Valtteri Karppinen

DEVOPS JA SEN VAIKUTUKSET JÄRJESTELMÄ-EVOLUUTION HALLINTAAN

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2017

TIIVISTELMÄ

Karppinen, Valtteri
DevOps ja sen vaikutukset järjestelmäevoluution hallintaan
Jyväskylä: Jyväskylän yliopisto, 2017, 88 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja(t): Halttunen, Veikko

Tietojärjestelmät ovat integroituneet syvälle nyky-yhteiskuntaan. Ohjelmistot nähdään kriittisinä tukien organisaatioiden sisäistä toimintaa ja kilpailukykyä. Turbulenteista markkinoista johtuen organisaatiolta vaaditaan jatkuvaa muuntautumista, jonka voidaan nähdä heijastuvan järjestelmävaatimusten muutokseen. Näin ollen yhteiskunnan jatkuva muuttuminen sysää painetta yhä enenevissä määrin ohjelmistoja tuottavien organisaatioiden toimintaan. Yhteiskunnassa jatkuvan muutoksen paine kuvaa, että tietojärjestelmät ovat luonteeltaan evolutiivisia. Toisien sanoen järjestelmiä tulee jatkuvasti muuntaa, jotta niiden kyky toimia ympäristössään säilyy. Järjestelmäevoluutio kuvaa piirteitä, joita ohjelmistoja kehittävien tahojen on huomioitava järjestelmien jatkuvan muuntamisen takaamiseksi. Muutoksen rasitteista selvitäkseen ohjelmisto-organisaatiot etsivät jatkuvasti tehokkaampia toimintamalleja. Tätä kuvaa muun muassa ketterien ohjelmistokehitysmallien yleistyminen ja suosio 2000-luvun alkupuolella. Viimeisten vuosien aikana ohjelmistotuotannossa on havaittu uusi trendi, DevOps. DevOps toimintamallin lupauksena on virtaviivaistaa ja nopeuttaa tietojärjestelmien kehittämiseen, julkaisuun ja operointiin liittyviä toimia. Tieteellinen ja teollinen yhteiskunta ovat kuitenkin vielä kaukana ymmärtääkseen DevOps toiminnan vaikutuksia järjestelmäevoluution hallinnan näkökulmasta.

Tämä tutkimus on jaettu kahteen osioon. Ensimmäisessä kirjallisuuskatsaukseen perustuvassa osuudessa tutkitaan järjestelmäevoluutiota ja DevOpsia toisistaan irrallisina ilmiöinä. Osiossa analysoidaan järjestelmäevoluution vaikutuksia ohjelmistotuotantoon sekä valotetaan DevOps toimintamallin tavoitteita ja käytänteitä. Tutkielman toisessa osiossa esitellään laadullinen haastattelututkimus. Haastatellen kuutta suomalaista ohjelmistoalan ammattilaista osiossa selvitetään, kuinka DevOps toiminta vaikuttaa järjestelmäevoluution hallintaan.

Yleisellä tasolla tutkimustulokset osoittavat, että DevOpsin vaikutukset järjestelmäevoluution hallintaan ovat myönteisiä, mutta tulosten tulkinta havainnollistaa myös hankalasti havaittavia haasteita. Tehokkaamman evoluution hallinnan näkökulmasta DevOps helpottaa järjestelmien teknisen evoluution hallintaa. Toisaalta toimintamalli näyttäytyy myös johtamisprosessina, jolla optimoidaan ohjelmistotuotannon tehokkuutta ja järjestelmien läpimenoaikaa.

Asiasanat: devops, järjestelmäevoluutio, järjestelmäkehitys, ohjelmistotuotanto

ABSTRACT

Karppinen, Valtteri
Managing Software Evolution Through DevOps Practices
Jyväskylä: University of Jyväskylä, 2017, 88 p.
Information Systems, Master's Thesis
Supervisor(s): Halttunen, Veikko

IT is deeply embedded in todays' industries, where it is not only considered as an internal capability but also as a competitive factor. Because of turbulent business environments, organizations pursuing operational excellence are faced with constant change. This creates a pressure to change the IT within. Thus, the burden of changing business requirements is forwarded into organizations developing the IT systems. The constant change expresses that software systems tend to be evolutionary in their nature. Software evolution is a theoretical lens which explains the aspects faced by software development organizations when trying to handle the constant evolution. To cope with the encumbrance of software evolution, software organizations are continuously searching for more effective manners to operate. This was, for example, realized in the early 2000s as the emergence and popularity of agile development methodologies. A more novel trend in systems development is an operational model called DevOps. DevOps has a promise of streamlining and speeding up the development, release and operation of software systems. However, science and the software industry are far behind in understanding the influence that DevOps has on software evolution.

The aim of this research is to investigate the impacts of DevOps in managing software evolution. The paper includes two sections. The first section is a literature review of software evolution and DevOps. The section reveals aspects and importance of software evolution in software engineering context. Additionally, the review recognises the purposes and practices of DevOps. The second section of the paper introduces a qualitative study conducted by interviewing software professionals in three Finnish software producing companies. The qualitative section explains how DevOps and related practices influence the evolution of software systems.

The research results show that DevOps, indeed, has several significant impacts in managing software evolution. Generally, these impacts are of positive fashion. However, the findings also point out some unobvious challenges that are generated into the software engineering process. DevOps is seen to aid the management of software evolution in two facets. It does not only help the technical evolution of systems, but also serves as a managerial process to optimize the throughput of the software engineering pipeline.

Keywords: devops, software engineering, software evolution, software development

KUVIOT

TAULUKOT

SISÄLLYSLUETTELO

1 JOHDANTO

Liiketoimintaympäristöjen muutoksen ja teknologisen kehityksen tahti on viime vuosikymmeninä kiihtynyt yhä enenevissä määrin. Koska tietojärjestelmät ovat suuressa roolissa nykypäivän toimintaympäristöä, on selvää, että ympäristömuutosten paine aiheuttaa problematiikkaa suoraan järjestelmiä kehittävien organisaatioiden toimintaan. Jatkuvan muutoksen vuoksi ohjelmistoja kehittävien tahojen täytyy yhä nopeammin vastata muuttuvaan maailmaan ja muuttuviin järjestelmävaatimuksiin.

Järjestelmäevoluutio on eräs tapa kuvata muutosten aiheuttamaa painetta. Järjestelmien tapauksessa evoluutiolla tarkoitetaan ilmiötä, jossa järjestelmän ympäristötekijöiden vaikutus jollain tavalla muuttaa järjestelmän vaatimuksia ja täten aiheuttaa paineen sen toiminnan muutokselle (Lehman & Ramil, 2003). Järjestelmäevoluution näkökulmasta siis ajatellaan, että reaalimaailman muutoksen seurauksena järjestelmiä täytyy muuntaa, jotta niiden kyky suoriutua toimintaympäristössään säilyy. Meir Lehman (1974-1996) sekä monet muu tutkijat ovat havainneet, että järjestelmien evoluutiolla on tapana noudattaa tiettyjä lainalaisuuksia. Lainalaisuuksien kautta voidaan havaita millaisia ongelmia ja haasteita järjestelmiä kehittävät tahot kohtaavat järjestelmien elinkaaren aikana.

Jotta jatkuvaan muutokseen voidaan vastata, täytyy organisaatioiden muuntaa toimintaansa. Ohjelmistomaailmassa tämä on näkynyt toisaalta tehokkaampien teknologioiden käytön yleistymisenä ja toisaalta taas toimintatapojen tai -mallien muokkaamisena. Esimerkiksi ketterien kehitysmenetelmien yleistyminen ohjelmistomaailmassa 2000-luvun alkupuolella helpotti ohjelmistotuotannon kykyä vastata muutokseen. Ketterät kehitysmenetelmät syntyivät vastapainona hitaita ja muutosten suhteen joustamattomia ohjelmistoprosessimalleja korvaamaan. Pääsääntöisesti ketterät mallit koostuvat joukosta hyväksi havaittuja käytänteitä, joiden mukaan ohjelmistoja rakennetaan pienissä iteratiivisissa ja inkrementaalisissa sykleissä toimittaen toimintakykyisiä ohjelmistojulkaisuja säännöllisin väliajoin. Säännöllisen toimittamisen vuoksi ketterällä toimintatavalla pystytään mahdollisimman aikaisessa vaiheessa samaan pa-

lautetta järjestelmien toiminnasta ja täten vastaamaan nopeammin järjestelmävaatimusten muutokseen. (Beck ym. 2001.)

Ketterissä kehitysmenetelmissä perinteisesti otetaan huomioon ainoastaan kehityksen näkökulma. Tällöin järjestelmän julkaisuun ja operointiin liittyvät aktiviteetit jäävät toisarvoisiksi. Hütterman (2012, 36) näkeekin nykypäivän tavanomaisen ohjelmistotuotannon aiheuttavan jakautuneisuutta ohjelmistotuotantoprosessiin. Hän puhuu kehitysorganisaatioiden siiloutumisesta (engl. *organisational silos*), millä tarkoitetaan ohjelmistotuotantotoimintojen jakautumista erillisiksi funktionaalisiksi yksiköiksi, kuten kehitys-, laadunvarmistus- sekä operointitoiminnoiksi (Hütterman, 2012, 34).

Toimintojen jakautuneisuuden ongelmat konkretisoituvat, kun tiimien tavoitteet ovat erilaiset. Kehityksessä tähdätään toimivan ohjelmiston rakentamiseen, kun taas operoinnissa pyritään takamaan ohjelmistolle vakaa ja toimiva alusta. Ongelma nähdään tiimien arvomaailman erilaisuutena: kehitystoiminto haluaa tehdä muutoksia ohjelmistoon, kun taas operointi pelkää ohjelmistojulkaisun aiheuttamaa muutosta, sillä se voi aiheuttaa epävakautta ohjelmistoympäristöön (Humble & Molesky, 2011; Hütterman, 2012). Toisin sanoen ohjelmistojulkaisun yhteydessä kehitystoiminto siirtää vastuun ohjelmistosta operointitoiminnolle. Tällöin järjestelmän julkaisu- ja operointitoimet toimivat ohjelmistotuotantoprosessin pullonkaulana hidastaen muutosten toteuttamista.

DevOpsia pidetään yhtenä nykypäivän ohjelmistotuotantomaailman suosituimmista trendeistä. Yleisesti DevOpsista puhuttaessa tarkoitetaan toimintamallia, joka nimeensä viitaten pyrkii vähentämään kehityksen- (engl. *development*) ja operoinnin (engl. *operations*) välistä kitkaa. Toimintamallin tarkoituksena on lisätä ohjelmistotuotannon tehokkuutta yhdistämällä kehityksen ja operoinnin välistä yhteistyötä. Konkreettisesti toimintojen yhteen toimivuutta parannetaan tukeutumalla uudenlaiseen tiimirakennelmaan, tehokkaampiin automaatio- ja monitorointiteknologioihin sekä erilaiseen tapaan mitata tuotantotehokkuutta. Näiden DevOps käytänteiden hyödyllisyyden on nähty sujuvoittavan ohjelmistotuotantoprosessia ja parantavan ohjelmistojulkaisujen laatua. (Humble & Molesky, 2011; Hütterman, 2012; Virmani, 2015.). Tehokkaamman ohjelmistoprosessin voidaan nähdä parantavan ohjelmistotuotanto-organisaatioiden kykyä vastata jatkuvaan muutokseen.

IBM:n julkaiseman raportin (2013) mukaan noin 70% prosenttia ohjelmistoyrityksistä, jotka toimivat yhdistäen kehitys-, ja operointitoimintojaan pystyivät kilpailijoitaan paremman tuloksen tekemiseen. Myös Elliotin (2014) tutkimuksessa havaitaan samansuuntainen trendi. IBM:n raportti (2013) paljastaa myös, että jopa 25% ohjelmistoyrityksistä kokee heidän kehitys- ja operointitoimintonsa tuloksen kannalta tehottomiksi. Raporttien tulokset paljastavat, että DevOps toimintamallin hyödyntämisellä on suoria vaikutuksia ohjelmistoliiketoimintaan. DevOpsia tutkinutta tieteellistä tutkimusta on vielä kuitenkin varsin vähän. Tähän mennessä tutkimuksissa ei ole otettu huomioon, kuinka DevOps toiminnan hyödyntäminen parantaa ohjelmistotuotannon kykyä vastata järjestelmien toimintaympäristöjen muutokseen. Toisin sanoen järjestelmäevoluution näkökulmaa ja DevOps toimintaa yhdistävää tieteellistä tutkimusta ei

ole lainkaan. Tämän tutkielman tarkoituksena onkin selvittää, kuinka DevOps toimintamalli vaikuttaa ohjelmistotuotannon kykyyn mukautua jatkuvasti muuttuviin vaatimuksiin. DevOpsin ja järjestelmäevoluutionäkökulman yhdistämiseksi tutkimusongelma voidaan esittää tutkimuskysymyksenä seuraavasti:

- Miten DevOps toimintamallissa huomioidaan järjestelmäevoluutio ja sen erityispiirteet?

Tutkimusongelmaan vastattaessa tutkielma on jaettu käsiteellis-teoreettiseen osuuteen sekä empiiriseen osuuteen.

Tutkielman käsitteellis-teoreettinen osio havainnollistaa järjestelmäevoluutiota ja DevOps toimintamallia erillisinä ilmiöinä. Osion tarkoituksena on rakentaa aihepiireistä yhteinen teoreettinen pohja. Luvussa 2 paneudutaan tietojärjestelmien evoluutioon ohjelmistotuotannon näkökulmasta. Luvussa valotetaan yleismaailmallisesti ohjelmistotuotantoa sekä paneudutaan syvemmin tarkastelemaan järjestelmien kehittymisen dynamiikkaa, järjestelmäevoluution lainalaisuuksia sekä evoluution seurauksien aiheuttamia haasteita. Tutkielman kolmas luku puolestaan tarkastelee DevOpsia sekä sen merkitystä ja käytänteitä ohjelmistotuotannossa. Käsitteellis-teoreettisen osion viimeisessä luvussa (luku 4) pohditaan kirjallisuuteen perustuen DevOps toimintamallin ja järjestelmäevoluution suhdetta.

Teoriapohjan muodostamisessa on hyödynnetty kirjallisuuskatsausta menetelmänä. Katsauksen pohjana on käytetty tietojenkäsittelytieteiden ja ohjelmistotuotannon tieteellisiä julkaisuja sekä muuta alan kirjallisuutta. Kirjallisuutta on etsitty muun muassa seuraavista tietojenkäsittelytieteiden kannalta merkittävistä julkaisuista: European Journal of Information Systems (EJIS), Information Systems Journal (ISJ), Information Systems Research (ISR), MIS Quarterly (MISQ). Lisäksi ohjelmistotuotannon näkökulmasta kirjallisuutta on etsitty esimerkiksi seuraavista julkaisuista: IEEE Transactions on Software engineering (TSE), Communications of the ACM (CACM), Empirical Software Engineering (ESEJ), ja IEEE Software. Myös Googlen Scholar -palvelua on käytetty kirjallisuuden löytämisessä. DevOpsin tapauksessa tieteellinen lähdemateriaali on paikoin ollut mainituissa julkaisuissa vähäistä. Tästä johtuen DevOps aiheisia lähteitä on etsitty osittain myös muista julkaisuista. Lähteiden löytämisessä on ensisijaisesti hyödynnetty hakutermejä ”devops” ja ”software evolution”. Järjestelmäevoluutiokirjallisuuden hakemisessa on käytetty myös muita hakutermiyhdistelmiä, kuten ”agile software evolution”. Lähteitä valittaessa on lähdemateriaalin laatu perustettu muun muassa lähteeseen tehtyjen viittausten lukumäärään, lähteen julkaisun tunnettavuuteen sekä tekijöiden tunnettavuuteen. Järjestelmäevoluutiokirjallisuuden lähteet on valittu siten, että kyseinen lähde sisältää viitteitä Meir Lehmanin järjestelmäevoluutiokäsityksiin. DevOps-toimintamallin tuoreudesta johtuen sisältää osa aihepiirin lähdemateriaalista myös hieman vähemmän viitattuja artikkeleita.

Tutkielman empiirinen osuus luvuissa 5—7 tukeutuu vahvasti käsitteellis-teoreettisessa osuudessa rakennettuun teoreettiseen pohjaan. Luvussa 5 kuvataan, kuinka laadullista teemahaastattelua tutkimusmenetelmänä käyttäen on

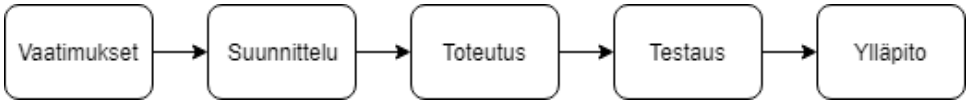
tutkittu DevOps toiminnan ja järjestelmäevoluution välistä suhdetta. Luvussa 6 esitellään haastattelututkimuksen tuloksia ja luvussa 7 puolestaan pohditaan saatuja tuloksia sekä niiden merkitystä tutkimuskysymyksen asettelun näkökulmasta. Tutkielman kahdeksannessa ja viimeisessä luvussa esitetään yhteenveto tutkimuksesta.

2 JÄRJESTELMÄEVOLUUTIO

Tässä luvussa paneudutaan tietojärjestelmien kehittymiseen eli järjestelmien evoluutioon. Luku jaettu kolmeen osioon, joista ensimmäisessä perehdytään yleisesti ohjelmistotuotantoon ja sen kehityksen, ylläpidon ja operoinnin toimintoihin. Luvun toisessa osiossa käsitellään tarkemmin järjestelmäevoluutiota sekä sen erityispiirteitä. Viimeisessä alaluvussa jäsennetään järjestelmäevoluutiota tutkielman teoreettisena viitekehyksenä.

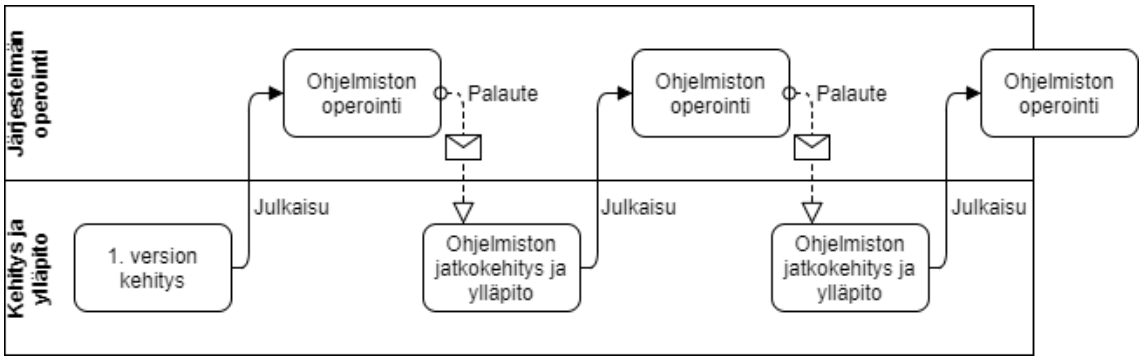
2.1 Ohjelmistotuotanto

IEEE:n (Institute of Electrical and Electronics Engineers) terminologia standardissa ohjelmistotuotanto (engl. *software engineering*) määritellään systemaattiseksi toiminnaksi, jossa ohjelmistoa kehitetään, operoidaan ja ylläpidetään (IEEE, 1990). Ohjelmistotuotannon prosessimalleilla kuvataan yksinkertaistettuna näitä ohjelmistotuotantoprosessiin liittyviä toimia (Sommerville, 2016, 45). Ensimmäinen esitys prosessimaisesta järjestelmäkehitystoiminnasta syntyi jo vuonna 1956 Herbert Beningtonin ajatuksista. Myöhemmin samankaltainen ohjelmistotuotannon prosessimalli yleistyi laajan yleisön tietouteen Winston Roycen vesiputousmalliksi (engl. *waterfall model*) nimettynä vuonna 1970. Beningtonin ja Roycen mallit jakavat kuvion 1 mukaisesti järjestelmäkehityksen vaiheet pääpiirteittäin viiteen toistaan seuraavaan vaiheeseen: vaatimusmäärittelyyn, suunnitteluun, toteutukseen, testaukseen ja ylläpitoon (Bengington, 1983; Royce, 1970). Vaiheittaisissa kehitysmalleissa kutakin vaihetta toistetaan eli iteroidaan niin pitkään, kunnes vaihe on suoritettu. Toisin sanoen seuraaviin vaiheisiin ei siirrytä ennen kuin edeltävät vaiheet on suoritettu loppuun. Näin ollen esimerkiksi ohjelmiston yksityiskohtaisempaa suunnittelua ei aloiteta ennen kuin tilaajan asettamat vaatimukset on lukittu.



KUVIO 1 Vaiheittainen ohjelmistoprosessi

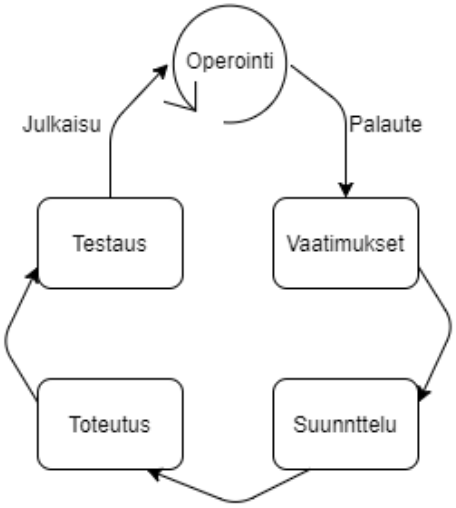
Vaiheittainen kehitysmalli ei kuvaa järjestelmäkehitystä todellisuudessa ja mallia onkin usein kritisoitu joustamattomuudestaan sekä jähmeästä muutosreagointikyvystään (Sommerville, 2016, 48-49). Syyt kritiikkiin johtuvat pääosin siitä, että vaiheittainen malli on hidas. Tämä johtuu siitä, että prosessissa ei siirrytä eteenpäin ennen kuin vaihe valmis ja tällöin ohjelmiston julkaisuun kuluu usein vahingollisen pitkä aika. Nopeasti muuttuvassa liiketoimintaympäristössä ohjelmistoja tuottavien organisaatioiden on kehitystoiminnassaan kyettävä reagoimaan nopeasti. Tämä tarkoittaa sitä, että ohjelmistotuotantoprosessi on luonteeltaan palautteeseen perustuva järjestelmä. (Lehman, 1996, 6). Ohjelmistosta saadaan palautetta vasta siinä vaiheessa, kun järjestelmä on otettu käyttöön. Kuvion 2 mukaisesti ohjelmistotuotantoprosessi alkaa kehitystiimin kehittäessä ensimmäisen ohjelmistoversion. Kun ensimmäinen ohjelmistoversio on valmiina, se julkaistaan tuotantoympäristöön käyttäjien käyttöön. Tällöin alkaa niin kutsuttu järjestelmän operointi, missä usein kehitystiimistä erillinen operointitoiminto hallitsee ja valvoo järjestelmää, sen ympäristöä ja sen käyttöä. Tyypillisesti vasta operointivaiheessa huomataan järjestelmän puutteet tai muut virheet (Sommerville, 2016). Tällöin järjestelmän operoinnin aikana saatava palaute ohjataan samaiselle kehitystiimille, joka palautteeseen perustuen ylläpitää tai jatkokehittää järjestelmää.



KUVIO 2 Järjestelmän kehitys, ylläpito ja operointi

Niin sanottujen ketterien kehitysmenetelmien (engl. *agile software development*) yleistyminen vuosituhannen alussa kertoo siitä, kuinka ohjelmistotuotantomaailmassa tarvittiin nopeampia keinoja muutosreagointiin. Ketterät toimintamallit tarjoavat ohjelmistokehitykseen keinoja poistamaan vaiheittaisen kehityksen joustamattomuutta muutosteen suhteen. Kantavana ajatuksena ketteräs-sä kehittämisessä on, että ohjelmistoja rakennetaan lyhyissä ja entistä nopeammissa sykleissä toimittaen jatkuvasti eli inkrementaalisesti uusia ohjelmistover-

sioita julkaistavaksi. Jatkuvan ja nopeamman toimittamisen etuna on se, että ohjelmistokehityksessä kyetään nopeammin ja varhaisemmassa vaiheessa saamaan palautetta järjestelmän toiminnasta. (Beck ym. 2001.). Ketterän kehittämisen yleistyttyä ohjelmistokehityksen prosessimalli nähdäänkin syklisenä kuvion 3 mukaisesti.

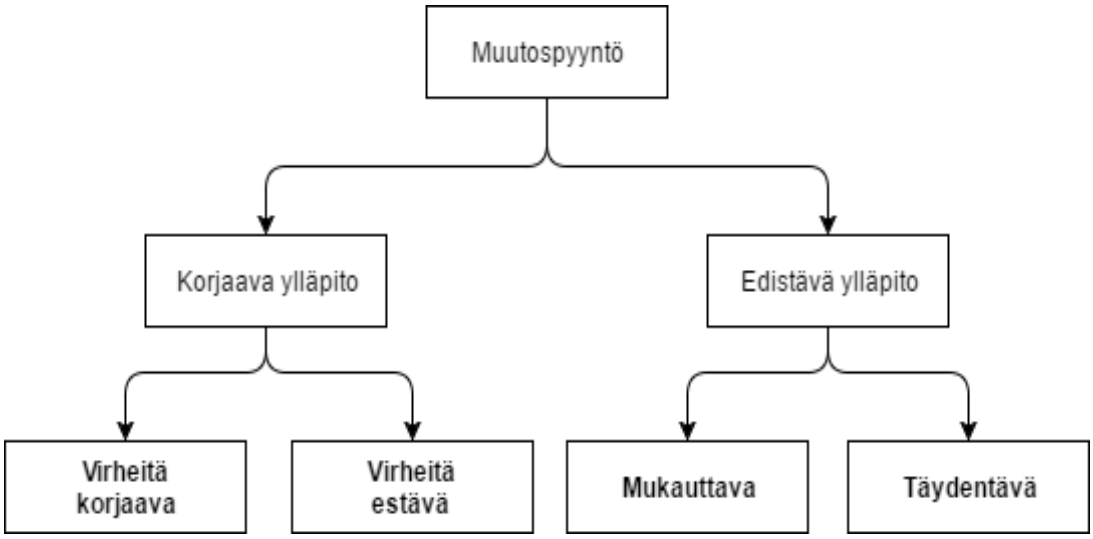


KUVIO 3 Ketterä kehitysmalli

Vanhojen vaiheittaisten ohjelmistokehitysmallien mukaisesti järjestelmien ylläpito ajateltiin varsinaisen kehittämisen jälkeisenä toimintana, kun järjestelmä oli otettu käyttöön (Royce, 1987). Kehitystä ja ylläpitoa on kuitenkin hankala määrittää erillisiksi toiminnoiksi, sillä ohjelmistoa jatkokehitetään, tai toisin sanoen ylläpidetään, ensimmäisen ohjelmistojulkaisun jälkeen (Bennett & Rajlich, 2000). Tämä huomataan etenkin ketterien ohjelmistokehitysmenetelmien yleistymisen myötä, kun ohjelmiston ylläpito on alkanut näyttäytyä osana kehitystoimintaa. Ketterissä menetelmissä ohjelmistojulkaisuja tehdään jatkuvasti (Beck ym. 2001), jolloin palautetta järjestelmän toiminnasta saadaan mahdollisimman nopeasti ja varhaisessa vaiheessa. Vasta palautteen perusteella järjestelmää jatkokehitetään suuntaan tai toiseen.

Järjestelmän ylläpitotoiminnan ajatellaan usein virheellisesti sisältävän ainoastaan järjestelmässä ilmenneiden virheiden korjaamista. ISO/IEC 14765–standardissa (ISO/IEC, 2006) ylläpito kuitenkin määritellään varsinaisen käyttöönoton jälkeen tapahtuvaksi toiminnaksi, missä se jaetaan kahteen näkökulmaan: korjaavaan (*engl. corrective*) ja edistävään (*engl. enhancing*) ylläpitoon. Kuvion 4 mukaisesti ylläpidon laji määräytyy tietyn muutos- tai palautepyynnön perusteella joko ohjelmistoa korjaavaksi tai sitä edistäväksi ylläpidoksi. Korjaava ylläpito jaetaan standardissa vielä edelleen kahtia virheitä korjaaviin (*engl. corrective maintenance*) ja virheitä estäviin (*engl. preventive maintenance*) toimintoihin. Virheitä korjaavan ylläpidon tarkoituksena on muutospyyntöjen seurauksena havaittujen virheiden korjaaminen järjestelmässä. Virheitä estävä ylläpito puolestaan pyrkii nimensä mukaisesti estämään tulevaisuudessa tapah-

tuvia ohjelmistovirheitä ennen kuin ne havaitaan järjestelmän käynnissä olon aikana.



KUVIO 4 Järjestelmäylläpidon luokittelu (ISO/IEC, 2006)

Edistävän ylläpidon tarkoituksena on lisätä järjestelmän toiminnallisuutta tai parantaa järjestelmän toimintakykyä tietyssä ympäristössä. Tällöin mukauttavaan ylläpitoon (engl. adaptive maintenance) perustuen pyritään muuttamaan jo toteutettun järjestelmän toimintaa vastaamaan paremmin asetettuihin ympäristövaatimukseen. Täydentävässä ylläpidossa (engl. perfective maintenance) tarkoituksena puolestaan on täydentää järjestelmän toimintaa uusilla ominaisuuksilla, parantaa järjestelmän ylläpidettävyyttä tai selkeyttää järjestelmän dokumentaatiota. (ISO/IEC, 2006, 3-4). ISO/IEC 14765–standardin mukaisesti jaoteltu ylläpito on käynyt läpi useita uudistuksia. Standardin mukainen jaottelu pohjaa jo 1980 julkaistuun Lientzin ja Swansonin teokseen ”Software maintenance management”. Teoksen mukaan ylläpito on yhtä lailla jaoteltu neljään esitettyyn ylläpidon alakategoriaan (Bennett & Rajlich, 2000, 76).

Järjestelmien ylläpidon ja jatkokehityksen merkitystä voidaan havainnollistaa tarkastelemalla ohjelmistoyritysten asettamaa panosta kyseiseen toimintaan. Vuonna 1980 Lientz ja Swanson havaitsivat, että huomattavan suuri osa, noin 50% järjestelmäkehityksen kustannuksista toteutuu vasta ylläpitovaiheessa (Bennett & Rajlich, 2000, 76). Kolmetoista vuotta myöhemmin vuonna 1993 tehtyyn katsaukseen perustuen useiden yritysten ylläpitokustannuksen olivat kasvaneet jo keskimäärin 75%:in (Banker & Slaughter, 1997, 1709). Vuonna 2000 arvioidut ylläpidon kustannukset saattoivat tapauskohtaisesti harpata jopa 90%:in (Erlikh, 2000). Glassin (2001) mukaan kuitenkin ylläpidosta aiheutuvat kustannukset asettuvat keskimäärin noin 60%:in järjestelmäkehityksen kokonaiskustannuksista. Vaikka esitetyt luvut tulevat hieman eri tyyppisten tutkimusten tuloksista ja tutkimuskohteina on ollut eri yrityksiä, huomataan niistä selkeä trendi.

Kustannusten kertyminen ylläpitovaiheeseen kertoo siitä, että suurin osa ohjelmistokehitystyöstä tapahtuu vasta tässä vaiheessa. On huomattava, että järjestelmien ylläpito on vasta ensimmäisen varsinaisen ohjelmistojulkaisun jälkeen tapahtuvaa toimintaa. Tällöin työmäärän kertyminen kertoo järjestelmäympäristön muuntumisesta. Toisin sanoen vasta varsinaisen julkaisun jälkeen ymmärretään, kuinka järjestelmä toimii ja kuinka sitä tulisi muuntaa. Tästä syystä järjestelmän tilaajien ja käyttäjien tyytyväisyyden takaamiseksi jatkokehitys- ja ylläpitotoimet tarvitsevat palautetta julkaistun järjestelmän käytöstä. Mikäli palautteensaanti on hidasta, ohjelmistotuotannossa ei kyetä vastaamaan muutospyyntöihin nopeasti. (Bennett & Rajlich, 2000, 76.)

2.2 Järjestelmäevoluutio ja evoluution lainalaisuudet

Ohjelmistotuotannon piirissä järjestelmien evoluutiolla tarkoitetaan järjestelmän kehittymistä sen elinkaaren aikana (Kemerer & Slaughter, 1999, 493). Järjestelmäevoluutiota tutkineilla on hieman erilaisia käsityksiä evoluutiosta. Toisaalla evoluution liitetään toimenpiteet, jotka sisältävät järjestelmäkehitystoimia ennen julkaisuja, toisaalla taas evoluution ajatellaan alkavan vasta ensimmäisen julkaisun jälkeen, ylläpito- ja jatkokehitysvaiheessa (Kemerer & Slaughter, 1999, 493).

Perry näkee (1994) järjestelmien evoluutioon liittyvän kolme keskeistä ulottuvuutta: järjestelmän ympäristö (*engl. domain*), järjestelmän ja sen ympäristön ymmärtäminen (*engl. experience*) ja järjestelmän kehitysprosessi (*engl. process*). Kyseiset ulottuvuudet kuvaavat kontekstia, jossa järjestelmän nähdään kehittyvän (Perry, 1994, 302). *Järjestelmän ympäristö* kuvaa reaalimaailmaa ja se on keskeisessä roolissa evoluution kannalta, sillä mikäli toimintaympäristössä tapahtuu muutoksia, on järjestelmän niin ikään muututtava sen ympäristön vaatimalla tavalla. Koska järjestelmien toimintaympäristö muuttuu tavoilla, joita ei kyetä kehityksessä ennalta ennustamaan, on myös *järjestelmän ja sen toimintaympäristön ymmärrys* yksi keskeisistä tekijöistä järjestelmän evoluution kannalta (Perry, 1994, 302). Tällöin järjestelmiä kehittävän tahon ymmärryksen ajatellaan lisääntyvän ajan kuluessa ja ymmärryksen lisääntyminen on avainasemassa evoluution näkökulmasta. Ymmärryksen Perry (1994) näkee vaikuttavan myös tapaan eli *prosessiin*, jolla järjestelmäevoluutiota hallitaan. Ymmärryksen lisääntyminen parantaa siis järjestelmän kehittäjien näkemystä siitä millaisilla menetelmillä, tekniikoilla ja työkaluilla järjestelmän evoluutiota tulee hallita (Perry, 1994, 302).

