Tampereen ammattikorkeakoulu

DevOps-toimintamalli

Janne Forsell

OPINNÄYTETYÖ

Syyskuu 2020

Tietojenkäsittelyn tutkinto-ohjelma

Game Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Game Production

FORSELL, JANNE:
DevOps-toimintamalli

Opinnäytetyö 35 sivua, joista liitteitä 2 sivua
Marraskuu 2020

Digitalisaation myötä kilpailu ohjelmistoliiketoiminnassa on kiristynyt. Ohjelmistoyritykset hakevat säästöjä ja kilpailuetua uusista toimintatavoista. Toimintatavat kehittyvät nopeaa ja alati kiihtyvää vauhtia. Asiakkaat vaativat uusia toiminnallisuuksia entistä nopeammassa syklissä, eivätkä perinteiset tavat kehittää ohjelmistoja pysty vastaamaan uusiin haasteisiin.

DevOps on vuodesta 2009 lähtien kehittynyt ohjelmistokehityksen malli, joka ammentaa lean-ajattelusta. DevOps pyrkii kaatamaan perinteisten ohjelmistoprojektinhallintamallien pystyttämiä aitoja organisaation osien välillä. Yhdistelemällä toimintoja syntyy monialaisia tiimejä, jotka DevOps-ideologian mukaan tehostavat ohjelmistokehitystä.

Tässä opinnäytetyössä käytiin läpi DevOpsin keskeiset periaatteet, joita sovelletaan Innofactor Oyj:n projektiliiketoiminnassa. Innofactorin DevOps-toimintamallia päivitettiin tämän raportin havaintojen mukaisesti; varsinkin automaatioon liittyvät asiat olivat kehittyneet sitten edellisen version julkaisun. Toimintamalliin lisättiin ensimmäistä kertaa maininta pilvi-infrastruktuurin automaattisesta varaamisesta.

DevOps-työkalut olivat kehittyneet harppauksin: Innofactorin teknologiakumppani Microsoftin DevOps-ratkaisut olivat kypsyneet hyväksi vaihtoehdoksi entiselle työkalukokoelmalle. Azure DevOps tarjosi nyt mahdollisuuden käyttää yhtä valmista ohjelmistoa usean ohjelmiston yhdistelyn sijaan. Vaikka Azure DevOpsin ominaisuuksissa havaittiin useita puutteita, mahdollisuus yhden työkalun ratkaisuun todettiin hyväksi kehityssuunnaksi.

Avainsanat: devops, azure, lean, automaatinen testaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Tietojenkäsittelyn tutkinto-ohjelma
Game Production

FORSELL, JANNE:
DevOps Standard Operating Procedure

Bachelor's thesis 35 pages, appendices 2 pages
November 2019

Digital transformation has caused the demand, and supply, of software to sky-rocket in recent years. Businesses are looking at new ways of working for an edge over the competition. Ability to react to customer demand fast is what makes and breaks services these days. New technologies and ways of working have emerged in the software industry over the last decade – collectively these new practices are known as DevOps.

DevOps is a set of lean enterprise practices applied to software business given a name in 2009. Primary problem DevOps is trying to solve is lack of communication between individuals who are supposedly working towards a common goal: a successful software. At the centre of DevOps is a cross functional team, formed by combining functions of previous organisational models.

This thesis work went through core principles and concepts of DevOps, as applied in Innofactor Plc project business. The standard operating procedure regarding DevOps within the company was updated according to the findings. Discussing automating public cloud infrastructure allocation was a new addition to the document.

DevOps-tools had improved significantly since Innofactor previously reviewed the operating procedure. Microsoft, Innofactor’s technology partner, had come up with many improvements to their Azure DevOps -suite. The suite now offers all-in-one-solution as opposed to a fragmented collection of software from different vendors, which makes for a smoother developer experience.

Key words: devops, azure, lean, test automation

SISÄLLYS

1 JOHDANTO

1.1 Tavoite ja tarkoitus

Tämän opinnäytetyön tavoite on päivittää toimeksiantajan, Innofactor Oyj:n, projektiliiketoiminnan DevOps-toimintamalli. Kuvaus tiimien toimintamallista tarvitaan usein tarjouksen liitteeksi. Julkisissa hankinnoissa tarjouspyyntöön liittyy pisteytettäviä tai poissulkevia vaatimuksia, joiden täyttymisen voi tällaisella mallilla osoittaa (Hannola 2015, 17). Dokumentoitu toimintamalli on myös tiimien apuna, kun pohditaan teknologiavalintoja ja yhteisiä pelisääntöjä.

Innofactor haluaa julkaista toimintamallin lähitulevaisuudessa. Innofactor uskoo omaan tapaan tehdä töitä ja tunnistaa tässä yhden mahdollisuuden parantaa mielikuvaa yhtiöstä: avoimuutta arvostetaan (Hirvilampi 2019, 22). Positiivinen julkisuuskuva on etu, kun kilpaillaan osaajista ja liiketoimintamahdollisuuksista.

Opinnäytetyössä on tarkoitus käydä läpi nykyinen, vuonna 2016 Innofactorin sisäisesti julkaistu toimintamalli. Innofactor on kehittänyt toimintaansa ketterästi ja määrätietoisesti: heikoksi havaituista toimintatavoista on luovuttu ja kokeiluista syntyy uusia. Kuvattu toimintamalli ei vastaa enää parhaita tunnettuja käytäntöjä.

Scrum on **Takeuchin** ja **Nonakon** (1986) kuvailema tapa kehittää ohjelmistoja, jossa keskiössä on monialainen tiimi. Tiimi vastaa ohjelmiston kehityksestä alusta loppuun. Scrum-tiimi työskentelee iteratiivisesti: palaute asiakkaalta ja palautteeseen reagoiminen on jatkuvaa. Nykymuotoista scrumia on kehitetty 1986 kuvaillun mallin perusteella vuodesta 1993 alkaen: virallisen scrum-oppaan ensimmäinen laitos julkaistiin 2010 (Lynch 2019).

Innofactorin projektiliiketoiminnassa pyritään noudattamaan ketterää scrum-projektinhallintamallia. Tuotettavassa toimintamallissa yhdistyvät scrum ja DevOps; DevOpsin yhteensovittaminen muiden projektinhallintamallien kanssa rajautuu pois opinnäytetyön toteutuksen laajuudesta.

Opinnäytetyössä päivitettävä toimintamallidokumentti ja tämä opinnäytetyöraportti mukailevat vuoden 2016 version rakennetta. Opinnäytetyöraportissa keskitytään DevOps-konseptien teoreettiseen pohjaan, toimintamallidokumentissa taas annetaan konkreettisia suosituksia.

1.2 Menetelmät

Tämän opinnäytetyön tarkoituksena on suositella DevOpsiin liittyviä toimintatapoja ja esittää ne tavalla, jonka ymmärtämiseen ei vaadita syvällistä asiantuntemusta. Numeerista vastausta näin aseteltuun tutkimuskysymykseen ei voida mitata.

Opinnäytetyö on toteutettu tapaustutkimuksena, jossa toimeksiantajan nykyistä toimintatapaa verrataan kirjallisuuteen (Luotio 2016). Annetut suositukset validoidaan henkilöstökyselyllä: tämän raportin sisällöstä muodostetaan lausumia, joihin kyselyssä pyydetään ottamaan kantaa.

2 MITÄ DEVOPS ON?

2.1 Taustat

Tiettävästi ensimmäisen kerran termiä DevOps käytti belgialainen **Patrick Debois** vuonna 2009 (Mezak 2018). Aiemmin samana vuonna yhteisöpalvelu Flickr'in työntekijöiden konferenssiesitys inspiroi Debois'in perustamaan oman konferenssinsa Devopsdaysin (Allspaw & Hammond 2019).

Vallalla olleessa tavassa kehittää ohjelmistoja Debois'ta turhautti eri tiimien välinen kulttuurinen etäisyys ja kommunikaation puute. Erityisesti kehittäjien ja ylläpitäjien välinen kommunikaatio oli olematonta ja toimintatavat täysin erilaisia. (Mezak 2018.)

Prosessin osien erottaminen on massatuotannon alkuajoilta periytyvä käytäntö. Vuonna 1908 **Henry Ford** jakoi autotehtaansa tuotantolinjan 84 eri vaiheeseen. Fordin tehtaalla työntekijät koulutettiin suorittamaan vain yhtä näistä tehtävistä. Työntekijät pystyivät suoriutumaan tehtävistä paremmin ja nopeammin, ja systemaattisista vioista päästiin eroon. (Esmaeilian, Mahmad, Sulaiman & Ismail 2020, 2.)

Fordin tuotantojärjestelmässä auto kulki hihnalla ja työntekijöiden piti sovittaa työtahtinsa linjan nopeuteen (Esmaeilian ym. 2020, 3). Maailmansotien jälkeen japanilainen autovalmistaja **Kiichiro Toyota** kehitti oman, Fordin järjestelmään perustuvan tuotantolinjansa. Toyotalla tuotteen ”työntäminen” linjalla nähtiin mahdollisena hukkana; sen sijaan linjan loppupään tuli ”vetää” linjaa todellisen tarpeen mukaan. Tätä periaatetta kutsutaan oikea-aikaisuudeksi. (Patoranta-Lötjönen 2012, 14–17.)

Huolimatta Toyotan innovaatioista, alussa ohjelmistokehityksen mallit perustuivat samoille periaatteille, kuin Henry Fordin tuotantolinja. Työtä tehtiin etupainoisesti, työvaiheiden seuratessa toisiaan: määrittely, suunnittelu, toteutus, testaus ja

käyttöönotto. Ongelmien ilmetessä vasta testaus ja käyttöönottovaiheessa valtavasta määrästä työtä aiemmissa vaiheissa tulee silkkaa hukkaa. Tätä vesiputoukseksi kutsutta mallia on kritisoitu jo 1970-luvulla. (Royce 1970.)

