

Opinnäytetyö (YAMK)

Teknologiaosaamisen johtaminen

2020

Tero Elmroos

# YRITYKSEN TUOTEKEHITYKSEN KEHITTÄMINEN DEVOPS- TOIMINTAMALLIN AVULLA

– Case: QEM Software Oy



Tero Elmroos

# YRITYKSEN TUOTEKEHITYKSEN KEHITTÄMINEN DEVOPS-TOIMINTAMALLIN AVULLA

- Case: QEM Software Oy

Opinnäytetyössä tarkasteltiin ja verrattiin toimeksiantajayrityksen tuotekehityksen toimintaa DevOps-toimintamalliin. Työn tavoitteena oli vertailun avulla selvittää tarvittavat kehitystoimenpiteet, jotta tuotekehityksen toiminta saadaan vastaamaan DevOps-toimintamallia. Työssä ei käsitelty miten toimenpiteet suoritetaan, vaan keskityttiin pelkästään tarvittaviin toimenpiteisiin. Opinnäytetyö rajattiin toimeksiantajan infonäyttöohjelmistoon, jotta työmäärä ei kasvanut liian suureksi ja kokonaisuus pystyttiin pitämään hallittuna.

Opinnäytetyössä tutkittiin toimeksiantajayrityksen tuotekehitystä laadullisen tutkimuksen toimintatutkimussuuntauksen menetelmien avulla. Tiedonkeruumenetelminä käytettiin haastattelua, dokumenttianalyysiä ja kirjoittajan omaa kokemusta toimeksiantajayrityksen tuotekehityksen toiminnasta. Nykyinen tuotekehityksen toiminta käytiin läpi DevOps-toimintamallin toimitusputken osien avulla toiminnan esittämiseksi mahdollisimman selkeästi. Kehitysehdotuksia etsittiin kuitenkin DevOps-toimintamallin käytäntöjen avulla, johtuen toimintamallin perustumisesta jatkuvien käytäntöjen yhdistämiseen ja toteuttamiseen.

Toimeksiantajayrityksen tuotekehityksen toiminnasta saatiin muodostettua DevOps-toimintamallin toteuttamiseen vaadittavia kehitysehdotuksia. Lisäksi tuotekehityksen toiminnan kehittämiseen löydettiin useita yleisesti toimintaa hyödyttäviä kehitysehdotuksia, joiden pääpainona oli valmistaa toimeksiantajayritystä sen kasvua varten. Opinnäytetyössä käsiteltiin kehitysehdotuksia hyvin yleisellä tasolla, jotta niiden hyödyntäminen toimeksiantajayrityksen muun tuotekehityksen jatkokehityksessä olisi mahdollisimman helppoa.

## ASIASANAT:

DevOps, jatkuva integraatio, jatkuva käyttöönotto, jatkuva palaute, jatkuva toimitus, tuotekehitys.

MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Master's Degree Programme in Technological competence management

2020 | 60 pages, 7 pages in appendices

Tero Elmroos

# IMPROVING A COMPANY'S PRODUCT DEVELOPMENT USING DEVOPS PRACTICES

- Case: QEM Software Ltd.

The present Master's thesis examines and compares the product development activities of the client company with the DevOps practices. The aim of the thesis is to identify the necessary development measures to make the product development activities correspond to those of the DevOps practices. The study focuses only on the necessary measures and it does not address the ways to execute the measures. The scope of the thesis is limited to the client company's digital signage software in order to ensure that the workload does not grow too great and it can be kept under control.

The product development of the client company was studied using the qualitative methods of action research. The data collection methods used were interviews, document analysis and the author's own experience of the client company's product development activities. The current product development activities were studied using the components of the DevOps delivery pipeline in order to present the activities as clearly as possible. Suggestions for development measures were sought through the practices of DevOps because, DevOps is based on the integration and implementation of continuous practices.

The thesis reached the goal of generating suggestions for development measures required for the implementation of the DevOps practices for the product development activities of the client company. In addition, several suggestions for developing general activities are presented to improve the product development operations. The suggestions for developing general activities focus mainly on preparing the client company for growth. The thesis discusses the development suggestions at a very general level to make the utilization of the suggestions as easy as possible in the further development of the client company's product development.

## KEYWORDS:

Continuous delivery, continuous deployment, continuous feedback, continuous integration, DevOps, product development.

# SISÄLTÖ

<b>KÄYTETYT LYHENTEET</b>	<b>6</b>
<b>1 JOHDANTO</b>	<b>7</b>
<b>2 DEVOPS-TOIMINTAMALLI</b>	<b>9</b>
2.1 Toimintamallin määritelmä	10
2.2 Hyödyt ja haasteet	13
2.3 Kulttuuri, automaatio, mittaus ja jakaminen	15
2.4 Toimitusputki ja sen osat	20
2.5 Jatkuvat käytännöt	25
2.5.1 Jatkuva integraatio	26
2.5.2 Jatkuva toimitus ja jatkuva käyttöönotto	35
2.5.3 Jatkuva palaute	40
<b>3 TUTKIMUKSEN TOTEUTUS</b>	<b>42</b>
3.1 Dokumenttianalyysin toteutus	42
3.2 Haastattelun toteutus	43
3.3 Tutkimuksen luotettavuus	43
<b>4 TOIMEKSIANTAJAN TUOTEKEHITYKSEN NYKYTOIMINTA</b>	<b>45</b>
<b>5 KEHITYSEHDOTUKSET TOIMEKSIANTAJAN TUOTEKEHITYKSEEN</b>	<b>48</b>
5.1 Jatkuva integraatio	48
5.2 Jatkuva toimitus ja jatkuva käyttöönotto	52
5.3 Jatkuva palaute	54
<b>LÄHTEET</b>	<b>57</b>

## LIITTEET

- Liite 1. Haastattelurunko.
- Liite 2. Suunnitteluvaiheen prosessikaavio.
- Liite 3. Ohjelmointivaiheen prosessikaavio.
- Liite 4. Kääntämisvaiheen prosessikaavio.
- Liite 5. Testausvaiheen prosessikaavio.
- Liite 6. Julkaisu- ja käyttöönottovaiheiden prosessikaavio.

Liite 7. Käyttö- ja valvontavaiheiden prosessikaavio.

## KUVAT

Kuva 1. DevOps kannustaa kehityksen ja ylläpidon yhteistyöhön (Lwakantare 2017, 19).	9
Kuva 2. Epäselvyyden seinä (Edwards 2010b).	13
Kuva 3. CAMS-malli (Stafford 2017).	16
Kuva 4. Ohjelmistokehityksen ja järjestelmäylläpidon yhteistyö (Wilsenach 2015).	17
Kuva 5. Mahdollinen DevOpsin toimitusputki (Kornilova, 2017).	21
Kuva 6. Jatkuvat käytännöt DevOpsin toimitusputkessa (Pennington 2019).	26
Kuva 7. Jatkuvan integraation toiminta (Pepgotesting 2019).	27
Kuva 8. Julkaisuhaarat ja ryhmähaarat (Humble & Farley 2010, 413).	30
Kuva 9. Blue-green-julkaisu (Humble & Farley 2010, 261).	38
Kuva 10. Canary-julkaisu (Humble & Farley 2010, 263).	39

## TAULUKOT

Taulukko 1. Ehdotetut määritelmät DevOpsille (Lwakatere 2017, 27).	11
--	----

## KÄYTETYT LYHENTEET

CAMS-malli	Culture, Automation, Measurement, Sharing. DevOpsiin liittyvät keskeiset osat. (Perera ym. 2017).
KPI-mittari	Key Performance Indicator. Suorituskykymittari, jonka avulla seurataan toiminnan kannattavuutta ja kehitystä. (Edwards 2009a).
TOC-teoria	Theory of Constraints. Ajattelutapa, jonka avulla pyritään hallitsemaan asioiden tavoitteiden täyttymisen esteitä. (Zisliis 2018).

# 1 JOHDANTO

Opinnäytetyön toimeksiantajayritys toimii ohjelmistojen suunnittelu ja valmistus -toimialalla. Yritys erikoistuu digitaalisten informaationäyttöjen ohjelmistoihin ja päätuotteenaan toteuttaa asiakkaille mainos- ja informaationäyttöratkaisuja. Digitaalisten ratkaisujen avulla asiakkaat pystyvät tavoittamaan omat asiakkaansa sekä hallinnoimaan informaatio- ja mainoskylttejään perinteisiä staattisia ratkaisuja, kuten paperikylttejä, paremmin.

Toimeksiantajayritys on perustettu vuonna 2008 Suomessa ja sen infonäyttöjärjestelmäpalvelulle on myönnetty vuonna 2019 Avainlippu-merkki, joka merkitsee palvelun olevan tuotettu Suomessa. Yrityksessä työskentelee kuusi henkilöä ja se on hierarkialtaan matala, jolloin jokaisella työntekijällä on itsellään vastuu omista työtehtävistä sekä projekteista. Lisäksi yrityksen toimitusjohtaja ja teknologiajohtaja ovat läheisesti mukana yrityksen tuotekehityksen järjestelmäylläpidossa sekä toimivat työnjohtajien rooleissa ohjelmistokehityksessä.

Yritys tekee yhteistyötä useiden elektroniikkavalmistajien, -jälleenmyyjien ja sisällöntuottajien kanssa mahdollistaakseen hyvän ja helpon käyttökokemuksen asiakkailleen. Yrityksen asiakkaina on monenlaisia yrityksiä useilta eri toimialoilta, kuten esimerkiksi ravintola-alalta sekä julkiselta sektorilta, joita kuitenkin yhdistää yksi tarve: viestiä informaatiota omille asiakkailleen.

Digitaaliset informaationäyttöratkaisut, eli digital signage -ratkaisut, ovat paperikylttien moderneja versioita, jotka poistavat perinteisten ratkaisujen logistiikkakustannukset ja tuovat animoidun kuvan avulla WOW-efektin tiedonvälitykseen. Välitettävänä tietona voi esimerkiksi olla yrityksissä erilaiset raportit tai ravintoloissa tarjouksessa olevat ruokannokset. Nämä pystytään päivittämään napin painalluksella ja jopa automatisoimaan näyttämään eri informaatiota tiettyinä päivinä tai kellonaikoina, minkä avulla taas pystytään säästämään työaika.

Tämä opinnäytetyö keskittyy toimeksiantajayrityksen tuotekehityksen kehittämiseen DevOps-toimintamallin avulla. Yrityksen tuotekehityksessä on jo pääosin käytössä ketterät menetelmät, jolloin sen kehittämisen seuraava askel on tutkia ohjelmistokehityksen ja järjestelmäylläpidon yhteistyötä DevOps-toimintamallin avulla sekä selvittää vaaditut toimenpiteet ja muutokset toimintamallin käyttöönottoon.

DevOps-toimintamalli keskittyy yrityksen ohjelmistokehityksen ja järjestelmäylläpidon lähentämiseen, johon usein pyritään joko yrityksen kulttuurin muuttamisella tai DevOps-toimintamallissa käytettävien toimintamenetelmien käyttöönotolla yrityksen prosesseissa. Yrityksen kulttuurin muuttaminen on usein ensisijainen ratkaisu suuremmissa yrityksissä, joissa kehitys ja ylläpito ovat erilliset osastot omilla työntekijöillään. Pienemmissä yrityksissä, kuten opinnäytetyön toimeksiantajayrityksessä, ohjelmistokehitys ja järjestelmäylläpito ovat kuitenkin usein yhdistetty ja samojen työntekijöiden vastuulla. Tästä syystä opinnäytetyössä keskitytään DevOps-toimintamallin mahdollistaviin toimintamenetelmiin, eikä niinkään yrityksen kulttuurin muuttamiseen.

Tutkimus rajataan opinnäytetyön toimeksiantajayrityksen infonäyttöohjelmisto-tuotteeseen ja sen tuotekehitykseen, jota tarkoitetaan tässä työssä yrityksen tuotekehityksestä puhuttaessa. Työ rajataan vain yhteen tuotteeseen, jotta suoritettavasta tutkimuksesta ei tule liian laajaa. Tutkimuksen jälkeen voidaan jatkaa DevOps-toimintamallin toteutusta myös yrityksen muuhun tuotekehitykseen, jossa taas pystytään hyödyntämään tästä tutkimuksesta opittuja asioita toimintamallin käytön laajentamisessa.

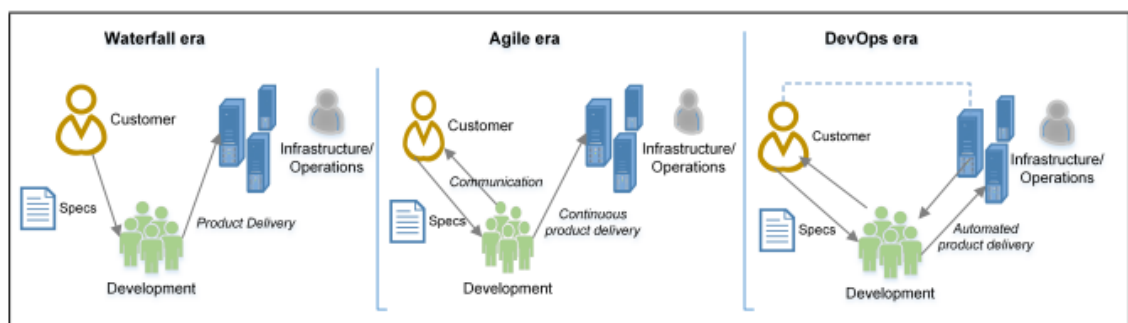
Opinnäytetyön tarkoituksena on tutkia laadullisen tutkimuksen toimintatutkimussuuntauksen menetelmien avulla toimeksiantajayrityksen tuotekehityksen tämänhetkistä tilaa. Tutkimuksen tiedonkeruumenetelminä käytetään kirjoittajan omaa kokemusta yrityksen tuotekehityksestä ja siitä tehtyä dokumentointia sekä yrityksen tuotekehitykseen liittyville työntekijöille toteutettavia haastatteluja. Tutkimuksesta saatujen tuloksien avulla tehdään kehitysehdotukset yrityksen tuotekehityksen tulevaisuuden kehityssuunnasta ja sen mahdollistavista toimenpiteistä. Pohjimmiltaan opinnäytetyön tavoitteena on kehittää toimeksiantajayrityksen tuotekehitystä vastaamaan koko ajan muuttuvan ja kehittyvän ohjelmistoalan tarpeita.



## 2 DEVOPS-TOIMINTAMALLI

Ohjelmistotuotantoprosesseissa käytettyjen perinteisten menetelmien, kuten vesiputouksmallin, puutteita paikkaamaan kehitettyjen ketterien menetelmien suosio on kasvanut huomattavasti niiden kehityksen jälkeen. Ketterät menetelmät hyödyntävät iteroivaa ja inkrementaalista kehitysmallia, joka mahdollistaa projektin suunnanmuutoksen hallitusti kesken projektin. Projektin hallittu suunnanmuutos taas mahdollistaa huomattavasti suuremman yhteistyön asiakkaiden kanssa, sillä toisin kuin perinteisissä menetelmissä, projektin jokaista yksityiskohtaa ei tarvitse suunnitella täysin ensimmäisissä asiakastapaamisissa, vaan asiakas pystyy helposti vaikuttamaan projektiin sen edetessä. (Canty 2015, 2–3; Perera ym. 2017).

Ketterät menetelmät pienentävät tehokkaasti asiakkaiden ja kehityksen välistä kuilua, mutta ne eivät kuitenkaan ota kantaa ohjelmistokehityksen ja järjestelmäylläpidon väliseen keskusteluun. Tähän on kuitenkin viime vuosina kehitetty DevOps-toimintamalli, jonka avulla pyritään lähentämään kehitystä ja ylläpitoa. Ohjelmistokehityksen ja järjestelmäylläpidon lähentämisen sekä niiden yhteistyön parantamisen avulla mahdollistetaan ohjelmiston nopea ja luotettava julkaisu sekä nopeampi arvon tuottaminen asiakkaalle. Kuvassa 1 on esitetty eri menetelmien sisältämiä vuorovaikutuksia ohjelmistotuotannossa. (Perera ym. 2017; Sharma & Coyne 2017, 5–6).



Kuva 1. DevOps kannustaa kehityksen ja ylläpidon yhteistyöhön (Lwakantare 2017, 19).

DevOps-toimintamalli pohjautuu useihin eri ajattelumalleihin, teorioihin ja järjestelmiin, kuten Lean-ajatteluun, TOC-teoriaan ja Toyotan tuotantojärjestelmään. Monissa yrityksissä kehittyikin ja oli käytössä osia toimintamallin periaatteista ennen sen varsinaista syntymistä, johon kuitenkin vaadittiin kaikkien näiden yksittäisten osien yhteen tuominen.

Tätä ilmiötä on DevOpsin kehittymiseen vaikuttanut John Willis kuvannut DevOpsin lähtymisenä. (Kim ym. 2016, 3-4).

DevOpsin muotoutuminen nykyiseen muotoonsa lähti liikkeelle vuonna 2009 pidetyn O'Reilly Velocity -konferenssin jälkeen, jossa useat informaatioteknologian ammattilaiset huomasivat muidenkin alkaneen pohtimaan ratkaisuja erinäisiin ohjelmistokehityksen ja järjestelmäylläpidon välisiin ongelmiin (Kim ym. 2016, preface xii–xiv). Näistä ammattilaisista Patrick Debois päätti järjestää vielä samana vuonna Belgiassa ensimmäisen Devopsdays-konferenssin (Devopsdays 2019). Konferenssin pohjalta toimintamalli saiikin vahingossa nimityksen DevOps. (Kim ym. 2016, preface xii). DevOps-nimitys taas on lyhennelmä sanoista ohjelmistokehitys (engl. development) ja järjestelmäylläpito (engl. operations) (Jabbari ym. 2016).

DevOps-toimintamallin päätavoitteena on hävittää ohjelmistokehityksen ja järjestelmäylläpidon välinen kuilu, joka johtuu siilomaisesta ajattelumallista. Tämä pyritään täyttämään neljän pienemmän tavoitteen kautta:

1. Jatkuvan ja korkealaatuisen palvelun toimituksen avulla mitattavan liiketoiminnan arvon korostaminen.
2. Yksinkertaisuuden ja ketterän kehityksen hyödyntäminen tuotekehityksen kaikilla alueilla.
3. Luottamuksen, innovaation ja jaetun omistuksen kautta esteiden poistaminen kehityksen ja ylläpidon väliltä.
4. Dynaamisten muuttujien hallinta tuotekehityksessä.

Näiden tavoitteiden täytyminen ei kuitenkaan merkitse kaiken toimivan täydellisesti koko ajan, vaan jatkuvaa yhteistyötä vaaditaan jokaiselta yrityksen tuotekehitykseen osallistavalta taholta. (Perera ym. 2017).

## 2.1 Toimintamallin määritelmä

Toimintamallilla ei ole varsinasta määritelmää eikä viitekehystä vaan sen toteutus ja määrittely saattavat vaihdella paljonkin eri organisaatioiden ja sen toteuttajien välillä (Lwakatare 2017, 19). Toisin kuin esimerkiksi ketterässä ohjelmistokehityksessä on manifesti, jossa määritetään ketterän ohjelmistokehityksen periaatteet, DevOpsille ei ole tehty manifestia (Agile Manifesto 2001; Little 2016). DevOps-toimintamallin manifestin puuttuminen johtuu DevOps-yhteisön avainhenkilöiden haluttomuudesta tehdä sitä, sillä

heidän mukaansa sen olemassaolo saisi ihmiset hakemaan ratkaisuja manifestista sen sijaan, että he pyrkisivät työskentelemään yhdessä ongelmien ratkaisemiseksi (Little 2016).

DevOpsin virallisen määritelmän puuttuminen taas johtuu luultavimmin sen yhteisöllisistä juurista sekä manifestin puuttumisesta. Toimintamallille on kuitenkin ehdotettu useampia, eri tavoitteisiin keskittyviä, määritelmiä, joista osa on esitetty taulukossa 1. Näistä ehdotuksista ei ole kuitenkaan saatu valittua yleistä määritelmää DevOpsille, sillä yhteisön sisällä ei olla päästy riittävään yksimielisyyteen oikeasta ja sille sopivasta määritelmästä. (Panko 2017).

Taulukko 1. Ehdotetut määritelmät DevOpsille (Lwakatare 2017, 27).

Lähde	DevOpsin määritelmä	Määritelmän painotus
Bass ym. 2015	<i>“DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production while ensuring high quality”</i>	Goal-oriented (fast delivery of quality software)
Dyck ym. 2015	<i>“DevOps is an organisational approach that stresses empathy and cross-functional collaboration within and between teams – especially development and IT operations – in software development organisations, in order to operate resilient systems and accelerate the delivery of changes”</i>	Means-oriented (empathy, cross-functional collaboration); and goal-oriented (operate resilient systems, accelerate change delivery)
Penners & Dyck 2015	<i>“DevOps is a mindset, encouraging cross-functional collaboration between teams – especially development and IT operations – within a software development organisation, in order to operate resilient systems and accelerate the delivery of changes”</i>	Means-oriented (empathy, cross-functional collaboration); and goal-oriented (operate resilient systems, accelerate change delivery)

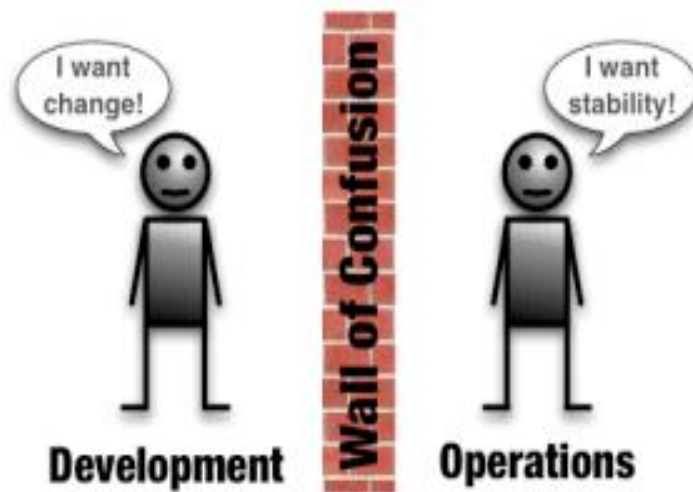
(jatkuu)

Taulukko 1 (jatkuu).

Lähde	DevOpsin määritelmä	Määritelmän painotus
Smeds ym. 2015	<i>“A set of engineering process capabilities supported by certain cultural and technological enablers”</i>	<i>Means-oriented (engineering capabilities)</i>
de França ym. 2016	<i>“DevOps is a neologism, representing a movement of ICT professionals addressing a different attitude regarding software delivery through the collaboration between software systems development and operations functions, based on a set of principles and practices, such as culture, automation, measurement and sharing”</i>	<i>Means-oriented (attitude, cross-functional collaboration)</i>
Jabbari ym. 2016	<i>“DevOps is a development methodology aimed at bridging the gap between Development and Operations, emphasising communication and collaboration, continuous integration, quality assurance and delivery with automated deployment, utilising a set of development practices”</i>	<i>Means-oriented (cross-functional collaboration, automated deployment)</i>

Ehdotetut määritelmät pystytään jakamaan usein kahteen kategoriaan: toimintatapa- tai tavoite-painotteisiin. Toimintatapa-painotteiset määritelmät painottavat tiettyjä toimintatapoja, erityisesti yhteistyötä ja ymmärrystä, kun taas tavoite-painotteiset määritelmät painottavat laadukkaan ohjelmiston toimituksen nopeutta ja sujuvuutta. Ehdotetut määritelmät voivat kuitenkin myös sopia molempiin kategorioihin. (Lwakatare 2017, 28).

