TURUN YLIOPISTO

TIIVISTELMÄ

	Kandidaatintutkielma
x	Pro gradu -tutkielma
	Lisensiaatintutkielma
	Väitöskirja

Oppiaine	Tietojärjestelmätiede	Päivämäärä	25.3.2021
Tekijä	Tommi Penttilä	Sivumäärä	62 + liitteet
Otsikko	Jatkuvan toimituksen käyttöönotto tietovarastokehityksessä		
Ohjaajat	Prof. Jouni Similä, Prof. Hannu Salmela		

Tiivistelmä

Ohjelmistokehityksessä ohjelmistokoodin nopea julkaisu loppukäyttäjien saataville saattaa tarjota yrityksille merkittävää kilpailuetua. Monet menestyvät yritykset julkaisevat ohjelmistostaan uuden version useita kertoja päivässä. Tiheän julkaisuvälin mahdollistaa niin kutsuttu jatkuva toimitus, joka on viime aikoina yleistynyt ohjelmistokehityksessä. Jatkuvan toimituksen menetelmiä ei sen hyödyistä huolimatta kuitenkaan juuri käytetä dataintensiivisessä kehityksessä, kuten tietovarastokehityksessä.

Tässä suunnitteluttutkimuksessa tarkastellaan jatkuvan toimituksen käyttöönottoa ja tavoitteena on luoda toimintamalli, jonka avulla jatkuva toimitus voidaan ottaa käyttöön tietovarastokehitysprojektissa. Tutkimuksessa tarkastellaan kirjallisuuskatsauksen avulla, millaiset tekijät vaikuttavat jatkuvan toimituksen käyttämiseen ja käyttöönottoon. Koska tietovarastokehityksen toimintoja, erityisesti jatkuvan toimituksen kontekstissa, on tutkittu hyvin vähän, tarkastellaan jatkuvaa toimitusta ohjelmistokehityksen ympäristössä. Tarkoituksena on yleistää tehdyt havainnot tietovarastokehitykseen vertailemalla ohjelmistokehityksen ja tietovarastokehityksen eroja jatkuvan toimituksen kannalta oleellisilta osilta.

Kirjallisuuskatsauksen perusteella luotua toimintamallia demonstroidaan KEHA-keskuksen tietovarastokehitysprojektissa, jossa jatkuva toimitus halutaan ottaa käyttöön. Toimintamallin toimivuutta arvioidaan seuraamalla jatkuvan toimituksen käyttöönottoa sekä haastattelemalla projektin henkilöstöä toimintamallin käyttämisen jälkeen. Seurannan ja haastattelujen avulla toimintamallia jatkokehitetään.

Tutkimuksessa havaittiin, että jatkuvan toimituksen käyttöönotto on usein haastavaa. Oikeita toimintatapoja ja teknologisia työkaluja hyödyntämällä näitä haasteita voidaan kuitenkin lieventää, jolloin jatkuvan toimituksen kiistattomat hyödyt saadaan käyttöön nopeasti ja kustannustehokkaasti. Tutkimus antaa myös näyttöä siitä, ettei jatkuvan toimituksen käyttäminen rajoitu vain ohjelmistokehitykseen vaan sitä voidaan hyödyntää myös tietovarastokehityksessä.

Avainsanat	Jatkuva toimitus, tietovarastokehitys

TURUN YLIOPISTO
Kauppakorkeakoulu

JATKUVAN TOIMITUKSEN KÄYTTÖÖNOTTO TIETOVARASTOKEHITYKSESSÄ

KEHA-keskuksen tietovarastointi

Tietojärjestelmätieteen
pro gradu -tutkielma

Laatija:
Tommi Penttilä

Ohjaajat:
Prof. Jouni Similä
Prof. Hannu Salmela

25.3.2021
Turku

SISÄLLYS

KUVAT

TAULUKOT

LYHENTEET

ETL-prosessi	Prosessi, jossa data poimitaan (engl. *Extract*) lähdejärjestelmistä, muokataan haluttuun muotoon (engl. *Transform*) ja ladataan tietovarastoon (engl. *Load*).
BI	Liiketoimintatiedon kokonaisvaltainen hyödyntäminen (engl. *Business Intelligence*).

BLOB	Kokoelma binääridataa yhdistettynä yhdeksi kokonaisuudeksi (engl. *Binary Large OBject*).
MPP	Tiedon prosessointitapa, jossa käytetään merkittävän suuria määriä rinnakkaisia tietokoneprosessoreita yhdenaikaisesti (engl. *Massively Parallel Processing*).
IaaS	Tietoteknisen infrastruktuurin, kuten palvelimien, palvelinsalien ja näiden ylläpidon tarjoaminen palveluna (engl. *Infrastucture as a Service*).
OLAP	Reaaliaikaista tiedon analyyttista käsittelyä (engl. *OnLine Analytical Processing*).
DV	Tietovaraston mallintamiseen tarkoitettu Data Vault -mallinnustapa.
CI-palvelin	Jatkuvan integraation toimintoihin käytettävä palvelin (engl. *Continuous Integration server*).
IaC	Tilapäisten palvelimien (ks. IaaS) hallinnointi käyttäen koneluettavia määrittelytiedostoja, joilla ympäristöjen rakentaminen voidaan automatisoida (engl. *Infrastructure as Code*).

1 JOHDANTO

1.1 Taustaa

Loppukäyttäjien käytössä olevaa ohjelmistoa kehitettäessä koodin kirjoittaminen tapahtuu usein erillisessä, vain kehitykseen tarkoitetussa ympäristössä. Tässä ympäristössä muutoksia voidaan tehdä vapaasti ja niiden perustoimintoja voidaan testata ilman, että muutokset heijastuvat ohjelmiston loppukäyttäjille. Ilman tällaisen ympäristön olemassaoloa ohjelmiston jatkokehitys olisi lähestulkoon mahdotonta. Erillinen ympäristö aiheuttaa kuitenkin merkittävää lisätyötä, kun samat muutokset täytyy tehdä usein myös testausympäristöön ja käyttäjärajapinnassa olevaan tuotantoympäristöön. Tässä tutkimuksessa termillä julkaisukäytänne viitataan niihin toimintoihin, joita kehitystiimin täytyy tehdä, että se saa muutokset julkaistua ympäristöstä toiseen.

Ohjelmistokehityksessä ohjelmistokoodiin tehtyjen muutosten julkaiseminen käyttäjärajapintaan on ollut perinteisesti aikaa vievä prosessi, johon liittyy paljon epävarmuutta ja manuaalista työtä, mikä hidastaa muutosten siirtämistä käyttäjärajapintaan (Soni 2015, 85; Zhu ym. 2016, 33). Markkinoiden nopeat muutokset kuitenkin aiheuttavat painetta muutosten nopeampaan siirtymiseen (Claps ym. 2015, 21; Rathod & Surve 2015, 1). Manuaalisen työn automatisointi myös luonnollisesti vapauttaa ohjelmistokehittäjien työtunteja varsinaisen kehitystyön tekemiseen, mikä nostaa motivaatiota julkaisutoimintojen automatisoimiseen. Pyrkimys kehitystyön manuaalisten toimintojen automatisointiin on johtanut DevOps-toimintamallin kehittymiseen. (Zhu ym. 2016, 33.) DevOps on akronyymi englannin kielen sanoista *development* ja *operations,* ja sillä viitataan kehitystyön ja operationaalisten toimintojen yhdistämiseen. Sen yhtenä tärkeimpänä tarkoituksena on lyhentää aikaa, joka kuluu kehittäjän tekemän muutoksen siirtymisessä käyttäjärajapintaan, ilman että korkeasta laadusta joudutaan tinkimään (Bass ym. 2015). Jatkuva toimitus (*engl. continuous delivery*) tähtää juuri tähän ja se onkin yksi DevOpsin tärkeimmistä prosesseista (Gill ym. 2018, 9). Jatkuvalla toimituksella viitataan tilanteeseen, jossa ohjelmistoa julkaistaan tuotantoon läpi sen elinkaaren, kehitystiimi panostaa kehitystyötä tehdessään siihen, että ohjelmisto on jatkuvasti julkaistavissa, kehittäjät saavat automaattisesti palautetta ohjelmiston julkaisuvalmiudesta muutoksia tehdessään ja julkaisut eri ympäristöihin tapahtuvat automaattisesti yksinkertaisilla komennoilla (Fowler 2013).

Jatkuvan toimituksen käyttäminen ohjelmistokehityksessä tuottaa merkittävää etua kehitystiimille ja koko organisaatiolle. Havaittuja hyötyjä on muun muassa ohjelmistojen

nopeampi ja luotettavampi markkinoille julkaisu (Fowler 2013; Chen 2015, 52–53; Leppänen ym. 2015, 67; Virmani 2015, 81; Soni 2015, 85; Chen 2017, 73) sekä vahvempi luottamus kehitystiimin ja lopputuotteen käyttäjän välillä, kun jatkuva toimitus lyhentää kehityssykliä antaen näin loppukäyttäjälle useampia mahdollisuuksia arvioida tehdyn kehitystyön soveltuvuutta (Humble & Molesky 2011, 11–12; Fowler 2013; Chen 2015, 52; Leppänen ym. 2015, 65–66; Soni 2015, 85–86; Chen 2017, 73; Siqueira ym. 2018, 40). Jatkuva toimitus mahdollistaa myös nopeamman reagoinnin muutoksiin (Soni 2015, 85; Siqueira ym. 2018, 40–41) ja jakaa vastuuta kehitystiimin ja operationaalisen tiimin välillä, lisäten molempien osapuolien osallistumista ja kiinnostusta sekä varsinaiseen kehitystyöhön että kehityksen jälkeisiin operationaalisiin tehtäviin (Leppänen ym. 2015, 67; Siqueira ym. 2018, 41). Jatkuva toimitus myös tarjoaa ohjelmiston julkaisulle parempaa riskienhallintaa vähentämällä inhimillisten riskien mahdollisuutta (Humble & Molesky 2011, 10–11; Humble 2018, 35). Kaikki edellä mainitut hyödyt yhdessä johtavat paremman ohjelmiston julkaisuun nopeammin ja luotettavammin. Myös käytännön esimerkkejä jatkuvan toimituksen hyödyistä on useita (ks. esim. Allspaw & Hammond 2009; Rossi ym. 2016; Hilton ym. 2017).

Huolimatta erinomaisista tuloksista, joita on saavutettu jatkuvan toimituksen käyttöönotolla ohjelmistokehityksen puolella, sitä ei ole kuitenkaan yleisesti omaksuttu toimintatavaksi tietovarastokehityksessä, vaikka varsinaisia syitä menetelmän käyttämättä jättämiselle ei olekaan (Puonti ym. 2018, 250). Tietovarastolla viitataan tässä tutkimuksessa yritystason tietovarastoa (engl. *enterprise data warehouse*), johon kerätään dataa useista eri tietolähteistä esimerkiksi raportointia varten. Olennaista on se, että tietovarastoa täytyy muuttaa usein, sillä muuten jatkuvan toimituksen käyttäminen on turhaa. Tällainen tietovarastointi ja sen avulla tehtävä tietojohtaminen on tunnistettu yhdeksi kuudesta fyysisestä ominaisuudesta, joita IT-infrastruktuuri vaatii, jotta sen avulla voidaan saavuttaa strategista ketteryyttä (Weill ym. 2002, 60). Tietovarastointi on kriittinen osa tietoteknologiaa, sillä se toimii perustana BI-toiminnoille ja tietolouhinnalle (engl. *data mining*) (Rahman ym. 2013). Niinpä vaikuttaisi siltä, että jatkuvan toimituksen käyttämättä jättäminen tietovarastokehityksessä aiheuttaa organisaatiolla merkittävää hyödyn menetystä.

Jatkuvan toimituksen käyttämisestä tietovarastokehityksessä ei myöskään Puontia ym. (2018) lukuun ottamatta löydy akateemista kirjallisuutta. Tietovarastoinnin tutkimus on keskittynyt lähinnä tietomallinnukseen, tietovaraston arkkitehtuuriin, OLAP-prosesseihin, tietovaraston suorituskykyyn, testaukseen ja tietoturvallisuuteen (Chandra

& Gupta 2019, 218–219). Varsinaiseen kehitystyöhön painottuvaa kirjallisuutta ei tämän tutkimuksen kirjallisuuskatsauksessa löydetty. Myös Ereth (2018, 6) toteaa, että DevOps-menetelmien hyödyntämisestä dataintensiivisissä ratkaisuissa tarvitaan syväluotaavia tapaustutkimuksia, koska aihetta ei ole vielä tutkittu tarpeeksi. Niinpä tässä tutkimuksessa tarkastellaan jatkuvan toimituksen ja tietovarastokehityksen suhteita. Tutkimuksen tavoitteena on luoda toimintamalli jatkuvan toimituksen käyttöönottamiseksi tietovarastokehityksen alueella sekä demonstroida ja arvioida toimintamallia tapaustutkimuksen avulla.

1.2 Tapaus

Tiedolla ja sen hyödyntämisellä on merkittävä rooli liiketoiminnassa. Dataan pohjautuva päätöksenteko parantaa yrityksen tuloksellisuutta ja datan hyödyntäminen on välttämätön osa monia liiketoimintaprosesseja (Seddon ym. 2017). Taloudelliset intressit ajavat yrityksiä kehittämään omia tietopohjaisia toimintojaan ja kehittämään tiedon hyödyntämistä organisaatiossaan, mutta myös julkiset toimijat ovat alkaneet osoittaa kiinnostusta omien datamassojensa hyödyntämiseen.

Tässä suunnittelututkimuksessa (engl. *design research*) tapauksena on KEHA-keskuksen tietovarastokehitysprojekti. KEHA-keskus on Suomen valtion virasto, joka vastaa ELY-keskusten ja TE-toimistojen kehittämis- ja hallinnointipalveluiden tuottamisesta (ks. KEHA-keskus). Esimerkkiprojektin tavoitteena on luoda KEHA-keskuksen useiden järjestelmien viidakossa yksittäinen ja kokonaisvaltainen tietovarasto sekä siihen sidottu yhtenäinen raportointiympäristö. Tietovarasto sisältää siis tietoa hyvin laajasti eri osa-alueilta muun muassa kuntakohtaisista työttömyystilanteista, erilaisten tukien maksatuksista, sisäisistä henkilöstöasioista sekä eri alueiden ympäristökuormituksesta. Tietovarastoon lisätään myös jatkuvasti uusia tietolähteitä, mikä nostaa tarvetta julkaisukäytänteiden automatisoinnille. Tässä tutkimuksessa keskitytään vain projektin tietovarastokehitys puoleen, eikä raportointiin liittyviin prosesseihin oteta kantaa.

Erityistä projektissa on tiedon varastointitavaksi valittu DV-menetelmä (ks. Lindstedt & Olschimke 2015) ja DV-mallinnukseen käytettävä kolmannen osapuolen tarjoama sovellus Data Vault Modeller, joka automatisoi DV-mallinnukseen yleensä liittyvää suurta manuaalista työtä. Sovellus luo mallinnuksen yhteydessä automaattisesti tietovaraston metadataa, jonka avulla julkaisuprosesseja olisi mahdollista automatisoida, ja DV-Modeller voisi toimia jatkuvan toimituksen alustana. Esimerkkiprojektissa tietovarasto sisältää kuitenkin GDPR-säädännön (ks. esim. Albrecht 2016; Zarsky 2016) alaista

tietoa, joten tietoturvaan on jouduttu kiinnittämään erityistä huomiota. Niinpä kehityksessä käytettävät eri ympäristöt on eriytetty toisistaan täysin, eikä DV-Modellerin sisäisiä julkaisuominaisuuksia voida käyttää. Tätä ongelmaa pyritään projektissa kiertämään projektinhallintatyökaluksi valitulla Azure DevOps -alustalla (ks. Azure DevOps dokumentaatio), jolla voidaan rakentaa julkaisuputkia (engl. *deployment pipeline*), jotka siirtävät ohjelmistoja melko ongelmattomasti erillisten ympäristöjen välillä. Alusta on jo käytössä projektissa, mutta jatkuvan kiireellisemmän tietovarastokehitystyön vuoksi sen julkaisukäytäntöjä automatisoivia osia ei ole ehditty ottaa käyttöön.

Edellä mainitut ongelmat osoittavat, etteivät julkaisukäytännöt esimerkkiprojektissa ole aivan yksinkertaisia. Vastaavia ongelmia esiintyy varmasti myös muissa tietovarastoprojekteissa, joissa joudutaan kiinnittämään erityistä huomiota tietoturvaan. Niinpä tutkimuksen tapauksena oleva projekti toimii erinomaisena esimerkkinä luodun toimintamallin testaamiseen.

1.3 Rakenne ja tutkimuskysymykset

Tämän suunnitteluttutkimuksen tavoitteena on luoda toimintamalli, jonka avulla ohjelmistokehityksessä yleisesti käytettyjä jatkuvan toimituksen menetelmiä voitaisiin ottaa käyttöön tietovarastokehityksessä. Tarkoituksena on perehtyä kirjallisuuskatsauksen avulla jatkuvan toimituksen käyttöönottoon ohjelmistokehityksen alueella sekä vertailla ohjelmistokehitystä ja tietovarastokehitystä jatkuvan toimituksen kannalta oleellisilta osilta. Kirjallisuuskatsauksen perusteella jalostetaan Chenin (2017) esittelemät jatkuvan toimituksen käyttöönoton strategiat toimimaan tietovarastokehityksen ympäristössä. Näin luotua toimintamallia demonstroidaan KEHA-keskuksen tietovarastokehitysprojektissa käytännössä ottamalla jatkuva toimitus käyttöön toimintamallin suosittamilla tavoilla. Toimintamallin toimivuutta arvioidaan seuraamalla jatkuvan toimituksen käyttöönottoa sekä haastattelemalla projektin henkilöstöä. Haastattelujen ja seurannan perusteella toimintamallia jatkojalostetaan.

Jotta toimintamalli tietovarastoprojektin julkaisukäytänteiden käyttöönottamiseksi saataisiin tehtyä, tutkimuksessa pyritään vastaamaan seuraavaan tutkimuskysymykseen: *Miten jatkuva toimitus voidaan ottaa käyttöön tietovarastokehityksessä?* Vastaus varsinaiseen tutkimuskysymykseen saadaan vastaamalla alla oleviin apukysymyksiin.

- Mitä tarkoitetaan jatkuvalla toimituksella, jatkuvalla integraatiolla ja jatkuvalla julkaisulla?
- Miten näitä hyödynnetään ohjelmistokehityksessä?

- Miten jatkuva toimitus otetaan käyttöön ohjelmistokehityksessä?
- Miten tietovarastokehitys eroaa ohjelmistokehityksestä julkaisukäytänteiden automatisoinnin kannalta oleellisilta osilta?

Etukäteen voidaan todeta, että rajoitteita tutkimustulosten arvioinnissa tulevat aiheuttamaan esimerkkiprojektissa käytetyt työkalut ja tietovaraston mallintamiseen valittu DV-menetelmä ja DV-mallinnukseen valittu DV-Modeller. Myös DevOps-työkalun valinta on varmasti merkittävä tekijä projektin julkaisukäytänteiden automatisoinnissa, mutta tämä jää esimerkkiprojektissamme tarkastelun ulkopuolelle, sillä työkalu on valittu ennen tutkimuksen aloittamista.

Tutkielman loppuosa on rakennettu seuraavasti. Luvussa kaksi käsitellään tutkimuksen tutkimusasetelmaa, eli valittuja tutkimusmenetelmiä ja tutkimusprosessia. Luvussa kolme kuvataan tutkimuksen teoreettinen viitekehys ja käydään läpi kirjallisuuskatsauksen tulokset. Luvussa neljä kuvataan tutkimuksen toteutus ja luodun toimintamallin käyttöönottoprosessi sekä haastattelujen tulokset. Luvussa viisi kuvataan tulosten perusteella tehdyt johtopäätökset, arvioidaan tutkimuksen tuloksia sekä käsitellään tutkimuksen rajoitteita ja jatkotutkimuksen tarpeita. Luku kuusi sisältää tutkimuksen yhteenvedon.

2 TUTKIMUSASETELMA

Tässä luvussa tutustutaan valittuun tutkimusmenetelmään ja sen teoriaan sekä tutkimuksessa esimerkkinä käytettävään tapaukseen. Tavoitteena on kuvata lähtötilanne ja perustella, miksi kyseiset menetelmät on valittu ja näin antaa lukijalle työkalut arvostella tuloksien luotettavuutta.