Bennettin (1996) näkemys järjestelmäevoluutiosta on hyvin samankaltainen. Hänen mukaansa järjestelmiä kehittävien organisaatioiden tulisi evoluutiota ajatellen huomioida kolmea tekijää. Bennettin (1996, 675) mukaan on tärkeää, että järjestelmän evoluutio on linjassa *organisaation tavoitteiden* kanssa. Tällöin, organisatoriset tavoitteet sanelevat sen, kuinka pitkälle järjestelmää on mielekästä jatkokehittää ja ylläpitää. Myös Bennettin (1996) näkemyksen mukaan *kehitysprosessin* merkitys on tärkeässä roolissa järjestelmäevoluution näkökul-

masta. Hallitun järjestelmäevoluution kannalta onkin tärkeää, että organisaation prosessit on usealla tasolla huomioitu. Täten, järjestelmän evolutiiviseen kehittymiseen liittyy toimintoja niin kehityksen, ylläpidon kuin operoinnin prosessien puolelta (Bennett, 1996, 675). Kolmas evoluution kannalta tärkeä tekijä on *teknisten ratkaisujen soveltaminen*. Bennettin (1996) tekninen näkökulma ottaa kantaa järjestelmän tekniseen toteutukseen ja prosessitasolla käytettäviin työkaluihin. Teknisestä näkökulmasta järjestelmän evoluutio on sitä hallitumpi mitä tehokkaampia työkaluja sen kehittämiseen ja ylläpitoon hyödynnetään.

Sekä Perry (1994) että Bennett (1996) pohjaavat järjestelmäevoluution käsitteistönsä saksalaisen Meir Lehmanin jo vuonna 1974 alkaneeseen järjestelmäevoluutiotutkimukseen. Havaintojensa pohjalta Lehman (Lehman & Ramil, 2003) määrittelee järjestelmän evoluution jatkuvaksi prosessiksi, jossa ympäristötekijöiden vaikutus jollain tavalla muuttaa järjestelmän vaatimuksia ja täten asettaa paineen myös järjestelmän muutoksille. Järjestelmäevoluutio pohjaa alkunsa reaalimaailman eli järjestelmän toimintaympäristön vaatimuksiin ja näiden vaatimusten alati muuttuvaan luonteeseen. Lehmanin ja Ramilin (2003) mukaan yritysten toimintaympäristön muutoksen nähdään paineistavan yrityksiä muuntautumaan ja tällöin muutospaine kohdistuu lisäksi yritysten käyttämiin tietojärjestelmin. Muutosten aiheuttama paine saa aikaan järjestelmävaatimusten muuttumisen. Tällöin järjestelmän toimintaa on muutettava, jotta sen kyky vastata uudenlaiseen toimintaympäristöönsä paranee tai pysyy vähintäänkin vakiona. Näin ollen järjestelmäkehityksen, -ylläpidon ja -operoinnin prosessit ovat järjestelmien evoluution kannalta kriittisimpiä toimintoja, sillä niillä ohjataan järjestelmän kehittymistä sekä sen laadun muodostumista. (Lehman & Ramil, 2003, 37.)

2.2.1 Evolutiiviset järjestelmät

Vuonna 1980 Lehman kategorisoi tietojärjestelmät kuuluvaksi kolmeen eri tyyppiin: S-, P- ja E-tyyppisiin järjestelmiin. Luokittelun tarkoituksena oli auttaa paremmin ymmärtämään tietojärjestelmien evoluutiota. SPE-luokittelu havainnollistaa järjestelmien kehittymisen eroavaisuutta. Toisin sanoen, tarkkojen spesifikaatioiden mukaisesti toteutetut (S-tyyppi) ja osana reaalimaailmaa olevat (E-tyyppi) järjestelmät kehittyvät eri tavoin. (Lehman, 1980).

Luokittelun S-tyypisillä (*engl. specification type*) järjestelmillä Lehman (1980) tarkoittaa sellaisia järjestelmiä, jotka on rakennettu tarkkojen spesifikaatioiden mukaisesti. S-tyyppinen järjestelmä on täten luotu ratkaisemaan täsmälleen yksittäinen ja tietty ongelma. Mikäli tehdään muutos S-tyyppisen järjestelmän spesifikaatioon, tällöin uuden spesifikaation täyttävä järjestelmä on täysin uusi. P-tyyppiin (*engl. problem type*) kuuluvat järjestelmät ovat monimutkaisempia ja ne on luotu mallintamaan reaalimaailman ilmiön toimintaa tai ongelman ratkaisua. Esimerkiksi shakkipeli voidaan käsittää P-tyyppisenä järjestelmänä. E-tyyppiset (*engl. evolutionary type*) järjestelmät ovat kolmikosta monimutkaisimpia ja ne on luotu toimimaan osana reaalimaailmaa. E-tyypin järjestelmät ovat siis julkaisunsa jälkeen tiiviitä osia ympäristössään ja sen toiminnassa. Reaali-

maailman dynamiikan seurauksena E-tyyppisten järjestelmien ylläpidolta vaaditaan kykyä muuntaa järjestelmiä muuttuvien vaatimusten mukaiseksi. Tätä kautta E-tyypin järjestelmillä on myös tapana muuttua ennen pitkää monimutkaisemmiksi (Lehman, 1980, 1061-1063.). Yleisesti nykyään käytettävät tietojärjestelmät tai digitaaliset palvelut voidaan tulkita kategorisoinnin perusteella kuuluvan E-tyyppisiin.

E-tyyppisten järjestelmien vaiheittaista monimutkaistumista Lehman kuvaa järjestelmäkehityksessä tehtävien olettamusten (*engl. assumptions*) kautta. Sen lisäksi, että reaalimaailman muutos aiheuttaa muutospaineen myös tietojärjestelmiin, on reaalimaailman ilmiöiden mallintamiselle ja järjestelmätoteutuksille olemassa ääretön määrä toteutustapoja. Tästä äärettömästä E-tyypin järjestelmän toteutustapakirjosta johtuen järjestelmäkehityksessä joudutaan tekemään olettamuksia siitä, kuinka jokin järjestelmä toteutetaan. Kehityksessä ei kuitenkaan voida ennustaa etukäteen, kuinka toimintaympäristö muuttuu ja kuinka muutos tulee vaikuttamaan järjestelmään. Muutoksen seurauksena aikaisemmin tehdyt olettamukset saattavat muuttua järjestelmän ylläpidon tai toiminnan kannalta haitallisiksi. (Lehman & Ramil, 2001.)

Jatkuvan kehittymisen seurauksena E-tyypin järjestelmillä on tapana kasvaa kooltaan. Evolutiivisten järjestelmien (E-tyyppi) koon kasvu puolestaan aiheuttaa kasvavaa monimutkaistumista, josta käytetään myös termiä *järjestelmän entropia* (Kemerer & Slaughter, 1999, 494). Fysiikassa termodynamiikan säännöt määrittelevät entropian kasvun aiheutuvan, kun suljettu termodynaaminen systeemi kasvaa ja aiheuttaa täten lisääntyvää epäjärjestystä. Tietojärjestelmien tapauksessa tilanne on vastaava. Järjestelmän muuttuva toimintaympäristö kasvattaa koko systeemin entropiaa eli epäjärjestystä, joka heijastuu varsinaiseen tietojärjestelmään. (Kemerer & Slaughter, 1999.)

Tietojärjestelmien tapauksessa evoluutio ja sen aiheuttama entropia voidaan havaita usealla tavalla ja useasta eri näkökulmasta. Esimerkiksi tiedon muuttuminen ja lisääntyminen tietokannoissa nähdään evoluution aiheuttamana entropiana. Toisaalta tietojärjestelmän toiminnallisuutta saatetaan joutua järjestelmän ympäristön muutoksen takia muuttamaan, jolloin entropia lisääntyy eri tavalla. E-tyypin järjestelmien tapauksessa evoluution aiheuttama entropia lisääntyy useassa eri muodossa ja tällöin monimutkaisimpien järjestelmien evoluutiota tulee hallita järjestelmäkehitysorganisaatioissa usean eri sidosryhmän panoksen kautta. (Cook, ym. 2006.)

2.2.2 Järjestelmäevoluution lainalaisuudet

Järjestelmäevoluutio ja sen aiheuttama entropia ovat ilmiöinä varsin monitahoisia. Lehmanin suurin ilmiötä konkretisoiva tutkimustyö kulminoituikin järjestelmäevoluution säännönmukaisuuksia määritteleviin lainalaisuuksiin (ks. taulukko 2). Järjestelmäevoluution lainalaisuudet käsittelevät jo esiteltyjä E-tyypin järjestelmiä. Ne perustuvat järjestelmäkehitysympäristössä tehtyihin havaintoihin usean kymmenen vuoden ajalta. Yleisesti lainalaisuudet ottavat kantaa muun muassa tietojärjestelmien kehittymisen yleisiin piirteisiin, järjestelmiä

kehittävien organisaatioiden toimintatapoihin ja prosesseihin sekä järjestelmäkehityksen teknologioihin (Lehman & Ramil, 2003, 35). Lehmanin lainalaisuuksia on myös kritisoitu esittämällä, etteivät ne luo tarpeeksi eksplisiittisiä säännönmukaisuuksia, kuten esimerkiksi fysiikan lait luovat (Cook ym., 2006, 100). Lehman on kuitenkin esittänyt, ettei lainalaisuuksia tulekaan käsittää fysiikan lakien kaltaisina absoluuttisina totuuksina. Pikemminkin niiden kehittämisen tarkoituksena on ollut dokumentoida tietoa järjestelmien evoluutioon liittyvästä käyttäytymisestä (Cook, 2006, 100).

Taulukossa 1 on esitettynä Lehmanin E-tyypin järjestelmien evoluutionkahdeksan lainalaisuutta. Ensimmäinen laeista kuvaa E-tyypin järjestelmien taipumusta *jatkuvaan muutokseen*. Tällä tarkoitetaan sitä, että ajan kuluessa järjestelmien toimintaympäristön muutoksesta johtuen myös järjestelmän tulee muuttua, jottei sen käyttö muodostu haitalliseksi tai epätyydyttäväksi loppukäyttäjille. (Lehman & Ramil, 2001, 543-545).

TAULUKKO 1 Järjestelmäevoluution lainalaisuudet (Lehman)

#	**Nimi (engl.)**	**Määritelmä**
I (1974)	Jatkuva muutos (*continuing change*)	E-tyypin järjestelmät ovat alttiita jatkuvalle muutokselle, minkä seurauksena niitä tulee muuntaa jatkuvasti.
II (1974)	Kasvava kompleksisuus (*increasing complexity*)	E-tyypin järjestelmän kehittyessä sen kompleksisuus kasvaa, ellei sitä pyritä tarkoituksenmukaisesti vähentämään.
III (1974)	Itsesäätely (*self regulation*)	E-tyypin järjestelmän evoluutioprosessi on itseään säätelevä.
IV (1978)	Organisatorisen vakauden säilyttäminen (*conservation of organization stability*)	Kehitysorganisaation panostamalla työmäärällä on tapana pysyä vakiona E-tyypin järjestelmän elinkaaren ajan.
V (1978)	Tuttuuden säilyttäminen (*conservation of familiarity*)	E-tyypin järjestelmiin tehtävien muutosten tahti vähenee pitkällä aikavälillä.
VI (1991)	Jatkuva kasvu (*continuing growth*)	E-tyypin järjestelmien toiminnallisuutta tulee lisätä, jotta käyttäjätyytyväisyys taataan koko järjestelmän elinkaaren ajan.
VII (1996)	Heikentyvä laatu (*declining quality*)	Mikäli E-tyypin järjestelmän toimintaympäristön muutoksia ei oteta huomioon, järjestelmän laatu heikkenee ajan kuluessa.
VIII (1996)	Palautejärjestelmä (*feedback system*)	E-tyypin järjestelmien evoluutioprosessit sisältävät jatkuvia ja eri tasoisia palauteketjuja usean sidosryhmän välillä.

Kasvavaa kompleksisuutta kuvaava toinen lainalaisuus tarkoittaa, että E-tyypin järjestelmän jatkuvan kasvun seurauksena myös sen kompleksisuudella on taipumus kasvaa. Toisin sanoen mitä suuremmaksi järjestelmä kasvaa, sitä hankalammaksi sen ylläpito ja jatkokehittäminen tulevat. Lehman ja Ramil (2001, 545-547) esittävät, että kompleksisuuden kasvun seurauksena on kehityksessä ja ylläpidossa niin prosessi- kuin teknologiatasoilla kyettävä hallitsemaan muutosta tavalla tai toisella.

Kolmas lainalaisuuksista kuvaa evoluutio- eli kehitysprosessin *itsesäätelevää* piirrettä. Prosessin itsesäätelevyydellä tarkoitetaan, että järjestelmäkehityksessä ei noudateta orjallisesti tiettyä kaavaa (esim. kehitysprosessimallia). Ennemminkin prosessi muuttuu ja sitä joudutaan muuttamaan suhteessa evoluution vaikutuksiin. Tällöin prosessia mukautetaan hyviksi havaittujen käytänteiden mukaiseksi (Lehman & Ramil, 2001, 547-549).

Organisatorisen vakauden säilyttäminen, neljäs lainalaisuus, kertoo organisaation toiminnan suhteuttamisesta järjestelmän koon ja kompleksisuuden kasvuun. Lehmanin havaintojen (Lehman & Ramil, 2001, 549-550) mukaan järjestelmäkehityksen ja ylläpidon työtahdilla (engl. work rate) on taipumus pysyä vakaana suhteessa järjestelmän kokoon. Tämä tarkoittaa sitä, että organisaation on säilytettävä työmäärä joko lisäämällä tai optimoimalla työhön käytettäviä resursseja (Lehman & Ramil, 2001, 549-550).

Viides lainalaisuus, *tuttuuden säilyttäminen*, puolestaan tarkoittaa, että järjestelmän evolutiivisen kehittymisen aikana järjestelmän kasvu tahti hidastuu, etenkin pitkällä aikavälillä (Lehman & Ramil, 2001, 550-551). Tämä johtuu siitä, että järjestelmän koon kasvaessa uusien muutosten tekeminen vaatii erilaisia lähestymistapoja. Suuremmassa järjestelmässä ei voida jatkuvasti lisätä toiminnallisuutta heikentämättä laatua, kuten toinen lainalaisuus, kompleksisuuden kasvu esittää. Koska järjestelmä kasvaa, niin sitä kehittävän ja ylläpitävän henkilöstön tietämys vähenee suhteessa järjestelmän kokoon. Tällöin kehityksen ja ylläpidon tuttuus järjestelmäkokonaisuuteen vähenee ajan kuluessa. (Lehman & Ramil, 2001, 550-551.)

Ensimmäinen lainalaisuuksista esittää, että reaalimaailman dynamiikan seurauksena myös siinä toimiva järjestelmä vaatii muutosta. Kuudes laki, *jatkuva kasvu*, taas esittää, että järjestelmän loppukäyttäjien huomioimiseksi E-tyypin järjestelmän toiminnallisuuden täytyy kasvaa jatkuvasti, jotta kyseinen sidosryhmä pysyy tyytyväisenä. Mikäli järjestelmää ei välillä muunneta siten, että uudet toiminnalliset ominaisuudet lisääntyvät, järjestelmän käyttäjät tai järjestelmän käytöstä muuten hyötyvät kokevat sen ennen pitkää epätyydyttäväksi. Lehmanin ja Ramilin (2001, 551-552) mukaan lainalaisuus pohjaa siihen, että ennen pitkää järjestelmästä puuttuvat ominaisuudet koituvat järjestelmän käytön pullonkauloiksi. Tällöin loppukäyttäjät haluavat jossain tulevaisuuden ajanhetkessä vielä järjestelmästä puuttuvat ominaisuudet käyttöönsä, jotta ne tyydyttävät heidän toiminnalliset tarpeensa. (Lehman & Ramil, 2001, 551-552.)

Lehmanin seitsemännen lainalaisuuden (*heikkenevä laatu*) mukaan pitkällä aikavälillä E-tyypin järjestelmän laadulla on tapana heikentyä. Lehmanin ja Ramilin (2001, 552-554) mukaan järjestelmän laatu heikkenee, mikäli muuttu-

van toimintaympäristön vaatimuksia ei seurata ja muutoksia toteuteta järjestelmään. Tällöin lain voidaan nähdä olevan suorassa yhteydessä ensimmäiseen lakiin jatkuvasta muutoksesta. Järjestelmän laatu voi heikentyä monessa ulottuvuudessa. Toisaalta toimintaympäristön vaatimusten mukaisten toiminnallisten ominaisuuksien puute heikentää laatua sekä loppukäyttäjien tyytyväisyyttä. Toisaalta taas jatkuvasta kasvusta johtuen järjestelmän toimintakyky saattaa heikentyä. Lisäksi voi olla mahdollista, että järjestelmäkoon kasvaessa muutostenhallinta alkaa muuttua hankalammaksi ja virheiden määrä lisääntyy. (Lehman & Ramil, 2001, 552-554.)

Kahdeksannen lainalaisuuden (*palautejärjestelmä*) mukaan ohjelmistotuotantoprosessi on luonteeltaan palautteeseen perustuva. Järjestelmän ominaisuuksia kyetään toteuttamaan, mutta ilman tehokasta palautteensaantia, toteutuksen seurauksista ei tiedetä eikä niihin voida vastata. (Lehman & Ramil, 2001, 554-556).

Järjestelmäevoluution lainalaisuudet hahmottavat evoluutioprosessinhallintaan liittyviä epävarmuustekijöitä. Lehman ja Ramil (2001, 559) esittävät, että vaikka järjestelmäkehitystä pidetään luonteeltaan teknisenä prosessina, lainalaisuudet ottavat kantaa myös prosessin johtamisen ja hallinnan näkökulmaan. Lehmanin ja Ramilin (2001, 559) mukaan lainalaisuuksien huomioiminen päätöksenteon näkökulmasta auttaa pitkällä aikavälillä kehitysorganisaatioita järjestelmän elinkaaren hallinnassa, riskien hallinnassa sekä liiketoimintahyötyjen tavoittelemisessa.

2.3 Yhteenveto

Edellisissä alaluvuissa selvitettiin järjestelmäevoluutiota ilmiönä. Evoluutioon perehtyminen aloitettiin kuvailemalla yleisesti ohjelmistotuotantoa ja siihen liittyvien kehityksen, ylläpidon ja operoinnin toimintoja. Ohjelmistotuotannon esitettiin olevan luonteeltaan palautteeseen perustuva, jatkuva prosessi. Tällä tarkoitetaan sitä, että järjestelmää kyetään muuntamaan paremmin ympäristöään huomioivaksi ainoastaan julkaisun ja käyttöönoton jälkeen. Toisin sanoen järjestelmästä ja sen toiminnasta saadaan palautetta vasta, kun järjestelmä on otettu käyttöön jossain kontekstissa.

Järjestelmäevoluutio voidaan käsittää kehityksen, ylläpidon ja operoinnin yhteistoimintana. Prosessi alkaa, kun järjestelmää aletaan toteuttaa. Järjestelmän evolutiivinen kehittyminen puolestaan päättyy, kun järjestelmän ylläpidosta ja operoinnista luovutaan lopullisesti. Lehman ja Ramil (2003) määrittelevät järjestelmäevoluution ilmiöksi, jossa ympäristötekijöiden vaikutus jollain tavalla muuttaa järjestelmän vaatimuksia ja täten asettaa paineen myös järjestelmän toiminnan muutoksille. Evolutiivisen kehittymisen vääjäämättömyys havainnollistuu, kun tarkastellaan muuttuvassa maailmassa toimivia yrityksiä. Muuttuva maailman paine aiheuttaa välttämättömän muutoksen siinä toimiville organisaatiolle ja niiden käyttämille tietojärjestelmille. Tällöin paineen nähdään kohdistuvan suoraan järjestelmiä kehittäviin organisaatioihin. Järjestel-

mäevoluutio tulee käsittää myös jatkuvaisluontoisena tapahtumana, sillä järjestelmien toimintaympäristöä ei voida käsittää ikinä valmiiksi. Tällöin myöskään turbulentissa ympäristössä toimivat E-tyypin järjestelmät eivät tule ikinä valmiiksi.

Jatkuvasta muutoksesta johtuen järjestelmillä on tapana kasvaa ja monimutkaistua. Mikäli evoluutioprosessia ei hallita järjestelmän nähdään pilaantuvan. Toisin sanoen järjestelmän sisäinen entropia lisääntyy termodynaamisen systeemin tapaan, koska järjestelmän toiminnallisuutta lisätään toimintaympäristön vaatimusmuutosten edellyttämällä tavalla. Lisäksi toiminnallisuuden kasvattaminen monimutkaistaa järjestelmäkokonaisuutta ja tekee sen jatkokehittämisestä ja ylläpidosta entistä haastavampaa.

Järjestelmien evolutiivista kehittymistä ja lisääntyvää entropiaa selvitettiin tarkemmin Lehmanin järjestelmäevoluution lainalaisuuksien kautta. Lainalaisuuksien kokonaisuus auttaa hahmottamaan piirteitä, joita tietojärjestelmiin (E-tyyppi) ja niiden tuotantoprosesseihin liittyy. Lainalaisuudet ovat jossain määrin päällekkäisiä ja lomittaisia, eli toinen laki voi näyttäytyä seurauksena toisesta ja päinvastoin. Kuitenkin Lehmanin ja Ramilin (2001) mukaan lait ovat toisistaan erillisiä ja pohjautuvat erilaisiin tutkimustuloksiin. Lainalaisuuksien huomioiminen evoluutioprosessin päätöksenteon näkökulmasta auttaa kuitenkin pitkällä aikavälillä kehitysorganisaatioita riskien hallinnassa sekä liiketoimintahyötyjen tavoittelussa (Lehman & Ramil, 2001, 559).

Suurin osa järjestelmäevoluution lainalaisuuksien puitteissa toteutetuista empiirisistä tutkimuksista on hyödyntänyt tilastollisia menetelmiä lainalaisuuksien vahvistamiseksi. Jatkuvan muutoksen sääntöä (*continuous change*) on tutkittu esimerkiksi havainnoimalla pitkällä aikavälillä tapahtuvien muutoksia ja niiden jatkuvuutta. Kasvavan kompleksisuuden (*increasing complexity*) lainalaisuutta on mitattu muun muassa analysoimalla pitkän aikavälin kehityksen aikana kasvavia järjestelmäkomponentteja sekä niiden välillä muodostuvia riippuvuuksia. Tuttuuden säilyttämisen (*conservation of familiarity*) sääntöä on puolestaan analysoitu mittaamalla pitkällä aikavälillä tapahtuvaa järjestelmän osien kasvutahtia. Organisatorisen vakauden säilyttämisen (*conservation of organizational stability*) lainalaisuutta on niin ikään havainnoitu pitkällä aikavälillä mitaten esimerkiksi yksittäisen kehittäjän tehtävien määrä ja määrän kasvua. Järjestelmän jatkuvan kasvun (*continuous growth*) lainalaisuutta on havainnoitu tutkimuksessa muun muassa analysoimalla järjestelmän osien määrän kasvua. Heikentyvää laatua (*declining quality*) on taas tutkittu analysoiden korjausten määrän kasvua järjestelmän pitkäaikaisen kehityksen aikana. Esitetyt kuusi lainalaisuutta on empirian kautta todettu vahvistetuiksi useissa tutkimuksissa. (Barry ym., 2007.)

Lainalaisuuksien paikkansapitävyyttä on tutkittu muun muassa ketterässä kehitysympäristössä. Sindhghatta ym. (2010) tutkivat esitettyjä kuutta lainalaisuutta erään järjestelmän tapauksessa, jonka kehittämisessä hyödynnettiin ketterän kehityksen periaatteita. Tutkimus osoitti, että kaikki edellä mainituista kuudesta lainalaisuudesta pitävät paikkansa myös ketterässä ympäristössä, huolimatta siitä, että lakien kehittäminen aloitettiin jo 1970-luvulla kauan ennen

ketterän ideologian alkua. Myös Capiluppin ym. (2007) sekä Kourin ja Singhin (2016) empiiriset tulokset ketterällä tavalla kehitetyistä järjestelmistä tukevat evoluution lainalaisuuksia.

Lakeja vahvistavilla tilastollisia menetelmiä hyödyntävillä tutkimuksilla on yhteinen piirre siinä, ne ottavat kantaa ainoastaan esitettyihin kuuteen lainalaisuuteen. Lisäksi lainalaisuuksia vahvistavien tutkimusten tulee olla pitkittäistutkimuksia, jotta kerätyt aineistot todella voivat esittää esimerkiksi järjestelmän koon kasvaneen (Barry ym. 2007). Itsesäätelevyyden (*self-regulation*) ja palautejärjestelmän (*feedback system*) lainalaisuudet poikkeavat muista kuudesta. Barry (ym. 2007) näkevät nämä kaksi lakia niin sanottuina meta-lakeina, sillä niissä esitetään kuinka kehitysprosessin ja organisaation näkökulma suhtautuu itse kehittyvään järjestelmään. Meta-lainalaisuuksien abstraktiotasosta johtuen niitä on vaikea vahvistaa empiirisillä malleilla. Barryn ym. (2007, 12) mukaan kuitenkin myös meta-lait voidaan vahvistaa epäsuorasti, sillä muiden lainalaisuuksien ollessa tosia myös evoluutioprosessin itsesäätelevyyden ja palautejärjestelmän täytyy joillain tasoilla toimia. Näin ollen Lehmanin esittämien järjestelmäevoluution lainalaisuuksien voidaan kaikkinensa nähdä olevan voimassa ja toiminnassa myös reaalimaailmassa.

3 DEVOPS

Tässä luvussa käsitellään tuoretta ohjelmistotuotannon trendiä, DevOpsia. Ensimmäinen luku keskittyy DevOpsin yleispiirteisiin. Aihepiiriä tarkastellaan aluksi havainnollistamalla ongelmia, joita DevOpsin nähdään ratkaisevan. Lisäksi DevOpsia konkretisoidaan tarkastelemalla sitä sen korostamista käytänteistä sekä esittämällä ohjelmistoalan tieteellisessä kirjallisuudessa tavattuja tapaustutkimuksia toimintamallin soveltamisesta ja haasteista. Viimeisessä alaluvussa esitetään yhteenveto DevOpsin tavoitteista ja käytänteistä.

3.1 DevOps ohjelmistotuotannossa

Kiinnostus ohjelmistomaailmasta nousseeseen DevOps-trendiin on kasvanut viimeisten vuosien aikana räjähdysmäisesti (Google, 2016). Kuten esimerkiksi Elliotin (2014) toteuttamasta tutkimuksesta ilmenee, pidetään ohjelmistoyritysten keskuudessa DevOpsia yhtenä alan kiinnostavimmista trendeistä. DevOpsista puhuttaessa tarkoitetaan toimintamallia, jonka mukaan pyritään hajottamaan funktionaalisesti järjestettyjen organisaatiorakenteiden aiheuttamia ongelmia. DevOpsissa kannustetaan ohjelmiston kehitys- ja operointitoimintojen välisen yhteistyön tiivistämiseen (Humble & Molesky, 2011; Hütterman, 2012; Lwakatare ym., 2016; McCarthy ym., 2015). Varsinaisesti DevOps-termin nimi muodostuukin kehityksen (engl. *development*) ja operoinnin (engl. *operations*) yhteistyöstä.

Syyt DevOps toimintamallin syntyyn juontuvat ohjelmisto-organisaatioiden kokemista ohjelmistokehityksen, -ylläpidon ja -operoinnin ongelmista ja hidasteista. Lwakatare ym. (2015) mukaan järjestelmäkehityksessä ongelmallisuudet liittyvät usein huonoihin kommunikoinnin toimintamalleihin, manuaalisesti tehtäviin julkaisutoimiin, kehityksessä ja laaduntarkkailussa tehtävien olettamusten epämääräisyyteen sekä kehityksen aikana automaattisesti syntyvän tiedon heikkoon hyödyntämiseen.

Kehityksen, ylläpidon ja operoinnin välistä kommunikoinnin heikkolaatuisuutta voidaan ymmärtää funktionaalisen organisaatiorakenteen puitteissa.

Bassin ym. (2013, 2) mukaan kehitys-, ylläpito- ja operointitiimien yhteistyön kehittämisen lähtökohtana tulisi olla, että tiimit ymmärtävät toistensa kohtaamia ongelmia. Hüttermanin (2012, 36-38) mukaan funktionaalisesti järjestettyjen tiimien eriytyneisyys johtaa erilaisiin konflikteihin ja sitä myötä mahdolliseen keskinäiseen syyttelyyn. Ristiriitojen alkuperä voidaan havaita esimerkiksi juuri ennen ohjelmiston julkaisua. Kehittäjät haluavat uudet ominaisuutensa nopeasti tuotantoon, mutta operoinnista vastaava osapuoli haluaa välttää nopeiden muutosten tekemisen järjestelmän vakaan toiminnan säilyttämiseksi (Hütterman, 2012, 35). Toisaalta konfliktit syntyvät myös ohjelmistojulkaisun seurauksena. Uuden huonolaatuisen toiminnon julkaisemisen myötä järjestelmän vakaus heikentyy ja voi huonossa tapauksessa aiheuttaa ohjelmiston kaatumisen (Hütterman, 2012, 37). Konfliktit saattavat aiheutua myös ohjelmiston suorituskykyyn liittyvistä ongelmista, sillä molemmat tiimit kokevat, ettei heidän tuotoksensa ole syynä kokonaissuorituskyvyn heikentymiseen (Hütterman, 2012, 37).

Huonoista toiminta- ja kommunikointimalleista johtuen on yleistä, että järjestelmien julkaisu osoittautuu hitaaksi ja vaivalloiseksi (Humble & Molesky, 2011). Kannattavan ohjelmistoliiketoiminnan kannalta on kuitenkin välttämätöntä, että järjestelmät julkaistaan nopeasti, jotta ne tuottavat mahdollisimman aikaisin arvoa niiden käytöstä maksaville sidosryhmille. Lisäksi julkaisutoimien hitaus näkyy myös teknisellä tasolla manuaalisesti tehtävistä julkaisutoimista (Lwakatare ym., 2015). Ensinnäkin järjestelmän kehitysversion toteuttaminen tuotantokelpoiseksi on hidasta, mikäli prosessi on manuaalinen. Hidasta tämä on etenkin silloin, kun operoinnista vastaavat osapuolet pyrkivät entisestään hidastamaan uusien muutosten toteuttamista järjestelmän vakauden säilyttämiseksi. Lisäksi ja ennen kaikkea suurten järjestelmien tapauksessa, myös systeemien monimutkaisuus osoittautuu julkaisua hidastavaksi tekijäksi. Tämä johtuu tavallisimmin siitä, että laajat järjestelmät koostuvat usein suuresta määrästä erillisiä kokonaisuuksia, joiden välisiä julkaisukohtaisia konfiguraatioita on erittäin työlästä hallita manuaalisesti. Liiketoiminnan kannattavuuden näkökulmasta julkaisu- ja operointiaktiviteetit osoittautuvat julkaisuprosessin ja täten myös organisaation tuloksen kerryttämisen hidasteiksi. (Hütterman, 2012.)

Lisäksi ohjelmistoja kehitettäessä on havaittu ongelmalliseksi, että kehityksen ja laaduntarkkailun tehokkuutta ei havainnoida tarpeeksi tarkoin mittarein (Lwakatare ym., 2015). Kyseisten toimintojen tehokkuudella on suora vaikutus tuotettavan järjestelmän julkaisunopeuteen ja laatuun. Kehitystyön mittaamisella voidaan arvioida esimerkiksi aikaa, joka kehitykseen kuluu. Näin ollen oikeilla mittareilla helpotetaan esimerkiksi julkaistavien ohjelmistoversioiden aikataulutusta. Mikäli kehityksen tehokkuuden ja laadun arviointi perustuu epämääräiseen mittaamiseen, on vaikea arvioida järjestelmän tilaa julkaisuhetkellä. Tällöin on selvää, että mittaaminen ja arviointi olisi tehokkaampaa, mikäli se perustuisi kerättävään ja tilastoituun tietomassaan (Lwakatare ym., 2015; Hütterman, 2012; McCarthy ym., 2015).

Järjestelmiä kehitettäessä tehdään oletuksia, siitä kuinka niiden todellisuudessa tulisi toimia. Oletuksia joudutaan tekemään tällöin myös testattaessa

järjestelmän laatua. Julkaisun jälkeen kuitenkaan järjestelmät eivät välttämättä toimi kehityksessä tehtävien oletusten mukaisesti. Tällöin myös laadun tarkkailussa oletetut seikat saattavat olla vääriä. Täten ongelmallista on, että julkaisukelpoiseksi oletettu järjestelmä ei todellisuudessa vastaakaan sille asetettuihin vaatimuksiin. Toisaalta palautetta järjestelmän toiminnasta ja sen laadusta saadaan aina vasta julkaistun version jälkeen, kun loppukäyttäjät ovat päässeet siihen käsiksi (Humble & Molesky, 2011). Lwakataren ym. (2015) mukaan ohjelmistotuotannossa ongelmallista on, että kehityksen aikana automaattisesti syntyvää tietoa ei kerätä eikä näin myöskään hyödynnetä. Tehokkaamman informaation keruun ja analysoinnin kautta ohjelmisto-organisaatiot kykenisivät tuottamaan nopeammin laadukkaampia järjestelmiä.

DevOps pitää sisällään joukon käytänteitä, joilla pyritään parantamaan eriytyneiden tiimien yhteistyötä sekä ratkaisemaan esitettyjä ongelmia. Humblen ja Moleskyn (2011, 7) mukaan DevOpsin tarkoituksena onkin, että kaikki ohjelmiston kehittämiseen osallistuvat tahot tähtäävät samaan lopputulokseen. Keskeisimpänä oletuksena on saada kehitys- ja operointitiimien välinen yhteistyö toimimaan siten, että nopeasti tehtävät muutokset saadaan julkaistua tuotantoon nopeasti ja luotettavasti laatua heikentämättä (Dyck ym. 2015; Humble & Molesky, 2011).

DevOpsista kiinnostuneiden ohjelmistoyritysten keskuudessa odotetaan, että toimintamallin hyödyntäminen nopeuttaisi ohjelmiston julkaisua ja parantaisi julkaisujen laatua (Elliot, 2014). DevOpsissa on kuitenkin päämärältään kyse muustakin kuin tiimiyhteistyön tehostamisesta. Yleisemmin puhutaan ketteristä ajatusmalleista, joiden korkean tason tavoitteet nähdään organisatorisesti merkittävinä. DevOps ajattelun tavoitteena nähdäänkin kyky parantaa ohjelmisto-organisaation kilpailukykyä nopeuttamalla ohjelmiston julkaisuun kuluvaa aikaa sekä parantamalla julkaisujen laatua (Fitzgerald & Stol, 2014; IBM, 2013; Humble & Molesky, 2011; Hütterman, 2012).

