OULUN AMMATTIKORKEAKOULU

Jari Komulainen

DEVOPS-TOIMINTAMALLIN KÄYTTÖÖNOTTO PIENESSÄ YKSIKÖSSÄ

DEVOPS-TOIMINTAMALLIN KÄYTTÖÖNOTTO PIENESSÄ YKSIKÖSSÄ

Jari Komulainen
Opinnäytetyö
Kevät 2021
Lean-johtaminen (yamk)
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Insinööri (YAMK), Lean-johtaminen

Tekijä(t): Jari Komulainen
Opinnäytetyön nimi: DevOpsin käyttöönotto pienessä yksikössä
Työn ohjaaja: Tauno Jokinen
Työn valmistumislukukausi ja -vuosi: Syksy 2021 Sivumäärä: 38 + 3 liitettä

Oulun yliopiston mittaustekniikan ohjelmistokehittäjillä on tarkoitus siirtyä Lean-pohjaiseen ajattelutapaan, ja sitä tukemaan on tehty tämä opinnäytetyö. Opinnäytetyön tarkoituksena on varmistaa, että yksikössä ymmärretään, kuinka mahdollistetaan jatkuva oppiminen ja vältytään toistamasta aiempia virheiltä jatkossa. Tarkoituksena on myös luoda ympäristö, jossa työssä kehittyminen on luontevaa ja helppoa.

Opinnäytetyössä keskitytään DevOpsiin ja sen eri mahdollisuuksien hyödyntämiseen. DevOps on laaja käsite ja uusi ajattelumalli, jonka toteuttamisesta ei ole vielä selviä ohjeita, kuinka sitä aina pitäisi toteuttaa. Sen keskeinen toimintamalli on hyödyntää kokonaisuuksia, joissa kaksi tai useampi osapuolta on mukana kehittämässä sähköisiä palveluja, jotta osapuolten välinen kommunikointi olisi tehokkaampaa. Toimintamallin tarkoituksena on myös saada poistettua negatiiviset ja toisiaan vastaan olevat ajattelutavat, sillä ne vähentävät työn tehokkuutta. Toimintamallin tarkoitus saada kaikki sähköisen palvelun osavaiheiden käyttäjät yhteen, jotta he voisivat oppia ja auttaa toisiaan sovussa.

Opinnäytetyön tutkimusongelma ratkaistaan esimerkiksi työpajoja käyttämällä ja tekemällä työpaikkakulttuuriin pieniä ja jatkuvia muutoksia. Kun nämä sisäistetään, auttavat ne DevOpsin kokonaisuuden käyttöönotossa. Uuden kulttuurin käyttöönotossa käytetään Hoshin ajattelumallia, jolla varmistetaan kaikkien työntekijöiden ymmärtävän aiheesta pidettävien työpajojen tarkoituksen. Jatkuvan parantamisen ja oppimisen ajattelumallien kautta ymmärretään kehittymisen tarve ja tavoite sen saavuttamiseksi.

Työn tuloksena tiedetään, millaisia asioita työntekijöiden ja esimiesten tulee oppia, ja näitä asioita tullaan harjoittelemaan esimerkiksi työpajoissa. Myöhemmin asiat otetaan myös käyttöön työympäristössä, jotta ohjelmistokehitystyö olisi tehokkaampaa.

Asiasanat: DevOps, lean, ketterät ohjelmointiympäristöt, soveltaminen

ABSTRACT

Oulu University of Applied Sciences
Master Degree Program, Lean Management

Author(s): Jari Komulainen
Title of thesis: Deploying DevOps in a small unit
Supervisor(s): Tauno Jokinen
Term and year when the thesis was submitted: autumn 2021
Number of pages: 38 + 3 appendices

The software developers of measurement technology at the University of Oulu wants to transfer with the help of this thesis to the Lean-based mindset. The purpose of the thesis is to make sure that it is understood in the department how to ensure continued learning while also avoiding the same mistakes. The purpose is also to create an environment where improvement in the work field is easy and natural.

The Thesis will focus on DevOps and how it can utilize its possibilities. DevOps is a broad notion and on top of that a very new way of thinking that doesn't have many clear directions about how it should be implemented yet. The central idea is to make use of entities in which two or more parties are developing electronic services so that the communication between the parties can be more efficient. The purpose of this mindset is to get rid of negative and conflicting ways of thinking because they decrease the efficiency of the work. The purpose of this mindset is to bring all of the users of the different electronic service stages together so that they can learn and help each other in harmony without fearing one another.

The results of the thesis are individual methods in software development that, while being internalized, help DevOps be utilized in its entirety. In the utilization of new culture, the Hosh mindset ensures that all workers understand the specific purpose of each workshop. Through the mindset of continual improvement and learning, the need for development and the goal of reaching it are understood.

As a result of the research, the things that workers and managers should learn are known and that these things should be practiced in the workshops, for example. These methods will be later implemented in a work environment so that software development would be more efficient.

Keywords: DevOps, lean, agile programming, application

SISÄLLYS

LYHENTEET JA SELITYKSET 7
1 JOHDANTO 8
2 DEVOPS 11
2.1 Rakenne 11
2.2 Kehityspolku 12
2.3 Dokumentointi 16
2.4 Versionhallinta 16
2.5 Hyödyt 18
2.6 DevOpsin käyttöönottaminen 19
2.6.1 Hoshin ajattelumalli 19
2.6.2 Jatkuvan parantamisen malli ja Kaizen 20
3 DEVOPSIN KÄYTTÖÖNOTTAMISEN MALLI 21
3.1 Työssä toteutetut DevOpsin kehitysaskeleeet 22
3.2 Versionhallinta 22
3.3 Gitin käyttöönotto ja käyttämisen ohjeet 24
3.4 DevOpsin käyttöönottamisen haasteet 24
4 DEVOPSIN OSAVAIHEIDEN TOTEUTUS MITYN OHJELMISTOKEHITYKSSÄ 29
4.1 Suunnittelu 29
4.2 Koodaaminen ja versionhallinnan päivittäminen 29
4.3 Testaaminen 30
4.4 Julkaisu ja käyttöönotto 30
4.5 Ohjelman käyttäminen ja käyttäjän palaute 31
4.6 Työkalut 31
4.7 Dokumentointi ja mallidokumentointipohjat 32
4.8 Tavoitteiden asettaminen 32
4.9 Mittarit arviointia varten 32
4.10 Säännölliset työpajat/workshopit 33
4.11 Työn käytännölliset havainnot 33
5 TULOKSET JA NIIDEN ARVIOINTI 36
5.1 Tulosten arviointi 36
5.2 Ajatukset jatkosta 37
6 YHTEENVETO 38
LÄHTEET 39

LIITTEET 45

LYHENTEET JA SELITYKSET

CD	Continuous delivery. Jatkuvan toimittamisen malli.
CI	Continuous Integration, jolla tarkoitetaan kehitysputkea, jolla varmistetaan sovelluksen eheys.
Dev	Development eli ohjelmistokehittäjä
DevOps	Toimintamalli sähköisten palvelujen tuotantoon: malli pyrkii automatisoimaan ohjelmistokehitykseen, testaamiseen ja ylläpitoon liittyvät IT- palvelutoiminnot välttääkseen virheiltä ja nopeuttaakseen ohjelmakehitystä. Nimike on peräisin Agile- ja Lean-konseptien soveltamisesta ohjelmistotuotannosta operatiiviseen toimintaan kuten palveluntarjontaan.
Git-arkisto	Projektin hakemisto, jossa on kaikki kyseisen projektin tiedostot.
Jenkins	Avoimen lähdekoodin automaatiopalvelin, jonka avulla kehittäjät ympäri maailmaa voivat rakentaa, testata ja ottaa käyttöön ohjelmistojaan.
Laadunvalvoja	Tarkastelee ohjelmia virheiden varalta ja varmistaa toiminnallisuuden.
Lean-ajattelu	Mahdollisimman paljon arvoa mahdollisimman pienellä määrällä muutoksia
Lähdekoodi	Sovelluksen koodi, jota käyttäjä voi muokata. Ennen kuin ohjelmisto toimii, pitää lähdekoodi kääntää tietokonetta varten sen ymmärtämään muotoon.
MITY	Oulun yliopiston Mittaustekniikan yksikkö
Ops	Operations eli ohjelmiston ylläpitäjä, käyttäjä
Revisio	Ohjelmiston versio. Dokumentoidaan ohjelmistomuutosten sisältöä tämän avulla.
SVN	Subversion. Lähdekoodin versionhallintasovellus.

1 JOHDANTO

Mittaustekniikan yksikön ohjelmointitiimin tavoite on kehittää dokumentointia ja ohjelmistoversionhallintaa. Haasteita luovat etenkin työn luonne, monen projektin yhtäaikainen suorittaminen ja projektien vaihtelevuus. Nämä vaikuttavat ohjelmistokehityksen menetelmien standardisoimiseen. Ohjelmistokehittäjien täytyy oppia uusia asioita jokaisen projektin alkaessa, mikä osaltaan hidastaa uusien asioiden oppimista ohjelmistokehityksessä. Kun projektien protoversiot saadaan kuntoon, siirrytään pian seuraavaan projektiin, jolloin viimeistelylle ja oppimiselle ei ole riittänyt tarpeeksi aikaa. Haasteita on lisännyt myös työntekijöiden vaihtuvuus. Dokumentoinnin vajavuuksien vuoksi uusien työntekijöiden perehdyttäminen ohjelmistoympäristöihin kestää toivottua pitempään. Tämä hidastaa ohjelmistojen laadun kehittymistä ja on työntekijälle stressaavaa.

Mittaustekniikan yksikössä on käytössä SVN-versionhallintajärjestelmä. Sen käyttömenetelmistä ei ole sovittu selvästi, ja sen käyttöohjeet ovat puutteelliset, mistä johtuen sitä myös käytetään puutteellisesti. Työntekijöiden vaihtuessa haasteet dokumentoinnissa ja kommunikoinnissa ovat johtaneet joidenkin tietojen menetykseen. Myös ohjelmoijien välistä tiedonkulkua ja työntekijöiden jatkuvaa kehittymistä koetetaan parantaa, sillä haasteet näissä osa-alueissa hidastavat ohjelmoinnin kestävää kehitystä. Opinnäytetyössä on tarkoitus paneutua etenkin näihin haasteisiin.

Opinnäytetyössä ratkaistaan, kuinka uusimpia ohjelmistokehityksen ajattelumalleja sovelletaan pieneen ohjelmistokehityksen yksikköön, sekä tehdään mallipohja dokumentaatiosta ja ohjeet versionhallintaan. Uuden toimintamallin käyttöönottamisen haasteita tarkastellaan, jotta siitä voidaan hyötyä mahdollisimman pian. Jotta opinnäytetyön esittämä toimintamalli saavuttaa parhaimman mahdollisen hyödyn, täytyy ymmärtää, kuinka sitä käytetään ja erityisesti mitä nykyisiä haasteita se voi ratkaista. Jotta ajattelumallin sisäistäminen olisi mahdollisimman helppoa, pitää myös sen heikkouksia ja haasteita ymmärtää, ettei käyttöönotossa tulisi virheitä.
Tässä opinnäytetyössä esitetään DevOps-toimintamalli ja keskitytään erityisesti ohjelmistokehityksen prosessiin. Ohjelmoijien tietoutta ohjelmistokehityksestä lisätään, jotta ohjelmistojen tekeminen osataan suunnitella oikeilla periaatteilla ja menettelytavoilla. Lisäksi tässä työssä nostetaan esiin DevOps-toimintamallin käyttöönottamisen yleisimmät haasteet. Lopuksi suunnitellaan toimenpiteet kohdeyritykselle toimintamallin käyttöönottamisen varmistamiseksi.

Opinnäytetyön tutkimusongelma ratkaistaan konstruktiivisen tutkimuksen menetelmillä (kuva 1). Opinnäytetyössä on tarkoitus luoda konstruktio kohdeyrityksen ongelmaa varten, mikä käytännössä tarkoittaa laajaa perehtymistä aiheeseen ja toimintamallin esittämistä ongelman ratkaisemiseksi sen pohjalta. Tutkimustyössä perehdytään monipuolisiin kirjallisuus- ja tutkimuslähteisiin, joissa käsitellään DevOpsia, dokumentointia, versionhallintaa ja ohjelmistokehityksen prosessia. Teorian pohjalta tavoite on tuottaa laadukas ja teoreettisesti perusteltu ratkaisu, joka soveltuu käytettäväksi ja sovellettavaksi jokapäiväisessä ohjelmistokehityksessä. Teoriaan pohjautuen tehdään ohjeet, koulutetaan henkilöstö ja varmistetaan yhdessä oikeat menettelytavat. Muutoksilla pyritään standardoimaan työn menettelytapoja ja sitä kautta parannetaan laatua ja tehokkuutta. (Aljundi 2018; Jokinen 2012.)

KUVA 1. Konstruktiivinen tutkimusote (Jokinen 2012)

Oulun yliopiston Mittaustekniikan yksikkö (MITY) perustettiin vuonna 2011 yhdistämällä mittaustekniikan ja analytiikan toiminnot Kajaanin yliopistokeskukseen. Yksikössä työskentelee noin 40 korkeasti koulutettua tutkijaa ja analytiikan ja mittaustekniikan asiantuntijaa. Yksikön pääsovellusalueet ovat cleantech sekä terveys ja hyvinvointi. (Kajaanin Yliopistokeskus 2021.)