Ohjelmistoalan onnettomuudeksi Yhdysvaltain puolustushallinto kuitenkin standardoi vesiputousmallin vuonna 1985 ja velvoitti alihankkijansa käyttämään sitä (Department of Defense 1985). Yhdysvaltain puolustushallinnon arvokkaat sopimukset vakiinnuttivat vesiputousmallin aseman yleisenä standardina ympäri maailmaa.

Ohjelmistobisneksen keskittyessä vesiputoukseen perinteinen teollisuus jatkoi Toyotan viitoittamalla tiellä. Yhdysvaltalaisessa Massachusetts Institute of Technologyssa kehitettiin termi *lean* kuvaamaan periaatteita Toyotan menestyksekkään tuotantojärjestelmän takana. Leanin kehittäjät uskoivat Toyotan mallin soveltamisen parantavan kaikkia tuotantoprosesseja. (Womack, Jones & Roos 1990, 4.) Lean kehittyi ja nykyisin tunnettu lean-ajattelu julkaistiin vuonna 1996. Sittemmin mallin esitelleestä Lean Thinking -kirjasta on julkaistu useita päivitettyjä laitoksia. (Womack & Daniel 2003.)

Ennen vuotta 2001 vesiputousmallia virtaviivaisempia toimintatapoja kutsuttiin kevyiksi menetelmiksi. Vuonna 2001 kevyitä menetelmiä käyttävien yritysten edustajia kokoontui keskustelemaan ohjelmistokehityksen tulevaisuudesta. Näiden keskustelujen perusteella syntyi Ketterän ohjelmistokehityksen julistus, joka on sittemmin muovannut käsitystä ohjelmistokehityksestä. (Haapala 2016, 7.)

Ketterät menetelmät eivät kuitenkaan riittäneet tyydyttämään Patrick Debois'ta. Ketterän ohjelmistokehityksen julistuksen sisältö oli projektinhallinnalle mullistava uudistus, mutta pelkkä projektinhallinnan uudistaminen oli vasta alkua. DevOps on Lean-ajattelun soveltamista ohjelmistokehitykseen. Innofactorin käyttämä scrum-malli soveltuu DevOpsin aisapariksi erinomaisesti: scrum ammensi Toyotan tuotantojärjestelmän opeista jo ennen Lean-ajattelun syntyä.

2.2 Määritelmät

DevOps on terminä kehnosti määritelty: kirjallisuus ei esitä yhtenevää määritelmää. Yhtenevän määritelmän puutteessa myöskään DevOpsin sisällöstä ei voida olla yksimielisiä. (Rütz 2019, 11; Zazour, Alhammand & Alenezi 2019, 4854; Erich & Amrit 2017, 2.)

Laajinta suosiota nauttii DevOpsin määrittely organisaatiokulttuurin muutoksena, jossa kommunikaatiota haittaavia raja-aitoja kaadetaan (Erich & Amrit 2017, 5; Rütz 2019, 5). Tämän määritelmän taustalla on Patrick Debois'in alkuperäinen havainto tiimien välisestä etäisyydestä ja kommunikaation puutteesta. Ylläpito- ja kehitystiimien yhdistäminen on DevOpsin mukainen toimintatapa. Näin syntyy scrumin kuvaama monialainen tiimi; tämä on yksi syy näiden kahden menetelmän hyvälle yhteensopivuudelle.

Suurissa yrityksissä ja projekteissa työntekijöiden yhdistäminen yhdeksi tiimiksi ei toimi. Scrum-pioneerit **Schwaber ja Sutherland** (2020) pitävät tiimin maksimikokona yhdeksää henkeä. DevOpsin käyttöönotto suuremmissa organisaatioissa vaatii kokonaisvaltaisen kulttuurin muutoksen.

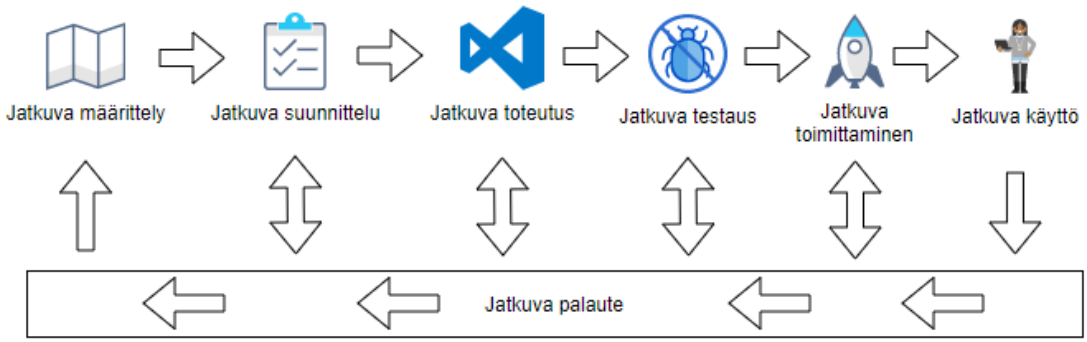
Organisaatiossa kaikki tiimit ja yksilöt täytyy saada työskentelemään kohti samaa päämäärää. Tämän saavuttamiseksi on pidettävä huolta, ettei organisaation sisällä ole ristiriitaisia kannustimia. Kaiken toiminnan tulee lean-ajattelun mukaisesti pyrkiä tuottamaan jotakin tunnistettua arvoa. Arvoa tuottamattomaan toimintaan usuttavat kannustimet pitää poistaa. (Erich & Amrit 2017, 6.)

Luottamuksen puute eri tiimien ja roolien välillä on kommunikaation este. Kehittäjät eivät saa kokea testaajia ja ylläpitoa vihollisina, jotka aiheuttavat harmia ja lisätyötä. (Erich & Amrit 2017, 6; Rütz 2019, 6.) Tiimien välinen juopa on peruja ei-ketteristä projektinhallintamalleista. Näissä malleissa kullakin osa-alueella on selkeä omistaja ja vastuu, jonka on havaittu aiheuttavan syyttelyä ja vastuunpakoilua. (Maroukian & Gulliver 2020.)

DevOpsiin liitetään myös joukko teknisiä ratkaisuja, joilla automatisoidaan ohjelmiston elinkaareen liittyviä rutiinitoimenpideitä. Näiden yksitoikkoisten toimenpiteiden automatisointi vähentää työtä ja inhimillisen virheen mahdollisuutta. (Erich

& Amrit 2017, 1.) Tekniset ratkaisut kehittyvät vauhdilla, ja osittain tai kokonaan automatisoitavien tehtävien osuus lisääntyy jatkuvasti (Zarour ym. 2019, 4854).

DevOps on myös tapa organisoida työtä. Vesiputousmallin peräkkäiset työvaiheet ovat mielekästä työtä. Tarve määritellä, suunnitella, toteuttaa, testata ja julkaista uusia versioita ei häviä vaihtamalla ketteriin toimintatapoihin. Sen sijaan, että työvaiheet ovat peräkkäisiä, DevOps tekee niistä jatkuvia (Jumani, Siddique & Saikh 2020, 553). Työvaiheitten jatkuvuutta havainnollistetaan *kuviossa 1*.



KUVIO 1. DevOps toimintamallissa työvaiheet ovat jatkuvia ja tukevat toisiaan.

DevOps-ratkaisuilla pyritään nopeuttamaan sovelluskehitystä. Käyttäjän tarpeen tunnistamisen ja tarpeeseen vastaavan ominaisuuden julkaisun välistä aikaa pyritään lyhentämään; kilpailijaa nopeampi julkaisusykli on kilpailuetu. (Erich & Amrit 2017, 1.) DevOps-liikkeen aloittanut yhteisöpalvelu Flickr'in toimintatapa tähtäsi kilpailuedun saavuttamiseen (Allspaw & Hammond 2019). Sittemmin muun muassa suoratoistopalvelu Netflix on menestynyt käyttäen DevOps-periaatteita (Rütz 2019).

DevOps on verrattain uusi, vielä muotoaan hakeva käsite, eikä sen kehitystä ohjata keskitetysti. Tarkan määrittelyn sijasta DevOps on helpompi käsittää ajattelumallina ja joukkona hyväksi havaittuja käytäntöjä (Maroukian & Gulliver 2020). Koska valmista sapluunaa ei ole, kunkin organisaation täytyy kehittää oma tapansa toimia (Erich & Amrit 2017, 17).

3 DEVOPS: TAPA ORGANISOIDA OHJELMISTOTYÖTÄ

3.1 Periaatteet

Tehtävän työn pitää perustua tunnistettuun arvoon (Jumani ym. 2020, 560). Tyypillisesti arvoa tuottaa loppukäyttäjän tarpeen tyydyttäminen. Jatkuva käyttäjätestaus ja analytiikka tunnistaa arvon nopeasti ja reagointiaika on merkittävästi perinteisiä menetelmiä lyhyempi.

Omistajuuden laajentaminen pienestä osasta koko prosessiin helpottaa arvon tunnistamista. Kehitysprosessin ongelmat ja pullonkaulat löytyvät helpommin, kun ne eivät ole jonkun muun ongelma. Käyttäjien ja kehitystiimin välissä on vähemmän välikäsiä: sovelluksen toiminnasta kerätään juuri oikeaa dataa ja palaute ei kulje rikkinäisen puhelimen kautta. (Jumani ym. 2020, 553.)