Käytännössä lähes jokaisessa DevOpsille ehdotetussa määritelmässä kuvataan ohjelmistokehityksen ja järjestelmäylläpidon yhteistyön parantamista, jolloin ideaalisessa DevOpsin toteutuksessa kehitys ja ylläpito toimivatkin täydessä harmoniassa keskenään. Tällöin niin sanottu epäselvyyden seinä (engl. wall of confusion), jossa ohjelmistokehityksellä ja järjestelmäylläpidolla on täysin eri tavoitteet, on saatu purettua ja on siirrytty siilomaisesta ajattelumallista yhteisen tavoitteen tavoittelemiseen, käyttäen yhteisiä työkaluja. Kuvassa 2 esitetään epäselvyyden seinää. (Edwards 2009b).



Kuva 2. Epäselvyyden seinä (Edwards 2010b).

DevOpsista on myös ajan myötä tehty useita eri mukautuksia, joiden tarkoituksena on kohdistaa DevOps-ratkaisu johonkin tiettyyn osa-alueeseen. Näitä mukautuksia voi olla esimerkiksi SecOps, DevSecOps tai DevTestOps. Mukautukset taas voivat hankaloittaa hyväksytyn DevOpsin määritelmän syntymistä lisäämällä uusia asioita DevOpsiin, tai helpottaa sitä pienentämällä tarvittavan määritelmän laajuutta. Tässä työssä keskitytään kuitenkin DevOpsin toteutukseen ilman eri mukautuksia. (Lietz 2015; Jawale 2017; Chokshi 2019).

## 2.2 Hyödyt ja haasteet

DevOpsilla pyritään samaan paljon hyötyjä ja samalla pitämään haasteet pieninä sekä helposti voitettavina. Vuonna 2016 DevOpsin käyttöönoton tuomista hyödyistä ja haasteista tehty tutkimus määrittelee saavutetuiksi hyödyiksi lyhentyneen julkaisusyklin, parantuneen laadun ja ohjelmistokehityksen sekä järjestelmäylläpidon parantuneen yhteistyön. DevOpsin käyttöönoton haasteiksi taas tutkimus löysi heikon kommunikoinnin, roolien muuttumisen ja DevOpsin epäselvän määritelmän. (Riungu-Kalliosaari ym. 2016).

DevOpsin avulla julkaisujen julkaisuväliä pystytään lyhentämään viikoista jopa alle tuntiin. Uudet ominaisuudet ja korjaukset pystytään viemään saumattomasti heti tuotantoon sen sijaan, että ensin kerättäisiin useampi ominaisuus tai korjaus ja sen jälkeen tehtäisiin

yksi suurempi päivitys. Uusien ominaisuuksien ja korjausten yksittäinen julkaiseminen pienentää myös riskiä ja virheiden korjauksen aikaa, kun käsiteltävänä on pienempi määrä ohjelmistokoodia. Lisäksi DevOpsin automaatio mahdollistaa automaattisen julkaisun, joka taas lyhentää julkaisuun kuluvaan aikaa. (Riungu-Kalliosaari ym. 2016; Lwakantare 2017, 41–42).

Parantunut laatu saavutetaan vaatimalla automaattisessa julkaisussa myös automaattista testausta, jolloin useimmat ohjelmointivirheet löydetään jo ennen kuin julkaisupaketti on päässyt tuotantoon. Tämä vähentää ja mahdollisesti jopa poistaa inhimilliset virheet ohjelmistotestauksesta, mikäli perinteistä henkilön tekemää ohjelmistotestausta ei toteuteta automaattisen testauksen ohella. Automaattisen testauksen lisäksi DevOps lisää jo tuotannossa olevien julkaisujen seuranta ja siten mahdollistaa nopean virheisiin reagoimisen. (Riungu-Kalliosaari ym. 2016; Lwakantare 2017, 43).

DevOps lisää ohjelmistokehityksen ja järjestelmäylläpidon yhteistyötä, joka johtaa onnistumisten kautta vahvempaan yhteistyön kulttuuriin yrityksessä. Tämä taas auttaa hävittämään perinteistä siilomaista ajattelumallia, jossa kapeasti nähdään vain oman osaston tehtävät eikä lopputuotetta. Siilomaisuuden rikkoutuminen ja parantunut yhteistyö mahdollistavat myös avoimen tiedon ja kokemusten jakamisen, jolloin yleinen sekä virheistä oppiminen parantuvat. (Riungu-Kalliosaari ym. 2016; Lwakantare 2017, 43).

Heikko kommunikointi voi vaikuttaa esimerkiksi ohjelmiston toiminnan mittaamiseen tai virheisiin reagointiin. Tällöin ei välttämättä osata kerätä merkitsevää tietoa ohjelmiston toiminnasta tai kerättyä tietoa ei välttämättä välitetä eteenpäin oikeille henkilöille. Toisaalta voidaan olla myös pääsemättä sopuun mitattavista asioista tai halutaan mitata toistensa kanssa ristiriidassa olevia tietoja, kuten esimerkiksi palvelimien käyttöaikaa ja julkaisujen tiheyttä järjestelmässä, jossa julkaisut aiheuttavat käyttökatkoksia. Lisäksi sähköisten viestintävälineiden käyttö kasvokkain keskustelun sijaan voi aiheuttaa virheiden reagointiin turhia viiveitä. (Riungu-Kalliosaari ym. 2016).

DevOps muuttaa ohjelmistokehityksen ja järjestelmäylläpidon vastuita, mikä johtaa jo muodostuneiden roolien uudelleen rakentumiseen. Perinteisesti DevOpsin toteutuksessa järjestelmäylläpidon tehtäviä, kuten päivystyksiä, laajennetaan ohjelmistokehitykselle, joka taas saattaa aiheuttaa kehityksen henkilöstössä vastustusta. Lisäksi järjestelmäylläpidon henkilöstö saattaa puolustaa omaa reviiriään ja vastustaa tehtävien jakoa ohjelmistokehitykselle. Muutoksen vastustaminen riippuu henkilöstöstä ja heidän

luottamuksesta toisiinsa sekä yrityksen johtoon. (Riungu-Kalliosaari ym. 2016; Lwakantare 2017, 44–46).

DevOps-toimintamallille ei ole kehitetty virallista määritelmää. Useissa eri määritelmissä on kuitenkin joitain samoja tunnusmerkkejä, mutta niiden ydin on usein epämääräinen. Tämä ja useat eri määritelmät johtavat helposti väärinkäsityksiin siitä, mitä oikeasti halutaan toteuttaa ja saada aikaan. Väärinkäsitykset taas luovat vääriä odotuksia, jotka aiheuttavat sekasortoa ja turhautumista. DevOpsin määritelmän lisäksi sille kehitetään jatkuvasti uusia työkaluja ja siihen liitetään uusia toimintatapoja, jolloin DevOps myös kehittyy ja muuttuu koko ajan. (Riungu-Kalliosaari ym. 2016; Lwakantare 2017, 44).

### 2.3 Kulttuuri, automaatio, mittaus ja jakaminen

Lähes jokaisessa DevOpsille ehdotetussa määritelmässä painotetaan ohjelmistokehityksen ja järjestelmäylläpidon lähentämistä, joka onkin DevOpsin päätavoite. Lähentämisen toteutustavat ovat kuitenkin toteuttajasta riippuvaisia. DevOps-pohjaisessa ohjelmistokehityksen ja järjestelmäylläpidon lähentämisessä voidaan kuitenkin puhua DevOpsiin liittyvistä keskeisistä osista, joita voidaan käyttää yleisinä periaatteina DevOps-pohjaisissa kehityksen ja ylläpidon lähentämisen toteutuksissa. (Lwakantare 2017, 29).

DevOpsin kehittämiseen osaltaan vaikuttaneet John Willis ja Damon Edwards loivat vuonna 2010 kuvassa 3 esitetyn CAMS-mallin kuvaamaan DevOpsin keskeisiä osia. Lyhenteen kirjaimet koostuvat sanoista Culture (Kulttuuri), Automation (Automaatio), Measurement (Mittaus) ja Sharing (Jakaminen). (Seppä-Lassila 2017, 5; Aljundi 2018, 18). Myöhemmin lyhenteeseen on myös erinäisten henkilöiden toimesta lisätty muita osia, mikä jälleen osoittaa DevOpsin olevan hyvin monimuotoinen ja sitä tarkastelevasta henkilöstä riippuvainen (Seppä-Lassila 2017, 5). Usein kuitenkin DevOpsin keskeisistä osista puhutaan kirjallisuuslähteissä suoraan tai epäsuorasti CAMS-lyhenteen osilla ilman siihen myöhemmin lisättyjä liitteitä, kuten toimitaan myös tässä työssä.



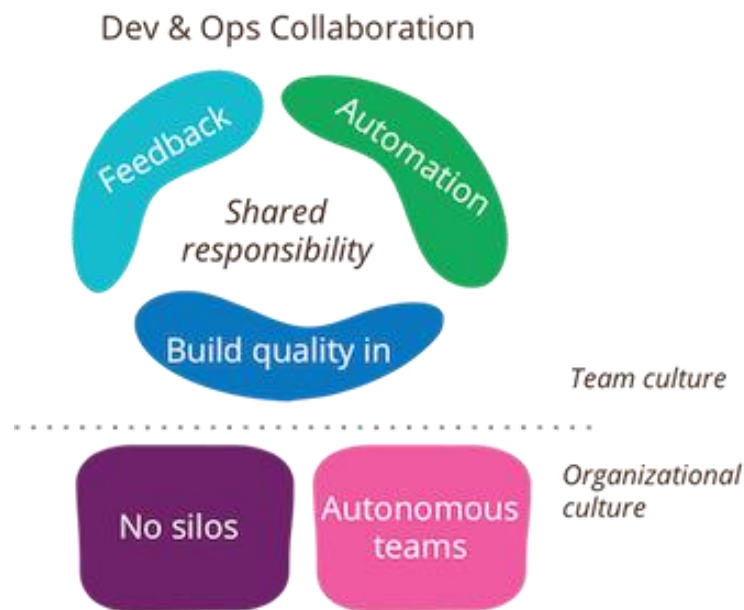
Kuva 3. CAMS-malli (Stafford 2017).

DevOps-kulttuurin rakentaminen perustuu työntekijöiden osallistuttamiseen DevOpsin käytössä. Yrityksellä voi olla useita tehokkaita ja pitkälle automatisoituja työmenetelmiä, mitkä ovat kuitenkin riippuvaisia ihmisistä, jotka käyttävät niitä tai, huonossa DevOps-kulttuurissa, ovat käyttämättä. Tämän vuoksi DevOps-kulttuurin rakentaminen ja toteutus koko työyhteisössä onkin hyvin tärkeää. (Wilsenach 2015; Sharma & Coyne 2017, 17).

DevOps-kulttuurin tarkoituksena on parantaa ohjelmistokehityksen ja järjestelmäylläpidon työntekijöiden yhteistyötä ja kommunikaatiota sekä saada heidät keskittymään yrityksen liiketoimintaan kokonaisuutena oman osaston toiminnan sijaan. Tässä tavoitteessa auttaa DevOpsin toteutusta varten tehdyt työmenetelmät, jotka taas on toteutettu vähentämään kitkaa ohjelmistokehityksen ja järjestelmäylläpidon välillä. Työmenetelmien käyttö kuitenkin edellyttää kehityksen ja ylläpidon työntekijöiden halukkuutta toteuttaa DevOpsia ja päästää irti vanhoista, mahdollisesti, huonoista tavoista. DevOps-kulttuurin tuominen onnistuneesti yritykseen on kuitenkin haaste, joka yrityksen johdon pitää sosiaalisia taitoja hyödyntäen selättää. (Sharma & Coyne 2017, 17–18).

Kuvassa 4 on esitetty hyvän DevOps-kulttuurin pääominaisuuksia ryhmä- ja organisaatiotasolla. Ohjelmistokehityksen ja järjestelmäylläpidon väliltä on saatu siilomainen ajattelumalli poistettua ja vastuu DevOpsin toteuttamisesta on saatu jaettua kaikille osallisille. Vastuun saattamisessa kaikille osallisille on tärkeää todella tuoda vastuu osallisille, eikä esimerkiksi luoda yritykselle erillistä DevOps-tiimiä tai luoda erillistä DevOps-roolia, joiden vastuuna olisi huolehtia DevOpsin toteuttamisesta. Nämä muusta kehityksestä erillään olevat DevOps-ratkaisut rikkovat helposti DevOpsin idean kehityksen ja ylläpidon lähentämisestä. (Wilsenach 2015).





Kuva 4. Ohjelmistokehityksen ja järjestelmäylläpidon yhteistyö (Wilsenach 2015).

DevOps-kulttuurin toteutumista voidaan mitata suoraan henkilöstökyselyillä tai epäsuoraan tarkkailemalla ohjelmistokehityksen ja järjestelmäylläpidon henkilöstöjen yhteistyötä ongelmanratkaisussa. Henkilöstökyselyillä pyritään esimerkiksi selvittämään, onko yhteistyö kasvanut työntekijöiden näkökulmasta ja onko tästä ollut hyötyä työtehtävien suorittamisessa. Tarkkailulla taas selvitetään työntekijöiden yhteistyöllisen ongelmanratkaisun lisääntymistä, jolla tarkoitetaan työntekijöiden suoraa kommunikointia keskenään ongelmien ratkaisemiseksi, ilman työnjohdon käyttämistä kommunikoinnin välineenä. (Sharma & Coyne 2017, 17–18).

Automaatio on usein alkuperäisenä syynä yritysten haluun siirtyä toteuttamaan DevOpsia, sillä sen avulla pystytään nopeuttamaan prosesseja ja tuomaan lisää arvoa käytetylle ajalle. Tällöin automaatio on usein myös kaikkein näkyvin osa DevOpsista, koska sen tuomasta kehityslinjan nopeuttamisesta puhutaan paljon. Nopeamman läpiviennin lisäksi automaatio mahdollistaa myös virheiden ehkäisyn, toimintojen yhtenäisyyden sekä johdonmukaisuuden ja muista riippumattoman toiminnan. (Seppä-Lassila 2017, 8).

Kehityslinjan nopeutus tapahtuu automatisoimalla aikaa vievät toistettavat prosessit ohjelmistokehityksessä, testauksessa ja järjestelmäylläpidossa. Kun kehitys ja ylläpito tekevät töitä yhdessä automatisoinnin eteen, päästään muista riippumattomaan julkaisuprosessiin. Tällöin esimerkiksi ohjelmistokehityksen ei tarvitse odotella tai häiritä järjestelmäylläpitoa julkaisuihin liittyvillä asioilla, ja ylläpidon resurssit vapautuvat silloin

muihin tärkeisiin tehtäviin. Tavoitteena onkin saada ohjelmiston julkaisu tapahtumaan kokonaisuudessaan vain yhdellä napin painalluksella. (Humble & Farley 2010, 6).

Prosessit ovat virhealttiimpia manuaalisesti toteutettuna kuin automaattisesti toteutettuna, eikä virheen tapahtumisesta saada myöskään tarkkaa tietoa. Toimintojen yhtenäisyydellä ja johdonmukaisuudella pystytään toistamaan onnistumiset ja tapahtuneet virheet, jolloin pystytään myös helpommin tutkimaan mitkä tekijät vaikuttivat näihin. Automaation avulla poistetaan inhimillisiä virheitä määritettyjen prosessien toteuttamisesta ja voidaan keskittyä varsinaisiin ohjelmakoodissa tehtyihin virheisiin. Tällöin voidaan toteuttaa mahdollisimman virheetön, tehokas ja luotettava integraatio sekä toimitus ohjelmistolle. (Humble & Farley 2010, 5–6).

Manuaaliset julkaisut vaativat paljon käsin tehtävää työtä kuten julkaisun dokumentoinnin tekoa ja päivitystä. Dokumentteja ylläpitävät useat eri henkilöt, jolloin dokumenteista usein puuttuu joitain tietoja tai niitä ei ole pidetty ajan tasalla. Lisäksi valitettavan usein manuaalisesti tehdyt dokumentoinnit ovat lähinnä muistioita niiden tekijöille, jolloin niistä ei ole paljoakaan hyötyä muille. Automaation kanssa dokumentointi voidaan siirtää automatisointiohjelmalle, jolloin se on aina ajan tasalla ja kertoo kehitysvaiheelle oleellisia asioita. (Humble & Farley 2010, 6).

Automaation avulla pystytään myös helpottamaan ohjelmistokehityksen ja järjestelmäylläpidon yhteistyötä, sillä se muun muassa poistaa palvelinresursseista syntyvää ristiriitaa. Automaatio toimiikin kehitystä ja ylläpitoa lähentävänä toteutuksena, mutta se ei varsinaisesti kuitenkaan pakota ohjelmistokehitystä ja järjestelmäylläpitoa yhteistyöhön. (Minick 2015). Automaation rakentaminen voidaan aloittaa yrityksen DevOps-kulttuurin rakentamisen ja ymmärtämisen jälkeen, jolloin kulttuuria silmällä pitäen tehdyt kehityksen ja ylläpidon väliset automaatioprosessit onnistuvat suuremmalla todennäköisyydellä (Willis 2010).

Mittauksen tarkoituksena on saada KPI-mittareiden avulla kehityslinjan toiminnasta ja tehokkuudesta tietoa, jota voidaan hyödyntää oppimisessa ja jonka avulla voidaan taas tehostaa kehityslinjan toimintaa. Hyvin ja tarkoituksenmukaisesti toteutettuna mittauksesta saatujen tulosten avulla pystytään tarkasti löytämään virheet ja kehityslinjaa hidastavat tekijät eikä ylimääräistä aikaa kulu niiden etsimiseen. Mittaustulokset antavat nopeaa ja konkreettista palautetta ohjelmistokehitykselle ja järjestelmäylläpidolle niin yleisestä toiminnasta kuin erilaisista kokeiluistakin. (Minick 2015; Stafford 2017).

Kehityslinjan toiminnan tehokkuuden mittaamiseen käytetään KPI-mittareita, eli suorituskykymittareita. Näiden mittareiden avulla pyritään saamaan konkreettista tietoa kehityslinjan kriittisistä toiminnoista, jotka voivat liittyä esimerkiksi resurssien käyttöön, epäonnistumisiin, kehityslinjan nopeuteen ja kerralla tehtyjen muutosten lukumäärään sekä suuruuteen. Käytettävät suorituskykymittarit riippuvat kuitenkin yrityksen yksilöllisestä toiminnasta, jolloin jokaisen yrityksen onkin itse päätettävä omat mittarit. Mittarit, jotka toimivat yhdellä yrityksellä, eivät välttämättä toimi toisella saman toimialan yrityksellä. (Edwards 2010a).

CAMS-mallin mittauksessa toinen ehkä jopa tärkeämpi puoli on ohjelmistokehityksen ja järjestelmäylläpidon kehityksen ja oppimisen mahdollistava palautteen saaminen. Palautetta kerätään jatkuvan kokeilun sekä riskinoton kautta, jolloin on tärkeää nähdä onnistumiset ja epäonnistumiset oppimismahdollisuuksina. Lisäksi pitää ymmärtää toiston ja harjoituksen tärkeys oppimisessa, eikä esimerkiksi kokeilua pidä lopettaa tapahtuneen epäonnistumisen tai epäonnistumisen pelon takia. (Kim, 2012).

Mittaus on kohdistettava oikeisiin ja tarkoituksenomaisiin asioihin, jolloin mittaustulokset ovat hyödyllisiä kaikille kehityslinjaan osallistuville tahoille. Useampaa monimutkaista, mutta tarkoituksenomaista mittaustulosta ei kuitenkaan kannata yhdistää tai tiivistää yhdeksi numeroksi, sillä se antaa useimmiten vain illuusion monimutkaisten mittausten ymmärtämisestä. (Hüttermann 2012, 33–34). Tarkoituksenomaisella mittauksella saadaan oleellista tietoa ja palautetta kehityksen kannalta, kuten esimerkiksi virheiden keskiverkorkorjausajasta, kun taas huonosta mittauksesta usein ajaudutaan kauemmas kehityksestä. Testien läpäisyprosentti tai tehtyjen virheiden lukumäärät ovat esimerkiksi huonoja mittauksia, sillä ne voivat johtaa usein syyttelyyn tai virheet on saatettu korjata ennen tuotteen pääsyä kehityslinjan loppuun, jolloin mittaustulokset ovat vääristyneitä. (Seppä-Lassila 2017, 20).

Mittaustulokset mahdollistavat oikeista asioista palkitsemisen, jolloin saadaan kaikki kehityslinjan toimintaan vaikuttavat henkilöt motivoitua työskentelemään yhteistä tavoitetta kohden. Oikeiden mittaustulosten perusteella myönnetty palkitseminen auttavat ohjelmistokehityksen ja järjestelmäylläpidon yhteistyön parantamisessa sekä näiden yhdeksi ryhmäksi muodostumisessa. Palkitseminen tietysti myös edellyttää kehittymistä ja palkkiot tulisi sitoa oikeisiin tuloksiin. (Hüttermann 2012, 66–67).

Mittaus on mielekästä aloittaa, kun kehityslinjalle on toteutettu vähintään alkukantainen automaatio. Tällöin on saatu poistettua ihmisistä johtuvia virheitä automaation

johdonmukaisuuden avulla ja pienennettyä väärrien mittaustulosten riskiä huomattavasti. Mittaustuloksia tulee myös tarkastella säännöllisesti, jotta niiden tuomasta informaatiosta on mahdollisimman paljon hyötyä. (Rehn 2016).

CAMS-mallin jakaminen tarkoittaa työkalujen, keksintöjen, toimintatapojen, ongelmien ja opittujen asioiden jakamista niin yritysten sisäisesti kuin ulkoisestikin. Yritysten kulttuuri on kriittisessä asemassa jakamisessa. Jos kulttuuria ei ole rakennettu, jäävät loistavat ideat ja innovaatiot usein yrityksen sisälle ja sielläkin vain pienen ryhmän hyödynnettäviksi. (Perera ym. 2017). Jakamisen kulttuuri myös edistää kunnioitusta sekä luottamusta yrityksessä ja toimii siten itseään vahvistavana ominaisuutena (Seppä-Lassila 2017, 21).

DevOpsin synnyn ja alun huomioon ottaen, jakamisen läpinäkyvyys ja avoimuus ovat sille hyvin luontaisia ominaisuuksia. (Seppä-Lassila 2017, 21). Lisäksi läpinäkyvyys ja avoimuus ovat myös yleisesti hyvin tärkeitä ominaisuuksia menestyvälle yritykselle (Perera ym. 2017). Jakaminen voidaan jakaa kahteen osioon: yrityksen sisäiseen jakamiseen ja yrityksen ulkoiseen jakamiseen (Seppä-Lassila 2017, 21).