Yliopistokeskukset perustettiin vuonna 2004, ja ne ovat yliopistolakiin perustuva yliopistojen yhteistyömuoto. Yliopistokeskusten tavoitteena on koota yliopistokaupunkien ulkopuolella tapahtuvaa yliopistotoimintaa muutamaan yliopistokeskukseen, joissa on monipuolista usean yliopiston toimintaa. Ne kokoavat alueellaan tapahtuvan yliopistollisen toiminnan alueen näkökulmasta yhtenäiseksi kokonaisuudeksi, jossa eri yksiköt toimivat. Kajaanin yliopistokeskus on yksi Suomen kuudesta yliopistokeskuksesta. (Kajaanin Yliopistokeskus 2021.)

Kajaanin yliopistokeskus on Oulun yliopiston johtama erillisyksikkö, ja sen alaisuuteen kuuluvat Mittaustekniikan yksikkö sekä Aikuis- ja täydennyskoulutuspalvelut AIKOPAn yliopistollinen osa. Oulun yliopiston koordinoimana Kainuun alueen yliopistollisten toimintojen osaamiskeskittymässä tekevät yhteistyötä myös pääasiassa siellä jo toimintaansa jalkauttaneet Itä-Suomen, Jyväskylän ja Lapin yliopistot. Yliopistokeskus tekee monitieteistä, kansainvälisen tason tutkimusta valituilla tieteellisillä painopistealueilla ja pyrkii edistämään elinikäistä oppimista. Yhteiskunnallisena tehtävänään Kajaanin yliopistokeskus kohottaa toiminta-alueensa osaamisen tasoa ja edistää sen kulttuuria, kilpailukykyä ja hyvinvointia käyttäen avuksi asiakaslähtöistä tutkimus-, koulutus- ja kehittämistoimintaa. Erityistehtävänä yliopistokeskuksella on tukea Kainuun ja Ylä-Savon alue- ja elinkeinokehitystä, kansainvälistymistä ja innovaatiotoimintaa. (Kajaanin Yliopistokeskus 2021.)

2 DEVOPS

Vuonna 2007 Patrick Debois konsultoi Belgiassa ja turhautui ohjelmistokehittäjien ja järjestelmänhaltijoiden erimielisyyksiin. Seuraavana vuonna Debois ilmoittautui ainoana ilmoittautujana Toronton Agile-konferenssissa seminaariin “Agile infrastruktuuri”, mistä johtuen seminaarin pitäjä perui seminaarinsa. Debois kuitenkin onnistui keskustelemaan luennoitsijan kanssa, minkä seurauksena he muodostivat yhdessä ketterän järjestelmän hallintaryhmän. (Paul 2014.)

Kesäkuussa 2009 Debois harmitteli Twitterissä, ettei päässyt osallistumaan O’Reilly Velocity -09 -konferenssiin henkilökohtaisesti. Hänelle twiitattiin takaisin ehdotus järjestää oma Velocity-tapahtuma Belgiassa. Saman vuoden lokakuussa Debois järjesti oman konferenssinsa, jolle hän antoi nimeksi DevOpsDays (kolme ensimmäistä kehitys- ja operaatiokirjainta, joiden lisäksi sana päivät). Vaikuttava määrä ihmisiä saapui paikalle. Konferenssin päätyttyä Debois lyhensi konferenssin hashtagiksi #DevOps, mistä lähtien ajattelumalli on tunnettu nimellä DevOps. Alkuaan DevOps tarkoitti ohjelmistokehittäjien ja -ylläpitäjien välistä saumatonta yhteistyötä. Myöhemmin tähän määritykseen on lisääntynyt ohjelmistokehityksen mahdollisimman pitkälle viety automatisointi ja työkulttuuri sen saavuttamiseksi. (Paul 2014.)

2.1 Rakenne

DevOps perustuu Agilen kehityksen liikkeelle mutta tukee paremmin nopeaa kehitystä ja julkaisutahtia. Nopeampi julkaisutahti auttaa hahmottamaan nykytilanteen ja seuraavat kehittämiskohteet. Toimintamallin saavuttaminen vaatii kaikkien yhteistä panosta, mutta erityisesti siinä korostuvat ohjelmistokehittäjien ja -käyttäjien yhteistyö. Toimintamallin tarkoitus on saada ryhmien välinen yhteistyö toimimaan paremmin. DevOpsiin kuuluu myös erilaisten työkalujen käyttäminen ja proseduurien seuraaminen. Toimintamallissa pyritään välttämään inhimillisiä virheitä ja helpottaa työtä pyrkimällä automatisaatioon ohjelmistopuolella. Olennaisimpiin asioihin keskittymällä toimintamalli yksinkertaistaa yhteistyötä eri työntekijäryhmien välillä. (Aljundi 2018.)

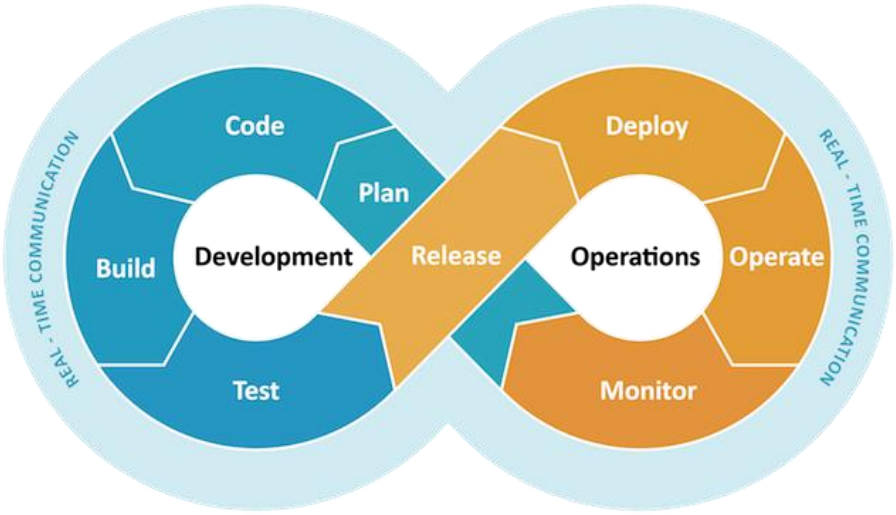
DevOpsin toimintamallin mukaan kehittäjät, testaajat ja operaattorit tekevät tiivistä yhteistyötä (kuva 2). Erilaiset työkalut mahdollistavat yhteistyön lisäksi myös jatkuvan ohjelmistokehityksen ja jakamisen mallin. Toimintamallissa työkalujen rooli on merkittävä. Ne voivat helpottaa versionhallintaa, ohjelmistorakenteen määritystä, valvontaa, tiedonhallintaa, virtualisointia ja automatisointia sekä mahdollistavat yhteistyön ja jatkuvan ohjelmistokehityksen ja jakamisen mallin. (Crawford 2019.)

KUVA 2. Ohjelmistokehityksen erilaisia toimintamalleja ja niiden eroavaisuudet (Analyze 2021)

DevOps-toimintamalli voidaan jakaa kahteen kategoriaan: työpaikkakulttuuriin ja ohjelmistoihin. Ohjelmistoilla edesautetaan ohjelmien suunniteltua toteutusta automaatiota hyödyntäen. Työpaikkakulttuuriin taas kuuluvat työntekijöiden väliset työskentelytavat ja työn johtaminen. DevOps pyrkii korjaamaan kommunikointiongelmia etenkin suurissa yrityksissä ja auttamaan ohjelmoijia, kun työkaluina käytetään työhön soveltuvia sovelluksia. Tämän vuoksi DevOps voidaan ymmärtää ohjelmistokehityksen työkaluna. (Laihonen 2018.)

2.2 Kehityspolku

DevOps ajattelumallin on jaettu kahdeksaan eri vaiheeseen, joista ohjelmiston kehityksessä jokainen käydään läpi alla olevassa kuvassa esitetyssä järjestyksessä (kuva 3). Nämä vaiheet ovat suunnittelu (plan), koodaus (code), rakentaminen (build), testaaminen (test), julkaiseminen (release), käyttöönotto (deploy), käyttö (operate) ja tarkkailu (monitor). Nämä DevOpsin kahdeksan vaihetta suoritetaan silmukassa jatkuvasti, kunnes haluttu tuotteen laatu saavutetaan.



KUVA 3. DevOpsin jatkuva kehittymisen elinkaari (Ecloudvalley 2021)

Suunnittelu

Suunnitteluvaihe kattaa kaiken, mitä tapahtuu, ennen kuin kehittäjät aloittavat koodin kirjoittamisen. Näitä ovat esimerkiksi työn aikataulutus ja tehtävien organisoiminen. Suunnitteluvaiheessa suunnitellaan, mitä työssä tehdään, miksi ja missä järjestyksessä. Sidosryhmiltä ja asiakkailta kerätään vaatimuksia ja palautetta työtä varten, ja niitä käytetään rakentamaan tuotesuunnitelmaa tulevaisuuden kehityksen ohjaamiseksi. (JakobTheDev 2019; Plu 2019.)

Koodaus

Koodaus on vaihe, jossa kehittäjät tekevät muutoksia työstettävän ohjelman lähdekoodiin. Päivityksiä voi tapahtua päivittäin tai viikoittain. Ehdotetut muutokset tarkistetaan ohjelman kääntäjällä, joka auttaa havaitsemaan varhaisia ongelmia, jos niitä esiintyy. Koodin kääntäminen sisältää koodin kasaamisen lisäksi myös jo tehdyn muutoksen testaamista, koodin tarkistuksen ja pakkaamisen. Uusia toimintoja ja päivitettyä koodia integroidaan jatkuvasti jo olemassa olevaan koodiin ja järjestelmiin, jolloin ohjelmisto kehittyy ja muutokset näkyvät loppukäyttäjille. Jenkins on koodausvaiheessa hyvin suosittu työkalu. Aina kun Git-arkistossa tapahtuu muutoksia, Jenkins hakee päivitetyn koodin ja valmistelee koodin koontiversion. Sitten tämä koontiversio välitetään testi- tai tuotantopalvelimelle. (Javatpoint 2018.)

Rakentaminen

Kun lähdekoodi on rakennettu, koodi lähetetään jaettuun tietovarastoon eli kommitoidaan. Tämä voidaan tehdä monella tavalla, mutta yleensä kehittäjä lähettää vetopyynnön, pyynnön uuden koodin yhdistämisestä jaettuun kooditietokantaan. Toinen kehittäjä tarkistaa tehdyt muutokset ja ollessaan tyytyväinen tulokseen ja huomatessaan, ettei ongelmia ole, hyväksyy vetopyynnön. Tällaisen manuaalisen tarkistuksen on tarkoitus olla nopea ja kevyt, ja se tunnistaa ongelmia jo varhaisessa vaiheessa. (JakobTheDev 2019.)

Testaaminen

Testausvaiheessa kehitettyä ohjelmistoa testataan jatkuvasti erilaisten virheiden varalta. Laadunvalvojat voivat testata useita koodipohjia rinnakkain varmistaakseen perusteellisesti, ettei toiminnallisuudessa ole virheitä. Tällaiseen työhön suunniteltuja työkaluja ovat muun muassa Selenium-, TestNG- ja JUnit-nimiset sovellukset. Selenium suorittaa automaatiotestausta, ja TestNG tuottaa raportteja. Testausympäristön simulointiin taas on hyvä käyttää erilaista, esimerkiksi Docker Containers -nimistä sovellusta. Koko testausvaihe voidaan automatisoida jatkuvan integroinnin työkalulla, esimerkiksi Jenkins-nimisellä sovelluksella. Automaatiotestaus säästää erittäin paljon aikaa ja vaivaa, kun sitä vertaa testauksen suorittamiseen manuaalisesti. Raporttien luominen on suuri etu. Testipaketissa epäonnistuneiden testitapausten arviointi yksinkertaistuu automaatiotestauksen myötä, ja testitapausten suorittaminen voidaan ajoittaa ennalta määrättyihin aikoihin. Testauksen jälkeen koodi integroidaan jatkuvasti olemassa olevaan koodiin. (Arvind 2020.)

Julkaiseminen

Julkaisuvaiheessa ohjelmistoversio on valmis käyttöönotettavaksi. Tässä vaiheessa jokainen muutos koodissa käy läpi sarjan manuaalisia ja automaattisia testejä. Kehittäjät voivat piilottaa ja testata uusia ominaisuuksia, jotta asiakkaat eivät näe niitä, ennen kuin ne ovat valmiita toimintaan. (AppStudio 2020.)

Käyttöönotto

Ohjelmiston tai muutoksen manuaalinen käyttöönotto on usein virhealtista ja hidasta. Nämä kaksi ominaisuutta ovat DevOpsin automaation tavoitteellisuuden vastaisia. Inhimillisten virheiden minimoimiseksi ja ajan säästämiseksi automaattinen käyttöönotto on lähes aina hyvä tapa edetä. Kun halutaan pystyä palauttamaan muutos tai muutoksia nopeasti ja luotettavasti ongelmatilanteissa,

on jokaisella muutoksella oltava testattu palautussuunnitelma. Se merkitsee käytännössä muutoksen peruuttamista tai uuden ominaisuuden poistamista käytöstä. (Ryan 2019.)