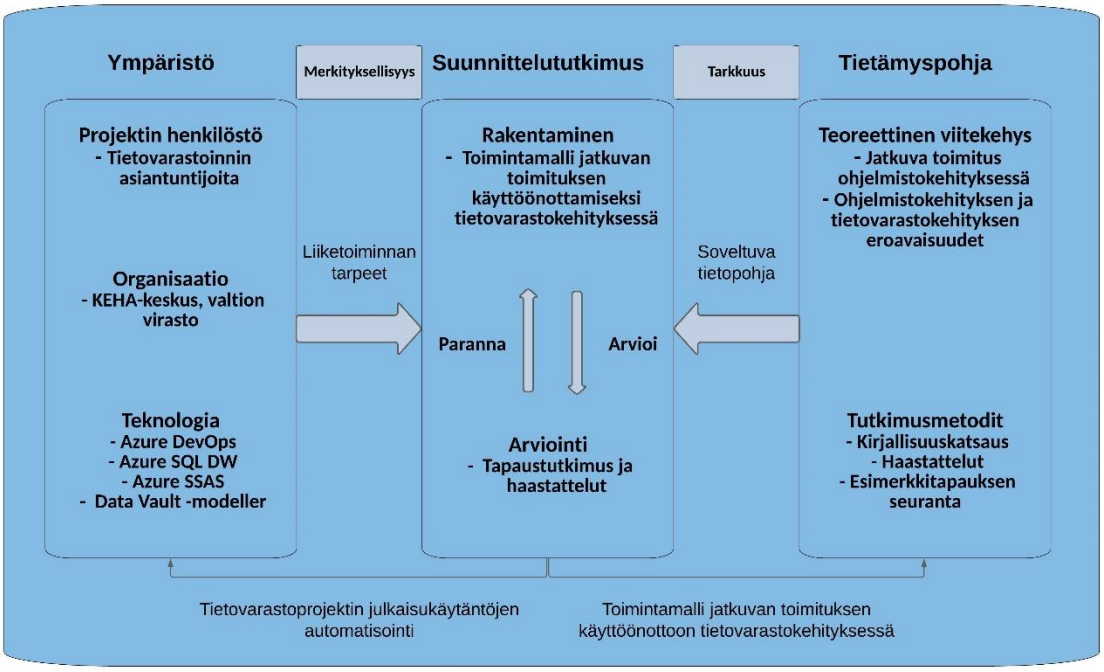
2.1 Tutkimusmenetelmä: suunnittelututkimus

Tieteellinen tutkimus voidaan määritellä laveasti toiminnaksi, joka pyrkii lisäämään ymmärrystä jostain ilmiöstä, sen olemisesta ja sen toiminnasta (Simon 1969, 111; Kuhn 1970). Hevner ym. (2004, 75) jakavat tietojärjestelmätieteen tutkimuksen karkeasti kahteen osaan: käyttäytymistieteellinen tutkimus (engl. *behavioral-science*) ja suunnittelututkimus (engl. *design-science*). Käyttäytymistieteellisellä tutkimuksella he tarkoittavat paradigmaa, joka pyrkii kehittämään ja vahvistamaan teorioita, jotka on tarkoitettu ihmisten tai organisaatioiden käyttäytymisen ennustamiseen. Heidän mukaansa suunnittelututkimuksessa tavoitteena taas on toiminnan ennustamisen sijaan pyrkiä laajentamaan ihmisten ja organisaatioiden kyvykkyyksiä luomalla uusia ja innovatiivisia artefakteja. Nämä artefaktit ovat konstrukteja, malleja, metodeita tai toteutuksia (Järvinen 2006, 26).

Tieteellisessä tutkimuksessa, erityisesti luonnontieteissä ja edellä mainitussa käyttäytymistieteellisessä tutkimuksessa, tieteenfilosofinen lähestymistapa on usein ollut jonkin olemassa olevan totuuden löytäminen. Suunnittelututkimuksessa sen sijaan tavoitteena ei ole löytää totuutta, vaan luoda hyötyä. Tämä hyödyn luominen tapahtuu luomalla uusi innovatiivinen artefakti. Totuus ja hyöty ovat kuitenkin siinä mielessä erottamattomat, että totuus usein näyttää, mistä hyöty on löydettävissä, mutta toisaalta hyöty näyttää, olemmeko löytäneet lopullisen totuuden. (Hevner ym. 2004, 79–80.) Tässä yhteydessä on tärkeää erottaa tavanomainen suunnittelutoiminta suunnittelututkimuksesta. Tärkein ero on uuden tiedon tuottaminen: jos suunnittelutoiminnassa yhdistellään parhaita toimintatapoja ja otetaan käyttöön olemassa olevia artefakteja, ei kyseessä ole tutkimus. Jotta voitaisiin puhua suunnittelututkimuksesta, täytyy suunnittelutoiminnan yhteydessä luoda täysin uutta tietoa. (Hevner ja Chatterjee 2010, 7.)

Tämän tutkimuksen tavoitteena on täyttää akateemisessa tutkimuksessa ja käytännön elämässä oleva aukko, jossa jatkuvan toimituksen menetelmiä otettaisiin käyttöön tietovarastokehityksessä. Toisin sanoen, tarkoituksena on luoda uusi artefakti ja arvioida sen toimivuutta, mikä istuu suunnittelutieteen määritelmään (Järvinen 2006, 26). Näin ollen,

suunnittelututkimus on luontainen valinta tutkimusmenetelmäksi. Tutkimusasetelman kuvaamiseksi käytetään Hevner ym. (2004) esittelemää viitekehystä tietojärjestelmätieteen tutkimuksen tekemiseen. Tutkimusasetelma on esitetty kuviossa 1.

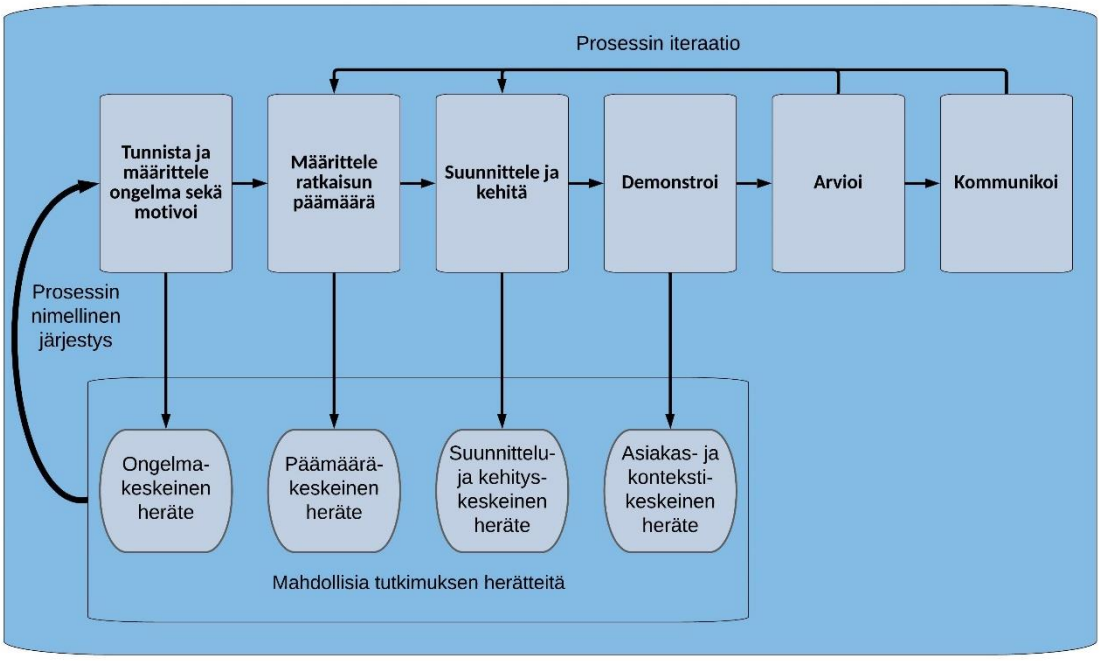


Kuvio 1. Tutkimusasetelma (mukaillen Hevner ym. 2004).

Tutkimuksen ympäristönä toimii KEHA-keskuksen tietovarastointiprojekti, joka kuvataan tarkemmin luvussa 2.4. Tutkimuksen merkityksellisyys syntyy KEHA-keskuksen liiketoiminnallisista tarpeista, eli tietovarastokehityksen julkaisukäytäntöjen parantamisesta. Tutkimuksen tietämyspohja muodostuu olemassa olevasta kirjallisuudesta, jonka avulla pyritään tunnistamaan jatkuvan toimituksen kannalta olennaisia ohjelmistokehityksen osia ja vertailemaan niitä tietovarastokehityksen vastaaviin osiin. Näin tutkimukselle luodaan teoreettinen viitekehys. Kirjallisuuskatsauksen avulla kerättyä tietoa syvennetään tapaustutkimuksen avulla seuraamalla tosielämän tietovarastointiprojektia. Näin luodun tietopohjan avulla luodaan toimintamalli jatkuvan toimituksen käyttöönottamiseksi tietovarastokehityksessä. Mallia arvioidaan seuraamalla sen käyttämistä käytännössä. Tätä tietoa syvennetään haastattelemalla projektin henkilöstöä, jotka ovat tietovarastoinnin asiantuntijoita. Haastattelujen ja seurannan tavoitteena on arvioida ja parantaa toimintamallia edelleen.

2.2 Tutkimusprosessi

Tutkimusprosessin pohjana käytetään Peffers ym. (2007) esittelemää metodologiaa tietojärjestelmätieteiden suunnittelututkimuksen laatimiseksi. Metodologian prosessikaavio on esitelty kuviossa 2.



Kuvio 2. Suunnittelututkimuksen prosessikaavio (mukaillen Peffers ym. 2007, 54).

Tarve suunnittelututkimuksen tekemiseksi voi syntyä neljällä tapaa (ks. kuvio 2). Herätteen jälkeen ensimmäinen toiminto on ongelman tunnistaminen, määritteleminen ja motivointi. Ongelman määrittely on tärkeää, koska ratkaisu luodaan määrittelyn avulla. Jotta ratkaisu voisi olla onnistunut, täytyy myös ongelman määrittelyn kuvata hyvin ongelman moninainen luonne. (Peffers ym. 2007, 52–54).

Toinen toiminto on ratkaisun päämäärän määritteleminen. Tavoitteena on siis päätellä mitä on mahdollista saavuttaa ja mikä riittää ratkaisemaan ongelman. Jos ongelmaan on jo olemassa jokin ratkaisu, uuden ratkaisun täytyy luonnollisesti olla tehokkaampi. Ratkaisun päämäärät täytyy päätellä rationaalisesti ongelman määrittelyn avulla. Jotta tässä onnistutaan, täytyy ongelma, sekä sen mahdolliset ratkaisut ja niiden tehokkuudet tuntea hyvin. (Peffers ym. 2007, 54).

Kolmantena toimintona on artefaktin luominen. Tähän kuuluu artefaktin suunnittelu, haluttujen toiminnallisuuksien ja arkkitehtuurin suunnittelu sekä artefaktin varsinainen

luominen. Tähän vaaditaan asiaan liittyvän teorian tuntemusta, jotta päämäärät saadaan muotoiltua ratkaisuksi. (Peffers ym. 2007, 54).

Seuraavaksi luodun artefaktin toimintaa demonstroidaan käytännössä ratkaisemalla määritelty ongelma. Mahdollisia keinoja demonstrointiin on koeasetelma, simulointi tai esimerkiksi tapaustutkimus. Demonstroinnin avulla voidaan suorittaa viides toiminto, artefaktin arviointi. Tässä toiminnossa toteutuneita tuloksia verrataan asetettuihin päämääriin. Arvioinnin avulla artefaktia voidaan mahdollisesti jatkokehittää. Arvioinnissa voidaan huomata, että päämäärät oli asetettu väärin, tai että artefakti ei toiminut halutulla tavalla. Iteraatioita voidaan toteuttaa useita. Kun artefakti on iteraatioiden avulla saatu haluttuun tilaan, viimeisenä toimintona saadut tulokset kommunikoidaan eteenpäin. Kommunikointi voi tapahtua monella tavalla. (Peffers ym. 2007, 54).

2.3 Tiedonkeruumenetelmät

Tutkimuksen tiedonkeruumenetelmiä ovat kirjallisuuskatsaus, haastattelut ja esimerkkiprojektin seuranta sekä sen dokumentointiin tutustuminen. Osana projektin dokumentointiin tutustumista perehdytään myös valittujen työkalujen dokumentaatioihin.

2.3.1 Kirjallisuuskatsaus

Kirjallisuuskatsaus suoritetaan käyttäen Googlen Scholar- ja Volter-tietokantoja. Volter-tietokanta on Turun yliopiston palvelu ja se kattaa 485 tietokantaa, joihin kuuluu muun muassa ACM, IEEE, SAGE, ScienceDirect ja Wiley Online Library (Utuguides). Jatkuvaa toimitusta koskevissa hauissa halutaan varmistaa tiedon tuoreus ja haut pyritään rajaamaan alle viisi vuotta vanhoihin teoksiin. Tietovarastointia ja jatkuvaa integraatiota koskevissa hauissa vastaavaa rajausta ei tehdä, sillä monet tärkeimmistä niitä koskevista oivalluksista on tehty jo aiemmin. Jatkuvaa toimitusta koskevien artikkelien etsimisessä käytetään hakusanoja ”Continuous Delivery”, ”Continuous Integration” ja “Continuous Deployment” sekä näiden eri yhdistelmiä sanojen ”Implementation” ja ”Adoption” kanssa. Tietovarastointia koskevissa artikkeleissa hakusanoina käytetään “Data Warehousing”, “Enterprise Data Warehouse” ja ”Development Practices”. Tietovarastointiin ja jatkuvaan toimitukseen liittyviä hakutermejä ristiin käyttämällä ei löytynyt Puontia ym. (2018) lukuun ottamatta varteenotettavia artikkeleita, mikä korostaa tutkimusaukon olemassaoloa.

2.3.2 Haastattelut

Haastattelut ovat yleisesti käytetty tapa kerätä kvalitatiivista tietoa. Haastattelut voidaan jakaa karkeasti rakenteellisiin (engl. *structured*) ja vapaamuotoisiin (engl. *unstructured*) haastatteluihin. Vapaamuotoisissa haastatteluissa tavoitteena on saada mahdollisimman paljon tietoa laajasti määritellystä aiheesta, kun haastattelija ei tunne käsiteltävää aihetta riittävän hyvin voidakseen rajata haastattelun rakennetta. Tarkoituksena on siis antaa haastateltavan määrätä haastattelun rakenne, jotta haastattelija ei väärän rakenteen takia tulisi rajanneeksi tärkeitä osa-alueita haastattelujen ulkopuolelle. Rakenteellisissa haastatteluissa tutkijalla taas on selkeä kuva siitä, minkälaista tietoa haastattelulla pyritään saamaan, joten kysymykset voivat olla melko pikkutarkkoja. (Seaman 1999, 562.)

Tässä tutkimuksessa haastattelut suoritetaan kahdessa vaiheessa. Ensimmäisten haastattelujen tarkoituksena on syventää ymmärrystä esimerkkiprojektista ja sen työkaluista. Tässä vaiheessa haastattelujen tavoitteena on auttaa yleisen kuvan luomista projektista, sen työkaluista, ympäristöstä ja vallitsevasta tilanteesta. Haastattelujen avulla pyritään myös ymmärtämään tarkemmin ratkaistavana olevaa ongelmaa sekä ratkaisun päämäärää ja tilaa, johon projektissa tämän tutkimuksen tuloksien avulla pyritään. Haastattelujen tuloksia ei ole tarpeen jäsennellä selkeästi, vaan ne syventävät dokumentaatiosta ja muusta seurannasta saatua tietoa. Tällaiseen tiedon keräämiseen parhaiten sopii vapaamuotoinen ja keskustelunomainen haastattelutekniikka, koska haastattelun rakenteen ei haluta piilottavan tärkeitä, haastattelijalle ennestään tuntemattomia tietoja (Seaman 1999, 562; Turner 2010, 755).

Tutkimuksen toisella haastattelukerralla tavoitteena on arvioida luotua toimintamallia. Koska haastateltavat ovat tietovarastokehityksen asiantuntijoita, haastatteluista pyritään saamaan myös täysin uutta, toimintamallia kehittävää tietoa. Jotta toimintamallia voitaisiin arvioida jäsennellysti, täytyy haastatteluissa käyttää rakenteellista haastattelumenetelmää. Toisaalta haastatteluista halutaan saada myös täysin uutta tietoa, joten haastattelussa täytyy jättää tilaa myös vapaammalle keskustelulle. Tämän takia haastattelumenetelmäksi valikoitui standardoitu avoimien kysymyksien menetelmä. Tässä menetelmässä haastateltaville esitetään samat avoimet kysymykset, joihin heidän annetaan vastata vapaasti (Turner 2010, 756). Samojen kysymyksien esittäminen takaa, että kaikilta haastateltavilta saadaan tietoa, jonka avulla toimintamallia voidaan arvioida. Haastateltavia kannustetaan myös vastaamaan vapaasti ja laajasti, eikä haastattelija rajoita vastauksien pituutta. Haastattelujen lopussa haastateltaville annetaan myös mahdollisuus puhua

aiheesta vapaasti ja heiltä kysytään kysymyksiä aiheesta laajemmin jättäen tilaa pohdinnalle. Toisen haastattelukierroksen kysymykset on esitelty liitteessä 1. Haastattelukysymykset on luotu esitetyn toimintamallin perusteella siten, että vastausten perusteella voitaisiin mahdollisimman tehokkaasti arvostella toimintamallin tehokkuutta.

2.3.3 Seuranta ja dokumentointiin tutustuminen

Kolmantena tiedonkeruumenetelmänä tutkimuksessa on esimerkkitapauksen seuranta sekä projektin ja sen työkalujen dokumentaatioon tutustuminen. Projektin seurannan tavoitteena on helpottaa kokonaiskuvan luomista projektista, sen ympäristöstä, henkilöstöstä ja käytetyistä teknologioista (Stake 1995, 60). Projektin käytännön toimintojen seurannalla voidaan saada tietoja, jotka muuten jäisivät huomaamatta. Ohjelmistokehittäjät paljastavat ajatusprosessinsa luonnollisimmin kommunikoidessaan kollegoidensa kanssa. (Seaman 1999, 558.) Tämän takia seurantaa suoritetaan osallistumalla projektin työn suunnittelu- ja retrospektiivisiin palavereihin ja tarkastelemalla projektin henkilöstön työskentelytapoja. On myös tärkeää, etteivät tarkkailun kohteet ajattele jatkuvasti olevansa tarkkailtavina (Seaman 1999, 558). Tässä auttaa se, että tarkkailija toimii tutkimuksen ajan yhtenä kehittäjistä kehitystiimissä.

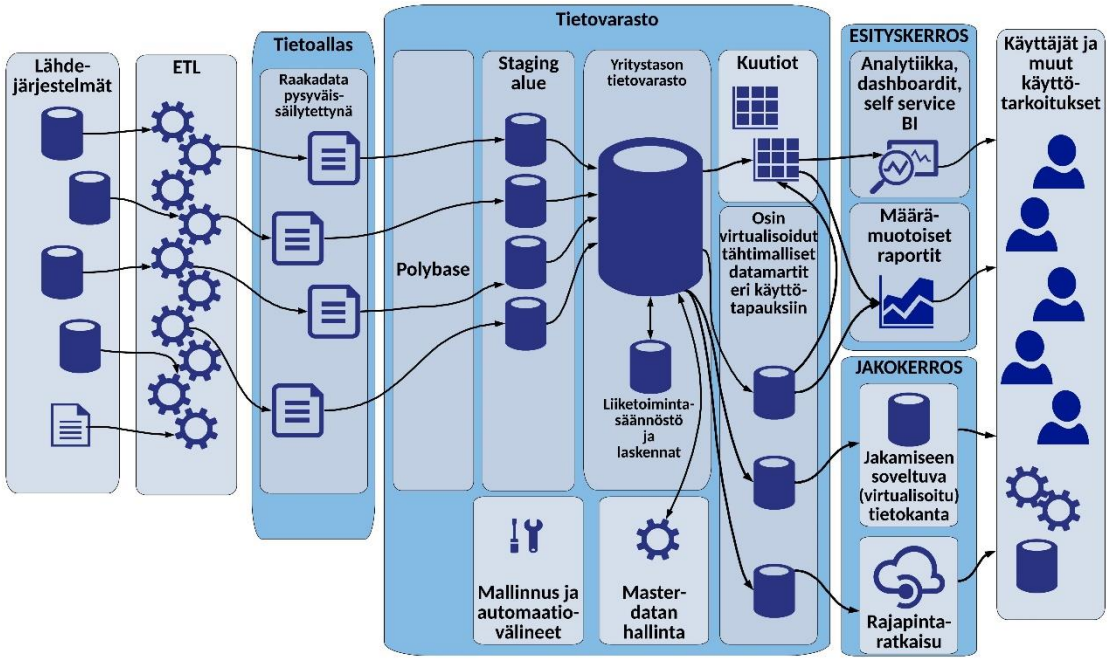
Dokumentointiin tutustutaan tarpeellisilta osin, jotta projektista saadaan kattava kuva. Projektista on laadittu kattava dokumentaatio, joka sisältää projektin tavoitteet, käytetyt teknologiat ja rakennettavan arkkitehtuurin (ks. kuvio 3). Myös käytettyjen teknologioiden dokumentaatioon tutustutaan, jotta luotavan toimintamallin demonstrointiin tarvittava kehitystyö saataisiin tehtyä.

2.4 Tapaus: KEHA-keskuksen tietovarastointiprojekti

Suunnittelututkimuksessa luotava artefakti sijoitetaan aina johonkin ympäristöön, joten tutkijan tulee tuntea tämä ympäristö (Järvinen 2006, 26). Tässä tutkimuksessa esimerkkiprojektina on KEHA-keskuksen tietovarastointiprojekti. KEHA-keskus on Suomen valtion virasto, joka vastaa ELY-keskusten ja TE-toimistojen kehittämis- ja hallinnointipalveluiden tuottamisesta (ks. KEHA-keskus). Esimerkkiprojekti on KEHA-keskuksen tietovarastoinnin kehitysprojekti, jonka tavoitteena on luoda yksi yhtenäinen tietovarasto, johon useiden eri järjestelmien tiedot varastoidaan. Osana projektia on myös tarkoitus luoda raportointirajapinta tähän tietovarastoon, mutta raportoinnin tarkastelu jätetään tutkimuksen ulkopuolelle.