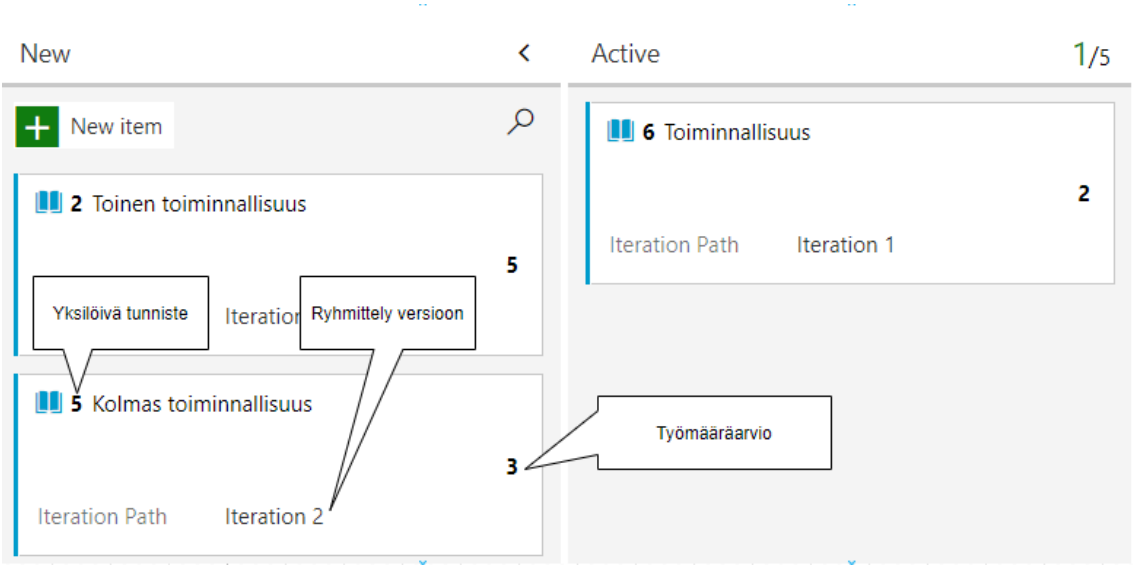
Työ pitää jakaa hallittaviin, itsenäisiin kokonaisuuksiin. Työllä on oltava konkreettinen tavoite, joka voidaan yhdistää sen tuottamaan arvoon. Jos työtä tehdään tietämättä syytä, toimitaan kuin Fordin tuotantolinjassa: työskennellään työn itsensä vuoksi. Kun mikään tarve ei toimi työn veturina, on työ lean-ajattelun tarkoittamaa hukkaa.

Työ pitää organisoida siten, että sen etenemistä pystytään seuraamaan. Riskienhallinta kuuluu myös DevOpsiin ja ketteriin menetelmiin. (Schwaber & Sutherland 2020.) Seuranta helpottaa ennakointia ja hyvä ennakointi vähentää kiirettä. Kiire on DevOpsin jatkuvan parantamisen ideologian pahin vihollinen; kiire kasvattaa laatuvelkaa, huonosti hoidetut pohjatyöt hidastavat tulevaa vauhtia ja lisäävät kiirettä. Mitä aiemmin ongelma tunnistetaan, sitä helpompi siihen on puuttua.

DevOps-mallissa jokainen on vastuussa prosessin parantamisesta; käytäntö periytyy Toyotan tuotantolinjalta, jossa työntekijät velvoitettiin ilmoittamaan kaikista havaitsemistaan puutteista ja parannusehdotuksista (Patoranta-Lötjönen 2012, 10). Tarpeen vaatiessa työn mielekkyys ja tarpeellisuus pitää kyseenalaistaa. Tarvittaessa työ keskeytetään ja palautetaan uudelleen määriteltäväksi tai suunniteltavaksi.

3.2 Työjonojen hallinta

Työn hallintaan tarkoitetuissa työkaluissa aiemmin kuvailtua työn yksikköä kutsutaan asiaksi (engl. *Backlog item*). Kuhunkin asiaan liittyy yksilöivä tunniste, jolla asia linkitetään tehtävään työhön. Yksilöivä tunniste linkittää koodimuutokset, asian kuvaamassa toiminallisuudessa havaitut laatupoikkeamat ja toiminnallisuuteen liittyvät versiopäivitykset toisiinsa. (Microsoft 2019; Atlassian 2020.) Työjonojen hallintaan liittyviä käsitteitä esitellään *kuviossa 2*.



KUVIO 2. Asioita työjonossa Microsoftin Azure DevOps -työkalussa.

Asiat voivat olla suuri- tai vähätöisiä. Asioiden hallintaan tarkoitetut työkalut tukevat työmääräarvioita, joka on kehitystiimin arvio asian toteuttamiseen vaadittavasta työstä (Schwaber & Sutherland 2020; Tyynismaa 2014, 68). Kehitystiimi valitsee itse käyttämänsä arviointitavan: tyypillisesti käytetään abstrakteja, asioiden välillä vertailukelpoisia pisteitä. Kehityksen vauhtia mitataan käyttäen samaa yksikköä: yleensä yksiköksi ei kannata valita työpäivää tai -tuntia.

Asioita pitää pystyä ryhmittelemään jollakin tavalla. Ryhmittelemällä asiat jaetaan julkaisuihin ja versioihin. (Rintala 2015, 21). Näiden versioiden tulisi linkittyä ohjelmistosta julkaistaviin versioihin, jolloin työtä on helpompi seurata. Jos versionumeroinnista ei pidetä huolta, on vaikea tietää, mitkä ominaisuudet on julkaistu. Väärin versioitujen laatupoikkeamien tapauksessa ei voida tietää mistä on kyse:

eikö korjaavaa versiota ole asennettu; onko korjaus epäonnistunut; vai onko kyse uudesta, samankaltaisesta ongelmasta?

Projektin asiat järjestetään työkalussa jonoksi, jossa jonon kärkeen sijoitetaan kriittisimmän asiat. Kun edellinen asia valmistuu, kehittäjä alkaa työstämään työjonon kärjessä olevaa asiaa. Aloitetut asiat poistuvat työjonosta, eikä niiden yli pitäisi priorisoida toista asiaa. (Schwaber & Sutherland 2020; Rintala 2015, 44.) Työjonoihin liittyvissä yksityiskohdissa on pieniä eroja eri ketterien menetelmien välillä.

4 DEVOPS: YLEISPÄTEVÄ TÄSMÄLÄÄKE OHJELMISTOKEHITYKSEN ONGELMIIN?

4.1 Käyttöönotto: tavoitteet, toiveet, huolet ja murheet

Digitalisaatio on ilmiö, jossa tietotekniikan käyttö yleistyy kaikilla elämän osa-alueilla; bisnes- ja ansaintamallien tulee sopeutua uuteen sähköiseen ympäristöön (Mergel, Edelmann & Haug 2019). Netflix on tunnettu esimerkki digitalisaation myötä muuttuneesta liiketoimintamallista. Ensin yhtiö siirsi elokuvavuokrauksen kivijalasta nettikauppaan. Teknologian edelleen kehittyessä DVD-levyjen postituksesta luovuttiin ja elokuvia alettiin tarjota suoraan verkosta katsottavaksi. (Netflix 2020.)

Bisneksen siirtäminen verkkoon ei ole enää keino erottua – se on elinehto. Yritykset kilpailevat asiakkaista pyrkimällä tuottamaan parempia palveluita ja julkaisemalla uusia ominaisuuksia. Nopean julkaisusyklin on havaittu olevan merkittävä kilpailuetu. (Maroukian & Gulliver 2020; Rütz 2019, 5; Jumani ym. 2020, 1.)

Ketterät mallit ja DevOps mielletään osaksi nykyaikaista, kilpailukykyistä bisnestä. Tutkimuksesta ja tutkimustavasta riippuen 30% - 85% ohjelmistoyrityksistä käyttää mallia, jonka voi tulkita ainakin osittain DevOpsiksi (Gartner 2019; Jumani ym. 2020, 556.) Suurten yritysten, kuten Airbnb, Netflix ja Etsy, DevOps-menestystarinat inspiroivat toisia yrityksiä kokeilemaan DevOpsia (Erich & Amrit 2017, 1). DevOps nähdään keinona lisätä ohjelmistokehityksen vauhtia ja lyhentää uusien ominaisuuksien läpimenoaikaa. DevOpsin toivotaan myös parantavan ohjelmistojen laatua. (Jumani ym. 2020, 557.)

DevOpsin käyttöönottoon liittyy pelkoja. DevOpsia ei voi ottaa käyttöön ilman kunnollisia pohjatöitä, sillä ei ole olemassa yleispätevää soveltamismallia. DevOps on kokonaisvaltainen yrityskulttuurin ja ajattelutavan muuton, jota pitää ajaa eteenpäin määrätietoisesti: muutosjohtamiseen liittyvät haasteet ja toimintatavat ovat kuitenkin kattavasti kuvattu alan kirjallisuudessa. (Erich & Amrit 2017, 17.)