Käyttö

Operatiivinen tiimi käyttöönottovaiheessa varmistaa, että kaikki toimii sujuvasti. Isäntäpalvelun kokoonpanon perusteella ympäristö mukautuu automaattisesti kuormitukseen, jotta se pystyy käsittelemään aktiivisten käyttäjien lukumääriä. Organisaatiot ovat kehittäneet asiakkailleen erilaisia tapoja antaa palautetta palveluistaan. Tätä tarkoitusta varten on kehitetty työkaluja, jotka auttavat keräämään ja priorisoimaan palautetta, jotta tuotetta voidaan jatkossa kehittää asiakkaiden toiveita vastaavaksi. Loppukäyttäjät tietävät parhaiten, mitä tuotteelta halutaan. Heidän käyttäjäkokemuksensa on ohjelmistokehittäjälle arvokasta. (AppStudio 2020.)

Tarkkailu

Tarkkailu on tärkeä vaihe, jossa seurataan jatkuvasti sovelluksen suorituskykyä ja tallennetaan kaikki tärkeät tiedot ohjelmiston käytöstä. Tiedot analysoidaan sovelluksen oikean toiminnallisuuden tunnistamiseksi. Järjestelmävirheet, kuten vähäinen muisti tai palvelimen tavoittamattomuus korjataan tässä vaiheessa, minkä lisäksi kaikkien ongelmien perimmäinen syy määritetään. Tarkkailu ylläpitää palvelujen turvallisuutta ja saatavuutta. Ongelmat korjataan mahdollisimman pian niiden havaitsemisen jälkeen. (Arvind 2020.)

Tarkkailuun osallistuu olennaisesti ylläpitotiimi, joka seuraa käyttäjien toimintaa virheiden tai järjestelmän virheellisen toiminnan varalta. Suosittuja työkaluja tähän ovat muun muassa Splunk, NewRelic ja Sensu. Työkalujen avulla voidaan auttaa seuraamaan tarkasti sovelluksen suorituskykyä ja palvelimia, ja niiden avulla voidaan myös tarkistaa järjestelmän kunto ennakoivasti. Oikeiden työkalujen käyttäminen parantaa tuottavuutta ja lisää järjestelmien luotettavuutta, mikä puolestaan vähentää IT-tukikustannuksia. Kaikista tärkeistä ongelmista ilmoitetaan kehitystiimille, jotta ne voidaan korjata jatkuvassa kehitysvaiheessa. Tämä johtaa ongelmien nopeampaan ratkaisemiseen. (Arvind 2020.)

2.3 Dokumentointi

Ohjelmistokehittäjän tavoitteena on luoda parhain mahdollinen ohjelma sitä tukevalla parhaimmalla dokumentaatiolla. Ohjelmistokehittäjien tulee kommunikoida niin toistensa kuin esimerkiksi asiakkaiden, palvelun myyjien ja tarjoajien kanssa. Näiden tahojen välinen kommunikointi tapahtuu pääasiassa erilaisten dokumenttien keinoin. Dokumenttien täytyy sisältää kaikki olennaiset asiat ohjelmistokehittäjille, jotta kommunikaatio voisi olla tehokasta ja kaikki ymmärtävät sen sisällön ja toiminnallisuuden.

Hyvä dokumentaatio
1. Toimii kehitysryhmän jäsenten viestintävälineenä
2. Kertoo vaadittavat tiedot ohjelmiston ylläpidolle
3. Antaa tietoja johdolle ja auttaa suunnittelemaan, budjetoimaan ja ajoittamaan ohjelmistojen kehitysprosessi
4. Kertoo käyttäjälle, kuinka ohjelmaa käytetään.

Kommunikaatio on tehokasta, jos dokumentaatiosta löytyy sitä käyttävien henkilöiden tarvitsemat tiedot. Eri kohderyhmille tehdään erilaisia dokumentteja, jotta pystytään ilmaisemaan halutut ja odotetut asiat kohderyhmälle tuomatta esille ylimääräisiä asioita. On kuitenkin haastavaa, että dokumentaatiossa on juuri tarvittavat asiat ja että ne pidetään ajan tasalla ja päivitettyinä. Dokumentoinnin käyttämiseen ei ole standardia, vaan jokainen organisaatio pyrkii löytämään ja luomaan keinon, jolla pystyy palvelemaan tehokkaasti kohderyhmiä. (Saini, Chomal 2014.)

2.4 Versionhallinta

Versionhallinta on työkalu, joka auttaa ohjelmistokehittäjiä rakentamaan ohjelman erilaisista osista ja palasista. Ensimmäisellä kerralla saadaan harvoin tehtyä lopullinen versio, minkä vuoksi on tarpeen tehdä ohjelmasta useampi versio. Isojen tai monimutkaisten ohjelmien versioita on vaikea hallita ilman oikeita työkaluja. Versionhallinta mahdollistaa useamman kuin yhden ohjelmoijan työskentelevän saman projektin kanssa yhtäaikaisesti sekoittamatta muiden työtä. (Atlassian 2021.)

Versionhallinnassa ohjelman pääversio on tallessa versionhallintapalvelimella, josta työntekijät voivat ladata työprojektista työkoneilleen uusimman version. Kun työntekijät ovat tyytyväisiä työhönsä tai ohjelman muutokseen, he lähettävät muutetun osan takaisin versionhallintapalvelimelle. Usein pyritään automaattisesti tarkistamaan muutos tai uusi osa, ennen kuin se hyväksytään muutokseksi itse pääversioon, jotta pääversio saadaan pysymään kunnossa. Kun muutos on hyväksytty pääversioon, on työntekijöiden mahdollista päivittää tietokoneillaan olevan projektin samanlaiseksi kuin pääversio. (Atlassian 2021.)

Yleisimmät versionhallintaohjelmat

Kolme yleisintä versionhallintaohjelmaa ovat Git, SubVersion (SVN) ja Mercurial. Mercurial on aina ollut huomattavasti pienemmässä käytössä verrattuna Gitiin tai SVN:nään. SVN oli pitkään suosituin versionhallintaohjelma, mutta vuosina 2010–2014 Gitin suosio ylitti sen. (Rhodecode 2021.)

Git on Linus Torvaldsin vuonna 2005 kehittämä avoimen lähdekoodin hajautettu ohjausjärjestelmä. Gitin painopiste on nopeudessa ja tiedon eheydessä, jossa ei ole keskitettyä yhteyttä. Versioiden haarautumisella ja yhdistämisillä jokainen kehittäjä saa oman arkiston ja paikallisen kopion, jossa he voivat muuttaa ohjelmaa. Se tukee epälineaarisia kehityshaaroja ja sovelluksia suurella määrällä kooditiedostoja. (Geeksforgeeks 2020.)

SVN eli Apache Subversion on avoimen lähdekoodin ohjelmistoversio ja versioiden hallintajärjestelmä Apache-lisenssillä. Se hallitsee arkistossa olevia tiedostoja ja kansioita (kuva 4). Ohjelmisto voi toimia verkon yli, mikä mahdollistaa SVN:n käytön useilla päätelaitteilla. Voi sanoa, että SVN-arkisto on kuin tavallinen tiedostopalvelin, jonka avulla ihmiset voivat käyttää sitä eri tietokoneella. (Geeksforgeeks 2020.)

GIT	SVN
Git is a Decentralized Version Control Tool	SVN is a Centralized Version Control Tool
Git contains the local repo as well as the full history of the whole project on all the developers hard drive. so if there is a server outage, you can easily recover code from local git repo.	SVN relies only on the central server to store all the versions of the project file
Push and pull operations are fast	Push and pull operations are slower compared to Git
Client nodes can clone the entire repositories on their local system	Version history is stored on the server-side repository
Commits can be done offline too	Commits can be done only online

KUVA 4. Git- ja SVN-versiohallintajärjestelmien eroja (Geeksforgeeks 2020)

2.5 Hyödyt

DevoOps pyrkii tarjoamaan jatkuvan kehityslinjan ohjelmistokehitykseen ja mahdollistamaan nopean ja säännöllisen versioiden julkaisemisen sekä automatisoidut testaukset. DevOps mahdollistaa myös nopeat ohjelmistomuutokset, jos asiakas niin toivoo. DevOpsin avulla kehittäjät ja ohjelman käyttäjät voivat työskennellä yhdessä integroimalla organisaatiojärjestelmiä, yksinkertaistamalla testausta ja laadunvarmennusta ja sekä tasoittamalla kuilua kehityksen ja toiminnan välillä. Toimintamalli avaa mahdollisuuksia eliminoida organisatoristen ja kulttuuristen haasteiden jakautumista ja käsittelee vikojen tunnistamisen kustannuksia jo projektin alkuvaiheessa. DevOps-ympäristössä koodin virheet korjataan heti ohjelmistokehityksen elinkaaren alussa sen jatkuvan käyttöönoton vuoksi. (Liu, Zhou 2017.)

DevOps auttaa luopumaan erilaisista osastojen välisistä viestintäprotokollista. Sen saavuttaminen vaatii kaikkien yhteistä panosta, mutta erityisesti siinä korostuu ohjelmistokehittäjien ja -käyttäjien yhteistyö. Toimintamalli pyrkii saamaan ryhmien välisen yhteistyön toimimaan paremmin ja estämään turhat kommunikointiongelmat, joiden syyt ja ratkaisut löytyvät usein työpaikan kulttuurista.

DevOps pyrkii välttämään inhimillisiä virheitä ja helpottamaan työtä pyrkimällä automatisaatioon ohjelmistopuolella. Keskittymällä olennaisimpiin asioihin toimintamalli yksinkertaistaa yhteistyötä eri työntekijäryhmien välillä. (Aljundi 2018.)

Versionhallinnan hyötyjä on ohjelmistokehityksen yksinkertaistaminen ja nopeuttaminen. Työntekijät voivat vapaammin työskennellä ryhminä ja tiimeinä ja vapaasti milloin vain minkä tahansa tiedoston kanssa rikkomatta ohjelman toiminnallisuutta. Tiedon, tiedostojen ja dokumenttien jakaminen on helppoa. Versionhallinnan käyttäminen säästää muistia ja tilaa, eikä jokaista muutosta varten tarvitse tehdä uutta projektia. Versionhallinnan muistissa on versiohistoria, josta näkee tehdyt muutokset kyseiseen versioon. Versioita on helppoa tarkastella ja niihin on helppo palata, jos esimerkiksi on vahingossa poistanut ja muuttanut jotakin osiota. (Zolkifli, Ngah, Deraman 2018.)

2.6 DevOpsin käyttöönottaminen

Pienet muutokset ovat parhaita nopean oppimisen ja uudistamisen kannalta. Tämän vuoksi työpaikkakulttuurin uudistamiseksi ja muuttamiseksi tehdään mahdollisimman vähän ja pieniä muutoksia kerralla, millä varmistetaan kulttuurin etenevän oikean suuntaan. DevOpsissa käytetään Hoshin ajattelumallia selkeyttämään päämäärät ja toimenpiteet niiden saavuttamiseksi. Näin myös varmistetaan, että ongelmien ratkaisu on yksinkertaista ja kehitystä on luontevaa jatkaa. Tämä tuo etuja etenkin isoissa organisaatioissa tai auktoriteettipohjaisissa johtamisympäristöissä, joissa perusasioiden muuttaminen kestää pitempään. Mitä pienempi muutos työntekijään kohdistetaan, sitä helpommin se otetaan käyttöön ja sitä paremmin sen hyöty nähdään. Pieniä muutoksia on myös helpompi ottaa käyttöön välittömästi. Muutosten dokumentointi helpottuu, ja työpaikalle on mahdollista luoda uutta standardia, joka varmistaa jatkuvaa kehittymistä. (Denning 2010.)

2.6.1 Hoshin ajattelumalli

Hoshin suunnittelumallilla tavoitteista luodaan selkeitä ja kannustavia. Hoshin suunnittelussa johdolta saadaan selvä päämäärä ja keskijohto työntekijöiden kanssa saa vapaasti asettaa siihen pääsemiseksi tavoitteet. Näin työntekijöillä on suuria vapauksia toteuttaa itseään töissä, mikä edesauttaa luovuutta ja sitouttaa työntekijöitä paremmin työhönsä. Tämä vaatii ylemmältä johdolta sel-

vää päämäärää, jotta tavoitteet voidaan asettaa sen saavuttamisen tukemiseen. kuva 5 auttaa selventämään sitä, mitä ollaan tekemässä ja miksi. Kun tarkoitus on selvä, uuden asian sisäistäminen on paljon helpompaa. (Jokinen 2021.)