Projektin henkilöstö koostuu kahden eri organisaation työntekijöistä. Projektin johtaminen on jaettu kahteen osaan: kokonaisuuden johtaminen sekä teknisen puolen johtaminen. Molempiin tehtäviin osoitetut henkilöt ovat KEHA-keskuksen palkkalistoilla. Lisäksi IT-infrastruktuurista ja järjestelmien administraatiosta vastaava henkilöstö on KEHA-keskuksen työvoimaa. Varsinaiseen tietovarastokehitykseen ja tietovaraston ylläpitämiseen on palkattu KEHA-keskuksen ulkopuolelta neljä työntekijää. Yhdessä teknisen johtajan kanssa nämä neljä vastaavat tietovarastokehityksestä ja tietovaraston ylläpidosta. Toisin sanoen tämän tutkimuksen kannalta projektissa on vain viisi mielenkiinnon kohteena olevaa henkilöä, joista yksi on myös tutkimuksen tekijä. Haastateltavien määrä rajautuu siis neljään.

Esimerkkiprojektin kohteena olevan tietovaraston arkkitehtuuri on esitetty kuviossa 3. Lähdejärjestelmät, ETL-prosessi, tietoallas, esitys- ja jakokerrokset sekä käyttäjärajapinta on jätetty kuvaan helpottamaan kokonaiskuvan muodostamista, vaikka tutkimuksen kannalta mielenkiinnon kohteena on vain kuvion keskellä oleva tietovarasto-osuus. Tietoaltaasta mainittakoon se, että kyseessä on pilvipohjainen ja skaalautuva strukturoimattoman tiedon tallennuspaikka Azure Storage, jonne data tallennetaan tekstitiedostoina, eli niin kutsuttuina blobeina. Tietovarasto itsessään on Azure Synapse Analytics MPP-klusteri. Tiedot siirretään tietoaltaasta tietovarastoon käyttäen Polybase-määrittelyjä, jotka mahdollistavat taulukkomuotoon tallennettujen tekstitiedostojen (esim. CSV- tai Parquet-tiedosto) manipuloinnin ja analysoinnin SQL-kielen syntaksilla. Tiedon latauksia ja tietovaraston kehitystä varten arkkitehtuuriin on lisätty virtuaalikone, johon on asennettu Windows Server -käyttöjärjestelmä sekä SQL Server -instanssi IaaS-palveluna. Näin käyttöön on saatu SQL Server Integration Services, jolla tietovarastolataukset voidaan automatisoida tehokkaasti ja luotettavasti. Lisäksi arkkitehtuurin on lisätty analyysipalvelin SQL Server Analysis Services (SSAS), jolla datasta tehdään raportoinnin pohjaksi dimensionaalisia malleja ja johon nämä mallit tallennetaan multidimensionaalisina- tai tabular-kantoina, joista tabular-kannat ovat esimerkkiprojektissa suositeltuja. Näitä dimensionaalisia malleja kutsutaan kuutioiksi tai SSAS-kuutioiksi.



Kuvio 3. Esimerkkiprojektin tietovarastoarkkitehtuuri.

Yritystason tietovarasto sisältää siis kaikki tietovarastoon tuodut tiedot historioituina ja DV-mallinnettuina. DV-mallinnus tapahtuu kyselyjen avulla hakemalla tiedot staging-alueelta, jossa lähdejärjestelmistä vedetyt tiedot säilytetään siinä muodossa, kun ne on lähdejärjestelmistä haettu. Jotta tätä tietoa voitaisiin käyttää helpommin, tiedot johdetaan yritystason tietovarastosta edelleen datamartteihin (engl. *data mart*). Datamartit toimivat tietovaraston ”kuluttajarajapintana” ja kukin datamart sisältää tietoa vain yhtä tarkoitusta varten. Datamartit voivat olla ”oikeita” tauluja, tai ne voidaan virtualisoida käyttäen näkymiä yritystason tietovarastoon. (Moody & Kortink 2000, 2.) Datamartit ovat siis eräänlaisia tietovaraston osajoukkoja. Datamarteista käytetään myös Data Vault -mallinnuksen yhteyksissä nimitystä tietomart (engl. *information mart*) kuvastaen sitä, että ne usein sisältävät jo liiketoimintalogiikkaa. Liiketoimintalogiikan ajatellaan muuttavan niiden sisällön datasta tiedoksi. (Lindstedt & Olschimke 2015, 27.) Tässä yhteydessä tyydymme kuitenkin käyttämään termiä datamart. Esimerkkiprojektissa datamartit ovat osin virtualisoituja ja ne on mallinnettu käyttäen tähtimallinnusta, jotta raportointi olisi helpompaa.

Esimerkkiprojektin työskentelyssä hyödynnetään Scrum-mallia (ks. Schwaber & Sutherland 2011), johon liittyviä käytännön toimia ylläpidetään Azure DevOps -alustalla. Työskentely on jaettu kolmen viikon sprintteihin. Projektissa käytetään myös Git-versionhallintajärjestelmää. Tietovarasto-objekteille ja datamarteille on käytössä yksi Git-putki, jossa on haarat kehitysympäristöön, testausympäristöön ja tuotantoympäristöön.

Kehittäjät luovat kehityshaarasta itselleen oman haaran tehdäkseen siihen muutoksia. Kun muutokset ovat valmiita ja testattuja, he yhdistävät oman haaransa kehityshaaraan. Näin kehityshaarassa on aina julkaisukelpoista koodia. Kun tehdyn muutoksen toimivuus on testattu kehitysympäristössä, voidaan muutokset viedä testausympäristöön ja sieltä edelleen testien jälkeen tuotantoympäristöön. Jokaiselle kuutiolle on käytössä oma Git-putkensa, joissa toimitaan samalla tavalla.

3 TEOREETTINEN VIITEKEHYS

Tämän luvun tavoitteena on kuvata tutkimuksen teoreettinen viitekehys. Luvussa esitellään käytetyt termit sekä Chenin (2017) strategiat jatkuvan toimituksen käyttöönotolle ohjelmistokehityksessä. Luvussa käsitellään lisäksi tietovarastokehitystä jatkuvan toimituksen kannalta oleellisilta osin, jotta strategiat voidaan johtaa toimintamalliksi jatkuvan toimituksen käyttöönotolle tietovarastokehityksen kontekstissa.

3.1 Jatkuva toimitus

Tässä alaluvussa käsittelemme kirjallisuuskatsauksessa jatkuvasta toimituksesta tehtyjä päätelmiä. Tarkoituksena on määritellä keskeisimmät termit ja kuvata aiheeseen liittyviä tärkeimpiä osia. Lopuksi esitellään, millaisia asioita jatkuvan toimituksen käyttöönotossa tulee huomioida.

3.1.1 Jatkuva integraatio, toimitus ja julkaisu

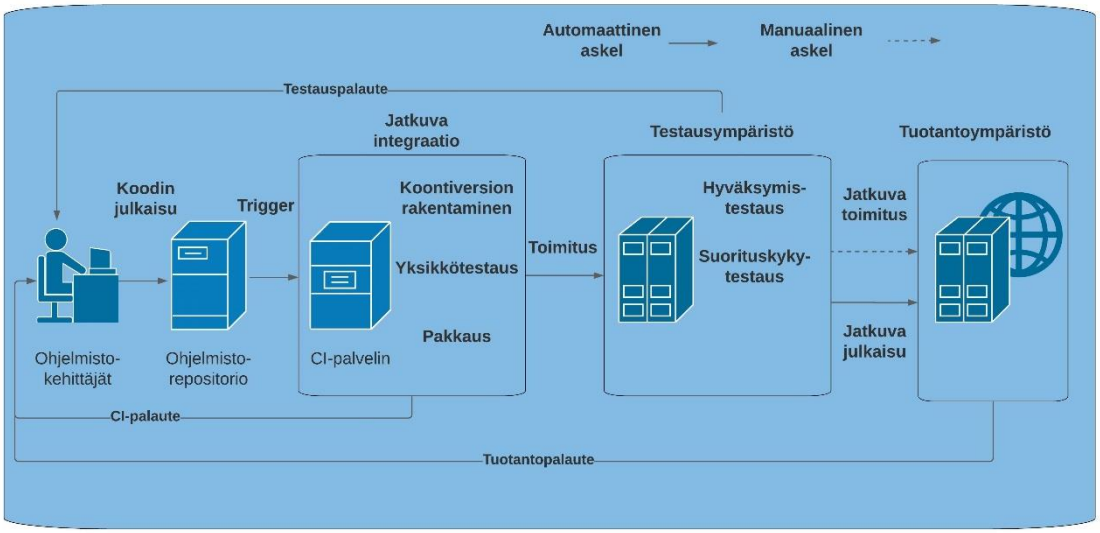
Jatkuva integraatio (engl. *continuous integration*), jatkuva toimitus ja jatkuva julkaisu (engl. *continuous deployment*) ovat ohjelmistokehityksen käytäntöjä, jotka mahdollistavat organisaatioille uusien toiminnallisuuksien julkaisun tuotantoon nopeasti ja luotettavasti (Shahin ym. 2017, 1061) ja ne ovatkin DevOpsin tärkeimpiä menetelmiä (Bass ym. 2015).

Jatkuvan integraatiota on hyödynnetty ohjelmistokehityksessä melko pitkään (Meyer 2014, 14) ja se on terminä melko vakiintunut. Sillä tarkoitetaan toimintatapaa, jossa ohjelmistokehittäjän tekemät muutokset integroidaan jatkuvasti kehityshaarasta päähaaraan. Tämä edellyttää koontiversioiden rakentamisen ja testaamisen automatisointia. (Fitzgerald & Stol 2017, 183.) Jatkuvan integraation työkalut siis yhdistävät kaikkien ohjelmistokehittäjien luomat koodit yhteen, rakentavat koontiversion ja tarkistavat automaattisesti, ettei virheitä päässyt syntymään (Fowler 2006, 1; Ebert ym. 2016, 96). Jatkuva integraatio mahdollistaa ohjelmistovirheiden huomaamisen aikaisin, helposti ja automaattisesti ilman manuaalista työtä (Duvall ym. 2007). Jatkuvan integraation käyttäminen ohjelmistokehitysprojekteissa on jo hyvin yleistä, ainakin Suomessa (Mäkinen ym. 2016, 190). Jatkuvan integraation hyödyiksi on lueteltu muun muassa projektin epäonnistumisen riskin pieneneminen (Fowler 2006) ja koodissa olevien virheiden nopeampi korjaaminen (Fowler 2006; Duvall ym. 2007). Jatkuvan integraation tehokkuuteen vaikuttaa merkittävästi se, kuinka hyvin koodia onnistutaan testaamaan. Jos testaus epäonnistuu,

jatkuvan integraation hyödyt eivät toteudu. Jatkuvaan integraatioon kuuluu myös melko laajasti CI-palvelimien käyttö. CI-palvelimella voidaan helposti määritellä ja suorittaa jatkuvaan integraatioon liittyvät prosessit. CI-palvelimien käyttäminen ei kuitenkaan ole välttämätöntä, ja on tärkeää huomata, että jatkuva integraatio ei ole vain jonkin ohjelmiston asentamista, vaan se vaatii laajempaa omistautumista ja toimintatapojen omaksumista. (Fowler 2006.)

Jatkuvassa integraatiossa ohjelmistokehittäjät siis lähettävät tekemänsä muutokset yhteiseen versionhallintarepositorioon (engl. *version control repository*), mikä käynnistää automaattisen koontiversion rakentamisen ja testaukset (Fowler 2006). Jatkuva toimitus taas sisältää jatkuvan integraation lisäksi automaattisia askeleita, jotka valmistelevat koodista täysin julkaisuvalmiin version. Jatkuvaan toimitukseen kuuluu siis myös ajotiedoston (engl. *executable*) koostaminen, sen työntäminen julkaisuympäristön kaltaiseen testiympäristöön ja sen siellä testaaminen. (Fowler 2013.) Tämä vaatii usein IaC:n käyttöä (Rahman ym. 2019, 75). IaC on toimintamalli, jossa virtuaalisia ympäristöjä luodaan koodin avulla, mikä mahdollistaa identtisten ympäristöjen luomisen vaivattomasti, nopeasti ja luotettavasti. Jatkuvassa toimituksessa IaC:tä käytetään testiympäristöjen luomiseen, jotta ohjelmistoa voidaan testata tuotantoympäristön kanssa identtisessä ympäristössä. (Rahman ym. 2019.)

Shahin ym. (2019, 1062) tekemän selvityksen perusteella jokseenkin yleinen käsitys jatkuvan toimituksen ja jatkuvan julkaisun eroista on tuotantoympäristöön tapahtuvan julkaisun automatisoinnin aste. Jatkuvassa toimituksessa julkaisu testiympäristöstä tuotantoympäristöön ei tapahdu ilman erillistä manuaalista toimintoa. Jatkuvassa julkaisussa sen sijaan myös julkaisu tuotantoympäristöön tapahtuu automaattisesti, kun ohjelmistokehittäjä integroi koodinsa ohjelmiston versionhallintarepositorioon. Jatkuva toimitus tarkoittaa siis sitä, että ohjelmisto pystytään julkaisemaan tuotantoon usein ja luotettavasti (Chen 2015) ottamatta kantaa siihen, kuinka usein julkaisu tapahtuu (Neely & Stolt 2013). Jatkuvassa julkaisussa taas jokainen koontiversio julkaistaan tuotantoon. Jatkuva toimitus siis mahdollistaa jatkuvan julkaisun. (Humble 2018, 34.) Kuviossa 4 on esitelty Shahin ym. (2019, 1063) näkemystä jatkuvan integraation, toimituksen ja julkaisun suhteista, jossa jatkuvan toimituksen ja jatkuvan julkaisun ainoa ero on se, tapahtuuko koontiversion julkaisu tuotantoon aina automaattisesti, vai tarvitaanko julkaisuun manuaalinen heräte. Vastaavaan lopputulemaan jatkuvan julkaisun ja toimituksen erosta on päätynyt myös Fowler (2013) ja Humble (2018, 34).



Kuvio 4. Jatkuvan integraation, toimituksen ja julkaisun suhteet (mukaillen Shahin ym. 2019, 1063).

Vaikka kuvio 4 antaa sellaisen kuvan, että jatkuva toimitus sisältää vain viimeisen askeleen, julkaisun testiympäristöstä tuotantoympäristöön, Shahin ym. (2019) selvästi tarkoittavat jatkuvan toimituksen sisältävän koko kuvion esittämän kokonaisuuden. Vastaavanlaiseen määritelmään ovat päätyneet myös Fowler (2013), Laukkanen ym. (2017, 56) ja Humble (2018, 34). Näin ollen tässä tutkimuksessa jatkuvan toimitus määritellään ohjelmistokehityksen toimintatavoiksi, jotka mahdollistavat sen, että jokainen koontiversio on julkaistavissa tuotantoympäristöön. Jatkuvan julkaisun ja jatkuvan toimituksen ero on tässä tutkimuksessa triviaali, joten jatkuvan julkaisun käsittely erikseen ei ole mielekästä. Näin ollen todetaan, että tässä tutkimuksessa tehdyt päätelmät jatkuvasta toimituksesta koskevat myös jatkuvaa julkaisua ja jatkossa puhutaan vain jatkuvasta toimituksesta.

3.1.2 Jatkuva toimitus ohjelmistokehityksessä

Jatkuva toimitus on ohjelmistokehityksen toimintatapa, jossa kehitystiimit luovat arvoa nopeissa sykleissä ja varmistavat, että ohjelmisto voidaan julkaista tuotantoon turvallisesti, kestävästi ja milloin vain (Chen 2015, 50; Humble 2018, 34) automatisoimalla koontiversioiden rakentaminen, testaukset ja julkaisu (Humble 2018). Lyhyemmät syklit ja tehostunut toiminta mahdollistavat tuotteiden julkaisun markkinoille nopeammin ja paremmin kysyntään vastaavasti ilman, että julkaisujen luotettavuus kärsii. Tämä johtaa luonnollisesti parempaan asiakastyytyväisyyteen. (Chen 2015, 52; Leppänen ym. 2015, 67; Humble 2018, 34.) Jatkuvan toimituksen toimivuudelle on kuitenkin useita vastaväitteitä (Humble 2018, 34). Seuraavaksi käsittelemme jatkuvaan toimitukseen liittyviä

erityispiirteitä näiden vasta-argumenttien avulla, sillä ne tarjoavat meille oivan väylän ymmärtää paremmin jatkuvan toimituksen ominaisuuksia ja niitä ohjelmistokehityksen osia, jotka ovat jatkuvan toimituksen kannalta olennaisia.

Humble (2018, 34) luettelee yleisimmiksi jatkuvan toimituksen käytön vasta-argumenteiksi seuraavat:

- Jatkuva toimitus ei sovi tiukasti säädeltyihin ympäristöihin
- Jatkuva toimitus sopii vain verkkosivujen kehitykseen
- Jatkuvan toimituksen menetelmiä ei voida soveltaa vanhojen järjestelmien (engl. *legacy systems,* ks. esim. Bennet 1995) kanssa
- Jatkuva toimitus vaatii laajempaa osaamista, kuin usein on tarjolla

Ajatus siitä, että jatkuva toimitus ei sopisi tiukasti säädeltyihin ympäristöihin on suoraan vastoin jatkuvan toimituksen perusperiaatetta, vähentää julkaisuihin liittyviä riskejä. Jatkuva toimitus itseasiassa varmistaa suuremman varmuuden jatkuvalla integraatiolla, automaattisilla testeillä ja julkaisuputkien käytöllä. (Humble 2018, 35.) Jatkuva toimitus toisin sanoen tarjoaa uudenlaisen riskienhallintastrategian, joka on vähintään yhtä tehokas kuin perinteisemmät strategiat ja tarjoaa lisäksi mahdollisuuden suurempaan julkaisutiheyteen (Humble & Molesky 2011, 10–11; Humble 2018, 35).

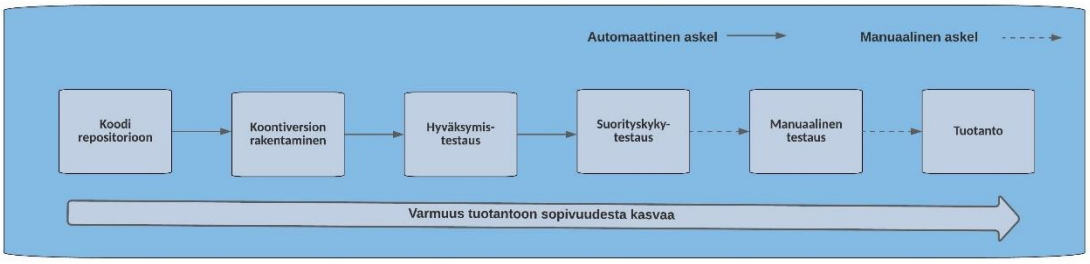
Väite siitä, että jatkuva toimitus soveltuu vain verkkosivujen kehitykseen, juontaa juurensa todennäköisesti siitä, että tarvittavien teknologioiden, kuten jatkuvan integraation, automaattisten testien ja julkaisuputkien, käyttäminen verkkokehityksessä on helpompaa. Humble (2018, 35) kuitenkin osoittaa, että jatkuvaa toimitusta voidaan käyttää missä tahansa ympäristössä. Hän kuitenkin myös toteaa, että jossain tapauksissa, erityisesti monimutkaisissa kokonaisuuksissa, tarvittava investointi voi olla huomattavan suuri.

Jatkuvaa toimitusta voidaan käyttää missä tahansa ympäristössä, jossa ohjelmistojen tai järjestelmien uskotaan muuttuvan merkittävästi sen elinkaaren aikana (Humble 2018, 35). Näin ollen, vanhojen järjestelmien mainitseminen jatkuvan toimituksen yhteydessä on hivenen outoa, sillä näiden järjestelmien kehityksestä on usein luovuttu, ja ne toimivat vain lähdejärjestelminä muille, uudemmille järjestelmille. Humble (2018, 36–37) kuitenkin osoittaa, että jatkuvaa toimitusta voidaan hyödyntää myös vanhojen järjestelmien osalta projekteissa, joissa vanhojen järjestelmien ympärille luodaan laajempi kokonaisuus. Jatkuvaa toimitusta voidaan siis hyödyntää vanhojen järjestelmien ympäristössä myös sille ominaisilla osa-alueilla, eivätkä sen hyödyt rajoitu vain puhtaaseen

ohjelmistokehitykseen. Jatkuvan toimituksen menetelmät toimivat myös laajemmin kokonaisvaltaisten järjestelmäratkaisujen kehittämisessä, kun kehityksen kohteena on mahdollisesti useampia järjestelmiä sekä näiden ohjelmistojen käyttöön liittyvät prosesseja ja käytänteitä.

Jatkuva toimitus on monimutkainen kokonaisuus ja se vaatii merkittäviä investointeja prosessi- ja teknologiakehitykseen (Humble 2018, 37). Väite siitä, että työntekijöillä ei ole riittävästi osaamista näiden investointien tekemiseksi, siirtää vastuun kuitenkin Humblen (2018, 37) mukaan turhaan työntekijälle, vaikka osaamisen puute johtuu useimmin puutteellisesta johtamisesta. Jatkuva toimitus kulminoituu kaizen-periaatteen (ks. esim. Manos 2007) mukaiseen jatkuvaan kehittämiseen, ja se kattaa myös henkilöstön kehittämisen, jolloin johdon tehtäväksi jää toimia tämän kehityksen mahdollistajana. Tämä vaatii organisaatiokulttuurin muovaamista sellaiseksi, jossa työntekijät haluavat ja saavat kehittyä.

Jatkuvan toimituksen käyttäminen siis asettaa tiettyjä vaatimuksia kehitystiimin toimintatavoille ja käytetyille teknologioille (Chen 2017, 1062; Humble 2018). Ehkä tärkeimpänä yksittäisenä teknologisena vaatimuksena on julkaisuputkien käyttöönotto (Chen 2015, 51; Gill ym. 2018, 130–131; Puonti ym. 2018, 258). Julkaisuputki on teknologinen toteutus, joka käynnistyy ohjelmistokehittäjän kirjoittaman koodin siirtämisestä (engl. *code commit*) yhteiseen versionhallintarepositorioon. Julkaisuputki rakentaa repositoriosta automaattisesti koontiversion ja suorittaa sille ohjelmoidut testit, kuten hyväksymis- ja suoritustestit. Näiden jälkeen koodia voidaan vielä testata manuaalisesti tai se voidaan julkaista automaattisesti tuotantoon, kuten jatkuvassa julkaisussa on tapana (ks. luku 3.1.1). Kuviossa 5 on esitelty esimerkki julkaisuputkesta.