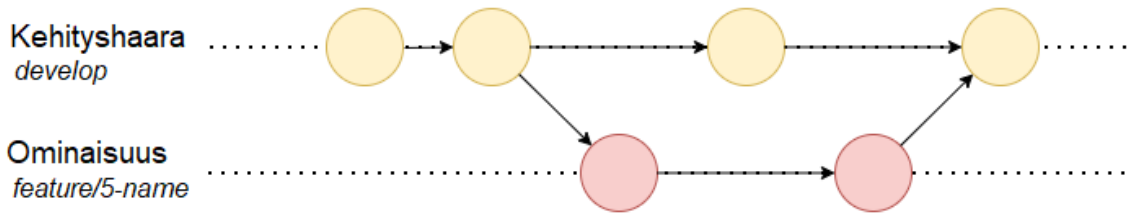
4.2 Versionhallinta

Vaikka ammattimaista ohjelmistokehitystä on vaikea kuvitella ilman jotakin versionhallintatyökalua, ei versionhallinnan käyttö ole kaikissa yrityksissä vielä selviö (Jumani ym. 2020, 572). Versionhallinnan käyttö on ensimmäinen askel kohti DevOpsin kaltaista toimintamallia: version hallinnan käyttö virtaviivaistaa kehittäjien välistä yhteistyöstä. Oikein käytettynä versionhallinta myös antaa suojan ohjelmistokoodin katoamista vastaan esimerkiksi laiterikon tapauksessa. (Hirsimäki 2018, 3.)

Versionhallintajärjestelmistä suosituin on avoimeen lähdekoodiin perustuva Git, joka tarjoaa kattavat ominaisuudet ja mahdollistaa työskentelyn ilman internetyhteyttä (Hirsimäki 2018, 5). DevOps-työkalujen tuki muille versionhallintajärjestelmille on surkea: Git on käytännössä edellytys DevOps-automaation rakentamiselle.

Git on tarkoitettu ohjelmakoodin hallintaan ja se tukee huonosti muiden kuin tekstitiedostojen versiointia. Jos ohjelmistoprojektissa käsitellään usein päivittyviä binääritiedostoja, pitää Git-versionhallintaan asentaa LFS-laajennus (*Large File Storage*). LFS-laajennus vaatii tuen palvelimella ja asiakasohjelmistossa. Tuki LFS-laajennokselle on hyvä. Jos käytetään jotakin eksoottista ratkaisua, esimerkiksi asiakkaan sisäverkkoon asennettua palvelinta, voi LFS-laajennoksen joutua asentamaan itse. (Microsoft 2018.)

Jotta DevOps-työkalupino toimisi oikein ja koodimuutokset voidaan kohdistaa työjonon asiaan, pitää versionhallintaa käyttää systemaattisella tavalla. Suosituin ja työkalujen parhaiten tukema malli on GitFlow. (Driessen 2010.) Git-versionhallinnassa ja GitFlow-työskentelytavassa kehittäjät työskentelevät ns. haaroissa, joissa työstettävä ominaisuus kehittyy erillään toisista ominaisuuksista. Ominaisuudet viimeistellään näissä haaroissa, jonka jälkeen valmis ominaisuus tuodaan osaksi kehityshaaraa. *Kuviossa 3* näytetään esimerkki GitFlow-työskentelytavan mukaisesta versionhallinnan käytöstä.



KUVIO 3. Git-versionhallinnassa työskennellään haaroissa.

GitFlow antaa ohjeet haarojen nimeämiseen. GitFlow:n nimeämiskäytäntöjen lisäksi DevOps-työkalut vaativat työjonojenseurantatyökalun asialle antaman yksilöivän tunnisteen käyttämistä haarojen nimissä. Tyypillisesti haaran nimi aloitetaan yksilöivällä tunnisteella; toiset työkalut taas vaativat tunnisteen mainitsemisen kommenttikentässä. Sovellettava käytäntö pitää valita työkalujen mukaan.

4.3 Jatkuva integraatio

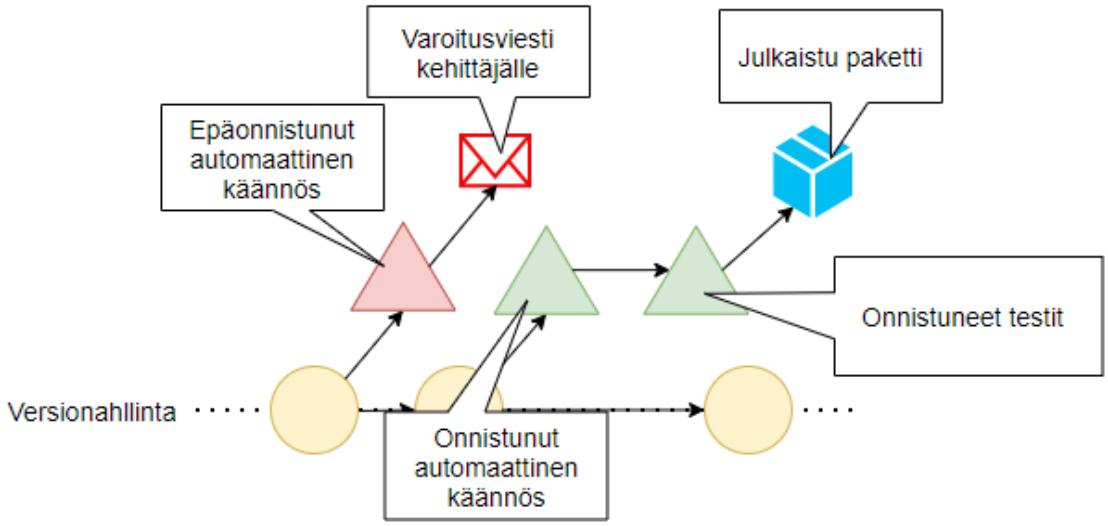
Jatkuvalla integraatiolla (engl. *Continuous Integration, CI*) tarkoitetaan työtapaa ja automaatiota, joka varmistaa, että kehittäjät työskentelevät ajantasaisen version pohjalta. Tavoitteena jatkuvalla integraatiolla on vähentää yhteensopivuusongelmia projekteissa, joissa työskentelee useita kehittäjiä, ja vähentää päällekkäistä työtä. (Fowler 2006.)

Ohjelmistoprojektissa tulee sopia yhteisistä jatkuvan integraation pelisäännöistä. Siinä missä versionhallintakäytännön, kuten GitFlow, määrittävät *miten* versionhallintaa käytetään, jatkuvan integraation käytännöt määrittävät *milloin* versionhallintaa käytetään. (Fowler 2006.) Alla on listattu toimeksiantajan nykyiset, hyväksi havaitut suositukset (Luotio 2016):

- Muutokset viedään keskitettyyn versionhallintaan välittömästi, vähintään työpäivän päätteeksi.
- Kehittäjä tuo muutokset omaan kehitys- tai ominaisuushaaraansa vähintään kerran päivässä.
- Tallennusviestissä noudatetaan projektissa sovittuja käytäntöjä. Tämä saattaa tarkoittaa työstettävän asian yksilöllisen tunnisteen mainitsemista, jos valitut työkalut sitä edellyttävät.

Jatkuvan integraation automaation tarkoitus on antaa kehittäjälle välitöntä palautetta (Jumani ym. 2020, 553). Automaatio rakentaa uuden ohjelmistoversion, eli kääntää koodin, jokaisesta koodimuutoksesta. Jos ohjelmaversion kääntäminen ei onnistu koodimuutoksessa on jokin karkea virhe. Jatkuvan integraation ratkaisut tyypillisesti varottavat kehittäjää ja saattavat estää virheellisen version viemisen kehitys- tai päähaaraan.

Koodin tiheä vieminen keskitettyyn versionhallintaan lisää näkyvyyttä: koko kehitystiimi näkee muutokset ja tallennusviestit. Automaation havaitsevat karkeat virheet näkyvät myös koko tiimille. *Kuviossa 4* havainnollistetaan jatkuvan integraation toimintaa.



KUVIO 4. Jatkuvassa integraatiossa tapahtuu: virheellinen koodimuutos ja onnistunut korjaus.

DevOps on ennen kaikkea positiivinen, yhteistyötä korostava ja rajoja rikkova ajattelumalli. Parhaimmillaan jatkuvan integraation pakottama avoimuus luo spontaania auttamista ja yhteistyötä, joka näkyy merkittävänä parannuksena tuottavuudessa ja laadussa. Avoimuuden tarkoitus ei ole löytää syyllisiä, vaan kehittyä tiiminä.

4.4 Jatkuva testaaminen

Ohjelmistojen testattu yhtä kauan, kuin ohjelmistoja on tehty ja testausta on automatisoitu jo kymmeniä vuosia ennen DevOpsia (Horne 2014). DevOps tuo ohjelmistojen testaukseen uuden automaation tason: automaattisia testejä ajetaan nyt automaattisesti. Jatkuva testaaminen on kenties DevOpsin laajin käsite. Automaattinen testaus on osa jatkuvaa integraatiota, jatkuvaa julkaisua ja jatkuvaa valvontaa.

Ohjelmisto kannattaa suunnitella alusta alkaen automaattisesti testattavaksi. Testejä ja ohjelmistoa tulee kehittää samanaikaisesti. Testejä jälkikäteen toteutettaessa törmätään usein testausta vaikeuttaviin suunnitteluvalintoihin, jolloin paras ratkaisu on tehdä suuriakin muutoksia jo toteutettuun ohjelmakoodiin. Vaikka kaikki koodi on tavalla tai toisella testattavissa, monimutkaisten testien toteuttaminen ei kannata: testeistä saattaa tulla testattavaa koodia mutkikkaampia, jolloin testitkin vaatisivat testejä. (Câmara 2019.)

Ohjelmistonkehitykseen on myös testivetoinen kehitystapa. Testivetoisessa kehitystavassa (engl. *Test Driven Development*) ohjelmakoodille luodaan ensin testit, jonka jälkeen kehittäjä kirjoittaa nämä testit läpäisevän koodin. (Beck 2002.) Testivetoisuus on eräs ketteryyden ideologian lempilapsia, jonka väitetään olevan ratkaisu lähes ongelmaan kuin ongelmaan. Testivetoisen kehityksen ympärillä vellovaa pöhinää ei kannata pelästyä, eikä myöskään purematta niellä.