KUVA 5. Hoshin suunnitteluympyrä (Jokinen 2021)

2.6.2 Jatkuvan parantamisen malli ja Kaizen

Hoshin suunnittelu selventää lähtötilannetta ja sitä, miksi pitäisi päästä tavoitteeseen. Jatkuvan parantamisen malli luo selvät portaat lähtötilanteesta lopputavoitteeseen. Jotta parantaminen olisi jatkuvaa, tarvitaan visio lopputuloksesta, oikeaa tietoa tilanteesta ja selvä päämäärä. Tämän jälkeen asetetaan tavoitteita ja välitavoitteita. Tavoitteita kohti työskennellään ja seurataan niiden edistymistä välillä tarkastellen, ollaanko saavuttamassa haluttua tavoitetta. Asetettujen tavoitteiden saavuttamisen olevan lähellä, asetetaan taas uusia tavoitteita. Tavoitteita asettamalla organisaatio pystyy jatkuvaan parantamiseen eikä jää sen suhteen paikalleen. (Rahko, Mira 2021.)

Kaizen on työkalu työntekijöiden näkemyksen vahvistamiseen omista mahdollisuuksistaan parantaa omaa työympäristöä. Asioiden visualisoiminen helpottaa ymmärtämistä ja yhdessä kommunikointia. Tämän vuoksi esimerkiksi ilmoitustaulun käyttäminen oppimisessa on kannattavaa. Asiat, joissa tarvitsee oppia ja kehittyä, käydään yhdessä läpi. (Rahko, Mira 2021.)

3 DEVOPSIN KÄYTTÖÖNOTTAMISEN MALLI

Työn tavoitteet suunnitellaan työpaikan muiden ohjelmoijien sekä esimiehen kanssa. Opinnäytetyössä luodaan uusi toimintamalli, jonka käyttöönottamisessa on tärkeää saada johdon ja esimiehen tuki. Johdon tuki on edellytys uuden kulttuurin sisäistämiseksi ja muutosten kannalta välttämätön. Esimiehen kanssa varmistetaan hyvissä ajoin toimintamallin sisältämät asiat, jotta se on selkeä ja saa toimivan linjauksen. Tämän jälkeen pidetään työpajoja toimintamallin sisältämistä asioista muiden ohjelmoijien kanssa, jotta kaikki sisäistäisivät toimintamallin sisältämät asiat. Näin kaikki pääsevät suunnittelemaan toimintamallia ja lisäämään siihen omia ajatuksia uuden toimintamallin sisältöön jo alkuvaiheessa. Hoshin ajattelumallissa uuden kulttuurin sisäistäminen on helpompaa, kun sen sisältöön on saanut vaikuttaa. (Jokinen 2021.)

Keskusteluilla ja työpajoilla pyritään kartoittamaan työpaikan nykyhetken tilanne. Työpajoilla ja keskusteluilla varmistetaan, että ymmärretään toisia ja käytetään samaa termistöä uutta toimintamallia käyttöönotettaessa. Uuden kulttuurin sisäistäminen on oppimista, ja se tapahtuu tehokkaimmin, kun kaikki ymmärtävät esitetyn aiheen. (Aljundi 2018.)

Kohdeyrityksessä kaikki ohjelmoijat työskentelevät samassa rakennuksessa. Työntekijöiden välillä ei näin ollen ole pitkää maantieteellistä matkaa, mistä syystä sen aiheuttamat kommunikaatiohaasteet on päästy välttämään. Työpajoissa keskustellaan eri toimintamalleista ja työn käyttöönottamisen tavoista ja sitoudutaan työn tuloksiin. Näin pyritään löytämään hyviä käytäntöjä töiden onnistumiselle. Työn tuloksilla pyritään siihen, että jokaisella työntekijällä on selkeä näkemys työstään ja työtehtävistään.

Uutta kulttuuria aloitettaessa huomioidaan jatkuvan kehityksen kannalta oleellisia asioita. Oppimisen ja mieluisuuden kannalta on merkittävää, että materiaali esitetään selvänä ja sillä on ymmärrettävä tarkoitus. Aluksi tehdään selväksi päämäärä. Sen jälkeen asetetaan tavoitteet ja välitavoitteet ja niiden merkitys, jotta päämäärä saavutetaan. Esimerkiksi Canban-taulukko on selvä, ja siitä näkee tavoitteiden saavuttamisen tilanteen.

3.1 Työssä toteutetut DevOpsin kehitysaskeleeet

Opinnäytetyön tulokset esitetään työpajoissa esimiehelle ja muille ohjelmoijille yksi asia kerrallaan. Jokaisen työpajan alussa käydään käsiteltävä asia läpi käyttäen Hoshin ajatusmallia: miksi asia on tärkeää sisäistää ja toimia sen mukaan. Työpajoissa käytetään Hoshin oppimispohja -dokumenttia (liite 2), jolla varmistetaan yhtenäinen oppimismenetelmä tämän toimintamallin sisäistämisessä. Tätä dokumenttipohjaa voidaan jatkossa käyttää myös muiden asioiden esittelyssä ja oppimisessa. Näin varmistetaan, että uuden asian oppiminen on helppoa ja mahdollisimman vaivatonta. Uutta asiaa ehtii myös päästä kokeilemaan viikon aikana, ennen kuin taas uusi asia opitaan seuraavalla viikolla. Uusia asioita tehtäessä on helpompi sisäistää myös uutta kulttuuria, sillä koko ajan ollaan tekemässä muutostyötä.

Esimiehen tuki ja sitoutuminen ovat välttämättömiä muutoksille ja uuden kulttuurin sisäistämiselle. Esimiehen kanssa sovitaan aina etukäteen työpajassa käsiteltävät asiat, jotta muutokseen sitoutuminen voidaan varmistaa. Aihe esitellään esimiehelle etukäteen lyhyesti ja pääkohdat esiin nostaen. Näin vältetään esimiehen kuormittamista liiaksi.

3.2 Versionhallinta

Ohjeet käyttöönottamisesta ja käyttämisestä

Käytetään ohjelmointikehityksessä teoriaosuudessa esitettyä Git-versionhallintajärjestelmää. Git on yleisesti vähän vaikeampi ottaa käyttöön verrattuna SVN-järjestelmän yksinkertaisuuteen. Jotta DevOpsin tavoitetilaan päästään, käytetään Gitiä sen monipuolisuuden ja muiden lisäosien vuoksi. Kaikki nämä lisäosat ovat yhdessä käytettävänä GitLab-ohjelmistokaaren kehitystyökalussa. Git-ympäristö on tuetumpi CI/CD-ohjelmistoputken kehitykseen kuin SVN. CI/CD-ohjelmistoputkella saadaan työvaiheita vähentämällä nopeutettua ohjelmistokehitystä ja vähennettyä mahdollisia käyttäjän tekemiä virheitä.

GitLabin käyttöönottamisen ohjeina käytetään Kajaanin ammattikorkeakoulun tekemiä videoita aiheesta. Videoista löytyy esimerkkejä, kuinka GitLabia käytetään eri tilanteissa. Videon linkki löytyy liitteestä 3.

Versionnumeron hallinta

Versionumeroinnin (x.x.x) ensimmäinen x tarkoittaa pääversiota eli MAJOR-versionumeroa. Pääversion lukua nostetaan, kun ohjelman taaksepäinyhteensopivuus katkeaa. MAJOR-version ollessa nolla ohjelma on niin sanotusti kehitysversio, johon ennen ensimmäistä virallista julkaisua voidaan tehdä paljon muutoksia. Kun MAJOR-versionumeroa nostetaan yhdellä, aloitetaan MINOR- ja PATCH-versioiden numerointi nollasta. MINOR-versionumeron (keskimmäinen x) nostaminen tarkoittaa sitä, että ohjelmaan on lisätty uusia ominaisuuksia, jotka eivät kuitenkaan riko taaksepäinyhteensopivuutta. Nostettaessa MINOR-versionumeroa yhdellä aloitetaan PATCH-versioiden numerointi nollasta. PATCH-versionumeroa (viimeinen x) nostetaan silloin, kun ohjelmaan tehdään korjauksia. Korjausversiot eivät sisällä uusia ominaisuuksia. (Nollatavu 2018.)

Versionhaaran työnkulku eli workflow

Keskushaaran rikkoutumisesta ei pitäisi olla huolta, kun yksi ihminen tekee projektia. Ongelmia ilmenee todennäköisemmin, kun tekijöitä on useita ja yhteisiä menettelytapoja ei ole. Keskushaara pidetään jatkuvasti toimivana noudattamalla seuraavanlaista menettelytapaa:

1. Kloonataan projekti:
$ git clone git@example.com:project-name.git

2. Luodaan uusi haara:
$ git checkout -b haara

3. Ilmoitetaan muille ohjelmoijille, mitä ominaisuutta aikoo kehittää.

4. Kehitetään ominaisuutta, minkä jälkeen tiedostot lisätään indeksiin kuten tavallisesti
$ git add .

5. Sidotaan muutokset uuteen solmuun:
$ git commit -m "Uusi ominaisuus"

6. Pusketaan haaran solmut etärepositorioon:
$ git push origin haara

7. Jatketaan haaran muokkaamista palaten kohtaan 3, tai liitetään se master-haaraan liitospyynnöllä (merge request) tai paikallisesti komentorivillä.

3.3 Gitin käyttöönotto ja käyttämisen ohjeet

Gitin käyttöönottamisesta on paljon ohjeita internetissä. Kohdeyrityksessä ohjelmointikehityksessä kaikki ovat suomalaisia, joten käytetään suomalaisia ohjeita. Käytetään Turun yliopiston helppokäyttöistä ohjekatalogia, jonka linkki löytyy liitteestä 3. Gitin käyttöönottamisen tueksi näihin ohjeisiin lisätään Gitin yleisimmät käskyt, joka löytyy liitteestä 3. Käskyjä on olemassa enemmänkin, mutta ne ovat harvinaisempia, ja ne haetaan tarvittaessa internetistä.

3.4 DevOpsin käyttöönottamisen haasteet

Johtaminen ja organisaatio

Johtamisessa yleisesti tavattuja ongelmia ovat esimerkiksi puutteellinen ohjelmistokehittäjien ja -ylläpitäjien välinen organisaatiorakenne ja ylimmän johdon osallistumisen puute. Ohjelmistokehittäjät ja järjestelmänvalvojat eivät osaa käyttää uutta tekniikkaa, työkaluja ja menetelmiä, ellei heitä kouluteta siihen. (Jones, Noppen, Lettice 2016.) DevOpsia ei osata aina organisoida tai hallinnoida järjestelmällisesti (Wettinger, Andrikopoulos, Leymann 2015), eikä yrityksen johtoa ja henkilöstöä ole välttämättä koulutettu DevOpsin tehokkaaseen käyttöön (Daneva, Amrit, Erich 2017). Työntekijöiden määrä vaikuttaa myös toimintamallin käyttöönottoon, eikä organisaatiosta ehkä löydy tarpeeksi päteviä työntekijöitä hyödyntämään kaikkia DevOpsin mahdollisuuksia (Pang, Hindle, Barbosa 2020). Pienten projektien julkaisunopeus kärsii ja pitenee, jos käyttöön otetaan DevOpsin täysi rakenteellinen muutos (Hussaini 2014). Tällöin työryhmän on suoritettava hyvin paljon työtä lyhyessä ajassa (Gee 2016), mikä taas rasittaa tarpeettomasti työntekijöitä. Kaikkien DevOps-standardien toteuttaminen hidastaa työryhmien toimintaa ohjelman raskaudella. (Callanan, Spillane 2015.)

Tavoitteet, joilla näitä haasteita lähdetään ratkaisemaan, liittyvät pääasiassa organisatorisiin muutoksiin, koulutukseen ja yksinkertaistamiseen. Esimiehen olisi hyvä olla sama niin ohjelmistokehittäjillä kuin ohjelmistojen ylläpitäjillä. Tällä menetelmällä eri työntekijäryhmien rajapinnat vähenevät, mikä parantaa tiedonkulkua. Yrityksen johdolla ja henkilöstöllä on tärkeää olla koulutusta DevOps-

toimintamallista (Jones, Noppen, Lettice 2016). Koulutuksen avulla johdolla ja työntekijöillä on parempi ja samansuuntainen käsitys DevOpsin toimintaperiaatteista. Jotta toimintamallin tarjoamaa tietoa voidaan käyttää paremmin, sitä on tärkeää osata hallita. Ilman asianmukaista johtamista ja standardeja uusien tuotteiden julkaisuaika pitkittyy, sillä ongelmia voi tulla runsaasti, jos ideologiaa ei sovelleta oikein. Revisiot ja standardit voivat vähentää johtamisen haasteita ja parantaa laatua. Pienessä yksikössä ei kuitenkaan kannata käyttää DevOpsin ajattelumallin kaikkia mahdollisuuksia, sillä niitä on niin paljon. Sen sijaan on hyvä yksinkertaistaa ja räätälöidä asioita yksikön tarpeisiin. (Premchand, Sandhya, Sharmila 2019.)