DevOps toiminnan mukaisesti ohjelmiston laatu paranee, kun koko ohjelmisto-organisaatio tähtää yhteisosaamisellaan toimintavarman järjestelmän kehittämiseen ja ylläpitoon. Roche esittää (2013, 5), että julkaisuun kuluvaa aikaa pyritään vähentämään DevOps käytänteiden mukaisesti tekemällä muutoksia tuotannossa olevaan ohjelmistoon jatkuvasti. Jatkuvan integraation periaate ja ohjelmiston laatunäkökulma konkretisoituvat tarkasteltaessa niitä DevOpsin eri perspektiivien kautta. Seuraavassa keskitytään tarkemmin avaamaan käytänteitä, jotka mahdollistavat laadukkaamman ja nopeamman julkaisuprosessin.

3.2 DevOpsin hyödyntäminen käytännössä

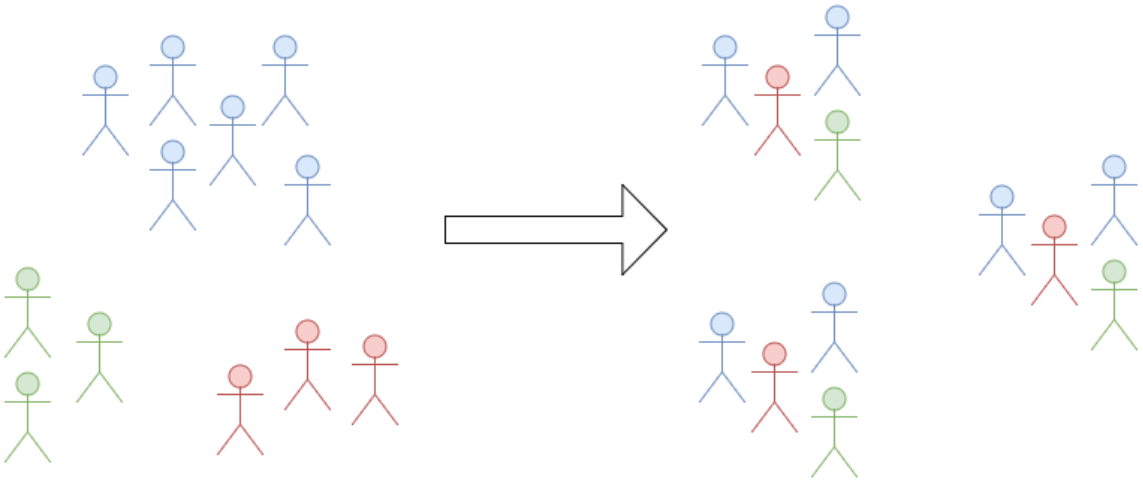
DevOpsin käytänteet ottavat kantaa ohjelmistoprosessin yhteistyön, automaation, monitoroinnin ja mittaamisen toimintatapoihin. Käytänteillä ohjelmisto-organisaatiot pyrkivät nopeuttamaan sekä helpottamaan ohjelmiston kehitykseen, julkaisuun, ylläpitoon ja operointiin liittyviä aktiviteetteja. Toisaalta käytänteiden tarkoituksena on myös parantaa ohjelmistojulkaisujen laatua (Roche, 2013). Toisin sanoen DevOpsissa pyritään yhteistyötä korostavan kulttuurin ja teknologisten ratkaisujen kautta saamaan aikaan nopea ja laatua tuottava ohjelmistotuotantoprosessi.

3.2.1 DevOps käytänteet

Tiiviimmän *yhteistyön* näkökulmasta DevOpsissa on tarkoituksena vähentää kehitys- ja operointitoimintojen eriytyneisyyttä. Yhteistyönäkökulman mukaisesti toimintamallissa julkaisuprosessin laatua ja nopeutta parannetaan jakamalla tiimien ja toimintojen välisiä tavoitteita, toimintatapoja sekä vastuita (Smeds ym., 2015, 6). Tehokkuuden nähdään laadun ja nopeuden suhteen paranevan, kun tiimien väliset tavoitteet, toimintatavat ja vastuut perustuvat samoihin lähtökohtiin (Swartout, 2014). Toisin sanoen voidaan ajatella koko ohjelmistoyrityksen tavoittelevan yhdessä samaa lopputulosta. Humblen ja Moleskyn (2011, 7) mukaan ohjelmistoyritysten tulee organisoida kehitystiiminsä siten, että operointitoiminto kommunikoi sujuvasti kehitysosapuolen kanssa mm. yhteisten tapaamisten tai kanavien kautta. Tällöin tiimien välinen kommunikaatio tehostuu ja ymmärrys lisääntyy eri alueen osaajilta opittujen asioiden johdosta (Hütterman, 2012, 26). Täten voidaan nähdä DevOpsin yhteistyöhön pohjaavan näkökulman korostaa vaivatonta tiimien välistä kommunikaatiota sekä jatkuvaa oppimista (Smeds ym., 2015, 6).

Esimerkki yhteistyötä korostavasta toimintatavasta nähdään Balalaien ym. (2016, 7) tapaustutkimuksesta. Heidän kuvaamassaan tapauksessa tehokas yhteistyön toimintatapa saatiin aikaan hajottamalla jakautunut tiimirakenne. Aikaisemmin erinäisinä organisaatioyksikköinä toimivat kehitys- ja operointitiimit järjesteltiin uudelleen pienimmiksi kokonaisuuksiksi. Näin uudet pienemmät tiimit koostuivat molempien toimintojen osaajista ja pystyivät ottamaan tehokkaammin vastuuta kokonaisista ohjelmistotuotteista ja koko ohjelmistoprosessin kulusta (Balalaie ym., 2016, 7). Lwakatare ym. (2016, 6) puolestaan havaitsivat tutkimuksessaan vastuita ja toimintatapoja jakavan yhteistyökulttuurin muodostuvan hieman eri tavalla. Heidän tutkimassaan organisaatiossa kullekin kehittäjätiimille oli varattu operointitiimin henkilö, joka auttoi kehitystä eteenpäin infrastruktuuriongelmien ilmetessä (Lwakatare ym. 2016, 6). Esitetyt esimerkit kuvaavat hyvin organisaatiokulttuurin arvoja, joita DevOps toimintamallissa haetaan. Hüttermanin (2012, 27) mukaan arvomaailman tulee pohjautua tiimien keskinäiseen arvostukseen sekä yhteisiin toimintatapoihin ja päämääriin. Kuten Swartout (2014, 22) huomauttaa, on tämä juuri se tapa, jolla

pienemmät ohjelmistoyritykset takaavat ketterän toimintatavan. DevOps ajattelutavan aikaansaamiseksi ohjelmistoyritysten tulisi organisoida tiiminsä useiden osaajien rakennelmiksi. Kun yhteistyö toimii eri tiimien välillä, on sillä suora vaikutus järjestelmän laatuun ja virhealttiuden poistamiseen. Lisäksi tehokas yhteistyö nopeuttaa julkaisu- ja ylläpitotoimia.



KUVIO 5 Monista osaajista koostuva tiimirakenne (Hütterman, 2012; Balalaie ym., 2016)

DevOps ajattelutavan *automaationäkökulma* on tärkeä huomioitava seikka ohjelmistoprosessin nopeuttamiseksi ja järjestelmien laadun takaamiseksi. Wallerin ym. (2015, 1) mielestä automaatio on yksin tärkein tekijä, joka vähentää julkaisuun tarvittavaa työmäärä ja aikaa, ja näin ollen myös vähentää ohjelmistoyritysten operatiivisia kustannuksia. Automaatiota DevOps ajattelussa pyritään soveltamaan julkaisuprosessin eri vaiheisiin (ks. kuvio 2) (Hütterman, 2012, 29-30; Smeds ym., 2015, 7). Automaatiolla ei ainoastaan nopeuteta julkaisuversioiden aikaansaamista, mutta myös parannetaan ohjelmiston laatua (Swartout, 2014, 44). Julkaisuautomaatio vähentää laadun heikkenemisen riskiä. Automatisoitu järjestelmäjulkaisujen kokoonpano on laadun kannalta manuaalista riskittömämpää, sillä käsintehtyinä julkaisukokoonpanoihin saattaa lipsahtaa virheitä tahattomasti.

Automatisoitu ohjelmistoprosessi mahdollistaa jatkuvan julkaisuintegraation (engl. *continuous integration*) ja jatkuvan julkaisun (engl. *continuous deployment*). Jatkuvan integraation ja julkaisun periaatteiden mukaisesti ohjelmistoyritykset kykenevät laajamittaisesti vähentämään aikaa, joka kuluu tuotteen saamiseksi loppukäyttäjille, asiakkaalle tai markkinoille. Jatkuvan integraation ja sitä seuraavan jatkuvan julkaisun ajatuksena on, että ohjelmistoa rakennettaan iteratiivisten ja inkrementaalisten periaatteiden mukaan tekemällä pieniä muutoksia ja julkaisemalla nämä nopeassa aikataulussa tuotantoon. Ohjelmiston laadun nähdään paranevan, koska inkrementaalisesti toteutettavat muutokset koskettavat vain pientä osaa koko ohjelmistosta. Suurten muutosten tekeminen puolestaan saattaisi heikentää ohjelmiston toimintavarmuutta (Roche, 2013, 5). Smeds ym. (2015, 7) näkevät, että julkaisuautomaatio mahdollistaa

myös aikaisempaa tuottavamman työnkuvan. Kun aikaisemmin manuaalisesti tehtävät työt korvataan automaatiolla, on yksittäisellä henkilöllä enemmän aikaa keskittyä avaintehtäviinsä (Smeds ym., 2015, 7).

Basiri ym. (2016) esittävät, että suoratoistovideopalvelua tuottava Netflix on mainio esimerkki ohjelmistotalosta, joka hyödyntää jatkuvan integraation ja julkaisun prosesseja laadun parantamiseen ja julkaisun nopeuttamiseen. Netflixillä ohjelmiston laatu ja jatkuva integraatio koetaan kriittiseksi, koska videopalvelun täytyy olla jatkuvasti toimintakykyinen. Uudet toiminnallisuudet täytyy täten viedä suoraan tuotannossa olevaan järjestelmään asiakkaiden käyttöön. (Basisiri ym., 2016, 38). Myös Liu ym. (2014) havaitsivat tapaustutkimuksessaan jatkuvan integraation periaatteen hyödyllisyyden. Tutkimuksessa seurattiin erään IBM:n ylläpitotoimintoyksikön prototyypin kehittämistä, missä tavoiteltiin kehitystiimien nopeampaa ja tehokkaampaa reagointia käyttäjärajapinnasta tuleviin muutosehdotuksiin. Ennen prototyypin käyttöönottoa, ohjelmistotuoteyksiköille sateli lukemattomia muutosehdotuksia ja suuresta määrästä johtuen joidenkin näistä toteuttaminen unohtui tai jäi muista syistä tekemättä. Prototyypin käyttöönoton jälkeen puolestaan yksikään muutosehdotuksista ei jäänyt toteuttamatta sen edellyttämässä aikataulussa (Liu ym., 2016). Toisin sanoen prototyypin käyttöönotto kehitti tuotantoprosessia automatisoidummaksi mahdollistamalla täsmällisemmän ja täten myös nopeamman reagoinnin palautteeseen.

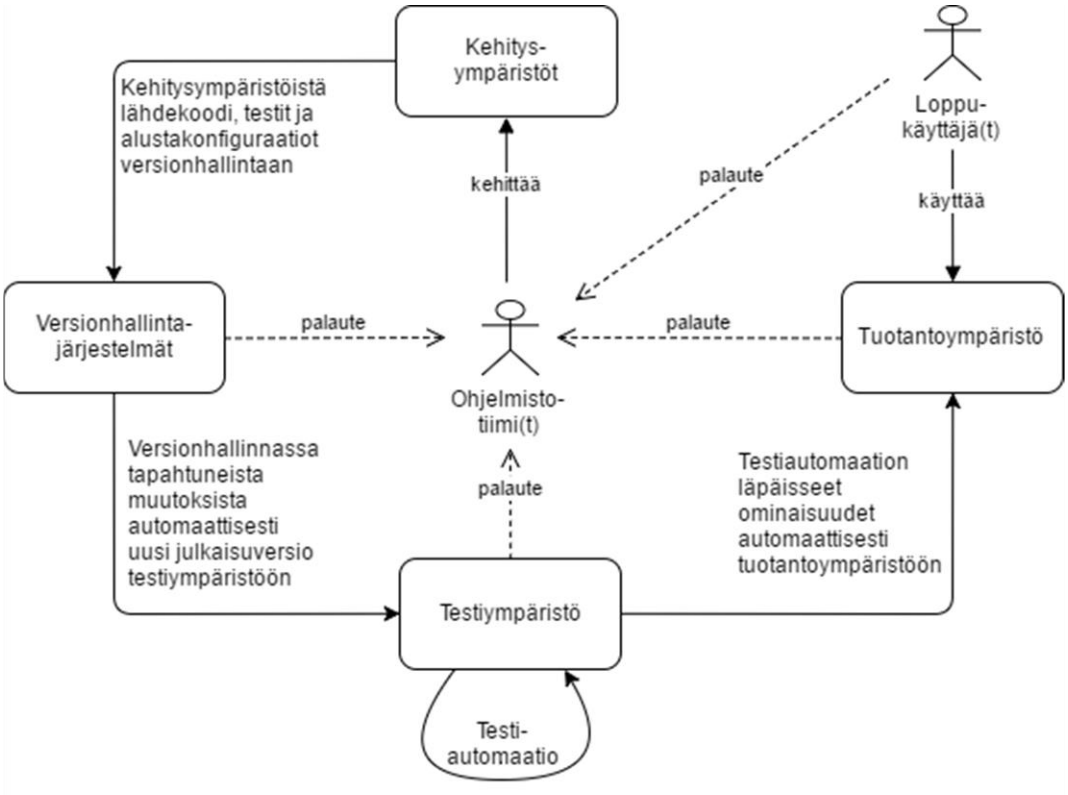
Julkaisunopeuden ja järjestelmän laadun parantamiseksi DevOps ajattelutavassa korostetaan myös ohjelmistotiimien tietämyksen lisäämistä *monitoroinnin* ja erilaisten *palautejärjestelmien* avulla (Smeds ym., 2015, 7; Hütterman, 2012). Lwakataren ym. (2016) mukaan tehokkaalla monitoroinnilla varustettu järjestelmä ja kehitysprosessi tarjoavat järjestelmäkehitykseen merkittävää tietämyksen lisäystä. Monitoroinnin avulla kerättävää informaatiota on tällöin myös mielekästä seurata. Monitorointiratkaisuilla voidaan havainnoida esimerkiksi järjestelmän suorituskykyä, erilaisia virheraportointitilastoja ja jopa loppukäyttäjien uusia toiminnallisuusehdotuksia (Lwakatare ym., 2016; Hüttermann, 2012). Monitoroinnin tarkoituksena on, että ohjelmistoprosessissa toimivat kehittäjät saavat tietoa sen hetkisestä järjestelmän toiminnasta ja toimintakyvystä (ks. kuvio 2).

Stillwell ja Coutinho (2015) havaitsivat tutkiessaan usean tiimin hajautettua ohjelmistokehitystä julkaisu- ja palauteautomaation hyödyttävän ohjelmistoprosessia usealla tavalla. Ensinnäkin tutkimuksen tulokset osoittivat, että tutkitussa hankkeessa mukana olleet kehittäjätiimit kykenivät toimimaan autonomisemmin ja tehokkaammin keskittyen avaintehtäviinsä, koska he saivat palautetta järjestelmän monitoroinnin kautta. Hankkeessa käytetty automatisoitu palauteketju taas vähensi tiimien välisen koordinoidun kommunikoinnin tarvetta pienten muutosten tekemiselle ja toisaalta mahdollisti eri kehittäjätiimien suopeamman suhtautumisen muutoksiin ja ripeisiin muutostoteutuksiin. (Stillwell & Coutinho, 2015.)

Myös Callanan ja Spillane (2016) havaitsivat automaatio- ja monitorointiratkaisuilla olevan merkittäviä myötävaikutuksia julkaisunopeuteen. Tutkitussa

ohjelmistoyksikössä alkuperäisenä ongelmana ilmeni, että kehitys- ja operointitoiminnot olivat aliresursoituja ja, että julkaisuprosessi oli tehty byrokraattisesti hankalaksi. Tämä näkyi muun muassa siinä, että ohjelmistojulkaisuun käytetty aika oli huomattavan pitkä ja julkaisuille tuli varata aikataulu etukäteen. Automaatio- ja monitorointiratkaisujen käyttöönoton jälkeen ohjelmistotestausta ajettiin jokaisen versionhallintamuutoksen sekä testiympäristöön julkaisun yhteydessä. Automatisoidun julkaisemisen seurauksena ohjelmistotiimit saivat nopeasti palautetta huonolaatuisista toiminnoista ja infrastruktuuriongelmista seuraamalla sekä automaattitesteistä saatuja tuloksia tuotantoympäristön käyttäytymistä.

Kuviossa 6 esitetään, kuinka automaatio- ja palautejärjestelmäratkaisut voidaan DevOps käytänteiden mukaisesti toteuttaa. Ohjelmistotiimit kehittävät järjestelmiä omissa kehitysympäristöissään. Kehitysympäristöt sisältävät järjestelmän lähdekoodia, testitapauksia sekä ohjelmistoalustan konfigurointitietoja (*engl. infrastructure as code*). Erityisesti pilvialustojen käytön yleistyminen on ollut mahdollistamassa sen, että ohjelmistoalustoja kyetään hallitsemaan erilaisten konfiguraatiotietojen avulla. (Jabbari ym., 2016).



KUVIO 6 Automatisointi ja palaute DevOps toimintamallissa

Kehitysympäristöstä ohjelmistokehittäjät vievät järjestelmätoteutukset kaikkine osineen (lähdekoodi, testit ja alustakonfiguraatiot) versionhallintajärjestelmään, jossa jatkuva integraatio käynnistyy. Versionhallintaan vietyjen uu-

sien muutosten seurauksena tiettyjä teknologisia ratkaisuja hyödyntäen koostetaan automaattisesti uusi julkaisuversio. Tämä julkaisuversio siirtyy testiympäristöön hyväksyttäväksi. Mikäli osa uuden julkaisuehdokkaan toiminnoista läpäisevät automaattisen testauksen, viedään ne automaattisesti testiympäristöstä jatkuvan julkaisun periaatteen mukaisesti tuotannossa olevaan järjestelmään loppukäyttäjien saataville. (Hütterman, 2012; Humble & Molesky, 2011.)

Kuvion 2 katkoviivoin esitetyt syötteet kuvaavat kehityksen ja operoinnin palauteketjuja. Versionhallintajärjestelmästä ohjelmistotiimit saavat tärkeää tietoa järjestelmäjulkaisun versiosta eli siitä kokonaisuudesta mikä on kulloinkin järjestelmän kokonaisrakenne. Lisäksi versionhallintajärjestelmistä on saatavilla myös tietoa tehdystä muutoshistoriasta. Kun julkaisuehdokas on versionhallinnan muutosten perusteella automaattisesti julkaistu testiympäristöön, tiimit saavat palautetta testiautomaation tuloksista sekä testiympäristön vakaudesta ja toiminnasta. Tällöin palautteen perusteella sekä toiminnallisiin että laadullisiin virheisiin pystytään reagoimaan nopeasti. Tuotantoympäristön monitoroinnista on ensisijaisesti hyötyä operointi mielessä. Tuotantoympäristössä toimivaa järjestelmää usein monitoroidaan usealla tavalla ja niillä saadaan tietoa esimerkiksi alustan tehokkuudesta kuormituksen aikana (Lwakatare ym., 2015). Toisaalta myös järjestelmän loppukäyttäjät toimivat palautteen välittäjinä. Heiltä ohjelmistotiimit saavat tietoa muun muassa huomioimattomista virheistä, tiettyjen ominaisuuksien hyväksynnästä sekä uusista mahdollisista toiminnoista. (Hütterman, 2012; Humble & Molesky, 2011.)

DevOps toiminnassa rikotaan tavanomaisen kehityksen raja-aitoja organisoimalla tiimitoiminta siten, että ohjelmistotiimit koostuvat sekä kehityshenkilöstöstä että operointihenkilöstöstä. Tästä syystä tavanomaisen järjestelmäkehitykseen suunnitellut mittaamisen tavat eivät ole yleispäteviä. Humblen ja Moleskyn (2011) mukaan DevOpsin mukaisella mittaamisen käytänteellä tarkoitetaan, että järjestelmäkehitysprosessissa mitataan ohjelmistoliiketoiminnan kannalta kriittisiä, korkean tason tavoitteitta. Ennen kehitystoimintoa mitattiin esimerkiksi suhteessa järjestelmän laatuun tai operointitoimintaa suhteessa järjestelmän vakauteen. DevOpsin mukaan toimittaessa mittaamisessa tulisi pikemminkin keskittyä järjestelmien läpimenoaikaan liittyviin tekijöihin eli siihen kuinka nopeasti järjestelmä saadaan kehityksestä loppukäyttäjien saataville.

3.2.2 Haasteet DevOpsin soveltamisessa

Vaikka DevOps käytäntöjen hyöty havaitaan edellä esitettyjen esimerkkien kautta, ovat useat tutkijat raportoineet myös ongelmista DevOps toiminnan soveltamisesta ja erityisesti sen käyttöönotossa. Liun ym. (2014, 42) mukaan ohjelmistoyritysten tuleekin ratkaista varsin suuri joukko erilaisia haasteita, jotta DevOpsin tarjoamat hyödyt konkretisoituvat. Heidän mukaansa haasteet korostuvat, kun yritykset pyrkivät parantamaan eriytyneiden tiimien yhteistyötä, saavuttamaan työn tehokkuutta palauteketjujen avulla sekä nopeuttamaan julkaisuprosessia. (Liu ym., 2014, 42.)

Tiimien välisen yhteistyön parantamisessa ongelmien nähdään yleisesti syntyvän organisaation ja tiimien arvomaailman eroavaisuuksista. Haasteet tehokkuuden aikaansaamisessa puolestaan nousevat esiin tarkasteltaessa ohjelmistoyritysten liiketoimintaprosesseja sekä niiden tehokkuutta ja vaikuttavuutta. Prosessien merkitys korostuu myös julkaisujen nopeuttamisessa, vaikka myös teknologisten ratkaisujen aiheuttamat ongelmat hankaloittavat näiltä osin DevOps käytänteiden hyödyntämistä.

Diel ym. (2016) tarkastelivat kehitys- ja operointitoimintojen kokemia kommunikoinnin haasteita maantieteellisesti hajautetussa ohjelmistoorganisaatiossa. Tutkittavassa organisaatiossa kehitystoiminnot oli sijoitettu Brasiliaan, kun taas järjestelmien operointitoiminnot sijaitsivat Euroopassa. Haastatellun henkilöstön mukaan suurin ongelma johtui maanosien välisestä aikaerosta. Tiimit eivät olleet käytettävissä samanaikaisesti. Näin ollen operointihenkilöstö ei kyennyt ottamaan yhteyttä kehitystiimiin ongelmien ilmetessä. Samainen ongelma koettiin myös kehityspuolella infrastruktuuriongelmien esiintyessä. Toisaalta tiimien kokemat haasteet aiheutuivat myös yhteisten käytäntöjen puutteesta. Brasiliassa sijaitseva kehitystiimi ei aina saanut selkeitä ilmoituksia operointitoimintojen tekemistä ohjelmistojulkaisuista eikä näin ollen tiennyt ohjelmistossa tapahtuneista muutoksista. Kummassakin yksikössä koettiin, että toisen työskentelytapojen parempi tunteminen sekä virallisen ja yhteisen kommunikointikanavan käyttö olisi hyödyttänyt havaittujen ongelmien ratkaisua. (Diel ym., 2013.)

Claps ym. (2015) tutkivat jatkuvan integraation prosessin käyttöönotosta aiheutuvia teknisiä ja sosiaalisia haasteita. He havaitsivat ongelmien heijastuvan niin työntekijöiden, organisaation kuin asiakassidosryhmänkin tasoille. Ohjelmistokehityksessä työskentelevä henkilöstö koki jatkuvan integraation aiheuttavan paineita, sillä esimerkiksi kehittäjät olettivat, että heidän täytyi tuottaa toimivaa lähdekoodia ohjelmistoalustan vakauden säilyttämiseksi. Toisaalta organisaation uusien työntekijöiden keskuudessa koettiin prosessin ymmärtäminen hankalaksi, koska sen kulkua ei oltu dokumentoitu tarkasti eikä sen soveltamisen tasolla noudatettu juuri lainkaan ohjelmistoalan standardeja. Johdon tasolla ei puolestaan ymmärretty, että prosessin käyttöönotto vaatii ponnistuksia lähes kaikissa organisaatioyksiköissä. Kaikissa yksiköissä käyttöönottoon ei kuitenkaan osattu motivoitua. Motivaation puute johtui osaltaan siitä, että tiimien täytyi omaksua uusia toimintatapoja, mikä puolestaan aiheutti muutosvastarintaa. Johdon tasolla havaittiin myös, että jatkuvan prosessin omaksuminen vaati ylimääräistä vaivannäköä usean eri tiimien koordinoinnin toteuttamiseksi. Lisäksi huomattiin, että uusi toimintamalli vaatii varsin kokeneita jäseniä jokaiseen tiimiin. (Claps ym., 2015.)

Smeds ym. (2015) puolestaan tutkivat yksilöiden kokemia DevOpsin käyttöönoton haasteita. Heidän tutkimuksessaan haastateltiin erään ohjelmistoyrityksen henkilöstöä ja selvitettiin henkilöstön kokemuksia uuden toimintamallin käyttöönottoon liittyen. Clapsin ym. (2015) tulosten mukaisesti Smeds ym. (2015) havaitsivat haasteiden heijastuvan yksilöiden ja organisaation tasolle. Haastateltavien työntekijöiden keskuudessa ei täysin ymmärretty uuden De-

vOps toimintamallin merkitystä. Ajatusmallin monitulkintaisuuden nähtiin johtavan epäselviin toimintatapoihin organisatoristen tavoitteiden saavuttamisessa. Niin ikään muutosvastarintaa havaittiin syntyvän monesta eri syystä. Henkilöstön keskuudessa koettiin DevOpsin soveltamisen tarkoittavan, että heiltä edellytettiin kehitys- sekä operointiosaamista. Tämä puolestaan johti siihen, että uudet toimintatavat koettiin kuormittavina ja työmäärää lisäävinä haittoina. Muutoksen vastustus huomattiin myös, kun DevOps koettiin uutena trendikkäänä suuntauksena, joka ei todellisuudessa toisi mitään uutta aikaisempaan tapaan toimia. Yleisesti koko organisaation tasolla kehitys- ja operointitiimit eivät olleet juurikaan kiinnostuneita toistensa tekemisistä. Näin ollen ei ylipäätään haluttu ottaa käyttöön toimintatapaa, joka vaikuttaisi olemassa oleviin käytänteisiin. Erityisesti ongelmalliseksi koettiin yhtenäisemmän kommunikaation ja yhteisten päämäärien saavuttaminen. Osittain henkilöstön keskuudessa oltiin huolestuneita myös vanhojen teknisten ratkaisujen uudistamista. Esimerkiksi useiden tuotantoympäristöjen päivittäminen koettiin haastavaksi ja työlääksi. Lisäksi kehittäjät esittivät laadun varmistuksen hankalammaksi, sillä kehitys- ja testiympäristöt eivät täysin vastanneet tuotantoympäristöä.

3.3 Yhteenveto

Edellä esitetyt haasteet kuvaavat, että DevOps koetaan ohjelmisto-organisaatioissa hankalasti hallittavana ja epämääräisenä kokonaisuutena. Myös Lwakataren ym. (2016) tutkimus osoittaa samankaltaisen trendin. Tämä on sinällään erikoista, sillä, kuten IBM:n (2013) ja Elliotin (2014) muutamaa vuotta aikaisemmin julkaistut raportit osoittavat, nähdään DevOps kuitenkin varsin houkuttelevana ajatuksena samaisten tahojen keskuudessa. Ristiriitaisuus huomioiden välillä johtunee siitä, että DevOpsia ei toteuteta organisaatioissa samoilla tavoilla (Lwakatare ym., 2016). DevOpsin voidaan nähdä kuitenkin rakentuvan esiteltyjen neljän näkökulman varaan: yhteistyö, automaatio, monitorointi ja mittaaminen ratkaisujen muodostavat konkreettisen DevOps toiminnan ytimen (Hütterman, 2012, 31).

Taulukossa 2 on esitettynä DevOpsin mukaiset käytänteet, niiden lyhyet selitteet sekä ongelmat, joita käytänteillä pyritään ratkaisemaan. Taulukkoon sekä esitettyihin tutkimuksiin ja niiden tuloksiin vedoten havaitaan, että kullakin käytänteellä on selkeästi vaikutusta järjestelmän laadun muodostumiseen ja julkaisuprosessin nopeuttamiseen.

Kuten DevOpsin käytänteitä esittelevien tapaustutkimusten yhteydessä havaittiin, ei käytänteiden soveltamiselle ole yhtä yhtenäistä tapaa. Tiiviimmän yhteistyön aikaansaamista tarkasteltiin perehtymällä erilaisiin tiimirakennelmiin muun muassa Balalaien ym. (2016) ja Lwakataren ym. (2016) tutkimusten pohjalta. Balalaien ym. (2016) tulosten mukaan tiimikulttuuri perustui monista osaajista koostuviin tiimeihin, kun taas Lwakataren ym. (2016) tapaustutkimuksen tulokset osoittivat, että kehitystoiminnolle oli varattu yksittäinen vastuuhenkilö operointitoiminnosta. Tiiviimmän yhteistyön lähtökohtana yleisellä

tasolla on poistaa funktionaalisen organisaatiorakenteen aiheuttamia ongelmia. Yhteistyöllä pyritään siihen, että ohjelmistokehityksen tiimit jakavat tavoitteet ja vastuun lopputuloksesta. Yhteisyötä korostava kulttuuri ensinnäkin vähentää julkaisuun liittyviä konflikteja sekä parantaa tiimien keskinäistä kommunikointia. Toisaalta hyvällä yhteispelillä luodaan myös julkaisuprosessista sulavampi, sillä ongelmien ilmetessä sekä kehittäjät että operoinnista vastaavat osallistuvat yhdessä ongelmien ratkaisuun.

Erot organisaatioiden tavoissa toteuttaa DevOps käytänteitä näkyvät myös automaatioratkaisujen erilaisuutena. Stillwell ja Coutinhon (2015) tutkimuksessa automaatiolla tavoiteltiin usean hajautetun tiimin toiminnan yhteensovittamista. Heidän tuloksensa osoittivatkin, että palauteketjujen automatisointi mahdollisti tiimien autonomisemman toiminnan ja suopeamman suhtautumisen muutoksiin. Callananin ja Spillanen (2016) tutkimustulokset puolestaan havainnollistivat, kuinka automaatiolla ratkaistiin byrokraattisesti jäykän julkaisuprosessin sujuvuus ja nopeuttaminen. Laajalla automaation hyödyntämisellä on suora ja selkeä vaikutus julkaisuprosessin nopeuteen ja tuotettavan järjestelmän laatuun. Ensinnäkin manuaalisten tuotantoprosessien automaatio nopeuttaa järjestelmien tuotannollisen version aikaansaamista. Toisekseen julkaisunopeutta edistää se, että yksittäisellä työntekijällä automaation seurauksena enemmän aikaa omiin avaintehtäviinsä. Lisäksi automaatio myös vähentää virheiden määrää järjestelmässä, sillä julkaisuprosessiin ei enää tarvita virhealttiita käsin tehtäviä toimenpiteitä.

TAULUKKO 2 Yhteenveto DevOpsin käytänteistä

Devopsin käytänne	**Käytänteen tarkoitus**	**Ratkaistavat ongelmat**
Yhteistyö	Yhteistyöllä tavoitellaan, että järjestelmäkehittäjät ja -operoijat toimivat tiiviimmin yhdessä. Yhteistyön lähtökohtana on, että tiimeillä on jaetut tavoitteet ja vastuut.	funktionaalinen organisaatiorakenne, tiimien väliset konfliktit, hidas julkaisuprosessi
Automaatio	Automaatiota pyritään soveltamaan mahdollisimman laajasti koko ohjelmistoprosessissa ja sitä hyödynnetään esimerkiksi tuotantoversion aikaansaamisessa, ohjelmistotestauksessa sekä palauteketjujen muodostamisessa.	hidas julkaisuprosessi, järjestelmän laatuongelmat, työskentelyn hitaus
Monitorointi	Erilaisilla monitoroinnin ratkaisuilla tarkkaillaan järjestelmän suorituskykyä ja mahdollisia toiminnallisuusvirheitä.	järjestelmien laatuongelmat, hidas muutosten toteuttaminen
Mittaaminen	Mittaamisessa keskitytään ohjelmiston läpimenoaikaa tarkastelevaan arviointiin.	hidas julkaisuprosessi, tiimien väliset konfliktit

Niin ikään monitoroinnin ratkaisut ovat sidoksissa ohjelmisto-organisaatioiden toimintatapoihin. Kyseistenkään ratkaisujen puitteissa ei ole mielekästä olettaa, että mikään organisaatio toteuttaisi niitä täysin samalla tavalla. Stillwell ja Coutinho (2015) tutkimuksessa mittaamisen ja monitoroinnin ratkaisuja käytettiin muun muassa mittaamaan järjestelmän suorituskykyä. Mittareiden ja monitoroinnin avulla ohjelmistokehitystiimit saivat tietoa sekä järjestelmän vakaudesta ja mahdollisista toimintavirheistä. Tiedon saannin seurauksena kyettiin nopeasti reagoimaan sekä alustan että toiminnallisuuden virheisiin. Nopea reagointi ennen kaikkea nopeutti muutosten toteuttamista mutta osaltaan myös vähensi tiimien välisen koordinoidun kommunikoinnin tarvetta. Tämä johtui siitä, että tiimit olivat vastuussa tietystä osasta järjestelmäkokonaisuutta ja keskittyivät reagoimaan tiettyihin monitoroitaviin aspekteihin. Callananen ja Spillanen (2015) tutkimus osoitti mittaamisen ja monitoroinnin ratkaisuilla olevan samankaltaisia tuloksia.

DevOpsin muokatessa ohjelmistotuotannon toimintatapaa ei toiminnan tehokkuutta voida enää mitata vanhojen tapojen mukaisesti. Aikaisemmin ohjelmiston kehitystoiminnan tuloksellisuutta mitatiin tyypillisesti arvioiden toteutetun järjestelmän laatua ja operointitoiminnan tehokkuutta tavallisimmin puolestaan suhteessa järjestelmän vakauteen. DevOpsin myötä tuloksellisuuden ja tehokkuuden mittaamisen perustana keskitytään ennemminkin järjestelmien läpimenoaikaan liittyviin aspekteihin. Toisin sanoen pyritään havaitsemaan ohjelmistotuotantoprosessin sujuvuutta hidastavia tekijöitä. (Humble & Molesky, 2011.)