Jatkuvaa testausta tehdään osana jatkuvaa integraatiota. Kun ohjelmakoodi on käännetty onnistuneesti, suoritetaan ohjelmalle kirjoitetut automaattiset testit. Jos käännös ei onnistu, testejä ei voi ajaa: tämä on tekninen rajoite. Testien tarkoitus on varmistaa, että ohjelma toimii loogisesti oikein. Osana jatkuvaa testausta ajetaan projektin kaikki testit. Näin pyritään varmistamaan, etteivät koodimuutokset riko mitään olemassa olevaa toiminnallisuutta.

Yhden testin epäonnistuminen ei estä muiden testien suorittamista. Jatkuva integraatio kerää listan kaikista epäonnistuneista testeistä; yhdenkin testin epäon-

nistuessa suoritetaan samat toimenpiteet kuin automaattisen käännöksen epäonnistuessa. Testien tapauksessa varoitusviestiin liitetään tiedot kaikista epäonnistuneista testeistä.

4.5 Jatkuva paketointi

Läpäistyjen testien jälkeen koodimuutoksen voi todeta onnistuneeksi. Jatkuvan integraation hyväksymisistä koodimuutoksista luodaan paketti. Paketti on ohjelmakoodista luotu tiedosto tai joukko tiedostoja, joita käyttäen ohjelmisto voidaan asentaa. Nämä paketit tallennetaan myöhempää käyttöä varten. (Immonen 2015, 10.)

Pakettien säilytysaika ei tyypillisesti ole rajaton, sillä niiden säilyttämiseen tarvitaan merkittävä määrä tallennustilaa. Valittujen työkalujen käytännöt pitää tarkistaa, jotta tärkeitä paketteja ei hukata. Eräissä työkaluissa jatkuva integraatio lakkaa toimimasta, jos paketeille varattu tallennustila täyttyy.

5 DEVOPS VÄHENTÄÄ YMPÄRISTÖIHIN LIITTYVÄÄ TUSKAA

5.1 Ympäristöjen automaattinen luominen

”Ympäristö koodina” (engl. *Infrastructure as Code, IaC*) on tapa automatisoida ylläpitotyötä. IaC:ssa resursseja, kuten virtuaalikoneita, tietokantoja ja palomuureja, ei luoda tai konfiguroida käsin. Ohjelmiston vaatima ympäristö kuvataan IaC-ohjelmiston vaatimalla tavalla lähdekoodina. (Jumani ym. 2020, 568.)

IaC-ratkaisuja on kehitetty jo 1990-luvulta alkaen. Ensimmäiset IaC-ohjelmistot tukivat palvelinten ja virtuaalikoneiden automaattista konfigurointia. Julkisen pilvi-infrastruktuurin myötä markkinoille tuli useita kolmannen osapuolen ratkaisuja pilviympäristön hallintaan (Morris 2018).

Kolmella suurimmalla[1] pilvipalveluntarjoajalla on nykyään omat, natiivit IaC-ratkaisunsa. Natiivit ratkaisut tukevat pilviympäristön uusimpia ominaisuuksia aiemmin, ja niiden kanssa on tyypillisesti helpompi työskennellä. Natiivit IaC-ratkaisut toimivat kuitenkin vain omassa pilviympäristössään.

IaC on verrattain monimutkainen aihe, joka vaatii kehittäjiltä paneutumista. Organisaation IaC-työkaluvalinta on syytä standardoida. Jos yritys toimittaa ohjelmistoja useampaan julkiseen pilveen, täytyy valita jokin kolmannen osapuolen ratkaisu.

Automatisoinnin lisäksi IaC mahdollistaa ympäristöjen testaamisen (Jumani ym. 2020, 568). Ympäristön testaaminen eri juurikaan eroa ohjelmakoodin testaamisesta: ympäristö luodaan, sitä vasten ajetaan testejä ja sitten ympäristö poistetaan. Ympäristöjen osien testaamiseen on joitakin ohjelmistoja, mutta ohjelmistotestauksesta tuttuja kattavia paketteja ei vielä ole tarjolla. (Porter 2019.)

[1] Amazon AWS, Microsoft Azure, Google Cloud

Pilvipalvelujen eduksi mainitaan usein skaalautuvuus: pilvipalvelu tarjoaa tarvittaessa välitöntä lisäkapasiteettia. Tuki lisäkapasiteetin automaattiselle tilaamiselle riippuu kuitenkin palvelusta ja pilvestä; automatiikan puuttuessa tuki voidaan kuitenkin toteuttaa IaC-ratkaisuin. (Jumani ym. 2020, 570.)

5.2 Jatkuva julkaisu

Jatkuvalla julkaisulla tarkoitetaan koodimuutoksista käännettyjen ohjelmistoversioiden automaattista tai puoliautomaattista julkaisua (Rütz 2019, 6). IaC, ympäristö koodina, voi olla osa jatkuvan julkaisun prosessia, mutta IaC ei ole jatkuvan julkaisun edellytys.

Riippuen kehitettävän ohjelmiston tyypistä, jatkuva julkaisu voi pitää sisällään mitä erilaisimpia vaiheita. Karkeasti ottaen jatkuva julkaisu asentaa jatkuvan integraation tuottaman paketin johonkin ympäristöön. Ympäristöllä tarkoitetaan pilveä, palvelinta tai laitetta. (Jumani ym. 2020, 557.)

Tyypillisesti ohjelmisto täytyy asentaa useampaan ympäristöön. Esimerkiksi verkkopalveluista ylläpidetään usein eri versioita eri käyttötarkoituksiin. Verkkopalvelun loppukäyttäjille tarkoitettua varsinaista versiota kutsutaan tuotantoympäristöksi. Tuotantoympäristön lisäksi ylläpidetään ympäristöjä uusien versioiden testaamista, beta-versioita ja käyttäjien koulutusta varten. (Jumani ym. 2020, 570.)

Jatkuva testaaminen on osa paitsi jatkuvaa integraatiota, myös jatkuvaa julkaisua. Jatkuvan julkaisun osana ajetaan niin sanottuja savutestejä (engl. *Smoke testing*). Savutestien tarkoitus varmistaa, ettei asennuksessa tai asennetun palvelun konfiguroinnissa ole tapahtunut karkeaa virhettä. (Shakurova 2015, 5.) Savutesteissä testataan ohjelmiston kriittisimmät ja helpoimmin testattavat osat: lähteekö palvelin ja palvelu käyntiin? Pystyykö palvelu kommunikoimaan muiden tarvitsemiensa palveluiden kanssa? Onko verkkosivusto päivittynyt, vai näkyykö vielä vanha versio?

Jatkuvan julkaisu on jaettu vaiheisiin. Vaiheita voidaan suorittaa rinnakkain tai peräkkäin. Jatkuvan integraation tapaan virheet estävät etenemisen seuraaviin vaiheisiin ja kehittäjille ilmoitetaan virheistä samalla tavalla. (Jumani ym. 2020, 567.) Myös onnistuneista asennuksista tulee lähettää ilmoitus: koko tiimin on hyvä pysyä kartalla asennetusta versiosta kussakin ympäristössä. Jatkuvan julkaisun työkalun olisi hyvä tukea yhteenvetonäkymää, jossa eri ympäristöihin asennettuja versioita voi seurata.

5.3 Jatkuva valvonta

Jatkuvasta valvonnasta (engl. *Continuous Monitoring*) käytetään myös suoraan englannista käännettyä rinnakkaistermiä jatkuva monitorointi. Jatkuva valvonta laajentaa jatkuvan testaamisen käsitettä. Ohjelmakoodin testaamisen sijaan testejä ajetaan ohjelmiston toiminnasta kerättyä dataa vasten. Jatkuva valvonta on jatkuvaa testausta jatkuvampaa: testejä ei ajeta tietyssä vaiheessa DevOps-prosessia vaan keskeytyksessä. (Kohomäki 2019, 13.)

Edellytys onnistuneelle jatkuvalle valvonnalle on lokitietojen kerääminen ohjelmistosta. Lokeihin tulee kirjata vain merkityksellistä tietoa ja lokimerkintöihin pitää pystyä kohdistamaan tehokkaita hakuja, esimerkiksi rajaamaan hakutuloksia aikaleiman ja merkinnän tyypin perusteella. Lokimerkinnät ovat hyödyttömiä, jos tietoa ei löydä.

Jatkuvaa seurantaa voi tehdä myös ympäristö- ja infrastruktuuritasolla. Tavallisesti seurataan palvelun osien pulssia (engl. *heartbeat*). Tarkkaan ottaen pulssi on palveluiden ja laitteiden säännöllisesti tuottama ”olen kunnossa” -sanoma; usein jatkuvan valvonnan työkalu kuitenkin aktiivisesti hakee vastaavan sanoman kuuntelun sijaan. (Aguilera, Chen & Toueg 1997, 3.)

Jatkuvan valvonnan havaitsemissa ongelmatilanteissa reagoidaan kuten jatkuvan integraation ja jatkuvan julkaisun aikana tapahtuvissa ongelmatilanteissa: ongelmasta lähetetään varoitus. Jatkuva valvonta voi käynnistää automaattisia prosesseja, joissa varajärjestelmiä otetaan käyttöön.

Nämä automaatiot ovat kuitenkin erittäin monimutkaisia ja vikaherkkiä, ja tarvitsevat omat testinsä ja seurantansa. Julkinen pilvi helpottaa vikasietoisen ohjelmiston kehitystä. Natiivien pilvien IaaS[2] ja Paas[3] -ratkaisut hoitavat palveluiden kahdentamiset ja maantieteelliset hajautukset automaattisesti – lisämaksusta. (Microsoft 2020; Amazon n.d.; Google n.d..)