Kommunikaatio ja viestintä

Yleisimpiä kommunikaatioon ja viestintään liittyviä haasteita ovat etenkin suuressa yrityksessä maantieteellisen jakauman ongelmat ja lähikontaktien uupuminen (Diel, Marczak, Cruzes 2016). Työntekijöillä voi olla vaikeuksia käyttöönottaa DevOps-toimintamallia ja tehdä yhteistyötä tavalla, josta heillä ei ole aiempaa kokemusta (Stackpole 2016). Ohjelmistokehittäjien ja ylläpitäjien yhteistyössä saatetaan myös havaita puutteita (Hussaini 2014) ja muita ryhmiä syytetään, kun ongelmia ilmenee. (Luz, Bonifácioc, Pinto 2019)

DevOpsin käytössä kommunikaatio on yksi tärkeä avain menestykseen. Ohjelmistokehityksen ja -ylläpidon välisen viestinnän luominen on haastavaa, sillä ne ovat aiemmin olleet kaksi erillistä osastoa. Viestintä- ja yhteistyökuilu kehityksen ja toiminnan välillä vaikeuttaa DevOpsin käyttöönottoa. Ilman hyvää viestintää kehittäjien ja käyttöhenkilöstön yhteistyö ei voi toimia hyvin. Viestintä- ja yhteistyökuilu johtaa myös konflikteihin, työn huonoon laatuun ja DevOpsin käyttöönoton estymiseen. (Liu, Zhou 2017.)

Myös maantieteellisellä ja ajallisella etäisyydellä on DevOps-mallissa vaikutusta, eikä henkilökunta välttämättä pysty kommunikoimaan tarpeeksi nopeasti. Ideoiden ja omien työkokemusten jakaminen eri ryhmien välillä voi lieventää haasteita viestinnässä. Tämä tarjoaa myös mahdollisuuden ottaa yhteyttä muihin ja jakaa mielipiteitään. Hyvän viestinnän avulla monet ongelmat voidaan ratkaista jo varhaisessa vaiheessa, ja organisaation henkilökunnan on helpompi luottaa toisiinsa ja auttaa toisiaan. Samalla ryhmän vastuu vahvistuu ja jokaiselle tehtävälle on mahdollista löytää vastuuhenkilö, mikä vähentää työryhmän jäsenten välistä keskinäistä syyttelyä ja lisää henkilökunnan yhtenäisyyttä. (Liu, Zhou 2017.)

Työpaikkakulttuuri

Hyvä työpaikkakulttuuri on vaatimus DevOpsin tehokkaalle käyttöönotolle. Rakentava kulttuuri sisältää avointa viestintää, kannustimia ja vastuun suuntaamista, kunnioitusta sekä luottamusta DevOpsiin. DevOpsin ajattelutapa voi aiheuttaa epätoivottua muutospainetta ja näin myös pelkoa. Tavoitteiden ja kannustimien yhdenmukaistaminen on ensiarvoisen tärkeä kulttuurihaaste ratkaistavaksi, kuten myös palkkioiden ja hyväksyntöjen toteuttaminen. (Liu, Zhou, 2017), (Taft, 2014) DevOps tekee kulttuurimuutoksen tiimien työssä, ja jo itse kulttuurin muutos voi olla organisaatiolle suurempi haaste kuin ohjelmien hallinta (Diel, Marczak, Cruzes 2016). Toimintamalli vaatii sekä kehitys- että ohjelmanylläpitotietoa ja -taitoa (Nybom, Smeds, Porres 2016). Yleensä kehittäjiä ja ohjelmanylläpitäjiä ei kiinnosta toistensa työ, minkä lisäksi kehittäjien työmäärä saattaa kasvaa DevOpsin käyttöönoton myötä. Monimutkainen organisaatiorakenne ja työpaikkakulttuuri tuovat haasteita toimintamallin käyttöönottoon. (Liu, Zhou 2017). Lisäksi työntekijöiden vastarinta voi tuoda haasteita DevOpsin käyttöönottoprosessissa. (Jones, Noppen, Lettice 2016.)

Työpaikkakulttuurin haasteita pystyy vähentämään esimerkiksi pyrkimällä luomaan tiimien ja yksilöiden keskuudessa parempaa luottamusta. Tähän päästään kannustamalla heitä tehokkaaseen viestintään ja keskinäiseen oppimiseen, sillä ne luovat vankan pohjan DevOps-kulttuurille. (Aljundi 2018.) Nopea palaute auttaa lyhentämään oppimisen sykliä. Tilannetta kannattaa tarkastella useasti ja niin auttaa työyhteisöä eteenpäin. Lisäksi yhteinen työ ja jaettu vastuu tuovat luottamusta yhdessä tekemiseen. (Shahin 2015.) Mitä vähemmän muutoksia tehdään kerralla ja mitä selkeämmät vaatimukset ovat, sitä helpompi työpaikkakulttuuria on muokata ja sen haasteita vähentää. Työntekijöiden tulee olla sitoutuneita tehtäviinsä ja johdon sitoutua tukemaan alaisten oppimista. (Kropp, Meier 2015.)

Sanasto ja määrittely

DevOps on uusi ja epäselvä käsite, ja työyhteisöllä voi olla monta erilaista ajatusta siitä, mitä se käytännössä tarkoittaa. Tämän vuoksi työntekijöiden ja yksiköiden välille voi muodostua väärinymmärryksiä. Eri yksiköissä jo saman sanan määritelmä voi olla erilainen. Keskeinen ratkaisu on keskittyminen yhteisiin tavoitteisiin, joihin kaikki osapuolet ovat sitoutuneita. Jotta kommunikointi saadaan toimivaksi, tarvitaan myös halua kehittyä siinä. (Aljundi 2018.)

Työyhteisön yhteisymmärrys siitä, mitä DevOps tarkoittaa, saadaan asettamalla eri ryhmille yhteisiä tavoitteita. Näin tunnistetaan kriittisimmät haasteet ja väärinymmärrykset kehittymisen ja oppimisen kautta sekä ymmärretään mahdollisimman aikaisin, mitä on tarkoitus tehdä. Tavoitteet on tärkeä pitää niin yksinkertaisina, että kaikkien on helppo ymmärtää ne. Kun edetään sellaisten asioiden kanssa, joita voidaan mitata, on mahdollista kehittyä. (Hussaini 2014.)

Arviointi ja laatu

Arviointi ja testaus ovat tapoja, jotka auttavat tietämään, millainen vaikutus DevOpsilla on työhön ja organisaatioon. Suoriutuminen riippuu siitä, kuinka hyvin eri sidosryhmien tarpeet, erityisesti välttämättömät, täyttyvät. Luomalla yhteiset suoritusmittarit, jotka kaikki sidosryhmät ymmärtävät, voidaan suoriutuminen todentaa ja pitää tavoitteet selvillä koko ajan. Myös itse mittareita on hyvä välillä tarkastella arvioiden, kuinka hyviä ne ovat ja ovatko ne päivittämisen tarpeessa. (Hussaini 2014.) Testauksen ja käyttöönoton täydellinen automatisointi asettaa kuitenkin haasteita turvallisuudelle ja luotettavuudelle (Weber, Nepal, Zhu 2016). Testauksen ja arvioinnin haasteeksi voidaan mainita myös testauksen ja laadunvarmistuksen yhteensopivuus jatkuvan kehityksen ja julkaisun parissa. (Farroha, Farroha 2014)

Työkalut

Työkaluilla ja välineillä on merkittävä rooli DevOpsin toteuttamisessa. Työkalut voivat helpottaa versionhallintaa, suunnittelua, ohjelmointiympäristöjen hallintaa, valvontaa, säilöntää, virtualisointia ja automatisointia. Niillä on kuitenkin tarkasti määritellyt tarkoituksensa. (Aljundi 2018.) Vanhojen ohjelmistojen käyttäminen DevOpsin käyttöönotossa on erittäin vaikeaa, ja eri ohjelmistoversiot ja niiden riippuvuudet muista ohjelmistoista vaihtelevat paljon (Ebert, Gallardo, Hernantes, Serrano 2016). Vaikeuksia tulee usein eri ohjelmien käytössä ja etenkin niiden saumattomasti yhdistämisessä. Organisaation valitsemilla työkaluilla voi myös olla haastavaa saavuttaa kattava automaatio ja vähentää manuaalisen työn tarvetta. (Wettinger, Breitenbücher, Leymann 2014.)

DevOpsin jatkuvan kehityspolun osien tukemiseen on suunniteltu erilaisia työkaluja. Kaikissa organisaatioissa samat työkalut eivät kuitenkaan toimi täsmälleen samalla tavalla. Sellaisten työkalujen, jotka eivät tue organisaation jatkuvaa kehityspolkua, ei ole kannattavaa käyttää. (Aljundi 2018.) Jatkuvan integroinnin työkalut auttavat kehittäjiä integroimaan koodin jaettuun arkistoon eri tiimien kesken. Käyttöönottotyökaluilla hallitaan projektien lähettämistä ja vastaanottamista eri ryhmien välillä, jolloin jokainen työntekijä tietää, mikä on kullekin kuuluva projektin osa. (Vijaya, V 2016.)

Työkalu voi olla myös automatisoitu, mikä nopeuttaa ja helpottaa työn tekemistä. Koontityökaluja kutsutaan toisinaan myös projektin kääntötyökaluiksi, ja niitä käytetään suoritettavien sovellusten luomiseen lähdekoodista. Koontityökalut linkittävät, paketoivat ja kokoavat koodin sekä muuntavat sen suorituskelpoiseksi. (Vijaya, V 2016.) Seurantatyökalut ovat DevOpsille välttämättömiä. Kerätty ja saatu tieto voi auttaa ratkaisevasti palvelujen suorituskykyyn ja tehokkuuden varmistamiseksi tarvittaviin päätöksiin. Seurantatyökaluja voi olla useampi, ja niillä on oma painopistealueensa, esimerkiksi ohjelmiston rakenne, suorituskyky tai lokit. Koodianalyysityökalut automatisoivat vikojen ja tietoturvariskien etsimisen ja löytämisen lähdekoodista. Nämä työkalut voivat myös varmistaa koodin laadun ja löytää sille määriteltyjen standardien rikkominen. (Austel, Chen, Mikalsen, Rouvellou, Sharma 2018.)

Sopeutuminen

Sopeutuvuus tarkoittaa DevOps-sovelluksen laadun tai laajuuden omaksuttavuutta. DevOpsin käyttöönoton määritelmä ja tavoitteet on laadittava tarpeeksi selkeiksi, jotta sitä käyttävillä ihmisillä olisi sama käsitys DevOpsin käyttöönoton tavoitteista. Tulkinnanvaraisen ymmärtämisen takia henkilöstö tarvitsee aikaa käsityksen uudelleen omaksumiseen ja johdonmukaiseen ymmärtämiseen. On hyvä myös huomioida, ettei DevOps välttämättä sovi kaikille asiakkaille eivätkä kaikki asiakkaat ehkä halua DevOpsia. (Liu, Zhou 2017.) Tärkeää on myös huomioida se, missä kaikissa tilanteissa organisaatiossa kannattaa käyttää DevOps-toimintamallia ja missä määrin. Paras tilanne on se, että asiat pystytään pitämään mahdollisimman yksinkertaisina ja että toimintamallista otetaan käyttöön vain kaikki organisaation kannalta hyvät asiat. (Aljundi 2018.)

4 DEVOPSIN OSAVAIHEIDEN TOTEUTUS MITYN OHJELMISTOKEHITYKSSÄ

4.1 Suunnittelu

Suunnitteluvaiheessa ohjelmoijan tulee ymmärtää uuden ohjelman tarkoitus ja käyttökohde, minkä jälkeen siltä vaadittavat ominaisuudet ja toiminnallisuusmenetelmät hahmotetaan. Ohjelman ominaisuuksista tehtävä lista helpottaa versionhallintaa ja työn etenemisen seuraamista ja auttaa myös seuraavissa työvaiheissa. Jotta listasta saadaan helposti ymmärrettävä, kannattaa käyttää siihen sopivaa luokkakaaviota. Ohjelman rakenne kannattaa suunnitella jo tässä työvaiheessa, jotta se olisi helposti muokattava ja modulaarinen, sillä muutoksia alkuperäiseen suunnitelmaan tulee todennäköisesti jossain vaiheessa. Tiedonsiirtojen mahdolliset rajapinnat ja siinä vaadittavat ominaisuudet vaativat myös selvitystä.

Projektin alkuvaiheessa, jossa määritellään sovelluksen tarkoitus ja käyttökohde, tarkistetaan ohjelmalle laissa asetetut määräykset. Tällaisia rajoituksia ovat esimerkiksi GDPR (Euroopan unionin yleinen tietosuoja-asetus), rekisterinpitäjän velvollisuudet, mahdollisesti henkilötietojen siirtäminen EU:n tai ETA-alueen ulkopuolelle ja seloste käsittelytoiminnasta.

4.2 Koodaaminen ja versionhallinnan päivittäminen

Ohjelmaan koodataan kerrallaan yksi ominaisuus, ja sen toimivuus testataan. Kun yksi ominaisuus on kunnossa, siirrytään ominaisuuslistan seuraavaan ominaisuuteen ja kasvatetaan versionumeroinnin MINOR-numeroa yhdellä. Näiden toimenpiteiden jälkeen muutokset päivitetään Master-haaraan. Tehtävät korjaukset listataan ja kirjoitetaan versionhallintaan ja dokumentaatioon. Näin syntyy kattava teos koko projektista, mitä on tehty ja mitä haasteita on ollut ja kuinka ne on korjattu.