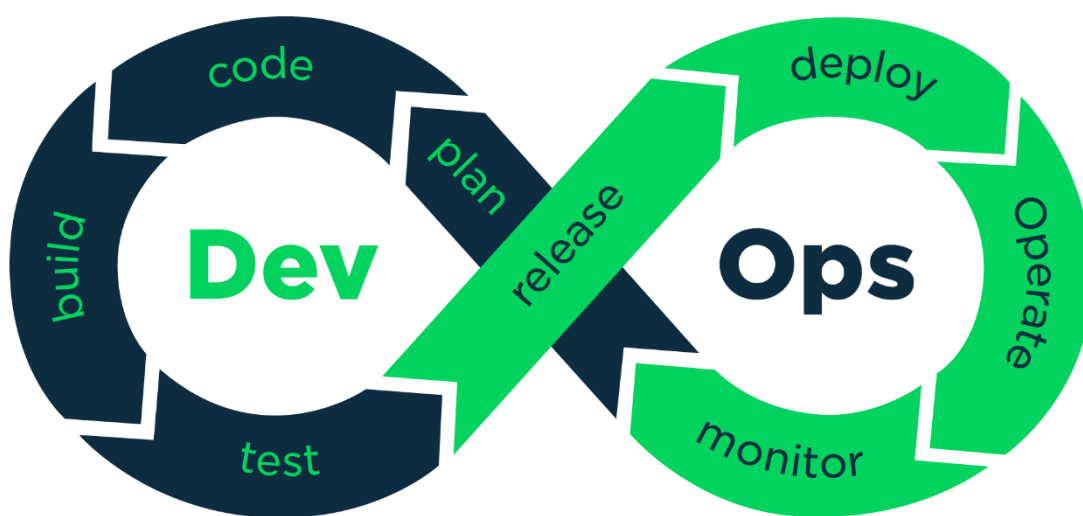
Yrityksen sisäinen jakaminen tarkoittaa ryhmät ylittävää jakamista, jolloin hyvät ratkaisut pääsevät liikkumaan yrityksessä vapaasti ja ongelmiin pystytään löytämään nopeammin ratkaisuja. Sisäinen jakaminen lisäksi lähentää yrityksen henkilöstöä ja mahdollisesti poistaa turhaa päällekkäistä työtä, kun asioita ei tarvitse tehdä kahteen kertaan eri ryhmässä. Yrityksen ulkoinen jakaminen sisältää työkalujen, ratkaisujen ja innovaatioiden jakamista muille yrityksille, joka näkyy esimerkiksi avoimena lähdekoodina. Lisäksi konferensseihin osallistumiset ja siellä muiden alan ammattilaisten kanssa ideoiden jakamiset ovat hyviä esimerkkejä ulkoisesta jakamisesta. Ulkoisen jakamisen hyötynä on teknologian nopeampi kehittyminen ja uudet innovaatiot, joita yritys kykenee taas hyödyntämään. (Seppä-Lassila 2017, 21–22).

## 2.4 Toimitusputki ja sen osat

Ohjelmistokehityksen toimitusputki (engl. delivery pipeline) on kokoelma järjestyksessä suoritettavia vaiheita, joiden tarkoituksena on tuottaa ideasta tuote tai ominaisuus ja toimittaa se asiakkaan saataville. Jokaisella yrityksellä on käytössä jonkinlainen toimitusputki, vaikka sitä ei kutsuttaisikaan toimitusputkeksi eikä sitä välttämättä olisi edes määritelty. Tämän kaltaisessa tilanteessa toimitusputki on useimmiten syntynyt usein

käytettyjen määrittämättömien, mutta hyväksi havaittujen, käytäntöjen vakiintumisesta tuotteen valmistuksessa. (Riley 2014).

Kuten useimmat ohjelmistokehityksen käytännöt, myös toimitusputki on kehittynyt ajan kuluessa. Putki on alun perin ollut suora linja ideasta tuotteeseen, mutta ketterän kehityksen iteratiivisen ja inkrementaalisen kehityksen myötä se on kehittynyt osaksi itseään toistavaksi. DevOps hyödyntää monia ketterän ohjelmistokehityksen käytäntöjä ja tämä näkyy myös sen toimitusputkessa, joka on kehittynyt täysin itseään toistavaksi. Kuvassa 5 on esitetty esimerkki DevOpsin toimitusputkesta. (Riley 2014).



Kuva 5. Mahdollinen DevOpsin toimitusputki (Kornilova, 2017).

DevOpsin toimitusputki rakentuu erinäisistä vaiheista, jotka ovat eri DevOpsin toteutuksessa usein samat, mutta niissä voi kuitenkin olla toteuttajasta riippuvaisia eroja. Jotkut toteuttajat esimerkiksi siirtävät suunnittelu- ja luomisvaiheet erilleen DevOpsin toimitusputkesta ja aloittavat putken suoraan koodin kääntämismuutoksesta. Lisäksi toimitusputken vaiheita voidaan kutsua myös eri nimellä tai niitä voidaan olla yhdistetty. Vaiheiden sisältämät toiminnot myös vaihtelevat toteuttajasta ja tarpeesta riippuen, koska yritykset ja ohjelmistot ovat harvoin samanlaisia. Tässä työssä käsitellään kuvan 5 sisältämät DevOpsin toimitusputken osat. (Riley 2014).

Suunnitteluvaihe sisältää tuotteen, uuden ominaisuuden tai päivityksen suunnittelun ja vaatimusten viimeistelyn. Tämä kattaa ohjelmiston arkkitehtuurin sekä varsinaisen suunnitelman luomisen. Lisäksi suunnitteluvaiheessa jaetaan tehtävät ja asetetaan niille

aikarajat, joita voidaan seurata jonkin tehtävien seurantajärjestelmän avulla. (Pennington 2019).

Suunnitteluvaiheessa on tärkeää tuottaa määrittelydokumentti sekä vaatimusanalyysi, jotta näitä ei jouduta tuottamaan samalla kun rakennetaan ohjelmistoa. Määrittelydokumentti ja vaatimusanalyysi selkeyttävät seuraavaa vaihetta toimitusputkessa ja ehkäisevät siten epäselvää rakennetta tuotteessa. Epäselvä rakenne taas hankaloittaa tuotteen jatkokehitystä sekä aiheuttaa helposti, selvällä rakenteella ehkäistävissä olevia, virheitä. (Bourque & Fairley 2014, 1-10–1-11).

Tavoitteena suunnitteluvaiheella on mahdollistaa kaikkien osallisten, niin ohjelmistokehityksen ja järjestelmäylläpidon henkilöstön kuin johdon ja muidenkin osallisten, vaikuttaminen tehtävään tuotteeseen. Tällöin saadaan parempi käsitys kaikkien tehtävistä ja niiden suorittamiseen vaadittavista resursseista, joka taas tarjoaa läpinäkyvyyttä siitä mitä tehdään milloinkin ja antaa mahdollisuuden havaita mahdolliset pullonkaulat aikaisin. Suunnitteluvaiheeseen osallistuminen vaikuttaa myös henkilöstön motivaatioon ja ettujen tavoitteiden kautta, jotka ovat syntyneet, kun henkilöstö on päässyt itse vaikuttamaan niiden määrittelymiseen. (Seroter 2014).

Ohjelmointivaihe sisältää varsinaisen ohjelmistokoodin luomisen suunnitteluvaiheessa tehtyjen määrittelydokumentin ja vaatimusanalyysin pohjalta. Tällöin myös viimeistellään ohjelmiston kokoonpano sekä asetukset ja tehdään koodin kirjoittajan toimesta staattista koodianalyysiä. Staattisen koodianalyysin lisäksi voidaan suorittaa paikallisessa kehitysympäristössä testausta, mikäli ympäristö sekä kehitettävä asia mahdollistavat sen. (Patel 2019).

Saman projektin kehittäjillä on usein samat, yhdessä sovitut, asetukset ja laajennukset asennettuna omiin kehitysympäristöihin, joiden tarkoituksena on selventää sekä yhtenäistää lähdekoodia ja sen arkkitehtuuria. Laajennuksilla pyritään myös ehkäisemään huonoja ohjelmointitapoja ja -rakenteita. Näin saadaan lähdekoodista johdonmukaisempaa ja siten parannettua lähdekoodin ylläpitoa sekä ehkäisemään epäselkeydestä johtuvia virheitä. (Pennington 2019).

Ohjelmointivaiheen viimeinen tehtävä on lähdekoodin lataaminen jaettuun tietovarastoon (engl. repository) kaikkien osallisten saataville. Lähdekoodin lataaminen useimmiten aloittaa automaattisesti toimitusputkessa seuraavana olevan kääntämisvaiheen. (Patel 2019).

Lähdekoodi voi olla jaetussa tietovarastossa usean eri haaran alla, jolloin ennen sen saattamista käyttövalmiiksi paketiksi, se pitää hakea, mahdollisesti kääntää ja koota näistä haaroista. Tällöin suoritetaan lähdekoodille myös automaattista yksikkö- ja integraatiotestausta, jotta yksinkertaisimmat virheet löydetään mahdollisimman aikeisessa vaiheessa. Koonnin ja kääntämisen jälkeen lähdekoodi paketoidaan ajettavaksi paketiksi, jota pystytään taas seuraavassa vaiheessa testaamaan erilaisin tavoin. (Tuli 2018; Patel 2019).

Kääntämisvaiheen aikana, ennen kuin lähdekoodi paketoidaan julkaisupakettiin, lähdekoodille ajetaan usein monta erilaista pienempää kääntöprosessia. Kääntöprosessit voivat olla esimerkiksi dokumenttien luomista tai testauksia. Kääntämisvaiheen prosessit riippuvatkin hyvin paljon tuotteesta ja sen halutusta tuotoksesta. (Smith 2011, 7).

Onnistuneen lähdekoodin kääntämisen jälkeen julkaisupaketti asennetaan koekäyttöympäristöön (engl. staging environment), joka jäljittelee lopputuotteen toimintaympäristöä. Ympäristö voi olla aiemmin tehty tai se voidaan rakentaa pakettikohtaisesti, jolloin usein ympäristön rakentamisessa hyödynnetään infrastruktuuri koodina -menetelmää. Koekäyttöympäristön tarkoituksena on toimia erillisenä testiympäristönä, jolloin julkaisupaketin ominaisuuksien testaus ei häiritse ohjelmiston kehitystä tai sen käyttöä. (Pennington 2019).

Koekäyttöympäristössä suoritetaan julkaisupaketille tarkempia manuaalisia ja automaattisia testejä. Manuaaliset testit voivat olla esimerkiksi käyttöhyväksyntätestejä, joissa ohjelmistoa käytetään kuten loppukäyttäjät sitä käyttäisivät ja pyritään näin löytämään mahdolliset virheet tai parannettavat asiat ennen paketin toimitusta tuotantoon. Automaattiset testit voivat sisältää esimerkiksi turvallisuustestejä, rakenteen tarkastusta ja suorituskyky- sekä rasiustestejä. Suoritettavat testit riippuvat yrityksestä ja ohjelmiston tarpeista. (Patel 2019; Pennington 2019).

Julkaisuvaiheessa julkaisupaketti on valmis toimitettavaksi, jolloin se on läpäissyt useat manuaaliset ja automaattiset testit ja siten riskit mahdollisille virheille on saatu minimoitua. Julkaisupaketti voidaan toimittaa julkaisuvaiheen jälkeen joko automaattisesti asiakkaille tai jättää odottamaan manuaalista hyväksyntää ennen toimitusta. Manuaalisella hyväksynnällä pystytään valikoimaan loppukäyttäjälle toimitettavat julkaisupakettien versiot sekä kontrolloimaan toimituksen ajankohtaa, jos yritys esimerkiksi haluaa toimittaa uuden version tietyillä aikaväleillä. (Pennington 2019).

Julkaisupaketista voidaan myös säätää loppukäyttäjälle näkyviin vain tietyt ominaisuudet, jolloin pakettiin voidaan liittää vielä kehitysvaiheessakin olevia ominaisuuksia. Ominaisuuksien näkyvyyden säätäminen mahdollistaa, DevOpsille ominaisen, jopa usean paketin päivittäisen julkaisun. Lisäksi näkyvyyden säätäminen antaa mahdollisuuden toteuttaa sujuvammin esimerkiksi uusien ominaisuuksien beetestausta tai ennakkojulkaisua loppukäyttäjillä. Näistä yritys saa taas tietoa ominaisuuden käytöstä ja siitä onko ominaisuus haluttu vai ei. (Patel 2019; Pennington 2019).

Käyttöönottovaihe voidaan suorittaa automaattisesti ilman odotuksia heti julkaisuvaiheen jälkeen tai suorittaa vasta kun julkaisupaketti on saanut manuaalisen hyväksynnän. Tuotantoympäristön rakentamiseen voidaan käyttää hieman muokatuilla arvoilla samaa infrastruktuuri koodina -menetelmää kuin testausvaiheessa koekäyttöympäristön rakentamisessa. Tällöin menetelmän tiedetään tuottavan toimiva tuotantoympäristö ja lisäksi riskit virheille ovat hyvin pienet, sillä julkaisupaketin testaus on suoritettu lähes identtisessä ympäristössä. (Pennington 2019).

Tuotantoympäristön päivitys vanhasta uuteen pystytään tekemään ilman käyttökatkoksia hyödyntämällä erilaisia julkaisustrategioita, kuten blue-green- ja canary-julkaisuja. Julkaisustrategiat mahdollistavat käyttäjien siirtämisen käyttämään uutta julkaisua ilman heidän sitä huomaamattaan, joka taas mahdollistaa helposti vanhaan julkaisuun takaisin siirtymisen, mikäli uudesta julkaisusta löytyy jokin kriittinen virhe. Vanhaan julkaisuun palaamisella pystytään pitämään palvelu koko ajan käytettävissä sekä luomaan virheiden korjaamiseen lisää aikaa ja siten varmistamaan, että virhe todella saadaan korjattua. (Pennington 2019).

Käyttövaiheessa tuote on paketoitu, testattu ja asetettu asiakkaille käyttöön, jolloin seuraava tehtävä on taata asiakkaille jatkuva ja luotettava pääsy tuotteeseen. Tällöin tuote on sijoitettuna johonkin isännöintipalveluun, josta se on asiakkaiden saatavissa tai käytettävissä. Jotkin isännöintipalvelut tukevat maksimikäyttäjämäärän automaattista skaalausta tarvittavan käyttäjämäärän mukaan. Tällöin isännöintipalvelusta ei jouduta koko ajan maksamaan mahdollisen maksimikäyttäjämäärän mukaan, vaan palvelusta maksetaan vain toteutuneiden käyttöjen ja resurssien kulutusten mukaan. (Pennington 2019).

Käyttövaiheessa yrityksen tehtävänä on myös kerätä informaatiota tuotteen käytöstä. Tämä mahdollistetaan hyödyntämällä valmiita työkaluja sekä luomalla tehtävään ohjelmistoon erilaisia tiedonkeruumenetelmiä. Informaation keräämisen tarkoituksena on saada ohjelmiston ja laitteiden toiminnasta hyödynnettävissä olevaa tietoa, jonka avulla



pystytään määrittämään tulevat uudet ominaisuudet tai tarvittavat muutokset sekä korjaukset. (Humble & Farley 2010, 318–319).

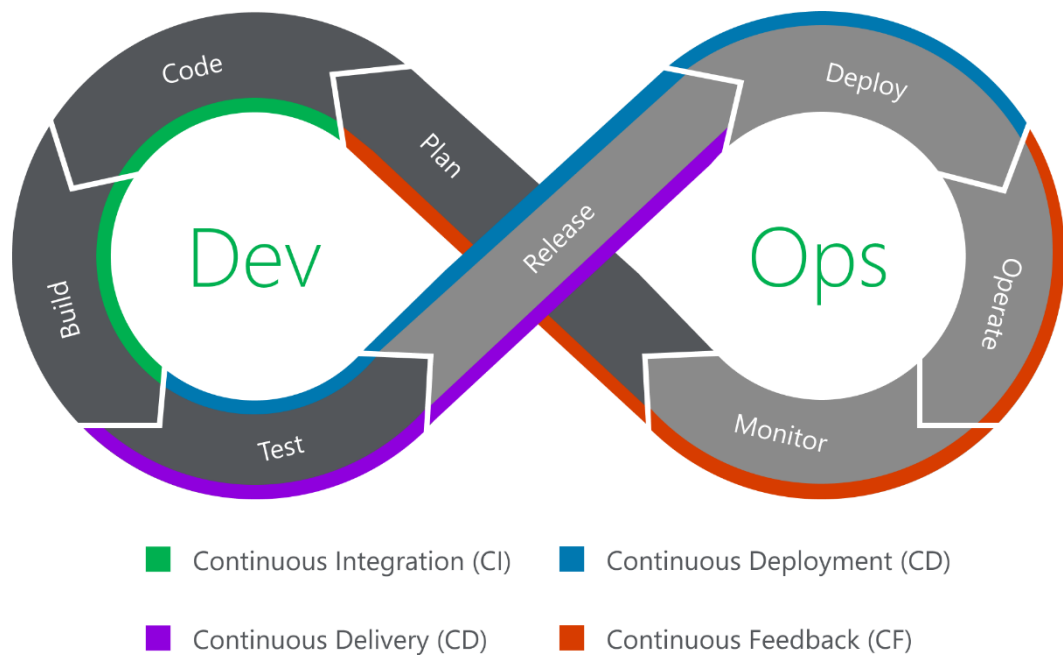
Valvontavaihe on viimeinen vaihe ennen toimitusputken uudelleen alkamista. Valvontavaiheessa analysoidaan kaikki toimitusputken suorituksen aikana kerätyt informaatiot ja esitetään ne reaaliaikaisesti esimerkiksi graafisessa käyttöliittymässä. Kerättyä informaatiota käytetään ohjelmiston kehittämisen lisäksi myös toimitusputken kehittämiseen. Tämän tarkoituksena on tehostaa toimitusputken toimintaa ja poistaa siinä olevia mahdollisia hidasteita. (Pennington 2019).

Valvontavaiheessa myös varsinaisesti valvotaan ohjelmiston toimintaa erilaisten ilmoitusten avulla. Ohjelmistossa tai ympäristöissä tapahtuvista virheistä lähetetään vastuuhenkilöille ilmoitus, esimerkiksi sähköpostilla tai jollain pikaviestimellä. Tämän tarkoituksena on nopeuttaa virheisiin reagoimista sekä helpottaa virheiden etsintää ilmoitukseen liitetyn toimintaraportin avulla. (Humble & Farley 2010, 323).

## 2.5 Jatkuvat käytännöt

DevOps-toimintamalli hyödyntää erilaisia jatkuvia käytäntöjä sen toimitusputken suorituksessa. Nämä jatkuvat käytännöt ovat oleellisen osa DevOpsia ja niiden tarkoituksena on mahdollistaa DevOpsin toimitusputken nopea ja sujuva suoritus. Käytäntöjen avulla monet toimitusputken osat pystytään suorittamaan automaattisesti, jolloin niiden suoritus tapahtuu nopeammin ja virheiden mahdollisuus pienenee. Käytännöt myös ohjaavat manuaalisesti suoritettavissa osissa hyviin ja yhtenäisiin toimintatapoihin, jolloin myös näissä osissa tapahtuvat virheet vähenevät ja ne pystytään löytämään ennen julkaisupaketin pääsyä tuotantoympäristöön. (Senapathi 2018, 5–6).

Kuten monet muutkin asiat DevOpsissa, sen toimitusputken suorittamiseen liitettävät jatkuvat käytännöt riippuvat suuresti sen toteuttajasta. Useimmiten kuitenkin DevOps-toiteutuksessa käytetään vähintään jatkuva integraatio - ja jatkuva julkaisu -käytäntöjä, vaikka nämäkin on saatettu pilkkoa pienemmiksi osiksi tai nimetty eri tavalla eri toteutuksissa. Tässä työssä käsitellään kuvan 6 mukaisesti jatkuva integraatio -, jatkuva julkaisu -, jatkuva käyttöönotto - ja jatkuva palaute -käytännöt. (Sharma & Coyne 2017, 9–10).



Kuva 6. Jatkuvat käytännöt DevOpsin toimitusputkessa (Pennington 2019).

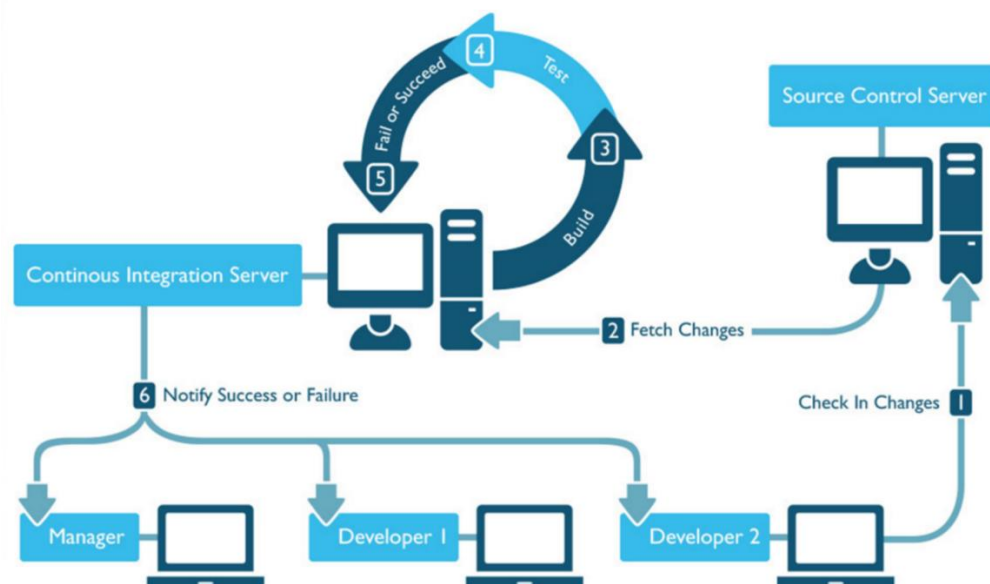
### 2.5.1 Jatkuva integraatio

Jatkuva integraatio (engl. continuous integration) on ohjelmistokehityksessä laajasti tunnettu käytäntö, jonka avulla useat kehittäjät pystyvät luotettavasti kehittämään samanaikaisesti samaa ohjelmistoa. Kehittäjät lataavat tekemänsä muutokset versionhallintaan useasti päivän aikana, mieluiten vähintään kerran, jolloin muutokset integroidaan ja varmennetaan automatisoidun paketin kääntämisen avulla. Jatkuva integraatio vähentää huomattavasti julkaisupaketin mahdollisia integraatio-ongelmia ja mahdollistaa kokonaisten ohjelmistojen nopeamman kehittämisen. (Fowler 2006; Shahin ym. 2017, 2).

Jatkuvalla integraatiolla on useita hyötyjä, joista suurimmat ovat nopeammin ja useammin tapahtuvat julkaisut, kehittäjien korkeampi tuottavuus sekä vähentyneet riskit ja korkeampi laatu. Jokainen versionhallintaan tehty muutos integroidaan ja käännetään julkaisupaketiksi, jolloin uusimmat muutokset sisältävä julkaisupaketti on lähes aina mahdollista ottaa heti käyttöön. Integrointi ja julkaisupaketin käänнос tapahtuvat automaattisesti ja tällöin kehittäjien työpanos voidaan siirtää toistuvien kääntötehtävien suorittamisesta varsinaiseen ohjelmiston kehittämiseen. Julkaisupaketin integraation ja käännos aikana tapahtuneet virheet pysäyttävät toimitusputken ja ilmoittavat kehittäjille

tapahtuneesta virheestä, jolloin riskit toimimattoman julkaisupaketin käyttöönotolle ovat hyvin pienet. (Duval ym. 2007, 29–31).

Käytännössä jatkuvan integraation prosessi alkaa, kun kehittäjä lataa tekemänsä muutokset jaettuun versionhallintaan. Tällöin manuaalisesti tai automaattisesti ladataan uusimmat muutokset versionhallinnasta ja käynnistetään kääntöskripti (engl. build script), joka yhdistää ja paketoi tarvittavat tiedostot julkaisupakettiin. Paketoimisen lopuksi sen onnistumisesta tai epäonnistumisesta ilmoitetaan määritetyille vastuuhenkilöille jollain tavalla, esimerkiksi sähköpostilla. Kuvassa 7 on havainnollistettu jatkuvan integraation toimintaa, jossa hyödynnetään erillistä jatkuvan integraation palvelinta automaattiseen versionhallinnan muutosten havaitsemiseen. (Duval ym. 2007, 4–5).



Kuva 7. Jatkuvan integraation toiminta (Pepgotesting 2019).

Jatkuva integraatio vaatii varsinaisesti vain muutaman rakennuspalasen: versionhallinnan, automaattisen kääntämisen ja palautejärjestelmän (Duvall ym. 2007, 12; Humble & Farley 2010, 56–57). Usein kuitenkin jatkuvan integraation toiminnan ja hyötyjen korostamiseksi siihen liitetään myös automaattisen testauksen ja tarkastelun suorittaminen sekä henkilöstön sitoutumisen saavuttaminen (Duvall ym. 2007, 15–17; Humble & Farley 2010, 57; Mikita ym. 2012, 5). Jatkuva integraatio käsittää ohjelmointi- ja kääntämisvaiheet DevOpsin toimitusputkesta.