5.4 Asetusten hallinta

Ohjelmistot täytyy rakentaa niin, että ne ovat konfiguroitavissa. Nämä konfiguraatiot määrittävät ohjelmiston toimintalogiikan eri ympäristöissä. Esimerkiksi verkkokaupan tuotantoversio kommunikoi maksupalveluntarjoajan tuotantoversion kanssa; verkkokaupan testiversion taas kommunikoivat maksupalveluntarjoajan testiversion kanssa, jolloin testiympäristössä ostoksia voi tehdä ilman oikeita maksuja. (Kostecký 2019, 36.)

Osana jatkuvaa julkaisua asennettava ohjelmisto on konfiguroitava asennusympäristöön sopivilla asetuksilla (Erich & Amrit 2017, 7). Perinteisesti asetuksia hallitaan manuaalisesti: ympäristö on konfiguroitu tietyllä tavalla ja konfiguraatiota muutetaan tarvittaessa (Jumani ym. 2020, 560). Tietyn version asentamiseksi ylläpitäjän täytyy tietää, miten versio konfiguroidaan. Kustakin julkaistusta versiosta tarvitaan käyttöohje, joka kertoo tarvittavat asennustoimenpiteet.

Asennuksia ei siis voi automatisoimatta ilman hienostuneempaa asetusten hallintaa. Naiivissa ratkaisussa asetuksia voitaisiin hallita osana sovelluksen lähdekoodia; tätä tapaa suositellaan vanhemmassa kirjallisuudessa (Erich & Amrit 2017, 7). Asetusten hallintaa lähdekoodina ei kuitenkaan voi suositella.

Jos asetuksia hallitaan lähdekoodissa, menetetään monet DevOpsin hyödyistä. Ohjelmistot eivät ole enää skaalautuvia; kehittäjän täytyy tietää kaikki tuetut ym-

[2] Infrastructure as a Service
[3] Platform as a Service

päristöt – ympäristöjä ei voi luoda spontaanisti lisää. Myös vanhan version asentaminen häiriötilanteessa on epävarmaa: vanhan version ulkoiset riippuvuudet eivät välttämättä enää toimi.

Lähdekoodissa ylläpidetyistä asetuksista on siirrytty hallitsemaan asetuksia jatkuvan toimittamisen työkaluissa. Näissä työkaluissa asetukset on sidottu ympäristöön ja ympäristöt voivat jakaa asetuksia keskenään. Asetukset jatkuvan toimituksen työkaluissa ratkaisee skaalautumiseen liittyvät ongelmat ja helpottaa versiopalautuksia.

Microsoft on julkaissut vuoden 2020 alkupuolella *Configuration-as-a-Service*-palvelun nimeltä App Configuration. Palvelu lisää abstraktiokerroksen konfiguraation ja sovelluksen välille: sovellukseen on konfiguroitu linkki julkisen pilven asennuspalveluun. (Microsoft 2020.). Palvelulla voi hallita keskitetysti suurta joukkoa sovelluksia, palveluita ja palvelimia. Aika näyttää millaista tosielämän hyötyä tämä lähestymistapa tarjoaa.

5.5 Salaisuuksien hallinta

Sovellukset käyttävät salaisuuksia muodostamaan luottamuksellisia yhteyksiä toistensa välillä. Salaisuudet voivat olla salasanan kaltaista dataa tai kryptokrafisia salausavaimia. Salaisuuden tyypistä riippuu, miten salaisuutta käytetään luottamuksellisen yhteyden muodostamiseen. (D'Souza, Jao, Mironov & Pandley 2011, 1.)

Salaisuuksien vuotaminen voi johtaa mittaviin vahinkoihin. Paljastuneita salaisuuksia hyväksikäyttämällä on mahdollista varastaa palvelun keräämää henkilötietoa tai toimia palvelun nimissä (Brady & Atwood 2012). Jatkuvan toimittamisen järjestelmässä salaisuuksien puolivillainen hallinta on merkittävä tietoturvariski.

Salaisuudet ovat osa sovelluksen konfiguraatiota, mutta salaisuuksiin liittyvän riskin vuoksi niitä ei pidä käsitellä muun konfiguraation tapaan. Huonoin vaihtoehto on säilyttää salaisuuksia lähdekoodissa: lukuoikeus versionhallintaan tarkoittaa tällöin pääsyä kaikkiin salaisuuksiin.

Jatkuvan toimittamisen työkalussa salaisuuksien käsittely muun konfiguraation tapaan aiheuttaa myös ongelmia: asennuksen yhteydessä salaisuus saatetaan kirjottaa sellaisenaan lokitiedostoon. Useissa työkaluissa asetuksen pystyy merkitsemään luottamukselliseksi, jolloin salaisuutta ei kirjoiteta lokiin.

Kaikissa pelkästään jatkuvan julkaisun työkaluihin perustuvissa salaisuuksienhallintamalleissa salaisuus asennetaan lopulta palveluun tai palvelimelle, josta salaisuus on luettavissa. Asetusten päivittäminen ja tarkastaminen on normaali ylläpitotoimenpide, jonka yhteydessä ei ole perusteltua päästä näkemään salaisuuksia.

Salaisuuksien hallintaan tulee käyttää palvelua, jossa pääsyä salaisuuksiin voidaan rajata ja valvoa. Palvelun konfiguraatio ei enää sisällä salaisuutta, vaan viittauksen salaisuuteen. Sovellus hakee salaisuudet palvelusta tarvittaessa. Pääsy salaisuuksiin suojataan jollakin tehokkaalla, salaisuuksiin perustumattomalla tavalla (Brady & Atwood 2012). Käytännössä tämä tarkoittaa palvelinvarmenteisiin perustuvaa tunnistautumista. Julkisen pilven palveluntarjoajat ovat tuoneet kukin[4] markkinoille oman muutamalla klikkauksella käyttöönotettavan ratkaisun.

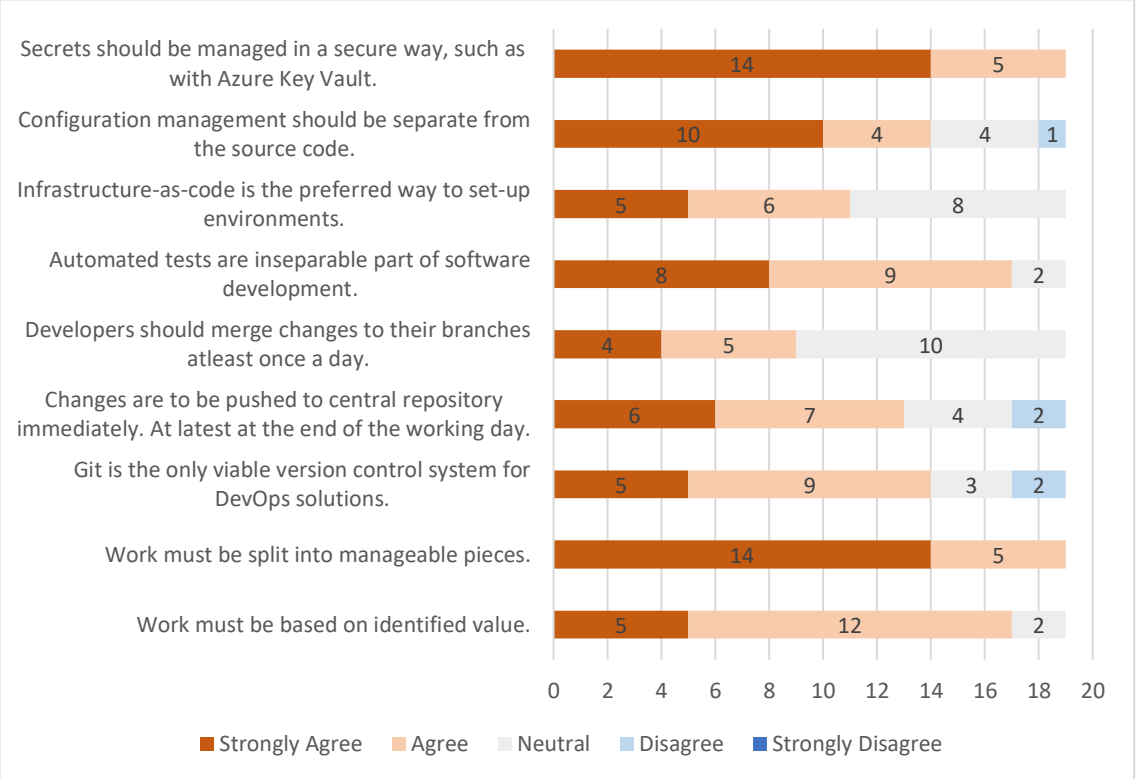
Julkinen pilvi tuo salaisuuksien hallintaan vieläkin paremman ratkaisun: salaisuuksista luopumisen. Lähes kaikki Microsoft Azure palvelut tukevat *Managed Identity* -teknologiaa, jossa teknologiaa tukevien palvelujen väliset riippuvuudet voidaan suojata palvelinvarmenteilla. (Microsoft 2020.) Pääsynhallinta on tällöin ohjelmistolle täysin näkymätöntä. **Salaisuuksia, joita ei ole, ei voi vuotaa**.