Versionhallinnan ja dokumentaation päivittäminen tulee tehdä aina, kun yksi ominaisuus tai korjaus saadaan tehtyä. Kun näin tehdään, voidaan helposti tarkastaa tehty työ ja arvioida sen vaikutus.

Päivittäminen onnistuu parhaiten, kun asiat ovat vielä tuoreessa muistissa, mikä myös auttaa ylläpitämään laissa vaadittuja tietoja. Lisäksi se auttaa hahmottamaan ohjelman kehityskaaren ja aikataulullisten tavoitteiden etenemisen.

4.3 Testaaminen

Testaamisessa koodin tarkastamiseen käytetään ohjelmointialustan omaa työkalua tai kolmannen osapuolen sovellusta. Kun uusi ominaisuus on valmis, se täytyy aina tarkistaa mahdollisten virheiden varalta, jotta pääversion toiminnallisuus ei katkea. Myös ohjelman koodin oikeellisuus tarkastetaan turhan koodin eliminoimiseksi.

Kohdeyrityksessä tehdään ohjelmia cleantech- sekä terveys ja hyvinvointi -puolelle. Ohjelmat, joilla on vaikutusta ihmisten terveyteen ja hyvinvointiin, tulee testata huolellisesti. Ohjelman riskejä peilataan sen terveysvaikutuksiin. Riskien ja mahdollisesti haitallisten terveysvaikutusten kasvaessa entistä tärkeämmäksi nousee nimenomaan testaaminen ja mahdollisten uhkien hallinta.

4.4 Julkaisu ja käyttöönotto

Ohjelman julkaisussa ja käyttöönotossa pyritään hyödyntämään mahdollisimman paljon automaatiota. Ohjelmistokehittäjällä on vain yksi paikka, jonne uusin versio lisätään ja jonne ohjelma ohjelma päivittyy loppukäyttäjille. Loppukäyttäjän ohjelma huomaa käynnistysvaiheessa uuden version olevan olemassa ja pakottaa lataamaan sen. Näin vältetään vanhentuneen ohjelman käyttö, jo ratkaistujen virheiden ilmestyminen tai puuttuvat ominaisuudet.

Mittalaitteet, joissa on internet- tai WIFI-yhteys, lataavat aina uusimman version palvelimelta. Kaikissa mittalaitteissa ei kuitenkaan ole tällaista ominaisuutta. Niiden päivittäminen täytyy tehdä osittain tai täysin manuaalisesti. Osittain manuaalisesti tarkoittaa käytännössä sitä, että koska mittalaitteessa ei ole internetyhteyttä, se ei voi heti päivittää uusimpaan versioon. Ensin muodostetaan mittalaitteeseen yhteys älylaitteella, minkä jälkeen ohjelmisto voi päivittyä muuten automaattisesti.

Täysin manuaalisissa päivityksissä jokainen laite täytyy päivittää yksitellen, mikä on hidasta ja virhealtista. Älylaitteen ja mittalaitteiden versionhallinta täytyy olla kunnossa, jotta ne ovat yhteensopivia.

4.5 Ohjelman käyttäminen ja käyttäjän palaute

Käyttäjän ja ohjelmoijan välisen kommunikaation pitää olla matalakynnyksinen, jotta ongelmat ja kehitysideat kulkevat eteenpäin. Käyttäjä ei käytä sovellusta täysin samalla tavalla kuin ohjelmoija. Tätä ajatusta hyödyntäen erilaisia virheitä ja näkemyksiä ohjelman sujuvuudesta löydetään. Käyttäjä ja ohjelmoija listaavat yhdessä tarvittavat muutokset ja uudet ominaisuudet. Yhdessä listaamisen etu on väärinymmärrysten vähentäminen. Tämä muutoslista on oltava selvästi nähtävillä projektin dokumentaatiossa.

Loppukäyttäjä otetaan mukaan ohjelmistokehitykseen mahdollisimman varhaisessa vaiheessa. Tavoite on se, että loppukäyttäjä testaa jokaisen ominaisuuden sitä mukaa, kun ne valmistuvat. Tiivis yhteistyö ohjelmoijan ja käyttäjän kanssa varmistaa halutunlaisen ohjelman.

4.6 Työkalut

Kohdeyrityksessä on monia ohjelmointiympäristöjä, eikä työntekijöillä ole ohjeita kaikkien käyttämiseen. Heillä on kuitenkin ohjeet, mistä välttämättömät työkalut, esimerkiksi koodin virheentarkistus tai työkalu ohjelman toiminnallisuuden tarkastamiseen (myöh. myös debug), löytyvät omasta ympäristöstä. Koodin virheentarkistustyökalulla ohjelman koodista saa turhat ja sen toimintaa hidastavat koodipätkät helposti pois. Jotkin työkalut myös ehdottavat koodille tehokkaampia ratkaisuja. Debug-työkalut tarkastavat ohjelman toiminto toiminnolta ja auttavat määrittämään tarkasti, missä kohdassa koodia ongelma tai virhe sijaitsee. Työkalujen osalta ohjelmaympäristön täytyy olla ohjelmoijalle tuttu, jotta julkaisuvaihe olisi mahdollisimman pitkälle automatisoitu. Ohjelman automatisoitu julkaiseminen tarkoittaa ohjelman tai päivityksen siirtymistä automaattisesti loppukäyttäjän laitteisiin, kun ohjelma on valmis ohjelmointiympäristössä. Julkaisun automatisointi vähentää virheitä ja nopeuttaa työnkulkua.

4.7 Dokumentointi ja mallidokumentointipohjat

Uutena dokumentointipohjana käytetään MITYn vanhaa dokumentointipohjaa pienillä muutoksilla (liite 1). Dokumentointiin käytetään iterointimenetelmää, mikä tarkoittaa dokumenttipohjan olevan alkuun hyvin yksinkertainen, mutta mahdollistaa tarpeen vaatiessa erilaisetkin lisäykset. Ohjelmistokehittäjät voivat tarkistaa dokumentoinnista vaaditut toiminnallisuudet, mikä helpottaa sovitussa suunnitelmassa pysymistä. Dokumenttiin tehdyt muutokset hyväksytään aina esimiehen kautta ennen muutosten käyttöönottamista ohjelmistokehityksessä.

4.8 Tavoitteiden asettaminen

Ohjelmistoprojekteille tehdään lyhyet kuukausitavoitteet, jotka ovat tiedossa projektin työntekijöillä. Tavoitteiden asettaminen selventää työtehtäviä. Tavoitteet asetetaan ilmoitustauluun, josta käy nopeasti ilmi nykytilanne ja edistyminen. Kanban-ilmoitustaulua voidaan käyttää ajattelumallin pohjana. Tämä helpottaa resurssien ja ajan suunnittelussa ja edesauttaa projektin työntekijöiden yhteistyötä. Tavoitteiden edistyminen raportoidaan viikoittain. Työntekijät pääsevät toteuttamaan Hoshin ajattelumallia käytännössä, ja tavoitteiden kautta työn kehitys nopeutuu.

4.9 Mittarit arviointia varten

Jatkuvan kehityksen varmistamiseen tarvitaan mittareita, joiden perusteella tiedetään kehitystä tapahtuvan. Mitattavien tietojen pitää olla helposti kerättäviä, sillä kohdeyrityksen resursseja ei haluta kuormittaa. Mittareina ovat neljä seuraavaa asiaa:

1. Loppukäyttäjien käytettävissä olevan version julkaisunopeus
2. Julkaisun jälkeen tehtävät pakolliset korjaukset
3. Uusien toimintojen julkaisutahti
4. Keskimääräinen palautumisaika, kuinka kauan kestää toipuminen osittaisesta palvelun keskeytyksestä

Näitä mittareita ei ole vielä automatisoitu ohjelmistokehitykseen, joten ohjelmistokehittäjien täytyy alkuun laskea mittarin arvot itse. Tarvittavia tietoja mittareihin löytyy sähköpostista, dokumenteista,

versionhallinnasta ja tietokannoista. Mittareiden tulokset käydään läpi kuukausittain ja kehitetään ohjelmistokehityksen prosessia yhdessä. (Atlassian 2021.)

4.10 Säännölliset työpajat/workshopit

Jotta DevOpsin käyttöönottoa tuettaisiin, pidetään viikoittain 30 - 60 min mittaisia työpajoja tämän opinnäytetyön aiheista, yhdestä tai kahdesta aiheesta kerrallaan. On tärkeää, että jokainen ymmärtää periaatteet toimintatavoista ja työkaluista, jotka tukevat ohjelmistokehitystä. Kun DevOpsin toimintamalli on omaksuttu, siirrytään työpajoissa käsittelemään esimerkiksi standardeja, jotka täytyy ottaa huomioon ohjelmistoprojekteissa.

Työpajojen lisäksi pidetään viikoittainen viikkopalaveri, jossa tarkastellaan ohjelmistokehittäjien ohjelmistodokumentteja, ohjelmistokehityksen kulkua ja mahdollisia tarpeita työn onnistumiseen. Dokumentin käyttämisestä ja päivittämisestä tulee helpommin rutiini, kun sen edistymistä tarkastellaan viikoittain.

4.11 Työn käytännölliset havainnot

Työpajoilla varmistetaan, että ohjelmistokehittäjien käytössä olisi mahdollisimman paljon tietoa ja kehitystoimenpiteistä saadaan mahdollisimman kattavat ja monipuoliset. Laajan tiedonkeruun lisäksi työpajat antavat henkilökunnalle mahdollisuuden osallistua kehittämiseen ja vaikuttaa omien työtehtäviensä tuleviin toimintamalleihin ja työkaluihin. Henkilöstön osallistaminen kehitystoimiin antaa myös hyvät jatkomahdollisuudet jatkuvalle parantamiselle ja kehittämiselle.

Työpajat auttavat sisäistämään yleisiä ajattelutapoja ohjelmistokehityksen suunnitteluun. Uuden kulttuurin käyttöönottaminen ja sisäistäminen on pitkä prosessi. Epävarmuutta työntekijöille kuitenkin tuo se, ratkaisevatko opinnäytetyön tulokset keskeisimmät ongelmat vai jääkö vielä joitain asioita ratkaisematta. Tulevaisuudessa jää nähtäväksi, kuinka hyvin nämä kehitystoimenpiteet hyödyttävät ohjelmoijia ja parantavat yrityksen kilpailukykyä. Kohdeorganisaatiossa ei ole ennen tehty näin mittavaa kehitystoimenpidettä ohjelmistokehityksessä, ja tästä syystä voidaan olettaa, että

suurin osa niin ohjelmistokehityksen arkipäiväisistä kuin myös strategisista toimista menee merkittävästi eteenpäin, ja voidaan todeta projektin olevan juuri tälle organisaatiolle tarpeellinen ja onnistunut.

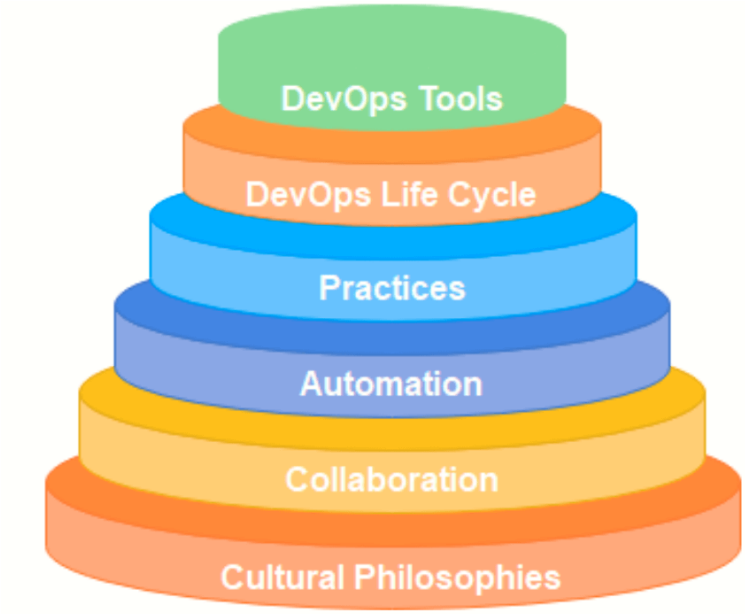
Uskon, että kaikki tässä kehityshankkeessa käynnistetyt toimenpiteet tulevat auttamaan kohdeorganisaatiota parantamaan ohjelmistokehitystä, ja mahdollistavat kaikkien työmukavauuden parantumisen. Tämän johtopäätöksen perustan niin omiin lyhyen aikavälin kokemuksiin jo kehityshankkeen aikana havaituista hyödyistä kuin aiheesta tehtyihin tutkimuksiin. Toki on syytä muistaa, että tämä kehityshanke on tehty juuri kyseiseen organisaatioon opinnäytetyönä, joten työn tuloksiin tulee suhtautua kriittisesti varsinkin niiden soveltuvuuden osalta toisessa organisaatiossa.

Työssä on käsitelty DevOpsin perusteet, hyödyt ja käyttöönottamisen haasteet. DevOps tarkoittaa ohjelmistokehittäjien ja -ylläpitäjien yhteistyön kehittämistä. Tämä ajattelumalli liitetään yleisesti ohjelmistokehityksen työkalujen automatisoitiin ja ohjelman kehitysputkeen. Yhteistyön kehittämisessä työkulttuuri on merkittävä asia, joten DevOps käsittelee myös sitä.