Kuvio 5. Esimerkki julkaisuputkesta (mukaillen Chen 2015, 51).

Vertaamalla kuvion 5 julkaisuputkea ja kuviossa 4 esiteltyjä jatkuvan integraation, toimituksen ja julkaisun suhteita voidaan huomata niiden välillä monia yhteneväisyyksiä. Nämä yhteneväisyydet korostavat julkaisuputkien käytön tärkeyttä jatkuvassa

toimituksessa ja voidaankin todeta, että teknologisesti jatkuva toimitus kulminoituu julkaisuputkiin.

Jotta julkaisuputken ensimmäinen askel saadaan automatisoitua, täytyy kehitystiimillä olla käytössään versionhallintajärjestelmä (Humble & Farley 2010, 32–33; Chen 2015, 51; Mäkinen ym. 2016, 185; Puonti ym. 2018, 258). Ilman versionhallintajärjestelmän sisäistä yhteistä versionhallintarepositoriota julkaisuputki ei voi suorittaa koontiversion rakentamista. Versionhallintajärjestelmiin liittyy myös paljon muita hyötyjä ja niiden käyttäminen onkin ollut yleistä ohjelmistokehitysprojekteissa jo pidempään (Fowler 2006, 3; Mäkinen ym. 2016, 190). Humble (2018, 34) myös toteaa, että jokaisen koontiversion julkaisu tuotantoon yleensä vaatii, että julkaistava ohjelmisto on pilvipohjainen. Voidaankin todeta, että pilvipohjaisissa ratkaisuissa julkaisuprosessi on usein helpommin automatisoitavissa.

Kokonaisarkkitehtuuriin (engl. *enterprise architecture*) liittyen Chen (2015, 39) nostaa esille kaksi tärkeää seikkaa. Kehitettävän järjestelmän tulee olla arkkitehtuuriltaan testattava (engl. *testable*) ja julkaistava (engl. *deployable*). Testattavuudella Chen tarkoittaa sitä, että kehittäjien tulee itse voida testata ohjelmiston toimivuutta omalta työasemaltaan. Heidän ei tulisi tarvita monimutkaisia integroituja ympäristöjä tai muiden kehittäjien tai tiimien apua. Myös Neely & Stolt (2013, 127–128) korostavat testien tärkeyttä ja suosittelevat testauksen suorittamista julkaisuympäristön kanssa identtisessä ympäristössä. Julkaistavuudella Chen (2015, 39) taas tarkoittaa sitä, että uudet versiot tai parannukset voidaan julkaista tuotantoon nopeasti ilman merkittäviä huoltokatkoja. Vastauksena tähän Chen (2015, 38–39) ehdottaa monoliittisen (engl. *monolithic*) järjestelmän hajauttamista pienempiin itsenäisiin moduuleihin, joita voidaan päivittää ja muokata ilman, että sillä on vaikutusta muihin moduuleihin. Tätä hajauttamista kutsumme jatkossa modularisoinniksi (engl. *modularization*). Vastaavanlaiseen lopputulemaan ovat tulleet myös Shahin ym. (2019, 1075), Humble (2018, 39) sekä Laukkanen ym. (2015, 72–73). On tärkeää myös huomata, etteivät arkkitehtuuriin tehtävät muutokset ole aina helppoja (Debroy ym. 2018, 856), mutta näiden ongelmien käsittely jää tämän tutkimuksen rajauksen ulkopuolelle.

Teknisten vaatimusten lisäksi jatkuvan toimituksen monimutkainen luonne vaatii erityisiä toimintatapoja kehitystiimiltä. Humble (2018, 38–39) korostaa tässä kulttuurin merkitystä. Hän käyttää kulttuurista Scheinin (1999, 27) määritelmää, joka on vapaasti suomennettuna joukko hiljaisia oletuksia, jotka tiimi on oppinut ratkaistessaan sisäisiä ja ulkoisia ongelmia, ja jotka ovat toimineet riittävän hyvin, että ne voidaan opettaa tiimin

uusille jäsenillä oikeina tapoina ajatella ja tarkastella vastaavia ongelmia. Westrum (2004, 23–24) määrittelee produktiivisen tiimin olevan suoritusorientoitunut, yhteistyöhön ja kommunikaatioon kannustava niin tiimin sisällä, kuin muidenkin tiimien kanssa, riskin kaikkien jäsenien kesken jakava, virheet selvittävä sekä uusiin ideoihin ja itsenäiseen ajatteluun kannustava. Nämä ovat sellaisia kulttuurillisia piirteitä, joita jatkuva toimitus tiimiltä vaatii (Humble 2018, 38–39). Humble (2018, 38–39) kannustaa kehitystiimin ja operationaalisen tiimin väliseen vahvaan kommunikaatioon sekä virheiden käsittelemiseen oppimismahdollisuuksina. Jatkuva toimitus on osa laajempaa DevOps-kokonaisuutta, jonka nimikin kuvastaa kehityksen ja operationaalisten toimintojen yhdistämistä. Virheistä oppiminen taas mahdollistaa tiimin kehittymisen, mikä on jatkuvan toimituksen keskiössä (Gill ym. 2018, 131–133). Kaikki tämä kulminoituu tiimin kommunikaatioon niin sisäisesti kuin tiimistä ulospäinkin.

3.1.3 Jatkuvan toimituksen käyttöönotto

Jatkuvan toimituksen käyttöönottoon liittyy monia teknisiä sekä kulttuuriin, prosesseihin ja organisaatiomuutokseen liittyviä haasteita (Shahin ym. 2019, 1064). Suurimpia haasteita ovat siiloutuneet liiketoimintayksiköt, ihmisten muutosvastarinta, vastuunkanto ja väärinkäsitys siitä, että nopea toiminta vaarantaisi laadun (Gill ym. 2018, 132). Käyttöönotto voi myös olla erityisen hankalaa suuremmissa organisaatioissa (Chen 2015, 50). Chen (2017, 83) on listannut strategiat jatkuvan toimituksen käyttöönottamiseksi ohjelmistokehityksen puolella. Strategiat on lueteltu taulukossa 1. Näiden strategioiden pohjalta on luodaan myöhemmin tutkimuksen tavoitteena oleva artefakti vertailemalla tietovarastokehityksen ja ohjelmistokehityksen eroja jatkuvan toimituksen kannalta.

Taulukko 1. Jatkuvan toimituksen käyttöönoton helpottamisen strategiat (mukaillen Chen 2017, 83).

Strategia	Kuvaus
Myy jatkuva toimitus ratkaisuna ongelmaan.	Tunnista sidosryhmien ongelmat, jotka voidaan ratkaista jatkuvan toimituksen avulla ja myy jatkuva toimitus ratkaisuna tähän ongelmaan.
Perusta jatkuvan toimituksen kehittämiselle omistautunut tiimi, jossa on monialaista osaamista.	Perusta ryhmä, jonka ainoana tehtävänä on jatkuvan toimituksen kehittäminen, jotta se voi keskittyä vain siihen.
Toimita jatkuvasti jatkuvaa toimitusta.	Järjestele jatkuvan toimituksen jalkauttaminen siten, että se tuottaa arvoa mahdollisimman usein ja aikaisessa vaiheessa.
Aloita helposta, mutta tärkeästä ohjelmistosta.	Helppo kohde mahdollistaa arvon näyttämisen aikaisessa vaiheessa ja takaa projektin kehittämisen jatkumisen myös tulevaisuudessa.
Visualisoi jatkuvan toimituksen julkaisuputken runko.	Anna kehityksessä olevalle tiimille visuaalinen kuvaus jatkuvan toimituksen julkaisuputkesta jättäen tyhjäksi ne vaiheet, joita ei voida vielä implementoida.
Jalkauta asiantuntija.	Jalkauta yksi jatkuvan toimituksen kehitystiimin jäsenistä osaksi kehityksen kohteena olevaa tiimiä nostamaan motivaatiota tiimin sisällä.

Jatkuvan toimituksen käyttöönotto vaatii muutoksien tekemistä manuaalisten töiden automatisoinnin, kokonaisarkkitehtuurin, käytännön kehitystyön ja kulttuurin alueilla. Tällaisten suurten muutosten tekeminen vaatii tukea monilta eri sidosryhmiltä. Kaikilla näillä sidosryhmillä on kuitenkin omat päivittäiset ongelmansa, joten heidän sitouttamisensa jatkuvan toimituksen kehittämiseen on usein haastavaa. (Chen 2017, 74.) Vastauksena tähän, Chen (2017, 74) esittelee strategian jatkuvan toimituksen myymisenä ratkaisuna johonkin heidän omista ongelmistaan. Tässä strategiassa tavoitteena on siis tunnistaa kunkin sidosryhmän omat ongelmat ja yrittää keksiä, miten jatkuva toimitus voisi toimia kipulääkkeenä tähän ongelmaan. Kun sidosryhmät ymmärtävä, kuinka jatkuva toimitus voisi kyetä helpottamaan heidän jokapäiväistä työtään, heidän asenteensa sitä kohtaan paranee, jolloin sen käyttöönotto helpottuu.

Jatkuvan toimituksen käyttöönotto ei ole yksinkertainen tai nopea tehtävä. Jotta työ saataisiin suoritettua loppuun, täytyy sen suorittamiseen osoittaa omistautunut tiimi, jolla

ei ole muita tehtäviä. Tiimin nimittäminen takaa sen, että työlle osoitetaan riittävästi sen vaatimia resursseja. Tiimin omistautuminen vain jatkuvan toimituksen kehittämiselle taas takaa sen, että siihen käytetään riittävästi aikaa. (Chen 2017, 76.) Chen (2017, 76) toteaa, että kun jatkuvaa toimitusta yritetään ottaa käyttöön osa-aikaisesti muiden projektien ohessa, muut työt priorisoidaan jatkuvan toimituksen ohi, jolloin sen kehitys jää kesken. Vastaavaa ilmiö havaittiin tapahtuneen myös esimerkkiprojektissamme ennen tämän tutkimuksen aloittamista. Nimitetyn tiimin osaamisprofiilin täytyy myös olla moninainen. Chen (2017, 76–77) toteaa, että jatkuvan toimituksen käyttöönotossa pelkkä ohjelmistokehittämiskokemus ei riitä, vaan tiimistä täytyy löytyä osaamista myös järjestelmäylläpidosta, operationaalisista toimista ja prosessikehityksestä.

Kuten yllä jo mainittiin, jatkuvan toimituksen käyttöönotto on aikaa vievä prosessi. Nykyaikana liiketoimintaympäristöt muuttuvat nopeasti ja organisaatiossa saattaa tapahtua merkittäviä muutoksia lyhyessäkin ajassa. Jos jatkuvan toimituksen hyödyt realisoituvat liian kaukana tuntemattomassa tulevaisuudessa, saatetaan kehitystyö lopettaa kesken. Jos esimerkiksi johtoportaassa tapahtuu henkilöstömuutoksia, saatetaan jatkuva toimitus nähdä tarpeettomana, jos se ei ole tuottanut lisäarvoa aikaisessa vaiheessa. Tämän takia kolmas strategia on jatkuvan toimituksen jatkuva toimittaminen. Jotta useiden sidosryhmien tuki säilytetään koko pitkän prosessin ajan, täytyy hyötyä realisoitua jatkuvasti. (Chen 2017, 77.)

Läheisesti edelliseen kohtaan liittyy myös Chenin (2017, 78–79) neljäs strategia: aloita helposta, mutta tärkeästä sovelluksesta. Helposta sovelluksesta aloittaminen nostaa ensimmäisen käyttöönoton onnistumisen todennäköisyyttä ja nopeuttaa hyötyjen realisoitumista. Tämä takaa ylemmän johdon tuen jatkuvan toimituksen kehittämiselle laajemminkin. Toisaalta tärkeästä sovelluksesta aloittaminen mahdollistaa riittävien resurssien käyttämisen, kun tärkeitä prosesseja halutaan kehittää. Tärkeän prosessin siirtäminen jatkuvan toimituksen piiriin myös lisää jatkuvan toimituksen näkyvyyttä organisaatiossa ja helpottaa sitä kohtaan esiintyvää muutosvastarintaa. Hieman vastaavanlaista lähestymistapaa ehdottavat Neely ja Stolt (2013, 127) sillä erotuksella, että heidän mukaansa täytyy aloittaa kaikista hitaimmasta manuaalisesta prosessista. He siis korostavat erityisesti tärkeimmästä osiosta aloittamista. Tässä yhteydessä täytynee siis tasapainoilla hyödyn ja riskin välillä – hitaimman prosessin automatisointi eittämättä tuottaa paljon hyötyä, mutta Chen (2017, 78–79) jatkaa tätä pohdintaa sillä, että jos riski epäonnistumiselle on liian suuri, täytyy harkita vähemmän hyötyä tuottavasta, mutta varmemmin automatisoitavissa olevasta prosessista aloittamista.

Viidentenä strategiana on jatkuvan toimituksen julkaisuputken rungon visualisointi (Chen 2017, 79). Tämän strategian tarkoituksena on tarjota kehityksen kohteena olevalle tiimille visuaalinen kuvaus siitä, mihin jatkuvan toimituksen käyttöönotolla pyritään. Kuvaus sisältää kaikki askeleet, jotka valmiissa julkaisuputkessa tulevat olemaan, mutta ne askeleet, joita ei vielä syystä tai toisesta kyetä automatisoimaan, jätetään ikään kuin tyhjiksi. Aina kun ohjelmistokehittäjä suorittaa manuaalisesti ohjelmiston testausta, hän näkee julkaisuputken rungon ja muistaa, että tämäkin askel voitaisiin automatisoida. Tämä muistutus lisää tiimin motivaatiota kehittää prosessejaan siten, että jatkuva toimitus saataisiin otettua käyttöön.

Erityisen vaikeissa tapauksissa tarvittavien muutoksien tekeminen tiimin prosesseihin sen ulkopuolelta voi olla liian vaikeaa. Chenin (2017, 79–80) viimeinen strategia onkin asiantuntijan jalkauttaminen kehityksen kohteena olevaan tiimiin. Asiantuntijan täytyy ymmärtää koko jatkuvan toimituksen käyttöönottoprosessi syvällisesti ja hänellä tulee olla kokemusta sen läpiviennistä. Tiimin sisältä käsin asiantuntijalla on parempi mahdollisuus ymmärtää sen sisäisiä ongelmia sekä kommunikoida tiimille jatkuvan toimituksen käyttöönoton tärkeydestä.

Chenin (2017) strategiat keskittyvät jatkuvan toimituksen käyttöönottoon melko korkealla tasolla, eivätkä juuri ota kantaa teknologisiin toteutuksiin tai käytännön kehitystyöhön. Jotta tämäkin puoli jatkuvan toimituksen käyttöönotosta saataisiin huomioitua, jatketaan hänen ajatuksiaan seuraavaksi luvussa 3.1.2 tehdyillä havainnoilla.

Chenin (2017) strategioiden lisäksi jatkuvaa toimitusta käyttöönotettaessa täytyy siis ohjelmiston arkkitehtuuri rakentaa modulaarisesti (Laukkanen ym. 2015, 72–73; Humble 2018, 39; Shahin ym. 2019, 1075). Lisäksi kehitykseen osallistuvan henkilöstön kulttuuriin täytyy pyrkiä vaikuttamaan siten, että tiimin sisäinen ja tiimistä ulospäin suuntautuva kommunikaatio paranee, jotta erityisesti kehitystoiminnasta ja operationaalisesta toiminnasta vastaavat henkilöstöt tietävät samat asiat ja pyrkivät samaan lopputulokseen (Humble & Molesky 2011, 10–11; Humble 2018, 38–39). Myös loppukäyttäjien aktiivinen mukaan ottaminen kommunikointiin vahvistaa jatkuvan toimituksen vaikutuksia (Neely & Stolt 2013, 128). Gill ym. (2018, 133–134) ehdottavat ratkaisuksi kulttuuriin liittyviin ongelmiin vallitsevan tilanteen ja halutun tilanteen erojen kartoittamista sekä näitä eroja pienentävien toimintojen suunnittelua. Ebert ym. (2016, 99) taas ehdottavat ratkaisuksi sitä, että kehittäjät ottavat niin kutsutun full-stack-kehitysroolin, jossa he vastaavat kehitystyön lisäksi myös julkaisuista ja operationaalisesta toiminnasta. Tämä luonnollisesti parantaa kommunikaatiota kehityspuolen ja operationaalisen puolen välillä, kun

molemmista puolista vastaa sama tiimi. Chenin (2017) esittelemien strategioiden lisäksi muusta kirjallisuudesta löydetyt strategiat on esitelty taulukossa 2.

Taulukko 2. Strategiat jatkuvan toimituksen käyttöönottamiseksi muun kirjallisuuden perusteella.

Strategia	Kuvaus
Modularisoi arkkitehtuuri (Laukkanen ym. 2015, 72–73; Humble 2018, 39; Shahin ym. 2019, 1075)	Ohjelmiston arkkitehtuuri tulee rakentaa niin, että eri osiot ovat itsenäisiä moduuleja, joihin tehtävät muutokset eivät vaikuta muihin itsenäisiin osiin.
Kartoita kulttuuriin liittyvät ongelmakohdat ja rakenna suunnitelma niiden ratkaisemiseksi (Humble & Molesky 2011, 10–11; Neely & Stolt 2013, 128; Humble 2018, 38–3; Gill ym. 2018, 133–134)	Kulttuurin tulee mahdollistaa saumaton kommunikointi kehitystiimin ja operationaalisen tiimin sekä loppukäyttäjien välillä. Yhtenä mahdollisena vaihtoehtona niin kutsuttu full-stack-kehitys (Ebert ym. 2016, 99).

3.2 Tietovarastokehitys

Tässä luvussa tarkastelemme tietovarastokehitystä jatkuvan toimituksen kannalta oleellisin osin. Tarkoituksena on esitellä yhteneväisyyksiä ja eroavaisuuksia tietovarastokehityksen ja ohjelmistokehityksen käytännön työskentelytapojen välillä jatkuvan toimituksen kannalta. Tavoitteena on eroavaisuuksien tunnistamisen avulla muokata jatkuvan toimituksen käyttöönoton malli ohjelmistokehityksen kontekstista tietovarastokehityksen ympäristöön sopivaksi.

Kirjallisuuskatsauksessa huomattiin, että tietovarastointiin liittyvissä artikkeleissa käsitellään huomattavan vähän varsinaista kehitystyötä ja kehitykseen liittyviä käytänteitä. Tämän takia lähteitä on käytössä melko rajallisesti ja niistä tehtyjä johtopäätöksiä on pyritty jatkamaan esimerkkitapauksemme kehitysprojektista kerätyillä havainnoilla.

3.2.1 Tietovarastokehitys ja jatkuva toimitus

Tässä tutkimuksessa tietovarastolla tarkoitetaan yritystason tietovarastoa, johon kerätään tietoa keskitetysti useista eri lähteistä ja johon täytyy tehdä usein muutoksia. Esimerkkiprojektin tietovarastoa käytetään useiden raporttien lähteenä, eli kyseessä on BI-projekti. Uusia raportointitarpeita herää usein, ja uudet raportit halutaan tuotantoon mahdollisimman nopeasti, joten tietovarastoa täytyy kehittää jatkuvasti. Tämä lisää jatkuvan

toimituksen käyttöönoton tarvetta. Vaikka esimerkkitietovarastoamme käytetään raportointiin, voidaan tutkimuksen tulokset yleistää myös muihin tietovarastoprojekteihin, joissa tietovarastoa kehitetään tai muutetaan usein. Staattisten tietovarastojen tarkastelu tässä yhteydessä ei ole mielekästä, sillä harvoin tehtäviä julkaisuprosesseja ei ole useinkaan kannattavaa automatisoida.

Tietovarastokehitys nopeammat julkaisut tuottavat organisaatiolle lisäarvoa, kun eri järjestelmiin varastoitua tietoa saadaan nopeammin hyödynnettävään muotoon ja iteratiivinen toiminta mahdollistaa korkeamman laadun (Herschel 2012, 297). Tältä osin tietovarastokehitys ei siis eroa ohjelmistokehityksestä. Puonti ym. (2018, 250) toteavat, että jatkuvan toimituksen kannalta ajateltuna tietovarastokehitys ei eroa ohjelmistokehityksestä muuten, kuin yleisesti käytettyjen työkalujen, prosessien ja toimintamallien osalta. Koska työkalut otetaan tässä tutkimuksessa annettuina, tarkastelemme seuraavaksi tietovarastokehityksessä yleisesti käytettyjä prosesseja ja toimintamalleja.

Tyypillisessä tietovarastokehitysprojektissa kehitystiimit käyttävät yhtä ja samaa kehitysympäristöä samanaikaisesti, eikä mitään versionhallintajärjestelmää ole käytössä. Tämä tarkoittaa sitä, että kaikkien kehittäjien täytyy omaa kehitystä tehdessään huomioida muiden kehittäjien tekemät muutokset. Näin ollen julkaisut täytyy koordinoida niin, ettei kenelläkään ole keskeneräistä kehitystyötä menossa, mikä tarkoittaa, että kaikkien pitää aikatauluttaa työnsä erikseen sovitun julkaisuaikataulun mukaisesti. Tämä yleensä johtaa entistäkin pidempään julkaisusykliin ja ylimääräisiin kustannuksiin. (Puonti ym. 2018, 251.) Toisaalta tietovarastokehityksessä voidaan käyttää versionhallintajärjestelmiä siinä, missä ohjelmistokehityksessäkin. Näin on tehty tutkimuksen esimerkkiprojektissa, jossa versionhallintaa suoritetaan Git-versionhallintajärjestelmällä. Ilman versionhallintajärjestelmää jatkuvan toimituksen käyttäminen tietovarastokehityksessä on lähestulkoon mahdotonta.