Yhteenvetona voitaneen todeta, että DevOps ei suoranaisesti ole ketterien menetelmien tapaan ohjelmistokehitysmenetelmä. Ennemminkin DevOps on käsitettävä toimintamallina, jonka mukaan ohjelmistoprosessissa tulee hyödyntää yhteistyön, automaation, monitoroinnin ja mittaamisen käytänteitä helpompien, nopeampien ja laadukkaampien kehitys-, julkaisu-, operointiprosessien aikaansaamiseksi.

4 DEVOPS JA JÄRJESTELMÄEVOLUUTIO

Tässä luvussa esitetään tulkintoja ja yhteenvetoa aikaisemmissa luvuissa kuvatuista järjestelmäevoluution ja DevOpsin ilmiöistä sekä niiden yhteyksistä. Luku on jaettu kahteen alaosioon, joista ensimmäisessä kerrataan DevOpsin ja järjestelmäevoluution merkitystä ohjelmistotuotannon näkökulmasta. Lisäksi ensimmäisessä alaosiossa selvitetään tarvetta, jota DevOpsin tutkiminen järjestelmäevoluution näkökulmasta edesauttaa. Luvun toinen alaosio puolestaan havainnollistaa konkreettisemmin DevOpsin ja sen käytänteiden ilmenemistä järjestelmäevoluution sekä sen hallinnan näkökulmasta.

4.1 Tarve DevOpsin ja järjestelmäevoluution kytkökselle

Edellisessä luvussa on kuvattu varsin tuoreen ohjelmistokehitystrendin, DevOpsin, merkitystä ja käytänteitä. DevOps toimintamallin synnyn havaittiin perustuvan käytännön ohjelmistotuotannossa ilmeneviin ongelmiin. Pääsääntöisesti näiden ongelmien kuvattiin johtuvan ohjelmistokehityksen ja -operoinnin välisistä ongelmista ja hidasteista. Tarkemmalla tasolla osoitettiin, että ongelmat konkretisoituvat esimerkiksi silloin, kun ohjelmistoprosessissa hyödynnetään heikkoja kommunikoinnin ja yhteistyön toimintamalleja. Toisaalta kirjallisuuteen perustuen havaittiin myös, että riskialttiit manuaalisesti tehtävät julkaisutoimet saattavat aiheuttaa hidasteita ja ongelmia. Lisäksi osasyynä hidasteiden ilmenemiselle huomattiin olevan heikohko digitaalisen tiedon hyödyntäminen.

Kolmas luku osoitti, että DevOpsissa tarkoituksena on yhteistyön, automaation, monitoroinnin ja mittaamisen käytänteillä tarjota järjestelmäkehitysympäristöön toimintamalli, joilla esitettyjä ongelmia kyetään lieventämään. Yleisesti havaittiin, että DevOps toimintamallin mukaiset käytänteet ennen kaikkea helpottavat ja nopeuttavat ohjelmistokehitys-, julkaisu- sekä operointitoimintaan liittyviä toimenpiteitä. Lisäksi käytänteillä havaittiin olevan vaikutusta järjestelmien laadun rakentumisessa.

Tutkielman toisessa luvussa tarkasteltiin ohjelmistotuotantoa ja järjestelmäevoluutiota. Järjestelmäevoluutiosta puhuttaessa tarkoitetaan järjestelmän elinkaareen liittyvää ilmiötä, jossa järjestelmän ympäristötekijöiden vaikutus jollain tavalla muuttaa järjestelmän vaatimuksia ja asettaa näin paineen myös järjestelmän toiminnan muutokselle.

Järjestelmäevoluutio pohjaa reaalimaailman ennustettavuuden epämääräisyyteen eli siihen, ettei järjestelmien toimintaympäristöjen muutoksia kyetä missään tilanteessa etukäteen ymmärtämään. Tutkielman toisessa luvussa järjestelmäevoluutiota tarkasteltiin Meir Lehmanin (1974-1996) esittämien järjestelmäevoluution lainalaisuuksien kautta (*engl. laws of software evolution*). Lainalaisuudet kuvaavat järjestelmien kehittämisen ja ylläpidon erityispiirteitä ja haasteita. Osaltaan ne myös auttavat hahmottamaan evoluutioprosessinhallintaan liittyviä epävarmuustekijöitä. Toisessa luvussa osoitettiin, että Lehmanin lainalaisuuksia voidaan pitää pätevinä vertailukohtina järjestelmäevoluutiota tutkittaessa, sillä ne on luotu empiiristen tulosten pohjalta. Lisäksi lainalaisuudet on vahvistettu myös tutkittaessa modernimpaa järjestelmäkehitysympäristöä.

Ketterien kehitysmenetelmien tapaan DevOps toimintamallin tarkoituksena on tietyin käytäntein parantaa ohjelmistoja tuottavien tahojen toimintaa. Ketterissä menetelmissä toiminnan parantaminen näkyy pääsääntöisesti ohjelmistokehitystoiminnassa siten, että ketterästi toimittaessa kehitettävästä järjestelmästä saadaan palautetta mahdollisimman varhaisessa vaiheessa. Kehityksen ketteryys näin ollen paranee, koska palautteen saannin nopeutuessa myös kehitettävää järjestelmää kyetään muuntamaan nopeammin. DevOps toimintamalli on ketteriä ohjelmistokehitysmenetelmiä kokonaisvaltaisempi. DevOps toiminnassa ei oteta huomioon ainoastaan kehityksen näkökulmaa ja siihen liittyviä aktiviteetteja. DevOps ei siis ketterien menetelmien tapaan sanele kehitysprosessin kulkua. Pikemminkin toimintamalli ehdottaa ajatuksia ja käytänteitä tehostamaan koko ohjelmisto-organisaation toimintaa.

Tuottavan ohjelmistoliiketoiminnan näkökulmasta ohjelmisto-organisaatioiden on kyettävä muuttamaan kehittämiään järjestelmiä muuttuvan liiketoimintaympäristön vaatimalla tavalla. Lehmanin esittämä kuva järjestelmäevoluutiosta asettaa yleispätevät reunaehdot ja edellytykset sille, kuinka tietojärjestelmiä ja niiden kehitystä tulisi hallita ajan kuluessa. DevOps toimintamallin merkitys pohjaa tieteellisen kirjallisuuden perusteella järjestelmäkehityksen ja -operoinnin väliseen kitkaan ja siitä seuraavaan ohjelmistoprosessin hidastumiseen ja järjestelmien laatuongelmiin. Näin ajateltuna DevOps toimintaa ja sen käytänteitä ei varsinaisesti ole luotu ratkaisemaan järjestelmäevoluution hallinallisia kysymyksiä. Järjestelmäevoluution huomioiminen on kuitenkin havaittu ohjelmisto-organisaatioiden toiminnan kannalta strategisesti merkittäväksi.

DevOps toimintaa eikä etenkään siihen liittyviä käytänteitä ole vielä tieteellisessä tutkimuksessa järjestelmäevoluution liittyen tutkittu. DevOps on myös varsin tuore ilmiö ohjelmistotuotannon toimintakentässä. Järjestelmäevoluution strategisen merkityksen ja DevOpsin tutkimattomuuden vuoksi De-

vOpsin ja järjestelmäevoluution kytkösten tutkiminen on merkittävää. Ilmiöiden välisten yhteyksien tutkimisen kautta voidaan ennen kaikkea selvittää, onko DevOpsin lupaus tehokkaammasta ohjelmistotuotantotoiminnasta todellinen. Lisäksi tarkempi tutkimus DevOpsin käytänteiden ja järjestelmäevoluution lainalaisuuksien välillä voi paljastaa uusia näkökulmia toimintamallin hyödyllisyydestä.

4.2 Järjestelmäevoluutio DevOpsin näkökulmasta

DevOpsissa ei suoranaisesti oteta kantaa järjestelmäevoluutioon. Kuitenkin tarkasteltaessa toimintamallin käytänteitä havaitaan, että ne osittain huomioivat Lehmanin esittämiä säännönmukaisuuksia (ks. taulukko 3). Esimerkiksi jatkuvan muutoksen (*engl. continuing change*) lainalaisuuden huomiointi havaittiin Liun ym. (2016) tutkimuksessa. Toisaalta tutkimuksessa osoitettiin, että järjestelmän toimintaympäristö on jatkuvassa muutoksessa, sillä ohjelmistoprosessin sateli muutospyyntöjä. Kuitenkin Liun ym. (2016) esimerkki myös kuvaa kuinka DevOps käytänteiden mukaisilla palautejärjestelmillä muutoksen haaste saatiin haltuun.

Kasvavan kompleksisuuden (*engl. growing complexity*) lainalaisuuden huomiointi DevOpsissa näkyy esimerkiksi tiimien vastuun jaon ja automaation käytänteistä. Tiimirakenteet DevOpsin mukaisesti pyritään järjestämään siten, että vastuut jakautuvat tasaisemmin. Tällöin voidaan ajatella järjestelmän kompleksisuuden hallinnan myös helpottuvan. Automaation käytöllä kyetään poistamaan manuaalisten toimintojen virheet. Tällöin automaation hyödyntämisen voidaan nähdä olevan suorassa yhteydessä järjestelmän laatuun. Toisaalta DevOps suosii, että automaatiota sovelletaan mahdollisimman kattavasti koko ohjelmistoprosessissa. Tällöin ohjelmistotestaus tehdään mahdollisuudet huomioiden myös automaattisesti. Testausautomaatiolla voidaan havaita jo aikaisessa vaiheessa, mikäli järjestelmäkokonaisuuden monimutkaisuus aiheuttaa laadun heikkenemistä. Osaltaan myös monitoroinnin ja siihen yhdistettyjen palauteketjujen nähdään huomioivan kasvavaa kompleksisuutta. Esimerkiksi järjestelmän monitoroinnin kautta voidaan havaita tuotantoversion hidastumista, joka puolestaan välittyy tietyn palauteketjun kautta suoraan ohjelmistotiimeille. Näin sekä monitorointi- että palauteratkaisut voivat osaltaan edesauttaa kasvaneen monimutkaistumisen ratkaisua.

Kolmannen itsesäätelevyyden (*engl. self-regulation*) lainalaisuuden todettiin olevan niin sanottu meta-laki, sillä se ei suoranaisesti ota kantaa kehittyvään järjestelmään vaan siihen, kuinka itse ohjelmistoprosessi suhtautuu kehittyvään järjestelmään. Kehitysprosessin itsesäätelevyyttä kuvaa esimerkiksi se, että kehitys ei noudata orjallisesti mitään tiettyä ohjelmistokehitysmenetelmää (Kour & Singh, 2016; Capiluppi ym., 2010). Kuten DevOps toimintamallin käyttöä esittelevissä tutkimuksissa on todettu, ei myöskään DevOpsin käytänteitä noudateta tarkkojen sääntöjen mukaisesti lähes missään organisaatiossa (Lwakatare ym. 2016; Balalaie ym., 2016; Claps ym., 2015). Myös DevOpsin monito-

roinnin ja mittaamisen käytänteet kuvaavat toimintamallin itsesäätelevää piirrettä. Mittaamisen ja monitoroinnin avulla saadaan tietoa ohjelmistoprosessin kyvykkyydestä ja järjestelmän tilasta. Tällöin voidaan ajatella, että DevOpsin mukaisesti ohjelmistoprosessi säätelee itseään monitoroinnin ja mittaamisen kautta opitun perusteella.

Organisatorisen vakauden säilyttäminen (*engl. conservation of organizational stability*) ei suoraan näy DevOpsin mukaisista toimintatavoista. Voidaan kuitenkin ajatella, että DevOpsissa tiimit pyritään järjestämään siten, että ne pystyvät ottamaan vastuulleen tietyn osan koko ohjelmistosta (Balalaie ym., 2016; Lwakatare ym., 2016). Tällöin voidaan organisatorisen vakauden nähdä säilyvän, sillä tiimit ohjautuvat vastaamaan tietystä järjestelmän osasta.

Viides lainalaisuus, tuttuuden säilyttäminen (*engl. conservation of familiarity*) esittää, että evolutiivisiin järjestelmiin tehtävien muutosten tahti vähenee pitkällä aikavälillä. DevOpsin tapauksessa tämän voidaan ajatella pitävän paikkansa, mikäli automaatioratkaisuja käytetään esimerkiksi lähdekoodin generoinnissa. Tällaisessa tapauksessa automatisoitu generointi tekee nopeassa aikataulussa suurimman osan esimerkiksi järjestelmän infrastruktuurikomponenteista ja tällöin järjestelmämuutosten tahti vähenee radikaalisti jo generoinnin jälkeen. Tosin, kuten Lehmankin esitti, vaatimusten muutokset ovat tuskin ikinä täysin ennustettavissa. Täten ei myöskään voida varmasti sanoa kuinka paljon muutoksia automaattisesti generoituun lähdekoodiin täytyy jälkeenpäin tehdä. Näin ollen, automaatio ei välttämättä, tapauksesta riippuen, takaa sitä että, muutosten tahti vähenisi pitkässä juoksussa.

Kuudes jatkuvan kasvun (*engl. continuing growth*) lainalaisuudella kuvataan, että järjestelmän toiminnallisuutta on lisättävä, jotta loppukäyttäjien tyytyväisyys voidaan taata. DevOpsin näkökulmasta jatkuvan integraation ja julkaisun periaatteet osaltaan auttaa hallitsemaan kuudetta lainalaisuutta. Automaation hyödyntäminen mahdollistaa nopeasti tehtävät järjestelmäintegraatiot ja -julkaisut suoraan tuotannolliseen versioon. Jatkuvan julkaisun kautta DevOps toimintamallissa voidaan taata, että loppukäyttäjillä on lähes koko ajan saatavilla laadukkain ja ajantasaisin järjestelmäversio. Toisaalta merkityksellisten ominaisuuksien saattaminen tuotantoversioon vaatii loppukäyttäjien toiveiden seurantaa. Täten myös tehokkaat monitorointi ja palautetekniikat edesauttavat loppukäyttäjien toiveiden huomioisen oikeiden ominaisuuksien integroinnissa.

Ajan kuluessa heikentyvää laatua (*engl. declining quality*) kuvaava seitsemäs lainalaisuus kertoo, että jatkuvien muutosten ja kasvavan kompleksisuuden seurauksena järjestelmillä on tapana muuttua laadultaan heikommiksi. Kirjallisuuden perusteella DevOpsin käytänteiden voidaan tulkita huomioivan heikkenevän laadun. Ensinnäkin erilaisilla järjestelmän monitoroinnin ratkaisuilla voidaan seurata järjestelmää tuotannossa. Virheiden ilmaantuessa tai alustan vakauden järkkyessä virheisiin voidaan puuttua nopeassa aikataulussa. Myös testiautomaation voidaan ajatella ennalta ehkäisevän heikon laadun päätymistä tuotantoon. Lisäksi automaation ratkaisuilla voidaan vähentää käsin tehtyjen vastineiden virhealttiutta.

Lehmanin kahdeksas lainalaisuus, palautejärjestelmä (*engl. feedback system*) kuvaa ennen kaikkea DevOpsin mukaista toimintaa. Monitorointi- ja automaatiokäytänteiden vuoksi ohjelmistoprosessiin virtaa palautetta useasta eri lähteestä. Versionhallintajärjestelmien, testiympäristöjen ja testiautomaatiotulosten sekä tuotantoympäristön tarkkailu tuottavat tärkeää hyödynnettävää informaatiota. Lisäksi loppukäyttäjiltä saatava palaute antaa suuntaa järjestelmän parannus- ja korjaustarpeista. Mainittujen tekijöiden johdosta voidaan todeta, että DevOps toiminta sisältää jatkuvia ja eri tasoisia palauteketjuja usean sidosryhmän välillä, kuten kahdeksas lainalaisuus olettaa.

TAULUKKO 3 DevOps järjestelmäevoluution näkökulmasta

Evoluution lainalaisuus	**Miten DevOps huomioi haasteet**
Jatkuva muutos	- Monitorointi- ja palauteratkaisut huomioivat paremmin muutoksen aiheuttaman paineen (Liu ym., 2014). - Automaatioratkaisut nopeuttavat reagointia muutokseen (Barry ym., 2007, 20).
Kasvava kompleksisuus	- Tiimirakenteet muodostetaan siten, että vastuu järjestelmäkokonaisuudesta jakautuu tasaisesti ja kompleksisuuden hallinta helpottuu. (Balalaie ym., 2016; Lwakatare ym., 2016; Stillwell & Coutinho, 2015) - Automatisoinnilla vähennetään käsin tehtävistä konfiguroinnista aiheutuvaa tahatonta kompleksisuuden kasvua (Callanan & Spillane, 2016). - Testausautomaatiolla havaitaan kompleksisuuden aiheuttamia virheitä jo aikaisessa vaiheessa (Basisiri ym., 2016). Automaatio hidastaa kompleksisuuden muodostumista (Barry ym., 2007, 22).
Prosessin itsesäätelevyys	- Mittaamisen ja monitoroinnin kautta järjestelmästä ja ohjelmistoprosessista kerätään informaatiota, jonka perusteella molempia voidaan optimoida (Lwakatare ym., 2016; Humble & Molesky, 2011).
Organisatorisen vakauden säilyttäminen	- Tiimien yhteistyörakenteet muodostetaan siten, että yksittäinen tiimi ottaa vastuulleen osan järjestelmästä ja ohjelmistoprosessista (Balalaie ym., 2016; Lwakatare ym., 2016). - Automaation käyttö parantaa kehittäjien tuottavuutta (Barry ym., 2007, 26)
Tuttuuden säilyttäminen	- Käytettäessä automaatiota järjestelmän koko kasvaa radikaalisti kehityksen alussa. Pitkällä aikavälillä automaatiolla on stabilisoiva vaikutus järjestelmä suhteelliseen kasvuun (Barry ym., 2007, 24.) - Tiimit ottavat vastuulleen osan ohjelmistotuotteesta (Basiri ym., 2016)
Jatkuva kasvu	- Monitorointi- ja palauteratkaisuilla voidaan seurata vaatimusten ja toivottujen ominaisuuksien kehittymistä käyttäjärajapinnassa (Liu ym., 2014, Lwakatare ym., 2016).

	- Automaation mahdollistava jatkuvalla integraatiolla ja jatkuvalla julkaisulla pidetään huoli siitä, että tuotannollisessa versiossa on sekä ajantasaisin että laadukkain järjestelmäversio (Basiri ym., 2016). - Automaation avulla saadaan nopeammin yhä suurempi määrä vaatimuksia toteutettua. Lisäksi automaation on havaittu lisäävän toiminnallisuuden kasvun hallittavuutta. (Barry ym., 2007, 21-22.)
Heikentyvä laatu	- Monitoroinnin ratkaisuilla voidaan seurata järjestelmän tuotannossa olevan version toimintaa ja sen laatua (Basiri ym., 2016). - Automaatioratkaisuilla vähennetään käsin tehtyjen konfiguroinnin riskialttiutta (Swartout, 2014). - Automaatio lisää ohjelmistovirheiden määrää lyhyellä aikavälillä, mutta laskee niitä radikaalisti pitkässä juoksussa (Barry ym., 2007, 23).
Palautejärjestelmä	- Versionhallintajärjestelmien, testiympäristöjen ja testiautomaatiotulosten sekä tuotantoympäristön tarkkailu tuottaa palautteen kaltaista tietoa (Callanan & Spillane, 2016; Lwakatare ym., 2016). - Lisäksi palautetta saadaan tiimien keskinäisestä toiminnasta sekä loppukäyttäjien suunnasta (Lwakatare ym., 2016, Liu ym., 2014).

Esitetyn perusteella ja lähdekirjallisuuteen vedoten voidaan todeta, että DevOps näennäisesti vastaa järjestelmäevoluution piirteisiin. Kuten kolmannessa luvussa todettiin, on DevOps toimintamallin tavoitteena helpottaa ja nopeuttaa ohjelmistotuotannon toimintaa ja etenkin ohjelmistojulkaisuun liittyviä toimenpiteitä. Toiminnan parantamista voidaan ajatella konkreettisina toimenpiteinä, jotka jollain tavalla optimoivat ohjelmistotuotantoprosessin sujuvuutta sekä tuotettavan ohjelmiston laadun muodostumista. Koska DevOpsissa näennäisesti huomioidaan ohjelmistotuotannon kannalta kriittiset evoluution lainalaisuudet, voidaan DevOpsin toimintamallin hyödyntämisen nähdä auttavan ohjelmistoyrityksiä kehittämään toimintaansa suhteessa alati muuttuvaan maailmaan.

DevOps on kuitenkin esitetyn mukaan, etenkin tieteellisessä kirjallisuudessa, varsin tuore ilmiö. Lisäksi toimintamallin käytänteitä sovelletaan yrityksissä hyvin eri tavoin. Tällöin ei voida olla täysin varmoja siitä, että analysoidusta kirjallisuudesta on onnistuttu löytämään kaikki sellaiset erikoistapaukset, jotka ottaisivat eri tavoin kantaa järjestelmäevoluution lainalaisuuksiin. Kyseinen rajoite ja DevOps aiheisen tieteellisen kirjallisuuden vähäinen määrä huomioiden seuraavassa luvussa esitellään, kuinka DevOpsin ja järjestelmäevoluution välistä yhteyttä voidaan tutkia laadullisen tutkimuksen keinoin. Laadullisen tutkimuksen kautta voidaan ennen kaikkea löytää tarkempia kuvauksia DevOpsin mukaisiin käytänteisiin. Lisäksi sen kautta myös järjestelmäevoluution huomioimista voidaan arvioida huomattavasti tarkemmalla tasolla.

5 HAASTATTELUTUTKIMUKSEN TOTEUTUS

Tässä luvussa käsitellään tutkielman empiirisen aineiston keruuta ja aineiston analysointia. Tarkemmalla tasolla luvussa käsitellään, kuinka haastattelututkimuksen keinoin on haettu vastauksia tutkimuksen ongelmaan ja tutkimuskysymykseen. Seuraavat alaluvut käsittelevät tutkimushaastattelun tavoitteita, haastattelua empiirisenä tutkimusmenetelmänä, sen valintaan johtaneita perusteita sekä haastattelun avulla saatavaa aineistoa ja sen analysointia.

5.1 Empiirisen tutkimuksen tavoite

Ohjelmistot nähdään nyky-yhteiskunnassa kriittisinä organisaatioiden toiminnan näkökulmasta. Tietoyhteiskunnan jatkuvasta muuttumisesta johtuen organisaatioiden on kyettävä yhtä lailla muuntautumaan muutoksen tahdissa. Tällöin muutoksen paine heijastuu yhteiskunnassa käytettäviin järjestelmiin. Koska tietojärjestelmät ovat syvästi integroituneita reaalimaailmaan yhteiskunnan jatkuva muuttuminen sysää painetta yhä enenevissä määrin ohjelmistoja tuottavien organisaatioiden toimintaan. Tällöin ohjelmistoja kehittävien organisaatioiden tulee jatkuvasti muuntaa kehittämiään järjestelmiä, jotta niiden kyky toimia ympäristöissään säilyy. Tässä tutkielmassa järjestelmäevoluutiota on käytetty teoreettisena viitekehyksenä kuvaamaan piirteitä, joita ohjelmisto-organisaatioiden on järjestelmäkehityksessä otettava huomioon, jotta kehitettävien järjestelmien toimintakyky jatkuvasti muuttuvassa maailmassa taataan. Jatkuvasta muutoksesta selvitäkseen ohjelmisto-organisaatiot etsivät jatkuvasti tehokkaampia toimintamalleja. Tässä tutkielmassa DevOps toimintamalli on tuoreudestaan johtuen nähty mahdollisena uutena keinona vastata jatkuvaan muutokseen, sillä se lupaa sujuvoittaa järjestelmien kehittämiseen, julkaisuun ja operointiin liittyviä toimia.

DevOps on kuitenkin niin tieteellisessä kuin teollisessakin yhteiskunnassa varsin tutkimaton ilmiö. Etenkään sen vaikutuksista järjestelmäevoluution hallintaan ei ole tehty tutkimusta. Tämän tutkimuksen tavoitteena onkin selvittää,

voidaanko ilmiöiden välillä havaita yhteyttä. Toisin sanoen voidaanko DevOps toimintamallilla ja siihen liittyvillä käytänteillä huomioida järjestelmäevoluution piirteitä. Tutkimuksen asettelu onkin muodostettu tutkimuskysymykseksi seuraavasti:

- Miten DevOps toimintamallissa huomioidaan järjestelmäevoluutio ja sen erityispiirteet?

5.2 Tutkimusmenetelmän valinta

Edellisissä luvuissa esiteltyä DevOpsia on kirjallisuudessa kuvattu toimintamallina, jonka tavoitteena on nopeuttaa ja sujuvoittaa ohjelmistotuotantoprosessia ja samalla parantaa tuotettavien ohjelmistojen laatua. Kuten on esitetty, toimintamallissa näiden tavoitteiden saavuttamiseen pyritään tiettyjen käytänteiden kautta. Ensinnäkin DevOpsissa korostetaan kehitys- ja operointitoimintojen välistä yhteistyötä. Toisaalta myös automaation, järjestelmän monitoroinnin ja tehokkuuden mittaamisen käytänteillä on havaittu olevan tärkeä rooli. Tutkielman toisessa luvussa esitelty järjestelmäevoluution käsitteistö kuvaa piirteitä ja haasteita, joita järjestelmän kehittymiseen ja sen kehittämiseen liittyy. Tässä tutkielmassa järjestelmän evoluutiota on päädytty tarkastelemaan Lehmanin (1974-1996) järjestelmäevoluution lainalaisuuksien valossa.

Lehman ym. (2000) näkevät, että järjestelmäevoluutiota voidaan tieteellisestä näkökulmasta tutkia kahdella tavalla. Joko tutkimalla ilmiötä itsessään tai tutkimalla sitä, kuinka järjestelmäevoluutio tietyssä kontekstissa tapahtuu. Tutkittaessa järjestelmäevoluutiota itsessään ollaan kiinnostuneita siitä, millainen ilmiö järjestelmäevoluutio on (Lehman, 2000). Tämä on tieteellisessä yhteisössä varsin tutkittu suuntaus ja Lehmanin lainalaisuudet kuvaavatkin juuri tätä näkökulmaa järjestelmäevoluutiosta. Myös suurin osa muista järjestelmäevoluutiotutkimuksista ovat käsitelleet tätä nimenomaista näkökulmaa. Tutkimuksissa ei kuitenkaan varsinaisesti ole tuotettu syvempää ymmärrystä järjestelmäevoluution lainalaisuuksista. Pikemminkin niissä on vahvistettu Lehmanin esittelemiä evoluution piirteitä uusissa ympäristöissä, kuten modernissa ohjelmistokehityksessä. Toisin sanoen suurin osa tutkimuksista on vahvistanut Lehmanin lainalaisuuksien paikkansapitävyyden myös nykyaikaisessa ohjelmistotuotantoympäristössä. Tästä syystä järjestelmäevoluution lainalaisuuksien voitaisiin nähdä pätevän myös DevOps toimintamallin kontekstissa.

DevOpsin tapauksessa on kuitenkin mielekkäämpää tutkia esitetyistä näkökulmista jälkimmäistä: kuinka järjestelmäevoluutio tapahtuu tietyssä kontekstissa. Tämä johtuu kahdesta seikasta. Ensinnäkin tutkittaessa tapaa, jolla järjestelmäevoluutio tapahtuu, on merkityksellistä tutkia juuri ohjelmistotuotantoprosessin aikaansaamaa vaikutusta (Dittrich, 2016, 20; Lehman, 2000). Tämän tutkielman tapauksessa järjestelmäevoluution ilmenemisen tavalla tarkoitetaan sitä, kuinka DevOps toimintamallin mukaiset käytänteet huomioivat järjestelmien evolutiivisia piirteitä ja niiden aiheuttamia haasteita. Toisekseen ja

kaikessa yksinkertaisuudessaan DevOps toimintamallin ja järjestelmäevoluution yhteyttä ei ole tämän hetkisen käsityksen valossa tutkittu lainkaan.

Aiemmissa luvuissa esitellyn kirjallisuuden pohjalta on havaittu, että DevOps on tieteellisellä mittapuulla vielä varsin määrittelemätön ja jokseenkin epäselvä ilmiö. Kun taas toisaalta huomioidaan se, että toimintamallin ja järjestelmäevoluution suhdetta toisiinsa ei ole tutkittu, voidaan todeta laadullisen tutkimuksen olevan oiva menetelmä näiden kahden ilmiön välisen suhteen hahmottamiseen. Tämä johtuu erityisesti siitä, että laadullinen tutkimusote on määrällistä järkevämpi silloin, kun tutkitusta ilmiöstä ei ole aikaisempaa tietoa (Hirsjärvi ym., 2004).

Järjestelmäevoluution ja DevOps toimintamallin yhteyden tutkimiseksi voitaisiin myös Lehmanin lainalaisuuksia pyrkiä vahvistamaan DevOpsin käytänteitä hyödyntävän ohjelmistotuotantoprosessin puitteissa. Lainalaisuuksien ilmentymisen havainnointi vaatii kuitenkin pitkittäisen tutkimusotteen tarkoittaen sitä, että kerättäisiin määrällistä informaatiota tietyn järjestelmän kehittymisen ajalta, eli varsin pitkältä aikaväliltä. Pitkittäistutkimus ei kuitenkaan ole mahdollinen toteutustapa pro gradu -tutkielman suppeuden vuoksi. Määrällinen tutkimus ei muutoinkaan ole yhtä luotettava tutkimusote tuoreen ilmiön tutkimisessa johtuen siitä, että ilmiötä ei ole tutkittu eikä näin ollen sitä kuvaavia muuttujia ja mittareita ole vielä löydetty. Näin ollen tässä tutkielmassa onkin koettu mielekkäämmäksi, tutkia tietyllä ajanhetkellä laadullisin keinoin niitä DevOps-toimintamallin piirteitä, joilla koetaan olevan merkitystä järjestelmäevoluution hallinnan kannalta. Tutkittaessa kyseistä kysymyksen asettelua laadullisen tutkimuksen keinoin voidaan DevOp -toiminnasta löytää järjestelmäevoluution kannalta merkittäviä piirteitä. Lisäksi laadullisen tutkimuksen kautta voidaan löytää mahdollisten jatkotutkimusten kannalta merkittävää tietoa - esimerkiksi syvempää ymmärrystä määrällisten instrumenttien rakentamiseksi.

5.3 Haastattelu aineistonkeruumenetelmänä

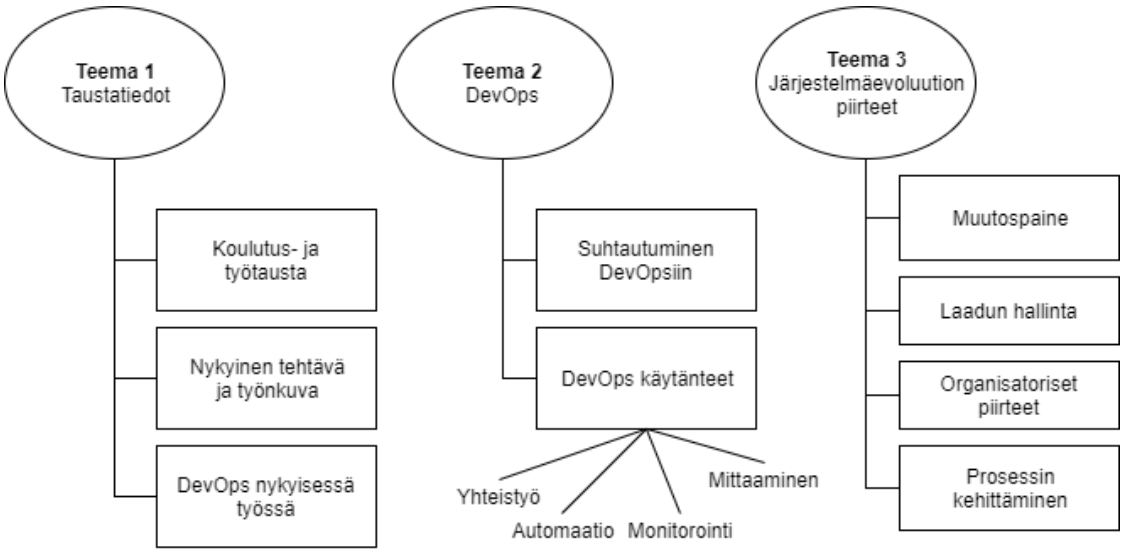
Kuten aiemmissa tutkielman luvuissa kirjallisuuteen perustuen todettiin, on DevOps, erityisesti tieteellisessä yhteisössä, varsin tutkimaton ilmiö (Virmani, 2015). Toisaalta katsaus DevOpsia käsittelevään tieteelliseen kirjallisuuteen paljasti myös, että toimintamallia toteutetaan ohjelmistotuotanto-organisaatioissa hyvin eri tavoilla (Lwakatare ym., 2016). Johtuen DevOps-toimintamallin toteutustapakirjosta, on aineiston kerääminen päätetty tässä tutkielmassa toteuttaa siten, että aineiston muodostumista ei ohjata tahattomasti kirjallisuuden kautta tehtyjen päätelmien mukaiseksi. Tällä tarkoitetaan sitä, että kirjallisuudesta tehdyt päätelmät DevOps toimintamallin ja järjestelmäevoluution lainalaisuuksien yhteyksistä (ks. luku 4) eivät edusta totuutta. Pikemminkin laadullisella aineistonkeruulla pyritään löytämään uusia vastauksia ja mahdollista tukea tehtyihin päätelmiin.

Tässä tutkimuksessa haastattelut ovat valikoituneet aineistonkeruumenetelmäksi ja ne on käyty ohjelmistotuotannon ammattilaisten kanssa. Aineistonkeruumenetelmänä haastattelu on erityisesti tämän tutkielman tapauksessa oiva, sillä sen kautta saadaan nostettua esiin yksilöiden kokemuksia ja näkemyksiä tutkittavasta ilmiöstä (Metsämuuronen, 2006). Tämä on DevOps-toimintamallin tarkoitusperien tunnistamisen kannalta tärkeää, sillä mallia toteutetaan eri organisaatioissa eri tavoin. Lisäksi, koska järjestelmäevoluution ja DevOps toimintamallin yhteyttä ei ole tutkittu, on alan ammattilaisten haastatteleminen järkevä tapa löytää uusia yhteyksiä toimintamallin ja järjestelmäevoluution välillä. Haastattelun on todettu olevan viisas tapa tiedon keruuseen tutkittaessa vähän kartoitettua aihealuetta (Hirsjärvi & Hurme, 2000). Toisaalta eri ohjelmisto-organisaatiota edustavien haastateltavien vastausten vertailun avulla on mahdollista löytää toinen toistaan etevämpiä toimintatapoja järjestelmäevoluution hallintaan.