Yleisesti versionhallinnan järjestelmää olisi hyvä käyttää kaikessa ohjelmistokehityksessä, mutta jatkuvalle integraatiolle se on elintärkeä. Se toimii yhtenä jaettuna tietokantana jokaiselle kehittäjälle, joka lataa tiedostoja versionhallintaan tai hakee niitä sieltä. Tämä tarkoittaa jokaisen tehdyn uuden ominaisuuden tai ohjelmointivirheen korjauksen tai muun vastaavan päivityksen tulevan mukaan suoraan automaattisesti käännettävään julkaisupakettiin. Tämän lisäksi versionhallinta tallentaa jokaisen tiedoston muutoshistorian, jolla taas mahdollistetaan julkaisupaketin aiempien versioiden lataus versionhallinnasta ja siten usean eri version yhtäaikaisen kehityksen ja käytön eri tuotantoympäristöissä. (Duvall ym. 2007, 7–8).

Ennen muutosten lataamista versionhallintaan, kehittäjä suorittaa omassa kehitysympäristössään paikallisia testejä, jotka ovat usein samoja kuin automaattisessa kääntämisessä suoritettavat. Nämä testit koostuvat usein pääasiassa yksikkötesteistä ja staattisesta koodianalyysistä, mutta niiden lisäksi voidaan suorittaa myös muuta testausta, jos esimerkiksi on havaittu jonkin testin usein löytävän virheitä myöhemmin toimitusputkessa. Tällä toiminnalla pyritään ehkäisemään tehtyjen virheiden pääsyä versionhallintaan asti. (Humble & Farley 2010, 66–67).

Versionhallintaan kannattaa laittaa kaikki ihmisen luettavissa olevat tiedostot, joita tarvitaan ohjelmiston kääntämiseen, testaukseen, julkaisuun ja käyttöönottoon. Lähdekoodin lisäksi versionhallintaan on hyvä tuoda dokumentit, asetustiedostot, luomisskriptit, kirjasot ja kaikki näiden kaltaiset tiedostot. (Smith 2011, 392–402). Versionhallintaan ei kuitenkaan pidä tuoda objektitiedostoja tai ohjelmistoja tai mitään tiedostoja, jotka luodaan ohjelman kääntämisvaiheessa. Myöskään versionhallintaan ei kannata lisätä käännöshallintaskriptejä, jotka esimerkiksi määrittävät käytettävän kääntökoneen perustuen sen hetkiseen kuormitukseen, sillä ne liittyvät enemmänkin ympäristöön kuin itse kehitettävään tuotteeseen. (Smith 2011, 402–406). Oikein toteutettuna uusi kehittäjä voi ladata versionhallinnasta kaikki tarvittavat tiedostot, ajaa käännösskriptin ja saada tuotettua toimivan julkaisupaketin (Fowler 2006).

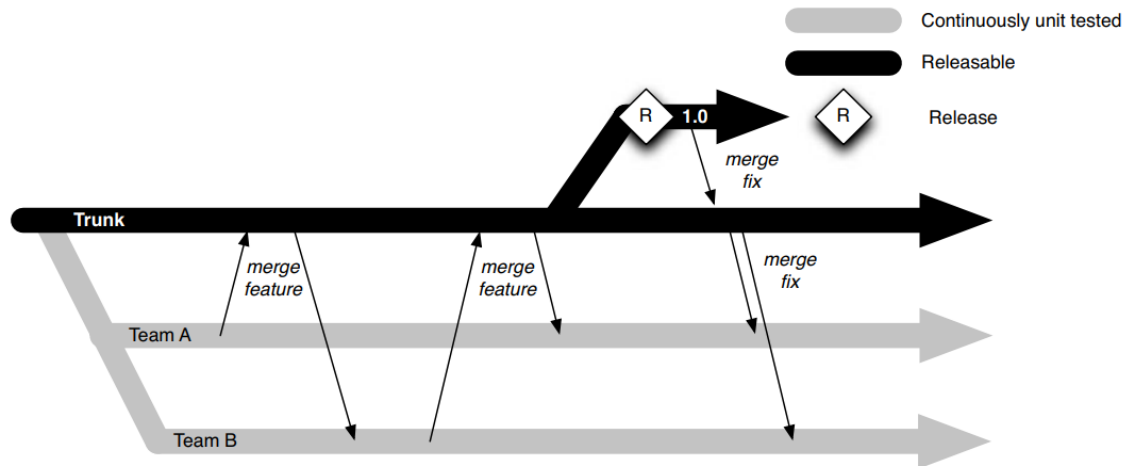
Jatkuvassa integraatiossa kannustetaan kehittämään ja tekemään muutokset versionhallinnan päärunkoon, eikä juurikaan käyttämään erillisiä kehityshaaroja. Tällä varmistetaan julkaisupaketin sisältävän uusimmat päivitykset ja sen pysyvän lähes koko ajan julkaistavassa tilassa. Lisäksi, kun useat eri haarat jätetään kokonaan luomatta versionhallintaan, erilliset haarasta toiseen päärungon kautta tehtävät ja muistettavat yhdistämiset vähenevät huomattavasti. Tällöin tapahtuu paljon vähemmän erilaisia integraatio-

ongelmia, joiden korjauksiin kuluisi ylimääräistä aikaa ja julkaisupaketti on nopeammin valmiina käyttöönotettavaksi. (Humble & Farley 2010, 390–393).

Versionhallinnan päärunkoa vasten kehittäessä voidaan käyttää erilaisia kehitysstrategioita, joilla pystytään pitämään ohjelmisto käyttöönottovalmiina samanaikaisesti, kun useampi kehittäjä tekee siihen muutoksia. Näitä kehitysstrategioita ovat uusien ominaisuuksien piilottaminen, muutosten tekeminen pieninä osina inkrementaalisesti, komponenttipohjaisen kehityksen toteuttaminen sekä ohjauskerroksen käyttö. Strategioilla pystytään käytännössä jättämään kehityshaarojen käyttö kokonaan pois tinkimättä kuitenkaan korjausten, uusien ominaisuuksien tai muiden vastaavanlaisten päivitysten kehittämisestä. (Humble & Farley 2010, 346–347).

Jos versionhallinnassa halutaan kuitenkin haaroitusta välttämättä käyttää, voidaan käyttää jotain kolmesta strategiasta: julkaisuhaaroitusta, ominaisuushaaroitusta tai ryhmähaaroitusta. Julkaisuhaaroituksessa eri julkaisuversiot haaroitetaan päärungosta juuri ennen julkaisun tapahtumista ja haaroituksen jälkeen siihen tehdään enää vain korjauksia, jotka taas yhdistetään päärunkoon. Ominaisuushaaroituksessa haarojen tulee olla hyvin lyhyt ikäisiä ja ne liitetään päärunkoon vasta ominaisuuden valmistuttua ja testien läpäistyä. Lisäksi ominaisuushaaraan on tärkeää yhdistää usein, vähintään kerran päivässä, päärunkoon tehdyt muutokset. (Humble & Farley 2010, 408–412).

Ryhmähaaroituksessa jokaiselle kehittäjäryhmälle luodaan oma haara, joka toimii käytännössä sen ryhmän versionhallinnan päärunkona ja johon ladataan vähintään kerran päivässä varsinaisen päärunгон muutokset. Tällöin jokaisen ryhmän tekemän muutoksen valmistuttua, haaralle suoritetaan yksikkö- ja hyväksyntä testausta ja testien läpäisyn jälkeen se yhdistetään päärunkoon, jossa taas suoritetaan kaikki testit. Kuvassa 8 on esitetty julkaisuhaaroitusta sekä ryhmähaaroitusta. (Humble & Farley 2010, 412–415).



Kuva 8. Julkaisuhaarat ja ryhmähaarat (Humble & Farley 2010, 413).

Kääntäminen käynnistyy useimmiten automaattisesti, kun versionhallintaan ladataan muutoksia. Automaattisen kääntämisen voi käynnistää erillinen jatkuvan integraation palvelin kyselemällä muutoksista versionhallinnalta tai se voidaan käynnistää tapahtumien kuuntelijan (engl. event listener) avulla. Kääntäminen voidaan myös käynnistää haluttaessa manuaalisesti komentoriviltä tai ohjelmointiympäristön kautta. (Duvall ym. 2007, 8). Ohjelmointiympäristön kautta käännettäessä on kuitenkin hyvä huomioda, että sen lisäksi tarvitaan komentoriviltä käynnistettäviä käännösskriptejä jatkuvan integraation toteuttamiseksi (Duvall ym. 2007, 10).

Jatkuvan integraation palvelin käsittää erillisen koneen, joka toimii käännön suorittamisen alustana. Tällä pystytään ehkäisemään virheitä, jotka johtuvat kehitysympäristön asetuksista. Käytännössä saadaan ehkäistystä ”kyllä se toimi minun koneellani”-tyyppiset virheet ennen kuin ne pääsevät käyttöönottoon asti ja siten taas saadaan ennalta ehkäistystä aikaa vieviä korjauksia. (Duvall ym. 2007, 81–82). Koneeseen on useimmiten asennettu jokin jatkuvan integraation automaatiopalvelin, kuten Jenkins tai CruiseControl. Valmiita työkaluja ei kuitenkaan ole pakollista käyttää jatkuvan integraation toteuttamiseksi, vaan kehittäjät voivat rakentaa oman jatkuvan integraation palvelimensa tai toteuttaa manuaalista integraatiota. Useat jatkuvan integraation palvelimet kuitenkin jo valmiiksi sisältävät erilaisia, usein haluttuja, ominaisuuksia, kuten käännöksen seurantarajapintoja, mittauksen suorittamisia ja dokumentaatioiden luomisia. (Duvall ym. 2007, 85–86).

Kääntämisvaiheen alussa käynnistetään käännösskripti, joka sisältää varsinaisen kääntämisen prosessin vaiheet. Näihin vaiheisiin kuuluu vähintään tiedostojen koonti, niiden

kääntäminen ja julkaisupaketin rakentaminen. Vaiheisiin kuitenkin voi ja useimmiten kuuluu myös muita vaiheita, kuten esimerkiksi testauksen ja tarkastelun suorittamista. Käännöskriptiin liitettävät vaiheet riippuvat lopulta sen tarpeista. (Duvall ym. 2007, 10). Mikäli tiedostojen kääntämisen aikana tapahtuu virheitä, tulee koko kääntämisen vaihe keskeyttää ja ilmoittaa epäonnistuneeksi. Tämän avulla säästetään taas aikaa, koska tiedostoja ei ole saatu käännettyä koneluettavaan muotoon, jolloin julkaisupaketti ei voi toimia. (Humble & Farley 2010, 120).

Varsinaisen kääntämisen tulisi tapahtua toimitusputken iteraation aikana vain kerran, jolloin koneluettavaksi käännetyt tiedostot ja muodostettu julkaisupaketti pysyvät samoina koko iteraation ajan. Tällä pystytään ehkäisemään eroavaisuuksia, jotka voivat johtua esimerkiksi toimitusputken myöhemmässä vaiheessa käytetyn kääntäjän eri versiosta. Lisäksi pystytään poistamaan toistuvaa työtä, kun hyödynnetään ensimmäisen kääntämisen tuotoksia toimitusputken joka vaiheessa. (Humble & Farley 2010, 113–115).

Jatkuvan integraation pitää pystyä tuottamaan nopeasti palautetta kääntämisen edistymisestä, jotta sen käyttö olisi tehokasta. Hyvänä tavoitteena palauteen saamiselle kääntämisestä on viisi ja pisimmillään 10 minuuttia, jonka kehittäjä voi odottaa ennen seuraavaan kehitystehtävään siirtymistä. Jatkuvan integraation toimintatapoihin kuuluu myös virheiden korjaaminen heti niiden tapahduttua, jolloin kääntämisen odottaminen palauteen saamiseksi ja julkaisupaketin käyttöönotettavuuden varmistamiseksi ei saa kestää kovin kauaa. (Fowler 2006).

Palautteessa on sen nopeuden lisäksi tärkeää välittää oikeaa informaatiota, oikeille henkilöille, oikeaan aikaan ja oikealla tavalla. Tällöin pystytään luottamaan siihen, että mahdolliset ongelmat tulevat korjatuksi nopeasti ja tehokkaasti, eivätkä ne pääse leviämään. Palauteen tärkein tavoite on kuitenkin lopulta saada tarvittavat toimenpiteet käynnistettyä. (Duvall ym. 2007, 205).

Kääntämisen valmistuttua, sen tulos pitää ilmoittaa, oli se sitten onnistunut tai epäonnistunut. Onnistuneessa käännössä riittää useimmiten lyhyt ilmoitus, mutta epäonnistuneessa käännössä on syytä muodostaa kääntämisen tuloksista yhteenveto epäonnistumisen syyn löytämiseksi. Tämä yhteenveto sisältää kääntämisen eri osien, kuten testien ja varsinaisen kääntämisen, tulokset ja mahdolliset virheilmoitukset. Ongelmaa korjaavan henkilön on myös hyvä päästä käsiksi kääntämisen aikaiseen konsolitulostukseen, sillä siitä voi ilmetä tarkemmin ongelman alkuperä. (Humble & Farley 2010, 171).

Kääntämisen palautteen vastaanottajiksi voi olla houkuttavaa määrittää kaikki projektissa työskentelevät henkilöt, mutta tällä lähestymistavalla viestejä voi tulla niin paljon, ettei tärkeisiin viesteihin enää huomata reagoida. Tämän takia palautetta on hyvä lähettää vain niille henkilöille, joille se on oleellista. Henkilöiden roolit projektissa auttavat usein määrittämään heille lähetettävän palautteen tyyppin. Esimerkiksi kehittäjille oleellista tietoa on heidän lataamiensa muutosten käynnistämisen kääntämisen tulos, kun taas projektipäällikölle oleellista tietoa on julkaisupaketin käyttöönotettavuus ja siihen tehdyt uudet ominaisuudet. (Duvall ym. 2007, 207–208).

Palautteen saamisen nopeus hyödyttää huomattavasti ongelmien etsimisessä ja korjaamisessa, sillä virheet on useimmiten helpompi korjata mitä aiemmin ne huomataan. Tällöin ongelman aiheuttanut muokkaus ei ole päässyt vielä unohtumaan ja virheen etsintä on helpompaa, sillä virheen aiheuttaneen muokkauksen jälkeen ohjelmisto ja lähdekoodi ei ole välttämättä ehtinyt muuttumaan paljoakaan. Tämä kuitenkin edellyttää, että kehittäjät seuraavat jatkuvan integraation toimintatapaa tehdä kerralla vain pieniä ja hallittuja muutoksia. (Humble & Farley 2010, 171).

Nopea palaute on myös nopeasti muuttuvaa, tällöin oleellisen ja ajankohtaisen palautteen toimittamiseksi tarvitaan myös tapa, jolla palaute saadaan toimitettua nopeasti vastuhenkilöille. Yleensä varsinkin tarkemman teknisen palautteen tai raportin toimittamiseen hyvä keino on lähettää vastuhenkilöille sähköposti tai tekstiviesti tai viesti jonkin pikaviestimen kautta. Palauteviestin ei kuitenkaan itsessään tarvitse sisältää raportteja, vaan ne voidaan lukea jostain muualta, kuten jatkuvan integraation palvelimelta. Tällöin riittää jokin yksinkertainen mekanismi, jolla viestitään kääntämisen olevan valmis ja mahdollisesti kääntämisen tulos. Tähän tarkoitukseen käy edelleen hyvin aiemmin mainitut tavat, mutta tapa voi olla myös personalisoitu, kuten tietyn äänimerkin kuuluminen tai tietyn värisen valon syttyminen kääntämisen valmistuessa. (Duvall ym. 2007, 209–221).

Testausta ja tarkastelua suoritetaan paikallisessa kehitysympäristössä ennen muutosten lataamista versionhallintaan ja versionhallinnasta kääntämisen aikana, jos ne on jatkuvan integraation toteutukseen sisällytetty. Ilman testausta ja tarkastelua, onnistunut kääntäminen merkitsee kuitenkin vain, että ohjelmisto on saatu käännettyä ja julkaisupaketti koottua, jolloin sen varsinaisesta toimivuudesta ei ole suuria takeita. Tämän vuoksi on hyvin suositeltavaa ja melko tavanomainen käytäntö liittää jatkuvan integraation toteutukseen lähdekoodin testaus ja tarkastelu. (Humble & Farley 2010, 60).



Tyypillisesti suoritettavat testit koostuvat pääosin yksikkö- ja integraatiotesteistä, mutta niihin voidaan liittää myös muun tyyppisiä testejä, jos ne koetaan tarpeellisiksi. Muun tyyppisiä testejä ovat yleensä testit, jotka löytävät usein virheitä toimitusputken myöhemmissä osissa. Näiden liittäminen mahdollisimman aikaiseen vaiheeseen toimitusputkessa nopeuttaa virheiden löytämistä ja säästää siten aikaa. Paikallisessa kehitysympäristössä ja käännön aikana suoritettavien testien tarkoituksena on löytää yksinkertaiset virheet nopeasti. (Humble & Farley 2010, 120–121).

Kaikkien suoritettavien testien pitää suoriutua ilman virheitä ennen seuraavaan vaiheeseen siirtymistä. Ei ole kuitenkaan tarkoituksenomaista lopettaa testausta, mikäli jokin testi havaitsee virheen, sillä lähdekoodissa saattaa olla useampia virheitä. Tällöin testauksen alkuvaiheet joudutaan aina odottamaan uudestaan, kunnes törmätään seuraavaan virheeseen. Virheen löytyessä, kääntövaiheen pysäyttämisen sijaan, tulisikin suorittaa kaikki testit loppuun ja ilmoittaa tulos vastuuhenkilöille, jolloin kaikki löytyneet virheet voidaan korjata ennen testauksen uudelleen suorittamista. Paikallisessa kehitysympäristössä kehittäjä saa suoraan palautteen testauksesta, kun taas kääntämisen aikaisesta palautteen toimittamisesta vastuuhenkilöille huolehtii palautejärjestelmä. (Duvall ym. 2007, 156–157).

Kääntämisen aikana voidaan suorittaa myös lähdekoodin tarkastelua, jonka tarkoituksena on pitää huolta siitä, että lähdekoodi seuraa tiettyjä sille määritettyjä sääntöjä ja ilmoittaa mikäli jotain sääntöä on rikottu. Käytännössä tarkastelu on tietokoneen suorittamaa staattista koodianalyysiä. Tarkasteltaviksi säännöiksi voidaan asettaa esimerkiksi tiettyjen luokkien tai objektien riippuvuuksien lukumäärä tai yksinkertaisempana sääntönä tiedostojen rivien lukumäärä. Tarkastelusta muodostetaan yhteenveto kääntämisen loppuraporttiin ja julkaisupaketti hylätään, mikäli sääntörikkomuksia on tapahtunut. Yhteenvedon avulla kehittäjät pystyvät paikantamaan ja korjaamaan ongelmakohdat, kun ne vielä ovat helposti korjattavissa. (Duvall ym. 2007, 162–163).

Jatkuva integraatio nopeuttaa huomattavasti julkaisusykliä ja vähentää tehokkaasti julkaisupakettiin pääseviä virheitä, mutta näitä hyötyjä ei saavuteta ilman henkilöstön sitoutumista toteuttaa käytäntöä. Tässä korostuu DevOpsin kulttuurin muuttaminen myönteiseksi ja sitä tukevaksi. Jatkuvan integraation käytäntö on sinänsä helppo toteuttaa, sillä se ei usein vaadi henkilöstöltä uusien erillisten ohjelmistojen opettelua, vain tiettyjen vaiheiden ja toimintatapojen noudattamista. (Mikita ym. 2012, 4–5).

Jatkuvaa integraatiota toteuttavan henkilöstön kannattaa sisäistää seuraavat seitsemän vaihetta:

1. Tarkasta tapahtuuko kääntäminen tällä hetkellä, jos tapahtuu, odota sen loppua.
2. Kääntämisen onnistuneesti valmistumisen jälkeen hae viimeisimmät muutokset versionhallinnasta.
3. Aja kääntöskriptit ja testit paikallisesti omassa kehitysympäristössäsi.
4. Käännösskriptien ja testien onnistuneesti valmistumisen jälkeen lataa tekemäsi muutokset versionhallintaan.
5. Odota julkaisupaketin kääntäminen loppuun.
6. Jos kääntäminen epäonnistuu, korjaa virheet välittömästi omassa kehitysympäristössäsi ja jatka kohdasta kolme.
7. Jos kääntäminen onnistuu, siirry seuraavaan kehitystehtävään.

Näiden vaiheiden seuraaminen jokaisen tehdyn muutoksen kanssa, varmistaa julkaisupaketin pysymisen käyttöönottettavassa tilassa tehtyjen päivityksen jälkeen. (Humble & Farley 2010, 58–59).

Mainittujen vaiheiden lisäksi yhtenä tärkeänä toimintatapana on ladata kaikki, hyvin pienetkin, muutokset usein versionhallintaan. Tällöin riski integraatio-ongelmille pysyy pienenä ja pystytään ylläpitämään lähes koko ajan julkaistavassa tilassa olevaa, viimeiset muutokset sisältävää, julkaisupakettia. Lisäksi pienten muutosten usein versionhallintaan lataaminen helpottaa huomattavasti virheenetsintää, mikäli kääntäminen epäonnistuu. (Humble & Farley 2010, 59; Mikita ym. 2012, 5).

Julkaisupaketin rikkoutuessa muiden on tärkeää olla lataamatta versionhallintaan lisää muutoksia, sillä ne vain hankaloittavat virheenetsintää ja siten pidentävät sen korjaukseen kuluvaa aikaa. Jatkuvassa integraatiossa tapahtuneiden virheiden korjaaminen onkin aina etusijalla muuhun kehitykseen nähden ja se on myös yksi jatkuvan integraation tärkeistä toimintatavoista. Rikkinäiset muutokset pitäisi pyrkiä korjaamaan samana päivänä, eikä niitä kannata jättää seuraavaan päivään, sillä muokatut asiat voivat unohtua varsinkin viikonlopun aikana tai kehittäjä voi sairastua. Työpäivän lopun lähestyessä tämä voi tarkoittaa versionhallinnan palauttamista aiempaan versioon virheiden korjaamiseksi ja julkaisupaketin käyttöönottilan varmistamiseksi. Yleisesti voi olla myös suositeltavaa ladata tehdyt muutokset vasta seuraavana päivänä, mikäli muutokset on saatu valmiiksi lähellä työpäivän loppua. (Humble & Farley 2010, 66–69).

### 2.5.2 Jatkuva toimitus ja jatkuva käyttöönotto

Jatkuva toimitus (engl. continuous delivery) ja jatkuva käyttöönotto (engl. continuous deployment) jatkavat jatkuvan integraation käytäntöä sisältämään tuotantoon julkaisemiseen tarvittavat toimenpiteet. Tämä käsittää toimitusputkesta jatkuvan toimituksen tapauksessa testauksen ja julkaisun vaiheet ja jatkuvan käyttöönoton tapauksessa lisäksi vielä käyttöönoton vaiheen. Jatkuva toimitus ja jatkuva käyttöönotto ovatkin keskenään lähestulkoon samanlaiset, mutta ne eroavat viimeisellä vaiheellaan. Jatkuvan toimituksen tuotos on hyväksyntätestattu julkaisupaketti, joka voidaan halutessa ottaa käyttöön tuotannossa. Jatkuvasa käyttöönotossa taas testattu julkaisupaketti otetaan käyttöön automaattisesti. Jatkovaa käyttöönottoa ei voida suorittaa ilman jatkuvaa toimitusta. (Fowler 2013).