Versionhallintajärjestelmän ollessa käytössä, julkaisuputkien käyttäminen tietovarastokehityksessä ei juuri eroa ohjelmistokehityksestä (Puonti ym. 2018, 260). Huomionarvoista on kuitenkin se, että tietovarastokehityksessä saattaa olla tehtäviä, joita ei esiinny perinteisessä ohjelmistokehityksessä, mutta jotka voitaisiin mahdollisesti automatisoida jatkuvan toimituksen menetelmiä hyödyntäen. Esimerkiksi Puonti ym. (2018, 261) ovat käyttäneet kolmannen osapuolen työkalua, jolla tiedot saadaan mallinnettua helpommin staging-alueelta tietovarastoon. Tiedon mallinnus manuaalisesti tehtynä on hyvin työläs ja aikaa vievä prosessi, ja sen automatisoinnilla on saavutettavissa merkittävää hyötyä.

Aikaisemmin esitelty esimerkkiprojektissa käytössä oleva DV-Modeller on vastaavanlainen työkalu.

Luvussa 3.1.2 käsittelimme jatkuvaan toimitukseen parhaiten soveltuvaa arkkitehtuuria. Parhaiten jatkuvaan toimitukseen sopii modulaarinen arkkitehtuuri, jossa muutoksia voidaan tehdä yksittäisiin moduuleihin ilman, että ne vaikuttavat muihin moduuleihin. Yritystason tietovarasto vaikuttaa ensisilmäyksellä Chenin (2015, 38–39) ja Shahinin ym. (2019, 1075) kuvaileman monoliittiselta kokonaisuudelta, jossa muutosten vaikutusta voi olla vaikea arvioida, ja jonka hajottaminen pienempiin moduuleihin voi olla hankalaa. Tähän voidaan kuitenkin vaikuttaa tiedonmallinnustavan valinnalla. Esimerkkiprojektissamme mallinnustavaksi on valittu Data Vault (ks. Lindstedt & Olschimke 2015), jossa muutosten tekeminen yksittäisiin alueisiin on mahdollista ilman, että malliin vaikutetaan laajemmin (Krneta ym. 2016, 473–474). Tämän voidaan olettaa helpottavan jatkuvan toimituksen käyttöönottoa.

Tietovarastossa laajemmin, siis yritystason tietovaraston ulkopuolella, modularisointia voidaan tehdä helpommin. Esimerkiksi datamarttien osalta tämä onnistuu helposti käyttämällä jokaiselle datamartille omaa skeemaansa, kuten esimerkkiprojektissamme on tehty. Tällöin jokaiseen skeemaan täytyy ottaa kaikki siihen kuuluvat taulut mukaan. Tällaisella menettelyllä varmistetaan se, että kuhunkin datamarttiin voidaan tehdä muutoksia ilman, että vaikutetaan muiden datamarttien toimintaan. Kun yhtä datamarttia kehittää vain yksi kehittäjä, hän voi tehdä ja julkaista haluamansa muutokset koska vain ilman huolta laajemmista vaikutuksista.

Kehitystiimin kulttuuriin liittyvät vaatimukset eivät juuri eroa tietovarastokehityksen ja ohjelmistokehityksen välillä. Kommunikointia tietovarastokehityksessä helpottaa se, että tiimit ovat usein pienempiä ja sama tiimi hoitaa sekä kehityksen että operationaalisen toiminnan (Puonti ym. 2018, 250–251). Silti kommunikaation merkitystä ei voida unohtaa. Myös tietovarastokehityksessä loppukäyttäjien merkitys kehitystyön määrittelyssä on hyvin olennaista, joten avoimen kommunikaatioyhteyden saavuttaminen on tärkeää (Ang & Teo 2000, 18).

3.2.2 Jatkuvan toimituksen käyttöönotto tietovarastokehityksessä

Edellä mainitun perusteella voidaan todeta, että taulukossa 1 esitellyistä Chenin (2017, 83) strategioista osa on suoraan käyttökelpoisia jatkuvan toimituksen käyttöönotossa myös tietovarastokehityksen kontekstissa. Jatkuvan toimituksen myyminen ratkaisuna ongelmaan, jatkuvan toimituksen jatkuva toimittaminen ja julkaisuputken rungon

visualisointi ovat kaikki käyttökelpoisia sellaisenaan. Helpoista ja tärkeistä ohjelmistoista aloittaminenkin sopii tietovarastokehitykseen, jos ohjelmistojen sijaan puhutaankin tietovaraston osasta. Muut esitellyt strategiat vaativat hieman muokkaamista. Lisäksi strategioihin tarvitaan lisäyksiä, jotta tietovarastokehityksen erityispiirteet saadaan huomioitua.

Chenin (2017) esittelemät strategiat on luotu hyvin suuren organisaation laajojen toimintojen muuttamiseksi jatkuvan toimituksen mukaisiksi. Tätä kuvastaa strategia jatkuvan toimituksen kehittämiselle omistautuneen tiimin perustaminen. Tässä tutkimuksessa tutkimuksen kohteena on kuitenkin vain yksittäisen projektin ja sen kehitystiimin toiminnan kehittäminen. Tätä tarkoitusta varten kokonaisen tiimin perustaminen olisi eittämättä hyödyllistä, mutta myös kohtuutonta resurssien tuhlausta. Voidaan todeta, että tämä strategia on suuremmissa organisaatioissa varmasti tarpeellinen, mutta ei sovellu pienempiin projekteihin. Niinpä se jätetään tarkastelun ulkopuolelle.

Samassa strategiassa on myös maininta tiimin monialaisesta osaamisesta, mikä on erittäin tärkeää jatkuvassa toimituksessa (Chen 2017, 76). Tätä voida jättää tarkastelun ulkopuolelle tietovarastokehityksessäkään. Koska kokonaisen tiimin luominen yhtä kehityskohdetta varten vaikuttaa resurssien tuhlaukselta, yhdistetään moniosaaminen asiantuntijan jalkauttamiseen. Vaadittavia osaamisalueita myös rajataan Chenin (2017, 77) ehdottamista ohjelmistokehityksestä, järjestelmäylläpidosta, operationaalisista toimista ja prosessikehityksestä kapeammalle alueelle, koska näiden kaikkien osa-alueiden kokonaisvaltainen hallitseminen olisi yhdelle henkilölle liian suuri vaatimus. Lisäksi osaamisvaatimuksia voidaan pienentää, sillä Chenin (2017) esittelemän tiimin tuli kyetä kehittämään teknologioita ja prosesseja monenlaisissa tiimeissä, joissa prosessit ja teknologiat saattoivat olla hyvin vaihtelevia. Jatkuvan toimituksen tuominen tietovarastokehityksen ympäristöön vaatii huomattavasti kapeampaa osaamista. Niinpä jalkautettavalle asiantuntijalle voisi riittää kokemus tietovarastokehityksestä sekä jatkuvan toimituksen käytänteistä. Kokemus tietovarastokehityksestä takaa, että asiantuntija ymmärtää tiimin käyttämät kehitykseen liittyvät prosessit sekä kehitettävän järjestelmä kokonaisuutena. Ymmärrys jatkuvasta toimituksesta taas takaa asiantuntijan kyvyn ymmärtää sitä, miten tiimin toimintoja tulee kehittää, jotta haluttuun lopputulokseen päästään. Luonnollisesti kokemus varsinaisesta jatkuvan toimituksen käyttöönotosta on merkittävä etu.

Chenin (2017) esittelemien strategioiden lisäksi jatkuvan kehityksen käyttöönotossa täytyy huomioida arkkitehtuurin modularisointi (Laukkanen ym. 2015, 72–73; Humble 2018, 39; Shahin ym. 2019, 1075) ja organisaation kulttuuri (Neely & Stolt 2013, 128; Humble 2018, 38–3; Gill ym. 2018, 133–134), kuten taulukossa 2 on todettu. Lisäksi

jatkuvan toimituksen käyttöönotossa tietovarastokehityksessä täytyy ottaa huomioon siihen liittyvät erityispiirteet. Tietovarastokehityksessä ei välttämättä käytetä versionhallintajärjestelmää (Puonti ym. 2018, 251) ja mallinnusmenetelmän valinta saattaa vaikuttaa jatkuvan toimituksen käyttöönottoon. Niinpä tietovarastokehityksessä jatkuvan toimituksen kannalta tärkeitä askelia ovat myös versionhallintajärjestelmän käyttöönotto ja helposti muokattavan tietovaraston mallinnustavan valinta.

Edellä mainitun perusteella luotu toimintamalli esitellään taulukossa 3.

Taulukko 3. Toimintamalli jatkuvan toimituksen käyttöönottamiseksi tietovarasto-kehityksessä.

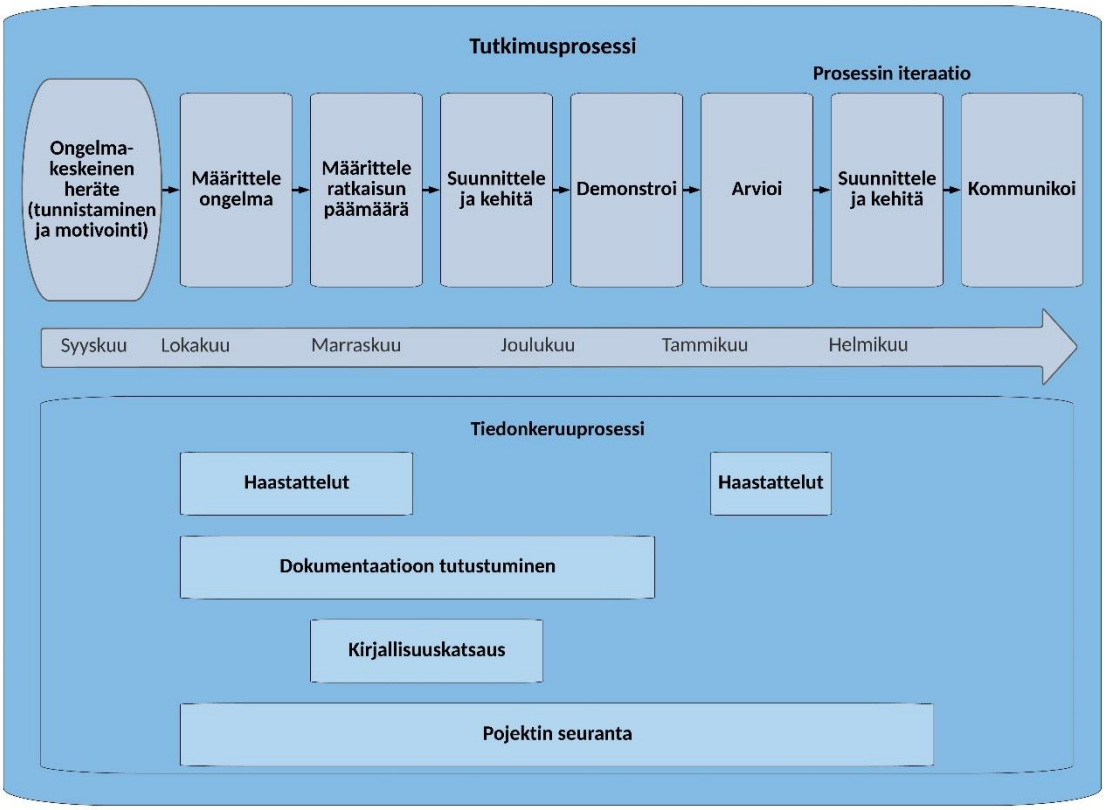
	Strategia	Kuvaus
1.	Versionhallintajärjestelmän käyttöönottaminen (Humble & Farley 2010, 32–33; Mäkinen ym. 2016, 185; Puonti ym. 2018, 258).	Ota kehitysprojektissa käyttöön versionhallintajärjestelmä, jossa jokainen kehittäjä pitää omaa kehityshaaraansa ja jossa on omat haaransa yhteiselle kehitysympäristölle, testausympäristölle ja tuotantoympäristölle.
2.	Helposti muokattavissa olevan yritystason tietovaraston mallinnustavan valitseminen (Chen 2015, 38–39; Laukkanen ym. 2015, 72–73; Shahin ym. 2019, 1075).	Valitse tietovaraston mallintamiseen sellainen tapa, jossa tietovaraston yksittäisiä osia voidaan helposti muokata ilman, että se vaikuttaa muihin tietovaraston osiin.
3.	Tietovaraston modularisoiminen (Chen 2015, 38–39; Laukkanen ym. 2015, 72–73; Shahin ym. 2019, 1075).	Modularisoi tietovaraston osat kuten datamartit ja kuutiot siten, että muutokset yhteen osaan eivät vaikuta muihin osiin.
4.	Kulttuuriin liittyvien ongelmakohtien kartoittaminen ja suunnitelman rakentaminen niiden ratkaisemiseksi (Humble & Molesky 2011, 10–11; Neely & Stolt 2013, 128; Humble 2018, 38–3; Gill ym. 2018, 133–134)	Kulttuurin tulee mahdollistaa saumaton kommunikointi kehitystiimin ja operationaalisen tiimin sekä loppukäyttäjien välillä. Yhtenä mahdollisena vaihtoehtona niin kutsuttu full-stack-kehitys (Ebert ym. 2016, 99).
5.	Jatkuva toimituksen myyminen ratkaisuna ongelmaan (Chen 2017, 83).	Tunnista sidosryhmien ongelmat, jotka voidaan ratkaista jatkuvan toimituksen avulla ja myy se ratkaisuna tähän ongelmaan.
6.	Moniosaavan asiantuntijan jalkauttaminen (Chen 2017, 83).	Asiantuntijan tulee työskennellä osana kehitystiimiä ja ainoana tehtävänä tulee olla jatkuvan toimituksen kehittäminen, jotta hänen aikaansa ei kulu muihin tehtäviin. Asiantuntijalla tulee olla kokemusta tietovarastokehityksestä ja osaamista jatkuvan toimituksen alueella.
7.	Helposta ja tärkeästä tietovaraston osasta aloittaminen (Neely & Stolt 2013, 127; Chen 2017, 83).	Helppo kohde mahdollistaa arvon näyttämisen aikaisessa vaiheessa ja takaa projektin kehittämisen jatkumisen myös tulevaisuudessa.
8.	Jatkuvan toimituksen jatkuva toimittaminen (Chen 2017, 83).	Järjestele jatkuvan toimituksen jalkauttaminen siten, että se tuottaa arvoa mahdollisimman usein ja aikaisessa vaiheessa.
9.	Jatkuvan toimituksen julkaisuputken rungon visualisoiminen (Chen 2017, 83).	Anna kehityksessä olevalle tiimille visuaalinen kuvaus jatkuvan toimituksen julkaisuputkesta jättäen tyhjäksi ne vaiheet, joita ei voida vielä implementoida.

4 TUTKIMUSTULOKSET

Tässä luvussa esitellään tutkimuksen vaiheet ja aikajänne. Lisäksi esitellään toimintamallin käyttöönoton ja haastattelujen aikana kerätyt tutkimustulokset. Tavoitteena on antaa lukijalle kuva tutkimusprosessin etenemisestä ja luodun toimintamallin käyttöönotosta esimerkkiprojektissa.

4.1 Tutkimusprosessi ja sen aikana tehdyt havainnot

Tutkimus toteutettiin KEHA-keskuksen tietovarastointiprojektissa ja tutkimus kesti kuusi kuukautta. Tutkimusprosessi ja tiedonkeruuprosessi sekä niiden aikajanat on esitelty kuviossa 6.



Kuvio 6. Tutkimusprosessi (mukaillen Peffers ym. 2007, 54) ja tiedonkeruuprosessi.

Tarve julkaisukäytänteiden automatisoinnille oli kasvanut esimerkkiprojektissa jo pitkään, mutta heräte realisoitui syyskuussa. Peffersin ym. (2007, 54) mukaan herätteen jälkeen ensimmäinen suunnitteluttutkimusprosessin askel sisältää ongelman tunnistamisen, määrittelyn ja motivoinnin. Esimerkkitapauksessamme ongelmakeskeinen heräte syntyi kuitenkin ongelman tunnistamisesta ja motivaatiosta puuttua siihen. Ensimmäiseksi vaiheeksi jäi siis ongelman määrittely, jonka aikana haastateltiin projektin henkilöstöä,

tutustuttiin projektin ja valittujen työkalujen dokumentaatioihin sekä seurattiin projektin toimintoja yleisesti. Ongelmaksi määriteltiin se, että kehityksen kohteena olevan tietovaraston julkaisuprosessit veivät tarpeettoman suuren osuuden kehittäjien työajasta. Tarkkoja lukuja julkaisuprosessien ajankulutuksesta ei ollut saatavilla, joten ongelman määrittelyssä tukeuduttiin kehittäjien tuntemuksiin. Ongelman määrittelyn yhteydessä paljastui lisäksi, että projektissa oli jo käytössä työkalut, joiden avulla julkaisuprosesseja voitaisiin automatisoida, mutta niitä ei ollut vielä otettu kokonaisvaltaisesti käyttöön johtuen kiireellisemmistä kehitystöistä. Automatisointia oli yritetty aloittaa useaan otteeseen, mutta valmista kokonaisuutta ei ollut saatu aikaiseksi.

Läheisesti ongelman määrittelyyn liittyi myös prosessin toinen vaihe, ratkaisun päämäärän määrittely. Ongelman määritelmän perusteella looginen ratkaisun päämäärä on julkaisuprosessien nopeuttaminen. Koska projektissa oli jo käytössä julkaisukäytänteiden automatisointiin kykenevä työkalu, oli sen ominaisuuksien kokonaisvaltainen käyttöönotto melko luonteva ratkaisu, jolla päämäärään päästäisiin. Kuitenkin, jotta aiheesta saataisiin kattavasti tietoa, tässä vaiheessa tietoa alettiin keräämään myös tutustumalla olemassa olevaan akateemiseen kirjallisuuteen.

Kirjallisuutta haettiin ensin tietovarastoinnista ja DevOps-toimintamallista. Hyvin pian kirjallisuus tarkentui kuitenkin DevOpsista kokonaisuutena sen yhteen osa-alueeseen, jatkuvaan toimitukseen, jonka tunnistettiin kuvaavan paremmin projektissa olevaa puutetta. Niinpä haastattelujen, projektin seurannan ja kirjallisuuskatsauksen perusteella projektin kannalta parhaimmaksi ratkaisuksi määriteltiin jatkuvan toimituksen käyttöönotto. Kirjallisuuskatsauksessa havaittiin kuitenkin, että jatkuvaa toimitusta tietovarastokehityksessä oli käsitellyt aikaisemmin ainoastaan Puonti ym. (2018). Heidän tutkimuksensa keskittyi siihen, millaisia piirteitä tietovaraston kehitysprojektissa tulee olla, että jatkuva toimitus voidaan ottaa käyttöön, eikä varsinaisesti siihen, miten jatkuva toimitus otetaan käyttöön. Tarvittiin jatkotutkimusta, joten ratkaisun päämääräksi asettui sellaisen toimintamallin luominen, jonka avulla jatkuva toimitus voitaisiin ottaa käyttöön tietovarastokehitysprojektissa.

Kolmas vaihe, suunnittelu ja kehitys, tapahtui pääasiallisesti kirjallisuuskatsauksen perusteella. Kirjallisuuskatsauksessa löydettiin toimintamalli, jossa oli tunnistettu strategioita jatkuvan toimituksen käyttöönottamiseksi ohjelmistokehityksen kontekstissa. Lisäksi tunnistettiin ohjelmistokehityksessä jatkuvan toimituksen kannalta olennaisia elementtejä, joita vertailtiin tietovarastokehityksen vastaaviin elementteihin. Tämän

vertailun avulla ohjelmistokehityksen kontekstiin luotu toimintamalli saatiin muunnettua sopivaksi tietovarastokehityksen kontekstiin.

Demonstrointi tapahtui joulukuun ja tammikuun aikana. Taulukossa 3 esitellyn toimintamallin ensimmäiset neljä askelta eivät aiheuttaneet toimenpiteitä, sillä ne oli otettu projektissa jo huomioon. Projektissa oli valmiiksi käytössä versionhallintajärjestelmä ja yritystason tietovaraston mallinnustavaksi oli valittu DV-menetelmä, joka soveltuu hyvin inkrementaalisten muutosten tekemiseen. Lisäksi tietovaraston osat, kuten datamartit ja kuutiot oli jo modularisoitu. Voidaan kuitenkin todeta, ettei jatkuvaa toimitusta olisi voitu ottaa käyttöön ainakaan niin kuin se otettiin, jos näitä asioita ei olisi huomioitu. Neljäskään askel, kulttuuriin liittyvien ongelmakohtien tunnistaminen ja ratkaiseminen, ei aiheuttanut toimenpiteitä, sillä esimerkkiprojektissa sama tiimi vastasi kehitystoiminnasta ja operationaalisesta toiminnasta full-stack-kehityksen omaisesti (ks. Ebert ym. 2016, 99). Projektitiimin ja loppukäyttäjien kommunikoinnista toisaalta löytyisi varmasti parantamisen varaa, mutta projektissa päätettiin priorisoida muita osa-alueita.

Toimintamallin viides askel, jatkuvan toimituksen myyminen ratkaisuna ongelmaan, jäi vähemmälle huomiolle, sillä koko jatkuvan toimituksen käyttöönoton ajatus oli lähtenyt liikkeelle ongelman ratkaisusta. Esimerkkiprojektissa jatkuva toimitus siis ikään kuin myi itse itsensä ratkaisuna ongelmaan. Tämä epäilemättä helpotti sen käyttöönottoa.