Yleisesti haastattelun nähdään olevan joustava malli tiedon keruuseen. Haastattelu antaa haastattelijalle mahdollisuuden kontrolloida tiedon keruun hetkeä. Haastattelun jatkuvasta interaktiivisuudesta johtuen haastattelijalla on vapaus tarttua ad hoc -tyyppisesti tutkimuksen kannalta merkittävien vastausten perusteellisempaan läpikäyntiin (Hirsjärvi & Hurme, 2000). Toisaalta haastattelun etuna voidaan pitää sitä, että haastateltavat ovat tavoitettavissa myös varsinaisen haastattelun jälkeen. Tämä on tärkeää etenkin sellaisissa tilanteissa, joissa tiettyjen vastausten analysoiminen vaatii lisäselvitystä. (Hirsjärvi ym, 2004.). Schultze ja Avital (2011, 3) toteavat, että haastattelu aineistonkeruumenetelmänä on tarkoituksenmukaisempi tilanteissa, joissa haastateltavien annetaan vapaasti kertoa kokemuksistaan. Tästä syystä tässä tutkielmassa haastattelun lajiksi on valittu niin kutsuttu teemahaastattelu (Hirsijärvi & Hurme, 2000). Teemahaastattelu on haastattelutilanteen erikoistapaus, jossa keskustelu on pääosin vapaamuotoista. Kuitenkin haastattelu tilannetta ohjaamaan on etukäteen kehitetty tietyt teemat, joista haastateltavien kanssa keskustellaan. Tyypillisesti teemahaastattelun teemat luodaan tutkittavien teorioiden ja ilmiöiden pohjalle.

5.3.1 Haastatteluteemat

Tässä tutkielmassa teemahaastattelun tiedonkeruun instrumentiksi luotu haastattelurunko koostuu kolmesta teemakokonaisuudesta (ks. kuvio 4). Teemat on pääosin luotu suoraan tutkittavien ilmiöiden, DevOpsin ja järjestelmäevoluution pohjalta. Devops ja järjestelmäevoluutio -teemojen lisäksi tiedon keruuseen on yhdistetty osio haastateltavan taustatietoja varten. Koko teemahaastattelun runko on esitettynä liitteessä 1.



KUVIO 7 Haastatteluteemojen rakentuminen

Ensimmäinen haastatteluteema käsittelee haastateltavan taustatietoja. Osiossa käydään läpi haastateltavan koulutus- ja työtaustaa, nykyistä työtehtävää ja työnkuvaa sekä DevOps -toimintamallin ilmentymistä sen hetkisessä työtehtävässä. Haastatteluteeman tarkoitus on kaksiulotteinen. Toisaalta taustatietojen ymmärtämisen kautta voidaan arvioida haastateltavan ammattitaitoa ja kykenevyyttä vastaamaan DevOpsin ja järjestelmäevoluution piirteitä koskeviin kysymyksiin. Kykenevyydellä tässä tapauksessa tarkoitetaan sitä, että haastateltavalla on tarvittavan pitkää ja laaja-alaista kokemusta järjestelmäkehityksestä. Toisaalta taustatietojen keräämisen kautta kyetään myös peilaamaan haastateltavan taustaa hänen esittämiinsä vastauksiin DevOpsin tai järjestelmäevoluution saralla.

Haastattelun toisessa teemassa keskustellaan DevOpsista ja sen ilmenemisestä haastateltavan yrityksen ohjelmistotuotantoprosessissa. Osion tarkoituksena on selvittää, miten tieteellisen kirjallisuuden kautta havaitut DevOps käytänteet ilmenevät haastateltavan yrityksen kehitysprosessissa. Lisäksi DevOps -haastatteluteema pyrkii selvittämään haastateltavan asennetta DevOpsia kohtaan. DevOps käytänteiden selvittämisen kautta voidaan ennen kaikkea ymmärtää paremmin toimintamallia käytännössä sekä vahvistaa lähdekirjallisuudessa toimintamallista oletettuja käsityksiä. Toisaalta haastateltavat voivat myös antaa sellaisia vastauksia DevOps käytänteistä, jotka tuovat uutta tietoa tieteelliseen yhteisöön. Haastateltavan asenteen ymmärtäminen puolestaan auttaa ymmärtämään koetaanko toimintamallin yleistymistä ylipäätään järkevänä ohjelmistotuotannossa.

Järjestelmäevoluution piirteet -teema pureutuu tarkemmin järjestelmäevoluution hallintaan. Osiossa selvitetään, kuinka haastateltavat kokevat yrityksensä ohjelmistotuotantoprosessin ja DevOps toiminnan huomioivan järjestelmäevoluution piirteitä ja edellytyksiä. Edellytykset on teemassa johdettu lähes suoraan Lehmanin (1974-1996) järjestelmäevoluution lainalaisuuksista. Koska

lainalaisuudet ovat kuitenkin lähes yksinomaan tieteellisessä yhteisössä esiintyviä käsitteitä, on ne haastatteluteemassa kategorisoitu neljään helpommin ymmärrettävään kokonaisuuteen. *Muutospaine* -alaosiossa pyritään kysymyksen asetteluilla tulkitsemaan jatkuvan muutoksen (1. laki) ja jatkuvan kasvun (6. laki) huomiointia. *Laadun hallinta* -alaosio puolestaan analysoi kasvavaan monimutkaistumiseen (2. laki) ja heikkenevään laatuun (7. laki) liittyviä keinoja. *Organisatoriset piirteet* -alaosio tulkitsee organisatorisen vakauden (4. laki) ja tuttuuden säilymisen (5. laki) huomiointia. *Prosessin kehittäminen* -alaosiossa puolestaan selvitetään itsesäätelevyyden lain (3. laki) ja palautejärjestelmien (8. laki) ilmentymistä ja merkityksiä.

5.3.2 Haastateltavien valinta

DevOps -toimintamallia ajatellen tutkimuskohteet ja haastateltavat voitaisiin teoriassa valita hyvinkin laajasta skaalasta ohjelmistoalan toimijoita. Laadullista tutkimusta tehtäessä on kuitenkin ensisijaisen tärkeää, että tutkimuksen kohteet on huolellisesti valittu (Hirsjärvi ym., 2004). Tässä tutkielmassa haastateltavien ohjelmistokehitysammattilaisten yritykset on valikoitu ainoastaan suurten toimijoiden joukosta. Tämä siitä syystä, että tällöin valitulla organisaatiolla on pitkä historia ohjelmistojen toteuttamisesta. Lisäksi suuremmalla organisaatiolla on, pieniä yrityksiä todennäköisemmin, kokemusta ja näkemystä suurten ja pitkäikäisten tietojärjestelmien kehittämisestä. Toisaalta taas järjestelmäevoluution aiheuttamaa jatkuvaa kasvua ajatellen myös pienet organisaatiot joutuvat ennen pitkää organisatorista kasvua edellyttävään tilanteeseen. Täten pienet, kasvua hakevat organisaatiot kohtaavat kasvaessaan samoja järjestelmäevoluution haasteita kuin suuremmat organisaatiot. Tästä syystä tutkimuksen tulokset voivat hyödyttää myös pieniä ohjelmisto-organisaatiota. Tähän tutkielmaan haastateltavia etsittiin kolmesta ohjelmistoalan yrityksestä. Yritykset ja niitä edustavat henkilöt on jätetty nimeämättä.

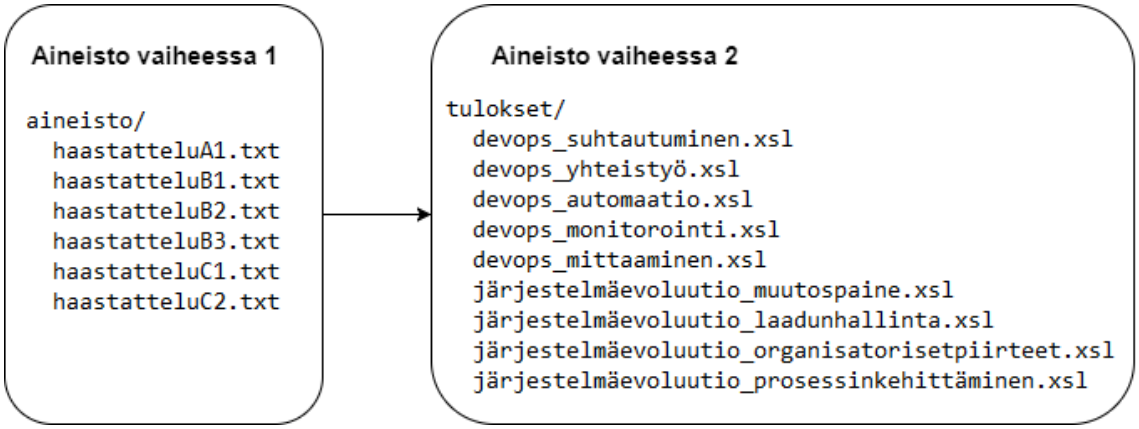
Haastateltavia ohjelmistoammattilaisia kartoitettiin henkilökohtaisten kontaktien kautta. Yhteydenpito aloitettiin kuhunkin organisaatioon lähestymällä joko projektitason tai henkilöstötason johtohenkilöä. Kustakin organisaatiosta löytyi ensimmäisten yhteydenottojen jälkeen varsin nopeasti henkilöt haastateltaviksi. Ensimmäistä haastattelua pidettiin haastattelurunkoa testaavana haastatteluna, tästä syystä ensimmäisen ja toisen haastattelu välille oli varattu aikaa haastattelurungon mahdollisille muokkauksille. Ensimmäisen haastattelun jälkeen muokkaukset nähtiin kuitenkin tarpeettomiksi. Tarkempi tilastointi haastateltavista on nähtävissä 6. luvun ensimmäisessä alaluvussa.

5.3.3 Aineiston analysointi

Aineiston tulkinnan ja analysoinnin helpottamiseksi haastattelut nauhoitettiin sanelulaitteella. Jokaisesta haastattelutilanteesta tallennettiin noin yhden tunnin mittainen nauhoite. Kun kaikki haastattelut nauhoituksineen oli suoritettu asianosaisten kanssa, haastatteluaineiston purkaminen aloitettiin litteroimalla. Ai-

neiston litterointi oli kaksivaiheinen (ks. kuvio 5). Ensimmäisessä vaiheessa jokaisesta haastattelutilanteesta syntynyt nauhoite kuunneltiin läpi ja tallennettiin omaan tekstidokumenttiinsa. Tässä vaiheessa tekstidokumenttien sisällöt noudattivat haastattelun kulun ja haastattelurungon mukaista järjestystä. Kussakin dokumentissa lisäksi korostettiin haastatellun tärkeimpiä havaintoja ja huomioita.

Litteroinnin toinen vaihe käynnistettiin, kun kaikki nauhoitteet oli kirjoitettu puhtaaksi omiin tekstidokumentteihinsa. Toisen vaiheen tarkoituksena oli tarkemmin teemoitella kussakin haastattelussa syntyneet havainnot haastatteluteemojen mukaiseen luokitteluun. Litteroinnin toisen vaiheen aluksi toteutettiin yhdeksän uutta taulukkodokumenttia, joista jokaiseen kirjattiin tutkimustulosten kannalta merkittävimpien haastatteluteemojen (ks. luku 5.2.1. haastatteluteemoista) mukaisia asioita.



KUVIO 8 Haastattelujen litteroinnin eteneminen

DevOps -haastatteluteeman mukaisesti luotiin suhtautumiseen sekä kuhunkin toimintamallin käytänteeseen (yhteistyö, automaatio, monitorointi ja mittaaminen) omat taulukkodokumentit. Järjestelmän evoluutio -haastatteluteeman mukaisesti puolestaan luotiin neljä taulukkodokumenttia teeman alakokonaisuuksia (muutospaine, laadun hallinta, organisatoriset piirteet ja prosessin kehittäminen) varten. Kukin yhdeksästä taulukkodokumentista sisälsi täten usean haastateltavan näkemyksiä tietystä aiheesta.

Kaksiosainen litterointi nähtiin tärkeänä kahdesta syystä. Ensinnäkään haastatellut eivät noudattaneet vastauksissaan toivottua johdonmukaisuutta. Tästä syystä haastattelutilanteen aikana jouduttiin välillä palaamaan johonkin aiempaan kysymykseen tai vaihtoehtoisesti etenemään ohittamalla tiettyjä kysymyksiä. Mahdollisista ohittamisista huolimatta haastattelutilanteissa toki huomioitiin, että kaikkiin toivottuihin kysymyksiin saatiin vastaus. Haastattelutilanteiden satunnaisten epäjohdonmukaisuuksien seurauksena oli siis välttämätöntä ensin saada kokonaiskuva yksittäisestä haastattelusta (litteroinnin ensimmäinen vaihe), jonka jälkeen voitiin paremmin yleistää ja teemoitella haastateltujen vastauksia (litteroinnin toinen vaihe) tutkimuskysymyksen asettelun

kannalta. Toisekseen litteroinnin toisen vaiheen jälkeen, kun haastateltujen vastaukset tiettyä aihekokonaisuutta koskien olivat samoissa dokumenteissa, oli tulosten tulkinta ja pohdinta huomattavasti helpompaa.

6 TUTKIMUSTULOKSET

Tässä luvussa käydään läpi teemahaastatteluissa kerättyä aineistoa ja siitä saatuja tuloksia. Tulokset on edellisen luvun perustelujen mukaisesti luokiteltu kolmeen alalukuun. Ensimmäisessä alaluvussa käydään tarkemmin läpi haastateltavia ja heidän yrityksiään. Toinen alaluku puolestaan pureutuu haastateltavien antamaan kuvaan DevOpsista ja sen käytänteistä. Toisessa alaluvussa myös vertaillaan kirjallisuuden pohjalta annettua käsitystä käytänteistä suhteessa haastateltujen antamiin vastauksiin. Tulosluvun viimeinen alaluku valottaa tutkimuskysymyksen kannalta olennaisinta eli miten haastatellut näkevät DevOps toimintamallin ja sen käytänteiden vaikuttavan järjestelmäevoluution ja sen lainalaisuuksien huomiointiin.

6.1 Yritykset ja haastateltavat

Haastattelujen sopimiseksi otettiin yhteyttä kolmeen ohjelmistoja kehittävään suomalaiseen yritykseen. Yritysten valitsemissa tärkeinä huomioitavina seikkoina pidettiin sitä, että yrityksellä oli pitkä historia ohjelmistokehityksessä. Lisäksi yrityksen tuli ainakin osassa projekteistaan hyödyntää DevOpsin mukaista toimintaa. Valikoiduista yrityksistä kaksi oli suuria yrityksiä, joiden ydintoimintana oli järjestelmäkehitys ja siihen liittyvä konsultaatio. Kolmas valikoiduista yrityksistä ei tuottanut ydintoimintonaan järjestelmiä. Yrityksessä oli kuitenkin sisäinen järjestelmäkehitysyksikkö, jolla on yli 30 vuoden kokemus sekä organisaation sisäisille että ulkoisille sidosryhmille tarjottavasta ohjelmistotuotantotoiminnasta.

Yritysten henkilöstöstä valikoitui haastateltavia yhteensä kuusi kappaletta. Haastateltavat valikoitiin siten, että he edustivat eri ohjelmistotuotannon rooleja. Lisäksi valinnassa pidettiin tärkeänä, että haastateltavalla oli pitkä kokemus alalta ja ohjelmistotuotannosta. Tällöin voitiin varmistaa, että DevOpsista ja sen vaikutuksista saatiin ei ainoastaan kattava, mutta myös asiantunteva kuvaus. Kaikki haastatellut olivat toimineet pitkään ohjelmistokehityksen parissa (vä-

hintään 15 vuotta). Kyseisen havainnon perusteella todettakoon, että kutakin haastateltua voitiin pitää ohjelmistokehityksen ammattilaisena. Täten myös heidän antamiaan lausuntoja muihin kysymyksiin kyettiin pitämään ammattitaitoisina. Yhteenveto haastatelluista ja heidän taustoistaan näkyy alla olevassa taulukossa 4.

TAULUKKO 4 Yhteenveto haastatelluista henkilöistä

Koodi	**Yritys**	**Titteli**	**Kokemus ohjelmistoalalla (nykyisessä organisaatiossa)**
A1	A	Senior Systems Designer	yli 15 vuotta (15 vuotta)
B1	B	Lead enterprise architect	yli 15 vuotta (10 vuotta)
B2	B	Vice President, DevOps and Solution Foresight	~30 vuotta (5 vuotta)
B3	B	Lead Systems Architect	yli 20 vuotta (5 vuotta)
C1	C	Senior Project Manager	~30 vuotta (8 vuotta)
C2	C	Senior Systems Developer	~15 vuotta (10 vuotta)

6.2 DevOps ja sen käytänteet yritysten ohjelmistotuotantoprosessissa

Tutkimushaastattelun DevOpsia käsittelevä teema aloitettiin kysymällä haastateltavilta heidän näkemyksiään ja suhtautumistaan DevOps toimintamalliin. Kysymysten tarkoituksena oli toisaalta selvittää toimintamallin käyttötapaa haastateltavien organisaatioissa ja toisaalta selvittää vastaavatko haastateltavien näkemykset tieteellisen kirjallisuuden antamaa kuvaa toimintamallista ja sen tarkoitusperistä.

Yleisellä tasolla haastateltujen vastaukset määrittelivät DevOpsia toimintamallina samankaltaisesti kuin mitä ilmiöstä on kirjallisuuden perusteella aiemmin tässä tutkielmassa todettu. Kaikki neljä kirjallisuudessa havaittua DevOps käytännettä (yhteistyö, automaatio, mittaaminen ja monitorointi) olivat vastausten perusteella konkreettisia osia toimintamallin aikaansaamisessa. Esimerkiksi haastatellut toivat esiin, että DevOps, nimeensä viitaten, pyrkii tiivistämään kehityksen ja operoinnin yhteistyötä. Usean haastatellun mukaan tiivistämisellä tavoitellaan ensisijaisesti ohjelmistojulkaisun jälkeisten toimintojen jatkuvuutta. Tällä he tarkoittivat sitä, että ohjelmistokehittäjät eivät sysäisi vas-

tuuta järjestelmästä operoinnin huoleksi julkaisun yhteydessä. Tämä on yhteneväinen ajatus kirjallisuuteen verrattuna (esim. Humble & Molesky, 2011; Hütterman, 2012; Lwakatare ym., 2016).

Lisäksi automaation rooli oli kaikkien haastateltujen mukaan yksi avain tekijä DevOpsin mukaisesti toimittaessa. Heidän mukaansa automaatioteknologioilla pyrittiin muun muassa automatisoimaan ja näin sujuvoittamaan ohjelmiston julkaisuprosessia. Tämä näkemys ohjelmiston julkaisuprosessin automatisoinnista (continuous integration, continuous deployment) vastaa sitä käsitystä, jonka muun muassa Humble ja Molesky (2011) sekä Lwakatare ym. (2016) antavat.

Myös kirjallisuudessa tavattu monitoroinnin käytänne näkyi haastateltujen mukaan heidän DevOps toiminnassaan. Monitoroinnin tavat tosin vaihtelivat haastatellusta, yrityksestä ja projektista riippuen. Haastateltujen mukaan tuoreemmissa projekteissa monitorointia oli viety pidemmälle kuin vanhempien järjestelmäprojektien tapauksessa. Kuitenkin jonkinasteista monitorointia esiintyi poikkeuksetta kaikissa heidän kehittämissään järjestelmissä. Kaikkien haastateltujen mukaan monitoroinnilla pyrittiin havaitsemaan mahdollisimman monipuolisesti järjestelmien vikatilanteita. Niin ikään myös tieteellisissä lähteissä esitetty mittaamisen käytänne oli tavalla tai toisella iskostettu haastateltujen DevOps toimintaan. Tähän eivät kuitenkaan muut kuin johtotehtävissä toimivat haastatellut osanneet kattavasti vastata. Annettujen vastausten perusteella kantavana ajatuksena mittaamisessa oli keskittyä ohjelmistoprosessin sujuvuudesta kertoviin mittareihin.

Yleisesti kaikki haastatellut ohjelmistoyritysten asiantuntijat kokivat DevOpsin hyödyllisenä ja erittäin tervetulleena ilmiönä. Toimintamallin he näkivät hyödyttävän järjestelmäkehitystä useasta näkökulmasta. Lähempänä tavallista järjestelmäkehitystehtävää työskentelevät haastatellut (järjestelmäsuunnittelijat ja -arkkitehdit) kokivat DevOpsin käytänteiden parantavan erityisesti ohjelmistoprosessin sujuvuutta kehittäjien näkökulmasta. Heidän mielestään DevOpsia mukailevassa järjestelmäkehitystoiminnassa kehittäjäkokemus on huomattavasti parempaa. Erityisesti kehittäjät olivat tyytyväisiä siihen mitä teknologista parannusta toimintamalli tuo kehitysympäristöön. Toisaalta samaisten haastateltujen mielestä toimintamallin ongelmana on, että yksittäiseltä kehittäjältä vaaditaan huomattavasti suurempaa roolia kuin aikaisemmin. Esimiestehtävissä toimivat haastatellut näkivät DevOpsin ja sen käytänteiden hyödyn näkyvän enemmän myös kustannussäästösyistä. Erityisesti automaation ja hyvin yhteen toimivien tiimien suhdetta kustannussäästöihin pidettiin merkittävänä. Tästä näkökulmasta hyvän esimerkin eräs haastateltu antoi vertaamalla toimintamallia valmistavan teollisuuden tuotantolinjojen automatisointiin ja optimointiin. Seuraavat alaosiot valottavat tarkemmin DevOps toimintamallin neljää käytännettä: yhteistyötä, automaatiota, monitorointia ja mittaamista.

6.2.1 Yhteistyö

Kuten tieteelliseen kirjallisuuden kautta jo aiemmin todettiin, on DevOpsin mukaisen yhteistyön ajatuksena, että ohjelmistotuotannossa tiivistettäisiin eriytyneiden kehitys- ja operointitiimien toimintaa ja näin ratkaistaisiin mahdollisia ohjelmistojulkaisuun liittyviä ongelmia (ks. luku 3.1.). Kuten Balalaien ym. (2016, 7) ja Lwakataren ym. (2016, 6) tutkimuksissa, myös kaikkien haastateltujen organisaatioissa tiimien välistä eriytyneisyyttä vähennettiin uudelleen organisoinnilla. Tuloksena tiimit oli muodostettu monista osaajista koostuviksi. Haastateltujen mukaan moniosaajatiimin tuli organisoinnin tuloksena vähintäänkin sisältää tavallisia kehittäjiä sekä infrastruktuuriosaajia. Eräs haastateltava kiteytti DevOps tiimin tavallisimmin koostuvan noin 5-6 hengestä seuraavasti:

> ”Tyypillisesti siinä on henkilö, joka pitää lankoja käsissä eli ymmärtää businesspuolta ja pystyy keskustelemaan asiakkaan kanssa, kuten Scrummaster. Sitten sieltä tietenkin löytyy muutama sovelluskehittäjä. Aika paljon tänä päivänä halutaan nähdä sellaisia full stack -kehittäjiä. Sitten siellä on se henkilö, joka hoitaa tämän automaatio- ja infrastruktuuripuolen. Suuremmissa kokonaisuuksissa tämä tiimirakenne sitten kertaantuu eli on useita pieniä tiimejä ison tiimin sisällä.”

Haastateltujen mukaan tiimiorganisoinnilla pyritään ohjelmistojulkaisun jälkeisten toimintojen jatkuvuuteen. Toisin sanoen pyritään välttämään ristiriitatilanteita, joita voi syntyä, kun vastuu ohjelmistosta siirtyy organisaatioyksiköltä toiselle. Monista osaajista koostuva tiimi kykenee sekä julkaisemaan itse järjestelmän tai sen osan ja jopa ottamaan vastuun järjestelmän operoinnista ja ylläpidosta julkaisun jälkeen. Uudenlaisen organisoinnin voidaan täten nähdä ratkaisevan aikaisemmin vaiheittaisia ja ketteriä kehitysmalleja leimanneen operointivastuun siirron ongelmia.

Johtotehtävissä toimiva haastateltu B2 totesi, että tärkeää tiimitoiminnan aikaansaamisessa olevan sen, että vastuukulttuurin muutos saadaan läpivietyä. Toisin sanoen, kuinka tiimit rakennetaan järkevästi sekä kuinka kaikki osalliset ymmärtävät toimintamallin muutoksen ja sen vaatiman vastuun. Parhaiten hänen mukaansa muutos on saatu iskostettua, kun tiimit joutuvat suorittamaan julkaisun jälkeisiä operointitoimia itse. Tällöin he joutuvat ottamaan vastuuta ja kaikilla on sisäinen intressi tehdä laadukasta työtä.

Tuotannollisen vastuun lisääntyminen on kuitenkin haastateltujen A1, B1, B2 ja B3 mukaan tuonut myös ongelmia. Erityisesti tämä nähtiin yksittäisen kehittäjän vaatimustason kasvuna. DevOpsin mukaisessa toiminnassa yksittäinen henkilö joutuu ottamaan vastuuta aikaisempaa enemmän, esimerkiksi automaatioon ja infrastruktuuriin liittyvästä kehittämisestä. Tämä tarkoittaa sitä, että tehokkaan ohjelmistotiimin muodostaminen on muuttunut entistäkin hankalammaksi. Ydintoimintonaan ohjelmistoja tuottavien yritysten haastatellut B1, B2 ja B3 kuitenkin totesivat, että ongelmaan on jo valveuduttu järjestämällä yhteisöllisiä oppimistapahtumia ja verkkoyhteisöjä. Tämä osaltaan kertoo myös

siitä, että toimintamallin mukainen yhteistyö ei näy ainoastaan tiimin sisäisessä toiminnassa, vaan myös halki organisaation.

6.2.2 Automaatio

Kaikki haastatellut olivat yhtä mieltä siitä, että konkreettisin uusi asia, jonka DevOps tuo ohjelmistotuotantomaailmaan, on automaatio. Heidän mukaansa automaatio näkyy toimintamallin yleistymisen myötä ohjelmistoprosessissa kahdella tavalla. Toisaalta järjestelmän infrastruktuurin rakentaminen on automatisoitu, toisaalta taas automaation avulla on saatu aikaan vaivaton ja putkimainen ohjelmistojulkaisu.

Järjestelmäinfrastruktuurin rakentamista ja hallintaa tehtiin haastateltujen yrityksissä kahdella tapaa. Toisissa projekteissa infrastruktuuri oli tarjottu kolmannen osapuolen alustana, toisissa taas yrityksellä oli sisäinen toiminto tai oma organisaatioyksikkö, joka vastasi infrastruktuurin tuottamisesta. Yhteistä molemmille tavoille on, että tarjottua alustaa voidaan hallita lähdekoodin avulla (engl. *infrastructure-as-code, platform-as-code*). Haastateltu A1 kuvasi uutta infrastruktuurin hallinnan tapaa seuraavasti:

> ”Mielestäni tämä (DevOps) on tullut käsi kädessä pilvipalveluiden kanssa. Pilvipalvelut ovat tuoneet infrastruktuurin hallittavaksi lähdekoodin avulla. Nykyään (kehittäjille) infra ei varsinaisesti ole niitä fyysisiä tietokoneita ja palvelimia. Näiden asentamisen automatisointihan on lähes mahdotonta. Pilvipalveluiden yleistyminen on ruokkinut infran hallinnan ja automatisoinnin työkalujen ilmestymistä. Nykyään siis riittää, että komennetaan lähdekoodilla pilvialustaa, kun halutaan vaikka kymmenen uutta palvelinta.”

Usea haastateltu totesi, että automatisoidun infrastruktuurin rakentamisen lisäksi esimerkiksi käyttöjärjestelmä- ja tietokanta-asennukset voidaan toteuttaa lähes napin painalluksella tiettyjä työkaluja käyttäen (mm. Ansible). Tällainen automatisointi tekee tarpeettomaksi esimerkiksi ennen manuaaliseksi luetun käyttöjärjestelmäasennuksen tai konfiguroinnin. Haastateltujen mukaan DevOps toimintamalli mahdollistaa sen, että tavallinen ohjelmistokehittäjä ottaa vastuun myös järjestelmän infrastruktuurin rakentamisesta ja hallinnasta. Tästä voidaan huomata, että DevOps työkalujen käytöllä voidaan rikkoa vanhat siiloutuneet organisaatiorakenteet. Nyt yksittäinen tiimi ottaa vastuulleen sekä järjestelmän että sen infrastruktuurin kehittämisen.

Automaatio oli haastateltujen mukaan DevOps toiminnassa levinnyt myös varsinaisen sovelluksen julkaisuprosessiin. Automatisoitu julkaisu oli kaikkien haastateltujen mukaan lähestulkoon identtinen, kuin miten se oli tieteellisissä lähteissä esitetty (ks. kuvio 3). Sovelluskehittäjä vie lähdekoodin versionhallintaan, jossa ajetaan ns. jatkuva integraatio (engl. *continuous integration*) eli paketoidaan kaikesta versionhallinnan lähdekoodista uusi sovellusversio, joka siirtyy testiympäristöön. Testiympäristössä ajetaan testiautomaatio, jonka onnistuessa suoritetaan automaattinen julkaisu (engl. *continuous deployment*) suoraan

loppukäyttäjien saatavile. Haastateltujen mukaan automaattista julkaisua ei kaikissa sovellusprojekteissa kuitenkaan toteutettu. Haastateltujen C1 ja C2 mukaan esimerkiksi monitoimittajaympäristössä, joissa koostetaan isompi järjestelmä usean toimittajan kehittämistä järjestelmistä, ei automaattista julkaisua käytetty. Tämä johtui heidän mukaansa siitä, että tällaisen järjestelmäympäristön julkaisu vaatii yhtäaikaisesti usean järjestelmän muuttumisen.

Vastausten perusteella automatisoinnin tavat haastateltujen ohjelmistoprosesseissa olivat samankaltaisia kuin miten ne on kuvattu useissa tieteellisissä lähteissä. Niin ikään automatisoinnista saatavat hyödyt olivat tulkittavissa aikaisemmissa tieteellisissä tutkimuksissa ilmenneiden kaltaisiksi. Tärkeimpänä automaation käytön hyötynä oli havaittavissa sen mahdollistama nopeutus ohjelmistokehitykseen. Kun aikaisemmin työläät ja monimutkaiset alustan rakentamisen ja ohjelmistojulkaisuun liittyvät toimenpiteet korvataan omalla automatisoiduilla prosesseillaan, on selvää, että ohjelmisto saadaan julkaistua nopeammin loppukäyttäjien saataville. Haastateltujen vastausten perusteella automatisointi ei ainoastaan nopeuta julkaisuun kuluvaa aikaa, mutta se myös vähentää julkaisuun tarvittavaa työmäärää. Tällöin usean haastatellun mukaan kehittäjäresurssit voidaan hyödyntää varsinaiseen kehitystyöhön ja koko organisaation tuottavuus paranee. Lisäksi automaation korvatessa aikaisemmin manuaalisesti tehtyjä toimia ohjelmiston laatu paranee manuaalisten virheiden välttämisen seurauksena.

Osa, etenkin johtotehtävissä toimivista, haastatelluista (B2, B3, C1) näki automaation mukanaan tuoman operatiivisten kustannusten tippumisen yhtenä ydinhyödyistä. Kustannussäästöt heidän mukaansa tulivat etenkin infrastruktuuri- ja julkaisuautomaation mahdollistamasta aikaedusta sekä työntekijöiden tuottavuuden lisääntymisestä. Kirjallisuudessa havaittujen hyötyjen lisäksi lähempänä tavallista kehitystyötä olevat haastatellut A1, B1 ja C2 kokivat automatisointityökalujen parantavan ns. kehittäjäkokemusta tarkoittaen, että kehittäjät kykenevät paremmin keskittymään ydintehtäväänsä välittämättä esimerkiksi aikaisemmin manuaaliseen julkaisuun liittyvistä toimenpiteistä. Tätä kautta myös kehittäjän parempi tuottavuus konkretisoituu.

DevOps uutena toimintamallina ei haastateltujen mielestä kuitenkaan ollut juurisyy siihen, miksi automaatiota nykypäivän ohjelmistotuotannossa käytetään niin paljon. Haastateltujen A1, B3 ja C2 mielestä automaattinen jatkuva integraatio on ollut varsin yleistä myös ennen DevOps-aaltoa. Sen sijaan heidän mielestään automaattinen julkaisu (engl. *continuous deployment*) on puolestaan nostanut päätään DevOpsin myötä. Lisäksi haastateltujen A1 ja C1 mukaan ainakin heidän prosessissaan testausautomaatio on tullut merkittävämmäksi osaksi. Tähän he arvelivat syyksi sen, että automaatiotyökalujen myötä testaaminen, etenkin testiautomaatio, on tullut helpommaksi.

6.2.3 Monitorointi

DevOps käytänteistä myös monitorointi toistui haastateltujen organisaatioissa hyvin samankaltaisesti. Tieteellisten lähteiden (esim. Lwakatare ym., 2016;

Hütterman, 2012) mukaan järjestelmän monitoroinnin tavoitteena on tarjota ohjelmistoprosessiin tietämyksen lisäystä erilaisten automaattiraporttien ja -palautteiden kautta. Lisäksi monitoroinnin nähdään raportointiin perustuen parantavan kehitettävän järjestelmän laatua. Haastatellut ohjelmistoyritysten edustajat kuvasivat monitoroinnin käytänteitään hyvin samantapaisesti.

Kaikkien haastateltujen mukaan järjestelmän monitoroinnilla pyritään ensisijaisesti havaitsemaan järjestelmän virheitä niin kehityksen, julkaisun kuin tuotannonkin eri vaiheissa. He kokivat myös monitoroinnin auttavan järjestelmävirheistä toipumista proaktiivisesti, sillä tiettyjen monitorien avulla kyetään havaitsemaan ongelmallisuuksia jo ennen niiden tapahtumista. Monitoroinnin kuvattiin tarkkailevan järjestelmää eri tasoilla. Toisaalta se näkyi fyysisten laitteiden tasolla esimerkiksi palvelimissa ja niiden levyjen käytössä sekä niihin yhdistetyissä hälytyksissä. Toisaalta monitorointia oli liitetty myös esimerkiksi tietokantojen, prosessien sekä muistin käyttöön. Monitorointia tehtiin myös sovellustasolla lokipohjaisesti tarkoittaen, että sovellukset kirjoittivat lokia, jonka avulla voitiin jäljittää sovellusten tekemiä kutsuja.