[4] Amazon AWS, Microsoft Azure, Google Cloud

6 TULOSTEN KÄSITTELY JA POHDINTA

6.1 Tulosten hyödynnettävyys

Tämän raportin pohjalta laadittiin yhdeksän lausumaa, joihin Innofactorin projektiliiketoiminnan työntekijöitä pyydettiin ottamaan kantaa. Kyselyssä vastaaja sai esittää mielipiteensä viisiportaisella Likert-asteikolla: Vahvasti samaa mieltä, Samaa mieltä, neutraali, Eri mieltä, Vahvasti eri mieltä. Kyselyyn liittyi myös vapaatekstikenttä otsikolla *"Ideas, Propose a recommendation, Topics that should be covered."* Kysely toteutettiin anonyyminä Office 365 lomakkeena, johon vastaaminen oli vapaaehtoista. Alkuperäiset kysymykset ja vastausten jakauma esitelty *kuviossa 5*. Vastaajien avoimet vastaukset löytyvät *liitteestä A* ja käytetty kyselylomake *liitteestä B*.



KUVIO 5. Henkilöstökyselyyn saatiin 19 vastausta. Kyselyyn vastaaminen oli vapaaehtoista ja vastaaminen tapahtui Microsoft Teamisissa jaetun linkin kautta. Kyselyn jakelu oli 138 henkilöä. Kyselyn kvantitatiivisen osan vastaukset esitettynä pylväsdiagrammina

Henkilöstön mielipide oli pääosin tämän raportin takana: epäsuosituimmaksi osaksi raporttia osoittautuivat jatkuvaan integraatioon liittyvät versionhallintakäytännöt. Kyselyn tulosten julkistamisen jälkeisissä purkukeskustelussa selvisi, että käytännöt ovat ihan hyviä ja kannatettavia, mutta inhottavia askareita.

Ympäristö koodina -käsite tunnettiin heikosti. Lähes puolet vastaajista suhtautui käsitteeseen neutraalisti. Eräs vastaaja piti lähestymistapaa liian työläänä yleiseksi käytännöksi; toinen taas ei tuntenut käsitettä.

Vastausten perusteella voidaan olettaa selvitystyössä syntyneen mallin vastaavaan toimeksiantajan tarvetta. Käytännöt ovat maksimissaan niin hyviä, kuin henkilöstön sitoutuminen niihin. Raportin sisältö on edellä mainittuja seikkoja lukuun ottamatta sellainen, johon henkilöstö voi sitoutua.

Yhdeksi riskiksi muodostuu pieni, 19 vastauksen otos: vastaajat eivät edusta koko perusjoukkoa. Perusjoukon ollessa pieni, 138 työntekijää, luotettavan tuloksen saavuttamiseksi pitäisi tehdä kokonaistutkimus. Kokonaistutkimuksessa vastaukset haetaan koko perusjoukolta. (Mattila 2003.)

6.2 Tavoitteen ja tarkoituksen toteutuminen

Tunnistetuista riskeistä huolimatta opinnäytetyössä päästiin asetettuun tavoitteeseen: vuoden 2016 DevOps-toimintamalli on päivitetty tässä raportissa kuvatun uusimman tiedon pohjalta. Suurin parannus edellisen versioon oli asetusten- ja salaisuuksien käsittely: edellisessä versiossa asetustenhallintaa käsiteltiin kappaleen verran ja salaisuuksienhallintaa ei lainkaan.

Myös muut toimintamallin osiot käytiin läpi, kuten opinnäytetyössä oli tarkoituksena. Kaikki osat tarkastettiin, ja niitä tarkennettiin ja korjattiin. Pääosin havaitut puutteet selittyvät tiedon tarkentumisella: DevOps on kehittynyt huimasti sitten vuoden 2016. Osaa DevOpsiin liittyvistä aiheista oli käsitelty liian suppeasti – näitä osia laajennettiin.

DevOpsin taustat ja DevOpsiin liittyvä teoria käytiin läpi. DevOpsiin liittyvän historian tunteminen auttaa lukijaa ymmärtämään ongelmat, joita DevOps pyrkii ratkaisemaan. Raportissa taustojen ja teorian oikeellisuuteen kiinnitettiin erityistä huomiota: lähteitä käytettiin runsaasti ja lähteiden valintaan kiinnitettiin erityistä huomiota.

Opinnäytetyössä DevOpsiin liittyvät termit avattiin kattavasti ja selkokielisesti: tämän raportin perusteella lukija saa selkeän kokonaiskuvan DevOpsista. Opinnäytetyön sisältö on asetettujen tavoitteiden mukaisesti ymmärrettävissä ilman teknistä asiantuntemusta.

6.3 Jatkokehitystarpeet

DevOps on jatkuvaa parantamista; myös DevOpsin kuvaamisen täytyy perustua jatkuvaan parantamiseen. DevOps kehittyy alati, ja toimintamalli vaatii jatkuvaa ylläpitoa. Opinnäytetyön toteutuksessa havaittiin kuitenkin myös akuutteja parannuskohteita.

Konfiguraationhallintaan keskittyvä Azure App Configuration -tuote soveltunee erinomaisesti moniin toimeksiantajan asiakasprojekteihin. Palvelua tulee kokeilla ja kokemukset jakaa organisaation sisällä.

Ympäristöt koodina, IaC, oli henkilöstön keskuudessa harmillisen vähän tunnettu ja käytetty konsepti. Eräissä toimeksiantajan projekteissa IaC-ratkaisuja on kuitenkin käytetty erittäin onnistuneesti. Tietotaitoa organisaation sisällä täytyy saada jaettua tehokkaammin.

6.4 DevOpsin tulevaisuus

Tulevaisuudessa DevOps saa uusia ulottuvuuksia. Esimerkiksi Liitteessä A mainittu *DevSecOps*, jossa tietoturva tuodaan jatkuvaksi prosessiksi, on jo käsitteenä olemassa. Tavoitteena on tuoda tietoturva mukaan päivittäiseen ohjelmistokehitykseen; usein tietoturvaa aletaan miettiä projektin loppuvaiheessa, kun

teetetään tietoturva-auditointi. Tulevaisuus näyttää, millä tavoin auditointi voidaan tuoda jatkuvaksi ja automatisoiduksi osaksi ohjelmistokehitystä. (RedHat n.d..)

Data, tekoäly ja koneoppiminen ovat viimevuosien trendejä. Tekoälyratkaisut perustuvat usein vähäkoodisiin ja lähdekoodittomiin teknologioihin. Näitä ohjelmistoja saatetaan kehittää esimerkiksi graafisella käyttöliittymällä. Lähdekoodittomien ratkaisujen automaattisen testauksen täytyy kehittyä lähivuosina merkittävästi.

Koneoppimisen ympäristöt eivät ole virtuaalikoneita kummempia, ja näitä voi hallita jo nykyisillä DevOps työkaluilla. Koneoppimisen automaattinen seuranta puolestaan on tulossa oleva konsepti: seurantaa voidaan tehdä sekä datalle että tuotetulle mallille. (Karbhari 2020.)

Tekoäly lienee tulevaisuudessa osa ohjelmistojen jatkuvaa seurantaa. Markkinoilla on jo nyt tekoälyyn perustuvia analytiikkasovelluksia. (Berman 2017; Microsoft 2019.) Tiedon karttuessa ja teknologian kehittyessä nämä ratkaisut tulevat kehittymään; samalla kerätyn telemetrian tulee kehittyä, ja tämä vaatii kehittäjiltä uutta osaamista.

DevOps lienee erottamaton osa tulevaisuuden ohjelmistokehitystä; ohjelmistojen tulevaisuus on DevOpsin tulevaisuus. Tässä raportissa esitellyt toimintatavat ovat osa pitkää, paljon ohjelmistoja ja tietotekniikkaa vanhempaa jatkumoa. Terminä DevOps hävinnee aikanaan ja korvautuu toisilla.

Ohjelmistotuotannon, ja teollisen tuotannon, seuraavia trendejä ei voi kuin arvailla. DevOps lähtee tarpeesta tuottaa arvoa ja välttää hukkaa. Jatkuvan lisäarvon tavoittelu saattaa kuitenkin tulla pian tiensä päähän: kenties seuraavat tavat järjestää työtä ja tuotantoa perustuvat inhimillisille- ja ympäristöarvoille? (Emiliani 2020.)

LÄHTEET

Aguilera, M. K., Chen, W. & Toueg, S. 1997. *Heartbeat: A Timeout-Free Failure Detector for.* Ithaca: Cornell University.

Allspaw, J. & Hammond, P. 2019. 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. *Web Performance and Operations Conference.* San Jose: O'Reilly. Luettu 19.9.2020. https://www.youtube.com/watch?v=LdOe18KhtT4

Amazon. n.d. Luettu 27.9.2020. https://docs.aws.amazon.com/elemental-cl3/latest/ug/redundancy-and-failover.html

Atlassian. 2020. *What is an issue?* KUVIO 27.9.2020. https://support.atlassian.com/jira-software-cloud/docs/what-is-an-issue/

Beck, K. 2002. *Test-driven development by example.* Boston: Addison-Wesley. Verkkoversio, sivunumerointia ei saatavilla.

Berman, D. 2017. *The Growing Role of Machine Learning in Monitoring*. Luettu 16.10.2020. https://logz.io/blog/machine-learning-in-monitoring/

Câmara, P. 2019. *Writing Testable Code*. Luettu 22.9.2020. https://medium.com/feedzaitech/writing-testable-code-b3201d4538eb

Department of Defense. 1985. *Defense System Software Development.* Washington DC: Department of Defense.

Driessen, V. 2010. *A successful Git branching model*. Luettu 22.9.2020. nvie.com: https://nvie.com/posts/a-successful-git-branching-model/

D'Souza, R., Jao, D., Mironov, I. & Pandley, O. 2011. Publicly Verifiable Secret Sharing. *INDOCRYPT* (s. 290-309). Berlin: Springer-Verlag.