Jatkuvan toimituksen käyttöönottamiseksi täytyi kuudentena askeleena kehitystiimiin jalkauttaa moniosaava asiantuntija. Uuden asiantuntijan palkkaaminen pieneen kehitystiimiin lisää henkilöstökuluja merkittävästi, eikä tähän aina olla valmiita. Näin kävi myös esimerkkiprojektissamme, eikä uutta asiantuntijaa saatu palkattua. Moniosaavien asiantuntijoiden löytäminen lyhyellä aikataululla osoittautui myös hankalaksi. Ongelma kierrettiin määrittelemällä yhdelle tiimin alkuperäisistä jäsenistä tehtäväksi jatkuvan toimituksen käyttöönotto. Tällä henkilöllä oli vankkaa osaamista tietovarastokehityksestä ja hieman kokemusta jatkuvan toimituksen käyttöönotosta. Käyttöönotto aikataulutettiin lomakaudelle, jotta muut kehitystyöt olisivat tauolla, eivätkä häiritsisi jatkuvaa toimitusta. Ulkopuolisen tarkastelun perusteella tällä järjestelyllä päästiin vähintään tyydyttävään lopputulokseen melko nopealla aikataululla ja pienillä kustannuksilla.

Esimerkkiprojektissa tietovaraston pääasiallinen tarkoitus oli toimia lähteenä erilaisille liiketoimintatietoraporteille. Vaikka kaikki tietovaraston osat ovat välttämättömiä tämän tehtävän suorittamiseksi, ehkä kaikkein tärkeimpiä ovat raportoinnin pohjana olevat dimensionaaliset mallit, eli kuutiot. Sen lisäksi, että kuutiot ovat tärkeä tietovaraston osa, ne ovat myös melko hyvin modularisoituja ja niistä jokaista hallitaan kuutiokohtaisen

ohjelmistorepositorion avulla. Tämä tekee niistä helposti toimitettavia. Niinpä seitsemännen askeleen, helposta ja tärkeästä tietovaraston osasta aloittamisen kohteeksi asettuivat kuutiot.

Azure DevOps -alustalla jatkuvan toimituksen julkaisutoiminnot on jaettu kahteen osioon: koontiversioputket (engl. *build pipeline*) ja julkaisuputket (engl. *release pipeline*) (ks. Azure DevOps dokumentaatio). Näistä ensimmäinen hoitaa siis käytännössä katsoen jatkuvaa integraatiota, jossa kehittäjän versionhallintarepositorioon julkaisemasta koodista rakennetaan automaattisesti koontiversio. Koontiversioputki luo ohjelmistosta myös julkaistavan ajotiedoston, joka voidaan julkaisuputkien avulla siirtää kehitysympäristöstä testausympäristön kautta julkaisuympäristöön. Koontiversioputki voidaan siis asettaa onnistuneen suorituksen jälkeen automaattisesti käynnistämään julkaisuputki. Esimerkkiprojektissa jokaiselle kuutiolle rakennettiin oma koontiversioputkensa. Koontiversioputket asetettiin käynnistymään automaattisesti, kun kehittäjä yhdisti kehityshaaransa päähaaraan. Koontiversioputken onnistunut suoritus asetettiin myös käynnistämään automaattisesti julkaisuputken suoritus.

Julkaisuputket yksinkertaisuudessaan poistavat julkaisun kohteena olevasta ympäristöstä julkaistavan kuution, luovat sen uudestaan koontiversioputken luoman ajotiedoston perusteella, muuttavat kuutioon ympäristökohtaiset tiedot, kuten osoitteet ja kirjautumistiedot, ja päivittävät kuution datasisällön. Näiden askelien jälkeen kuutioilla suoritetaan kullekin kuutiolle erikseen määritellyt testit. Jokaisessa kuutiossa koontiversioputki käynnistää julkaisuputken testiympäristöön. Joissain kuutioissa on haluttu kuitenkin varmistaa tehtyjen muutosten toimivuus myös manuaalisesti, jolloin tuotantoon julkaisut on jätetty manuaalisiksi.

Kuutioista aloittaminen johti jatkuvan toimituksen jatkuvaan toimittamiseen. Kun ensimmäinen julkaisuputki oli saatu rakennettua, muiden kuutioiden julkaisuputkien rakentaminen oli huomattavasti helpompaa ja nopeampaa, jolloin tuloksia alkoi syntyä jatkuvasti. Tämä voitiin havaita nostavan kehittäjien ja projektin johdon motivaatiota jatkuvan toimituksen käyttöönoton jatkamiselle.

Yhdeksäs askel, jatkuvan toimituksen julkaisuputken rungon visualisoiminen, tapahtui projektiin valitussa työkalussa automaattisesti samalla kuin putkia rakennettiin. Rakentaminen tapahtui kuitenkin niin nopeasti, ettei visualisoinnista tuntunut olevan merkittävästi apua motivaation parantamisessa. Chen (2017, 79) toteaakin julkaisuputkien visualisoinnin olevan erityisen tärkeää projekteissa, joissa jatkuvan toimituksen käyttöönotto on erityisen hankalaa ja aikaa vievää, sillä visualisointi voi lisätä motivaatiota.

Ulkoisen tarkastelun perusteella esimerkkiprojektissa motivaatio-ongelmaa ei ehtinyt syntyä, joten visualisoinnista ei ollut merkittävää hyötyä.

Suurimpia ongelmia jatkuvan toimituksen käyttöönotossa esimerkkiprojektissa vaikutti aiheuttavan kolmannen osapuolen tarjoama, DV-mallinnukseen käytetty DV-Modeller. Ongelmia aiheutti ohjelmiston luoma yritystason tietovaraston metadata, jonka siirtäminen ympäristöstä toiseen on hankalaa. Näitä metatietoja lukuun ottamatta tietovarastosta saatiin luotua julkaistava dacpac-tiedosto (ks. SQL Server dokumentaatio), joka voitiin siirtää julkaisuputkien avulla ympäristöstä toiseen. Dacpac-tiedostoon täytyi kuitenkin lisätä manuaalisesti luotu SQL-skripti, jonka avulla DV-mallintajalla tehdyt muutokset saatiin automaattisesti tehtyä myös seuraavassa ympäristössä. Manuaalinen työskentely sotii jatkuvan toimituksen periaatteita vastaan, mutta ongelmaan ei lyhyellä aikavälillä löydetty parempaa ratkaisua. Skriptin luomista pyrittiin automatisoimaan niin pitkälle kuin mahdollista, mutta luomisprosessiin jäi vielä joitain manuaalisia askeleita. Monet muutoksista eivät vaadi uutta mallintamista, joten skriptejä ei tarvitse luoda jokaisen julkaisun yhteydessä. Tavoitteena on kuitenkin lähitulevaisuudessa saada myös mallintajan tietojen toimitus täysin automatisoitua. Tähän ei vielä toistaiseksi kuitenkaan keksitty järkevää ratkaisua.

4.2 Haastattelujen tulokset

Haastattelukysymykset muodostettiin luodun toimintamallin perusteella. Liitteessä 1 on esitelty kysymykset sekä osoitettu kysymysten liitoskohdat luotuun toimintamalliin ja sieltä edelleen tarkasteltuihin kirjallisuuslähteisiin. Haastattelujen tavoitteena oli arvioida luodun toimintamallin toimivuutta ja kehittää toimintamallia edelleen.

Versionhallintajärjestelmän käyttäminen mainittiin haastatteluissa usein tärkeänä osana jatkuvaa toimitusta. Jatkuvan toimituksen kuvailtiin myös pohjautuvan versionhallintajärjestelmään, mikä korostaa versionhallintajärjestelmän käyttämisen tärkeyttä. Tärkeimmäksi työkaluksi jatkuvan toimituksen käyttöönoton kannalta useimmin mainittiin projektissa käytössä ollut Azure DevOps -alusta, jonka avulla suoritettiin versionhallintaa ja rakennettiin julkaisuputkia. Haastatteluissa nousi myös esille se, että käytettyjen työkalujen tulee olla mahdollisimman hyvin yhteensopivia. Jatkuva toimitus onnistui parhaiten niillä osa-alueilla, joita hallittiin Microsoftin luomilla työkaluilla. Eniten vaikeuksia tuotti DV-Modeller, jonka luomia metadatoja ei Azure DevOps -alustan työkaluilla saatu siirrettyä automaattisesti. Yksi haastateltavista ilmaisia asian näin: *”Azure DevOps nivoo nuo kaikki Microsoftin työkalut tosi nätisti yhteen. Kuutiot ja niiden putkitukset, Git-*

versionhallinta, Synapse-tietovarasto ja SSAS kaikki toimii hyvin. Mutta sitten toi Modeller ei taivu tähän ollenkaan." Modelleriin liittyvillä hankaluuksilla viitataan sen luomien metadatojen siirtojen aiheuttamiin vaikeuksiin. Toisaalta haastatteluissa nousi esille, että metadatojen siirto tuottaa aina hankaluuksia: *"Metadatan siirto on aina vaikeaa, joten jos sitä olisi hallinnoitu jossain tietovaraston ulkopuolella, niin automatisointi olisi voinut olla helpompaa."* Metadatojen helpompaa hallintaa mahdollistavaa työkalua ei kuitenkaan osattu nimetä.

Haastatteluissa ei noussut esille muita työkaluja ja käytäntöjä, jotka olisivat helpottaneet jatkuvan toimituksen käyttöönottoa. Esille nousi kuitenkin käyttöoikeuksiin liittyviä ongelmia: *"Laajemmat käyttöoikeudet olisivat helpottaneet työskentelyä, jotta DevOps-Market Placesta olisi pystynyt asentelemaan kolmannen osapuolen komponentteja."* Ongelma syntyi siis siitä, että kehittäjien käyttöoikeudet eivät riittäneet komponenttien käyttöönottamiseen ja testaamiseen. Toisin sanoen kaikista tärkeimpänä pidetyn työkalun täysi potentiaali jäi saavuttamatta, mikä epäilemättä vaikutti jatkuvan toimituksen käyttöönottoon.

DV-mallinnuksen osalta mainittiin seuraavaa: *"Data Vault vaikuttaisi tukevan CI/CD toimintaa, koska Data Vault tarjoaa mahdollisuuden iteratiiviseen tietovaraston rakentamiseen."* Esille nousi myös se, että DV-mallinnukseen käytetty DV-Modeller aiheutti kaikista suurimmat ongelmat, sillä sen luomia metadatoja ei saatu siirrettyä julkaisuputkissa automaattisesti. DV-Modellerin käyttö vaati SQL-skriptien kirjoittamista, jotta metadatat saatiin luotua julkaisujen jälkeen uudessa ympäristössä, mikä aiheutti manuaalista, monimutkaista ja virheherkkää työtä. Toisaalta mallinnustavasta mainittiin, että vastaavia hyötyjä ja haittoja löytyy myös muista mallinnustavoista. DV-mallinnus ei siis sinänsä vaikuttaisi olevan edellytys jatkuvan toimituksen käyttöönottoon. Mallinnustavan iteratiivinen luonne kuitenkin tunnistettiin helpottavaksi tekijäksi tietovaraston jatkuvan toimituksen käyttöönotossa.

Modularisoinnin havaittiin helpottavan jatkuvan toimituksen käyttöönottoa. Julkaisuprosessit ovat usein herkkiä virheille, joten pienempien kokonaisuuksien julkaisu keräsi kiitosta ketteryydestä erityisesti SSAS-kuutioiden osalta, sillä ne vaativat usein paljon hienosäätöä ja useita julkaisuja. Datamarteista ja SSAS-kuutioista mainittiin seuraavaa: *"Datamartit ja SSAS-kuutiot toimivat oikein mallikkaasti. Toisaalta tämä saattaa johtua myös siitä, että ne ovat Microsoftin [työkaluja], toisin kuin [DV-mallinnettu] tietovarasto. Modularisointi vaikuttaisi kuitenkin olevan positiivinen asia."* Haastatteluissa nousi myös esille se, ettei tietovaraston osia voida välttämättä koskaan täysin erotella

toisistaan, sillä tietovarasto on alusta loppuun yhtenäinen putki. Vähintään data yhdistää aina eri osiot toisiinsa. Toisaalta julkaisukäytäntöjen näkökulmasta tällä ei pitäisi olla väliä.

Esimerkkiprojektissamme kommunikaatio jatkuvan toimituksen aikana nähtiin monilta osin puutteellisena. Tärkeimpänä syynä tähän pidettiin sitä, että jatkuvasta toimituksesta vastasi yksi henkilö. Näin kehitystiimin sisällä ymmärrys jatkuvasta toimituksesta ja sen käytännön hyödyntämisessä projektissa ei ollut samalla tasolla, eivätkä tiimin jäsenet aina ymmärtäneet toisiaan: *”Kommunikaation abstraktiotaso oli yleisessä kommunikoinnissa liian korkea. Tekijä itse tiesi tarkalla tasolla, mutta kukaan muu ei, joten kommunikaatio oli hyvin hankalaa.”* Toinen haastateltava taas mainitsi: *”Puuttumaan jäi hyvät dokumentaatiot ja käyttöohjeet julkaisuputkien käytöstä. Jäi kertomatta ja dokumentoimatta tarkalla tasolla putkien toiminta. Tämä aiheuttanut paljon ylimääräistä työtä, koska vain yksi henkilö tunsi putkien sielunelämän.”*

Yhden henkilön vastuuttaminen jatkuvan toimituksen käyttöönotossa aiheutti myös sen, että ideoita ei voitu *”pallotella”* tai *”brainstormata”,* mikä saattoi aiheuttaa sen, ettei se valittu ratkaisu ollut välttämättä paras mahdollinen. Ratkaisuna kommunikaatio-ongelmiin nähtiin se, että jatkuvaan toimituksen käyttöönottoon otetaan mukaan useampia henkilöitä. Toisaalta yhden henkilön rakentamaa kokonaisuutta pidettiin myös yhtenäisempänä kuin isomman tiimin tekemää. Yksi henkilö sai tarvittavat kehitystyöt myös tehtyä huomattavan nopeasti, joten useamman henkilön resursointi tehtävään olisi voinut olla tuhlausta. Voitaisiin siis todeta, että resurssien määrä oli riittävä, mutta kommunikaatiota tiimin sisällä olisi pitänyt parantaa jollain tavalla. Haastattelujen perusteella kommunikaatio-ongelmaan ei löytynyt muuta ratkaisua, kun useamman henkilön osallistaminen kehitystyöhön.

Muun, suoraa arvoa tuottavan, kehitystyön ajateltiin yleisesti haittaavan jatkuvan toimituksen käyttöönottoa. Esimerkkiprojektissamme jatkuvan toimituksen käyttöönotolle oli kuitenkin pyhitetty riittävä aika, eikä tätä ongelmaa päässyt syntymään. Tämä johtui pääosin siitä, että työ aikataulutettiin lomakauden päälle, jolloin monet muista projekteista olivat tauolla. Lisäksi työ priorisoitiin korkealle muiden töiden ohi. Toisin sanoen muun työn häiritsevää luonnetta ei tässä tutkimuksessa päästy empiirisesti arvioimaan. Haastattelujen perusteella kehitystiimissä kuitenkin yleisesti oltiin sitä mieltä, että muun työn täytyy luonnollisestikin haitata jatkuvan toimituksen käyttöönottoa.

Tärkeimpinä taitoina, joita jatkuvan toimituksen käyttöönottamiseksi esimerkkiprojektissa olisi haastateltavien mukaan vielä tarvittu, oli asiantuntijuus käytetyissä

työkaluissa. Esimerkiksi jo aiemmin mainittiin, että Azure DevOps -alustan kolmannen osapuolten luomien komponenttien tuntemus olisi ollut hyödyllistä. Tähän ratkaisuna nähtiin pidempiaikainen kokemus sovelluksen käytöstä. Lisäksi jatkuvan toimituksen käyttöönottoa helpottavaksi tekijäksi mainittiin yleisesti kokemus jatkuvan toimituksen käyttöönotosta, kokemus tietovarastokehityksestä sekä mieluusti kokemus jatkuvasta toimituksesta tietovarastokehityksen kontekstissa. Täytynee mainita, että tällaista osaamista omaavia henkilöitä ei maailmasta kovin montaa löydy. Tämä vahvistaa päätelmää siitä, että jatkuvan toimituksen käyttöönottoon ei riitä yksi henkilö. Haastatteluissa nousi myös moneen kertaan esille ongelmat, jotka liittyivät DV-Modellerin luoman metadatojen julkaisuihin. Tähän ratkaisuksi nähtiin DV-Modellerin kehitystiimin konsultointi jatkuvan toimituksen käyttöönotossa: *”Jos olisi tajuttu, että [metadatojen julkaisu] menee noin vaikeaksi, niin olisi ehkä alusta asti otettu [Modellerin kehittäjä] mukaan.”*

Motivaatio jatkuvan toimituksen käyttöönottoon esimerkkiprojektissa vaikuttaa syntyneen puhtaasti siitä, että julkaisuprosessit olivat hyvin työläitä, aikaa vieviä ja virheherkkiä: *”Ilman putkia julkaisu on hidasta ja riskialtista. Usein julkaisut ovat yhden henkilön takana, joten automatisointi mahdollistaa muidenkin tehdä julkaisuja.”* Jatkuvalla toimituksella nähtiin voitavan parantaa julkaisujen varmuutta ja vapauttaa kehittäjien työaikaa arvoa tuottavaan työhön. Virheherkillä julkaisuprosesseilla nähtiin olevan myös taipumus kasautua yhden kehittäjän harteille, sillä se, joka niitä eniten teki, onnistui niissä myös todennäköisemmin. Tämä johtaisi pitkässä juoksussa siihen, että vain yksi henkilö olisi kykenevä suorittamaan julkaisuja, ja tällaista henkilösidonnaisuutta haluttiin välttää. Jatkuvan toimituksen avulla vastuuta nähtiin siis voitavan siirtää myös muille kehitystiimin jäsenille.

Motivaatioon jatkuvan toimituksen käyttöönoton aikana vaikutti selkeästi tiedostetut hyödyt. Haastatteluissa mainittiin myös seuraavaa: *”Jos kehitys olisi viivästynyt enemmän tai mitään konkretiaa ei olisi saatu, niin homma olisi voinut jäädä kesken.”* Motivaation laskusta mainittiin myös metadataan liittyvät ongelmat, joiden selättäminen oli hankalaa. Yleisesti pienet onnistumiset ja konkreettisten tulosten näkyminen aikaisessa vaiheessa mainittiin motivaatiota lisääviksi tekijöiksi ja ongelmat ja vaikeudet motivaatiota laskeviksi.

Kehitystiimille jäi sellainen tunne, että jatkuvan toimituksen käyttöönotto olisi voinut onnistua paremmin, jos tiimin sisäinen kommunikaatio olisi toiminut paremmin ja kaikilla olisi ollut selkeä kuva jatkuvan toimituksen käyttämisestä projektissa. Asiasta kommentoitiin seuraavasti: *”Kokonaisuus olisi pitänyt kommunikoida tiimille tarkemmin.*

Kaikki eivät ymmärtäneet kokonaisuutta, mikä aiheutti turhia työtunteja." Yhteistyö DV-Modellerin kehitystiimin kanssa olisi myös johtanut julkaisuputkien automatisoinnin parantamiseen. Erityisen hyvin onnistuneena pidettiin tietovaraston modulaaristen osuuksien julkaisuputkien toimintaa sekä jatkuvan toimituksen käyttöönoton nopeutta.

Kehitystiimiltä kysyttiin haastatteluissa, millaisen arvosanan he antaisivat jatkuvan toimituksen käytölle projektissa ennen toimintamallin hyödyntämistä. Arvosana-asteikoksi annettiin 0–5, jossa 0 = ”ei lainkaan” ja 5 = ”täydellisesti”. Tällä arvostelulla kolme haastateltavista antoi arvosanaksi 0 ja yksi antoi arvosanaksi 1. Edellä esitellyn toimintamallin hyödyntämisen jälkeen arvosanoiksi annettiin 3, 3, 4 ja 4. Toimintamallin avulla suoritettu jatkuvan toimituksen käyttöönotto oli siis ainakin tällä mittarilla jokseenkin onnistunut, varsinkin jos huomioon otetaan käyttöönoton nopea aikataulu ja käytössä olleet resurssit. Nopean aikataulun voidaan ajatella myös olleen ratkaisutekijä moniin ongelmakohtiin, sillä nopeassa projektissa motivaatio-ongelmia ei välttämättä pääse niin herkästi syntymään.

5 JOHTOPÄÄTÖKSET JA RAJOITTEET

Tässä luvussa yhdistetään tutkimuksen tulokset aikaisempaan teoriaan ja arvioidaan niitä käyttäen Hevnerin ym. (2004, 83) esittelemää suunnittelututkimuksen ohjeistusta. Lisäksi käsitellään tutkimuksen rajoitteita ja jatkotutkimuksen tarpeita.

5.1 Johtopäätökset

Versionhallintajärjestelmän käyttämisen on todettu olevan edellytyksenä jatkuvan toimituksen käyttämiselle (esim. Humble & Farley 2010, 32–33; Mäkinen ym. 2016, 185; Puonti ym. 2018, 258). Tietovarastokehityksessä versionhallintaa ei kuitenkaan aina käytetä (Puonti ym. 2018, 251). Niinpä jatkuvan toimituksen käyttöönottamisen tueksi muodostetun toimintamallin ensimmäiseksi askeleeksi asetettiin versionhallintajärjestelmän käyttöönotto. Versionhallintajärjestelmän hyödyllisyys havaittiin jatkuvan toimituksen käyttöönottoa seuraamalla sekä käyttöönoton jälkeisissä haastatteluissa. Versionhallintajärjestelmän käyttäminen vaikuttaisi jatkuvan toimituksen käyttöönoton kannalta jopa välttämättömältä. Monissa jatkuvaa toimitusta käsittelevissä akateemisissa artikkeleissa versionhallintajärjestelmää ei edes erikseen mainita, vaan sen käyttämistä jatkuvan toimituksen kontekstissa pidetään itsestään selvyytenä (esim. Chen 2015, 51; Shahin ym. 2019, 1062–1063). Niinpä versionhallintajärjestelmän käyttäminen on toimintamallin ensimmäinen askel, vaikka se monissa yhteyksissä voi vaikuttaa itsestään selvältä.

Toimintamallin toinen askel, helposti muokattavissa olevan tietovaraston mallinnustavan valinta, vaikutti kirjallisuuskatsauksen perusteella hyvältä keinolta modularisoida tietovarastoa. Kuitenkaan, vaikka esimerkiksi DV-mallinnus tarjoaakin mahdollisen iteratiiviseen kehitykseen, se ei erottele tietovaraston osia toisistaan. Haastattelujen perusteella voidaan todeta, että niin kauan, kun yritystason tietovarasto pidetään kokonaisuudessaan yhdessä ohjelmistorepositoriossa, mallinnustavan valinta ei vaikuta julkaisuprosessiin. Toisaalta jatkuvan toimituksen ideana on nopeat kehityssyklit ja tiheät julkaisut (Chen 2015, 50; Humble 2018, 34), jotka vaativat mallinnustavalta iteratiivista luonnetta. Myös haastatteluissa nousi esiin se, että DV-mallinnus tuki jatkuvan toimituksen käyttöönottoa nimenomaan iteratiivisuutensa takia. Niinpä toimintamallin toinen askel muutetaan korostamaan mallinnustavan iteratiivista luonnetta.

Toimintamallin käyttöönoton seurannan sekä käyttöönoton jälkeisten haastattelujen perusteella tietovaraston modulaaristen osien julkaisuprosessien automatisointi oli jatkuvan toimituksen käyttöönotossa helpointa. Vastaaviin tuloksiin ovat tulleet

ohjelmistokehityksen puolella Chen (2015, 38–38), Laukkanen ym. (2015, 72–73) ja Shahin ym. (2019, 1075). Haastatteluissa ilmeni, että erityisesti kuutioiden julkaisuprosesseja pidettiin helppoina. Tähän syyksi nähtiin se, että jokainen kuutio oli omassa ohjelmistorepositoriossaan. Datamarttien osalta julkaisuprosessin automatisointia pidettiin onnistuneena, mutta julkaisuja häiritsi se, että muukin tietovarasto oli samassa ohjelmistorepositoriossa. Voidaan kuitenkin todeta, että tietovaraston modularisointi on tärkeää jatkuvan toimituksen käyttöönotossa. Tähän askeleeseen lisätään haastattelujen perusteella maininta siitä, että jokaiselle moduulille tulee käyttää omaa ohjelmistorepositoriotansa.

Haastattelujen perusteella jatkuvan toimituksen käyttöönoton aikana kommunikaatiossa olisi ollut parannettavaa. Ongelmat kommunikaatiossa ilmenivät muun muassa niin, etteivät muut kehitystiimiläiset ymmärtäneet jatkuvaan toimitukseen käytettyjä teknologioita riittävän syvällisesti osatakseen itse käyttää niitä tai auttaa niihin liittyvien ongelmien kanssa. Toisaalta tämä aiheutui myös siitä, että projektissa tehtiin tietoinen päätös jättää kehitystyö yhden henkilön vastuulle. Kommunikaatioon liittyviä kulttuurillisia ongelmia ei siis edes yritetty kartoittaa, saati ratkaista, sillä niiden merkitystä käyttöönoton onnistumiselle ei pidetty kovin merkittävinä. Toisaalta ajatuksena oli, että koska kehitystä tehtiin Ebertin ym. (2016, 99) esittelemä full-stack-kehitysmenetelmä mukaisesti, ei kommunikaatioon tarvitsisi kiinnittää niin paljon huomiota. Haastatteluissa kuitenkin ilmeni, että useimpien kehittäjien mielestä helpoin tapa parantaa käyttöönoton onnistumista olisi ollut parantaa tiimin sisäistä kommunikaatiota. Myös aiempi tutkimus korostaa kulttuurin kehittämistä kommunikaation parantamiseksi (Humble & Molesky 2011, 10–11; Neely & Stolt 2013, 128; Humble 2018, 38–39; Gill ym. 2018, 133–134). Näin ollen myös neljännen askeleen hyödyllisyydelle löytyy tukea. Mallista kuitenkin poistetaan maininta full-stack-kehityksestä, sillä se ei esimerkkiprojektissa yksin riittänyt.

Viides askel, jatkuvan toimituksen myyminen ratkaisuna ongelmaan, on Chenin (2017, 83) esittelemä ajatus. Suoranaista tukea tälle askeleelle ei esimerkkiprojektista saatu, sillä jatkuvaa toimitusta ei tarvinnut myydä tiimille ensinkään, koska heräte jatkuvan toimituksen käyttöönotolle syntyi tiimin sisältä. Kuitenkin haastatteluissa tärkeimmäksi motivaatiota lisääväksi tekijäksi jatkuvan toimituksen käyttöönottamiseksi sekä käyttöönoton aikana nähtiin nimenomaan sen toimiminen ratkaisuna ongelmaan. Niinpä Chenin (2017) ajatukset vaikuttaisivat saavan tukea myös tässä tutkimuksessa, vaikkakaan suoranaista empiiristä näyttöä asiasta ei saatu.

Chenin (2017, 83) ajatus asiantuntijan jalkauttamisesta kehitystiimiin perustuu tapaustutkimukseen, jossa yksi DevOps-tiimi suoritti jatkuvan toimituksen käyttöönottoa

suuressa organisaatiossa yksi kehitystiimi kerrallaan. Tapauksen kohdeorganisaatiossa oli siis useita eri kehitystiimejä, ja yksi tiimi, joka vastasi kaikkien tiimien jatkuvan toimituksen käyttöönotosta. Tässä asetelmassa huomattiin, että erityisen vaikeissa tapauksissa jatkuvan toimituksen asiantuntija kannattaa asettaa toimimaan tiimin kanssa päivittäisessä toiminnassa, jotta hän voisi tiimin sisältä toimia jatkuvan toimituksen puolestapuhujana. Tämän tutkimuksen esimerkkiprojektissa tilanne oli hieman erilainen, sillä jatkuva toimitus haluttiin ottaa käyttöön vain yhdessä tiimissä ja motivaatio käyttöönottoon syntyi tiimin sisältä. Seurannan perusteella vaikuttaisi siltä, että kehitystiimissä oli selkeästi yksi jatkuvan toimituksen puolesta puhuja, joka tunsi jatkuvasta toimituksesta saatavat hyödyt. Haastattelujen perusteella hänen merkitystään motivaation lisääjänä ei kuitenkaan voida todentaa.

Chen (2017, 83) korostaa myös, että jatkuvan toimituksen käyttöönotossa tarvitaan monialaista asiantuntijuutta, eikä pelkästään kontekstin tunteminen riitä. Toisin sanoen, jos jatkuvaa toimitusta otetaan käyttöön ohjelmistokehityksen kontekstissa, on kokemus ohjelmistokehityksestä tärkeää, mutta se ei yksin riitä, vaan osaamista tarvitaan myös järjestelmäylläpidosta, operationaalisesta toiminnasta ja prosessikehityksestä. Toimintamallia luodessa asiantuntijan osaamisvaatimuksiksi kuvattiin kokemus tietovarastokehityksestä ja osaaminen jatkuvan toimituksen alueella. Vaatimukset jätettiin tarkoituksella melko avoimiksi, koska tutkimusta aiheesta ei löytynyt. Haastattelujen perusteella erityiseksi taitovaatimukseksi nousi käytettyjen työkalujen tunteminen. Myös DevOps-periaatteiden syvällinen ymmärtäminen mainittiin käyttöönottoa helpottavana tekijänä. Niinpä vaikuttaisi siltä, että jatkuvan toimituksen käyttöönotossa tietovarastokehityksen ympäristössä vaaditaan asiantuntijuutta sekä tietovarastokehityksestä että DevOpsista tai jatkuvasta toimituksesta.

Kuten jo mainittiin, Chenin (2017) tapauksessa jatkuvaa toimitusta otettiin käyttöön suuressa organisaatiossa ja useissa kehitystiimeissä. Tällöin on luontevaa olettaa, että jatkuvan toimituksen käyttöönottoon on käytössä huomattava määrä resursseja. Tämän tutkimuksen esimerkkiprojektissa tilanne oli kuitenkin toinen, eikä kokonaisen tiimin perustaminen ollut realistinen vaihtoehto, kuten se ei ole useimmissa muissakaan tietovarastokehitysprojekteissa. Niinpä toimintamallissa asiantuntijatiimi korvattiin yhdellä asiantuntijalla. Haastattelujen perusteella vaikuttaisi kuitenkin siltä, että useamman henkilön käyttäminen olisi helpottanut tiimin sisäistä kommunikaatiota ja että kommunikaation parantaminen olisi ollut yksittäisenä toimintona eniten jatkuvan toimituksen käyttöönottoa helpottanut tekijä. Niinpä jatkuvan toimituksen käyttöönoton vastuuttaminen yhdelle

asiantuntijalle ei enää vaikuta järkevältä. Toisaalta resurssien niukkuus aiheuttaa sen, ettei jatkuvan toimituksen käyttöönottoon voida nimittää useampia henkilöitä. Emme myöskään voi olettaa, että useimmissa tietovarastokehitysprojekteissa kehitystiimissä olisi asiantuntijuutta jatkuvasta toimituksesta. Jokaisessa tällaisessa projektissa on kuitenkin asiantuntijuutta tietovarastokehityksestä. Niinpä parhaalta ratkaisulta vaikuttaisi jatkuvan toimituksen asiantuntijuuden palkkaaminen tiimin ulkopuolelta ja kehitystiimin osallistaminen jatkuvan toimituksen käyttöönottoon tietovarastoasiantuntijoina. Näin jatkuvan toimituksen käyttöönottoon saadaan tarvittava asiantuntijuus, eikä tiimiin tarvitse palkata kuin yksi uusi henkilö. Tiimin osallistaminen jatkuvan toimituksen käyttöönottoon parantaa myös kommunikaatiota tiimin sisällä sekä ymmärrystä jatkuvan toimituksen merkityksestä kehitystoiminnassa.

Jatkuvan toimituksen käyttöönotto esimerkkiprojektissa kävi hyvin nopeasti, eikä tästä syystä varsinaisia motivaatio-ongelmia päässyt syntymään. Haastatteluissa kuitenkin mainittiin, että jos käyttöönotossa olisi tapahtunut merkittäviä takaiskuja tai se olisi myöhästynyt merkittävästi, olisi käyttöönotto saatettu keskeyttää. Lisäksi onnistuneet suoritukset mainittiin motivaatiota kasvattavina tekijöinä. Niinpä voidaan todeta, että toimintamallin seitsemännen askeleen, helposta ja tärkeästä tietovaraston osasta aloittamisesta, saatiin näyttöä. Tämä tulos on yhteydessä Neelyn ja Stoltin (2013, 127) sekä Chenin (2017, 83) tulosten kanssa. Hyvin läheisesti seitsemänteen askeleeseen liittyy myös kahdeksas askel, jatkuvan toimituksen jatkuva toimittaminen. Kuten edellä todettiin, jatkuvan toimituksen käyttöönotto kävi esimerkkiprojektissa hyvin nopeasti, joten näytöt kahdeksannen askeleen hyödyistä osuvat paremmin seitsemänteen askeleeseen.

Yhdeksännen askeleen hyötyjen arviointi on tässä tutkimuksessa hankalaa. Toimintamallia demonstroidessa ajateltiin, että DevOps-alustan automaattisesti visualisoimat julkaisuputket olisivat riittävät täyttämään tämän askeleen tarkoituksen. Jälkikäteen ajateltuna voidaan kuitenkin todeta, ettei näin ollut. Jotta askeleen hyöty olisi saatu esiin, olisi yksittäisten julkaisuputkien lisäksi pitänyt visualisoida projektin jatkuva toimitus kokonaisuudessaan, jotta kehitystiimi olisi voinut sisäistää jatkuvan toimituksen käyttöönoton tarkoituksen sekä nähdä, missä osissa on vielä parannettavaa. Näin he olisivat voineet antaa oman panoksensa ongelmien ratkaisemiseksi. Tällaista kokonaisvaltaista visualisointia ei projektissa kuitenkaan tehty, joten sen arviointi jälkikäteen on mahdotonta. Voidaan kuitenkin todeta, että visualisointi olisi voinut lisätä motivaatiota ja helpottaa kommunikaatiota tiimin sisällä. Demonstraation aikainen huolimattomuus kuitenkin rajaa tämän askeleen tutkimuksen ulkopuolelle.

Edellä mainitun perusteella luotu toimintamallin jatkokehitetty versio on esitelty taulukossa 4. Mallissa on huomioitu vain ne askeleet, joiden hyödyistä saatiin näyttöä esimerkkiprojektin seurannan ja haastatteluiden perusteella. Mallin yksityiskohtia on myös muokattu saatujen tulosten perusteella.

Taulukko 4. Toimintamallin jatkokehitetty versio.

	Strategia	**Kuvaus**
1.	Versionhallintajärjestelmän käyttöönottaminen (mukaillen Humble & Farley 2010, 32–33; Mäkinen ym. 2016, 185; Puonti ym. 2018, 258).	Ota kehitysprojektissa käyttöön versionhallintajärjestelmä, jossa jokainen kehittäjä pitää omaa kehityshaaraansa ja jossa on omat haaransa yhteiselle kehitysympäristölle, testausympäristölle ja tuotantoympäristölle.
2.	Iteratiivisen muokkauksen mahdollistavan yritystason tietovaraston mallinnustavan valitseminen (mukaillen Chen 2015, 38–39; Laukkanen ym. 2015, 72–73; Shahin ym. 2019, 1075).	Valitse tietovaraston mallintamiseen sellainen tapa, jossa tietovarastoa voidaan kehittää iteratiivisesti mahdollistaen lyhyemmät julkaisusyklit.
3.	Tietovaraston modularisoiminen (mukaillen Chen 2015, 38–39; Laukkanen ym. 2015, 72–73; Shahin ym. 2019, 1075).	Modularisoi tietovaraston osat kuten datamartit ja kuutiot siten, että muutokset yhteen osaan eivät vaikuta muihin osiin. Kutakin moduulia tulee hallita omassa ohjelmistorepositoriossaan.
4.	Kulttuuriin liittyvien ongelmakohtien kartoittaminen ja suunnitelman rakentaminen niiden ratkaisemiseksi (mukaillen Humble & Molesky 2011, 10–11; Neely & Stolt 2013, 128; Humble 2018, 38–3; Gill ym. 2018, 133–134)	Kulttuurin tulee mahdollistaa saumaton kommunikointi kehitystiimin ja operationaalisen tiimin sekä loppukäyttäjien välillä.
5.	Jatkuvan toimituksen asiantuntijan nimittäminen (mukaillen Chen 2017, 83).	Jatkuvan toimituksen käyttöönotosta vastaava asiantuntija voi olla tiimin sisältä tai ulkopuolelta palkattu henkilö. Hänen tulee olla ensisijaisesti jatkuvan toimituksen asiantuntija, ja muuta kehitystiimiä tulee osallistaa jatkuvan toimituksen käyttöönottoon, jotta heidän asiantuntijuutensa saataisiin hyödynnettyä.
6.	Helposta ja tärkeästä tietovaraston osasta aloittaminen (mukaillen Neely & Stolt 2013, 127; Chen 2017, 83).	Helppo kohde mahdollistaa arvon näyttämisen aikaisessa vaiheessa ja takaa projektin kehittämisen jatkumisen myös tulevaisuudessa.

Haastattelujen ja projektin seurannan perusteella voidaan todeta, että toimintamallin käyttäminen helpotti jatkuvan toimituksen käyttöönottoa. Voidaan myös todeta, että jatkuvan toimituksen käyttäminen tietovarastokehityksessä vapauttaa kehittäjien aikaa

varsinaiseen kehitystyöhön ja tehostaa näin kehitystiimin toimintaa. Lisäksi jo aikaisessa vaiheessa huomattiin, että muutosten välitön siirto eri ympäristöihin tarjosi mahdollisuuden välittömään palautteeseen loppukäyttäjiltä, mikä parantaa lopputuotteen laatua ja luottamusta kehitystiimin ja loppukäyttäjien välillä (esim. Humble & Molesky 2011, 11–12; Fowler 2013; Chen 2015, 52).

5.2 Tulosten arviointi

Hevner ym. (2004, 83) esittelivät suunnittelututkimusta käsittelevässä artikkelissaan seitsemän ohjenuoraa suunnittelututkimuksen tekemiseksi. Nämä ohjenuorat on esitelty taulukossa 5. Ohjenuorien avulla jäsennellään tämän suunnittelututkimuksen tulosten arviointia.

Taulukko 5. Suunnittelututkimuksen ohjenuorat (mukaillen Hevner ym. 2004, 83).

	Ohjenuora	Kuvaus
1.	Suunnittele artefaktiksi	Suunnittelututkimuksen täytyy tuottaa artefakti, joka on konstrukti, malli, metodi tai toteutus.
2.	Ongelman merkityksellisyys	Suunnittelututkimuksen tavoitteena on kehittää teknologiaperusteinen ratkaisu tärkeään ja olennaiseen liiketoiminnan ongelmaan.
3.	Suunnitelman arviointi	Luodun artefaktin hyödyllisyys, laatu ja tehokkuus täytyy demonstroida perusteellisesti käyttäen hyvin toteutettuja arviointimenetelmiä.
4.	Tutkimuksen vaikutukset	Tehokkaan suunnittelututkimuksen tulee vaikuttaa selkeästi ja todennettavasti artefaktin, suunnittelun perusteiden tai suunnittelumetodologioiden alalla.
5.	Tutkimuksen täsmällisyys	Suunnittelututkimus perustuu perusteellisten metodien käyttöön artefaktin luomisen ja arvioinnin yhteydessä.
6.	Suunnittelu etsimisprosessina	Tehokkaan artefaktin etsiminen vaatii tarvittavien menetelmien käyttöä samalla huomioiden ympäristön asettamat reunaehdot.
7.	Tutkimuksen kommunikointi	Suunnittelututkimus täytyy esitellä tehokkaasti sekä teknologisesti orientoituneille että johtamisorientoituneille yleisöille.

Tässä suunnittelututkimuksessa luotiin artefakti, joka on toimintamalli jatkuvan toimituksen käyttöönottamiseksi tietovarastokehityksen kontekstissa. Artefakti on siis ensimmäisen ohjenuoran kuvauksessa mainittu malli. Se on Hevnerin ym. (2004, 84) kuvailema tiedollinen työkalu (engl. *intellectual tool*) jolla helpotetaan tietojärjestelmän kehitystyötä. Hevner ym. (2004, 84) toteavat tällaisen artefaktin olevan tietojärjestelmätutkimuksen kannalta merkityksellinen vain, jos sen toimivuudesta on epävarmuutta. Kuten luvuissa 1.1 ja 2.3.1 osoitetaan, tietovarastokehityksessä ei toistaiseksi tuntemattomasta syystä olla omaksuttu jatkuvan toimituksen käytäntöjä, vaikka niiden hyödyt ovat kiistattomat. Jos jatkuvan toimituksen käyttöönotosta koituu hyötyä, mutta sitä ei silti oteta käyttöön, voitaneen olettaa, että käyttöönottoon liittyy epävarmuutta, jota ei olla valmiita kantamaan. Luotu artefakti ei tietenkään poista kaikkia jatkuvan toimituksen käyttöönottoon liittyvää riskejä, mutta se toimii työkaluna helpottamaan sellaista kehitystyötä, johon ilman artefaktin käyttöä liittyy vielä enemmän epävarmuutta.

Tietojärjestelmätieteen tutkimuksen on tarkoitus tuottaa tietoa ja ymmärrystä, jotka mahdollistavat uusien tietoteknisten ratkaisujen kehittämisen ja käyttöönoton, jotta tärkeitä ja toistaiseksi selvittämättömiä liiketoimintaongelmia voidaan ratkaista (Hevner ym. 2004, 84). Tässä tutkimuksessa liiketoimintaongelmana oli tietovarastokehityksen ja erityisesti sen julkaisuprosessien kankeus. Kuten Puonti ym. (2018, 250) toteavat, tietovarastokehityksessä ei vielä aktiivisesti käytetä jatkuvaa toimitusta, jolla kehitystyötä voitaisiin ketteröittää. Niinpä luotu artefakti on uusi, käyttöönottoa helpottava malli, joka ratkaisee aikaisemmin selvittämättömän ongelman.

Suunnittelututkimuksen tärkeä osa on luodun artefaktin hyödyllisyyden, laadun ja tehokkuuden perusteellinen arviointi (Hevner ym. 2004, 85). Hevner ym. (2004, 86) esittelevät erilaiset yleisesti hyväksytyt arviointimenetelmät, joista tässä tutkimuksessa parhaiten sopivaksi valittiin tapaustutkimus, joka on havainnointiin perustuva menetelmä. Tapaustutkimus valikoitui parhaana menetelmänä, koska jatkuva toimitus on hyvin kontekstisidonnainen. Jatkuvan toimituksen käyttöönottoa on mahdoton demonstroida ilman tosielämän projektia. Lisäksi artefaktin arviointiin ei ollut käytössä muita mittareita, kuin asiantuntijoiden näkemykset. Tapaustutkimukseen sisällytettyjen haastattelujen avulla tutkimukseen saatiin helposti usean tietovarastokehityksen asiantuntijan näkemys artefaktin tehokkuudesta. Näiden näkemysten perusteella voidaan todeta, että toimintamallin käyttäminen paransi jatkuvan toimituksen hyödyntämisen astetta esimerkkiprojektissa. Toisaalta tutkimuksessa ei päästy vertailemaan vastaavaa käyttöönottoprojektia ilman

toimintamallin käyttöä, joten toimintamallin tehokkuutta ei päästy vertailemaan tilanteeseen ilman toimintamallin käyttöä.

Suunnittelututkimus voi myötävaikuttaa tietojärjestelmätieteen tutkimukseen kolmella tapaa. Yleisin tapa on vaikuttaa artefaktiin itseensä, jolloin artefakti ratkaisee jonkun aiemmin ratkaisemattoman ongelma tai laajentaa artefaktin toimintaa uuteen ympäristöön. Toiseksi suunnittelututkimus voi vaikuttaa suunnittelututkimuksen perusteisiin luomalla artefakteja uudella ja innovatiivisella tavalla. Viimeiseksi suunnittelututkimus voi kehittää uusia artefaktien arviointitapoja, mikä viittaa myös suunnittelututkimuksen itsensä kehittämiseen. (Hevner ym. 2004, 87.) Tässä tutkimuksessa vaikutukset painottuvat artefaktiin itseensä. Tutkimuksessa jatkuvan toimituksen käyttöönoton toimintamallia laajennettiin ohjelmistokehityksen kontekstista tietovarastokehityksen kontekstiin. Vaikutukset suunnittelututkimukseen itseensä jäivät toissijaiseksi.

Suunnittelututkimuksen täsmällisyyden varmistamiseksi tutkimuksessa täytyy kiinnittää erityistä huomioita menetelmiin artefaktin luomisen ja arvioinnin yhteydessä (Hevner ym. 2004, 87–88). Tässä tutkimuksessa artefaktin luomiseksi käytettiin kirjallisuuskatsausta ja esimerkkiprojektin seurantaa. Näin saatiin yhdistettyä teoriapohja käytännön elämään, millä varmistettiin tutkimuksen täsmällisyys ja merkityksellisyys. Arvioinnin yhteydessä havaittiin, että jatkuvan toimituksen käyttöönoton onnistumista on hankala mitata ulkoisilla, kvantitatiivisilla mittareilla. Niinpä arvioinnissa tukeuduttiin esimerkkiprojektin henkilöstöön ja heidän asiantuntijuuteensa. Parempia menetelmiä tässä yhteydessä ei löydetty. Täsmällisyyttä eniten haittaava tekijä lienee edellä mainittu vertailuprojektin puute. Koska toimintamallin toimivuutta ei voida vertailla mihinkään, on toimintamallin tehokkuutta itseasiassa mahdoton arvioida.

Suunnittelututkimus on iteratiivinen prosessi, jossa pyritään etsimään parasta ratkaisua ongelmaan kehittämällä ja testaamalla mallia syklisesti (Hevner ym. 2004, 88–89). Tässä tutkimuksessa syklisyyteen päästiin siinä mielessä, että mallin luomisen jälkeen sitä demonstroitiin ja arvioinnin perusteella jatkokehitettiin. Jatkokehitetyn mallin testaaminen jää kuitenkin tarkastelun ulkopuolelle, eli toista sykliä ei ehditty suorittaa loppuun.

Suunnittelututkimuksessa on myös tärkeää esitellä niin, että kukin yleisö ymmärtää siitä olennaiset osat. Teknologisesti orientoituneen yleisön täytyy osata tarvittaessa ottaa luotu artefakti käyttöön ja johtamisorientoituneen yleisön taas täytyy esittelyn perusteella osata arvioida artefaktin käyttöönoton yhteydessä tarvittavien resurssien määrää. (Hevner ym. 2004, 90.) Nämä asiat on pyritty kommunikoimaan tämän tutkielman avulla. Lukijan on kuitenkin hyvä ymmärtää, että jatkuvan toimituksen käyttöönottoon liittyy aina paljon

tapauskohtaisia asioita, jotka vaikuttavat vaadittaviin resursseihin huomattavasti. Niinpä jokaiseen tilanteeseen sopivia ohjeita käyttöönotosta ja vaadittavista resursseista ei voida antaa. Tutkielmassa on kuitenkin pyritty antamaan tarvittavat tiedot kulloisenkin tilanteen arviointiin.

5.3 Rajoitteet ja jatkotutkimus

Suunnittelututkimuksissa vaarana on liiallinen painottuminen kulloisenkin tilanteen ja teknologian tuomiin rajoitteisiin (Hevner ym. 2004, 98). Nämä rajoitteet tunnistetaan myös tässä tutkimuksessa. Ensimmäiseksi, teknologiat määräytyivät kohdeorganisaation valintojen perusteella, eikä valittujen työkalujen vaikutusta kontrolloitu tutkimuksessa. Eri työkalut saattavat rajata toimintamahdollisuuksia, mutta toisaalta tuoda myös täysin uusia vaihtoehtoja, joita tässä tutkimuksessa ei saatu esille. Toimintamallin hyödyntämisestä suuremmissa organisaatioissa tai erilaisen teknologiaportfolion omaavissa organisaatioissa voitaisiin saada arvokasta lisätietoa. Toisaalta tarjolla olevista teknologioista voitaisiin jo itsessään tehdä kartoittava tutkimus. Selvitys siitä, millaisia työkaluja ja teknologioita on tarjolla tarjoaisi arvokasta tietoa toimintamallin kehittämiseksi, jos työkaluilla voitaisiin ratkaista tutkimuksessa havaittuja ongelmia. Esimerkiksi keino modularisoida tietovarastoa tai säilyttää tietovaraston metadataa tietovaraston ulkopuolella tarjoaisi toimintamalliin arvokkaita lisäaskeleita.

Toiseksi tutkimustuloksia tarkasteltaessa täytyy ottaa huomioon, että kehitystiimi ja kohdeorganisaatio ovat hyvin pieniä, eikä tuloksia voida näin ollen yleistää suurempiin organisaatioihin ja tiimeihin. Alkuperäinen viitekehys taas on luotu hyvin suuren organisaation tarpeisiin (Chen 2017, 73). Jatkuvan toimituksen käyttöönotto on usein myös erityisen hankalaa nimenomaan suuremmissa organisaatioissa (Chen 2015). Niinpä lisätutkimusta tarvitaan toimintamallin hyödyntämisestä suuremmissa organisaatioissa.

Jatkotutkimusta tarvitaan myös Chenin (2017) ehdottoman julkaisuputken rungon visualisoinnin merkityksestä kehitystiimin motivaatioon, sillä tämän strategian hyödyt jäivät tässä tutkimuksessa havainnollistamatta. Muutenkin motivaatioon liittyvät seikat jäivät tässä tutkimuksessa tahattomasti tarkastelun ulkopuolella, koska jatkuvan toimituksen käyttöönotosta suoriuduttiin hyvin nopeasti. Suuremmassa organisaatiossa tehtävä tutkimus toimisi oletettavasti ratkaisuna myös motivaatioon liittyvien aiheiden käsittelyyn, sillä suuremmassa organisaatiossa käyttöönotto kestäisi todennäköisesti pidempään. Pidempään kestävässä projektissa motivaation ja erityisesti sen puutteen merkitys kasvaa. Lisätutkimusta olisi hyvä saada siis myös käyttöönottoprojektin keston suhteesta

kehitystiimin motivaatioon. Tällainen tutkimus voisi tarjota arvokasta tietoa siitä, kuinka käyttöönotto kannattaa suunnitella resurssien ja ajankäytön suhteen.

Toimintamallia voidaan myös jatkokehittää siten, että siinä tarkasteltaisiin laajemmin BI-projektien jatkuvaa toimitusta. Tässä tutkimuksessa esimerkkiprojektin tietovarastokehityksen ulkopuoliset toiminnot jätettiin kokonaan tarkastelun ulkopuolelle, mutta tutkimalla jatkuvan toimituksen hyötyjä dataintegraatioissa ja raportoinnissa, toimintamallista voitaisiin jatkokehittää kokonaisvaltainen työkalu jatkuvan toimituksen käyttöönottoon BI-projekteissa.

6 YHTEENVETO

Tässä suunnittelututkimuksessa luotiin toimintamalli jatkuvan toimituksen käyttöönottamiseksi tietovarastokehitysprojektissa. Esimerkkiprojektina käytettiin KEHA-keskuksen tietovarastointi- ja raportointiprojektia, josta keskityttiin tietovarastointipuoleen. KEHA-keskus on Suomen valtion virasto, joka vastaa ELY-keskusten ja TE-toimistojen kehittämis- ja hallinnointipalveluiden tuottamisesta (ks. KEHA-keskus). Esimerkkiprojektin tavoitteena taas on luoda KEHA-keskukselle tietovarasto, johon kerätään tietoja useista eri järjestelmistä. Näitä tietoja tullaan käyttämään lähteenä yhtenäiselle raportointiympäristölle.

Tutkimuksen heräte syntyi esimerkkiprojektin sisältä, kun tietovarastokehityksen eri julkaisuversioiden julkaisuprosessien havaittiin vievän tarpeettoman paljon kehittäjien aikaa. Ratkaisuksi ongelmaan löydettiin selvityksen perusteella jatkuvan toimituksen käyttöönotto, mutta käyttöönottoa helpottamaan ei löydetty valmista toimintamallia. Toimintamallin luominen käynnistyi kirjallisuuskatsauksella, jossa tutustuttiin jatkuvaan toimitukseen, sen käyttöönottoon sekä tietovarastoinnin erityispiirteisiin jatkuvan toimituksen kannalta oleellisilta osin. Kirjallisuuskatsauksen ohella tutustuttiin esimerkkiprojektin toimintamalleihin ja teknologiaportfolioon. Kirjallisuuskatsauksen ja esimerkkiprojektiin tutustumisen perusteella luotiin toimintamallin ensimmäinen versio. Toimintamalli on joukko strategioita, joita noudattamalla jatkuvaan toimitukseen usein liittyvät ongelmakohdat saadaan kierrettyä. Ensimmäisen version yhdeksän strategiaa on esitelty taulukossa 3.

Luotua toimintamallia demonstroitiin esimerkkiprojektissa joulukuussa 2020 ottamalla jatkuva toimitus käyttöön toimintamallin suosittelemia strategioita noudattaen. Käyttöönotosta tehtiin havaintoja, joiden avulla toimintamallia saatiin jatkokehitettyä muuttamalla strategioiden yksityiskohtia. Demonstroinnin jälkeen käyttöönoton ja toimintamallin tehokkuutta arvioitiin haastattelemalla projektin kehitystiimiä. Haastattelujen perusteella toimintamallia jatkokehitettiin edelleen ja toistaiseksi lopullinen versio esitellään taulukossa neljä. Demonstroinnin ja jatkokehityksen jälkeen toimintamalliin jäi kuusi strategiaa, joita noudattamalla jatkuvan toimituksen käyttöönotto tietovarastokehitysprojektissa helpottuu.

Toimintamallin viimeisintä versiota ei päästy vielä demonstroimaan, mutta ensimmäisen version demonstroinnin perusteella voidaan todeta, että toimintamalli helpotti jatkuvan toimituksen käyttöönottoa tutkimuksen esimerkkiprojektissa. Toimintamallin

toistaiseksi lopullinen versio ei eroa ensimmäisestä versiosta niin merkittävästi, että sen toimivuuden voitaisiin olettaa eroavan ainakaan huonompaan suuntaan. Toimintamallin kehittäminen on iteratiivinen prosessi, jonka seuraavat iteraatiot jätetään tämän tutkimuksen ulkopuolelle.

Tutkimuksen aikana havaittiin, että jatkuvan toimituksen käyttöönotto on usein haastavaa. Oikeita toimintatapoja ja teknologisia työkaluja hyödyntämällä näitä haasteita voidaan kuitenkin lieventää, jolloin jatkuvan toimituksen kiistattomat hyödyt saadaan käyttöön nopeasti ja kustannustehokkaasti. Tutkimus antaa myös näyttöä siitä, ettei jatkuvan toimituksen käyttäminen rajoitu vain ohjelmistokehitykseen vaan sitä voidaan hyödyntää myös dataintensiivisessä kehityksessä, kuten tietovarastokehityksessä.

LÄHTEET

Albrecht, J. P. (2016). How the GDPR will change the world. *European Data Protection Law Review (EDPL), 2*(3), 287–289.

Allspaw, J., & Hammond, P. (2009). 10 deploys per day: Dev and ops cooperation at Flickr. *Velocity: Web Performance and Operations Conference.*

Ang, J., & Teo, T. S. (2000). Management issues in data warehousing: Insights from the housing and development board. *Decision Support Systems, 29*(1), 11–20.

Azure DevOps dokumentaatio. <https://azure.microsoft.com/en-us/services/devops/>, haettu 17.12.2020.

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A software architect's perspective* Addison-Wesley Professional.

Bennett, K. (1995). Legacy systems: Coping with success. *IEEE Software, 12*(1), 19–23.

Chandra, P., & Gupta, M. K. (2018). Comprehensive survey on data warehousing research. *International Journal of Information Technology, 10*(2), 217–224.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software, 32*(2), 50–54.

Chen, L. (2017). Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software, 128*, 72–86.

Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology, 57*, 21–31.

Debroy, V., Miller, S., & Brimble, L. (2018). Building lean continuous integration and delivery pipelines by applying DevOps principles: a case study at Varidesk. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* 851–856.

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk* Pearson Education.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software, 33*(3), 94–100.

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

Fitzgerald, B., & Stol, K. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176–189.

Fowler, M. (2006). Continuous integration. <https://martinfowler.com/articles/continuousIntegration.html>, haettu 15.12.2020.

Fowler, M. (2013). Continuous delivery. <https://martinfowler.com/bliki/ContinuousDelivery.html>, haettu 15.12.2020.

Gill, A. Q., Loumish, A., Riyat, I., & Han, S. (2018). DevOps for information management systems. *VINE Journal of Information and Knowledge Management Systems, 48*(1), 122–139.

Herschel, Richard T. "Agile Development in Data Warehousing." Principles and Applications of Business Intelligence Research. IGI Global, 2012. 286–300.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1) , 75–105.

Hevner, A., & Chatterjee, S. (2010). Design science research in information systems. *Design research in information systems, 22,* 9–22.

Hilton, M., Nelson, N., Tunnell, T., Marinov, D., & Dig, D. (2017). Trade-offs in continuous integration: assurance, security, and flexibility. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,* 197–207.

Humble, J. (2018). Continuous delivery sounds great, but will it work here? *Communications of the ACM, 61*(4), 34–39.

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation.* Pearson Education.

Humble, J., & Molesky, J. (2011). Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal, 24*(8), 6–12.

Järvinen, P. (2006). Onko innovaatioiden suunnittelu tiedettä. *Systeemityö, 2006*(2), 25–27.

KEHA-keskus. <https://www.keha-keskus.fi/>, haettu 15.12.2020

Krneta, D., Jovanovic, V., & Marjanovic, Z. (2016). An approach to data mart design from a data vault. *INFOTEH-Jahorina BiH, 15, 473–478.*

Kuhn, T. S. (1970). *The structure of scientific revolutions.* University of Chicago press.

Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology, 82*, 55–79.

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V., Itkonen, J., Mäntylä, M. V., et al. (2015). The highways and country roads to continuous deployment. *IEEE Software, 32*(2), 64–72.

Linstedt, D., & Olschimke, M. (2015). *Building a scalable data warehouse with data vault 2.0.* Morgan Kaufmann.

Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A., Laukkanen, E., Pagels, M. (2016). Improving the delivery cycle: A multiple-case study of the toolchains in finnish software intensive enterprises. *Information and Software Technology, 80*, 175–194.

Manos, A. (2007). The benefits of kaizen and kaizen events. *Quality Progress, 40*(2), 47–48.

Meyer, M. (2014). Continuous integration and its tools. *IEEE Software, 31*(3), 14–16.

Moody, D. L., & Kortink, M. A. (2000). From enterprise models to dimensional models: a methodology for data warehouse and data mart design. *DMDW,* 5, 1–12.

Neely, S., & Stolt, S. (2013). Continuous delivery? easy! just change everything (well, maybe it is not that easy). *2013 Agile Conference,* 121–128.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45–77.

Puonti, M., Järvi, J., & Mikkonen, T. (2018). A continuous delivery framework for business intelligence. *Information Modelling and Knowledge Bases XXIX,* 248–259.

Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology, 108*, 65–77.

Rahman, N., Rutz, D., & Akhter, S. (2013). Agile development in data warehousing. *Principles and applications of business intelligence research*, 286–300.

Rathod, N., & Surve, A. (2015). Test orchestration a framework for continuous integration and continuous deployment. *2015 international conference on pervasive computing (ICPC),* 1–5.

Rossi, C., Shibley, E., Su, S., Beck, K., Savor, T., & Stumm, M. (2016). Continuous deployment of mobile software at Facebook (showcase). *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 12–23.

Schein, E. H. (1999). *The corporate culture survival guide.* John Wiley & Sons.

Schwaber, K., & Sutherland, J. (2011). The scrum guide. *Scrum alliance, 21,* 1–19.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering, 25*(4), 557–572.

Seddon, P. B., Constantinidis, D., Tamm, T., & Dod, H. (2017). How does business analytics contribute to business value? *Information Systems Journal, 27*(3), 237–269.

Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access, 5*, 3909–3943.

Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering, 24*(3), 1061–1108.

Simon, H. A. (1969). *The sciences of the artificial.* MIT press.

Siqueira, R., Camarinha, D., Wen, M., Meirelles, P., & Kon, F. (2018). Continuous delivery: Building trust in a large-scale, complex government organization. *IEEE Software, 35*(2), 38–43.

Soni, M. (2015). End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery. *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM),* 85–89.

SQL server dokumentaatio. <https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/data-tier-applications?view=sql-server-ver15>, haettu 28.12.2020,

Stake, R. E. (1995). *The art of case study research*. Thousand Oaks, California: SAGE Publications.

Turner III, D. W. (2010). Qualitative interview design: A practical guide for novice investigators. *The Qualitative Report, 15*(3), 754–760.

Utuguides. <https://utuguides.fi/az.php?a=s>, haettu 5.11.2020

Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. *Fifth International Conference on the Innovative Computing Technology (INTECH 2015),* 78–82.

Weill, P., Subramani, M., & Broadbent, M. (2002) Building IT Infrastructure for Strategic Agility. *MIT Sloan management review 44*(1), 57–65.

Westrum, R. (2004). A typology of organisational cultures. *BMJ Quality & Safety, 13*(2), 22–27.

Zarsky, T. Z. (2016). Incompatible: The GDPR in the age of big data. *Seton Hall L.Rev., 47*, 995–1020.

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and its practices. *IEEE Software, 33*(3), 32–34.

LIITEET

Liite 1. Toisen haastattelukierroksen haastattelukysymykset

Hakasuluissa on merkitty liitoskohta luodun toimintamallin numerointiin. Kysymykset 11–13 on tarkoitettu toimintamallin arviointiin kokonaisuudessaan.

1) Mitkä työkalut (ohjelmistot) helpottivat jatkuvan toimituksen käyttöönottoa? [1]
2) Olisiko jatkuvan toimituksen käyttöönottoa voitua helpottaa, jos saatavilla olisi ollut muita työkalulla? [1]
3) Miten Data Vault -mallinnustavan käyttäminen vaikutti jatkuvan toimituksen käyttöönottoon? [2, 3]
4) Modularisoinnilla tarkoitetaan arkkitehtuurin rakentamista niin, että osat toimivat omina kokonaisuuksinaan, eivätkä muutokset yhteen osioon vaikuta muiden osien toimintoihin. Esimerkiksi projektissa SSAS-kuutiot olivat modularisoitu omiksi kokonaisuuksiinsa. Eräänlaista modularisaatiota on myös skeemojen käyttäminen datamarteissa.
    a) Kuinka kuutioiden ja datamarttien modularisointi vaikutti julkaisuputkien käyttöönottoon? [3]
    b) Mitä muita osia, kuutioiden ja datamarttien lisäksi, tietovarastosta voitaisiin modularisoida? [3]
5) Kuinka kommunikaatio sujui jatkuvan toimituksen käyttöönoton aikana kehitystiimin sisällä? [4]
6) Miten kommunikaatiota olisi voitu parantaa? [4]
7) Julkaisuputkien kehittäminen oli jätetty pääasiallisesti yhden henkilön tehtäväksi.
    a) Oliko tämä järkevää? [6]
    b) Olisiko tehtävä vaatinut enemmän resursseja? [6]
    c) Häiritsikö muu kehitystyö julkaisuputkien kehittämistä? [6]
    d) Millaista osaamista julkaisuputkien kehittämisessä tarvittiin/olisi vielä tarvittu? [6]
8) Mitkä tekijät nostivat motivaatiota jatkuvan toimituksen käyttöönottamiseksi? [5]
9) nostivat motivaatiota jatkuvan toimituksen käyttöönoton aikana? [5, 7, 8, 9]
10) Mitkä tekijät laskivat motivaatiota jatkuvan toimituksen käyttöönoton aikana? [5, 7, 8, 9]
11) Asteikolla 0-5, jossa 0 = ”ei lainkaan” ja 5 = ”täydellisesti”
    a) Kuinka hyvin jatkuvaa toimitusta käytettiin projektissa ennen julkaisuputkien kehitystä?
    b) Kuinka hyvin jatkuvaa toimitusta käytetään projektissa nyt?

12) Miten jatkuvan toimituksen käyttöönotto olisi voinut onnistua paremmin?
13) Mitä jatkuvan toimituksen käyttöönotossa onnistui erityisen hyvin?