Eri tason monitoroinnin antamat hälytyksen ja palautteet lähetettiin kaikkien haastateltujen mukaan helposti kehitystiimin saataville. Etenkin lähempänä varsinaisia kehitystehtäviä toimivat haastatellut (A1, B1 ja C3) korostivat monitoroinnin merkitystä palautekanavana ja laadun takaamisen keinona. A1 muun muassa kuvasi, että heillä on käytössään erityinen näyttöpääte, jossa voidaan näyttää tilastotietoa esimerkiksi palvelinprosessien käynnissä olosta, sovellusversioiden tilasta tai sovellusjulkaisujen onnistumisesta. Tällaisia ns. dashboard-sovelluksia oli käytössä kunkin haastatellun organisaatiossa, usein web-käyttöliittymän kautta. Työkalujen avulla (mm. SonarQube) saatiin lisäksi hyvinkin yksityiskohtaista tietoa jopa kirjoitetun lähdekoodin laadusta. Niiden avulla ei ainoastaan saatu tietoa koodin virheistä, mutta kyseinen monitorointisovellus osasi myös ehdottaa kehittäjälle parempia käytänteitä lähdekoodin kirjoittamiseen. Haastatellun B1 mukaan analysointityökalu mahdollisti myös historianäkymiä, joiden kautta nähtiin kuinka tietyn sovellusmoduulin virheiden eli bugien määrä oli ajanjuoksussa kehittynyt.

Tieteellisen lähdekirjallisuuteen lisänä haastatellut toivat esiin monitoroinnin kytkennän järjestelmän liiketoiminnallisiin vaatimuksiin. Yrityksen B edustajien (B1, B2 ja B3) mukaan monitorointia ja siitä saatavaa palautetta oli joissain projekteissa laajennettu myös asiakkaan liiketoiminnan käyttöön. Haastateltu B2 esimerkiksi havainnollisti asiaa esimerkillä:

> ”Me voidaan tarkkailla vaikuttaako sovelluksen käyttöliittymän muutos järjestelmän käyttäjien tapaan käyttää järjestelmää. Tämän kautta saadaan sitten syötettä jatkokehitykseen.”

Yleisesti haastatellut olivat samaa mieltä siitä, että monitorointi ei ole DevOps-aallon mukana tullut käytänne. A1 totesi, että järjestelmän monitorointia on käytetty jo pitkään ennen DevOps toimintaa. Hänen mielestään toimintamallin mukana yleistyneet työkalut kuitenkin tekevät monitoroinnin aikaansaamisesta ja käyttämisestä monta kertaa helpompaa aikaisempaan kehitysympäristöön

verrattuna. B1 puolestaan huomautti, että vaikka DevOps ei hänen mielestään tuokaan monitorointiin mitään uutta, on toimintamallin yleistymisen myötä yleinen kiinnostus järjestelmän monitorointiin kasvanut. B3 mukaan monitorointi onkin DevOpsin mukana tulleessa automatisoidussa kehitysympäristössä lähes välttämättömyys, jotta kehitystiimit pysyvät mukana muutosten tahdissa. B3 mukaan myös dashboard-sovellukset, joilla visualisoidaan monitoroinnin kautta saatavia tilastoja ovat hänen mukaansa yleistyneet. Liiketoiminnan näkökulmasta toimintamallia tarkastellut B2 puolestaan oli sitä mieltä, että edellä mainittu monitorien kytkeminen liiketoiminnan vaatimuksiin on yleistynyt huomattavasti DevOpsin myötä.

6.2.4 Mittaaminen

Tieteellisen kirjallisuuden mukaan järjestelmän monitoroinnin kautta saatavaa tilastotietoa ja palautetta tulisi DevOps toiminnassa hyödyntää myös yleisesti ohjelmistotuotantotoiminnan kehittämiseen. Tätä kutsutaan toimintamallin mittaamisen käytänteeksi. DevOps aiheisessa lähdekirjallisuudessa korostettu mittaaminen ja sen politiikat olivat haastateltujen keskuudessa aiheista hankalimpia. Etenkään lähempänä tavallista kehitystyötä toimineet eivät osanneet antaa vastauksia mittaamista käsitteleviin kysymyksiin.

Haastatelluista B1 ja C1 totesivat DevOpsissa järjestelmäkehitystoiminnan mittaamisen sisältävän samoja elementtejä kuin aiemmissa vesiputoustyyppisissä ja ketterissä ohjelmistoprojekteissa. B1 mukaan vesiputousmallista tutut liiketoimintamerkitysanalyysit (ns. function point analyysit) ja ketterien mallien tiimitoimintaan orientoitunut mittaaminen oli edelleen käytössä heidän projekteissaan. Myös C1 oli samaa mieltä siitä, että suurin osa mittaamisesta pohjautuu ketterien menetelmien myötä yleistyneisiin mittauskäytänteisiin.

C1, B2 ja B3 osasivat kuitenkin avata tarkemmin DevOpsin myötä tullutta muutosta mittauspolitiikkaan. Heidän mukaansa toimintamallin hyödyntämisen seurauksena on pyritty enemmän mittaamaan tuotantoon viennin tehokkuutta. He näkivät ohjelmistojulkaisutahdin ja ajan, joka kuluu asiakaskäyttöönottoon tärkeimmiksi mittareiksi. B2 mukaan myös loppukäyttäjien kokemuksia mitattiin arvioimalla uusien ohjelmistojulkaisujen käyttöönoton kattavuutta. Lisäksi järjestelmien laadun mittaaminen koettiin tärkeänä. Laatua mitattiin muun muassa koodianalysaattorien raportoiman virhe- eli bugihistorian ja ongelmatilanteiden toipumisaikojen perusteella. Kiteytettynä siis DevOpsin myötä näkyvä uudenlainen mittaamisen tapa keskittyi tavanomaisimmin laatuun eli virhehistoriaan ja virheistä toipumiseen sekä siihen kuinka nopeasti jatkuvan integraation ja julkaisun prosessit on saatu toimimaan.

Mittaaminen on DevOpsissa muuttunut yhä enemmän projekti- ja järjestelmäkohtaiseksi. Esimerkiksi haastateltu B3 totesi, että mittaamista räätälöidään kullekin järjestelmälle, sillä kullakin on oma liiketaloudellinen merkityksensä. DevOpsin myötä mittaamisen säätäminen on helpottunut monitorointityökalujen kehittymisen kautta. Hänen mukaansa järjestelmäprojektien alkutaipaleella projekteille määritetään tärkeimmät mitattavat asiat ja näiden pohjal-

ta rakennetaan aiemmin mainittuja dashboard -sovelluksia kehittäjien ja jopa tilaajaorganisaation käyttöön. Tällöin järjestelmän ja projektin tilat ja kehitys ovat nähtävillä niin kehitystiimeille, johdolle kuin tilaajaorganisaatiollekin.

B2 mukaan mittaamisen tulee tiiviistä tiimeistä ja laajasta automaation käytöstä johtuen olla neutraalia, jotta todelliset tehokkuuden hidasteet havaitaan. Haastatelluista B2 oli erityisen kiinnostunut DevOpsin mukana tulleesta mittaamisen mahdollisuudesta ja kokikin sen yhtenä tärkeimmistä toimintamallin ulosanneista:

> ”Ennen asiakkaan näkökulmasta mitattiin kehityksen työmäärää ja sen hintavaikutusta. Nyt ollaan menossa enemmän siihen suuntaan, että mikä se ulosanti on. Eli organisaation (järjestelmän toteuttajan) tulovirta alkaa olla kiinni siitä, kuinka asiakas pystyy tekemään heille toteutetulla järjestelmällä rahallista tulosta. Mittaaminen on erityisesti nostanut päätään DevOpsin tulon myötä. DevOps on vähän niin kuin tuotantolinjan automatisointi tehtaassa, ja jotta sitä tuotantoputkea saadaan säädettyä, niin asioita täytyy mitata.”

6.3 DevOps järjestelmäevoluution hallinnassa

Luvussa 5 kuvattiin, kuinka haastatteluteemat rakennettiin Lehmanin järjestelmäevoluution lainalaisuuksia perustana käyttäen. Seuraavissa alaosioissa käsitellään haastateltujen antamia vastauksia tutkimushaastattelun viimeiseen, tietojärjestelmien evoluution haastatteluteemaan. Teemassa kysymykset oli aseteltu siten, että haastatellut esittivät näkemyksiään DevOps toimintamallin ja sen käytänteiden heijastumisesta järjestelmäevoluution lainalaisuuksiin. Tutkimuskysymys huomioiden, tulosten kannalta merkittävänä pidettiin sitä, kuinka DevOps ja sen käytänteet huomioivat Lehmanin esittämiä järjestelmäevoluution erityispiirteitä.

6.3.1 Muutospaine

Haastateltujen keskuudessa Lehmanin ensimmäiseen ja kuudenteen lainalaisuuteen perustuva muutospaine -kokonaisuus nähtiin kahdesta näkökulmasta. Toisaalta painetta järjestelmämuutoksiin aiheuttivat sisäiset tekijät eli loppukäyttäjiltä tai yleisimmin tilaajalta tulevat vaatimusmuutokset. Toisaalta taas ulkoisten tekijöiden muutoksen, kuten kehitysteknologioiden jatkuvan muutoksen, nähtiin aiheuttavan painetta järjestelmien kehittämiseen. Haastateltujen mukaan DevOps toimintamallin hyödyntäminen helpotti reagointia edellä mainittujen sisäisten tekijöiden aiheuttamaan paineeseen, mutta toisaalta toimintamallin teknologiariippuvaisuuden nähtiin hankaloittavan ulkoisten tekijöiden luomaa painetta.

Tieteellisen lähdekirjallisuuden mukaan automaation todettiin nopeuttavan reagointia vaatimusmuutoksiin (mm. Barry ym., 2007). DevOps toimintamallin tapauksessa useat haastatellut kokivat automaatiolla olevan samankal-

taisia vaikutuksia erityisesti sisäisten tekijöiden aiheuttamaan muutokseen. Yleisellä tasolla haastateltujen keskuudessa koettiin julkaisuautomaation vaikuttavan positiivisesti reagointinopeuteen. Automaation nopeutuksen seurauksena myös palautteen saaminen nopeutuu ja näin ollen organisaatioilla on kykyä vastata muutoksiin nopeammin. Edellä mainittu kävi ilmi haastateltujen A1, B1, C1, C2 ja B3 antamista vastauksista. Heidän mukaansa julkaisuautomaatiolla (jatkuva integraatio ja jatkuva julkaisu) on ollut merkittävä vaikutus eritoten nopeuteen, joilla muutoksia voidaan viedä tuotantoon. A1 huomautti myös, että automaation mahdollistamat nopeat muutokset myös ruokkivat entistä nopeammin uusia muutostarpeita:

> "Kun meillä on nyt kykyä muuttaa tiettyä järjestelmää nopeammin, niin se (järjestelmä) alkaa imemään muutoksia myös nopeammin. Eli, mitä nopeammin sitä kyetään muuttamaan, niin sen enemmän sitä ihan oikeasti myös muutetaan. Tällöin se järjestelmän skouppi (laajuus) alkaa myös kasvamaan."

B3 mukaan julkaisuputken automatisoinnin seurauksena muutosten tekemistä ei myöskään tarvitse entiseen tapaan vältellä, koska testiautomaatio on integroituna julkaisuprosessiin. Tällöin hänen mukaansa voidaan pitää paremmin huolta, etteivät heikkolaatuiset muutokset ikinä päädy tuotantoon asti ja aiheuta tällöin ylimääräistä virhetilanteiden selvittelyä. B2 mukaan automaattisissa julkaisuissa tehtävät muutokset ovat kooltaan verrattain pienempiä kuin ennen. Julkaisukoon hän näki edesauttavan nopeutta, jolla muutoksiin voidaan reagoida, sillä pienemmät julkaisut ovat helpommin hallittavissa eivätkä näin ollen aiheuta ylimääräistä tukea ja ylläpitoa. Erityisen tärkeänä B2 koki julkaisukoon pienuuden korkean turvallisuuden järjestelmissä (esim. pankkijärjestelmät).

Tutkitun lähdekirjallisuuden valossa myös erilaisten monitorointi- ja palauteratkaisujen käytön nähtiin parantavan kehittäjäorganisaatioiden kykyä reagoida muutokseen (Liu ym., 2014, Lwakatare ym., 2016). Myös valtaosa haastatelluista näki asian samoin. Tärkeimpänä muun muassa B1 ja C2 totesivat, että hyvin toteutetun monitoroinnin myötä voidaan esimerkiksi havaita järjestelmän sisäisiä ongelmatilanteita proaktiivisesti eli ennen varsinaisten käyttöongelmien ilmaantumista. Tällaista informaatiota saatiin heidän mukaansa esimerkiksi palvelimien kuormituksen muutoksista ja laitteiden suoritintehon muutoksista. B2 näki monitoroinnin välttämättömänä vahvasti automaatioon nojaavassa ohjelmistoprosessissa. Hänen mielestään automatisoinnin aikaansaama muutostahti on niin hurja, että monitorointi on ainoa keino, jolla kehittäjät voivat ylipäätään pysyä mukana muutosten tahdissa. Toisin sanoen vain tehokas monitorointi ja siihen kytketty palautejärjestelmä (esim. dashboard-sovellukset) voivat kertoa rikkooko jokin muutos jotain. Myös B3 korosti monitoroinnin merkitystä muutosten hallinnassa. Hän oli projekteissaan törmännyt tilanteisiin, joissa loppukäyttäjien käyttäytymismalleihin yhdistetyllä monitoroinnilla kyettiin saamaan syötettä järjestelmä jatkokehityksen ja ylläpidon tarpeisiin.

Lähdekirjallisuudesta tulkituista näkemyksistä poiketen haastatelluista valtaosa näki lisäksi uudenlaisen tiimirakenteen hyödyttävän muutospaineiden

hallintaa. DevOpsin mukaisen moniosaajatiimin avulla reagoinnin nähtiin paranevan useasta syystä. B2 mukaan uudenlainen tiimirakenne parantaa reagointikykyä, koska tiimien osaaminen on aikaisempaa poikkileikkaavaa. Tällöin hänen mukaansa asioita kyetään jo ennalta ottamaan paremmin huomioon. C1 näki asian samoin. Hänen totesi muun muassa, että uusien kehitysprojektien aloittaminen saadaan aloitettua nopeammin, sillä laajan osaamisen myötä uusiin, esimerkiksi kehitysteknologisiin asioihin perehtyminen ei vie enää niin paljon aikaa. C1 mukaan reagointikykyä parantaa myös se, että ohjelmistotiimit joutuvat operoimaan tuotostaan tuotannossa jonkin aikaa. Tällöin kehityksestä ja operoinnista vastaavat täsmälleen samat henkilöt ja muutosten havaitsemisen, viestimisen ja niiden toteuttamisen välillä ei ole ylimääräisiä eikä hidastavia välikäsiä. Lähellä tavallista kehitystyötä toimiva C2 totesi, että nykyinen kehitystä ja operointia yhdistävä tiimitoiminta nopeuttaa huomattavasti reagointia. Hänen mukaansa muun muassa se, että nyt tavallisilla kehittäjillä on näkyvyys infrastruktuuriasioihin helpottaa virheiden selvittämistä ja niihin puuttumista. B2 näki asiaa samoin. Hänen mukaansa se, että nyt, kun infrastruktuuri on myös lähdekoodia, sitä voidaan säilyttää järjestelmän toiminnasta vastaavan lähdekoodin kanssa versionhallintajärjestelmässä osana samaa projektia. Tällöin infrastruktuurimuutosten tekeminen on ketterämpää aikaisempaan toimintatapaan verrattuna.

Tieteellisistä lähteistä poiketen osa haastatelluista totesi DevOps toimintamallin mukanaan tuomien työkalujen aiheuttavan haasteita järjestelmäkehitykseen. Tähän viitattiin edellä ulkoisten tekijöiden aiheuttamana muutospaineena. Teknologiariippuvaisuuden aiheuttamaa muutospainetta ei kuitenkaan koettu ongelmaksi kaikkien haastateltujen keskuudessa. A1 mukaan DevOpsin myötä ulkoisten tekijöiden, kuten sovellusalustojen ja sovelluskirjastojen määrä on noussut hyvin suureksi ja tekniikkaa kehittyy koko ajan kiihtyvällä tahdilla. Hänen mukaansa tämä ei ainoastaan aiheuta painetta toimitettavien järjestelmien muutoksille, vaan yhä enenevissä määrin myös itse kehitysprosessissa käytettäville työkaluille. Esimerkiksi kehittäjät joutuvat välillä muuttamaan kehitysprosessissa käytettyjä työkaluja, jotta mitään ei mene rikki. Kuitenkin valtaosa haastatelluista koki teknologian kehittymisen ainoastaan teoreettisesti haasteellisena. Esimerkiksi B3 ja C2 totesivat, että ulkoista muutospainetta ei ole varsinaisesti havaittu missään projektissa. He kuitenkin totesivat, että asia on tiedostettu heidän organisaatioissaan. B2 sen sijaan koki haasteen merkittävänä. Erityisesti siitä syystä, että toimittajia (esim. sovellusalustan toimittajia) on nykyään niin valtavasti. DevOpsin myötä siis toimittajiin lukkiutumisen umpikujaa ei kannattane yrittää välttää ainoastaan toimitettavien järjestelmien kohdalla, vaan sitä tulee välttää myös kehityksessä käytettävien työkalujen kohdalla. B3 totesikin, että nykyään toimittajaekosysteemin valinta on entistäkin hankalampaa.

Eräiden haastateltujen mukaan myös järjestelmäarkkitehtuurin merkitys korostuu entisestään automaatioon nojaavassa kehitysputkessa. B1 ja C2 mukaan järjestelmäkehityksessä kyetään parempaan muutosreagointiin, kun järjestelmän arkkitehtuuri on sallivampi. Niin sanottu mikropalveluarkkitehtuuri

onkin yleistynyt DevOpsin tulemisen myötä (Balalaie ym., 2016). B1 totesikin arkkitehtuurin ja muutosreagoinnin yhteydestä seuraavasti:

> ”Yleensä muutospaine havaitaan vasta pitkässä juoksussa. Eli liiketoiminnallisen vaatimusmuutoksen seurauksena tarvitaan uusi työntekijä, joka ei kuitenkaan pääse tuottavaan työhön. Tämä johtuu usein siitä, että järjestelmän arkkitehtuuri on jähmeä muutosten toteuttamiselle. Sen takia me suositaankin mikropalveluarkkitehtuuria kaikissa uusissa järjestelmätoteutuksissa.”

6.3.2 Laadunhallinta

Laadun hallinnan haastattelukokonaisuus koostettiin Lehmanin toisesta (Kasvava kompleksisuus) ja seitsemännestä (Heikkenevä laatu) lainalaisuudesta. Tieteellisessä lähdemateriaalissa järjestelmän laadun hallinnan keinot nähtiin kolmesta näkökulmasta. Ensinnäkin uudenlaisen yhteistyö- ja tiimimallin nähtiin edesauttavan laadun takaamista (Balalaie ym., 2016; Lwakatare ym., 2016; Stillwell & Coutinho, 2015), toisaalta taas automaatio koettiin laadun hallinnassa merkittäväksi (mm. Basiri ym., 2016; Swartout, 2014). Lisäksi monitorointi ja siihen kytketty palautejärjestelmä nähtiin tärkeänä laadun takaamisen menetelmänä (mm. Basiri ym., 2016). Haastateltujen ammattilaisten mukaan samat näkökulmat olivat nähtävissä myös heidän ohjelmistotuotantoprosesseissaan.

DevOps toimintamallin mukaisen yhteistyö- ja tiimirakenteen vaikutus järjestelmien laadun hallintaan nähtiin haastateltujen keskuudessa paranevan kahdesta syystä. B1 mukaan laadun kehittymisen ja sen hallinnan kannalta oli tärkeää, että vastuu järjestelmän kokonaisvaltaisesta kehittämisestä on siirtynyt enemmän kehitystiimien käsiin. Se, että tavallinen järjestelmäkehittäjä pystyy itse kehittämään ja ylläpitämään järjestelmäinfrastruktuuria tehostaa laadunhallintaa. Tällöin ensinnäkin kehitystiimit pääsevät nopeammin kiinni mahdollisiin alustavirheisiin ilman, että heidän täytyisi konsultoida erillistä alustavastaavaa asiasta. Toisekseen kehittäjät voivat itse ottaa vetovastuun alustakehityksestä sen sijaan, että he syyttäisivät alustan tuottanutta yksikköä huonolaatuisesta infrastruktuurista.

Lisäksi laadun hallinnan näkökulmasta tärkeänä nähtiin, että uuden tiimimallin seurauksena ohjelmistotiimillä on enemmän poikkileikkaavaa osaamista. Erityisesti haastatellut B2, C1 ja C3 korostivat laajan osaamisen merkitystä. B2 ja C1 mukaan moniosaajatiimi kykenee ymmärtämään järjestelmän teknistä laatua entistä kattavammin sekä toiminnallisesta että infrastruktuurin näkökulmasta. Lisäksi tavallisimmin tiimistä löytyy myös osaamista liiketoiminnallisen ja asiakkaan kontekstin ymmärtämisestä. B2 mukaan myös se, että eri alueiden spesialistit toimivat fyysisesti tiiviimmässä yhteistyössä edesauttaa reagointia mahdollisiin laatua heikentäviin tekijöihin. Hänen mukaansa samassa tilassa työskentelevät infrastruktuuri- ja kehitysosaajat pystyvät nopeammin puuttumaan toistensa kohtaamiin ongelmiin.

Haastateltavat näkivät aikaisempaa kattavamman automaation käytön edesauttavan järjestelmän laadunhallintaa kahdesta näkökulmasta. A1, B2, C1 ja C2 olivat sitä mieltä, että infrastruktuurin-, integraatio- ja julkaisuautomaatio

poistavat radikaalisti virheitä, joita samat manuaalisesti tehtävät operaatiot tavallisimmin ja aikaisemiin ovat aiheuttaneet. C1 muun muassa kuvasi jatkuvaan integraatioon ja jatkuvaan julkaisemiseen liittyvän automaation yhteyttä järjestelmän laatuun seuraavasti:

> ”Koska paketit rakennetaan ja julkaistaan automaattisesti, niin meillä ei yleensä ole tullut sellaisia ’ei se toiminutkaan’ -efektejä. Eli automaation kautta pienennetään manuaalisten toimien kautta ilmenevien virheiden mahdollisuutta huomattavasti, koska yksinkertaisesti ohjelmistojulkaisua ei rakennetta eikä viedä tuotantoon, mikäli virheitä havaitaan.”

Tähän kuvattuun tilanteeseen vaikutti oleellisesti myös entistä kattavampi testausautomaation hyödyntäminen. A1 ja C2 mukaan DevOpsin mukaisen julkaisuautomaatioputken myötä testiautomaation hyödyntäminen on tullut tavallisemmaksi ja monta kertaa helpommaksi aikaisempiin toimintatapoihin verrattuna. Julkaisuputkeen yhdistetyn testiautomaation tarkoituksena on, että julkaisuehdokasta ei edes rakenneta, saati sitten julkaista, mikäli testauksessa tapahtuu virheitä. B1 ja B3 mukaan testiautomaation laajamittaisen käytön seurauksena kehittäjillä on uskallusta tehdä entistä vaikuttavampia muutoksia, sillä automaatioputken ja erityisesti testiautomaation laadun seurantaan voidaan luottaa manuaalisempaa toimintatapaa enemmän. B1 totesi seuraavasti testiautomaation käytön merkityksestä:

> ”Testiautomaatiosta johtuen kehittäjillä on uskallusta tehdä melko radikaaleja muutoksia. Jos ei uskalleta tehdä tarpeeksi kattavia muutoksia, niin se tarkoittaa sitä, ettei uskalleta tehdä juuri mitään ja ennemmin tai myöhemmin alkaa syntyä huonolaatuista kuraa sinne järjestelmään.”

Järjestelmän yleisen monitoroinnin sekä automaatioputkeen ja testaukseen kytketyn monitoroinnin nähtiin niin ikään parantavan järjestelmän laatua. B1 ja B2 mukaan järjestelmämonitoroinnilla kyetään ennaltaehkäisevästi lieventämään tuotantojärjestelmissä tapahtuvia virhetilanteita, koska tuotantojärjestelmiin asetettuun monitorointiin on kiinnitetty automaattisia hälytyksiä, jotka lauetessaan informoivat asianosaista ohjelmistotiimiä. Myös A1 ja C2 näkivät monitorointi- ja palauteratkaisujen parantavan laadun hallintaa. C2 mukaan esimerkiksi jatkuvaan integraatioon, missä uusi julkaisuehdokas rakennetaan, kytketyt lähdekoodin analysointityökalut parantavat merkittävästi myös yksittäisen kehittäjän tuotosta suosittelemalla vähemmän kompleksisia ratkaisuja sekä parempia ohjelmointikonventioita. A1 mukaan ’tavanomainen monitorointi’ oli kuitenkin ilmiselvän merkittävää. Hänen mukaansa järjestelmän laatu koetaan kuitenkin vasta loppukäyttäjän päässä. Loppukäyttäjien kokemaa laatua hän kuitenkin piti erittäin hankalasti monitoroitavana asiana.

6.3.3 Organisatoriset piirteet

Organisatoristen piirteiden kysymyskokonaisuudella pyrittiin hahmottamaan, kuinka Lehmanin neljäs (organisatorisen vakauden säilyttäminen) ja viides (tuttuuden säilyttäminen) lainalaisuus huomioidaan DevOps toimintamallissa ja sen käytänteissä. Lähdekirjallisuus ei juurikaan antanut selkeitä vastauksia aiheeseen. Tosin kirjallisuudesta oli tulkittavissa, että esimerkiksi uudenlaisella tiimirakenteella, automaatiolla ja monitoroinnilla olisi merkitystä kyseisten lainalaisuuksien huomioinnissa. Samat asiat kävivät ilmi myös haastateltujen vastauksista.

Balalaie ym., (2016), Basiri ym. (2016) ja Lwakatare ym. (2016) mukaisesti uudenlainen tiimirakenne auttaa kehitystiimiä sekä tuttuuden että organisatorisen vakauden säilyttämisessä. Haastatellut B2 ja B3 pitivät uuden tiimirakenteen tuoman laajemman osaamisen merkityksellisenä tuttuutta lisäävänä tekijänä. Toisin sanoen, infrastruktuuri- ja kehityspuolen tekijät yhdistävä tiimirakenne lisää tiimien näkemystä ja tietämystä ja tätä kautta tiimeissä osataan paremmin huomioida kehitykseen liittyviä asioita. B2 mukaan myös erilaista osaamista yhdistävä ja tiivistävä toimintamalli mahdollistaa paremmin kehitystiimin sisäistä oppimista. Tällöin infrastruktuuriosaajien tietämys siirtyy tehokkaammin tavallisille kehittäjille ja päinvastoin. C1 mukaan uudella tavalla vastuuta ottava tiimirakenne lisää tietämystä huomattavasti, erityisesti ylläpidon kannalta:

> "Tässä on just se mitä minun mielestä DevOpsissa on ajateltu. Se, että tietoisuus ylläpidolle ja jatkokehitykselle säilyy, koska kehityksestä ja tuotannosta vastaa itseasiassa samat henkilöt. Ei ole sitä ongelmaa, että tiimi hajoaisi julkaisun jälkeen. Tällöin uuden tekeminen saadaan nopeasti työn alle verrattuna aikaisempaan tilanteeseen, jossa jonkun täytyi ensin perehtyä asioihin syvemmin."

Haastateltujen A1 ja B1 mukaan DevOpsin myötä kehittäjien vaatimustaso on kasvanut huomattavasti ja tällöin uusia kehittäjiä on hankalampi löytää ja rekrytoida. Kehitystyötä oli kuitenkin pyritty standardisoimaan kaikissa haastatelluissa organisaatioissa. Organisaation B haastatellut B2 ja B3 muun muassa kertoivat koko organisaation halki toimivasta DevOps-yhteisöstä, jonka tarkoituksena on ollut jakaa tietämystä toimintatavasta ja sen käytänteistä halukkaille tiimeille. B2 kertoi myös, että DevOps toiminnassa alkuun pääsemiseksi tiimeille voidaan nimetä mentorihenkilöitä, jotka opastavat esimerkiksi työkalujen ja automaatioputkien rakentamisessa.

Standardisointi näkyi haastateltujen mukaan myös DevOpsin teknisemmällä puolella. A1 ja C2 mukaan automaation hyöty infrastruktuurin hallinnassa ja julkaisuputken sujuvoittamisessa konkretisoituu nimenomaan standardisoinnin kautta. A1 mielestä automaation standardisointi auttaa tavallista kehitystyötä, sillä kehittäjien ei tarvitse huolehtia automaatioympäristöjen muuttumisesta. C1 mukaan taas standardisointi tässä suhteessa osaltaan vakauttaa järjestelmän toimintaa eri ympäristöissä. Toisin sanoen, kun on standardoitu tapa tehdä automaatiota, on suuri todennäköisyys, että sama tapa toimii myös

muissa projekteissa ja järjestelmäympäristöissä. Lisäksi useat haastatellut näkivät automaation vaikuttavan merkittävästi kehittäjien ja täten koko organisaation tuottavuuteen. C2 automaatio ei ainoastaan paranna nopeuttaa, joilla järjestelemiä voidaan kehittää, mutta se myös poistaa tarpeettoman tekemisen. Tällöin siis kehitystyössä voidaan keskittyä enemmän ja tarkemmin järjestelmän liiketoiminnallista hyötyä tuoviin ydintoiminnallisuuksiin. A1 ja B2 näkivät automatisoidun julkaisuputken tuoman tuottavuuslisän hieman toisesta näkökulmasta. Heidän mukaansa se, että yksittäisten muutosten koko pidetään mahdollisimman pienenä, lisää organisatorista vakautta ja tuttuutta:

> ”Automatisointi helpottaa muutosten tekemistä, koska muutokset ovat riittävän pieniä. Eli tarpeeksi pienten muutosten tekemistä on paljon helpompi seurata.” (A1)

> ”Kukaan ei välttämättä tiedä mitä menee rikki, jos on isoja big bang -julkaisuja. Suurien julkaisujen vikatilanteista on vaikeampia toipua." (B2)

Teknologisista ratkaisuista myös monitorointi- ja palauteratkaisujen käytön nähtiin edistävän kehitystyötä tuttuuden säilyttämiseksi. A1 ja C2 mukaan näiden tekniikoiden soveltaminen on lisännyt kehitystiimien tilannetietoisuutta ja organisaation kokonaistehokkuutta merkittävästi. Esimerkiksi A1 totesikin, että palvelinprosesseihin, integraatioihin ja julkaisuihin kytketyt monitoroinnit saadaan visualisoitua erilaisten dashboard-sovellusten kautta. Hänen mukaansa reaaliajassa tapahtuvan aktiivisen seurannan vuoksi, on epätodennäköisempää joutua tilanteeseen, jossa kukaan ei tiedä mitä tuotantojärjestelmässä on rikki. B1 ja C2 näkivät monitoroinnin myös yksittäisen kehittäjän tietoisuuta lisäävänä tekijänä. Esimerkiksi julkaisuputkeen kytketyn lähdekoodin analysointityökalun he näkivät edistävän tilannetietoisuuden lisäksi yksittäisen kehittäjän oppimista. B1 mielsi monitoroinnilla olevan myös strategisia merkityksiä koko organisaatiolle. Hänen mukaansa esimerkiksi analysointityökalusta voidaan saada pitkää aikaväliä tarkastelevia virhehistoriaraportteja. Tällöin järjestelmäsuunnittelussa ja kehityksessä kyetään tarkastelemaan pitkän aikavälin muutosten vaikutusta järjestelmän kokonaislaatuun.

6.3.4 Ohjelmistoprosessin kehittäminen

Ohjelmistoprosessin kehittämisen haastattelukysymykset muotoiltiin pääasiassa Lehmanin kolmannen lainalaisuuden (prosessin itsesäätelevyys) ja kahdeksannen (palautejärjestelmä) pohjalta. Haastatellut A1, B1 ja C2 kertoivat konkreettisista prosessin kehittämisen toimista lähempää tavallista kehitystyötä. Heidän mukaansa prosessia kehitettiin lähinnä uudistamalla työkaluja, monitorointi- ja palauteratkaisuja tai kehittämällä itse uusia ominaisuuksia näihin. Teknologisessa uudistamisessa heidän mukaansa pyrittiin automatisoimaan useasti toistuvia asioita ja tätä kautta helpottamaan ja nopeuttamaan työskentelyä sekä tehostamaan palautteen saantia.

B2, B3 ja C1 puolestaan havainnollistivat prosessin kehittämisen toimintaa korkeammalta tasolta. Kuten aiemmin mainittiin, olivat yritykset B ja C luoneet DevOps sidonnaista yhteisötoimintaa organisaatioidensa halki. Yhteisöjen tarkoituksena on ollut tarjota tietämyksen siirtoa tiimiltä toiselle yhteisten tapaamisten ja keskustelupalstojen kautta. B2 ja C1 mukaan yhteisöllistä toimintaa on ollut myös ennen DevOps toimintamalliin yleistymistä, mutta DevOpsin myötä yhteisötoimintaan on panostettu enemmän ja toiminnasta on myös kiinnostuttu enemmän muun henkilöstön keskuudessa. Molemmat haastatellut B2 ja C1 näkivät yhteisöllisen toimintamallin tuottavan laatua ja asiakastyytyväisyyttä ja täten olevan liiketoiminnallisesti merkittävää.

Prosessin kehittämisen käytänteet vaihtelivat haastatelluissa organisaatioissa. Erityisesti systemaattisuus asioiden eteenpäin viemisessä oli kehittyneempää yrityksissä, jotka toteuttivat järjestelmiä ydintoimintonaan. A1, joka edusti ei ydintoimintonaan ohjelmistoja toteuttavaa organisaatiota, totesi, että prosessin ja työkalujen kehittäminen ei ole projektoitua toimintaa. Hänen mukaansa heillä oman toiminnan kehittämiseen käytettävä aika pihistetään varsinaisen ydintoiminnon suorittamisesta. Hänen mukaansa kuitenkin ainoa tapa skaalata organisaation toimintaa ja viedä prosessia eteenpäin, on kehittämällä omaa toimintaa.

B1 mukaan DevOpsin myötä tuottavuus on koettu entistä tärkeämpänä ja tästä syystä oman toiminnan ja ohjelmistoprosessin kehittämisen suunnittelu on syytä projektoida. B1 ja B3 arvioivat, että heidän projekteissaan toiminnan kehittämiselle on allokoitu noin viidennes henkilöstöresurssista. Tosin B1 mainitsemissa tapauksissa kehitys ja sen suunnittelu on myöskin ollut varsin epäformaalia, sillä toiminnan edistäminen ei ole perustunut varastoituun tietoon:

> "Primääristi me ollaan toimittu niin, että, jos johonkin tekemiseen kuluu paljon aikaa tai näyttää siltä, että parin sprintin päästä on apinaduunia tiedossa, niin porukalla mietitään et mitä kannattaisi tehdä."