DevOpsin käyttöönottamisen keskeisimpiä haasteita ovat johdon tuen ja sitoutumisen puute. Ilman johdon tukea uuden kulttuurin on vaikea jalkautua ohjelmistokehityksen käytäntöihin. Johdon tuen lisäksi muita yleisiä käyttöönottamisen haasteita ovat väärinymmärrykset ja heikko kommunikaatio. DevOps perustuu jatkuvaan kehitykseen, ja ilman jatkuvaa arviointia ja laadun analysointia kehitys ei ole järjestelmällistä. Ohjelmistokehityksen työkalujen pitää tukea automatisointia ja nopeaa versioiden julkaisutahtia. Kulttuuri ja sopeutuminen voivat myös olla haasteena DevOpsin käyttöönottamisessa, jos uusien asioiden hyväksyminen on hidasta.

Aluksi tarvitaan työpajojen kautta koulutusta yksittäisistä aiheista, jotta Lean-mallinen ohjelmointi voidaan aloittaa. Kun tätä tietoa päästään käyttämään, pystytään luomaan uutta kulttuuria (kuva 6). Vasta sitten yhteistyö ohjelmistokehityksen osapuolten kanssa voi alkaa. Jotta toimintamallin kehitys ei päättyisi tähän, tarvitaan jokaisen työntekijän panostusta ja sitoutuminen sen jatkamiseen. Kun mallille on saatu tuki ja hyväksyntä, aloitetaan ohjelmistokehityksen automatisointi. Kaikki toistettavat ja virhealttiit kohdat, erityisesti ohjelman julkaisu ja päivitysversioiden hallinta, pyritään automatisoimaan. Lopulta työ toteutetaan käytännössä ja sitä päästään tarkastelemaan. Tarkastelun kautta kehitetään toimintatapoja, joiden kautta luodaan hyviä menetelmiä, joista kiinni

pitämällä varmistetaan jatkuva kehitys. Jatkuva kehitys tarkastelee säännöllisesti ja kokonaisvaltaisesti sitä, mitä tehdään. Sen avuksi on suunniteltu työkaluja, jotka helpottavat kokonaisuuden hallinnassa. (Tutorialandexample 2021.)



KUVA 6. DevOpsin käyttöönoton toimintamalli (Tutorialandexample 2021)

5 TULOKSET JA NIIDEN ARVIOINTI

Opinnäytetyö on konstruktiivinen tutkimus, jossa käsitellään DevOps-toimintamallin käyttöönottamista. Teoriatasolla työssä perehdyttiin DevOpsiin ja sen hyötyihin, käyttöönottamisen haasteisiin ja uuden kulttuurin sisäistämiseen. Konstruktio luotiin kohdeorganisaatiolle uuden kulttuurin sisäistämismalliksi, jota käytetään sisäistämään Devopsin toimintamallia ohjelmistokehityksessä. DevOps toimintamallin käyttöönotossa hyödynnetään Hoshin ajattelumallia, joka vahvistaa organisaation sitoutumista. Lean-johtamiselle tyypilliseen tapaan DevOpsin käyttöönottaminen tapahtuu vaiheittain jatkuvan oppimisen periaatteella.

Työssä luodulle konstruktiolle nostetaan keskeisempiä aiheita DevOpsin käyttöönottamiselle, ja osoitetaan sille käyttöönottomenetelmä. Yksittäisiä asioita on paljon, ja niistä muodostuu isompi kokonaisuus. Ennen DevOps-toimintamallin käyttöönottamista varmistetaan johdon tuki ja sitoutuminen työpajojen sisältöön. Työpajojen ohjaajalla täytyy olla ymmärrys esitettävästä aiheesta ja hänen täytyy käyttää termistöä, jonka kaikki ymmärtävät, jotta kommunikointi on kaikille ymmärrettävää. Pienessä yksikössä kaikkien ohjelmoijien osallistuminen työpajoihin on merkittävää yhteneväisen kulttuurin luomiseksi.

5.1 Tulosten arviointi

Tässä työssä sovelletaan DevOpsin käyttöönottamisen ajattelumallia yhdistettynä tunnettuun uuden kulttuurin sisäistämiseen. Työn kohdeorganisaatio on hyvin pieni, ja näin ollen se poikkeaa suurten ja keskikokoisten organisaatioiden DevOpsin käyttöönottamisen menettelytavoista. Työssä keskityttiin myös kohdeyrityksen yksittäisiin ennalta määriteltyjen tavoitteiden ratkaisemiseen. Työssä ratkaistaan tunnetuilla teorioilla kohdeyrityksen yksittäisiä ongelmia. (Gunawan, Budiardjo 2021.)

Yleisesti tutkimusten lopputulosta arvioidaan heikolla tai vahvalla markkinatestillä, joilla arvioidaan työn käytännöllinen arvo. Vahvalla markkinatestillä tarkoitetaan sitä, että organisaatiossa on saavutettu todistettavasti taloudellista tai käytännön hyötyä. Heikko markkinatesti taas tarkoittaa koh-

deyrityksen hyväksyneen ratkaisun ja ottaneen sen toiminnassaan käyttöönsä. Tämä työ arvioidaan heikolla markkinatestillä, sillä työn toimeksiantajan hyväksyttyä työn esittämän toimintamallin se on tarkoitus ottaa käyttöön kohdeyrityksessä. Työn edistymisestä keskustellaan säännöllisesti, tarkastellaan, eteneekö työ suunnitelmien mukaisesti, ja sitä korjataan tarvittaessa. Työn päämäärän ja kehitysaskeleiden ollessa selviä DevOpsin käyttöönotto aloitetaan. (Jokinen 2012.)

Työn konstruktio on kohdeyritykseen suunniteltu toimintamalli. Näin ollen konstruktio ei ole yhtenevä muiden vastaavien ongelmien ratkaisujen kanssa. Erityisesti isommissa kohdeorganisaatioissa DevOpsin käyttöönotto on erilaista, sillä tämän työn konstruktion luomisessa jokainen ohjelmoija oli aktiivisesti mukana. Opinnäytetyö tehtiin kohdeyrityksen tarpeisiin, eli työllä vastattiin tutkimusotteen kysymyksiin. Opinnäytetyö kertoo, kuinka työ otetaan käyttöön. Näiden asioiden vuoksi työn tulos on yksittäistapaus DevOpsin käyttöönottamisesta. (Jokinen 2012.)

5.2 Ajatukset jatkosta

Kehityshankkeessa luotu suunnitelma sekä ohjelmointikehityksen toimintamalli antaa organisaatioille mahdollisuuden päivittää osaamista ja kannustaa yhteistyön hengessä oppimaan uutta ja kehittämään ohjelmoimisen menetelmiä. Toimintamalli mahdollistaa jatkuvan kehityksen niin yksilöiden toiminnan kuin projektien toimintamallin osalta. Jatkoa ajatellen luotiin yksinkertaiset työkalut, joita ylläpitämällä varmistetaan jatkossa keskeisten asioiden kanssa työskentely ja vältetään turhaa tekemistä.

Organisaatioiden harteille jää kulttuurin muodostaminen ja sen kehittäminen. Uusien asioiden sisäistäminen ja menettelytapojen varmistaminen rutiineissa vaatii esimiehiltä ja työntekijöiltä yhteistä sitoutumista. Väärin ymmärrettynä toimintamallin käyttäminen organisaatioissa lisää ajankulumista, mitä sillä juuri pyritään välttämään. Yhdessä työskentely ja kehittyminen on kaikille työn ilo.

Toimintamallin lisäksi tulisi miettiä DevOpsin näkyvyyttä ulkopuolisille tahoille ja sen lisäämistä. Organisaatioiden on hyvä selvittää, olisiko toimintamallille järkevää lisätä näkyvyyttä eri yhteistyötahojen kanssa ja esimerkiksi markkinoinnissa. Sosiaalinen media on hyvä vaihtoehto tämäntyyppiselle markkinoinnille.

6 YHTEENVETO

Opinnäytetyössä luotiin toimintamalli ohjelmistokehitystyöhön. Toimintamalli toimii ohjeena ohjelmistokehitystyön vaiheiden suunnittelussa ja toiminnassa. Opinnäytetyössä päästiin tavoitteeseen ja luotiin malli ohjelmistokehityksen kehittämiseen. Työssä onnistuttiin luomaan selvät työpajojen aiheet, mistä kohdeyritys voi aloittaa DevOps toimintamallin käyttöönottamista. Työn tekemisessä oli haasteena nähdä yhteneväisyyksiä ohjelmoijien työssä, sillä jokainen työskentelee omassa ohjelmistoympäristössään. Tulevaisuudessa pitäisi mitata DevOpsin käyttöönottamisen astetta, jotta tämän työn vaikuttavuutta voitaisiin tarkemmin todeta.

Teorian perusteella rakennettiin konstruktio, joka tässä työssä oli tehdä ohjeet pienelle yksikölle DevOpsin käyttöönottamisesta. Toimintamallin laatimisessa otettiin teorian lisäksi huomioon toimeksiantajayrityksen sisäinen ohjeistus ja tarpeiden täyttäminen. Toimintamallin luontivaiheessa tehdään yhteistyötä lopullisten käyttäjien mielipiteiden kuulemiseksi. Lopulta teoriaa, ohjeistuksia ja käyttäjien kommentteja kuulemalla luodaan toimintamalli DevOpsin käyttöönottamisesta. Opinnäytetyössä toteutettu kehittämistyö on kohdeyrityksen kannalta järkevää toteuttaa.

LÄHTEET

Aljundi, Mohamed 2018. *Tools and Practices to Enhance DevOps Core Values*. Haettu 25. Helmikuu 2021 https://lutpub.lut.fi/bitstream/handle/10024/148944/DevOps.pdf?sequence=1

Analyze. 2018. *The Transition to DevOps*. Noudettu osoitteesta 8. Joulukuu 2021. https://analyze.co.za/the-transition-to-devops/

AppStudio. 2020. *DevOps: The Complete Guide To Understand DevOps Lifecycle*. Haettu 19. Toukokuu 2021. https://www.appstudio.ca/blog/devops-lifecycle/

Arvind. 2020. *DevOps Life cycle: Everything You Need To Know About DevOps Life cycle Phases*. Haettu 19. Toukokuu 2021. https://www.edureka.co/blog/devops-lifecycle/

Atlassian. 2021. *DevOps metrics Why, what, and how to measure success in DevOps*. Haettu 11. Marraskuuta 2021. https://www.atlassian.com/devops/frameworks/devops-metrics

Atlassian. 2021. *What is version control?* Haettu 12. Marraskuuta 2021. https://www.atlassian.com/git/tutorials/what-is-version-control

Austel, Paula, Chen, Han, Mikalsen, Thomas, Rouvellou, Isabelle, Sharma, Upendra, Silva-lepe, Ignacio, Subramanian, Revathi 2018. *Continuous Delivery of Composite Solutions: A Case for Collaborative Software Defined PaaS Environments*. Haettu 20. Toukokuu 2021. https://dl.acm.org/doi/pdf/10.1145/2756594.2756595

Callanan, Matt, Spillane, Alexandra 2015. *DevOps Making It Easy to Do the Right Thing*. Haettu 18. Maaliskuu 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7436644

Crawford, Andrea 2019. *DevOps*. Haettu 9. Maaliskuu 2021. https://www.ibm.com/cloud/learn/devops-a-complete-guide

Daneva, M., Amrit, C., Erich, F. M.A. 2017. *A qualitative study of DevOps usage in practice*. Haettu 10. Marraskuuta 2021. https://onlinelibrary.wiley.com/doi/full/10.1002/smr.1885

Denning, Stephen 2010. *New lessons for leaders about continuous innovation*. Haettu 12. Maaliskuu 2021. https://www.emerald.com/insight/content/doi/10.1108/SL-11-2014-0083/full/pdf?title=new-lessons-for-leaders-about-continuous-innovation

Diel, Elisa, Marczak, Sabrina, Cruzes, Daniela 2016. *Communication Challenges and Strategies in Distributed DevOps*. Haettu 18. Maaliskuu 2021. https://repositorio.pucrs.br/dspace/bitstream/10923/14121/2/Communication_Challenges_and_Strategies_in_Distributed_DevOps.pdf

Ebert, Christof, Gallardo, Gorka, Hernantes, Josune, Serrano, Nicolas 2016. *DevOps*. Haettu 18. Maaliskuu 2021. https://ieeexplore.ieee.org/abstract/document/7458761./authors#authors

Ecloudvalley 2021. *What is DevOps?* Haettu 8. Joulukuu 2021. https://www.ecloudvalley.com/my/devops/

Farroha, B.S., Farroha, D.L. 2014. *A Framework for Managing Mission Needs, Compliance and Trust in the DevOps*. Haettu 10. Maaliskuu 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6956773

Gee, David 2016. *Why Agile and DevOps is shaking up management*. Haettu 18. Maaliskuu 2021. https://www2.cio.com.au/article/599283/why-agile-devops-shaking-up-management/