Jatkuvan toimituksen ja jatkuvan käyttöönoton automaation avulla uuden julkaisupaketin tuomiseen tuotantoympäristöön ei enää tarvita erillisiä tehtäväjärjestelmiä tai sähköpostiketjuja, joilla selvitetään lähdekoodin valmiutta julkaisuun. Tällöin jokainen tehty muutos saadaan automaattisesti julkaistua. (Humble & Farley 2010, 130). Julkaisupaketteja ei kuitenkaan julkaista heti tuotantoon, vaan niille suoritetaan vielä tarkempaa hyväksyntätestausta. Tätä varten tarvitaan jokin tai joitain mahdollisimman paljon tuotantoympäristöä jäljitteleviä ympäristöjä, joissa testausta voidaan suorittaa häiritsemättä kehitystä tai tuotantoa. Jatkuvasa toimituksessa ja jatkuvasa käyttöönotossa rakennuspalasina toimivat jatkuvan integraation lisäksi erilaisten ympäristöjen luonti, niissä suoritettavat hyväksyntätestaukset ja varsinainen tuotantoon julkaiseminen. (Humble & Farley 2010, 126).

Useasti jatkuvasa toimituksessa ja jatkuvasa käyttöönotossa on monia eri ympäristöjä, joiden tarkoituksena on suorittaa eri tasoisia testejä julkaisupaketille. Näiden lisäksi on tietenkin yksi tai useampia tuotantoympäristöjä, joissa ylläpidetään julkaistusta ohjelmistosta eri versioita asiakkaita varten. Ympäristöihin julkaiseminen voi tapahtua sarjassa, aina edellisen testiympäristön testien läpäisemisen jälkeen, tai rinnakkain, jolloin julkaisupakettia testataan yhtä aikaa monessa eri ympäristössä. (Humble & Farley 2010, 126–127).

Ympäristöjen luominen tulisi tapahtua automaattisesti. Tällä pyritään varmistamaan asetustiedostojen pitämisen päivitettyinä sekä ehkäisemään vaikeasti tulkittavien virheiden syntymistä. Vaikeasti tulkittavilla virheillä tarkoitetaan virheitä, jotka ovat syntyneet, kun

ympäristön asetuksia on menty muuttamaan käsin, eikä muutosta ole dokumentoitu mihinkään. Lisäksi automaattisella ympäristöjen luomisella pystytään luomaan helposti uusi ympäristö sen sijaan, että alettaisiin korjaamaan vanhaa. (Humble & Farley 2010, 49–50).

Ympäristöjen tärkeänä osana ovat niiden asetukset, jotka toimivat pohjana automaattiselle ympäristöjen luomiselle. Ympäristön asetukset sisältävät usein tiedot käyttöjärjestelmistä, käytettävistä ohjelmistopaketeista, verkon topologiasta, ulkoisista palveluista ja muista ympäristön tarvitsemista resursseista, kuten tietokannoista. Asetukset ovat hyvä ladata versionhallintaan, jotta niistä on muutoshistoriat saatavilla ja ne ovat käytettävissä automaattista ympäristön luomista varten. (Duvall ym. 2007, 194–195; Humble & Farley 2010, 50–51).

Ympäristön asetusten avulla pystytään automaattisesti luomaan julkaisupaketille tuotantoympäristö, johon se voidaan julkaista. Tätä ennen julkaisupaketti tarvitsee kuitenkin vielä testata erillisissä testiympäristöissä, joiden on hyvä jäljitellä mahdollisimman paljon tuotantoympäristöä. Mikäli tuotantoympäristöjä on useampia, kuten esimerkiksi käyttäjän koneelle asennettavalla ohjelmistolla, tulisi pyrkiä suorittamaan hyväksyntätestaukset mahdollisimman monessa eri tuotantoympäristössä jäljittelevässä testiympäristössä. (Duvall ym. 2007, 196; Humble & Farley 2010, 124–125).

Testiympäristöihin julkaiseminen voi tapahtua automaattisesti onnistuneen kääntämisen jälkeen tai manuaalisesti esimerkiksi lisätestausta varten. Useasti toimitusputken tässä vaiheessa suoritettavat testit kestävät pidemmän aikaa, jolloin voi olla hyvä pohtia kaikkiin mahdollisiin testiympäristöihin julkaisemisen mielekkyyttä. Tämän kaltaisia tilanteita voivat olla esimerkiksi hyvin pienet tehdyt muutokset tai kriittisten virheiden korjaukset, jotka eivät kuitenkaan ole vaikuttaneet suureen osaan lähdekoodia. (Humble & Farley 2010, 127).

Hyväksyntätestauksen tarkoituksena on varmistaa, että tuotettava ohjelmisto toimii kuten asiakas on halunnut, eikä niin kuinka kehittäjä on ajatellut. Tämä tarkoittaa, että sitä testataan sekä toiminnallisuuden, eli asiakkaiden toiveiden, että toimivuuden, eli regression, kannalta. Hyväksyntätestauksen tarkoituksena on myös saada kaikki osalliset ajattelemaan mitä jokainen asetettu kriteeri oikeasti tarkoittaa, ja tämän kautta määrittelemään ja toteuttamaan ohjelmisto käyttötärpeiden mukaisesti. (Humble & Farley 2010, 188; Hüttermann 2012, 157–158).

Jatkuvassa toimituksessa ja jatkuvassa käyttöön otossa suoritetaan jatkuvan integraation avulla tuotetulle julkaisupaketille automaattista testausta, jossa varmistetaan asiakkaan vaatimusten täyttyminen ja julkaisupaketin toimivuus tuotantoympäristössä. Suoritettavat testit ovat suurimmaksi osaksi asiakkaan määrittämiä toiminnallisia testejä (engl. functional test), joissa testataan julkaisupakettia kokonaisena systeeminä, tarvitsematta varsinaisesti tietää sen sisäisestä toiminnasta juuri mitään. (Humble & Farley 2010, 124). Toiminnallisissa testeissä ohjelmistolle annetaan jokin syöte, kuten napin painallus tai arvo tekstikenttään ja arvioidaan systeemin antamaa tulosta (Zalavadia 2019).

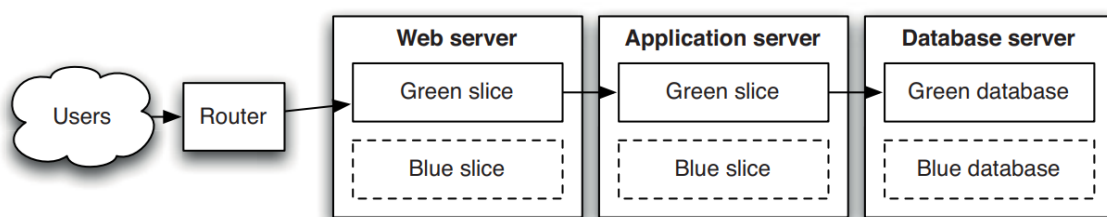
Toiminnallisten testien lisäksi voidaan ja useimmiten tarvitsee suorittaa automaattisesti ei-toiminnallisia testejä, joiden tarkoituksena on testata esimerkiksi julkaisupaketin suorituskykyä ja turvallisuutta. Ei-toiminnalliset testit ovat tärkeitä varmistamaan julkaisupaketin toimivuus todellisessa käytössä, jolloin vaarana voivat olla esimerkiksi tietomurron tapahtuminen tai liiallisesta rasituksesta johtuva järjestelmän romahtaminen. Ei-toiminnalliset testit vaikuttavat useasti hyvin paljon kehitettävän ohjelmiston arkkitehtuuriin, jolloin niiden kriteerit tulisi päättää jo projektin alussa. Tällöin projektia pystytään kehittämään alusta asti päätetyt kriteerit mielessä pitäen. (Humble & Farley 2010, 225–227).

Julkaisupaketille voidaan suorittaa myös manuaalista testausta, joka voi olla esimerkiksi kokeilutestausta tai käytettävyydestestausta, mutta kuitenkin useimmiten testausta, jota ei pystytä tai on hyvin hankalaa automatisoida. Hyvin suoritettu manuaalinen testaus vie lähes aina enemmän aikaa kuin julkaisupaketille suoritettavat automaattiset testit. Testaajan työpanoksen säästämisen vuoksi, manuaalista testausta tulisikin suorittaa vasta kaikkien automaattisten testien läpäisyn jälkeen. Manuaalisen testauksen tarkoituksena ei ole suorittaa toistuvia regressiotestauksia, vaan varmistaa, että automaattisesti suoritettavat ja läpäistyt hyväksyntätestit todella täyttävät niiden hyväksyntäkriteerit. (Humble & Farley 2010, 128; Honda 2015).

Kaikki toimitusputken aiemmat vaiheet ovat tähänneet toimivan julkaisupaketin tuotantoon julkaisemiseen. Tässä vaiheessa julkaisupaketille on suoritettu useita eri testauksia, joiden avulla on varmistettu sen toimivuus sekä sille asetettujen kriteerien täyttyminen. Julkaiseminen tuotantoon ei kuitenkaan eroa juurikaan eri testausympäristöihin julkaisemisesta, sillä samaa julkaisuprosessia tulisi käyttää kummassakin tapauksessa. Erot tuotantoon ja testausympäristöön julkaisemisesta onkin lähestulkoon vain ympäristön asetustiedostoissa. (Humble & Farley 2010, 249).

Tuotantoon julkaisemisessa on tärkeää pystyä päivittämään ohjelmisto ilman käyttökatkoja. Päivitettyyn ohjelmistoon siirtymiseen on kehitetty kaksi tehokasta tapaa: blue-green-julkaisu ja canary-julkaisu. Tavat toimivat melko samalla tavalla, mutta niiden käyttötarkoituksissa on eroja. Molemmissa luodaan ensin tuotantoympäristölle identtinen ympäristö, jossa voidaan alkuun päivitetylle ohjelmistolle suorittaa testausta. Humble & Farley 2010, 260–263; Wilsenach 2016).

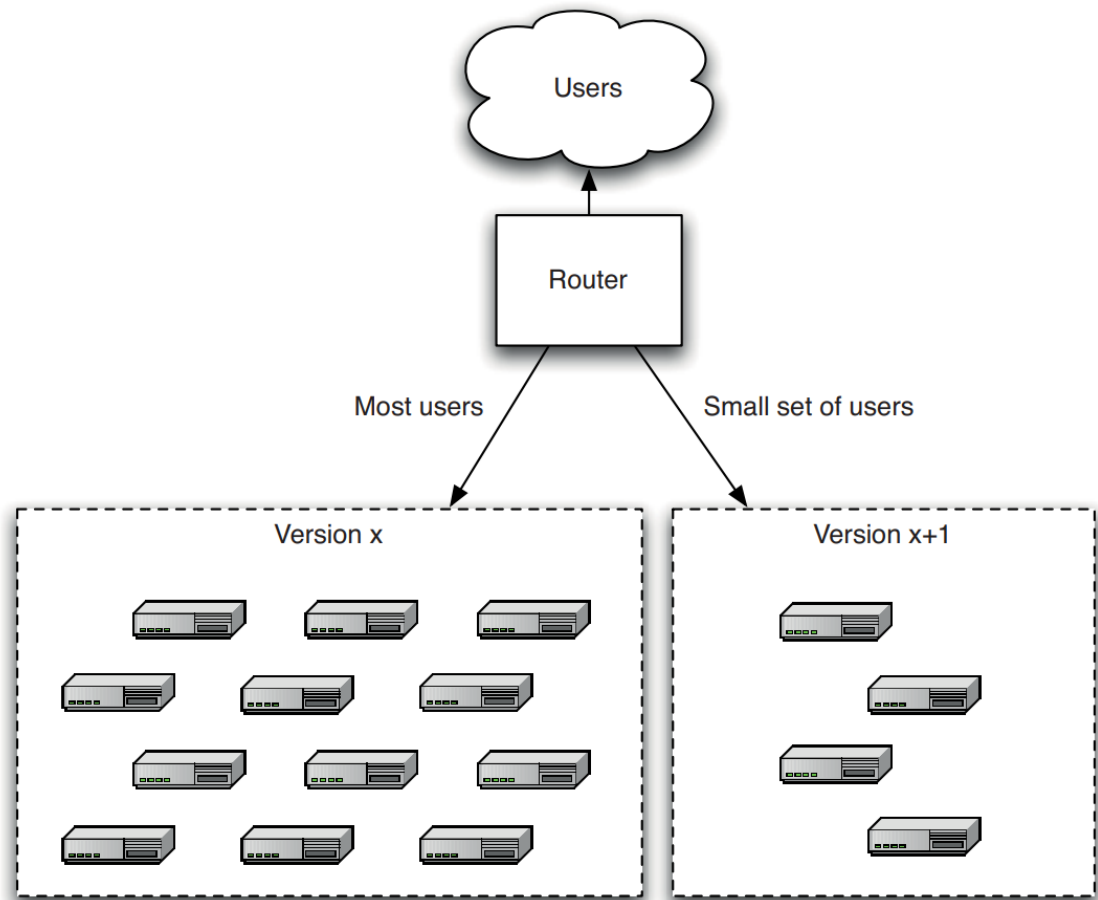
Blue-green-julkaisussa tarkoituksena on saada kaikki uudet käyttäjät heti siirrettyä käyttämään päivitettyä ohjelmistoa. Tällöin uudella ympäristöllä toteutettujen testausten jälkeen palvelin asetetaan viittaamaan siihen, jolloin kaikki uudet muodostetut yhteydet alkavat käyttämään sitä. Vanhaan ympäristöön ei tule enää uusia yhteyksiä ja kaikki yhteydet vanhaan ympäristöön häviävät vähitellen. Mikäli blue-green-julkaisun uudessa ympäristössä ilmenee yhteyksien siirron jälkeen jokin virhe, voidaan palvelin palauttaa ohjaamaan käyttäjät takaisin vanhaan ympäristöön. Blue-green-julkaisu on esitetty kuvassa 9. (Fowler 2010; Humble & Farley 2010, 261–262).



Kuva 9. Blue-green-julkaisu (Humble & Farley 2010, 261).

Canary-julkaisun tarkoituksena on saada ensin testattua päivitetty ohjelmisto pienellä osalla käyttäjistä ja vasta myöhemmin siirtää loput yhteydet uuteen ympäristöön. Usein ensimmäiset päivitetyn ohjelmiston käyttäjät ovat erikseen valikoituja, kuten esimerkiksi beetestaukseen ilmoittautuneita, käyttäjiä, joilta saadaan tietoa uusien ominaisuuksien käytöstä ja toimivuudesta. Lisäksi käyttäjämäärän kontrolloimisella pystytään myös testaamaan esimerkiksi ohjelmiston kapasiteettia. Kun ohjelmisto on todettu toimivaksi ja halutuksi pienellä osalla käyttäjistä, voidaan kaikki yhteydet säätää viittaamaan päivitetyn ohjelmiston ympäristöön. Mikäli taas uudessa ominaisuudessa ilmenee joitain virheitä tai huonouksia, voidaan käyttäjät palauttaa vanhaan ympäristöön helposti ja tutkia rauhassa kerättyä palautetta sekä suorittaa sen synnyttämiä korjaustoimenpiteitä ennen uuden ominaisuuden uudelleen julkaisua. Canary-julkaisussa voidaan myös käyttää kerralla useampaa kuin vain yhtä kopiota tuotantoympäristöstä, jolloin pystytään

valikoiduilla käyttäjillä testaamaan useampia uusia ominaisuuksia. Canary-julkaisu on esitetty kuvassa 10. (Humble & Farley 2010, 263–264; Sato 2014).



Kuva 10. Canary-julkaisu (Humble & Farley 2010, 263).

Tuotantoon julkaisemisen jälkeen ohjelmisto on käytössä oikeilla käyttäjillä oikeassa maailmassa, jolloin jossakin vaiheessa siinä varmasti ilmenee jokin virhe mikä on päässyt kaikkien testauksien läpi. Tällöin tarvitsee tehdä pikaisesti korjaus ja päivittää asiakkaiden käytössä oleva ohjelmisto. Vaikka korjauksella on kiire, se kannattaa kuitenkin ajaa toimitusputken läpi, jotta se on kontrolloitu ja testattu. Tällä pyritään ehkäisemään uusien ongelmien luonti vahingossa ja varmistamaan korjauksen dokumentointi. Lisäksi kun korjaus on tehty toimitusputken kautta, voidaan olla varmoja, ettei siihen tai tuotantoympäristöön ole tehty mitään manuaalisia muutoksia, joita ei taas olla dokumentoitu. Toinen vaihtoehto korjaukselle on palata aiempaan, toimivaksi tiedettyyn versioon ohjelmistosta, jolloin virheen etsimiseen ja korjaamiseen voidaan kuluttaa enemmän aikaa ja

siten pienentää uuden virheen syntymisen mahdollisuutta. (Humble & Farley 2010, 265–266).

### 2.5.3 Jatkuva palaute

Jatkuvassa palautteessa tarkkaillaan ja kerätään tietoa tuotantoympäristön tapahtumista, jotta niihin voidaan reagoida ja niitä voidaan käyttää toiminnan kehittämisessä ja tulevan toiminnan suunnittelussa. Jatkuva palaute kattaa DevOpsin toimitusputkesta käyttö- ja valvontavaiheet, joiden lisäksi jatkuvasta palautteesta saatuja tietoja hyödynnetään toimitusputken suunnitteluosassa. Jatkuva palaute sisältää neljä asiaa, jotka tulee ottaa huomioon sen toteutuksessa: infrastruktuuri, josta pystytään keräämään tarvittavat tiedot, tapahtumien tallennus, jotta ne ovat helposti noudettavissa, graafinen käyttöliittymä, jolla pystytään tutkimaan kerättyä tietoa ja ilmoitukset, jotka ilmoittavat oleellisille henkilöille tapahtumista. (Humble & Farley 2010, 317; Pennington 2019).

Kerätty informaatio voi sisältää monenlaista tietoa, kuten laitteiston lämpötiloja, käyttöjärjestelmän muistinkulutuksia, väliohjelmiston (engl. middleware) yhteyksien määriä tai sovellusten käyttötapoja. Useimmat näistä informaatioista pystytään keräämään erillisillä työkaluilla, mutta usein sovelluksista mitattavat tiedot ovat sovelluksesta riippuvaisia ja tällöin niiden keruu pitää erikseen ohjelmoida mukaan sovellukseen. Tällöin myös infrastruktuurissa tulee ottaa huomioon sovelluksen tiedonkeruumenetelmät, jotta kerättyihin tietoihin päästään helposti käsiksi ja ne pystytään tallentamaan analysointia ja hyödyntämistä varten. (Humble & Farley 2010, 318–319).

Tapahtumat tallennetaan lokien muodossa, jolloin niitä on helppo seurata ja niitä voidaan helposti hyödyntää käyttäjien suorittamien toimien sekä mahdollisten ongelmien tarkkailuun. Lokimerkintöjen tulee sisältää tapahtumien ajankohdat, tapahtumien vakavuustasot, tapahtumien sijainnit ja mahdolliset virhekoodit sekä -kuvaukset. Lokien pitää kuitenkin olla tarpeeksi lyhyitä, jotta niiden tarkasteleminen on tehokasta. Useimmat laitteistot, käyttöjärjestelmät ja väliohjelmistot sisältävät valmiiksi automaattiset lokien muodostamiset, mutta sovellukseen lokien muodostaminen pitää tehdä erikseen. (Humble & Farley 2010, 320).

Graafista käyttöliittymää käytetään viestimään selkeästi eri palvelujen tiloja ja siitä nähdään nopeasti yhdellä vilkaisulla palvelujen mahdolliset virhetilat. Se toimii korkeammalla tasolla kuin lokimerkinnät, jolloin graafisessa käyttöliittymässä tuleekin näkyä



aikatietojen lisäksi vain palvelun tämänhetkinen tilanne ja siihen liittyvä kuvaus. Graafinen käyttöliittymä vähentää turhaa lokien selailua, kun palvelu toimii virheettää. Virheiden ilmetessä, käyttöliittymä ohjaa kuvauksen avulla vastuuhenkilön suoraan oikeaan suuntaan, jolloin taas ajan kuluttamista lokien selailuun ja virheen etsimiseen saadaan pienennettyä. Graafisessa käyttöliittymässä hyödynnetään usein vihreää, keltaista ja punaista väriä palveluiden tilojen ilmaisemiseen, joista vihreä tarkoittaa palvelun toimivan odotetusti, keltainen tarkoittaa palvelun toiminnassa tapahtuneen jokin odottamaton tapahtuma ja punainen tarkoittaa palvelun olevan toimimaton. (Humble & Farley 2010, 321–322; Oh 2019).

Jatkuvan seurannan ilmoitukset voidaan toteuttaa tapahtumia seuraamalla ja testejä suorittamalla. Useimmissa graafisissa käyttöliittymissä on toteutettuna tapahtumien seuraamiseen perustuva ilmoitusjärjestelmä, joka virheen tapahtuessa lähettää vastuuhenkilölle ilmoituksen esimerkiksi sähköpostilla. Tapahtumien seuraamista voidaan toteuttaa myös lokien kautta, mutta tällöin ilmoitusjärjestelmästä tarvitsee usein toteuttaa jonkinlainen oma ratkaisu. Testien avulla toteutettu seuranta toimii samankaltaisesti kuin jatkuvan integraation palautejärjestelmä: testin epäonnistuessa, siitä ilmoitetaan vastuuhenkilöille määritellyllä tavalla. (Humble & Farley 2010, 323).

### 3 TUTKIMUKSEN TOTEUTUS

Tutkimuksen tiedonhankinnan tarkoituksena on kartoittaa toimeksiantajan tuotekehityksen tämänhetkinen tila ja siinä käytössä olevat käytännöt, jotta niitä voidaan verrata DevOps-toimintamallin käytäntöihin. Vertailulla pyritään saamaan selville DevOpsin toteutamista varten toimeksiantajayrityksen tuotekehitykseen vaaditut toimenpiteet sekä sitä mahdollisesti haittaavat asiat. Vertailun tuloksien avulla muodostetaan tarpeelliset kehitysehdotukset, jotka toteuttamalla yritys pystyy aloittamaan DevOpsin täyden hyödyntämisen.

Tutkimuksen tiedonhankinnan menetelminä käytetään kirjoittajan omaa kokemusta toimeksiantajayrityksen tuotekehityksestä, yrityksen henkilöstölle toteutettavia haastatteluja ja toimeksiantajayrityksen tuotekehityksestä luotuja dokumentteja. Dokumenttianalyysillä pyritään saamaan yleinen kokonaiskuva toimeksiantajan tuotekehityksen toiminnasta ja siihen liittyvistä asioista. Haastatteluilla taas pyritään täydentämään dokumenttianalyysistä saatavaa kokonaiskuva hyödyntämällä työntekijöiden dokumentoimaton ja kokemusperäistä tietoa, eli niin sanottua hiljaista tietoa. Dokumenttianalyysistä ja haastatteluista saatavaa tietoa täydennetään kirjoittajan omalla tiedolla ja kokemuksella toimeksiantajan tuotekehityksestä, jotta tiedoista saadaan muodostettua mahdollisimman selvä kokonaisuus.

#### 3.1 Dokumenttianalyysin toteutus

Toimeksiantajayrityksen tuotekehitykseen liittyvät dokumentit etsittiin toimeksiantajan käyttämästä Redmine-projektihallintaohjelmistosta. Ohjelmistossa dokumentteina toimivat sinne tallennetut wikisivut, jotka ovat järjestelty otsikoittain. Tämä taas helpotti yrityksen tuotekehitykseen ja QEM Publisher -tuotteeseen liittyvän informaation löytämistä.

Otsikoiden alla olevan tiedon lisäksi käytettiin projektihallintaohjelmiston etsintätyökalua, jolla etsittiin tietyillä avainsanoilla lisätietoa tuotekehityksestä ja QEM Publisher -tuotteesta. Avainsanoina käytettiin "qem publisher"-tuotenimeä sekä "qemds"-projektinimeä. Etsintätyökalun kanssa etsittiin avainsanoja "Documents"-, "Wiki pages"- ja "Messages"-kategorioiden alta.