Emiliani, B. 2020. *What Comes After Lean?* Luettu 10.16.2020. https://bobemiliani.com/what-comes-after-lean/

Erich, F. & Amrit, C. 2017. A Qualitative Study of Devops Usage in Practice. *Journal of Software: Evolution and Process*.

Esmaeilian, G. R., Mahmad, M. F., Sulaiman, S. & Ismail, N. 2020. Assembly Line and Balancing Assembly Line. Serdang: University Putra Malysia

Fowler, M. 2006. *Continuous Integration*. Luettu 22.9.2020. https://martinfowler.com/articles/continuousIntegration.html

Gartner. 2019. *Gartner Survey Finds 85 Percent of Organizations Favor a Product-Centric Application Delivery Model*. Luettu 22.9.2020. https://www.gartner.com/en/newsroom/press-releases/2019-02-19-gartner-survey-finds-85-percent-of-organizations-favor-a-product

Google. n.d. Luettu 27. 9 2020. https://cloud.google.com/compute/docs/tutorials/robustsystems

Haapala, T. 2016. *Ketterä ohjelmistokehitys ja asiakkuudet.* Espoo: Metropolia.

Hannola, H. 2015. *Julkisen hankinnan tarjouskilpailuun osallistuminen: Ikääntyneiden tehostettu palveluasuminen.* Kouvola: KYAMK.

Hirsimäki, V. 2018. *Versionhallinta ja parhaat käytänteet integraatiojärjestelmälle.* Espoo: Metropolia.

Hirvilampi, M. 2019. *Mielikuvamarkkinointi yrityksen liiketoiminnassa.* Seinäjoki: SeAMK.

Horne, G. 2014. *A (Very) Brief History of Test Automation*. Luettu 22.9.2020. https://www.linkedin.com/pulse/20141007123253-16089094-a-very-brief-history-of-test-automation

Immonen, J. 2015. *Web application security testing as part of continuous integration in .NET projects.* Jyväskylä: JAMK.

Jumani, A. K., Siddique, W. A. & Shaikh, A. A. 2020. Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud. *Indian Journal of Science and Technology*, 553-575.

Karbhari, V. 2020. *Continuous monitoring for data projects*. Luettu 16.10.2020. https://medium.com/acing-ai/continuous-monitoring-for-data-projects-11fb1c00c7a4

Kohomäki, S. 2019. *DevOps-käytännöt opetusjärjestelmien kehityksessä ja ylläpidossa.* Tampere: Tuni.

Kostecký, I. 2019. *An approach to Software Deployment.* Jyväskylä: Jamk.

Luotio, J. 2016. *DevOps-toimintamalli ja työkalut.* Espoo: Innofactor. Sisäinen lähde, ei julkisesti saatavilla.

Lynch, W. 2019. *The Brief of History of Scrum*. Luettu 24.9.2020. https://medium.com/@warren2lynch/the-brief-of-history-of-scrum-15efb73b4701

Maroukian, K. & Gulliver, S. R. 2020. *Leading DevOps Practice and Principal Adaption.* Reading: University of Reading,Henley Business School.

Mattila, M. 2003. *Otos ja otantamenetelmät*. Luettu 16.10.2020 osoitteesta https://www.fsd.tuni.fi/menetelmaopetus/otos/otantamenetelmat.html

Mergel, I., Edelmann, N. & Haug, N. 2019. Defining digital transformation: Results from expert interviews. *Government Information, Quarterly Volume 36, Issue 4*.

Mezak, S. 2018. *The Origins of DevOps: What's in a Name?* Luettu 19.9.2020. https://devops.com/the-origins-of-devops-whats-in-a-name/

Microsoft. 2018. *Manage and store large files in Git*. Luettu 22.9.2020. https://docs.microsoft.com/en-us/azure/devops/repos/git/manage-large-files?view=azure-devops

Microsoft. 2019. *Use #ID to link to work items*. Luettu 24.9.2020. docs.microsoft.com: https://docs.microsoft.com/en-us/azure/devops/notifications/add-links-to-work-items?view=azure-devops

Microsoft. 2019. *What is Application Insights?* Luettu 16.10.2020. https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview

Microsoft. 2020. Luettu 27.9.2020. https://docs.microsoft.com/en-us/azure/azure-app-configuration/overview

Microsoft. 2020. Luettu 27.9.2020. https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/services-support-managed-identities

Microsoft. 2020. Luettu 27.9.2020 osoitteesta https://docs.microsoft.com/en-us/azure/availability-zones/az-region

Morris, K. 2018. *Infrastructure as Code: From the Iron Age to the Cloud Age*. Luettu 27.9.2020. https://www.thoughtworks.com/insights/blog/infrastructure-code-iron-age-cloud-age

Netflix. 2020. *The Story of Netflix*. Luettu 21. 9 2020. netflix.com: https://about.netflix.com/en

Numm, J., Friedmann, M., Lukyanov, A., Rajagopalan, R., Hawley, R., Brady, S. & Atwood, B. 2012. *Yhdysvallat Patenttinro US8881249B2.*

Patoranta-Lötjönen, A. 2012. *5S-MALLIN HYÖDYNTÄMINEN SUNWIRE™ TUOTANNOSSA.* Pori: Samk.

Porter, S. 2019. *Infrastructure as code: testing and monitoring*. Luettu 24.9.2020. https://blog.sensu.io/infrastructure-as-code-testing-and-monitoring

RedHat. n.d. Luettu 16.10.2020. https://www.redhat.com/en/topics/devops/what-is-devsecops

Rintala, M. 2015. *Ketterä projektinhallinta Hämeen ammattikorkeakoulussa.* Tampere: TAMK.

Royce, W. W. 1970. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. *Proceedings, IEEE WESCON* (ss. 1-9). Piscataway: IEEE.

Rütz, M. 2019. *DEVOPS: A SYSTEMATIC LITERATURE REVIEW.* Wedel: Fachhochschule Wedel.

Schwaber, K. & Sutherland, J. 2020. *The Scrum Guide.* Schwaber & Sutherland.

Shakurova, O. 2015. *Automating UI Tests for a Web Application Using Test-Complete.* Helsinki: Haaga-Helia.

Takeuchi, H. & Nonako, I. 1986. The New New Product Development Game. *Harvard Business Review*. Luettu 27.9.2020. https://hbr.org/1986/01/the-new-new-product-development-game

Tyynismaa, M. 2014. *Lean- ja Kanban-menetelmät ohjelmistotuotannossa.* Seinäjoki: SeAMK.

Womack, J. P. & Daniel, J. D. 2003. *Lean Thinking.* New York: Simon Schuster.

Womack, J. P., Jones, D. T., & Roos, D. 1990. *The Machine That Changed the World.* New York: Macmillan/Rawson Associates.

Zarour, M., Alhammad, N. & Alenezi, M. 2019. A Research on DevOps Maturity Models. *International Journal of Recent Technology and Engineering*.

LIITTEET

Liite A: Henkilöstökyselyn vapaat vastaukset

IaC requires relatively much work so it's not relevant in most of our customer projects. Not sure if we could decide certain measures for each project to evaluate whether IaC should be applied or not or is it always case by case evaluation.
Infrastucture-as-code is unfamiliar for me. Should learn of it.
I think the client shouldn't have to tell us their site is down. We should as a minimum have some kind of monitoring and alerts for testing the site is up and running ok.
Acceptance Criteria (AC) is mandatory. If at least one criteria can't be defined, the requirement is too wage. Every Task related to the Product Backlog Item (PBI) should the related to at least to one AC. Given-When-Then -style is recommended to be used when defining AC.
No secrets i.e with Managed Identity and RBAC would be a preferred goal when using Azure
Security, like quality, should be built-in, thus DevSecOps!
CI- / CD- preferences platforms and recommendations
About environments 1. Environments need to be 100% distinct, e.g., apps in Development environment should not share _anything_ with apps in Production environment. Eg. when using azure storage account resources such as blob containers for persisting documents, then separate containers need to be defined for each instance. 2. Staging environments need to be 100% distinct from production environments, e.g., when using Azure web app slots to enable zero-downtime deployments, the staging environments need their own storage containers (databases, blob containers etc.) => 1. and 2. could be summarized as "Environments must be distinct". ++ The benefit of distinct environments is peace of mind during deployments, and the reduced risk of corrupting an environment should shared resources (such as accidentally applying automatic migrations to a shared database). (The following are a bit low level but what ya gonna do about it bub?) About version control 1. High quality commits and messages (e.g. https://chris.beams.io/posts/git-commit/) About automation (on developer's box) 1. lint-on-save 2. git hooks for running unit tests before commit/push

Liite B: Henkilöstökyselylomake

Proposed DevOps-recommendations - Agree or Disagree?

There is an internal DevOps standard operating procedure from 2016 that is being updated. The SOP is ment as a reference for the teams; and as an introduction to Innofactor's project business for prospective clients. Would love some feedback on the following statements.

1. Work *

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
Work must be based on identified value.	○	○	○	○	○
Work must be split into manageable pieces.	○	○	○	○	○

2. Version Control *

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
Git is the only viable version control system for DevOps solutions.	○	○	○	○	○
Changes are to be pushed to central repository immediately. At latest at the end of the working day.	○	○	○	○	○
Developers should merge changes to their branches atleast once a day.	○	○	○	○	○

3. Automation *

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
Automated tests are inseparable part of software development.	○	○	○	○	○
Infrastructure-as-code is the preferred way to set-up environments.	○	○	○	○	○
Configuration management should be separate from the source code.	○	○	○	○	○
Secrets should be managed in a secure way, such as with Azure Key Vault.	○	○	○	○	○

4. Ideas, Propose a recommendation, Topics that should be covered

Kirjoita vastaus