Tilanne on myös organisaatiossa C ollut samankaltainen. C1 mukaan toiminnan kehittämistä selkeästi resursoidaan, mutta työmäärän arviointi on perustunut lähinnä varsinaisten kehittäjien henkilökohtaisiin arvioihin. Ongelmalliseksi, erityisesti yksittäisten tiimien kohdalla, toiminnan kehittämisessä on koettu osaavan henkilöstön puute. B1 mukaan ongelma näkyy juuri DevOpsin myötä kasvaneessa vaatimustasossa:

> ”Minun mielestä ongelma on se, että DevOps konseptina on aika vaativa tiimille. Jos Google tai Facebook sanoo, että näin järjestelmiä kehitetään, niin ei keskiverto ohjelmistotiimi ole ikinä samaa luokkaa kuin Googlen tai Facebookin vastaava.”

Organisaatioissa B ja C haastateltujen C1 ja B2 mukaan toiminnan kehittämistä ja sen suunnittelua oli kuitenkin viety koko organisaation tasolla korkeammalle muun muassa yhteisö- ja mentorointitoiminnan kautta.

7 POHDINTA

Tämän luvun tarkoituksena on pohtia haastattelututkimuksesta saatuja tuloksia. Luku on jaettu kolmeen alaosioon, joista ensimmäisessä pohditaan DevOps toimintamallia, sen käytänteitä, sen aiheuttamia haasteita sekä siitä saatavia hyötyjä. Luvun toisessa osiossa tarkastellaan tarkemmin DevOps toimintamallia järjestelmäevoluution näkökulmasta vastaten tutkielman tutkimuskysymykseen. Viimeisessä luvussa arvioidaan tutkimuksen reliabiliteetti- ja validiteettikysymyksiä.

7.1 DevOps toimintamalli ohjelmistotuotannossa

Tutkielman teemahaastattelujen tulokset vastasivat käsitteellisesti tieteellisissä julkaisuissa annettua kuvaa DevOps toiminnasta. Tulosten perusteella tieteellisessä materiaalissa havaitut käytänteet olivat todella konkreettisia osia DevOps toimintamallissa. DevOps toiminnan mukaisella yhteistyöllä tiivistettiin kehityksen, ylläpidon ja operoinnin toimintoja organisoimalla tiimit yhdeksi moniosaajatiimiksi. Uuden tiimirakenteen myötä tulosten mukaan moniosaajatiimit ottavat vastuun myös järjestelmän operoinnista. Tällöin ei siis synny julkaisun jälkeistä vastuunsiirtoa, kuten useat tieteelliset lähteet havainnollistavat (Humble & Molesky, 2011; Hütterman, 2012, Lwakatare ym., 2016; Swartout, 2014). Tosin saadut tulokset osoittivat myös, että vastuun siirron seurauksena yksittäisen tiimin kehittäjien vaatimustaso on kasvanut. Tämän puolestaan voidaan tulkita hankaloittavan tehokkaan DevOps toimintamallin aikaansaamista.

Tulosten perusteella automaation tavat osoittautuivat keskeisimmiksi uutuuksiksi DevOpsin myötä. Automaatio nähdään sekä järjestelmän julkaisuprosessissa että infrastruktuurin hallinnassa. Automatisoitu julkaisuprosessi oli tulosten perusteella samankaltainen kuin mitä tieteellisissä lähteissä oli kuvattu (Humble & Molesky, 2011; Hütterman 2012; Roche, 2013; Waller, 2015). Saatujen tulosten mukaan pilvipalveluiden yleistymisen myötä myös infrastruktuurin hallinnan tapa on osittain automatisoitu, sillä nyt infrastruktuuria voidaan

komentaa lähdekoodin avulla. Juuri tästä syystä haastatellut kokivat kehittäjien vaatimustason kasvaneen, sillä nyt kehittäjiltä on alettu odottaa osaamista myös infrastruktuurin hallinnan tavoista. Infrastruktuurin hallinnan keinot ovat näin tulkittavissa myös yhdeksi syyksi siihen, että on alettu hyödyntää moniosaaja-ohjelmistotiimejä. Tällöin voidaan myös tulkita teknologian olevan mahdollistamassa DevOps toimintamallin mukaista tiimiyhteistyörakennetta.

Yleisellä tasolla automatisointi nähtiin enemmän hyötynä kuin haittana. Vaatimushaasteesta huolimatta automaation laajamittainen käyttö nopeuttaa ohjelmistojulkaisuja jatkuvan integraation ja jatkuvan julkaisun prosessien takia. Lisäksi automaation nähtiin parantavan järjestelmän laatua julkaisuprosessiin integroidun testiautomaation ja manuaalisten toimien vähentymisen seurauksena. Nopeampien ja laadukkaampien julkaisujen toteutuessa automaation käytöllä voitaneen nähdä yhteys asiakas- ja käyttäjätyytyväisyyteen sekä kehittäjäorganisaation kustannussäästöön. Kustannussäästöt realisoituvat ensisijaisesti lyhyempien kehitysaikojen ja vähentyneen ylläpitotarpeen vuoksi. Asiakastyytyväisyyden paranemisen nähtiin syntyvän nopeutuneen julkaisun seurauksena.

DevOps toiminnan mukainen monitorointi oli tulosten perusteella hyvin samankaltaista kuin miten sitä oli kuvattu tutkitussa tieteellisessä lähdemateriaalissa (Balalaie ym., 2016, Callanan & Spillane, 2016; Stillwell & Coutinho, 2015;). Haastattelutulosten mukaan monitoroinnilla tavoiteltiin järjestelmävirheiden havaitsemista kehityksen, julkaisun ja tuotannonkin eri vaiheissa. Monitorointi ei kuitenkaan saatujen tulosten perusteella ollut DevOpsin mukana tullut käytänne, mutta sen käytön nähtiin kuitenkin yleistyneen uusien työkalujen helppokäyttöisyyden ja vaivattoman implementoinnin myötä. Eräs mielenkiintoinen ja tieteellisessä lähdemateriaalissa korostamaton havainto oli monitoroinnin kytkeminen liiketoiminnan vaatimuksiin. Osa haastatelluista näkikin, että monitorointityökalujen helppokäyttöisyys on johtanut tilanteeseen, että järjestelmiä kyetään arvioimaan yhä enenevissä määrin niiden liiketoiminnallisen vaikutusten näkökulmasta. Tulokset eivät kuitenkaan anna riittävän konkreettista kuvaa liiketoimintavaatimusten ja monitoroinnin kytkemisestä. Mahdollisen jatkotutkimuksen kannalta olisikin mielenkiintoista selvittää, kuinka järjestelmänmonitorointia voidaan kytkeä tilaajan liiketoiminnan asettamiin vaatimuksiin.

Tieteellisten lähteiden mukaan monitoroinnin kautta saatavaa palautetta tulisi DevOps toiminnassa hyödyntää ohjelmistotuotantotoiminnan kehittämiseen (Humble & Molesky, 2011; Hütterman, 2012). Vain johtotehtävissä toimivat haastatellut osasivat arvioida mittaamisen käytännettä konkreettisesti. Haastattelutulosten mukaan mittaamisen tavat ovat DevOpsin myötä muuttuneet aikaisemmasta enemmän ohjelmistotuotantoprosessin tehokkuutta kuvaaviin mittareihin. Toisin sanoen kehitystiimiä ei enää vanhakantaisesti arvioida suhteessa järjestelmän laatuun eikä operointitoimintaa suhteessa järjestelmän vakauteen. Sen sijaan uuden tiimirakennemallin myötä mittaaminen keskittyykin pikemminkin tuotannon läpimenoaikaan sekä ohjelmistojulkaisua hidastavien tekijöiden arviointiin. Tällöin mittareilla pystytään paremmin arvioimaan tilaajan kokemaa arvoa eli aikaa, joka kuluu järjestelmän käyttöönottoon.

Saatuihin tuloksiin perustuen DevOps toiminta voidaan nähdä kahdesta näkökulmasta. Toisaalta toiminta näkyy koko ohjelmistotuotantoa ohjaavana. Tällöin voidaan ajatella mittaamisen antavan palautetta prosessin toimivuudesta, sujuvuudesta ja nopeudesta. Tätä näkökulmaa edustivat erityisesti johtavissa rooleissa toimivat haastatellut. Konkreettinen esimerkki tästä oli, kun eräs haastatelluista vertasi DevOpsia valmistavan teollisuuden tuotantolinjojen automatisointiin ja optimointiin. Johdon näkökulmasta DevOpsia voitaneenkin tällöin ajatella ohjelmistotuotantoa tehostavana ja optimoivana toimintana.

Lähempänä kehitystä toimineet haastatellut taas näkivät DevOps toiminnan teknisemmästä näkökulmasta. Heidän antamiensa vastausten perusteella tekninen kehitysympäristö automaation ja monitoroinnin avulla parantaa niin sanottua kehittäjäkokemusta. Kehittäjäkokemuksen parantuminen voitaneen tässä yhteydessä tulkita monesta ulottuvuudesta. Ensinnäkin automatiikan vuoksi muutosten tekeminen ja infrastruktuurin hallinta ovat helppoja ja vaivattomia toimenpiteitä. Toisekseen monitoroinnin ja testiautomaation integroiminen julkaisuprosessiin mahdollistaa varsin radikaalienkin järjestelmämuutosten tekemisen ilman, että järjestelmäkokonaisuus lakkaisi toimimasta. Lisäksi automaation korvatessa aikaisemmin manuaalisia ja aikavieviä toimenpiteitä kehityksessä voidaan keskittyä ydinasiaan eli toimintojen kehittämiseen.

7.2 DevOps järjestelmäevoluution näkökulmasta

Teemahaastattelussa Lehmanin järjestelmäevoluution lainalaisuudet koostettiin neljäksi helpommin ymmärrettäväksi kokonaisuudeksi. Seuraavaassa pohditaan sitä, miten evoluution lainalaisuuksien huomiointi näkyy DevOpsin käytänteissä.

7.2.1 Vaikutukset muutosreagointikykyyn

Teemahaastattelun muutospainekokonaisuus muodostettiin Lehmanin järjestelmäevoluution piirteitä kuvaavien ensimmäisen (Jatkuva muutos) ja kuudennen (jatkuva kasvu) lainalaisuuden perusteella. Ensimmäisen lainalaisuuden mukaan E-tyypin järjestelmiä tulee muuttaa jatkuvasti, jotta niiden käyttöä ei koeta epätyydyttäväksi. Kuudes lainalaisuus puolestaan kertoo, että E-tyypin järjestelmien toiminnallisuutta tulee lisätä, jotta käyttäjätyytyväisyys taataan koko järjestelmän elinkaaren ajan. Saatujen tulosten perusteella yleisesti ottaen järjestelmäkehityksessä koettu muutospaine oli kategorisoitavissa sisäisiin ja ulkoisiin tekijöihin. Sisäisillä tekijöillä tarkoitetaan tässä yhteydessä järjestelmän toiminnallisten tai ei-toiminnallisten vaatimusten muutosta. Ulkoiset tekijät taas kertovat käytetyn teknologiaekosysteemin muutoksesta.

Kuten jo tieteellisen lähdekirjallisuuden perusteella havainnoitiin, DevOpsin automaatiokäytänteet ovat avainasemassa muutospaineen hallinnan näkökulmasta (Barry ym., 2007; Basiri ym., 2016). Tulosten perusteella on selvää,

että automaatio nopeuttaa kehitystahtia, sillä automaatiolla korvataan aikaisemmin manuaalisesti tehtyjä toimia. Tällöin myös muutokseen reagointi nopeutuu. Toisaalta tulokset osoittivat myös, että julkaisuautomaatioputkessa järjestelmäjulkaisut ovat tyypillisesti varsin pienikokoisia. Tästä syystä voitaneen tulkita DevOps toimintamallin automaatiokäytänteen helpottavan myös järjestelmämuutosten hallintaa, sillä usein pienempään osaan järjestelmää vaikuttavat muutokset ovat helpommin hallittavissa. Toisekseen pienempi kokoisia muutoksia voidaan automaatioputken avulla tehdä jopa useita kertoja päivässä, minkä voidaan nähdä nopeuttavan muutokseen vastaamisen ketteryyttä. Lisäksi haastatteluista saadut tulokset osoittivat, että automaatioputkeen sisällytetyn jatkuvan integraation ja tähän liittyvän testiautomaation seurauksena järjestelmämuutosten toteuttamista ei tarvitse arastella. Tämä johtuu siitä, että jatkuvan integraation ja julkaisun prosessissa tapahtuvan testiautomaation epäonnistuessa järjestelmämuutosta ei missään tapauksessa julkaista tuotantoon.

Saadut tulokset monitorointikäytänteen ja muutospaineen hallinnan suhteen ovat saman suuntaisia kuin mitä tutkittu tieteellinen kirjallisuuden perusteella esitettiin (Liu ym., 2014; Lwakatare ym., 2016). Haastattelutuloksista käy ilmi, että monitoroinnin käytänne ei varsinaisesti ole DevOpsin myötä tullut trendi. Monitoroinnin käyttö on kuitenkin yleistynyt, sillä DevOps aallon mukana tullut teknologinen työkalupakki on entistä helppokäyttöisempi. Tällöin monitoroinnin työkalujen käyttöönotto on koettu entistä sujuvammaksi. Pääasiallisesti monitoroinnilla pyritään havaitsemaan järjestelmän virhetilanteita. Tämä tietenkin tarkoittaa, että tehokkaiden monitorointiprosessien käytöllä on suora yhteys muutosreagointiin. Lisäksi tulosten mukaan nykyisillä monitorointityökaluilla kyetään muun muassa havaitsemaan virhetilanteita proaktiivisesti eli esimerkiksi ennen mahdollisten katastrofien tapahtumista. Voitaneen siis tulkita monitorointikäytänteen edesauttavan muutoksiin reagointia ja muutoshallintaa. Tutkitussa tieteellisessä kirjallisuusmateriaalissa ei juurikaan mainittu syitä monitoroinnin yleistymiseen. Saatujen tulosten perusteella monitorointia pidettiin kuitenkin keinona pysyä kattavan automaation mahdollistamassa nopeassa muutostahdissa. Tämän seurauksena monitoroinnin voidaan ajatella olevan välttämättömyys, jotta järjestelmämuutoksia kyetään hallitsemaan pitkälle automatisoidussa kehitysympäristössä.

Tutkitussa lähdekirjallisuudesta ei ollut tulkittavissa yhteyttä DevOpsin yhteistyökäytänteen ja järjestelmämuutoksiin kantaa ottavien evoluution lainalaisuuksien yhteydestä. Haastattelutulosten perusteella kuitenkin yhteistyökäytänteillä nähtiin positiivisia vaikutuksia muutospaineen hallintaan. Esimerkiksi yhteistyökäytänteen mukaisen moniosaajatiimirakenteen nähtiin parantavan muutosreagointikykyä, sillä tiimien osaaminen on aikaisempia kehitystapoja poikkileikkaavampaa. Tällöin voidaan esimerkiksi muassa jo suunnitteluvaiheissa ottaa paremmin huomioon eri asioita, johtuen uudenmallisen kehitystiimin kasvaneesta ymmärryksestä. Lisäksi aikaisempaa syvempi ymmärrys koko ohjelmistotuotantoputkesta nopeuttaa tiimin omaksumiskykyä lisäperehtymistä vaativissa tilanteissa.

Saatujen tulosten valossa uudenmalliset ohjelmistotiimit ottavat vastuuta järjestelmän operoinnista myös tuotannossa. Tässä tilanteessa kehityksestä, ylläpidosta ja operoinnista vastaa täysin sama henkilöstö. Tämän voidaan nähdä edesauttavan muutosten hallintakykyä, sillä muutosten havaitsemisen ja toteuttamisen välillä ei ole hidastavia välikäsiä. Uuden tiimirakennelman myötä tavallisella kehitystiimillä on aikaisempaa syvempi näkyvyys järjestelmäinfrastruktuurin puolelle. Tämä osaltaan nopeutta esimerkiksi virheiden selvitystä ja infrastruktuurimuutosten tekemistä.

Vaikka DevOps toimintamallin hyödyntämisellä on tulosten mukaan suopeita vaikutuksia ohjelmistokehityksen muutosreagointikykyyn, myös tiettyjä hankaluuksia oli havaittavissa. Esimerkiksi julkaisuprosessiautomaation nähtiin paikoin ruokkivan tulevaa muutosta entisestään. Tällöin ongelmalliseksi koettiin, että kehitettävä järjestelmä alkaa yhä enenevissä määrin imemään muutosta. Toisaalta taas automaatio- ja monitorointityökalujen käytön seurauksena järjestelmäkehityksen teknologiariippuvuus on kasvanut entisestään. DevOps toimintamallissa ei voida tyytyä ainoastaan järjestelmän sisäisten muutosten hallintaan, sillä nyt myös ohjelmistotuotantoprosessi nähdään yhä enemmän omana teknologisena systeeminään ja on näin ollen altis esimerkiksi kehitystyökaluissa tapahtuville muutoksille. Toisin sanoen toimintamallin mukaisesti ei ainoastaan kehitetä järjestelmää tilaajalle, vaan kehitetään myös kehitysympäristöjärjestelmää, jolla varsinaisia tilaajavaatimusten mukaisia järjestelmiä tuotetaan.

7.2.2 Laadun takaaminen osana toimintaa

Teemahaastattelussa laadun hallinnan asiakokonaisuus muodostettiin Lehmanin kuvaamista toisesta (Kasvava kompleksisuus) ja seitsemännestä (Heikkenevä laatu) evoluution lainalaisuudesta. Toisen lainalaisuuden mukaan E-tyypin järjestelmän kehittyessä sen kompleksisuus kasvaa, ellei sitä pyritä tarkoituksenmukaisesti vähentämään. Seitsemäs puolestaan esittää, että mikäli E-tyypin järjestelmän toimintaympäristön muutoksia ei oteta huomioon, järjestelmän laatu alkaa heikentyä ajan kuluessa. Tutkielman lähdekirjallisuusosiossa esitetty tulkinta paljasti, että DevOpsin mukaisilla yhteistyön, automaation ja monitoroinnin käytänteillä olisi positiivisia vaikutuksia järjestelmien laadun hallinnan kannalta. Haastattelututkimuksessa saaduista tuloksista on havaittavissa yhteneväinen tulkinta. Tulosten perusteella kyseisillä käytänteillä on selkeitä myönteisiä seurauksia laatua käsitteleviin järjestelmäevoluution lainalaisuuksiin. Tulokset osoittavat, että DevOps käytänteiden hyödyntäminen voidaan nähdä tapana upottaa järjestelmän laadun tarkkailu entistä syvemmäksi osaksi ohjelmistotuotantoketjua.

Yhteistyökäytänteen ja uudenlaisen moniosaajatiimirakenteen vaikutus laadun hallintaan näkyy kahdesta syystä. Uuden tiimirakenteen myötä kehitystiimeillä on kokonaisvaltaisempi vastuu järjestelmän kehityksestä tarkoittaen, että he ottavat vastuun osittain myös järjestelmän operoinnista kehityksen lisäksi. Tällöin esimerkiksi järjestelmävirheiden tai muiden muutosten toteutta-

minen on nopeampaa ja mahdollisesta laadun heikkenemisestä päästään varhaisemmassa vaiheessa eroon. Toisaalta haastattelutulokset osoittivat, että uuden moniosaajarakenteen seurauksena kehitystiimeillä on kattavampaa ja poikkileikkaavampaa osaamista. Tällöin tiiviimmässä yhteistyössä toimivat eri alueiden spesialistit yhteisosaamisellaan ymmärtävät järjestelmää ja sen laatua sekä toiminnallisesta että sen infrastruktuurin näkökulmasta. Tämä näkemys oli yhteneväinen analysoituun lähdekirjallisuuteen verrattuna (Balalaie ym., 2016; Lwakatare ym., 2016; Stillwell & Coutinho, 2015).

Tulosten perusteella automaation vaikutus laatua käsitteleviin lainalaisuuksiin nähtiin niin ikään kahdesta näkökulmasta. Ensinnäkin virhealttiiden manuaalisten toimenpiteiden korvaaminen automatisoiduilla julkaisu- ja infrastruktuuriprosesseilla vähentää radikaalisti epähuomiossa tehtyjen virheiden määrää. Myös tutkittu lähdekirjallisuus paljasti samankaltaisia havaintoja (Callanan & Spillane, 2016; Swartout, 2014). Toisaalta haastattelutulokset osoittivat testiautomaation käytön parantavan järjestelmien laatua, mikä olikin osittain havaittu jo lähdekirjallisuudessa (Basisiri ym., 2016). Haastattelutulosten mukaan automaattisen testauksen on nähty helpottuneen DevOpsin myötä yleistyneiden työkalujen vuoksi. Automatisoidussa julkaisuputkessa testiautomaatio pitää huolen, ettei epäonnistuneen testauksen jälkeen virheitä sisältävä järjestelmäversio päädy tuotantoon asti. Testiautomaatio ei kuitenkaan tarkoita, että eri tyyppiset testit ilmestyisivät tyhjästä. Onkin huomioitava, että testiautomaatio tietenkin vain parantaa sellaisten ominaisuuksien laatua, joiden testitapaukset on toteutettu.

Myös monitoroinnilla on haastattelutulosten mukaan vaikutus järjestelmien laadun takaamisessa. Monitoroinnilla havaittiin olevan muun muassa virheitä ennalta ehkäisevä luonne, mikä oli osin todettu myös lähdekirjallisuuden valossa (Basiri ym., 2016). Monitorointityökaluina esimerkiksi palvelinprosesseihin yhdistetyt hälytykset voivat informoida asianosaista ohjelmistotiimiä jo siinä vaiheessa, kun kyseinen palvelin on alkamassa ylikuormittua. Tällöin heikkoon laatuun voidaan reagoida jo ennen kuin onnettomuutta on varsinaisesti päässyt tapahtumaan. Toisaalta tulosten mukaan myös monitorointiin yhdistetyillä lähdekoodin analysointityökaluilla havaittiin laatua parantava vaikutus. Toisissa haastatteluihin osallistuneissa organisaatioissa käytetyt analysointityökalut kykenivät nimittäin suosittelemaan yksittäisen kehittäjän tuotokseen yksinkertaisempia ratkaisuja sekä parempia ohjelmointikäytänteitä näin parantaen lähdekoodin laatua.

7.2.3 Ohjelmistotuotannon organisatorinen hallinta

Haastattelututkimuksen organisatoristen piirteiden teemakokonaisuus koostettiin Lehmanin neljännestä (organisatorisen vakauden säilyttäminen) ja viidennestä (tuttuuden säilyttäminen) järjestelmäevoluution lainalaisuudesta. Tutkitun lähdekirjallisuuden tulkinta ei antanut selkeitä yhteneväisyyksiä DevOps toiminnan ja kyseisten lainalaisuuksien välillä. Haastattelutulosten perusteella

käytänteillä kuitenkin oli havaittavia yhteyksiä lainalaisuuksien huomiointiin sekä positiivisessa että negatiivisessa valossa.

Yhteistyökäytänteen mukainen tiimirakennemalli voidaan tulosten mukaan tulkita parantavan kehitystiimien tuttuutta järjestelmiin. Samainen näkemys oli tulkittavissa myös Basirin ym. (2016), Balalaien ym. (2016) ja Lwakataren ym. (2016) katsausten perusteella. Tuttuuden paranemisen nähtiin johtuvan siitä, että eri alueiden osaamisen keskittyminen tiimin sisälle lisää tiimien näkemystä ja ymmärrystä kohdejärjestelmästä. Toisaalta tiimirakenne edesauttaa sen jäsenten oppimista, sillä esimerkiksi infrastruktuuriosaajat voivat opastaa tavallista kehityshenkilöstöä ja päinvastoin. Moniosaajatiimirakenne kuitenkin asettaa haasteen organisatorisen vakauden säilymiselle, sillä sen myötä tavallisen kehittäjän vaatimustason on nähty kasvaneen. Tällöin voidaan tulkita, että DevOpsin mukaisen tiimitoiminnan myötä uusien kehittäjien rekrytoiminen on muuttunut haastavammaksi, jolloin ohjelmistotuotannon organisatorista vakautta on vaikeampaa säilyttää.

Kasvanutta vaatimushaastetta oli kahdessa tutkimukseen osallistuneessa organisaatioissa kuitenkin helpotettu standardoimalla toimintaa ja teknistä ympäristöä. Toiminnan standardoinnin tarkoituksena oli, että DevOps toimintaa toteutetaan kaikissa tiimeissä tietyllä tavalla. Tällöin organisaatio oli muodostanut erityisiä yhteisöjä, joiden kautta DevOps toiminnassa alkuun pääsemistä oli helpotettu esimerkiksi mentorihenkilöiden avulla. Teknisemmästä näkökulmasta standardointi näkyi yhdenmukaistettuina kehitys- ja tuotantoympäristöinä sekä kehitystyökaluina. Standardoidun teknologisen työkalupakin voidaan nähdä lisäävän koko organisaation tuttuutta kehitettäviin järjestelmiin. Lisäksi standardoinnin voidaan tulkita parantavan organisatorista vakautta. Tämä johtuu muun muassa siitä, että vakiomuotoisen ympäristön käyttöönoton kautta ohjelmistokehityksessä voidaan keskittyä nopeammin tuottamaan tilattujen järjestelmien liiketoiminnallisia tavoitteita sen sijaan, että rakennettaisiin ensin kehitysympäristö kehitystyötä varten.

Tulokset osoittavat automaatiokäytänteen hyödyllisyyden erityisesti tuttuutta säilyttävästä näkökulmasta. Samainen näkemys automaatiosta ohjelmistokehitystä stabilisoivana menetelmänä on todettu jo useita vuosia sitten tehdyssä tutkimuksessa (Barry ym., 2007). Haastattelutulosten mukaan julkaisuautomaatioputken käytössä järjestelmäjulkaisut pidetään kooltaan mahdollisimman pieninä. Tämä edesauttaa tuttuuden säilymistä, sillä pienempikokoisia muutoksia ja niiden vaikutuksia on helpompi seurata. Suurikokoisten julkaisujen tapauksessa saatettaisiin päätyä tilanteeseen, jossa ei varsinaisesti tiedetä virheen aiheuttavan muutoksen syytä.

Monitoroinnin yhteys evoluution lainalaisuuksiin on tulosten perusteella tulkittavissa kehityksen tuttuutta edistävänä käytänteenä. Monitorien ja niihin yhdistettyjen palauteketjujen nähdään ennen kaikkea lisäävän kehitystiimien tilannetietoisuutta järjestelmien toiminnasta. Lisäksi esimerkiksi aiemmin havainnoidun lähdekoodin analysointityökalun voidaan tulkita parantavan yksittäisen kehittäjän tilannetietoisuutta ja tuttuutta kehitettävään järjestelmään. Toisaalta monitoroinnilla voidaan myös ajatella olevan strategista merkitystä

ohjelmisto-organisaation toiminnalle. Viisaasti kehitetyn monitoroinnin avulla organisaatiot voivat kyetä ohjaamaan ylläpitotyötä sellaisiin järjestelmiin, joissa virhetapausten raportointi on muita järjestelmiä tavallisempaa. Näin ollen monitoroinnin käytöllä voidaan tulkita olevan yhteys myös organisatorista vakautta edistävänä käytänteenä.

7.2.4 Tuotantoprosessin kehittäminen

Tutkimushaastattelussa ohjelmistotuotantoprosessin kehittämiseen liittyvät kysymykset koostettiin perustuen Lehmanin järjestelmäevoluution kolmanteen (prosessin itsesäätelevyys) ja kahdeksanteen (palautejärjestelmä) lainalaisuuteen. Kolmannen lainalaisuuden mukaan järjestelmäkehitysprosessi on piirteiltään itseään säätelevä, mikä tarkoittaa, että prosessin on muunnuttava ajan kuluessa. Palautejärjestelmää käsittelevä kahdeksas lainalaisuus puolestaan kertoo, että järjestelmien muuntaminen perustuu saatavaan palautteeseen. Tällöin hyvin toimivassa kehitysprosessissa on useita eritasoisia ja eri sidosryhmiä yhdistäviä palauteketjuja, joiden kautta saatavan palautteen perusteella kehitysprosessia ja siinä kehitettyjä järjestelmiä ylläpidetään.

Tutkittuun lähdekirjallisuuteen perustuen tutkielman kirjallisuuskatsausosiossa arvioitiin, että DevOpsin mittaamisen ja monitoroinnin käytänteillä olisi merkitystä ohjelmistoprosessin säätämisen kannalta (Lwakatare ym., 2016; Humble & Molesky, 2011). Samankaltaisia havaintoja on tehtävissä haastatteluista saatujen tulosten perusteella. Kehitysprosessin säätäminen on tulosten mukaisesti tulkittavissa teknisen ympäristön kehittämisen ja tuotantoprosessin optimoinnin näkökulmista. Prosessin kehittämistä kuitenkin toteutettiin yrityksissä vaihtelevalla systemaattisuudella. Syvemmin DevOps toimintaa ymmärtävät yritykset toteuttivat toiminnan kehittämistään organisoidummin.

Teknologisesta näkökulmasta tärkeänä koettiin esimerkiksi automaattisen julkaisuputken sekä kehitysympäristöjen ja -työkalujen kehittäminen. Tekninen kehittäminen oli ensisijaisesti lähempänä varsinaista kehitystyötä toimivien henkilöiden prioriteettina. Teknisessä kehittämisessä pyrittiin helpottamaan ja nopeuttamaan tavallista kehitystyötä tai esimerkiksi tehostamaan palautteen saantia. Tulosten mukaan järjestelmämonitorointi voidaan käsittää menetelmänä, jonka kautta saatavaa palautetta hyödynnetään teknologisen kehittämisessä. Teknologisen kehittämisen syötteenä toimivat esimerkiksi järjestelmävirheitä raportoivat monitorointityökalut. Saatujen tulosten valossa monitoroinnin käytännettä voidaan pitää myös Lehmanin palautejärjestelmä lainalaisuuden mukaisena palauteketjuna. Kyseiset havainnot tukevat Callanen ja Spillanen (2016) sekä Lwakatater ym. (2016) antamaa näkemystä monitorointikäytänteen hyödyllisyydestä.

Tuotantoprosessin optimointi osana toiminnan kehittämistä näkyy tuloksissa usealla tavalla. Organisaatiot olivat esimerkiksi perustaneet yhteisöjä, joilla muassa autettiin alkuun vielä DevOps toiminnassa harjaantumattomia ohjelmistotiimejä. Toisekseen erityisesti DevOpsin mukainen mittaamisen käytänne on tulosten mukaisesti tulkittavissa yhdeksi merkittävimmäksi osaksi

prosessin optimointia. Pääasiallisesti mittaamisen tarkoituksena on tuottaa informaatiota ohjelmistokehityksen tehokkuudesta havainnoimalla esimerkiksi ohjelmistojulkaisujen läpimenoaikoja. Tällöin mittaamisen kautta saatavalla palautteella voidaan säätää ohjelmistotuotantoprosessissa ilmeneviä hidasteita. Tässä valossa myös DevOpsin mittaamisen käytännettä voidaan niin ikään pitää Lehmanin kuvaamana palautejärjestelmänä.

Mittaaminen vaatii kuitenkin teknistä ympäristöä ja oikeanlaista informaatiota keräävän monitoroinnin olemassa oloa. Tällöin voidaan ajatella monitoroinnin ja mittaamisen käytänteiden olevan prosessin kehittämisen ja optimoinnin kannalta lomittaisia. Tässä mielessä siis mittaaminen ja monitorointi ovat toinen toistaan tukevia toimia ja samalla ohjelmistoprosessin kehittämisen kannalta olennaisimpia palautejärjestelmiä.

7.3 Tutkimuksen luotettavuuden arviointi

Tutkimuksen luotettavuutta voidaan arvioida reliabiliteetin ja validiteetin käsitteistön kautta. Reliabiliteetilla tarkoitetaan, että samantapaisella tutkimusmenetelmällä päädytään identtiseen tutkimustulokseen. Toisin sanoen tutkimustulosten reliaabelius kuvaa tulosten toistettavuutta. Tutkimuksen validiteetilla puolestaan tarkoitetaan sitä, kuinka hyvin käytetty tutkimusmenetelmää tutkii sitä, mitä sen on ajateltu tutkittavan. (Hirsijärvi ym., 2004.)

Tämän tutkimuksen tapauksessa saatujen tulosten reliabiliteettia on hankala todeta, sillä DevOps toimintamallin ja järjestelmäevoluution yhteydestä ei ole aikaisempaa vertailukelpoista tutkimustietoa. Tosin reliaabeliutta on pyritty parantamaan tarkasti dokumentoidulla tutkimusprosessilla. Erityisesti teemahaastattelun tapauksessa reliabiliteetin todentamista hankaloittaa myös se, että varsinainen haastattelutilanne on vapaamuotoinen ja täten luonteeltaan aiheesta toiseen poukkoileva. Tässä tapauksessa on mahdollista, että tutkijan kokemattomuus tutkimuksen tekemisessä ja haastattelutilanteen kontrolloimisessa on vaikuttanut siihen, kuinka tarkalla tasolla haastattelun teemoja on kussakin haastattelutilanteessa noudatettu. Tosin tarkasti mietityn ja dokumentoidun haastattelurungon voidaan tulkita edesauttavan teemoihin liittyvien asioiden läpikäyntiä. Lisäksi tulosten toistettavuuden näkökulmasta on mahdotonta täydellä varmuudella sanoa, että kaikki haastatelluista käsittivät haastattelurungon kysymykset absoluuttisen samalla tavalla. Saadut tulokset kuitenkin puhuvat puolestaan, sillä haasteltujen antamat lausunnot teemojen asioista noudattivat sisällöltään samoja asiakokonaisuuksia. Toisekseen toisistaan eriävät tulokset ja niiden tulkinta tulisi yleisesti nähdä laadullisen tutkimuksen rikkautena.

Tämän tutkimuksen validiutta arvioidessa huomio tulisi ensimmäisenä kiinnittää järjestelmäevoluutiota selittävään teoriaan. Toisin sanoen siihen voidaanko Lehmanin esittämiä lainalaisuuksia ylipäänsä käyttää järjestelmäevoluution tutkimuksen viitekehyksenä. Muut järjestelmäevoluutiota tutkineet ovat ainakin perustaneet rakentamansa mallit Lehmanin esittämien havaintojen

pohjalle. Lisäksi Lehmanin lainalaisuuksiin perustunutta järjestelmäevoluution viitekehystä on hyödynnetty myös suhteellisen tuoreissa 2000-luvun tutkimuksissa. Tällöin voidaan perustellusti tulkita, että Lehmanin lainalaisuudet ovat käyttökelpoisia tutkittaessa järjestelmäevoluutiota tietyissä konteksteissa.

Toinen validiteettiin liittyvä vaara tässä tutkimuksessa on se, kuinka järjestelmäevoluution lainalaisuudet operationalisoitiin. Tällä tarkoitetaan, että lainalaisuuksia ei haastattelutilanteessa käytetty sellaisenaan vaan ne koostettiin tutkijan käsityksen mukaan neljään aihekokonaisuuteen. Eli kahdeksasta alkuperäisestä lainalaisuudesta päädyttiin haastattelurungossa käyttämään muutospaineeseen, laadun hallintaan, organisatorisiin piirteisiin ja prosessin kehittämiseen liittyviä aihekokonaisuuksia. Valittu ratkaisu oikoo Lehmanin antamaa käsitystä järjestelmäevoluution piirteistä, mutta toisaalta sen on tässä tutkimuksessa nähty helpottavan järjestelmäevoluution ymmärtämistä haastattelutilanteessa.

Saadut tutkimustulokset nojasivat vahvasti myös tieteellisessä kirjallisuudessa esitettyyn näkemykseen DevOps toimintamallista ja sen käytänteistä. Validiteetin kannalta tässä suhteessa haastavaa on, että DevOps on tieteellisessä maailmassa vielä varsin tutkimaton ilmiö. Tällöin on vaikea sanoa, onko DevOps todellisuudessa sitä, mitä sen kirjoitetun lähdemateriaalin puitteissa kerrotaan olevan. DevOpsin tutkimattomuudesta johtuen haastatteluun sisällytettiin osio, jossa haastatelluilta kysyttiin DevOpsin ja sen käytänteiden sisällöstä ja tarkoituksesta. Ilmiön piirteitä vahvistavan haastatteluosion voitaneen tulkita toimivan validiutta parantavana keinona. Toisaalta saadut tulokset osoittavat, että kirjallisuuden ja haastateltujen tulkinta DevOpsista ja sen käytänteistä on melko identtinen.

Yleistettävyyden näkökulmasta tutkimusasetelma on suppea. Ensinnäkin, kuten tuloksista käy ilmi, toteutetaan DevOps toimintamallia organisaatioissa hieman eri tavoin. Toisekseen, tässä tutkielmassa haastatellut henkilöt olivat suomalaisia, työskennellen suomalaisissa yrityksissä, suomalaisessa yrityskulttuurissa ja pääosin suomalaisten asiakkaiden kanssa. Näin ollen, mikäli samankaltainen tutkimus toteutettaisiin kansainvälisissä yrityksissä tai erilaisessa kulttuurissa, saattaisivat tulokset jossain määrin poiketa tässä tutkimuksessa saaduista. Tieteen tekemisen ja jatkotutkimuksen kannalta saatuja tuloksia voitaneen kuitenkin pitää vähintään vertailun arvoisena.

8 YHTEENVETO

Tässä tutkielmassa on tarkasteltu DevOps toimintamallin yhteyttä järjestelmäevoluutioon ja sen piirteitä kuvaaviin lainalaisuuksiin. Ilmiöiden välistä yhteyttä tutkittiin kirjoitetun tieteellisen tutkimuksen sekä empiirisen tutkimuksen avulla. Tutkimuksen tavoitteena oli selvittää, kuinka DevOps tuoreena ja vähäntutkittuna toimintamallina huomioi järjestelmäevoluution eli järjestelmien jatkuvan kehittymisen piirteitä. Ilmiöiden yhteyden tutkimiseksi asetettiin tutkimuskysymys:

- Miten DevOps toimintamallissa huomioidaan järjestelmäevoluutio ja sen erityispiirteet?

Kysymyksen asettelua lähdettiin purkamaan luvussa 2 selvittämällä ohjelmistotuotannon ja järjestelmäevoluution käsitteitä tieteellisen kirjallisuuden pohjalta. Ohjelmistotuotantoa kuvattiin yleismaailmallisesti esittäen kehityksen, ylläpidon ja operoinnin toimintoja, jotka yhteisesti luovat edellytykset järjestelmien kehittymiselle eli järjestelmäevoluutiolle. Järjestelmäevoluutiolla kuvataan ilmiötä, jossa järjestelmän ympäristötekijöiden muutos vaikuttaa järjestelmän vaatimuksiin ja täten aiheuttaa paineen itse järjestelmän muutokselle. Järjestelmän toimintaympäristön muutos edellyttää myös järjestelmämuutoksia, jotta järjestelmän kyky vastata uudenlaiseen toimintaympäristöönsä säilyy.

Järjestelmäevoluution vaikutuksia havainnollistettiin Meir Lehmanin esittämien järjestelmäevoluution lainalaisuuksien valossa. Lehmanin kahdeksan lainalaisuuden mukaan järjestelmät ovat evoluutionsa aikana alttiita jatkuvalle muutokselle (*jatkuva muutos*). Jatkuvan muutoksen seurauksena järjestelmän toiminnallisuutta tulee lisätä, mikä johtaa järjestelmien koon (*jatkuva kasvu*) ja kompleksisuuden kasvamiseen (*kasvava kompleksisuus*) ja tätä myötä myös järjestelmän laadun heikkenemiseen (*heikkenevä laatu*). Järjestelmäevoluutiolla nähdään edellä mainituista syistä myös ohjelmistotuotannon organisatoriseen hallintaan liittyviä tekijöitä. Jatkuvan muutoksen seurauksena järjestelmää kehittävältä organisaatiolta vaaditaan jatkuvaa työpanosta (*organisatorisen vakauden säilyttäminen*). Toisaalta muutoksesta johtuen organisaation tietämys (*tut-*

tuuden säilyttäminen) järjestelmäkokonaisuudesta vähenee. Tästä syystä lainalaisuuksien mukaan järjestelmäevoluutioprosessilta vaaditaan kykyä muuntautua (*itsesäätelevyys*) sekä tukeutua tietämystä lisääviin palautejärjestelmiin (*palautejärjestelmä*). Yhteisesti lainalaisuuksien todettiin hahmottavan evoluutioprosessin hallintaan liittyviä piirteitä ja epävarmuustekijöitä.

DevOpsin ja järjestelmäevoluution yhteyden selvittämistä jatkettiin luvussa 3 syventyen DevOps toimintamalliin sekä sen tarkoituksiin ja käytänteisiin. Kirjallisuuteen perustuen toimintamallin todettiin yhtenäistävän ohjelmistotuotannon kehityksen ja operoinnin toimintoja. Tutkielman johdannossa ja luvussa 2 kuvattiin, kuinka ohjelmistojulkaisu koetaan tavanomaisesti ohjelmistotuotantoketjun vedenjakajaksi. Toisin sanoen ohjelmiston kehitystoiminto kehittää järjestelmän tuotantovalmiiksi ja tämän jälkeen siirtää vastuun ohjelmiston julkaisusta ja operoinnista erilliselle operoinnin toiminnolle. Tämän todettiin aiheuttavan haasteita sujuvan ohjelmistotuotantotoiminnan kannalta. DevOpsissa kehityksen ja operoinnin roolien yhtenäistämisen tarkoituksena on poistaa ohjelmistojulkaisussa konkretisoituva vastuunsiirto.

DevOps toimintamallin mukainen yhteistoiminta nähdään yleisesti sen neljän eri käytänteen kautta. Yhteistyö käytänteen valossa toimintamallissa rakennetaan uudenlaisia tiimejä, jotka koostuvat sekä kehityshenkilöstöstä että operointihenkilöstöstä. Tällöin ohjelmistojulkaisussa ennen nähdystä vastuunsiirrosta aiheutuvat ongelmat poistuvat, koska kehityshenkilöstö ja operointihenkilöstö ymmärtävät paremmin toistensa toimintaa ja kohtaamia ongelmia. DevOps nojaa vahvasti myös automaatioteknologioihin. Toimintamallissa automaatiota sovellettavan järjestelmäinfrastruktuurin rakentamisessa ja hallinnassa sekä ohjelmiston julkaisuprosessissa. Automaation pääsääntöisenä tarkoituksena on sujuvoittaa järjestelmäinfrastruktuurin hallintaa sekä ohjelmistojulkaisuun kuluvaa aikaa. Automaatiolla on myös merkitystä järjestelmä laadun kannalta, kun toisaalta julkaisuprosessissa hyödynnetään yhä enemmän testiautomaatiota ja toisaalta automaation käyttö poistaa manuaalisesti tehtävien virheiden määrää. Teknologisten ratkaisujen näkökulmasta DevOps toiminnassa tukeudutaan myös erilaisiin monitorointityökaluihin. Monitoroinnilla pääasiassa tarkkaillaan järjestelmän toimintaa ja sen ympäristön vakautta. Näin ollen monitorointityökalut lisäävät DevOps tiimien tietämystä järjestelmästä ja sen tilasta. Koska DevOps toiminnassa yhdistetään uudella tavalla kehityksen ja operoinnin toimintoja, ei ohjelmistotuotannon tuloksellisuutta tai tehokkuutta voida mitata aikaisemmin totutuilla tavoilla. Kehitystä ei mitata suhteessa järjestelmän laatuun eikä operointia suhteessa järjestelmän vakauteen. Pikemminkin uutta kehitys- ja operointitoiminnon tuloksellisuutta mitataan suhteessa tuotannon läpimenoaikaan.

Tutkielman käsitteellis-teoreettinen osuus päätettiin luvussa neljä DevOpsin ja järjestelmäevoluution yhteyden pohtimiseen. DevOps toiminnassa ei suoraan oteta kantaa järjestelmäevoluutioon, sen hallintaan tai sen lainalaisuuksiin. Tosin kirjallisuusanalyysi paljasti joitakin sellaisia DevOps sidonnaisia tekijöitä, joilla nähtiin teoreettinen merkitys järjestelmäevoluution hallinnan näkökulmasta (ks. taulukko 3). Tieteellisen kirjallisuuden perusteella tulkitut

yhteydet DevOpsin ja järjestelmäevoluution hallinnan välillä olivat joissain tapauksissa ilmiselviä. Toisissa tapauksissa yhteyttä taas oli vaikeampi havaita. Tämän todettiin ennen kaikkea johtuvan siitä, että DevOps toimintamallia käsittelevää tutkimusta on toteutettu verrattain vähän. Järjestelmäevoluution hallinnan näkökulman ja DevOpsin yhdistävää tutkimusta ei ole toteutettu lainkaan. Tutkimustiedon puutteen ja osittain selkeiden yhteyksien vuoksi kirjallisuuteen perustuvaa analysointia jatkettiin laadullisella haastattelututkimuksella. Laadullisen menetelmän kuvaus on tarkemmin esitetty luvussa 5.

Haastattelutulokset osoittautuivat kirjallisuuden perusteella tehtyjä tulkintoja tarkemmiksi. Ensinnäkin tämä johtuu tutkimuksen tekemisen prosessista. Tällöin pitkähkön tutkimusprosessin seurauksena ymmärrys ja tietämyksen taso tutkituista ilmiöistä kasvavat jatkuvasti. Toisaalta aihepiirinä DevOps on tieteellisessä yhteisössä melko tutkimaton ilmiö. Tällöin aiheeseen perehtymisen voidaan tietenkin ajatella parantavan tulosten syvyyttä. Varsinaiset tutkimuksen kannalta olennaiset haastattelutulokset olivat kaksiulotteisia. Ensinnäkin ne tukivat ja vahvistivat vähäisen DevOps aiheisen tutkimuksen antamaa kuvaa toimintamallin hyödyistä, periaatteista ja käytänteistä. Toisekseen tulokset selkeyttivät näkemystä DevOpsin ja järjestelmäevoluution hallinnan välillä. Lisäksi tulokset toivat kirjallisuusanalyysin jatkeeksi uutta tietämystä ilmiöiden välisistä vaikutuksista. Tarkempi analysointi ja tulkinta DevOpsin ja järjestelmäevoluution hallinnan välisistä yhteyksistä on esitettynä luvuissa 6 ja 7.

DevOpsin puitteissa kirjallisuudessa tavatut neljä käytännettä (yhteistyö, automaatio, mittaaminen ja monitorointi) olivat haastattelutulosten mukaan konkreettisia osia toimintamallin hyödyntämisessä. Kirjallisuudesta poiketen DevOps nähtiin kuitenkin eri tavalla tavallisten kehittäjien ja ohjelmistokehitystoimintaa johtavien näkökulmista. Tulokset osoittivat, että lähempänä kehitystä toimivat henkilöt näkevät toimintamallin parantavan niin kutsuttua kehittäjäkokemusta. Johdon ja toiminnan ohjauksen näkökulmasta DevOps toiminta puolestaan näyttäytyi ennemminkin ohjelmistotuotantoketjua teollistavana toimintana, missä toimintamallin eri käytänteiden avulla pyritään optimoimaan ohjelmistotuotantoprosessin tehokkuutta ja ulosantia.

Järjestelmäevoluution hallintaan vaikuttavat tekijät ovat tulosten perusteella nähtävissä neljässä ulottuvuudessa. Ensinnäkin DevOps toiminnan vaikutukset nähtiin ohjelmistotuotannon *muutosreagointikyvyn* kannalta. Yhteistyökäytänteen, automaation ja monitorointiin voidaan nähdä edesauttavan kykyä ja nopeutta, joilla ohjelmistokehitystiimit kykenevät reagoimaan ja tekemään muutoksia kehitettäviin järjestelmiin. Toisaalta automaation mahdollistaman nopeuden nähtiin paikoin lisäävän tulevaisuuden muutostarpeiden ilmaantumisen tahtia. Lisäksi DevOpsin entistä vahvemman tukeutumisen teknologisiin ratkaisuihin nähtiin kasvattavan teknisen kehitysympäristön monimutkaisuutta.

Saadut tulokset havainnollistivat myös sitä, kuinka DevOps toiminta vaikuttaa *laadun hallintaan* järjestelmäevoluution näkökulmasta. Tulosten perusteella järjestelmien laaduntakaamiseen liittyvät vaikutukset olivat pääsääntöisesti positiivisia. Yhteistyökäytänteen nähtiin edesauttavan laadun hallintaa,

sillä uudenlainen tiimirakenne sisältää aikaisempaa kokonaisvaltaisempaa osaamista järjestelmäkokonaisuudesta. Tiimien koostuessa operoinnin ja kehityksen osaajista myös vastuu järjestelmän toiminnasta on keskittyneempää organisaation sisäisesti. Myös automaatiokäytänteen nähtiin vaikuttavan laadun takaamiseen. Tulokset osoittivat, että automatisoiduilla toiminnoilla voidaan poistaa manuaalisten toimien virhealttiutta. Lisäksi DevOpsin myötä yleistyneen testiautomaation käytön havaittiin edesauttavan laadukkaampien järjestelmien rakentamista. Saatujen tulosten mukaan monitorointikäytänteen hyöty näkyi laadunhallinnan näkökulmasta tehokkaampana järjestelmän tilan seurantana, jolloin heikkoon laatuun puuttuminen oli helpompaa ja nopeampaa.

DevOps toiminnan vaikutus järjestelmäevoluution hallintaan näkyy tulosten perusteella myös ohjelmistotuotannon *organisatorisen hallinnan* näkökulmasta. Muun muassa DevOps toiminnan uudenlainen tiimirakenne (yhteistyö) edesauttaa järjestelmäkokonaisuuksien tuttuuden säilymisessä, sillä osaamisen kokonaisvaltaisempi keskittyminen tiimien sisäisesti lisää tiimien ymmärrystä ja näkemystä. Tulokset osoittivat kuitenkin, että organisatorisen vakauden näkökulmasta uudenlainen tiimirakenne on kasvattanut yksittäisen kehittäjän vaatimustasoa. Tällöin uusien tarpeeksi osaavien työntekijöiden löytäminen on muuttunut kehitysorganisaatiolle entistä hankalammaksi. Automaation ja monitoroinnin entistä kattavampi hyödyntäminen osoittautui myös kehitysorganisaation tuttuutta säilyttäväksi tekijäksi. Tulosten perusteella automaatioon nojaava julkaisuprosessi mahdollistaa hyvin pienikokoisten järjestelmämuutosten toteuttamisen. Tällöin tuttuuden säilymisen näkökulmasta voidaan pienemmät muutokset ajatella suurempia helpommin kontrolloitaviksi. Monitorointi puolestaan osoittautui lähes välttämättömäksi vahvasti automatisoidussa teknisessä ympäristössä. Monitorointityökalujen hyödyntäminen nähtiinkin kehitystiimien tilannetietoisuutta parantavana menetelmänä.

Tulokset havainnollistivat myös DevOps toiminnan yhteyttä järjestelmäevoluution lainalaisuuksien mukaiseen *ohjelmistoprosessin kehittämisen* näkökulmaan. Prosessin kehittäminen näkyy tulosten perusteella sekä teknisenä kehittämisenä että ohjelmistotuotantotoiminnon optimointina. DevOps toimintamallin käytänteistä monitorointi näkyy teknistä kehittämistä edistävänä keinona. Monitoroinnin avulla tuotetaan palauteinformaatiota, jonka avulla säädetään järjestelmäteknistä ympäristöä. Mittaamisen käytänne taas osoittautuu koko ohjelmistotuotantoprosessia säätävänä toimenpiteenä. Mittaamisen avulla pyritään löytämään informaatiota ohjelmistoprosessin hidasteista. Mittaamisen kautta saatua tietoa voidaan tällöin käsitellä syötteenä tuotantoprosessin optimoinnin näkökulmasta. Tulosten analysointi osoittaa, että DevOps toiminnan monitoroinnin ja mittaamisen käytänteet näyttäytyvät Lehmanin järjestelmäevoluution palautejärjestelmiä kuvaavina menetelminä. Molempien käytänteiden tarkoituksena on toimia syötteenä tekniseen kehittämiseen tai tuotannolliseen optimointiin.

Kokonaisuudessaan saadut tutkimustulokset osoittavat, että DevOps toiminnalla ja erityisesti sen käytänteillä on useita vaikutuksia järjestelmäevoluution hallinnan kannalta. Toisaalta tämä tukee sitä, että Lehmanin jo useita

kymmeniä vuosia sitten esittämä näkemys järjestelmäevoluutiosta ja sen lainalaisuuksista on paikkaansa pitävä myös varsin modernissa ohjelmistotuotantomaailmassa. Toisaalta taas Lehmanin lainalaisuudet kuvaavat hyvin yleismaailmallisesti havaintoja ohjelmistotuotantoprosessin toiminnasta. Näin ajateltuna ei ole kovin merkillistäkään, että DevOps yleisesti todettuine hyvine käytänteineen pyrkii vastaamaan Lehmanin esittämiin geneerisiin totuuksiin.

Saatuja tuloksia voitaneen pitää hyödyntämiskelpoisina ohjelmistoalan tahojen keskuudessa. Haastattelututkimuksen tulokset sisältävät kolmen eri ohjelmisto-organisaation näkemyksiä DevOps toiminnasta sekä järjestelmäevoluution hallinnan menettelyistä. Tulokset muun muassa osoittavat käyttökelpoisia ajatuksia DevOpsin hyödyntämisestä, mikä voi antaa näkemystä toimintamallia vähemmän tunteville alan toimijoille. Toisaalta järjestelmäevoluution hallintaan liittyvät tulokset voivat tuoda DevOpsia tunteville tahoille uutta tietämystä toimintamallin hyödyistä, haitoista tai käytänteiden hyödyntämisestä.

Tieteellisestä näkökulmasta saadut tulokset vahvistavat jo kirjoitetun kirjallisuuden antamaa näkemystä DevOps toiminnasta. Toisaalta järjestelmäevoluution näkökulman yhdistäminen tuo myös uutta tutkimustietoa ohjelmistotuotantoa käsittelevään tieteelliseen kirjallisuuteen. Tutkimuksen laadullisena instrumenttina käytetty teemahaastattelu on järjestelmäevoluutiotutkimuksessa harvinainen käytäntö. Tästä syystä tutkimuksen laadullisen menetelmän voidaan nähdä tuovan uutta tietämystä myös tapaan tutkia järjestelmäevoluutiota.

Koska tässä tutkimuksessa järjestelmäevoluutiota ja DevOpsia on tutkittu varsin suppeassa kontekstissa (kolme suomalaista ohjelmistoyritystä), jatkotutkimuksen kannalta ilmiöitä olisikin mielenkiintoista tutkia erilaisissa ympäristöissä. Tällöin ilmiöitä voisi tutkia myös toisistaan irrallisina. DevOps on tulosten perusteella lupaavan tehokas toimintatapa, mutta sen hyödyntämisestä esimerkiksi globaalisti toimivassa ohjelmistotuotannossa ei ole kovinkaan paljon tutkimustietoa. Niin ikään järjestelmäevoluutiotutkimusta tulisi jatkaa erilaisissa ohjelmistoliiketoiminnan ympäristöissä, jotta tieteellisissä yhteisöissä päästäisiin lähemmäksi Lehmaninkin tavoittelemaa yleispätevää ohjelmistokehityksen ja -tuotannon teoreettista mallia.

LÄHTEET

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. IEEE Software. 33(3). 42-52.

Banker, R. D., & Slaughter, S. A. (1997). A field study of scale economies in software maintenance. *Management science*, *43*(12), 1709-1725.

Barry, E. J., Kemerer, C. F., & Slaughter, S. A. (2007). How software process automation affects software evolution: a longitudinal empirical analysis. Journal of Software Maintenance and Evolution: Research and Practice, 19(1), 1-31.

Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos Engineering. IEEE Software, 33(3), 35-41.

Bass, L., Jeffery, R., Wada, H., Weber, I., & Zhu, L. (2013, May). Eliciting operations requirements for applications. In Proceedings of the 1st International Workshop on Release Engineering (pp. 5-8). IEEE Press.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning et al. "Manifesto for agile software development." (2001).

Bennett, K. (1996). Software evolution: past, present and future. Information and software technology, 38(11), 673-680.

Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 73-87). ACM.

Benington, H. D. (1983). Production of large computer programs. IEEE Annals of the History of Computing, (4), 350-361.

Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H. C., & Smith, N. (2007). An empirical study of the evolution of an agile-developed software system. In Proceedings of the 29th international conference on Software Engineering (pp. 511-518). IEEE Computer Society.

Callanan, M., & Spillane, A. (2016). DevOps: Making It Easy to Do the Right Thing. IEEE Software, 33(3), 53-59.

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

Cook, S., Harrison, R., Lehman, M. M., & Wernick, P. (2006). Evolution in software systems: foundations of the SPE classification scheme. Journal of Software Maintenance and Evolution: Research and Practice, 18(1), 1-35.

Diel, E., Marczak, S., & Cruzes, D. S. (2016). Communication Challenges and Strategies in Distributed DevOps. In Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on (24-28).

Dittrich, Y. (2016). What does it mean to use a method? Towards a practice theory for software engineering. Information and Software Technology, 70, 220-231.

Dyck, A., Penners, R., & Lichter, H. (2015). Towards definitions for release engineering and DevOps. In Proceedings of the Third International Workshop on Release Engineering. IEEE Press.

Elliot, S. (2014). DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. International Data Corporation (IDC).

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. IT professional, 2(3), 17-23.

Fitzgerald, B., & Stol, K. J. (2014, June). Continuous software engineering and beyond: trends and challenges. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (pp. 1-9). ACM.

Glass, R.L. 2001. Frequently Forgotten Fundamental Facts about Software Engineering. IEEE Softw. 18, 3 (May 2001), 112-111.

Google Trends hakusanalla "DevOps". Google Trends. <https://www.google.com/trends/explore?date=all&q=DevOps>. 18.11.2016.

Hirsjärvi, S. & Hurme, H. (2000). Tutkimushaastattelu. Helsinki: Helsinki University Press.

Hirsjärvi, S., Remes, P. & Sajavaara, P. (2004). Tutki ja kirjoita. (10. Osin uudistettu painos). Helsinki: Kustannusosakeyhtiö Tammi.

Humble, J. & Molesky, J. (2011). Why enterprises must adopt DevOps to enable continuous delivery. Cutter IT Journal. 24(8).

Hüttermann, M. (2012). DevOps for developers. Apress.

IBM. (2013). DevOps: The IBM Approach.

IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

ISO/IEC. (2006). Software Engineering – Software Life Cycle Processess – Maintenance. Haettu 9.1.2016 osoitteesta http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1703974

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps?: A Systematic Mapping Study on Definitions and Practices. In Proceedings of the Scientific Workshop Proceedings of XP2016 (p. 12). ACM.

Kemerer, C. F., & Slaughter, S. (1999). An empirical approach to studying software evolution. IEEE Transactions on Software Engineering, 25(4), 493-509.

Kour, G., & Singh, P. (2016). Using Lehman's laws to validate the software evolution of agile projects. In Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on (pp. 90-96). IEEE.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076.

Lehman, M. M. (1996). Laws of software evolution revisited. In European Workshop on Software Process Technology(pp. 108-124). Springer, Berlin, Heidelberg.

Lehman, M. M. (1991). Software engineering, the software process and their support. Software Engineering Journal, 6(5), 243-258.

Lehman, M. M., Ramil, J. F., & Kahen, G. (2000). Evolution as a noun and evolution as a verb. In SOCE 2000 Workshop on Software and Organisation Co-evolution. 9. 31.

Lehman, M. M., & Ramil, J. F. (2001). Rules and tools for software evolution planning and management. Annals of software engineering, 11(1), 15-44.

Lehman, M. M., & Ramil, J. F. (2003). Software evolution—background, theory, practice. Information Processing Letters, 88(1), 33-44.

Lientz, B.P. & Swanson, E. (1981). Problems in application software maintenance. Communications of the ACM 24 (11), 763-769.

Liu, Y., Li, C., & Liu, W. (2014). Integrated solution for timely delivery of customer change requests: A case study of using DevOps approach. International Journal of U-and E-Service Science and Technology, 7(2), 41-50.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2015). Dimensions of DevOps. In International Conference on Agile Software Development (pp. 212-217). Springer International Publishing.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016). An Exploratory Study of DevOps Extending the Dimensions of DevOps with Practices. *ICSEA 2016*, 104.

McCarthy, M. A., Herger, L. M., Khan, S. M., & Belgodere, B. M. (2015). Composable DevOps: Automated Ontology Based DevOps Maturity Analysis. In Services Computing (SCC), 2015 IEEE International Conference on (pp. 600-607). IEEE.

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. (2005). Challenges in software evolution. In Eighth International Workshop on Principles of Software Evolution (IWPSE'05) (pp. 13-22). IEEE.

Perry, D. E. (1994). Dimensions of software evolution. In Software Maintenance, 1994. Proceedings., International Conference on (pp. 296-303). IEEE.

Roche, J. (2013). Adopting DevOps practices in quality assurance. Communications of the ACM, 56(11), 38-43.

Royce, W.W., (1970). Managing the development of large software systems: concepts and techniques. 1970 WESCON technical papers. Vol. 14. 723.

Schultze, U., & Avital, M. (2011). Designing interviews to generate rich data for information systems research. Information and Organization, 21(1), 1-16.

Sindhgatta, R., Narendra, N. C., & Sengupta, B. (2010). Software evolution in agile development: a case study. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion (pp. 105-114). ACM.

Smeds, J., Nybom, K., & Porres, I. (2015). DevOps: a definition and perceived adoption impediments. In International Conference on Agile Software Development (pp. 166-177). Springer International Publishing.

Sommerville I. (2016). Software Engineering 10th Edition (International Computer Science). Pearson Education Limited. Essex, England

Stillwell, M., & Coutinho, J. G. (2015). A DevOps approach to integration of software components in an EU research project. In Proceedings of the 1st International Workshop on Quality-Aware DevOps (pp. 1-6). ACM.

Swartout, P. (2014). Continuous Delivery and DevOps–A Quickstart Guide. Packt Publishing Ltd.

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

Waller, J., Ehmke, N. C., & Hasselbring, W. (2015). Including performance benchmarks into continuous integration to enable DevOps. ACM SIGSOFT Software Engineering Notes, 40(2), 1-4.

LIITE 1 HAASTATTELURUNKO

HAASTATTELUTEEMA 1: Taustatiedot

- Työhistoria?
- Nykyinen
    - titteli ja työnkuva?
    - Miten DevOps näkyy työnkuvassa?

HAASTATTELUTEEMA 2: DEVOPS

Alateema 2.1.: Suhtautuminen DevOpsiin

- Yleistä DevOpsista:
    - kuinka kuvailisit DevOpsia?
    - Koetko, että DevOps tuo jotain uutta kehitysprosessiin? Mitä?

Alateema 2.2.: DevOps käytänteet

- Yhteistyö (Collaboration): DevOps nimensä mukaisesti yhdistää kehityksen ja ylläpidon toimintaa.
    - Kuvaile miten toimintojen yhdistäminen näkyy teidän toiminnassanne?
        i. Millä tavoin kehitysprosessi on järjestetty tiimeittäin? Miksi on näin?
        ii. Millä tavoin kehityksessä ja ylläpidossa kommunikoidaan?

- Automaatio (Automation): Tutkimusten mukaan DevOpsissa korostetaan mahdollisimman pitkälle vietyä automatisointia.
    - Kuvaile automaatiota teidän prosessissa (DevOpsissa)?
        i. Mitä kaikkea ohjelmistoprosessissa on automatisoitu? Mitä työvälineitä?

ii. Onko automatisoinnin tavat muuttuneet DevOpsin "käytöönoton" myötä? Tehdäänkö automaation avulla eri asioita kuin ennen? Jos tehdään niin mitä?

- Mittaaminen (Measurement): Tavallisesti (vanhanaikaisesti) on ajateltu, että ohjelmistotuotannossa kehityksen tehokkuuden mittarina on ”uuden” tuottaminen ja ylläpidon mittarina on toimintavarmuuden ylläpitäminen.
    - Miten tiimien tehokkuutta/tuloksellisuutta mitataan teidän prosessissa (DevOpsissa)? Esimerkkejä mittareista?
    - Onko mittaaminen muuttunut DevOpsin myötä?

- Monitorointi (Monitoring): Tutkimusten mukaan DevOpsissa korostetaan ohjelmiston/järjestelmän monitorointia.
    - Kuvaile millä tavalla teidän prosessissa järjestelmää monitoroidaan?
        i. Mitä asioita ohjelmistosta monitoroidaan? Millaista palautetta monitoroinnilla saadaan?
        ii. Miksi monitorointia tehdään?
        iii. Missä muodossa palaute on? / Miten ja kuka palautteeseen pääsee käsiksi?
        iv. Tehtiinkö monitorointia ennen DevOpsia?

- Olivatko neljä näkökulmaa tarpeeksi kattavat kuvaamaan DevOpsia?
    - Haluaako haastateltava nostaa jotain sellaista mitä ei hänen mielestään käsitelty?

HAASTATTELUTEEMA 3: Järjestelmäevoluution hallinta

Alateema 3.1: Muutospaineen huomiointi (Jatkuva muutos ja jatkuva kasvu)
"Tutkimuksissa on havaittu, että järjestelmävaatimusten muuttuminen on vääjäämätöntä. Tämä tietenkin tarkoittaa myös sitä, että järjestelmää täytyy muuntaa jatkuvasti".

- Miten teillä ohjelmistokehityksessä (DevOps toiminnassa) huomioidaan vaatimusten muuttuminen?
  - Onko mielestäsi DevOps tuonut tähän jotain uutta?
  - Mikä on teknologian ja työkalujen rooli vaatimusmuutosten huomioinnissa?

- *Lisäkysymykset ja väitteet:*
  - Näkisitkö, että kehityksen ja ylläpidon toimintojen tiivistäminen parantaa kykyä vastata muutokseen? Miksi/Miksi ei?
  - Onko automaatiolla vaikutusta nopeuteen, jolla vaatimusten muutoksiin voidaan reagoida? Minkälainen vaikutus?
  - Koetko, että järjestelmän monitorointiratkaisut vaikuttavat vaatimusten muuttumisen huomiointiin? Miten?

Alateema 3.2: Laadun hallinta (Kasvava kompleksisuus ja heikentyvä laatu)
"Tutkimusten mukaan, ajan kuluessa ja järjestelmän kasvaessa se monimutkaistuu ja näin ollen sen laatu heikkenee".

- Millä tavalla teidän prosessissa (DevOpsissa) huomioidaan, että järjestelmä on laadukas tai pysyy laadukkaana?

- *Lisäkysymykset ja väitteet:*
  - Näkisitkö, että kehityksen ja ylläpidon tiivistämisellä on merkitys laadukkuuden takaamisessa? Miksi/Miksi ei?
  - Koetko, että automaation hyödyntäminen vaikuttaa ohjelmiston laatuun? Millä tavalla?
  - Onko järjestelmän monitoroinnilla merkitystä laadun hallinnassa? Minkälainen merkitys? Miten saatua palautetta hyödynnetään?

Alateema 3.3: Organisatoriset piirteet (Tuttuuden säilyttäminen ja organisatorisen vakauden säilyttäminen)
Hallitun järjestelmäevoluution kannalta on tärkeää, että ohjelmistotuotantoprosessissa korostetaan jatkuvaa oppimista. Lisäksi myös työn tehokkuuden takaaminen on merkittävää.

- Miten ohjelmistoprosessissa (DevOpsissa) suhtaudutaan jatkuvaan oppimiseen? Miksi näin?
- *Lisäkysymykset ja väitteet:*
  - Näkisitkö, että kehityksen ja ylläpidon tiivistämisellä on merkitystä oppimiseen? Miksi?
  - Edistääkö järjestelmän monitorointiratkaisujen käyttö oppimista? Miten? Miksi ei?

- Organisaation kannalta on toki merkittävää, että työtä tehdään tehokkaasti. Miten teidän ohjelmistoprosessissa taataan, että työ on tuottavaa?
- *Lisäkysymykset ja väitteet:*
  - Minkälainen merkitys tehokkuuden/tuottavuuden mittaamisella on?
  - Automaatio parantaa tuottavuutta. Millainen merkitys automaation käytöllä tiimien tehokkuuteen?

Alateema 3.4: Ohjelmistotuotantoprosessin optimointi (Itsesäätely ja palautejärjestelmä)
Järjestelmän evoluution kannalta on tärkeää, että ohjelmistotuotantoprosessia kehitetään jatkuvasti.

- Millä tavoin teillä toimitaan ohjelmistoprosessin kehittämiseksi tai optimoimiseksi?
- *Lisäkysymykset ja väitteet:*
  - Millainen merkitys tehokkuuden/tuottavuuden mittaamisella on ohjelmistoprosessin optimointiin?
  - Näkisitkö, että järjestelmän monitoroinnilla on ohjelmistoprosessin kehittämiseen? Millainen merkitys?

- Mistä kaikkialta ohjelmistoprosessiin tulee palautetta?
- *Lisäkysymykset ja väitteet:*
  - Mikä on loppukäyttäjien rooli palautteen antajina?
  - Koetko, että kehityksen ja ylläpidon tiivistäminen parantaa palautteen saantia? Miksi?
  - Miten järjestelmän monitorointiratkaisut parantavat palautteen saantia?