### 3.2 Haastattelun toteutus

Haastattelut toteutettiin toimeksiantajayrityksen ohjelmistokehitykseen, järjestelmäylläpitoon ja muuten toimeksiantajan tuotekehitykseen sekä varsinkin QEM Publisher -tuotteeseen liittyville henkilöille. Haastattelut sovittiin haastatteluun suostuvien henkilöiden kanssa pidettäväksi heidän työajallaan joko pikaviestimiä käyttäen tai kasvotusten toimeksiantajayrityksen Turun toimiston tiloissa. Potentiaalisille haastateltaville kerrottiin haastattelun suorittamisen syy sekä miksi haastateltava on valittu haastatteluun. Lisäksi haastattelua sopiessa haastateltaville selvennettiin, ettei heidän nimeään paljasteta haastattelun tuloksissa, vaan haastattelun tulosten läpikäynnissä keskitytään vain heidän työtehtäviinsä ja toimintatapoihin.

Haastattelukysymykset rakentuivat vastuualueista ja työtehtävistä sekä DevOps-toimintamallin pääpiirteistä ja vaatimuksista. Haastattelun suunnittelussa käytettiin apuna Ruusuvooren ja Tiittulan (2005) sekä Kanasen (2014) teoksia haastattelusta ja laadullisesta tutkimuksesta. Kaikki suoritettavat haastattelut tehtiin kasvotusten haastateltavien kanssa ja niiden muistiinpanot kirjattiin ylös erilliselle paperille.

Haastattelut aloitettiin antamalla haastateltaville taustatietoa DevOps-toimintamallista, jotta haastattelukysymysten vastaukset saatiin ohjattua ja kohdennettua enemmän DevOpsin toteuttamiselle oleelliseen tietoon. Tämän jälkeen haastattelussa edettiin liitteessä 1 esitetyn haastattelurungon kysymysten mukaisesti ja esitettiin kohdissa jatkokysymyksiä, mikäli vastauksista jäi haastattelijalle jotain vielä epäselväksi. Lisäksi teemojen kysymysten jälkeen esitettiin avoin kysymys, jonka tarkoituksena oli antaa haastateltavalle mahdollisuus kertoa teemaan liittyvistä asioista, joita haastattelija ei ollut osannut kysyä.

### 3.3 Tutkimuksen luotettavuus

Haastattelut toteutettiin liian aikaisin työn käytännönosan kirjoittamiseen nähden ja työssä jouduttiinkin turvautumaan vahvasti haastatteluista tehtyihin muistiinpanoihin, koska selkeitä muistikuvia ei enää ollut kaikesta mitä haastatteluissa oli käyty läpi. Tästä saattoi aiheutua se, että joitain pieniä asioita on voinut jäädä ottamatta huomioon työssä, mikäli niitä ei kirjoitettu haastattelun aikana ylös muistiinpanoihin. Opinnäytetyön käytännön osaa kirjoittaessa selvisi myös, että haastattelukysymykset olisivat voineet olla

parempia, eli kirjoittajan olisi pitänyt kysyä joidenkin aihealueiden kohdalla hieman eri kysymyksiä. Kysymysten olisi pitänyt enemmän keskittyä vastuualueiden toimintoihin, toteutuksien sijaan. Tämä kuitenkin tarkoitti lähinnä vain sitä, että tarvittava tieto piti koostaa muista tietolähteistä. Dokumenttianalyysiä kuitenkin suoritettiin koko työn käytännönsan kirjoittamisen ajan.

Työssä käytettiin kohtalaisen paljon blogilähteitä, jotka laskevat luotettavuutta DevOpsin määrittelyyn, sillä niitä ei ole vertaistarkastettu samalla tasolla kuin tieteellisiä artikkeleita. Useiden blogilähteiden käyttö johtui kuitenkin tieteellisten artikkeleiden ja virallisten dokumenttien vähydestä, joka taas luultavasti johtuu DevOpsin puuttuvasta virallisesta määritelmästä. Kirjoittaja kuitenkin pyrki löytämään lähteen sisältämän tiedon useammasta eri lähteestä, varmistaakseen sen oikeellisuuden.

Opinnäytetyössä käytettyjen tutkimusmenetelmien avulla saatiin selvitettyä toimeksiantajayrityksen tuotekehityksen toiminta tarpeeksi hyvin, jotta sitä pystyttiin vertaamaan DevOps-toimintamalliin. Kirjoittaja oli lisäksi suurimman osan opinnäytetyön kirjoitusaikasta läheisessä tekemisessä toimeksiantajayrityksen tuotekehityksen kanssa, joka mahdollisti tuotekehityksessä tapahtuvien muutosten huomioon ottamisen työssä. Toimeksiantajayrityksen tuotekehitykseen ei löydetty montaa DevOpsille tärkeää kehitysehdotusta, joka vastasi opinnäytetyön tehtävänannossa kirjoittajalle sekä toimeksiantajalle muodostunutta kuvaa tarpeellisista kehityskohteista.

## 4 TOIMEKSIANTAJAN TUOTEKEHITYKSEN NYKYTOIMINTA

Tiedonkeruumenetelmien avulla hankittiin tietoa ja selvitettiin toimeksiantajayrityksen tuotekehityksen nykyistä toimintaa. Haastatteluiden ja dokumenttianalyysin kanssa saatiin melko hyvin selvitettyä tuotekehityksen tämänhetkinen toiminta. Puuttuvia tietoja täydennettiin kirjoittajan omalla kokemuksella ja tiedolla yrityksen tuotekehityksen toiminnasta. Kirjoittajan omaa kokemusta ja tietoa hyödynnettiin myös tiedonkeruusta saatujen tietojen järjestelyssä loogiseksi kokonaisuudeksi.

Selvitettyä toimeksiantajayrityksen tuotekehityksen toimintaa tarkastellaan DevOpsin toimitusputken osien mukaisesti, jotta DevOpsiin vertailu ja kehitettävien asioiden tuominen ilmi on helpompaa sekä selkeämpää. Osien toiminnasta muodostetaan julkisen hallinnon tietohallinnon neuvottelukunnan (2012) JHS 152 suosituksen mukaiset prosessikaaviot, joissa määritellään pääpiirteittäin tapahtumat ja tiedonkulku. Prosessikaaviot rakennetaan hyödyntäen tiedonhankintamenetelmien avulla saatua tietoa toimeksiantajan tuotekehityksen toiminnasta ja niitä selvennetään vielä selittävillä teksteillä.

Tämänhetkisen toiminnan mukauttaminen DevOpsin toimitusputken osiin saattaa kuitenkin aiheuttaa lyhyitäkin prosessikaavioita, sillä osa toimitusputkien vaiheista eivät sisällä montakaan toimintoa. Mikäli prosessikaavio jää niin lyhyeksi, ettei sitä ole mielekästä kuvata omana prosessikaavionaan, yhdistetään se toimitusputken aiemman tai seuraavan osan kanssa.

Suunnitteluvaihe tapahtuu liitteen 2 mukaisesti alkaen asiakkaan tekemästä tilauksesta, joka voi käytännössä olla tilaus uudelle ominaisuudelle tai räätälöidylle sisällölle. Tämän jälkeen tuotekehityksen johdon henkilöstö tekee päätöksen uuden ominaisuuden tai räätälöidyn sisällön toteuttamisesta ja ilmoittaa siitä asiakkaalle. Mikäli asiakkaan tilaus päätetään toteuttaa, tehdään siitä vaatimusmäärittely, jonka tekniset tiedot kirjataan järjestelmään kehittäjiä varten.

Kehittäjä saa ilmoituksen, hänelle määritellystä, järjestelmään kirjatusta tilauksesta järjestelmän kautta ja toteuttaa alustavan suunnitelman tilauksen tietojen perusteella. Alustavan suunnitelman avulla kehittäjä näkee vaikuttavatko tehtävät asiat laajemmin ohjelmistoon sekä sen rajapintoihin, jolloin voidaan tarvita palaveria muiden kehittäjien kanssa. Mikäli palaverille on tarvetta, tuotekehityksen johto järjestää palaverin

tarvittavan henkilöstön kanssa. Palaverin jälkeen tai mikäli palaveria ei ole tarvittu, kehittäjä toteuttaa lopullisen suunnitelman, jota käytetään asiakkaan tilauksen toteuttamisessa.

Ohjelmointivaihe alkaa aiemmassa vaiheessa luodun suunnitelman toteuttamisen aloittamisella ja se toteutetaan liitteen 3 mukaisesti. Kehittäjä tuottaa ohjelmoimalla suunnitelmassa määritellyn ominaisuuden tai virheen korjauksen ohjelmistoon, jonka aikana sekä sen jälkeen toteuttaa tehdyille muutoksille staattista koodianalyysia. Ohjelmoinnin ja staattisen koodianalyysin jälkeen kehittäjä viimeistelee ohjelmiston asetukset, mikäli asetusten muuttaminen on tarpeen.

Kehittäjä ajaa komentosarjan ohjelmiston paikalliseen kääntämiseen ja paketoimiseen ja odottaa ilmoitusta sen onnistumisesta tai epäonnistumisesta. Tällöin, komentosarjan avulla, ohjelmistolle toteutetaan myös useita eri tarkastuksia. Mikäli paketti ei käänny tai paketoitu onnistuneesti, etsitään siitä epäonnistumisen aiheuttava virhe ja palataan korjaamaan se. Jos taas paketti kääntyy ja paketoituu onnistuneesti, sille suoritetaan manuaalista testausta.

Kääntämisvaihe toteutetaan liitteen 4 mukaan. Kääntämisvaihe alkaa muutosten lataamisesta versionhallintaan, jolloin virtuaalipalvelin käynnistää automaattisesti paketin kääntämisen. Tällöin virtuaalipalvelin hakee versionhallinnasta uusimmat tiedostot ja ajaa käynnön komentosarjan sekä paketoit tiedostot julkaisupaketiksi. Virtuaalipalvelin toimittaa kehittäjälle tiedon kääntämisen ja paketoinnin onnistumisesta tai epäonnistumisesta. Mikäli kääntäminen ja paketointi onnistuu, virtuaalipalvelin siirtää julkaisupaketin koekäyttöympäristöön.

Testausvaihe tapahtuu liitteen 5 mukaisesti alkaen tehdyn muutoksen manuaalisesta testauksesta. Mikäli muutos ei läpäise testausta, ilmoitetaan kehittäjälle paketissa olevasta virheestä. Jos kuitenkin muutos läpäisee testauksen, tuotekehityksen johto päättää uuden julkaisupaketin tuomisesta tuotantoon, jolloin sille toteutetaan manuaalista julkaisutestausta ennen tuotantoon tuomista. Mikäli julkaisupaketti ei läpäise julkaisutestausta, ilmoitetaan siinä testauksen aikana ilmenneestä virheestä kehittäjälle.

Julkaisu- ja käyttöönottovaiheet toteutetaan liitteen 6 mukaisesti alkaen julkaisupaketin käyttöönoton päätöksestä. Mikäli julkaisupakettia ei oteta käyttöön, jäädään odottamaan useamman julkaisuun tulevan muutoksen valmistumista. Jos julkaisupaketti kuitenkin otetaan käyttöön, tuodaan se tuotantoympäristöön asiakkaiden saataville. Tämän jälkeen uuden julkaisun saatavilla olosta ilmoitetaan asiakkaille, jotka ottavat sen käyttöön.

Käyttö- ja valvontavaiheet alkavat uuden asiakkaan tapauksessa asiakkuuden luomisesta sekä tuotekehityksen johdon järjestämästä käyttökoulutuksesta ja se toteutetaan liitteen 7 mukaisesti. Sekä uudet että vanhat asiakkaat käyttävät ohjelmistoa ja ilmoittavat käytönaikana havaituista virheistä tuotekehityksen johdolle. Mikäli asiakkailta tulee ilmoituksia tapahtuneista virheistä, johto etsii ohjelmiston lokitiedoista lisää tietoa virheistä ja kirjaa ne järjestelmään korjausta varten.

## 5 KEHITYSEHDOTUKSET TOIMEKSIANTAJAN TUOTEKEHITYKSEEN

Toimeksiantajan tämänhetkinen tuotekehityksen toiminta käsiteltiin DevOpsin toimitusputken osien mukaan asioiden esittämiseksi selkeästi. DevOps kuitenkin pohjautuu eri näisiin käytäntöihin, jonka takia ehdotettavia muutoksia toimeksiantajan tuotekehitykseen käsitellään työssä käsiteltyjen käytäntöjen avulla. Tässä luvussa verrataan tuotekehityksen tämänhetkisen tilan vastaavuutta DevOpsissa käytettäviin käytäntöihin ja mikäli tarpeellista, ehdotetaan tämänhetkiseen toimintaan muutoksia. Näiden muutosehdotusten tarkoituksena on auttaa toimeksiantajaa aloittamaan DevOps-toimintamallin täyden toteuttamisen.

Kehitysehdotuksissa keskitytään selvittämään tarpeelliset toimenpiteet ottamatta juuriin kantaa miten nämä toimenpiteet tulisi toteuttaa. Joihinkin kehitysehdotuksiin voidaan ideoida hieman toteutuksien alkua, mutta lopullisten toteutuksien tutkiminen ja päättäminen jäävät toimeksiantajayrityksen tehtäväksi. Kehitysehdotuksien toteuttamisen jälkeen yritys voi lisäksi hyödyntää tämän työn tietoja ja tuloksia DevOps-toimintamallin laajentamisessa yrityksen muuhun tuotekehitykseen. Laajentamista varten toimeksiantajayrityksen kannattaa kuitenkin toteuttaa vastaavanlainen tutkimus myös yrityksen muille tuotteille, sillä ne eroavat vahvasti tässä työssä käsitellystä tuotteesta esimerkiksi jatkuvan toimituksen ja jatkuvan käyttöönoton kannalta.

### 5.1 Jatkuva integraatio

Versionhallinta on toimeksiantajan tuotekehityksessä ollut jo pidemmän aikaa käytössä. Toimeksiantaja on ymmärtänyt versionhallinnan hyödyt ja käyttää sitä kaikessa tuotekehityksessään. Versionhallintaan ei välttämättä siis tarvita minkäänlaisia muutoksia, mutta asioiden selkeyttämiseksi ja virheiden mahdollisuuden minimoimiseksi siihen ehdotetaan kuitenkin ominaisuuspohjaisen haaroituksen ja vetoehdotusten (engl. pull request) käyttöönottoa.

Ominaisuuspohjainen haaroitus selkeyttäisi eri tehtävien (engl. ticket) suorittamista ja niitä varten tehtyjen muutosten tutkimista. Käytännössä ominaisuuspohjaisessa haaroituksessa jokaisesta tehtävästä tehdään versionhallintaan oma haara, johon tehtävän



vaatimat muutokset tehdään. Tämä myös mahdollistaa helpon tehtävästä toiseen siirtymisen vain vaihtamalla työstettävää haaraa, mikäli esimerkiksi tehtävien tärkeysjärjestys muuttuu.

Haarat nimetään tehtävät identifioivilla merkkisarjoilla, jotta haarat ovat helppo tunnistaa ja yhdistää niitä vastaaviin tehtäviin. Lyhyt identifioiva merkkisarja voi olla esimerkiksi QEM Publisher -tuotteesta käytetyn QemDs-projektinimen lyhennelmä ja tehtävän juokseva numero: "QDS-1234". Merkkisarjan lisäksi haaran nimeen on hyvä lisätä muutaman sanan kuvaus tehtävästä, kuten esimerkiksi tehtävän otsikko. Tällöin haaran sisältämät muutokset pystytään tunnistamaan joutumatta etsimään tehtäväjärjestelmästä tehtävän koko kuvausta. Tehtäviä varten tehdyt haarat poistetaan, kun niiden sisältämät muutokset on todennettu toimiviksi koko ohjelmistossa. Ominaisuuspohjaisen haaroituksen tehokas käyttöönotto edellyttää kuitenkin tehtäväjärjestelmän nykyiseen käyttöön pieniä muutoksia, joita käsitellään myöhemmin.

Vetoehdotuksien käytössä versionhallinnan kehityshaaraan ladatuista muutoksista tehdään erillinen pyyntö, joka tarvitsee hyväksynnän toiselta kehittäjältä ennen kuin muutokset voidaan päivittää versionhallinnan päähaaraan. Tällä saadaan lisättyä muutosten hallintaa ja vähennettyä riskiä ongelmalliselle julkaisulle, koska jokainen tehty muutos on katsottu läpi vähintään kahdesti. Toimeksiantajayrityksen ei kuitenkaan kannata ottaa vetoehdotuksia käyttöön ennen kuin tuotekehityksen henkilöstömäärä on kasvanut hie-man, sillä tällä hetkellä yrityksen tuotekehityksen henkilöt ovat vahvasti erikoistuneet projekteissa tiettyihin osa-alueisiin ja näistä osa-alueista ei löydy kovin paljon päällekkäisyyttä.

Automaattinen kääntäminen on varsinaisen kääntämisen puolesta hyvässä mallissa. Jokainen versionhallintaan ladattu muutos käännetään automaattisesti virtuaalipalvelimen avulla omaksi julkaisupaketiksi. Käännön aikana julkaisupakettiin tulevat tiedostot haetaan sekä kootaan yhteen virtuaalipalvelimelle ja niihin tehdään muutokset, mikäli ne ovat kohdealustalla ohjelmiston toiminnan kannalta tarpeellisia. Lisäksi kääntämisen aikana suoritetaan lähdekoodille syntaksitarkastusta.

Jokaisesta kääntämisestä tuotetaan raportit, oli kääntäminen onnistunut tai epäonnistunut. Raporteissa ilmaistaan paketoinnin onnistuminen tai epäonnistuminen, julkaisupakettiin tulevat muutokset, muutosten tekijä, paketointi vaiheessa suoritettut paketoinnin vaiheet ja paketoinnin epäonnistuessa, epäonnistumisen aiheuttanut syy. Linkki raporttiin lähetetään pikaviestinsovelluksen kanavalle.

Palautejärjestelmä on toteutettu Slack-pikaviestintäsovelluksen avulla, johon lähetetään viesti jokaisen julkaisupaketin kääntämisen jälkeen. Viesti sisältää tiedon käännetyistä projektista, onnistumisesta tai epäonnistumisesta ja linkin virtuaalipalvelimen tuottamaan raporttiin kääntämisestä. Viesti lähetetään kääntämisen tuloksille tarkoitetulle kanavalle pikaviestinsovelluksessa, johon on liittynyt kaikki toimeksiantajayrityksen tuotekehityksen työntekijät. Tämä toimintamenetelmä toimii tällä hetkellä hyvin, koska kääntämissä ei tapahdu työpäivän aikana montaa eivätkä oleelliset ilmoitukset pääse hukkumaan informaatiovirtaan. Lisäksi tämä toimintamenetelmä mahdollistaa nopean palautteen kääntämisen onnistumisesta tai epäonnistumisesta, jolloin ilmaantuvat virheet päästään korjaamaan heti.

Mikäli kuitenkin toimeksiantajayrityksen ohjelmistokehittäjien ja työpäivän aikana käännettävien projektien määrät kasvavat paljon, voi yksi kanava kaikille kääntämisen viesteille olla riittämätön. Tällöin kanavalle voi alkaa tulemaan niin paljon informaatiota eri projekteista, että tuotekehityksen henkilöstö ei jaksakaan enää aktiivisesti seurata pikaviestinsovellukseen tulevaa tietovirtaa. Tätä pystytään ehkäisemään ottamalla käyttöön ilmoituksissa sähköpostien tai pikaviestinsovelluksessa kohdistettujen viestien lähettämisen, jolloin projektin kääntämisen tulokset voidaan toimittaa vain projektille oleellisille henkilöille, yleisen viestikanavan sijaan.

Testaus ja tarkastelu käsittää sekä ennen versionhallintaan lataamista paikallisesti tehdyt että kääntämisvaiheessa virtuaalipalvelimen tekemät testaukset ja tarkastelut. Toimeksiantajayrityksellä ei ole kirjoitettua ohjeistusta paikallisesti suoritettavasta testauksesta ja tarkastelusta. Tarkastelua toteuttaa kuitenkin jokainen ohjelmistokehittäjä automaattisesti kirjoittaessaan ohjelmakoodia ja paikallisen testauksen suorittaminen on jo lähestulkoon vakiintunut käytännöksi. Testauksesta voi kuitenkin olla hyvä tuottaa kirjallinen ohje, jonka tarkoituksena on toimia ohjeena uusille ja muistutuksena vanhoille työntekijöille. Lisäksi kirjallinen ohje toimii yhtenä todistuksena tuotteen laadusta.

Paikallisen testauksen ja tarkastelun suorittamisen jälkeen muutokset ladataan versionhallintaan, jolloin virtuaalipalvelin aloittaa kääntämisprosessin ja toteuttaa sen aikana syntaksitarkastusta. Kääntämisprosessissa ei kuitenkaan tällä hetkellä suoriteta ollenkaan yksikkö- eikä integraatiotestausta, joiden toteuttamisella voitaisiin lisätä varmuutta ohjelmiston toimivuudesta sekä sitä kautta antaa vahvemman todistuksen tuotteen laadusta. Lisäksi kääntämisen aikana voitaisiin toteuttaa erityyppisten sääntöjen tarkastelua, kuten koodirivien määrää yhdessä tiedostossa tai funktiossa. Näiden sääntöjen tarkoituksena olisi ylläpitää yrityksen sisäisiä ohjelmointistandardeja. Sääntöpohjaisen

tarkastelun toteuttamista varten toimeksiantajayrityksen tarvitsee myös päättää ja luoda kirjalliset ohjeet yrityksen haluamista ohjelmointitavoista, jotka helpottavat ja yhtenäistävät ohjelmistojen kehitystä.

Henkilöstö on saatu sitoutettua jatkuvan integraation kääntämisprosessin hyväksi havaittuihin toimintatapoihin, johtuen ainakin osittain toimeksiantajan tuotekehitykseen liittyvän henkilöstön vaikutusmahdollisuuksista kyseisiin toimintatapoihin. Lisäksi toimeksiantajayrityksen tuotekehityksen henkilöstöstä suurin osa on yrityksen pidempiaikaisia työntekijöitä, jotka ovat olleet luomassa näitä vakiintuneita toimintatapoja. Toimintatavat ovat kuitenkin lähinnä vain hiljaisena tietona tuotekehityksen nykyisellä henkilöstöllä ja niistä olisi syytä tehdä varsinaiset kirjalliset ohjeet, joita pystytään hyödyntämään esimerkiksi uusien työntekijöiden perehdyttämisessä.

Jatkuvan integraation käytännön kirjallisten ohjeiden luonnissa toimeksiantajayrityksen kannattaa osallistaa ja hyödyntää tuotekehityksen henkilöstöä, jotta heidän tietonsa saadaan sisällytettyä ohjeistukseen ja pystytään samalla antamaan tuotekehityksen henkilöstölle mahdollisuus esittää uusia ideoita ohjeistukseen. Kirjallisten ohjeiden teossa tulisi kuitenkin ottaa huomioon ja mukailla jatkuvan integraation käytännön seitsemää toimintavaihetta.

Toimeksiantajayrityksen tuotekehityksen henkilöstölle on hyvä painottaa versionhallinnan aktiivista käyttöä, jotta pienien muutosten usein lisäämisestä versionhallintaan syntyy vakiintunut käytäntö ja se tapahtuu luonnostaan. Tässä auttaa myös tehtäväjärjestelmän käytön lisääminen pienempien tehtävien ja suurempien tehtävien pienempiin osiin jakamisen kautta. Tehtävät pysyvät silloin selkeinä ja helposti hallittavina sekä seurattavina. Tehtävienjako ei myöskään tällöin pääse tapahtumaan ohimennen kahvipöydässä tai jonkun muun keskustelun lomassa, jolloin pystytään lähestulkoon poistamaan riski annetun tehtävän unohtamisesta.