Geeksforgeeks. 2020. *Difference Between GIT and SVN*. Haettu 9. Lokakuuta 2021. https://www.geeksforgeeks.org/difference-between-git-and-svn/

Gunawan, Fandi, Budiardjo, Eko Tammikuu 2021. *A Quest of Software Process Improvements in DevOps and Kanban: A Case Study in Small Software Company*. Haettu 15. Kesäkuuta. https://dl.acm.org/doi/fullHtml/10.1145/3451471.3451478

Hussaini, Syed 2014. *Strengthening harmonization of Development (Dev) and Operations (Ops) silos in IT environment through Systems approach.* Haettu 18. Maaliskuu 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6957687

JakobTheDev 2019. *The Eight Phases of a DevOps Pipeline*. Haettu 11. Maaliskuu 2021. https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba

Javatpoint 2018. *DevOps Lifecycle*. Haettu 19. Toukokuu 2021. https://www.javatpoint.com/devops-lifecycle

Jokinen, Tauno 2012. *Konstruktiivinen tapaustutkimus ja suunnittelutiede – kaksi insinööritieteisiin soveltuvaa tutkimusotetta*. Haettu 19. Syyskuu. https://blogi.oamk.fi/2021/02/19/konstruktiivinen-tapaustutkimus-ja-suunnittelutiede-kaksi-insinooritieteisiin-soveltuvaa-tutkimusotetta/

Jokinen, Tauno 2021. *Hoshin Kanri on strategista pallottelua*. Haettu 18. Marraskuu. https://oamk.sharepoint.com/sites/oamk_konewithpassion/Jaetut%20asiakirjat/Forms/AllItems.aspx?id=%2Fsites%2Foamk%5Fkonewithpassion%2FJaetut%20asiakirjat%2FTuotantotekniikka%5Fwith%5Fpassion%2FKurssien%20opetusmateriaalit%2Fstrateginen%5Flean%2FLean%2Dleh

Jones, Stephen 2020. *Changing Software Development Practice: A Case Study of DevOps Adoption*. Haettu 12. Kesäkuu 2021. https://www.researchgate.net/profile/Stephen-Jones-16/publication/349554530_Changing_Software_Development_Practice_A_Case_Study_

of_DevOps_Adoption/links/603644384585158939c6117b/Changing-Software-Development-Practice-A-Case-Study-of-DevOps-Adoption.pdf

Jones, Stephen, Noppen, Joost, Lettice, Fiona 2016. *Management Challenges for DevOps Adoption within UK SMEs*. Haettu 19. Toukokuu 2021. https://dl.acm.org/doi/pdf/10.1145/2945408.2945410

Kajaanin Yliopistokeskus 2021. Haettu 2. Helmikuu 2021. https://www.oulu.fi/mittaustekniikka/

Kajaanin Yliopistokeskus. 2021. Haettu 2. Helmikuu 2021. https://www.oulu.fi/kajaaninyliopistokeskus/node/53638

Kropp, Martin, Meier, Andreas 2015. *Agile Success Factors - A qualitative study about what makes agile projects successful*. Haettu 12. Syyskuu 2021. https://www.researchgate.net/profile/Martin-Kropp/publication/278024922_Agile_Success_Factors_-_A_qualitative_study_about_what_makes_agile_projects_successful/links/55797e1f08aeacff2003c190/Agile-Success-Factors-A-qualitative-study-about-what-makes-agile-

Laihonen, Paul. (17. Syyskuu 2018). *Adoption of DevOps Practices in*. Haettu 8. Joulukuu 2021. https://aaltodoc.aalto.fi/bitstream/handle/123456789/34400/master_Laihonen_Paul_2018.pdf?sequence=1&isAllowed=y

Lenarduzzi, Valentina, Taibi, Davide, Tosi, Davide, Lavazza, Luigi, Morasca, Sandro 2020. *Open Source Software Evaluation, Selection, and Adoption: a Systematic Literature Review*. Haettu 12. Syyskuu 2021. https://www.researchgate.net/profile/Valentina-Lenarduzzi/publication/341914055_Open_Source_Software_Evaluation_Selection_and_Adoption_a_Systematic_Literature_Review/links/5ed9474f92851c9c5e7cc323/Open-Source-Software-Evaluation-Selection-and-Adoption-a-S

Liu, Yilei, Zhou, Yiran 2017. *The Challenges and Mitigation Strategies of*. Haettu 9. Maaliskuu 2021. https://www.diva-portal.org/smash/get/diva2:1078159/FULLTEXT02

Luz, Welder, Bonifácioc, Rodrigo, Pinto, Gustavo 2019. *Adopting DevOps in the real world: A theory, a model, and a case study*. Heettu 15. Toukuu 2021 https://www.sciencedirect.com/science/article/pii/S0164121219301517

Nollatavu 2018. *Ohjelman versionumerointi*. Haettu osoitteesta 8. Lokakuu 2021. https://nollatavu.fi/2018/ohjelman-versionumerointi/

Nybom, Kristian, Smeds, Jens, Porres, Ivan 2016. *On the Impact of Mixing Responsibilities Between Devs and Ops*. Haettu 18. Maaliskuu 2021. https://www.researchgate.net/publication/303182727_On_the_Impact_of_Mixing_Responsibilities_Between_Devs_and_Ops

Pang, Candy, Hindle, Abram, Barbosa, Denilson 2020. *Understanding DevOps Education with Grounded Theory*. Haettu 4. Toukokuu 2021. https://dl.acm.org/doi/pdf/10.1145/3377814.3381711

Paul, Fredrik 2014. *The Incredible True Story of How DevOps Got Its Name*. Haettu 9. Maaliskuu 2021. https://blog.newrelic.com/engineering/devops-name/

Plu, Clémence 2019. *Understanding the DevOps Process*. Haettu 11. Maaliskuu 2021. https://www.padok.fr/en/blog/devops-process

Premchand, Anshu, Sandhya, M., Sankar, Sharmila 2019. *Simplification of application operations using cloud and DevOps*. Haettu 2. Kesäkuu 2021. https://www.researchgate.net/publication/330159554_Simplification_of_application_operations_using_cloud_and_DevOps/fulltext/5c55162a458515a4c7515563/Simplification-of-application-operations-using-cloud-and-DevOps.pdf?_sg%5B0%5D=lPKX53EHTMV4PcbwBdvUrUZsIYk

Rahko, M.;& Mira, K. (2021). *Kaizen*. Haettu 14. Lokakuu 2021. https://oamk.sharepoint.com/sites/oamk_konewithpassion/Jaetut%20asiakirjat/Forms/AllItems.aspx?id=%2Fsites%2Foamk%5Fkonewithpassion%2FJaetut%20asiakirjat%2FTuotantotekniikka%5Fwith%5Fpassion%2FKurssien%20opetusmateriaalit%2Fstrateginen%5Flean%2FLean%2Dleh

Rhodecode 2021. *Version Control Systems Popularity in 2016*. Haettu 26. Syyskuu 2021 https://rhodecode.com/insights/version-control-systems-2016

Ryan 2019. *DevOps Methodology, Lifecycle and Best Practices explained*. Haettu 19. Toukokuu 2021. https://www.ryadel.com/en/devops-methodology-lifecycle-best-practices-explained/

Saini, Jatinderkumar, Chomal, Vikas 2014. *Significance of Software Documentation in Software*. Haettu 15. Kesäkuu 2021. https://www.researchgate.net/profile/Jatinderkumar-Saini/publication/281965276_Significance_of_Software_Documentation_in_Software_Development_Process/links/55ffdc6008aeba1d9f841187/Significance-of-Software-Documentation-in-Software-Development-Process.pdf

Shahin, Mojtaba 2015. *Architecting for DevOps and Continuous Deployment*. Haettu 14. Toukokuu 2021 https://www.researchgate.net/profile/Mojtaba-Shahin/publication/283620959_Architecting_for_DevOps_and_Continuous_Deployment/links/5641837808aeacfd893661e8/Architecting-for-DevOps-and-Continuous-Deployment.pdf

Spinellis, Diomidis 2012. *Don't Install Software by Hand*. Haettu 11. Elokuu 2021 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6265084

Stackpole, Beth 2016. *Forecast 2016: 5 disruptors to keep on your radar*. Haettu 18. Maaliskuu 2021. https://www.computerworld.com/article/3010563/forecast-2016-5-disruptors-to-keep-on-your-radar.html

Taft, Darryl 2014. *Rackspace Survey Spotlights DevOps Business Benefits: Top 6 Findings*. Haettu 10. Maaliskuu 2021. https://www.eweek.com/development/rackspace-survey-spotlights-devops-business-benefits-top-6-findings/

Tutorialandexample 2021. *DevOps Architecture*. Haettu 1. Marraskuu 2021. https://www.tutorialandexample.com/devops-architecture/

Vijaya, Aparna, Venkataraman, Neelanarayanan 2016. *A Model Driven Framework for Portable Cloud Services*. Haettu 16. Huuhtikuu 2021. https://www.researchgate.net/profile/Neelanarayanan-Venkataraman-2/publication/288061220_A_Model_Driven_Framework_for_Portable_Cloud_Services_Proof_of_Concept_Implementation/links/56d3f99d08ae4d8d64a80287/A-Model-Driven-Framework-for-Portable-Cloud-Servic

Weber, Ingo, Nepal, Surya, Zhu, Liming 2016. *Developing Dependable and Secure Cloud Applications*. Haettu 18. Maaliskuu 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7465693

Wettinger, Johannes, Andrikopoulos, Vasilios, Leymann, Frank 2015. *Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications*. Haettu 18. Maaliskuu 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7092900

Wettinger, Johannes, Breitenbücher, Uwe, Leymann, Frank 2014. *DevOpSlang - Bridging the Gap between Development and Operations*. Haettu 18. Maaliskuu 2021. https://link.springer.com/content/pdf/10.1007%2F978-3-662-44879-3.pdf

Zolkifli, Nazatul, Ngah, Amir, Deraman, Aziz 2018. *Version Control System: A Review*. Haettu 14. Syyskuu 2021. https://www.sciencedirect.com/science/article/pii/S1877050918314819

Projektin nimi

System Architecture

Laatijat: xxxxxxx
Pohja: xx.x.2021
Päivitetty: X.X.2021, XX
Oulun yliopiston Mittaustekniikan yksikkö

Sisällys

1. Johdanto 4
1.1. Ohjelmiston tausta ja tarkoitus 4
1.2. Yleiskatsaus dokumenttiin 4
1.3. Yleiset vaatimukset 4
2. Yleiskuva järjestelmästä 5
3. XXXXX-mittalaite 6
3.1. Yleinen toiminta 6
3.2. Teknilliset kuvat 6
3.3. Ohjelmisto 6
3.4. Toiminnallisuudet/ohjelmistovaatimukset 6
3.5. Aikatavoitteet 7
4. Järjestelmän ulkoiset liittymät 8
4.1. Käyttöliittymä 8
4.2. Alustustiedot 8
5. Toteutusratkaisut 9
5.1. Toteutuksen toimintaperiaate 9
5.2. Muut käytetyt aputyökalut 9
5.3. Versionhallinta 9
6. Loppukäyttäjien palaute 10
6.1. Ohjelmassa hyviä asioita 10
6.2. Kehitettäviä asioita 10
Lähdeluettelo 11
Liitteet 12

Mahdollinen erikoissanasto ja käytetyt lyhenteet

Viitteet muihin mahdollisesti käytössä tarvittaviin dokumentteihin

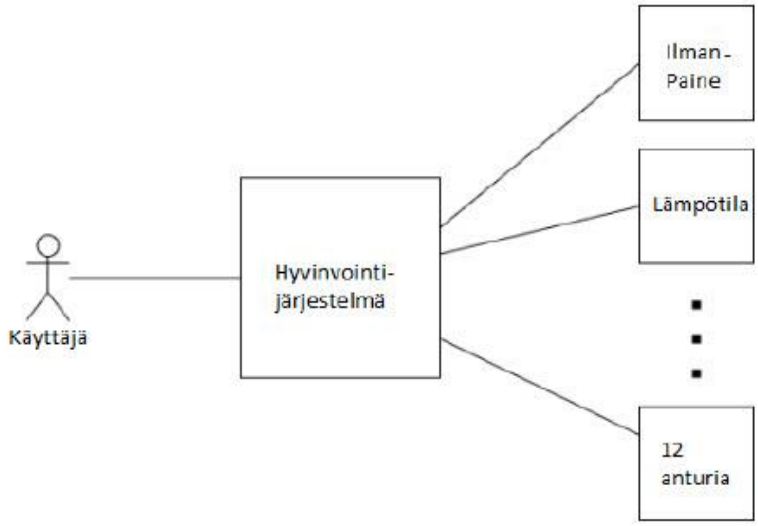
1. Johdanto

1.1. Ohjelmiston tausta ja tarkoitus

- Tähän kirjoitetaan esimerkiksi taustaa, käyttötarkoitusta, millaiselle laitteistolle &käyttöjärjestelmälle tarkoitettu, apukirjastot, käyttötapa ja jne...

1.2. Yleiskatsaus dokumenttiin

- Mitä tämä dokumentti käsittelee?



Kuva x. Kuvataan ohjelmistoa yksinkertaisella kuvalla. