneoppimisprosessi [53]*

saatiossa kun taas MlOps voidaan nähdä enemmän koneoppimismallin käyttöön sekä kehittämiseen liittyvänä menetelmänä.

4.5 Ohjelmistokehitysmenetelmän valinta

Ohjelmistoprojekteilla on kyseenalainen maine epäonnistuvina projekteina ja projektin epäonnistumisella voi erittäin suuria liiketoiminnallisia vaikutuksia yritykselle [54]. Ohjelmistoprojektin epäonnistuminen johtuu yleensä siitä, että organisaatiolla kuluu liikaa rahaa projektin toteuttamiseen, jolloin projekti joko joudutaan keskeyttämään tai vaihtoehtoisesti projektin lopputulemat eivät vastaa odotuksia esimerkiksi laadun näkökulmasta, jolloin ylläpitoon kuluu rahaa.

Projektin onnistuminen voidaan ajatella [55, 56] nelivaiheisena prosessina, joka jakautuu neljään osaan: projektin onnistuminen prosessinäkökulmasta, lopputuotteiden, liiketoiminnan ja kontekstin näkökulmasta.

Lagsted [55] esittää väitöskirjassaan menetelmän, jonka avulla organisaatio voi valita järjestelmäkehitykseen soveltuvan menetelmän mahdollisimman hyvin. Lagstedt esittää ohjelmistokehitysmenetelmän valintaan kontingenssimallia, joka pyrkii huomioimaan projektiin liittyvät valintakriteerit sekä liiketoimintaympäristön muutokseen liittyvät tekijät. Kontingenssi tarkoittaa sitä, että organisaation toimintaympäristö on jatkuvassa muutoksessa eikä organisaatio pysty itse hallitsemaan muutoksia. Lagstedtin mallin perusteella projektit, joissa on paljon liiketoimintaan ja kehittämiseen liittyvää epävarmuutta sopivat erityisen hyvin muutospohjaisiin eli käytännössä ketteriin malleihin. Jos organisaation toimin-

ta on hyvin suunnitelmallista sekä prosessien että liiketoiminnan näkökulmasta, suunnitelmapohjaiset mallit soveltuvat esitetyn mallin perusteella paremmin käytettäviksi. Tässä työssä suunnitelmapohjaisina (eng. *plan driven*) malleina on esitetty prosessimallit ja muutospohjaisina (eng. *change driven*) malleina ketterät sekä lean-menetelmät.

5 HAASTATTELU KETTERIEN MENETELMIEN HYÖDYNTÄMISESTÄ

Tässä osiossa perehdytään työssä käytettyyn tutkimusmenetelmään sekä aineiston keruuseen.

5.1 Haastattelu tutkimusmenetelmänä

Ohjelmistokehityksen tutkiminen on haastavaa, koska tutkittaviin järjestelmiin liittyy usein niin teknologiaan perustuvat ongelmat kuin myös organisaatioon tai ihmisiin liittyvät ongelmat. Laadullisen tutkimuksen avulla voidaan pyrkiä havainnoimaan ihmisiä tai kysymään asioista niistä tietäviltä ihmisiltä [57].

Tässä työssä tutkimusmenetelmänä on haastattelu sekä tarpeellisen tiedon keruu tieteellisestä kirjallisuudesta sekä muista lähteistä. Tätä tutkimusta voidaan pitää monilta osin laadullisena tutkimuksena. Laadullinen tutkimus on luonteeltaan konstruktiivista ja sen kautta saadut tulokset ovat luonteeltaan subjektiivisia, koska tulokset perustuvat haastateltavien kokemuksiin sekä haastattelijan tulkintaan haastattelussa keskustelluista asioista. [58]

Haastatteluun päädyttiin tutkimusmenetelmänä siksi, että haluttiin syventää kirjallisuudesta löytyviä malleja sekä teorioita siitä, miten ketteriä menetelmiä hyödynnetään koneoppimisprojekteissa sekä ohjelmistokehityksessä. Haastattelun etuina voidaan pitää sitä, että haastateltavat voivat vapaasti kertoa omia näkemyksiään sekä syventävää halutessaan vastauksia [59].

Verrattuna kyselytutkimukseen, haastatteluun liittyy niin haastattelijasta kuin haastateltavista johtuvia virheitä [59]. Tämän tutkimuksen osalta virheen mahdollisuutta lisää se, että haastateltavien henkilöiden joukko on verrattain pieni. Suomen kokoisessa maassa koneoppimisen parissa työskentelee rajallinen määrä ihmisiä, joista haastatteluun on mahdollista saada vain tietyt henkilöt. Kohderyhmän satunnaisuus riippuu siis osittain tutkimuksen tekijästä. Toinen virheen mahdollisuus syntyy siitä, että haastateltavat saattavat pyrkiä antamaan "sosiaalisesti suotavia vastauksia"[59].

Aineiston keruu haastattelulla voidaan jakaa joko strukturoituun tai puolistrukturoituun haastatteluun. Strukturoidussa haastattelussa kysymykset ovat valmiina ja haastattelun eteneminen on ennalta määriteltyä. Tämän tutkimuksen puitteissa käytetään puolistruk-

turoitua haastattelua, koska tavoitteena on haastatella mahdollisimman erilaisia asiantuntijoita, jolloin kaikille ei voida esittää samoja kysymyksiä ja kysymyksiä voidaan esittää eri järjestyksessä. Tämän vuoksi tässä työssä käytetään puolistrukturoitua haastattelututkimusta, jossa alan asiantuntijoita haastatellaan osittain strukturoitujen kysymyksien avulla tutkielman aiheeseen liittyen. Pyrkimyksenä on, että haastateltavien kanssa päästään mahdollisimman syvälliseen keskusteluun tutkielman aihealueista ja heidän kokemuksensa perusteella voidaan täydentää kirjallisuudesta löytyvää tietoa [59].

Haastattelu valittiin tämän tutkimuksen pääasialliseksi menetelmäksi siksi, että työn kannalta on oleellista koota yhteen aihetta tuntevien tietämystä. Toinen vaihtoehtoinen tapa tiedon keräämiseen olisi voinut olla kyselytutkimus, mutta riittävän kattavan ja luotettavan otoksen saaminen kasaan olisi voinut olla haasteellista. Kyselytutkimuksen kautta saatu kohderyhmä olisi pitänyt tausta-aineiston osalta validoida tarkemmin kuin haastattelussa, jossa jokaisen osallistujan asiantuntemus käytiin läpi haastattelun alussa.

5.1.1 Haastateltavien valinta ja haastattelujen toteutus

Haastateltavien kohderyhmänä olivat henkilöt, jotka kehittävät työkseen koneoppimista hyödyntäviä sovelluksia. Näitä henkilöitä voivat olla esimerkiksi datatieteilijät, koneoppimis- ja datainsinöörit sekä ohjelmistokehittäjät. Haastateltavien valinta oli suunnitelmallista. Aluksi valittiin sellaiset haastateltavat, jotka tiedetään entuudestaan sopiviksi tutkijan taustan vuoksi. Jokaiselle haastatteluun osallistuvalle luvattiin, että heiltä ei menisi haastatteluun yli tuntia ja annettiin mahdollisuus saada teemat tietoon etukäteen. Haastateltaville myös luvattiin, että heidän työhönsä tai asiakkaisiin liittyviä tietoja ei yksilöidä tässä työssä.

Haastattelut toteutettiin videohaastatteluina käyttäen Zoom-ohjelmistoa, jonka avulla puheluiden tallentaminen oli mahdollista. Jokainen haastattelu nauhoitettiin alkukeskusteluja ja lopputervehdyksiä lukuunottamatta. Lisäksi kahdessa haastattelussa nauhoitus laitettiin tauolle kesken haastattelun, koska haastateltavan puolelle tuli poikkeustilanne jota ei haluttu nauhalle. Nämä poikkeustilanteet johtuivat etätyön asettamista haasteista.

Jokaiselta haastateltavalta pyydettiin lupa haastattelun nauhoittamiseen sekä kerrottiin tutkimuksen tavoitteet sekä lyhyt johdatus. Tavoitteena oli, ettei haastattelussa kysytyt kysymykset johdatelleet osallistujia tiettyihin vastauksiin. Haastateltaville korostettiin haastattelun alussa tilaisuuden luotettavuutta ja heitä pyydettiin olemaan kertomatta liikesalaisuuksien alaisia asioita. Ongelmana haastattelussa saattoi olla se, että yksilöitymisen pelossa saattoi jäädä jotain kertomatta.

Kaikilla haastateltavilla on taustaa konsultoinnista eli ovat tehneet koneoppimisratkaisuja asiakkailleen. Tämä toi haastatteluun näkökulman, jossa monet havainnot riippuvat paljon siitä minkälaisen asiakkaan kanssa ollaan työskentelemässä ja minkälainen asiakkaan oma osaaminen on. Toinen havainto haastateltavista oli se, että heidän koulutustasonsa oli hyvin korkea ja mukana oli monta tohtorin tutkinnon suorittanutta. Haastateltavien joukko ei ollut muutenkaan kovin monimuotoinen vaan haastattelijat olivat taustoil-

Haastateltava	Koulutustaso	Koulutusala	Rooli	Yrityksen rooli
A	tohtorin tutkinto	fysiikka	datatieteilijä	konsultointi
B	ylempi korkeakoulututkinto	tietojohtaminen	datatieteilijä	konsultointi
C	tohtorin tutkinto	tietojenkäsittelytiede	datatieteilijä	konsultointi
D	tohtorin tutkinto	tietojenkäsittelytiede	datatieteilijä	tuotetalo & konsultti

Taulukko 5.1. *Haastateltavien taustat*

taan hyvin saman kaltaisia.

5.2 Haastattelun runko ja kysymykset

Jokainen haastattelu noudatti samaa runkoa. Aluksi haastateltavia pyydettiin kertomaan heidän taustastaan. Taustojen selvittämisen avulla pyrittiin validoimaan heidän soveltuvuuttaan haastateltaviksi ja tarvittaessa tässä vaiheessa olisi voitu vielä keskeyttää haastattelu, jos se olisi ollut tarpeen. Taustojen kannalta oleellisia kysymyksiä oli koulutustausta, työkokemus vuosina sekä se minkälaisissa projekteissa tai tehtävissä työskentelee.

Jokaiselle haastateltavalle esitettiin seuraavat kysymykset:

- Mitä vaiheita tyypilliseen projektiisi kuuluu?
- Minkälaisia rooleja projekteissa on?
- Ovatko ketterät menetelmät tuttuja?
- Minkälaisia haasteita koneoppimisprojekteihin liittyy?
- Ovat koneoppimisen prosessimallit (esim. CRISP-DM) tuttuja?

Näiden kysymyksien lisäksi kysymyksiä tarkennettiin niiltä osin kuin oli tarpeen. Ketterien menetelmien periaatteet, kuten kommunikaatio asiakkaan kanssa sekä työn näkyväksi tekeminen korostuivat tarkentavissa kysymyksissä.

Haastattelujen vastaukset kirjoitettiin pääosin ylös haastattelun aikana, mutta vastauksia tarkistettiin nauhoituksista kun aikaa nauhoituksen ja analyysin välillä alkoi olemaan paljon.

6 TUTKIMUKSEN TULOKSET JA ANALYYSI

Tässä luvussa käsitellään haastattelun tuloksia. Aluksi nostetaan esiin haastattelussa ilmi tulleita huomioita ketterien menetelmien käytöstä, jonka jälkeen pyritään löytämään vastauksia tutkimuskysymyksiin.

6.1 Koneoppimistekemisen iteratiivinen luonne

Eräs haastatteluissa usein esiin noussut asia on koneoppimisprojektien iteratiivinen luonne sekä kokeilut. Haastateltavat korostivat sitä, että koneoppimisprojekteissa on usein tilanne jossa ei voida projektin alussa tietää mitä voidaan tehdä saatavissa olevalla datalla. Tämä luo projektiin epävarmuutta, jota haastatellut pyrkivät osaltaan parantamaan sillä että lähteävät tietoisesti ratkaisemaan ongelmia mahdollisimman iteratiivisesti. Tämä sopii hyvin ketteriin periaatteisiin, mutta edellyttää hyvää kommunikaatiota asiakkaan tai muun sidosryhmän sekä kehittäjien välillä.

Iteratiivinen luonne korostuu myös mahdollisten kehitettyjen algoritmien parantamisessa, sillä tehtyjä algoritmeja voidaan parantaa jatkuvasti paremmiksi joko erilaisten mittareiden avulla tai jotenkin muuten määriteltynä.

Ohjelmistokehityksestä poiketen koneoppimisprojekteissa työn tekemisen sykli voi olla joko merkittävästi pidempi tai lyhyempi kuin ketterissä menetelmeissä määritellyt ajanjaksot. Esimerkiksi voi olla, että jonkin asian tutkiminen ja koneoppimismallin kehittäminen voi viedä enemmän aikaa kuin kaksi tai kolme viikkoa (Scrumin sprintin kesto). Toisaalta myös syklit voivat olla hyvin lyhyitä: asiat voivat edetä nopeasti eikä välttämättä ole tarpeen odottaa, että mennään seuraavalle sprintille tekemisen kanssa.

Iteratiivinen luonne sekä syklien pituuden vaihtelu vastaa haastattelujen perusteella kappaleessa 4.3 esiin tulleita Isomursun tutkimuksen [42] havaintoja siitä, miten UX-tekemisen aikataulutus ohjelmistokehityksen projekteissa voi olla haastavaa. Haastatteluissa nousi esiin se, että koneoppimisalgoritmien kehittäjä saattaa työskennellä yksittäisen tehtävän kanssa pitkään ja esimerkiksi kommunikoidessa muun tiimin kanssa, voi näyttää siltä että työ ei ole edennyt jos etenemistä pyritään ensisijaisesti tarkastelemaan esimerkiksi Kanban-taulun tai muun tehtävienhallintaratkaisun avulla.

6.2 Kommunikaatio asiakkaan kanssa

Kommunikaatio asiakkaiden kanssa nousi monissa keskusteluissa esiin. Koneoppimisprojektit saattavat olla monimutkaisia tai asiakkailla voi olla koneoppimisesta epärealistisia odotuksia, esimerkiksi siksi että siitä puhutaan hyvin paljon mediassa ja aiheen ympärillä on paljon hypeä. Tämän vuoksi moni haastateltavista näki tärkeäksi sen, että koneoppimisprojekteja tehdässä kommunikoidaan asiakkaan kanssa ja tarvittaessa autetaan ymmärtämään sitä, mistä asiassa on kyse. Tyypillinen haaste koneoppimisprojektien kanssa on se, että asiakas saattaa luulla että koneoppimisprojekteissa käytettävät tai kehitettävät algoritmit pystyvät käsittelemään ja tuottamaan mistä tahansa datasta hyviä tuloksia. Tämän valossa voisi ajatella, että koneoppimisprojekteissa ei ole aivan selkeää mikä on koneoppimisalgoritmin toimijuus: sen sijaan, että osattaisiin suhtautua koneoppimishankkeeseen teknisesti, mitattavana ja satunnaisuutta sisältävänä projektina, kuvitellaan että koneoppimisprojektin myötä luodaan järjestelmä, jolla on jokin inhimilliseen toimintaan verrattavissa oleva toimijuus.

Kommunikaatiossa asiakkaan kanssa korostuu haastattelujen perusteella myös yhteinen kieli. Yhteisellä kielellä ei tarkoiteta englantia tai suomea, vaan sitä että organisaation sekä konsultin tai muuten koneoppimisen parissa työskentelevän välillä käytetään asioista yhteneväisiä termejä sekä käsitteitä. Yhteisen kielen puute näkyy sekä kommunikaatiossa kuin siinä, että käytössä olevaan dataan perehtyminen vaatii paljon aikaa. Käytössä oleva data saattaa olla tietomalliltaan sekä termeiltään sellaisia, että niiden ymmärtämiseen menee aikaa.

Koneoppimisprojekteissa työskennellään datan kanssa, joten työn näkyväksi tekemisen välineenä voi hyödyntää myös datan visualisointiin liittyviä menetelmiä. Tämä voi mahdollisesti parantaa kommunikointia asiakkaan kanssa ja auttaa ymmärtämään paremmin sitä, mitä ollaan tekemässä. Datavisualisoinnin merkitys korostuu myös siinä kun perehdytään dataan liiketoiminnan ymmärtämisen näkökulmasta: poikkeamat saadaan helpommin esille visuaalisilla välineillä ja siten voidaan parantaa yhteistä ymmärrystä.

6.3 Projektin työryhmien kokoonpanot

Haastatteluissa haastateltavilta kysyttiin minkälaisilla työryhmillä he lähtevät ratkaisemaan koneoppimiseen liittyviä ongelmia. Vastauksissa korostui usein data- ja koneoppimislähestyminen eli tiimin kokoonpano kuvattiin pitkälti siitä näkökulmasta, että ollaan luomassa koneoppimissovelluksia. Tämä tarkoittaa käytännössä sitä, että usein projektissa on datatietelijä joko kehittämässä mallia tai vastaamassa myös siitä, että data saadaan oikeassa muodossa projektin hyödynnettäväksi. Datan hyödyntämisen näkökulmasta korostui myös niin kutsutun datainsinöörin (eng. *data engineer*) merkitys, jonka tehtäviin kuuluu tietokantojen sekä datan hakemiseen liittyvien prosessien kehitys. Datatietelijän ja datainsinöörin roolit kuvattiin osin päällekkäisinä ja osin syklisinä. Päällekkäisyys tarkoittaa sitä, että datatieteilijän tehtäviin saattaa kuulua myös datainsinöörin tehtävät ja

Rooli	Tehtävä koneoppimisprojektissa
Palvelumuotoilija / konsultti	Roolin tarkoituksena auttaa ymmärtämään ratkaistavaa ongelmaa. Voi olla myös datatieteilijä
Asiakas	Mukana projektissa vastaamassa siitä, että asiakkaan näkökulma tulee huomioitua (vrt. Scrumin Product owner)
Datatieteilijä	Tehtävänä itse mallin kehittäminen. Voi olla myös päällekkäinen rooli datainsinöörin kanssa
Datainsinööri	Vastaa siitä, että projektissa saadaan hyödynnettyä dataa mahdollisimman hyvin
Ohjelmistokehittäjä	Kehitetyn mallin hyödyntäminen lopullisessa ohjelmistotuotteessa

Taulukko 6.1. *Roolit koneoppimisprojekteissa*

syklisyys sitä, että näitä rooleja ei välttämättä tarvita samanaikaisesti.

Haastatteluissa kävi ilmi, että haastateltavien koneoppimisprojektit toteutetaan usein osana jotakin ohjelmistoprojektia eli koneoppimisen merkityksenä on tukea jotakin ohjelmistoa. Käytännössä tämä tarkoittaa sitä, että kehitettävä koneoppimisratkaisu nähdään sellaisena ratkaisuna, että jonkin loppukäyttäjän on pystyttävä sitä hyödyntämään omassa toiminnassaan haluammallaan tavalla. Esimerkki, joka nousi esiin haastatteluissa melko monta kertaa oli tuotesuosittelujärjestelmä jonka tarkoituksena on pyrkiä ennustamaan vaikkapa verkkokaupan käyttäjille mistä tuotteista he voisivat olla kiinnostuneita aiempien ostoksien ja muiden käyttäjien tekemisen perusteella. Tuotesuosittelujärjestelmän kaupallinen hyödyntäminen edellyttää sitä, että koneoppimista varten tehdyn ratkaisun on toimittava osana verkkokaupan ohjelmistoratkaisua. Koneoppimisprojektin näkökulmasta tämä tarkoittaa sitä, että tiimissä on usein mukana ohjelmistokehittäjiä vastaamassa esimerkiksi käyttöliittymän tai rajapinnan toteuttamista varten. Ohjelmistokehitystiimin osana toimiminen

Kaikissa haastatteluissa nousi esiin asiakkaan ongelmien ymmärtäminen. Ongelmien ymmärtäminen nähtiin sekä datatieteilijän vastuuna että erillisen konsultin, kuten vaikkapa palvelumuotoilijan vastuuna olevana ongelmana. Asiakkaan ymmärtämisen haasteita ovat kappaleessa 6.2 esitetyt asiat sekä se, että on varmistettava oikean ongelman ratkaiseminen.

Verrattuna luvussa 4.3 kuvattuihin Isomursun [42] esittämiin ratkaisuihin ja haasteisiin siitä, miten UX-työskentely ja ohjelmistokehitys voivat toimia yhdessä Scrum-projektissa, näiden haastattelujen perusteella voisi arvioida että koneoppimisasiantuntijoiden työskentely yhdessä ohjelmistokehittäjien kanssa voidaan nähdä hyvinkin luontevana. Koneoppimisen työtavat eli miten työskennellään, ovat lähempänä ohjelmistokehitystä kuin UX-suunnittelijan työ, johtuen siitä että koneoppimisessa käytetään tiedon analysointiin sekä mallien kehittämiseen ohjelmointikieliä, joko siihen erityisesti suunniteltuja (R) tai yleiskäyttöisiä kieliä siten, että hyödynnetään koneoppimiskirjastoja (esimerkiksi Python tai Java).

6.4 TK1: Koneoppimisen prosessimallien hyödyntäminen

Haastattelututkimuksen perusteella selvisi, että koneoppimisen prosessimallit olivat monille koneoppimisprojekteissa työskenteleville jollakin tasolla tuttuja, ainakin nimeltä, mutta niiden vaiheisiin tai yksittäisten prosessien yksityiskohtiin ei välitetty. Prosessimalleja oltiin myös käytetty esimerkiksi asikkaille lähetetyissä tarjouksissa luomassa ymmärrystä siitä, mitä ollaan tekemässä. Näiden mallien käytöstä ei oltu myöskään innokkaita.

Haastattelujen perusteella löydöksenä oli se, että koneoppimisprojekteissa hyödynnettiin tietynlaista prosessimaista lähestymistapaa: koneoppimissovelluksen tekeminen pilkottiin pienempiin osaongelmiin kuten ongelman ymmärtämiseen, datan mallintamiseen, mallin testaamiseen sekä muihin prosessimalleissa kuvattuihin seikkoihin. Taulukossa 6.2 on esitetty haastateltavien kuvaukset siitä minkälaisia prosesseja he noudattavat suurin piirtein projekteissa. Kaikissa haastatteluissa kävi ilmi kehittämisen iteratiivinen luonne: malleja saatetaan lähteä kehittämään hyvinkin pienellä tai osin jopa puutteellisella datalla, jolloin on palattava esimerkiksi liiketoimintaymmärryksen tai datan keruun puoleen.

Liiketoimintaymmärryksen rooli korostui useissa haastatteluissa. Konsultteina työskentelevät haastateltavat korostivat keskustelua asiakkaan kanssa sekä joko datatieteilijän tai esimerkiksi palvelumuotoilijan roolia siinä, että ymmärretään tarkemmin mitä pyritään tekemään. Datatieteilijän rooli ymmärryksen luomisessa on enemmän substanssipuolen eli koneoppimisen ymmärtämisen sekä mahdollisuuksien kartoittamista sekä opettamista kun taas palvelumuotoilijan roolina nähtiin se, että pyritään ymmärtämään tehdäänkö oikeita asioita eli ratkoitaanko sellaista ongelmaa jota on määrä ratkaista.

Dataan liittyvät ongelmat korostuivat myös. Yrityksien käytössä oleva data voi olla epäselvässä muodossa tai siinä voi olla ongelmia, joiden vuoksi mallintaminen voi olla haastavaa tai puutteellista. Tällöin on syytä palata tutkimaan onko jotain muuta dataa saatavilla tai voidaanko dataa esimerkiksi kerätä jostain. Mallinnuksen näkökulmasta datan on myös oltava olla sellaista, että sitä voidaan käyttää myös tuotannossa olevassa järjestelmässä. Koneoppimisen termien avulla ilmaistuna tämä tarkoittaa sitä, että datatieteilijän on huolehdittava että sekä tuotannon että kehitysvaiheen data on molemmat samasta jakaumasta eli mallin parametrin oletukset ovat voimassa tässä tilanteessa.

CRISP-DM-mallin mallinnus sekä mallin arviointi sekä Amershin malleissa esitetty piirteiden mallintaminen, mallin opettaminen sekä arvioiminen voidaan haastattelujen perusteella nähdä hyvin iteratiivisena prosessina. Käytössä olevan datan sekä mallin mittaristojen avulla arvioidaan miten hyvin malli toimii ja kehitys jatkuu joko mallia parantamalla tai viemällä sitä eteenpäin. Haastateltavat korostivat sitä, että mallin toimintaa on usein seurattava myös tuotannossa. Lisäksi on usein tarpeen kehittää mekanismeja siihen, miten loppukäyttäjän näkökulmasta koneoppiminen toimii.

Haastatteluissa kävi ilmi myös se, että koneoppimisen sovelluksia kehitetään usein osana isompaa kokonaisuutta: koneoppiminen tukee jonkin muun järjestelmän, esimerkiksi verkkokaupan toimintaa, jolloin projektissa korostuu laajemmin myös ohjelmistokehityk-

	Haastateltava			
Vaihe	**A**	**B**	**C**	**D**
1	Ongelman ymmärtäminen	Liiketoiminnan ymmärtäminen	Datan hakeminen Tarpeiden määrittely	Ongelman ymmärtäminen
2	Tavoitteiden määrittely	Iteratiivinen kehittäminen	Yksinkertaisen mallin testaaminen	Datan hankkiminen
3	Datan tutkiminen	Tuotantoonvienti	Uuden datan hakeminen	Iteratiivinen kehittäminen
4	Mallinnus ja visualisointi	Monitorointi	Mallinnusta iteratiivisesti	Mallin hyväksyntä
5	Sovelluksen vieminen tuotantoon			Kokeileminen tuotannossa
6	Monitorointi			Tuotantoonvienti

Taulukko 6.2. *Haastateltavien esittämät prosessit*

sen näkökulma.

6.5 TK2: Ketterien menetelmien hyödyntäminen

Kaikissa haastatteluissa kävi ilmi, että koneoppimisprojektit tehdään pääosin ketterillä menetelmillä tai niitä soveltaen. Haastatteluissa ei korostunut mitkään yksittäiset menetelmät, koska ne riippuvat usein siitä mihin tiimi tai asiakas on tottunut. Suurin osa haastatelluista ei ollut perehtynyt kovin syvällisesti ketterien menetelmien erityispiirteisiin vaan ymmärsivät menetelmistä perusteet ja pystyivät soveltamaan niitä omaan tekemiseensä. Ketterien menetelmien soveltamisessa nähtiin tärkeäksi esimerkiksi Scrum Master -rooli tiimin toimintaa kehittävänä roolina.

Kappaleessa 6.1 mainittu iteratiivinen luonne sopii hyvin ketteriin menetelmiin, koska ketterät menetelmät perustuvat iteratiiviseen kehittämiseen. Tämä nähtiin haastatteluissa myös tärkeänä syynä sille, että koneoppimisprojekteissa on pakko käyttää ketteriä menetelmiä koska työn tai projektin tavoite voi muuttua nopeastikin, jolloin projekteja ei voida hallita etukäteen suunnitellusti kuten prosessimalleihin perustuvissa menetelmissä. Koneoppimisprojekteihin voi liittyä myös riskejä liittyen siihen onko jotain ongelmaa mahdollista tehdä lainkaan koneoppimismentelmiä hyödyntäen, jolloin kokeilut ja inkrementaalinen kehittäminen mahdollistavat kehityksen.

Ketterien menetelmien näkökulmasta korostui myös se, että koneoppimisalgoritmien kehittäminen nähtiin helppona osana projektia ja projektin vaikeat osuudet saattavat olla esimerkiksi kommunikoinnissa asiakkaan kanssa tai sopivan datan saamisessa kasaan. Ketterän projektinhallinnan näkökulmasta haasteita koneoppimismallien kehittämiseen toi myös se, että projekteissa saattoi tulla odottelua etenkin datan saamisen kanssa, jos se on esimerkiksi tietokannassa jota hallinnoi jokin ulkopuolinen.

Haastatteluissa korostui myös työtapojen merkitys sekä etenkin vuoden 2020-2021 aikana tapahtunut muutos siitä, että tiimit eivät työskentele fyysisesti yhdessä vaan etänä. Tämä voi aiheuttaa haasteita sille, että ketteryyden perusperiaatteet kuten kommunikaatio ja työn näkyväksi tekeminen on haastavia toteuttaa.

Luvussa 4.4.2 esitettyjen MlOps-menetelmien käyttöön suhtauduttiin haastateltavien keskuudessa varauksella, mutta mielenkiinnolla ja niiden merkitys nähtiin etenkin siinä kun luodaan hyvin monimutkaisia koneoppimissovelluksia. Näitä työkaluja ei kuitenkaan tun-

nuttu käytettävän. Saattaa olla, että haastattelussa korostuneiden ohjelmistokonsulttitalojen työntekijöiden näkökulmasta on helppo ottaa projekteihin mukaan ohjelmistokehittäjiä luomaan esimerkiksi automaattiset tuotantoonviennit sekä rajapinnat, jolloin niiden tekeminen MlOps-työkaluilla ei ole nähty tarpeelliseksi. Uusien työkalujen ja kokonaisuuksien käyttöönotto asiakkaalla tai organisaatiossa voi myös olla kallista, jolloin mahdollisuus tehdä omaa erikoisalaa - eli tässä tapauksessa koneoppimista, voi heikentyä jos resurssit menevät uuden järjestelmän tai periaatteen käyttöönottoon.

Näiden lisäksi haastatteluissa keskusteltiin hieman AutoML-tekniikoista eli automaattisesta koneoppimisesta, jonka ajatuksena olisi automatisoida koneoppimiseen liittyvät tehtävät siten, että koneoppimisen tehtävät voisi jättää datatieteilijöiden ja koneoppimisen asiantuntijoiden sijaan jonkin alan substanssiosaajille. AutoML-tekniikassa ajatuksena on se, että automaation avulla kone etsii malliavaruudesta ongelmaan liittyvään dataan soveltuvia malleja sekä pyrkii näitä optimoimalla löytämään sopivan ratkaisun. [60] AutoML-teknologian käyttöönotto ja yleistyminen saattaisi vähentää koneoppimisprojektien määrää sekä vähentää niihin tarvittavaa työmäärää, koska erityistä asiantuntemusta ei tarvittaisi koneoppimisongelmien ratkaisemiseen. Ketterien ohjelmistokehitysmenetelmien näkökulmasta tarkasteltuna menetelmän hyötynä voisi nähdä sen, että itse teknologia ja ongelma saadaan saman henkilön tehtäväksi jolloin esimerkiksi kommunikaatiosta johtuvat ongelmat projekteissa voisivat vähentyä.

7 EHDOTUS KETTERIEN MENETELMIEN KÄYTÖSTÄ KONEOPPIMISPROJEKTEISSA

Tässä luvussa käsitellään työn teoriaosuuden sekä haastattelujen perusteella lähestymistapa siihen, miten organisaatioissa tai ohjelmistotiimeissä voidaan viedä koneoppimiseen liittyviä projekteja systemaattisemmin eteenpäin sekä hyödyntää sekä käytössä olevia koneoppimisen prosessimalleja, ketteriä menetelmiä sekä DevOps-, DataOps- sekä MlOps-menetelmiä.

7.1 Organisaation rakenne ja toiminta

Tässä osassa käydään läpi organisaation johtamiseen liittyvät näkökulmat, jotka korostuivat koneoppimisprojektien näkökulmasta. Haastattelujen pohjalta keskeisimmät huomiot liittyvät oppimiseen, prosessien läpinäkyvyyteen, työn läpinäkyvyyteen ja kommunikaatioon sekä iteratiivisuuteen.

7.1.1 Oppiminen

Haastattelujen perusteella koneoppimisprojektit voivat olla organisaatioille haastavia, koska koneoppimisen menetelmiä tai käyttökohteita ei tunneta tarpeeksi hyvin. Odotukset koneoppimisen mahdollisuuksista voivat olla myös korkeat eikä välttämättä tunneta tarpeeksi hyvin millä keinoilla jokin ongelma voidaan ratkaista.

Koneoppimisprojektien näkökulmasta organisaation oppiminen on hyvin tärkeässä roolissa. Projekteissa olisi hyvä varata aikaa siihen, että asiakkaan kanssa mietitään ratkaistavaa ongelmaa ja mitä sen ratkaiseminen vaatii organisaatiolta. Koneoppimisprojekti ei ole millekään organisaatiolle pakollinen projekti, joten ratkaisuna voi olla myös se ettei koneoppimisprojektia tehdä lainkaan vaan liiketoiminnan ongelma ratkaistaan jollakin muulla tavalla, esimerkiksi kehittämällä jokin perinteinen sovellus.

Ketteriin menetelmiin kuuluu läheinen työskentely asiakkaan kanssa sekä nopeat palautesyklit asiakkaan sekä kehittäjätiimin välillä. Koneoppimisprojektissa oppimista voidaan edistää esimerkiksi datan visualisoinnin avulla, jolloin dataan liittyvät ongelmat tai aikaansaadut mallit saadaan havainnollistettua paremmin asiakkaalle. Esimerkiksi kuvantunnistusalgoritmeja visualisoidaan niin, että tunnistetun objektin ympärille piirretään suorakulmio jonka vieressä on todennäköisyys sille, että kyseessä on tunnistettu objekti.

7.1.2 Prosessien läpinäkyvyys

Prosessien läpinäkyvyyden tarkoituksena on varmistaa se, että kaikki tarvittava tieto on hyvin saatavilla tai ainakin on tiedossa mistä mitäkin voi saada ja millä aikataululla. Prosessien läpinäkyvyyden näkökulmasta oleellista on se, että eri organisaatioiden osat sekä erilaiset toimijat (kuten vaikka konsultit) pystyvät toimimaan hyvin yhdessä.

Prosessien läpinäkyvyyden työkaluna projektissa voisi käyttää arvovirtojen mallinnusta siten, että eri prosessien vaiheiden pullonkaulat ja niistä koituvat mahdolliset viivästykset saadaan hallintaan mahdollisimman varhaisessa vaiheessa, jotta projektin eteneminen on mahdollisimman sujuvaa.

7.1.3 Työn läpinäkyvyys ja kommunikaatio

Työn läpinäkyvyys ja näkyväksi tekeminen kuuluu ketterien menetelmien perusperiaatteisiin. Kanbanissa työ tehdään näkyväksi taulun avulla, jossa nähdään tehdyt tehtävät ja mitä on meneillään, kun taas Scrumissa näkyvyyttä edistää päivittäiset rituaalit.

Koneoppimisprojekteissa työn näkyvyyden kannalta on oleellista ymmärtää se, että yksittäinen tehtävä saattaa olla pitkään työn alla eikä etene suoraviivaisesti eteenpäin. Monissa tapauksissa, etenkin numeerisen datan kanssa työskennellessä datatieteilijöiden ja projektissa mukana olevien voisi olla hyvä visualisoida koneoppimismallin kehitystä vaikkapa graafien ja datavisualisaatioiden avulla.

7.1.4 Iteratiivisuus

Koneoppimismallien kehittäminen on monilta osin iteratiivista työtä: malleja kehitetään ja parannetaan uuden datan sekä aiempien tuloksien osalta. Koneoppimissovellusten kehittäminen organisaatiossa kannattaa suunnitella alusta lähtien siitä näkökulmasta, että kehitys on iteratiivista. Organisaation on myös huomioitava se, että tuloksia ei välttämättä tule nopeasti tai ei lainkaan, jos käytettävissä oleva data ei tarjoa siihen mahdollisuuksia. Osana koneoppimisprojektia kannattaa panostaa siihen, että organisaatio huomioi tarvittavat kehityskohteet esimerkiksi datan keräämisen suhteen ja laatii tarvittavat suunnitelmat datan saamiseksi.

Iteratiivisuuden tueksi on määriteltävä kriteerit, joilla mallin hyvyys määritellään. Käytännön sovelluksissa voi olla tarpeen määritellä kriteerit niin, että ne vastaa liiketoimintatarpeita eikä koneoppimisen metriikoita.

7.1.5 Koneoppimisprojektin kokoonpano

Koneoppimisprojektin kokoonpanoon otettiin kantaa vain Team Data Science Process -mallissa, mutta haastattelujen perusteella kokoaisin koneoppimisprojekteja varten seuraavanlaisen tiimin:

- **Datatieteilijä**, jonka tehtävänä on vastata mallin kehittämisestä sekä siihen liittyvästä data-analyysistä
- **Datainsinööri**, jonka tehtäviin kuuluu vastata siitä että dataa on saatavilla sekä datatieteilijöillä että muulla organisaatiolla ennustuksien jälkeen. Datainsinöörin tehtäviin voi kuulua myös ohjelmistokehitys tai mallien ohjelmoiminen tuotantoon.
- **Ohjelmistokehittäjä**, jonka vastuulla on vastata koneoppimisprojektissa tarvittavien ohjelmistojen kehittämisestä.
- **Konsultti tai palvelumuotoilija**, jonka tehtäviin kuuluu ongelman muotoileminen yhdessä asiakkaan kanssa ja liiketoimintaongelman ratkaiseminen.

Näiden lisäksi projektissa on syytä olla mukana asiakas vastaamassa liiketoiminnan edustuksesta ja riippuen käytettävästä ketterästä menetelmästä, joko Scrum Master tai joku muu projektinhallinnasta vastaava henkilö.

7.2 Taksonomian ymmärtäminen

Luvussa 2.6 esitetään Lwakatare et al. [9] esittämä taksonomia eri tyyppisille koneoppimisprojekteille ja haastatteluissa nousi esiin vastaavanlaisia piirteitä myös haastateltavien koneoppimisprojekteissa. Koneoppimisprojektit kannattaa luokitella sen mukaan, minkälaisesta projektista on kysymys ja lähestyä projektiorganisaatiota sen kautta.

Taksonomian ensimmäisen tason projektien eli kokeiluiden osalta projekti kannattaa aloittaa mahdolllisimman ketterästi siten, että ymmärretään mahdolliset riskit sekä ollaan valmiita oppimaan teknologioista sekä niiden soveltuvuudesta ratkaisemaan liiketoiminnan ongelmia. Ensimmäisen tason ongelmissa iteratiivisuus, oppiminen sekä prosessin läpinäkyvyys tulee ottaa huomioon mahdollisimman korostetusti. Haastatteluissa nousi esiin kommunikaatio asiakkaan kanssa, johon sisältyy yhteinen kieli sekä yhteinen ymmärrys liiketoiminnasta. Haastatteluiden perusteella ensimmäisen tason projekteja on hyvin paljon. Projektiryhmän kokoonpanon näkökulmasta voi olla niin, että ensimmäisen tason projekteissa ei tarvita suurta tiimiä, esimerkiksi datatieteilijä sekä datainsinööri voivat olla alkuun riittäviä. Kuvan 7.2 mukaisessa prosessissa taksonomian ensimmäisen tason projetit vaativat erityisesti huomioita yhteistyöstä asiakkaan kanssa, muutoksiin reagoinnista sekä mallin testaamisesta.

Lwakataren taksonomian toisella tasolla on ei-kriittinen tuotantoonvienti, jossa kehitettyjä malleja sovelletaan oikeassa ympäristössä. Toisen tason projekteissa on taustalla jo jonkinlainen tietämys siitä, miten mallit toimivat organisaation käytössä olevalla datalla sekä taksonomian ensimmäisen tason projektin vaiheen tietämys. Toisella tasolla tavoitteena voidaan nähdä se, että mallia hyödyntävä sovellus saadaan käyttöön sekä toimimaan. Projektien näkökulmasta koneoppimisprojektin kokoonpano laajenee tässä vaiheessa: mukaan tulee ohjelmistokehittäjiä ja tehtyjä tuloksia viedään muiden järjestelmien käyttöön. Tässä vaiheessa mukaan nousee myös integraatiotestaus sekä järjestelmä kokonaisuutena.

Taksonomian kolmannen tason projekteissa on kyse kriittisista järjestelmistä ja kehitettyjen sovellusten on toimittava luotettavasti aina datan tuomisesta siihen, että malli viedään tuotantoon ja ja sitä hyödynnetään. Kolmannen tason projekteissa on syytä hyödyntää työkaluja, joilla malleja voidaan järjestelmällisesti monitoroida ja tarvittaviin poikkeamiin voidaan puuttua. Kriittisten tuotantoonvientien yhteydessä ohjelmistojen skaalautuvuus korostuu ja ohjelmistoprojektissa on huomioita näihin liittyvät tekijät.

Taksonomian neljännen tason projektit ovat ketjutettuja projekteja, jolloin kehitetyn komponentin toiminta riippuu toisista komponenteista. Ketjutetut komponentit riippuvat toisistaan, jolloin voi olla epäselvää mitkä järjestelmät hyödyntävät jonkin koneoppimisjärjestelmän tuottamaa dataa.

Esitetyn taksonomian perusteella voi ajatella, että koneoppimisprojektien maturiteettitaso sekä teknisesti että organisaation osaamisen perusteella kasvaa mitä korkeammalle taksonomiatasoilla noustaan. Projektien läpivientien näkökulmasta on tärkeää, että organisaatio ymmärtää mitä ollaan tekemässä ja projektiin saadaan mukaan oikeanlainen tiimi.

7.3 Työkalut koneoppimisprojektissa

Haastattelututkimuksessa ja kirjallisuudessa ei juurikaan oteta kantaa siihen millaisilla työkaluilla koneoppimisprojekteja tehdään. Haastattelujen perusteella kävi kuitenkin ilmi, että koneoppimisprojektit tehdään osana ohjelmistoprojektia ja mallinnustyön lopputuote on usein jokin loppukäyttäjälle näkyvä sovellus. Tässä osassa käydään läpi ehdotuksia niistä työkaluista, joita voidaan tarvita koneoppimisprojektin integraatioon projektiin liittyvään ohjelmistoprojektiin. Työkalujen tarjoaminen ratkaisuksi ei välttämättä ole keskeistä, kuten on opittu jo *The Mythical Man-Month*-kirjan [61] *There is no silver bullet*-esseestä, mutta kirjallisuuden ja haastattelujen perusteella perustellut ehdotukset erilaisista työkalukategorioista voi auttaa hahmottamaan projektin kulkua sekä tarjoamaan keinoja sille miten työtä saa läpinäkyvämmäksi ja organisaation oppimista tukevaksi ketterien periaatteiden mukaisesti.

7.3.1 Versionhallinta

Ohjelmistoprojekteissa käytetään versionhallintaa siihen, että ohjelmistojen lähdekoodit ja niihin tehdyt muutokset ovat hallittuja työskennellessä isoissa tiimeissä. Versionhallinnan avulla ohjelmistokehittäjät näkevät mikä osa ohjelmakoodista on muuttunut ja kuka sitä on muuttanut.

Koneoppimisprojekteissa versionhallinta voidaan laajentaa koskemaan myös malleja sekä mallin parametreja. Käytännössä mallit ovat ohjelmakoodia, jolloin niiden vieminen versionhallintaan on luontevaa. Iteratiivisessa kehityksessä, jossa pyritään löytämään parhaiten toimiva malli versionhallinta tukee kehitystä.

Mallin hyperparametrien versioinnilla voidaan hallita saman mallin eri versioiden testaa-

mista, jolloin voidaan kontrolloidusti nähdä miten malli toimii eri parametreilla. Tätä varten kehitettyjä työkaluja on luvussa 4.4.2 esitetyt MlOps-menetelmät.

Ehdotuksena koneoppimisprosessiin on, että siinä hyödynnetään laajalti versionhallintaa koska se mahdollistaa muutoksienhallinnan sekä seurannan.

7.3.2 Rajapinnat ja integraatiot

Koneoppimismallin hyödyntäminen sovelluksessa on haastattelujen perusteella riippuvainen sovelluksesta, jonka osana se on. Sovellus voi olla esimerkiksi verkkokauppa tai mobiilisovellus, mutta toisaalta sovelluksena voidaan ajatella myös esimerkiksi yrityksen raportointiin liittyviä työkaluja kuten erilaisia BI-järjestelmiä tai tiedon visualisointityökaluja.

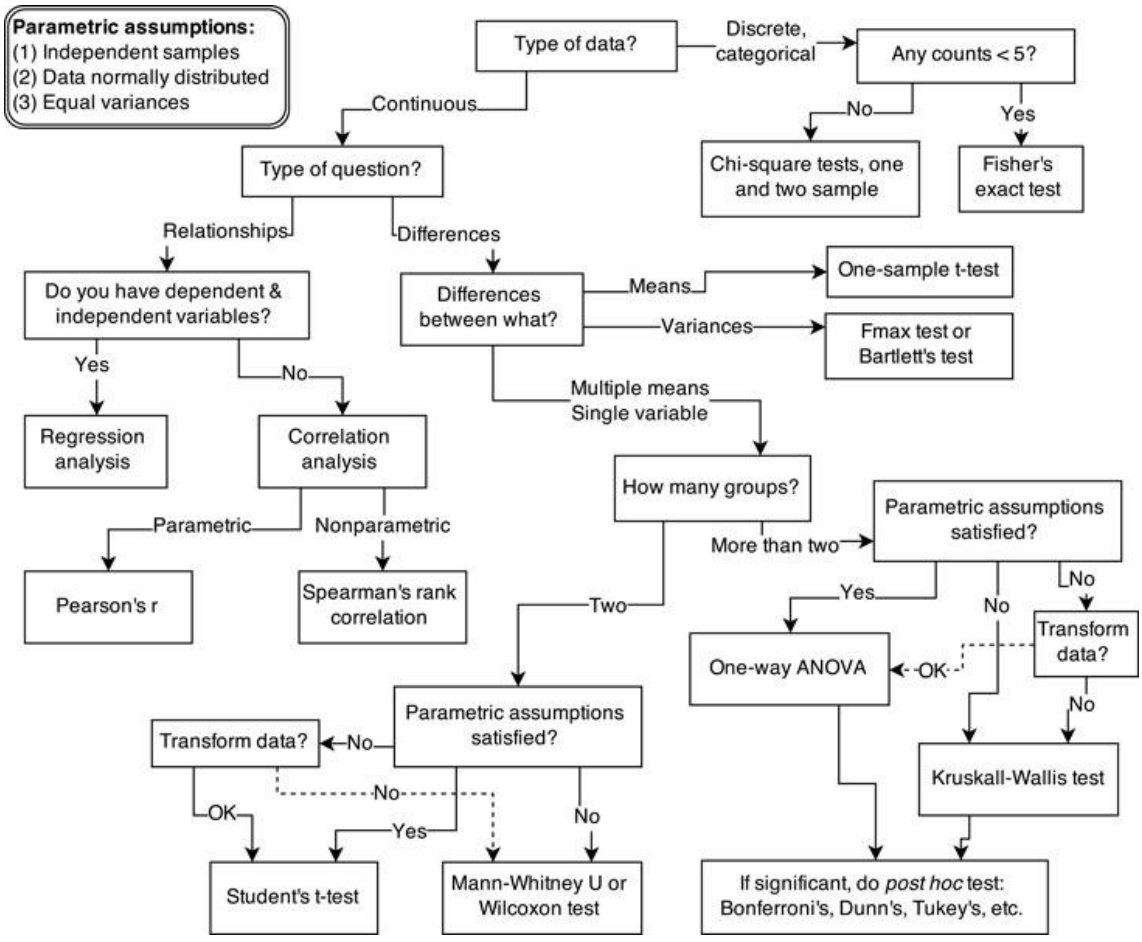
Ohjelmistokehityksen näkökulmasta koneoppimisprojektin lopputuote on yksi komponentti kehitettävään järjestelmään, joka tarjoaa muille järjestelmän osille tarvittavat ennusteet tai luokittelut.

Ohjelmiston suunnittelussa on otettava huomioon koneoppimiskomponentin hyödyntäminen niin, että sen käyttö vastaa tarpeita eikä aiheuta ylläpidollisesti lisätyötä tai ongelmia projektille. Järjestelmän arkkitehtuurin näkökulmasta tällainen suunnittelu tarkoittaa sitä, että koneoppimiseen käytettävä komponentti on arkkitehtuurisesti irrallaan muusta sovelluksesta.

7.3.3 Koneoppimisen ja data-analyysi

Miikka Koskinen esittää diplomityössään [62] hypoteesin siitä, miten syvien neuroverkkojen käyttö vaikuttaa koneoppimisprojektin ketteryyteen. Koneoppimisprojektit voidaan ajatella jakautuvan sellaisiin projekteihin, joissa käytetään valmista ja tunnettua mallia jonkin ongelman ratkaisemiseen. Esimerkiksi kuvantunnistukseen on olemassa hyvin tehokkaita ja testattuja algoritmeja, joiden avulla kuvista saadaan tunnistettua vaikkapa eläimiä mutta yrityksen liiketoiminnan kannalta keskeisissä analyyseissä valmis malli voi toimia ainoastaan pohjana sille miten ongelmaa lähdetään ratkaisemaan. Valmiin mallin sijaan keskiöön nousee silloin mallinnusosaaminen, jossa sovelletaan oikeita menetelmiä ongelman ratkaisemisemiseen.

Projektien läpiviennin näkökulmasta on keskeistä joudutaanko samaa ongelmaa ratkaisemaan useamman kerran vai voiko jotakin kehitettyä komponenttia käyttää uudelleen jonkin toisen ongelman ratkaisussa. Koneoppiminen perustuu menetelmällisesti pitkälti data-analyysin menetelmiin, joka taas puolestaan perustuu pitkälti tilastollisiin menetelmiin. Tilastollisessa päättelyssä on voinut datan muodon eli jakauman sekä muuttujien oletuksien, perusteella valita melko hyvin sen mitä mallia käytetään analyysien tekemisessä. Kuvassa 7.1 on esitetty *Statistical rethinking*-kirjasta [63] kuva, jonka avulla voi valita datan perusteella sopivan tilastollisen menetelmän.



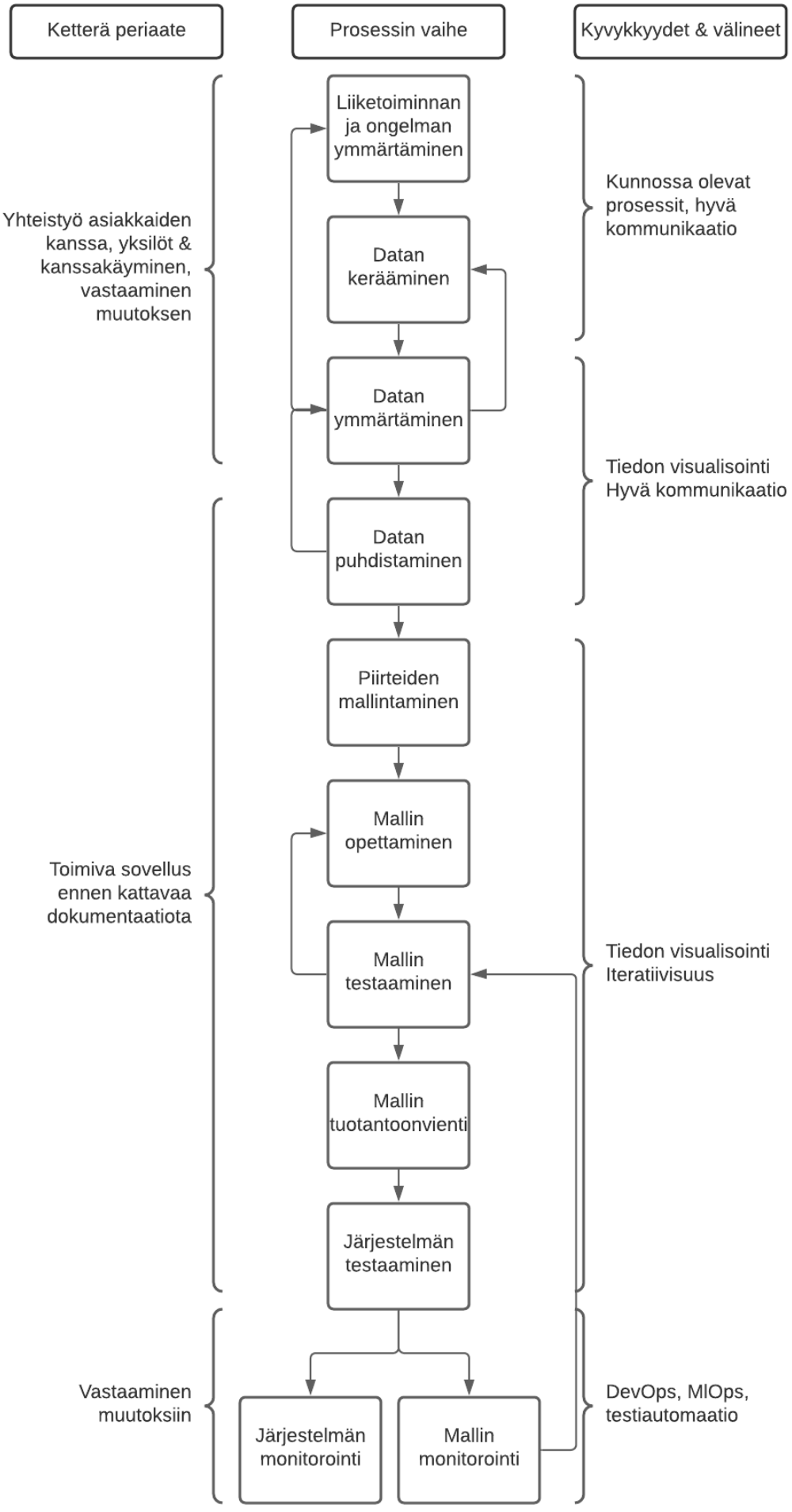
Kuva 7.1. *Kaavio tilastollisen menetelmän valinnasta [63]*

Koneoppimissovelluksien, etenkin syvien neuroverkkojen tapauksessa voi olla tilanteita, joissa uuden neuroverkon opettaminen vie aikaa hyvin paljon mutta tarjolla olisi johonkin toiseen ongelmaan suunniteltu malli. Eräs ratkaisu olisi käyttää apuna niin kutsuttua siirto-oppimista (eng. *transfer learning*, jonka ajatuksena on se että yhdellä aineistolla kehitetyn mallin tietämystä voidaan käyttää toiseen aineistoon siten, että dataa tarvitaan vähemmän kuin siihen että mallia lähdettäisiin kehittämään niin kutsutusti puhtaalta pöydältä. [1] Projektien läpiviennin kannalta siirto-oppimisen hyödyt voivat tulla esiin siinä, että kehitettyjä malleja voidaan hyödyntää myös toisissa ongelmissa.

Mallinnuksen sekä oppimisen työkalujen lisäksi data-analyysin sekä visualisoinnin keinot voivat olla hyödyksi koneoppimisprojekteissa. Datavisualisoinnin avulla datan ongelmista, kuten erilaisista jakaumista on helppo kommunikoida joten ne tulisi ottaa projektiin mukaan.

7.4 Koneoppimisen prosessimalli ketteriin koneoppimisprojekteihin

Edellä esitettyjen perustelujen mukaan rakennettu koneoppimisen prosessimalli ketteriä menetelmiä hyödyntäviin projekteihin näyttää kuvan 7.2 mukaiselta. Mallissa eri proses-



Kuva 7.2. *Esitetty koneoppimisprosessi*

sin vaiheet on jaettu ketterien periaatteiden [35] mukaisiin osiin. Prosessi on jaettu yhteentoista vaiheeseen, jotka ovat seuraavat

- Liiketoiminnan ja ongelman ymmärtäminen
- Datan kerääminen
- Datan ymmärtäminen
- Datan puhdistaminen
- Piirteiden mallintaminen
- Mallin opettaminen
- Mallin testaaminen
- Mallin tuotaantoonvienti
- Järjestelmän testaaminen
- Järjestelmän monitorointi
- Mallin monitorointi

.

Liiketoiminnan ja ongelman ymmärtäminen, datan kerääminen sekä datan ymmärtäminen voidaan nähdä ketterään kehitykseen liittyvänä yhteistyönä asiakkaan kanssa, muutoksiin vastaamisena sekä kanssakäymisenä eri yksilöiden ja tiimien välillä. Datan ymmärtämisessä korostuu vuorovaikutus ja kommunikaatio niin asiakkaiden kuin muiden tiimissä työskentelevien ihmisten kanssa ja sen kautta voidaan edistää organisaation oppimista. Koneoppimisen kannalta näillä on myös merkitys siinä, että ymmärretään onko oikeasti tarvetta kehittää koneoppimissovellusta vai onko jokin muu parempi ratkaisu liiketoiminnan ongelmiin. Organisaation kyvykkyyksien ja käytössä olevien välineiden kannalta keskeistä näissä vaiheissa on hyvät prosessit eli esimerkiksi se, että kehittäjät saavat tarvitsemansa datan käyttöön helposti. Etätyön aikana hyvässä kommunikaatiossa korostuu myös tekniset apuvälineet eli esimerkiksi videoneuvotteluratkaisuiden käyttö. Datan visualisoinnin avulla voidaan edistää kommunikaatiota.

Datan puhdistaminen, piirteiden mallintaminen, mallin opettaminen, testaaminen ja tuotantoonvienti sekä järjestelmän testaaminen korostavat toimivan sovelluksen tekemistä ennen kattavaa dokumentaatiota. Näissä prosessin vaiheissa korostuu haastatteluiden perusteella ohjelmointiympäristöjen käyttö. Näiden vaiheiden kanssa työskentely on hyvin iteratiivisista ja siinä pyritään siihen, että mallia saadaan muokattua paremmaksi.

Järjestelmän testaaminen sekä monitorointi sekä mallin monitorointi vastaavat mahdollisiin muutoksiin kehitetyssä järjestelmässä sekä osaltaan mahdollistavat myös automaattisen reagoinnin esimerkiksi muuttuneisiin liiketoimintatarpeisiin jos esimerkiksi järjestelmän hyödyntämä data muuttuu toisenlaiseksi tuotannossa. Tässä vaiheessa prosessia korostuu erityisesti DevOps- sekä MlOps-työkalujen käyttö. Projektien kannalta tarkasteltuna nämä vaiheet saattavat olla jo projektin jälkeisiä vaiheita, etenkin jos kehitetty järjestelmä on jatkuvassa käytössä mutta ei aktiivisessa kehityksessä.

7.5 Vertailu olemassa oleviin malleihin

Tässä työssä esitetty malli on pitkälti samankaltainen muiden kirjallisuudessa esitettyjen koneoppimisen prosessimallien kanssa. Mallien keskeisimpiä eroavaisuuksia on erot siinä miten tarkasti yksittäisiä vaiheita esitetään. Ketteriin periaatteisiin tai ketterien menetelmien käyttöön ei monissa malleissa oteta lainkaan huomiota, poislukien malleissa jotka on erityisesti suunniteltu vaikkapa Scrum-pohjaisten prosessien päälle.

Miikka Koskinen esittää diplomityössään prosessin, jossa kuvataan koneoppimisprojektin eri vaiheet [62]. Koskisen esittämä malli rakentuu hyvin pitkälti CRISP-DM-mallin päälle ja sen vaiheet on perusteltu hyvin tieteellisen tutkimuksen perusteella. Toisin kuin vaikkapa CRISP-DM-mallissa, Koskisen esittämä prosessi ottaa kantaa myös siihen että projekti päättyy joskus. Tämän tutkielman mallissa projektin päättymistä ei huomioida eksplisiittisesti.

CRISP-DM-malliin verrattuna tässä työssä esitetyssä mallissa on paljon enemmän vaiheita ja korostetaan myös sitä, että dataa on haettava jostain eikä se ole suoraan saatavilla. Erona voi olla se, että CRISP-DM -malli rakennetaan sille lähtökohdalle, että on olemassa jotakin dataa jota halutaan louhia tai mallintaa eikä välttämättä suoraan sille että on jokin liiketoiminnan ongelma ratkaistavana.

TDSP-malliin verrattuna tässä työssä esitetyssä mallissa on jonkin verran samaa, erityisesti siinä mielessä että tässä työssä otetaan kantaa myös liiketoiminnan ymmärtämiseen sekä joiltakin osin myös tarvittaviin työkaluihin. TDSP kuitenkin korostaa käytettäviä työkaluja sekä projektin rakennetta hyvin paljon, luultavasti siksi että se on rakennettu olemassa olevien tuotteiden käytön päälle.

8 YHTEENVETO

8.1 Johtopäätelmät

Tämän työn teoriaosassa perehdytiin prosessimenetelmiin, ketteriin menetelmiin sekä koneoppimiseen jotta voidaan ymmärtää ketterien menetelmien käyttöä koneoppimisprojekteissa paremmin. Prosessiosuudessa tarkasteltiin ohjelmistokehityksen klassista prosessimallia sekä koneoppimisprojekteihin kehitettyjä prosesseja.

Koneoppimisen tarkastelun tavoitteena oli ymmärtää mitä koneoppimissovellusten kehittämiseen liittyy ja tuoda esiin siihen liittyviä työvaiheita. Tärkein havainto koneoppimisesta on sen satunnaisuuteen perustuva luonne, joka muovaa menetelmiä ja tapoja, joilla tarkastellaan esimerkiksi jonkin algoritmin oikeellisuutta.

Ohjelmistokehityksen prosessimallit nähdään yleisesti ottaen ongelmallisena ohjelmistokehityksessä ja siihen nähdään liittyvän riskejä. Näiden mallien tarkastelu kuitenkin auttaa hahmottamaan eri vaiheita, joita ohjelmistojen kehittämiseeen liittyy. Koneoppimismallien kehittämiseen on kehitetty useita erilaisia malleja, joissa korostuu kaikissa samanlaiset piirteet. Eroavaisuuksia mallien välillä on joissakin yksityiskohdissa sekä siinä, miten niissä huomioidaan esimerkiksi ohjelmistokehitykseen liittyvät ominaispiirteet kuten tuotantoonvienti sekä tuotannossa olevan koodin monitorointi ja laadunvalvonta.

Ketterien menetelmienperehdytiin ketteriin menetelmiin organisaatioiden, ohjelmistokehityksen sekä UX-suunnittelun näkökulmasta siten, että pyrittiin löytämään mahdollisimman laajasti keinoja joilla työtä organisaatioissa voidaan muuttaa ketterämmäksi ja mitä ongelmia siihen liittyy. Ketterät menetelmät ovat hyödynnettävissä organisaatioissa niin liiketoiminnan kehittämisen yhteydessä kuin myös ohjelmistokehityksessä. Monet organisaation Lean-ajatteluun liittyvät menetelmät korostuvat myös ohjelmistokehityksessä, erityisesti DevOps-filosofiassa, jossa pyritään tehostamaan ja nopeuttamaan kehitettyjen ohjelmistojen viemistä tuotantoon koska tavoitteena on pyrkiä lyhentämään aikaa, joka tiimillä kuluu muutoksien käsittelyyn. Koneoppimisprojektien näkökulmasta Lean-menetelmät sekä organisaation kehittäminen on myös ratkaisevassa roolissa, koska mallien kehitätminen saattaa vaatia panosta myös projektin työryhmän ulkopuolelta: esimerkiksi tarvittava tieto ei välttämättä ole aina saatavilla, esimerkiksi organisaatioon liittyvien haasteiden tai sääntelyn vuoksi. Kirjallisuuden perusteella ketteriä menetelmiä ja niiden käyttöä on tutkittu paljon ja ohjelmistokehityksen menetelmien hyödyntäminen nähdään laajemmin osana organisaation toiminnan tehokkuutta sekä organisaatiossa olevien hen-

kilöiden hyvinvoinnin sekä kulttuurin yhtenä tekijänä.

Koneoppimisprojektien näkökulmasta etenkin ketterät menetelmät sekä DevOps (sekä sen johdannaiset kuten MlOps ja DataOps) tarjoavat koneoppimisprojekteille työkaluja, joiden avulla ongelmia voidaan ratkaista tehokkaasti organisaation muut tekijät huomioiden.

Kirjallisuuden perusteella ketterien menetelmien painopiste on siirtymässä siihen, että organisaatiot tukevat ihmisiä työssään ja pyrkivät auttamaan oppimisessa sekä parhaan mahdollisen työtuloksen saamisessa. Tavoitteena on pyrkiä luomaan nopea oppimissykli, joka on turvallista toteuttaa. Koneoppimisprojektien näkökulmasta kokeilukulttuuri sekä organisaation oppiminen ovat tärkeitä asioita, koska uusia menetelmiä luodaan jatkuvasti lisää ja mallinnukseen liittyvä epävarmuus edellyttää sitä, että mahdolliset epäonnistumiset saadaan kiinni mahdollisimman nopeasti.

Työssä tehdyn haastattelututkimuksen otos oli pieni, mutta haastattelujen tulokset olivat melko yhdenmukaiset siinä mielessä, että ketterät menetelmät nähtiin hyvin hyödyllisenä koneoppimisprojekteissa ja prosessimalleja ei juurikaan tunnettu eikä lainkaan hyödynnetty oikeissa koneoppimisprojekteissa. Haastattelun perusteella koneoppimisen prosessimallit voidaan nähdä teoreettisena mallina sille miten koneoppimista hyödyntäviä sovelluksia kehitetään, mutta käytännön työssä niille ei tuntunut löytyvän käyttöä tai kiinnostusta niiden käyttöön.

Myös ketterät menetelmät nähtiin haastatteluissa toissijaisena, mutta ketterän kehityksen peruspiirteet ja -periaatteet korostuivat tekemisessä ja haastateltavien vastauksissa. Tärkeänä nähtiin se, että valittava prosessi huomioi mahdollisen epävarmuuden projekteissa sekä tukee iteratiivista kehitystä.

Koneoppimisprojektit esiintyivät haastatteluissa usein osana muuta ohjelmistokehitystä, mutta koneoppimisen käytölle nähtiin myös tarvetta myös muunlaisissa ympäristöissä, esimerkiksi osana liiketoimintatiedon visualisointia tai antureista saatavan datan visualisointia. Tämä nostaa esiin tarpeen sille, että organisaation on oltava valmis tulemaan ketterä myös ohjelmistokehitysprojektien ulkopuolella. Ohjelmistokehityksen suuri korostuminen osana koneoppimissovellusten myöhäisempää käyttöä voidaan nähdä kuitenkin hyödyllisenä, koska silloin koneoppimisen kehittäjät voivat työskennellä hyvin osana ohjelmistokehitystä ilman, että tarvitaan erillisiä datatiede- tai koneoppimissprinttejä tai suunnitteluita.

Tässä työssä esitetty prosessimalli, joka luotiin haastattelujen sekä kirjallisuuden perusteella, vastaa hyvin paljon monia kirjallisuudessa esitettyjä malleja mutta erona toisiin malleihin voidaan pitää sitä, että siinä pyrittiin huomioimaan myös taustalla olevat ketterien menetelmien periaatteet sekä hyvin yleisellä tasolla myös erilaisia työkaluja tai menetelmiä, jotka mahdollistavat ketteryyden toteutumisen projekteissa.

Tämän työn perusteella voisi sanoa, että prosessimallien tarpeellisuus korostuu eri vaiheiden ymmärtämisessä sekä organisaation toiminnassa yksittäisten vaiheiden tarkan määrittelyn tai aikatauluttamisen sijaan. Voisi jopa ajatella, että prosessimallit vastaavat

organisaatiossa siihen miksi jotakin tehdään sen sijaan että mitä pitää tehdä.

Luvun 4.5 mainitun kontingenssimallin perusteella voidaan ajatella, että koneoppimisprojektin varmuus projektin lopputulemassa on aina epävarmuutta, jolloin ketterät menetelmät sopivat niiden kehittämiseen paremmin kuin tiukat prosessimallit. Kuitenkin, koneoppimismenetelmiä hyödynnetään monella eri toimialalla jolloin joillakin toimialoilla tiukkojen prosessien hyödyntäminen voi olla edullista.

8.2 Mahdolliset jatkotutkimukset

Tämän tutkielman haastateltava joukko oli melko suppea, joten tutkimusta voisi hyvin jatkaa haastattelemalla suurempaa joukkoa datatieteilijöitä tai tutkimalla ketterien menetelmien sekä ohjelmistotekniikan käyttöä koneoppimisprojekteissa kyselytutkimuksen avulla.

Eräs puuttuva näkökulma tässä tutkielmassa oli asiakkaan eli sen, jolle koneoppimissovelluksia tehdään, näkökulma. Olisi kiinnostavaa tutkia lisääkö koneoppimisprojekti organisaation ymmärrystä omasta datastaan tai erilaisista liiketoimintaprosesseista.

LÄHTEET

[1] Murphy, K. P. *Probabilistic Machine Learning: An introduction*. MIT Press, 2021. URL: http://mlbayes.ai.

[2] Friedman, J., Hastie, T. ja Tibshirani, R. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2017.

[3] Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

[4] Winder, P. *Reinforcement Learning*. O'Reilly Media, 2020.

[5] Goodfellow, I., Bengio, Y. ja Courville, A. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Thakur, A. *Approaching (almost) any machine learning problem*. Abhishek Thakur, 2020.

[7] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[8] Sidey-Gibbons, J. A. ja Sidey-Gibbons, C. J. Machine learning in medicine: a practical introduction. *BMC medical research methodology* 19.1 (2019), 1–18.

[9] Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H. H. ja Crnkovic, I. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. *International Conference on Agile Software Development*. Springer, Cham. 2019, 227–243.

[10] Sommerville, I. Software engineering 9th Edition. *ISBN-10* 137035152 (2011), 18.

[11] Moilanen, J., Niinioja, M., Seppänen, M. ja Honkanen, M. *API Economy 101: Changes Your Business*. BoD-Books on Demand, 2019.

[12] Royce, W. W. Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th international conference on Software Engineering*. 1987, 328–338.

[13] *Waterfall process*. Viitattu 9.6.2021. URL: https://martinfowler.com/bliki/WaterfallProcess.html.

[14] Fayyad, U., Piatetsky-Shapiro, G. ja Smyth, P. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM* 39.11 (1996), 27–34.

[15] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C. ja Wirth, R. The CRISP-DM user guide. *4th CRISP-DM SIG Workshop in Brussels in March*. Vol. 1999. 1999.

[16] Zhang, S., Zhang, C. ja Yang, Q. Data Preparation for Data Mining. *Applied Artificial Intelligence* 17 (toukokuu 2003), 375–381. DOI: 10.1080/713827180.

[17] *GDPR*. URL: https://tietosuoja.fi/gdpr (viitattu 29. 01. 2021).

[18] Géron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[19] *TensorFlow Extended (TFX) | ML Production Pipelines*. URL: https://www.tensorflow.org/tfx/.

[20] Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S. ja Mueller, K.-R. *Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology*. 2020. arXiv: 2003.05155 [cs.LG].

[21] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B. ja Zimmermann, T. Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE. 2019, 291–300.

[22] *Team Data Science Process Documentation*. Viitattu 30.5.2021. URL: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/.

[23] Souza Nascimento, E. de, Ahmed, I., Oliveira, E., Palheta Márcio Piedade ja Steinmacher, I. ja Conte, T. Understanding Development Process of Machine Learning Systems: Challenges and Solutions. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, 1–6.

[24] Lakshmanan, V., Robinson, S. ja Munn, M. *Machine learning design patterns*. "O'Reilly Media, Inc.", 2020.

[25] Weyuker, E. J. On testing non-testable programs. *The Computer Journal* 25.4 (1982), 465–470.

[26] Davis, M. D. ja Weyuker, E. J. Pseudo-oracles for non-testable programs. *Proceedings of the ACM'81 Conference*. 1981, 254–257.

[27] Murphy, C., Kaiser, G. E. ja Arias, M. An approach to software testing of machine learning applications (2007).

[28] SUGIMORI, Y., KUSUNOKI, K., CHO, F. ja UCHIKAWA, S. Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. *International Journal of Production Research* 15.6 (1977), 553–564. DOI: 10.1080/00207547708943149. eprint: https://doi.org/10.1080/00207547708943149. URL: https://doi.org/10.1080/00207547708943149.

[29] Singh, J. ja Singh, H. Kaizen philosophy: a review of literature. *IUP journal of operations management* 8.2 (2009), 51.

[30] Blank, S. Why the lean start-up changes everything. *Harvard business review* 91.5 (2013), 63–72.

[31] Torkkola, S. Lean asiantuntijatyön johtamisessa. *Helsinki: Talentum Pro* 2.3 (2015).

[32] Cantor, M., MacIsaac, B. ja Mannan, R. Steering Software Development Workflow: Lessons from the Internet. *IEEE Software* 33.5 (2016), 96–102.

[33] Martin, K. *Value stream mapping : how to visualize work and align leadership for organizational transformation*. eng. 1st edition. New York, New York: McGraw-Hill Education, 2014. ISBN: 0-07-182894-X.

[34] Poppendieck Mary ja Poppendieck, T. *Lean software development : an Agile toolkit*. eng. 1st edition. The agile software development series. Upper Saddle River, New Jersey: Addison Wesley Professional, 2003. ISBN: 0321150783.

[35] *Agile Manifesto*. URL: https://agilemanifesto.org/.

[36] Abrahamsson, P., Salo, O., Ronkainen, J. ja Warsta, J. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439* (2017).

[37] Leffingwell, D. *SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional, 2018.

[38] Schwaber, K. ja Sutherland, J. The scrum guide. *Scrum Alliance* 21 (2011), 19.

[39] J. Markkula ja M. Oivo, M. O. A. ja. Kanban in software development: A systematic literature review. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. 2013, 9–16. DOI: 10.1109/SEAA.2013.28.

[40] Miller, G. J. Agile problems, challenges, & failures. Project Management Institute. 2013.

[41] *Modern Agile*. Viitattu 31.5.2021. URL: https://modernagile.org/.

[42] Isomursu, M., Sirotkin Andrey ja Voltti, P. ja Halonen, M. User Experience Design Goes Agile in Lean Transformation - A Case Study. *2012 Agile Conference*. IEEE. 2012, 1–10.

[43] Gothelf, J. *Lean UX: Applying lean principles to improve user experience*. "O'Reilly Media, Inc.", 2013.

[44] Jabbari, R., Ali, N. bin, Petersen, K. ja Tanveer, B. What is DevOps? A systematic mapping study on definitions and practices. *Proceedings of the Scientific Workshop Proceedings of XP2016*. 2016, 1–11.

[45] Kim, G., Humble, J. ja Debois Patrick ja Willis, J. *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.

[46] *The Three Ways: The Principles Underpinning Devops*. Viitattu 1.5.2021. URL: https://itrevolution.com/the-three-ways-principles-underpinning-devops/.

[47] Davis, J. ja Daniels, R. *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. "O'Reilly Media, Inc.", 2016.

[48] *DevOps Culture: Westrum organisational culture*. Haettu 28.5.2021. URL: https://cloud.google.com/architecture/devops/devops-culture-westrum-organizational-culture.

[49] Westrum, R. A typology of organisational cultures. *BMJ Quality & Safety* 13.suppl 2 (2004), ii22–ii27. ISSN: 1475-3898. DOI: 10.1136/qshc.2003.009522. eprint: https://qualitysafety.bmj.com/content/13/suppl_2/ii22.full.pdf. URL: https://qualitysafety.bmj.com/content/13/suppl_2/ii22.

[50] Humble, J. ja Kim, G. *Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations*. IT Revolution, 2018.

[51] Ereth, J. DataOps-Towards a Definition. *LWDA* 2191 (2018), 104–112.

[52] Atwal, H. *Practical DataOps: Delivering Agile Data Science at Scale*. Springer, 2019.

[53] *MLOps: Continuous delivery and automation pipelines in machine learning*. Haettu 1.5.2021. URL: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[54] Group, S. *The Standish Group Report*. URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf.

[55] Lagstedt, A. Selecting the right method for the right project. Tohtorinväitöskirja. PhD Thesis. Turku University. http://urn. fi/URN, 2019.

[56] Howsawi, E. M., Eager, D. ja Bagia, R. Understanding project success: The four-level project success framework. *2011 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE. 2011, 620–624.

[57] Seaman, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25.4 (1999), 557–572.

[58] Petersen, K. ja Gencel, C. Worldviews, research methods, and their relationship to validity in empirical software engineering research. *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*. IEEE. 2013, 81–89.

[59] Hirsjärvi, S. *Tutkimushaastattelu : teemahaastattelun teoria ja käytäntö*. fin. Helsinki, 2008.

[60] He, X., Zhao, K. ja Chu, X. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* 212 (2021), 106622.

[61] Brooks Jr, F. P. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[62] Koskinen, M. Koneoppiminen ketterässä ohjelmistoprojektissa. Tutkielma. Tampereen yliopisto, marraskuu 2020.

[63] McElreath, R. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman ja Hall/CRC, 2018.