Henkilöstön sitouttamista voidaan myös edistää päivittäisillä palavereilla, joissa jokainen tuotekehityksen työntekijä kertoo lyhyesti edellisen palaverin jälkeen tehdyistä asioista sekä suunnitelmista ennen seuraavaa palaveria. Päivittäisillä palavereilla tuotekehityksen johto pystyy seuraamaan helposti kulloinkin toteutuksen alla olevia tehtäviä ja mahdollisia ongelmatapauksia voidaan ratkoa koko tuotekehityksen henkilöstön kanssa. Tällä hetkellä toimeksiantajayrityksen tuotekehityksen henkilöstön määrä on sen verran pieni, että tuotekehityksen johto on hyvin perillä toteutuksessa olevista tehtävistä ja ongelmatapauksissa asioista pystytään helposti keskustelemaan ilman erillisten palaverien

sopimista. Toimeksiantajayrityksen voi kuitenkin olla hyvä aloittaa päivittäisten palaverien pitäminen, jotta ne vakiintuvat yrityksessä normaaleiksi käytännöiksi ja ovat valmiiksi opeteltuina yrityksen tuotekehityksen kasvaessa.

Jatkuvan integraation kehitysehdotuksissa käsiteltiin käytäntöön kuuluvat osa-alueet, joita olivat versionhallinta, automaattinen kääntäminen, palautejärjestelmä, testaus ja tarkastelu sekä henkilöstön sitouttaminen. Vertaamalla käytännön osa-alueita toimeksiantajan tuotekehityksen tämänhetkiseen tilaan, löydettiin joitain muutosehdotuksia, mutta yleisesti toiminta on tällä hetkellä jo melko lähellä tavoitetta.

## 5.2 Jatkuva toimitus ja jatkuva käyttöönotto

DevOps-toimintamallin tuotantoympäristöt liittyvät usein vahvasti selainpohjaisiin tuotteisiin, joissa tuotteen tai palvelun tulee olla lähes koko ajan asiakkaan saatavilla. Toimeksiantajayrityksen tuote on tässä kuitenkin erilainen, koska sen käyttöympäristöinä toimivat erilaiset näyttölaitteet, jotka tarvitsevat vain asennuspaketin tuotantoympäristöstä. Asennuspakettia käytetään sekä ohjelmiston ensimmäiseen asennukseen että sen päivittämiseen, mutta näistäkin vain ensimmäinen asennus on kriittinen ohjelmiston käytön kannalta, sillä ellei asennuspakettiin pääse vakavaa virhettä, päivitykset sisältävät usein vain uusia ominaisuuksia. Lisäksi toimeksiantajayritys kontrolloi ohjelmiston käytön lisenssejä, jolloin yritys on hyvin tietoinen kulloinkin tapahtuvasta ensimmäisestä asennuksesta.

Toimeksiantajayrityksellä on tällä hetkellä erilliset kehitys- ja tuotantoympäristöt, joista molemmista julkaistaan asennuspaketit pysyviin koekäyttöympäristöihin. Koekäyttöympäristöt toimivat testausympäristöinä, joista hyväksyntätestatut julkaisupaketit siirretään asiakkaan saataville. Tämä toimii hyvin, sillä ympäristöjä ei juurikaan päivitetä, vaan päivitys tapahtuu päivittämällä niiden sisältämiä asennuspaketteja. Lisäksi yrityksen ei ole mielekästä pyrkiä toteuttamaan jatkuvaa käyttöönottoa, jossa määritetty testausympäristö olisi tarpeellinen. Tämä taas johtuu siitä, että yrityksen infonäyttöohjelmisto-tuotetta on hyvin hankala testata automaattisesti sen visuaalisuuden vuoksi ja se tarvitsee siis manuaalisen testauksen ennen kuin asennuspaketti voidaan siirtää asiakkaan saataville.

Hyväksyntätestauksessa suoritetaan tällä hetkellä manuaalisesti toiminnallisia ja ei-toiminnallisia testejä. Tuotteelle toteutettavia toiminnallisia testejä ovat yleisesti ohjelmiston toimivuutta varmistavia, kuten mediatiedostojen oikein näkymisen ja muiden

ominaisuuksien toimivuuden testaamista. Toteutettavat ei-toiminnalliset testit taas ovat lähinnä ohjelmiston suorituskykyä ja kestävyyttä testaavia testejä.

Käytännössä testaaminen tapahtuu toimeksiantajayrityksen hankkimilla erillisillä näyttölaitteilla, jotka vastaavat asiakkailta käytössä olevia laitteita. Laitteelle ladataan sitä vastaavan alustan julkaisupaketti ja siihen päivitetään esityslistat sekä säädetään käyttöön testattavat ominaisuudet, kuten esimerkiksi laitteen ajastettu sammutus ja käynnistys. Julkaisupaketin testaus tarvitsee toteuttaa manuaalisesti, sillä monet virheet eivät aiheuta laitteelle kirjattavaa virhetilaa, vaikka testattava ominaisuus ei toimitakaan oikein. Nämä virheet ovat usein alustasta riippuvaisia, mutta myös yleisiä alustasta riippumattomia virheitä löytyy. Tällainen yleinen virhe voi olla esimerkiksi esitettävän median näkyminen väärässä kohdassa näyttölaitteella, joka ei kuitenkaan aiheuta minkäänlaista ohjelmallista virhettä, jonka taas pystyisi kirjaamaan lokiin.

Alustoilla on myös usein omat ohjelmointiympäristöt, joiden kautta laitteiden toimintaa pystytään simuloimaan. Simulointi ei kuitenkaan ole tarpeeksi kehittynyt siihen, että kaikkia toimeksiantajayrityksen infonäyttöohjelmisto-tuotteeseen lisättyjä ominaisuuksia pystyttäisiin testaamaan laitteen simuloinnin kautta ohjelmointiympäristössä, sillä tietyt laitteissa olevat ominaisuudet eivät vain yksinkertaisesti toimi laitteen toimintaa simuloimassa. Mikäli laitevalmistajat kehittävät tulevaisuudessa laitteiden simulointia riittävälle tasolle, toimeksiantajayrityksen kannattaa tutkia ja harkita automaattisen testauksen toteuttamista julkaisupaketin soveltuville osille.

Hyväksyntätestaukseen, manuaalisen testauksen tueksi, toimeksiantajayrityksen kannattaa luoda testauksen suunnitelma ja ohjeet, sillä yrityksellä ei tällä hetkellä ole kirjallista ohjeistusta julkaisupaketin testaukseen. Tällä saadaan varmennettua jokaisen vaatimuksen täyttyminen ja julkaisupaketin kaikkien ominaisuuksien toiminta, jolloin yritys saa taas lisää kirjallista näyttöä läpinäkyvyydestä ja tuotteen laadusta. Ohjeistusta tukemaan yrityksen kannattaa myös luoda testaukseen määriteltyjä esityslistoja, jotka sisältävät jokaisen tuetun mediatyyppin, esityslistaan sidotut ominaisuudet ja ongelmia aiheuttaneet asiakkaiden lataamat mediatiedostot.

Julkaiseminen tuotantoon tarkoittaa yrityksen infonäyttöohjelmisto-tuotteella käytännössä sen asennuspaketin siirtoa asiakkaan saataville. Tämän jälkeen riippuen alustasta asiakkaan näyttölaitte joko lataa ja asentaa käynnistykseen yhteydessä automaattisesti uuden julkaisupaketin version tai asiakkaan tarvitsee erikseen laitteelta käskyttää julkaisupaketin päivitys. Tuotteen käyttö tapahtuu siis paikallisesti asiakkaan hankkimalla

laitteella ja tuotantoympäristönä toimii paikka, jossa pidetään julkaisupaketteja asiakkaiden saatavilla.

Yrityksen infonäyttöohjelmisto-tuotteen julkaiseminen tuotantoon tapahtuu tällä hetkellä melko harvoin, sillä uusille asennuspaketeille suoritettava manuaalinen testaus vie paljon aikaa. Tuotteen vahvan visuaalisuuden ja visuaalisten virheiden ilmoitusten puuttumisen vuoksi, manuaalista testausta ei kuitenkaan voida paljoa vähentää jatkuvan toimituksen testauksen osalta. Tällöin tuote ei myöskään luultavasti koskaan tule toteuttamaan jatkuvaa käyttöönottoa, jossa tuote julkaistaisiin automaattisesti tuotantoon.

Toimeksiantajayrityksen ei kannata myöskään yrittää nopeuttaa tuotteen julkaisua tuotantoon ottamalla käyttöön beetatestaajia ja jakamalla sitä kautta testaustaakkaa asiakkaiden kanssa, vaikka halukkaita beetatestaajia asiakkaista löytyisikin. Yrityksen infonäyttöohjelmisto-tuote on tarkoitettu näkyville julkisille paikoille, joissa yhtenä tärkeimmistä ominaisuuksista on tuotteen toimivuus. Tällöin ei asiakkaille kannata antaa testausvaiheessa olevia julkaisupaketteja, joissa riski virheille on suurempi kuin kunnolla testatuissa julkaisupaketeissa. Jokainen yleisölle näkyvä virhe heikentää luottamusta ja asettaa yrityksen tuotteen naurunalaiseksi.

Jatkuvan toimituksen ja jatkuvan käyttöönoton käytännön käsiteltyihin osa-alueisiin kuuluivat ympäristöt, hyväksyntätestaus ja julkaiseminen tuotantoon. Käytännön osa-alueista ei löydetty DevOpsin kannalta mitään suuria muutoksen tarpeita, mutta osa-alueisiin pyrittiin ehdottamaan toimintaa yleisesti hyödyttäviä muutoksia.

### 5.3 Jatkuva palaute

Yrityksen infonäyttöohjelmisto-tuotteeseen on tehty menetelmät, joiden avulla pystytään keräämään sekä näyttölaitteiden että ohjelmiston toiminnasta informaatiota. Näyttölaitteiden informaation, kuten lämpötilan tai muistin kulutuksen, haussa hyödynnetään laitteen omia mekanismeja informaation saamiseen. Ohjelmiston tapahtumien informaation keräämisessä hyödynnetään toimeksiantajayrityksen, tuotteeseen tehtyjä, lokitiedostoja ja niihin kirjoittamista. Tuotteeseen ei kuitenkaan ole rakennettu tapaa seurata ominaisuuksien käyttötapoja tai määriä, jotka voisivat antaa toimeksiantajayritykselle tarkempaa tietoa halutuista ominaisuuksista sekä ideoita niiden jatkokehityksen suunnalle. Tämän informaation kerääminen voisi olla hyödyllistä tuotteen uusien ominaisuuksien ideoimisen kannalta, jolloin niitä voisi myös hyödyntää uusien asiakkaiden hankkimisessa.

Ohjelmisto kerää tällä hetkellä tietoja tapahtumista ja tallentavat ne lokitiedostoihin, jotka voidaan ladata erillisen rajapinnan kautta tarkasteltaviksi. Kerätty tieto on suurimmaksi osaksi ohjelmiston tapahtumia, sillä näyttölaitteet antavat hyvin vähän informaatiota niiden yleisestä toiminnasta sekä mahdollisista virheistä. Laitteilta voidaan kuitenkin erillisillä kyselyillä hakea tietoja, mutta niistä saatavan informaation määrä riippuu kuitenkin paljon laitteen valmistajasta. Toimeksiantajayritys voi parantaa näyttölaitteista saatavaa informaation määrää lisäämällä tuotteeseen ajastettujen kyselyiden suorittamisen näyttölaitteilla, joiden vastaukset tallennettaisiin lokitiedostoihin ja siten tuotaisiin helpommin saataville. Kyselyissä yritys pystyy hyödyntämään tuotteeseen jo valmiiksi tehtyjä näyttölaitteen informaation kyselymenetelmiä.

Toimeksiantajayrityksellä on toteutettuna graafinen käyttöliittymä infonäyttöohjelmisto-tuotteen toiminnan seurantaan varten selaimella. Tämä on toteutettu yrityksen toisen tuotteen kautta, joka toimii infonäyttöohjelmiston hallintarajapintana. Vaikka graafinen käyttöliittymä onkin toteutettu yrityksen toisella tuotteella, se liittyy niin vahvasti infonäyttöohjelmistoon, että sitä käsitellään näissä kehitysehdotuksissa graafisen käyttöliittymän puolesta osana yrityksen infonäyttöohjelmisto-tuotetta.

Näyttölaitteiden toiminnan seuraaminen perustuu tällä hetkellä esityslistojen hakemiseen ja sen onnistumisen seurantaan. Mikäli infonäyttöohjelmisto on hakenut esityslistan tietyn aikavälin sisällä, annetaan sen tilan olla käyttöliittymässä väritön, joka ilmaisee näyttölaitteen olevan yhdistettynä hallintarajapintaan. Jos taas ohjelmisto ei ole hakenut esityslistaa, merkataan sen tila punaiseksi.

Graafista käyttöliittymää voidaan kehittää yhdistämällä ohjelmiston tilan päättelyyn esityslistan päivityksen lisäksi lokitiedostojen lukua ja lisäämällä mahdollisten tilojen värikoodeja. Käytännössä muutos tarkoittaisi nykyisen palvelimen puoleisten lokien lukemisen lisäksi ohjelmiston lokitiedostojen hakemista ja lukemista, joka voitaisiin toteuttaa rajapinnan puolella olevilla ajastetuilla kyselyillä tai esityslistojen hakemisesta laukeavilla kyselyillä. Ohjelmalta haetuista lokitiedostoista etsittäisiin tietyltä, esimerkiksi viimeisen 24 tunnin, ajalta merkintöjä virheistä ja merkittäisiin ohjelmiston tila käyttöliittymään löydösten perusteella. Aikarajan tarkoituksena olisi ehkäistä ohjelman pysyvät keltaiset tilat johtuen lokissa olevista vanhoista ei kriittisistä virhemerkinnöistä.

Muutoksen jälkeen ohjelmiston tila ja värikoodit olisivat seuraavan listan mukaiset:

- Vihreä - Ei virheitä
  - Esityslista haettu

- Ei virhettä
- Keltainen - Ei kriittisiä virheitä
  - Esityslista haettu
  - Virhe tapahtunut, joka ei vaikuta mediatiedostojen esittämiseen
- Punainen - Kriittinen virhe
  - Esityslista haettu
  - Virhe tapahtunut, joka vaikuttaa mediatiedostojen esittämiseen
- Harmaa - Ei yhteyttä
  - Esityslistaa ei haettu
  - Oletettavasti näyttölaitteelle ei ole yhteyttä

Haasteena ja jatkokehitettävänä asiana on tunnistaa ja saada kirjattua lokitiedostoihin ohjelmiston kaatavat kriittiset virheet. Tämä on haasteellista, sillä näyttölaitteet antavat harvoin minkäänlaista informaatiota ohjelmistolle tämän kaltaisista virheistä. Mikäli haaste saadaan ratkaistua, pystytään erottelemaan toisistaan ja kirjaamaan lokitiedostoihin oikein laitteen tarkoituksellinen sammuttaminen sekä ohjelmiston virheestä johtuva järjestelmän kaatuminen.

Toimeksiantajayritys saa tällä hetkellä tiedon ongelmista ohjelmistossa asiakkaan ilmoituksesta. Tätä voidaan kehittää automaattisiin ilmoituksiin, mikäli graafisen käyttöliittymän rajapintaa laajennetaan tarkkailemaan näyttölaitteissa olevien ohjelmistojen tiloja. Tällöin rajapintaan voidaan tehdä viestin lähetyksen vastuuhenkilöille virheiden sattuessa. Ilmoitukset tulee kuitenkin tällöin pystyä asettamaan pois päältä näyttölaite kohtaisesti, jotta kehitysvaiheessa ei pääse syntymään viestitulvaa. Tuotannossa mahdollisen viestitulvan ehkäisemiseksi, ei-kriittisten virheiden ilmoitukset voisi asettaa pois päältä tietyn ajanjakson jälkeen, jolloin huomio saadaan keskitettyä pääasiallisesti kriittisiin virheisiin. Kriittisten virheiden havaitsemisessa on kuitenkin aiemmin mainitut haasteet, joihin toimeksiantajayrityksen kannattaa pyrkiä saamaan toteutettua ratkaisu. Ratkaisu kuitenkin luultavimmin riippuu näyttölaittevalmistajista ja heidän laitteistojensa ohjelmistoista.

Jatkuvan palautteen käytännön käsiteltäviä osa-alueita olivat infrastruktuuri, tapahtumien tallennus, graafinen käyttöliittymä ja ilmoitukset. Osa-alueiden kehitysehdotukset auttavat antamaan paremman kuvan ohjelmiston ja näyttölaitteiden tilasta, mutta ne eivät ole välttämättömiä DevOpsin toimitusputken toiminnalle. Osa kehitysehdotuksista on myös paljon riippuvaisia laitevalmistajista, eikä toimeksiantajayritys välttämättä niitä edes pysty tällä hetkellä toteuttamaan.



## LÄHTEET

Agile Manifesto. 2001. Julistuksen takana olevat periaatteet. Viitattu 4.7.2019 <https://agilemanifesto.org/iso/fi/principles.html>

Aljundi, M. 2018. Tools and practices to enhance DevOps core values. Diplomityö. Tietotekniikan koulutusohjelma. Lappeenranta: Lappeenranta-Lahden teknillinen yliopisto. Viitattu 21.7.2019 <http://lutpub.lut.fi/bitstream/handle/10024/148944/DevOps.pdf>

Bass, L; Weber, I & Zhu, L. 2015. DevOps: A Software Architect's Perspective. New York: Addison-Wesley Professional

Bourque, P & Fairley, R. 2014. Guide to the Software Engineering Body of Knowledge. IEEE Press

Canty, D. 2015. Agile for Project Managers. Boca Raton: CRC Press

Chokshi, J. 2019. What You Need to Know About DevTestOps. Viitattu 5.1.2020 <https://dzone.com/articles/what-you-need-to-know-about-devtestops-1>

de França, B; Jeronimo, H & Travassos G. 2016. Characterizing DevOps by Hearing Multiple Voices. New York: ACM Press

Devopsdays. 2019. About Devopsdays. Viitattu 1.7.2019 <https://devopsdays.org/about/>

Duvall, P.; Matyas, S. & Glover, A. 2007. Continuous Integration: Improving Software Quality and Reducing Risk. Boston: Addison-Wesley

Dyck, A; Penners, R & Lichter, H. 2015. Towards Definitions for Release Engineering and DevOps. IEEE Press

Edwards, D. 2009a. How to measure the impact of IT operations on your business (Part 1). Viitattu 3.2.2020 <http://dev2ops.org/2009/11/how-to-measure-the-impact-of-it-operations-on-your-business-part-1/>

Edwards, D. 2009b. Q&A: Lee Thompson, former Chief of Technologist of E\*TRADE Financial. Viitattu 4.1.2020 <http://dev2ops.org/2009/09/qa-lee-thompson-former-chief-technologist-of-etrade-financial/>

Edwards, D. 2010a. How to measure the impact of IT operations on your business (Part 2). Viitattu 8.8.2019 <http://dev2ops.org/2010/01/how-to-measure-the-impact-of-it-operations-on-your-business-part-2/>

Edwards, D. 2010b. What is DevOps?. Viitattu 4.1.2020 <http://dev2ops.org/2010/02/what-is-devops/>

Fowler, M. 2006. Continuous Integration. Viitattu 21.11.2019 <https://martinfowler.com/articles/continuousIntegration.html>

Fowler, M. 2010. BlueGreenDeployment. Viitattu 4.1.2020 <https://martinfowler.com/bliki/Blue-GreenDeployment.html>

Fowler, M. 2013. ContinuousDelivery. Viitattu 27.11.2019 <https://martinfowler.com/bliki/ContinuousDelivery.html>

Honda, M. 2015. Why exploratory testing is critical for DevOps. Viitattu 5.1.2020 <https://tech-beacon.com/devops/why-exploratory-testing-critical-devops>

- Humble, J & Farley, D. 2010. Continuous Delivery – Reliable Software Releases through Build, Test, and Deployment Automation. 1. painos. Boston: Addison-Wesley
- Hüttermann, M. 2012. DevOps for Developers. 1. painos. New York: Apress
- Jabbari, R; bin Ali; Petersen, K & Tanveer, B. 2016. What is DevOps?. New York: ACM Press
- Jawale, C. 2017. SecOps: The Next Stride for DevOps. Viitattu 5.1.2020 <https://devops.com/secops-next-stride-devops/>
- Julkisen hallinnon tietohallinnon neuvottelukunta. 2012. JHS 152 Prosessien kuvaaminen. Viitattu 20.2.2020 <http://docs.jhs-suositukset.fi/jhs-suositukset/JHS152/JHS152.html>
- Kananen, J. 2014. Laadullinen tutkimus opinnäytetyönä – Miten kirjoitan kvalitatiivisen opinnäytetyön vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu
- Kim, G. 2012. The Three Ways: The Principles Underpinning DevOps. Viitattu 10.8.2019 <http://itrevolution.com/the-three-ways-principles-underpinning-devops/>
- Kim, G.; Humble, J.; Debois, P. & Willis, J. 2016. The DevOps Handbook. 1. painos. Portland: IT Revolution Press
- Kornilova, I. DevOps is a culture, not a role!. Viitattu 22.10.2019 <https://medium.com/@neon-rocket/devops-is-a-culture-not-a-role-be1bed149b0>
- Lietz, S. 2015. What is DevSecOps? Viitattu 5.1.2020 <https://www.devsecops.org/blog/2015/2/15/what-is-devsecops>
- Little, C. 2016. Why Is There No DevOps Manifesto? Viitattu 4.7.2019 <https://devops.com/no-devops-manifesto/>
- Lwakatare, L. 2017. DevOps adoption and implementation in software development practice. Väitöskirja. Tieto- ja sähkötekniikan tiedekunta. Oulun yliopiston tutkijakoulu. Oulu: Oulun yliopisto. Viitattu 4.7.2019 <http://jultika.oulu.fi/files/isbn9789526217116.pdf>
- Mikita, D.; DeHoldt, G. & Nezlek, G. 2012. The Deployment Pipeline. Viitattu 16.11.2019 <http://proc.conisar.org/2012/pdf/2215.pdf>
- Minick, E. 2015. Surprise! Broad Agreement on the Definition of DevOps. Viitattu 30.7.2019 <https://devops.com/surprise-broad-agreement-on-the-definition-of-devops/>
- Oh, D. 2019. 3 types of metric dashboards for DevOps teams. Viitattu 5.1.2020 <https://opensource.com/article/19/7/dashboards-devops-teams>
- Panko, R. 2017. The Elusive Definition of DevOps. Viitattu 6.7.2019 <https://devops.com/elusive-definition-devops/>
- Patel, C. Secure and Scalable CI/CD Pipeline With AWS. Viitattu 30.10.2019 <https://dzone.com/articles/secure-and-scalable-cicd-pipeline-with-aws>
- Penners, R & Dyck, A. 2015. Release Engineering vs. DevOps – An Approach to Define Both Terms. Aachen: RWTH Aachen University
- Pennington, J. 2019. The Eight Phases of a DevOps Pipeline. Viitattu 6.11.2019 <https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba>
- Pepgotesting 2019. Automated software testing in Continuous Integration (CI) and Continuous Delivery (CD). Viitattu 19.11.2019 <http://www.pepgotesting.com/continuous-integration/>
- Perera, P; Silva, R & Perera, I. 2017. Improve software quality through practicing DevOps. IEEE Press

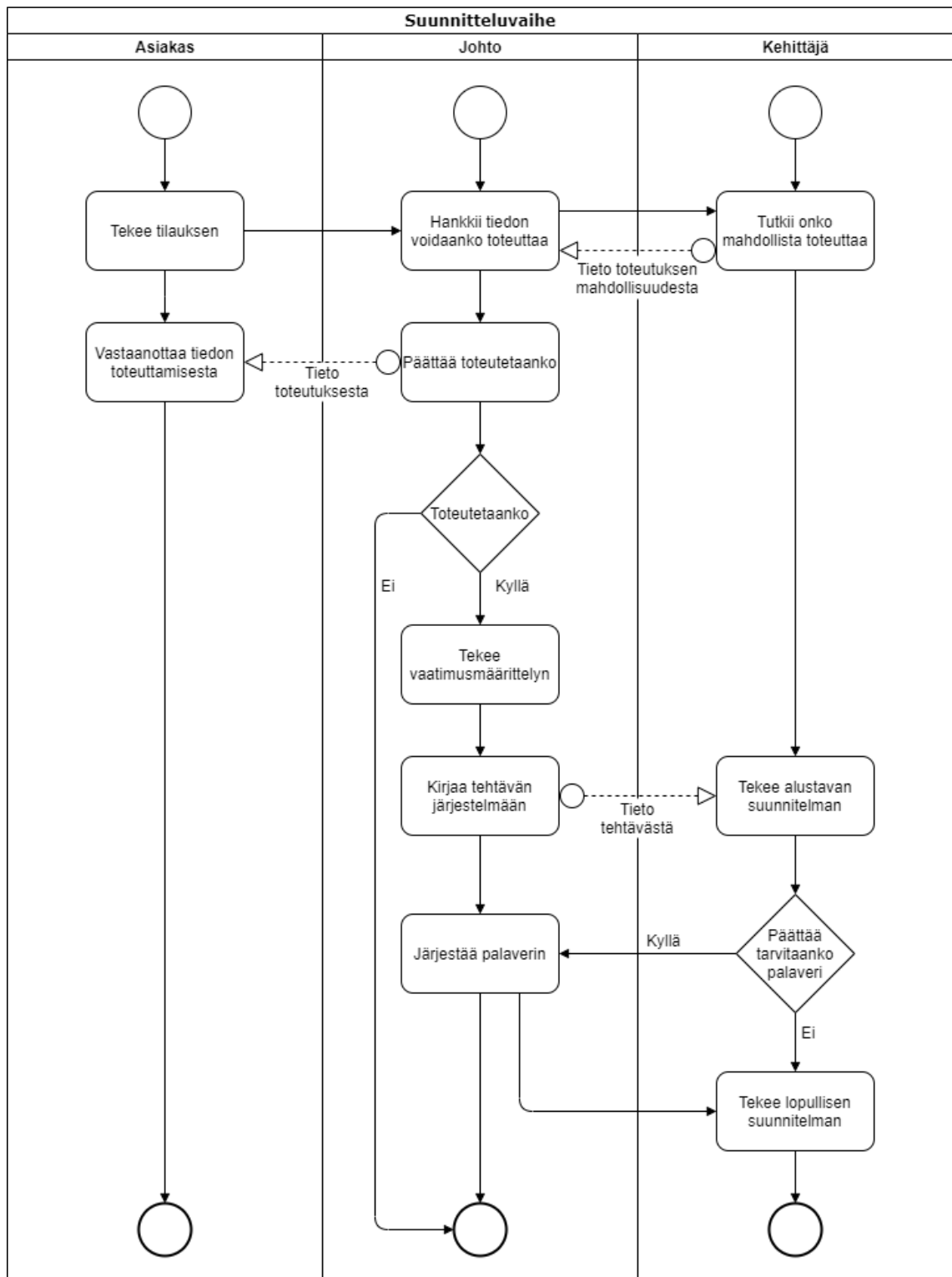
- Rehn, C. 2016. Preventing human error by automating application configuration. Viitattu 14.8.2019 <https://www.devopsonline.co.uk/preventing-human-error-by-automating-application-configuration/>
- Riley, C. 2014. The Delivery Pipeline is your DevOps Signature. Viitattu 22.10.2019 <https://devops.com/delivery-pipeline-devops-signature/>
- Riungu-Kalliosaari, L; Mäkinen, S; Lwakatare, L; Tiihonen, J & Männistö, T. 2016. DevOps Adoption Benefits and Challenges in Practice: A Case Study. New York: Springer Publishing. Viitattu 29.8.2019 <https://www.cs.helsinki.fi/u/jutiihon/publications/RiunguKalliosaari2016DevopsAdoptionBenefits.pdf>
- Ruusuvuori, J. & Tiittula, L. 2005. Haastattelu – Tutkimus, tilanteet ja vuorovaikutus. Tampere: Vastapaino
- Sato, D. 2014. CanaryRelease Viitattu 4.1.2020 <https://martinfowler.com/bliki/CanaryRelease.html>
- Senapathi, M; Buchan, J & Osman, H. DevOps Capabilities, Practices, and Challenges: Insight from a Case Study. New York: ACM Press
- Seppä-Lassila, T. 2017. An Assesment of DevOps Maturity in a Software Project. Pro gradu - tutkielma. Tietojenkäsittelytiede. Turku: Turun yliopisto. Viitattu 21.7.2019 [https://www.utu-pub.fi/bitstream/handle/10024/146617/gradu\\_seppa-lassila\\_final\\_pdfa.pdf](https://www.utu-pub.fi/bitstream/handle/10024/146617/gradu_seppa-lassila_final_pdfa.pdf)
- Seroter, R. 2014. Exploring the ENTIRE DevOps Toolchain for (Cloud) Teams. Viitattu 30.10.2019 <https://www.infoq.com/articles/devops-toolchain/>
- Shahin, M; Babar, M. & Zhu, L. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. Viitattu 16.11. 2019 <https://ieeexplore.ieee.org/document/7884954>
- Sharma, S & Coyne, B. 2017. DevOps for Dummies. 3. IBM rajoitettu painos. Hoboken: John Wiley & Sons, Inc
- Smeds, J; Nybom, K & Porres, I. 2015. DevOps: A Definition and Perceived Adoption Impediments. Cham: Springer
- Smith, P. 2011. Software Build Systems – Principles and Experience. 1. PAINOS. Boston: Addison-Wesley
- Stafford, T. 2017. How to Implement DevOps: The CAMS Approach. Viitattu 4.8.2019 <https://shadow-soft.com/how-to-implement-devops/>
- Tuli, S. 2018. Learn How to Set Up a CI/CD Pipeline From Scratch. Viitattu 3.11.2019 <https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch>
- Willis, J. 2010. What Devops Means to Me. Viitattu 21.7.2019 <https://blog.chef.io/2010/07/16/what-devops-means-to-me/>
- Wilsenach, R. 2015. DevOpsCulture. Viitattu 20.10.2019 <https://martinfowler.com/bliki/DevOpsCulture.html>
- Wilsenach, R. 2016. 3 DevOps techniques for stress-free release management. Viitattu 5.1.2020 <https://techbeacon.com/enterprise-it/3-devops-techniques-stress-free-release-management>
- Zalavadia, S. 2019. Functional Testing: A Complete Guide With Types And Examples. Viitattu 2.12.2019 <https://www.softwaretestinghelp.com/guide-to-functional-testing/>

Zislis, E. DevOps Transformation Using Theory of Constraints – Part 2. Viitattu 3.2.2020  
<https://dzone.com/articles/devops-transformation-using-theory-of-constraints-1>

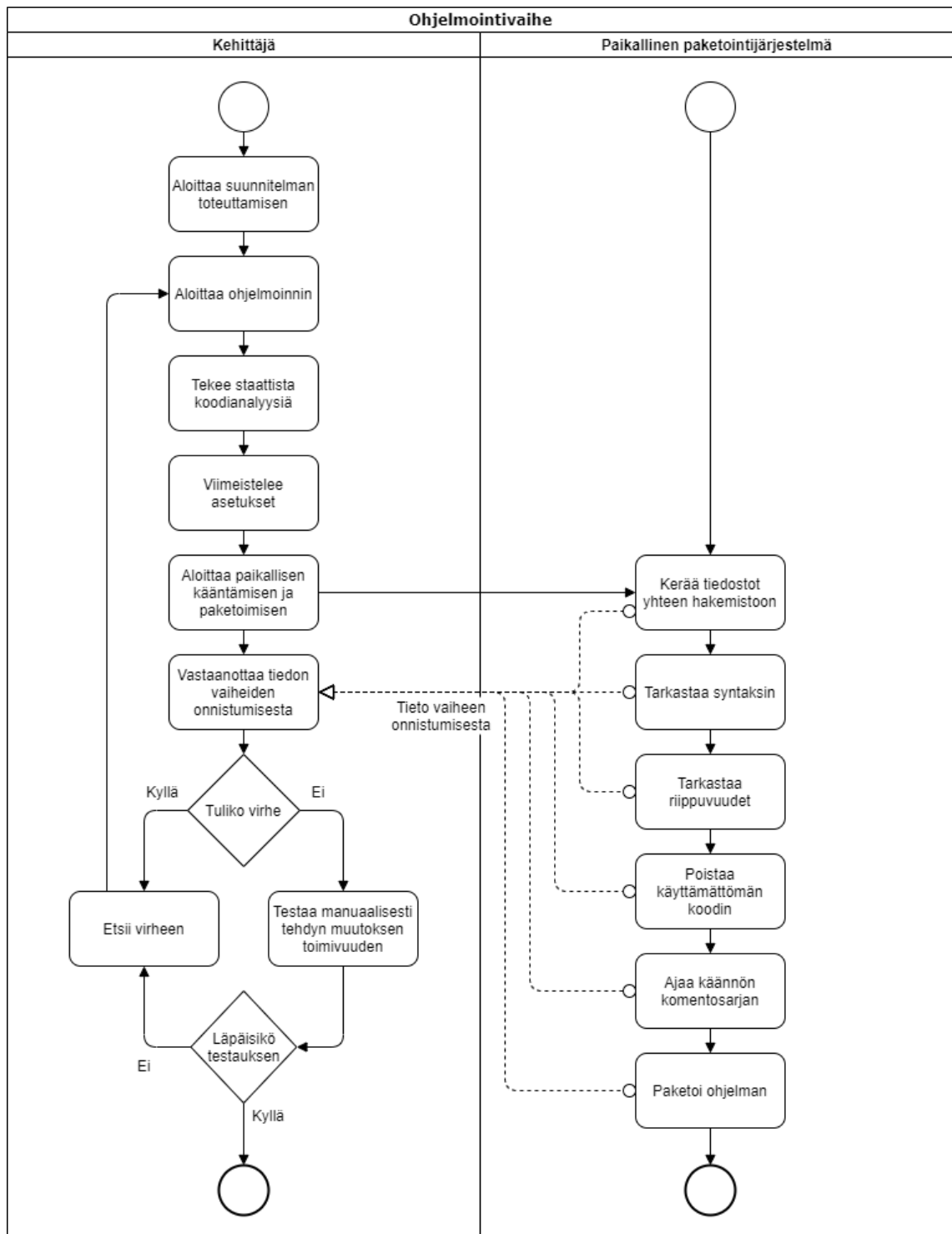
## Haastattelurunko

Yrityksen taustatiedot	
Nimi	Qem Software Oy
Toimiala	Ohjelmistojen suunnittelu ja tuottaminen
Henkilöstö	7 henkilöä (2019)
Haastattelun toteutus	
Haastattelija	Tero Elmroos
Ajankohta	
Haastattelun kesto	Arvioitu 45 min, toteutunut:
Haastateltava	
Asema	
Teemat ja kysymykset	
Teema 1	<p>Vastuualueet ja työtehtävät</p> <ul style="list-style-type: none"> <li>• Mitkä ovat vastuualueet tuotekehityksessä?</li> <li>• Miten vastuualueen työtehtävät toteutetaan?</li> <li>• Mitä ohjelmistoja käytetään vastuualueiden työtehtävissä?</li> </ul>
Teema 2	<p>Ketterien menetelmien toteutuvuus</p> <ul style="list-style-type: none"> <li>• Miten ketterät menetelmät toteutuvat vastuualueen työtehtävissä?</li> <li>• Miten tämän hetkinen toteutus on laajennettavissa?</li> <li>• Miten tämän hetkinen toteutus on muokattavissa?</li> </ul>
Teema 3	<p>Automatisointi</p> <ul style="list-style-type: none"> <li>• Mitä voidaan automatisoida vastuualueella?</li> <li>• Onko jotain automatisoitu jo vastuualueella? <ul style="list-style-type: none"> <li>○ Mitä?</li> <li>○ Miten?</li> </ul> </li> <li>• Mitkä vastuualueiden työtehtävistä vie eniten aikaa? <ul style="list-style-type: none"> <li>○ Voiko näitä automatisoida?</li> </ul> </li> <li>• Onko vastuualueiden työtehtävissä joitain toistuvia töitä? <ul style="list-style-type: none"> <li>○ Mitä?</li> </ul> </li> </ul>

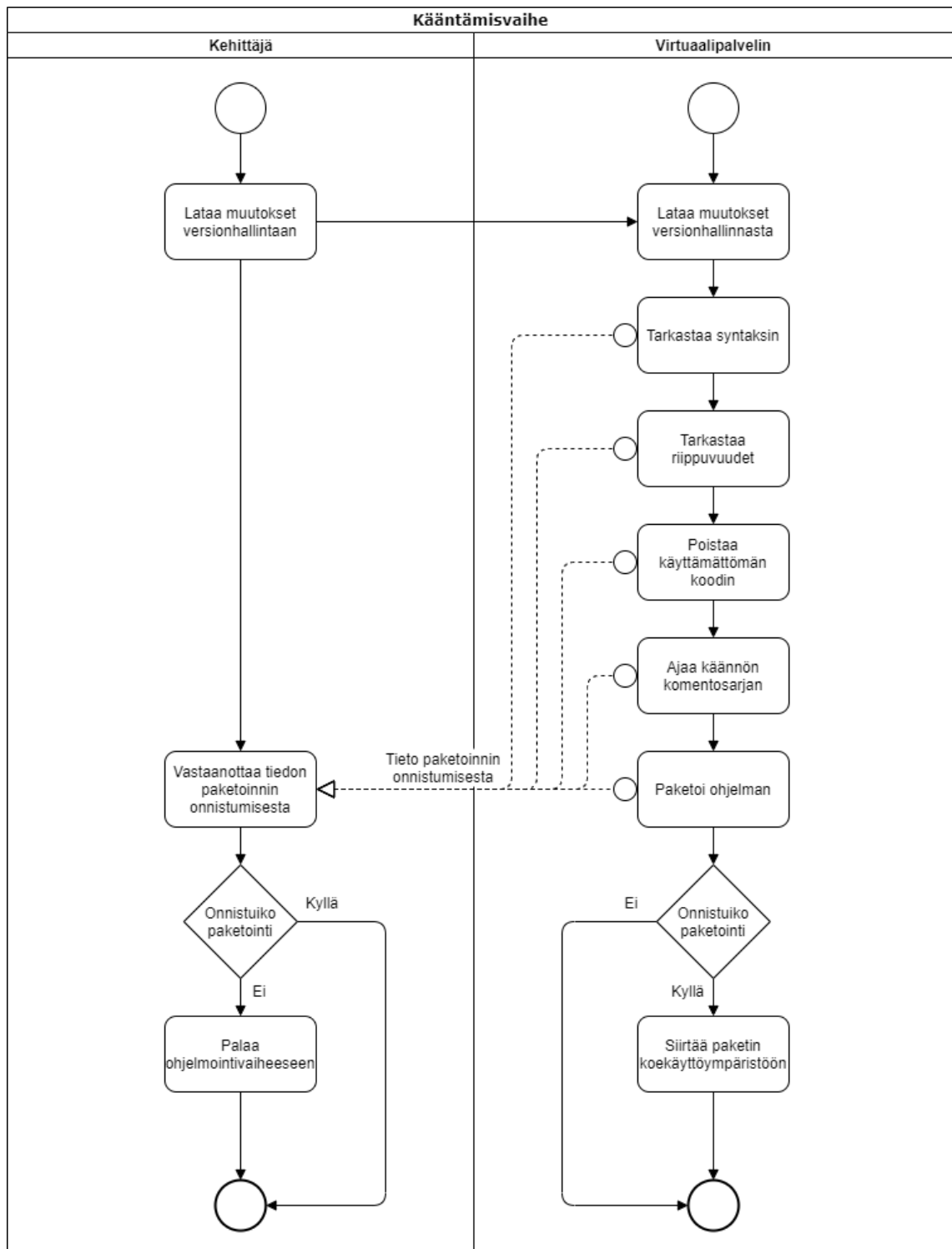
# Suunnitteluvaiheen prosessikaavio



## Ohjelmointivaiheen prosessikaavio

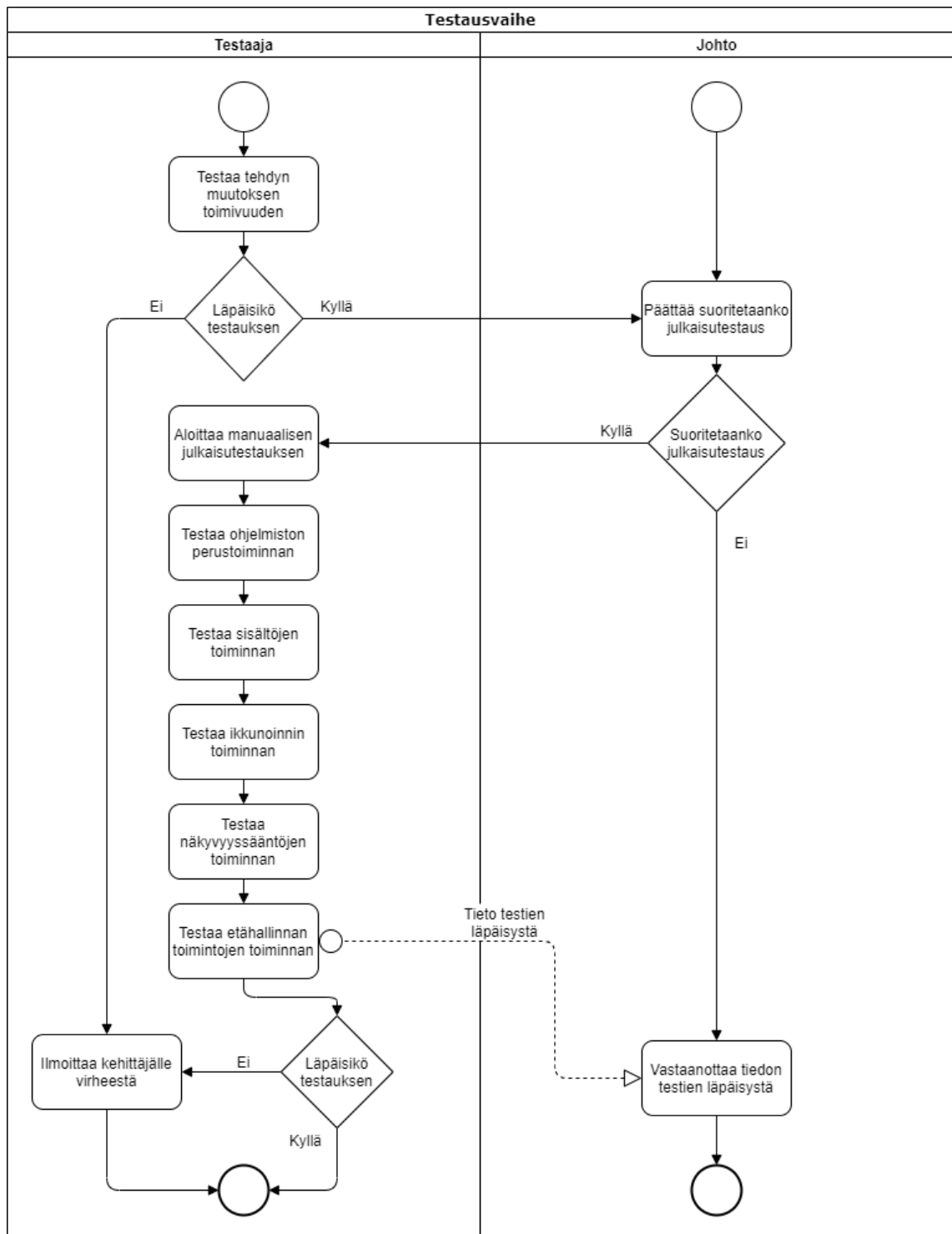


## Kääntämisvaiheen prosessikaavio

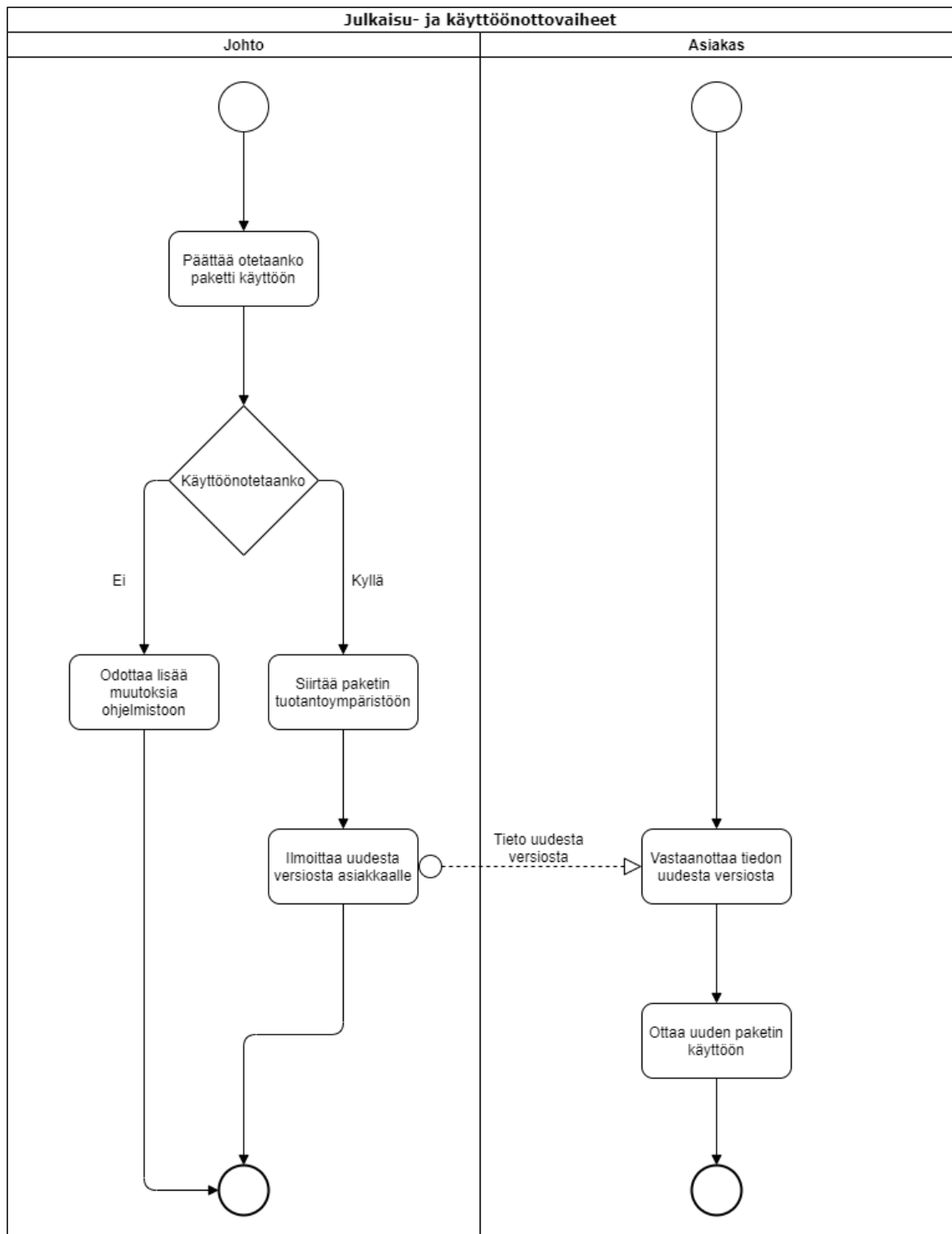




## Testausvaiheen prosessikaavio



## Julkaisu- ja käyttöönottovaiheiden prosessikaavio



## Käyttö- ja valvontavaiheiden prosessikaavio

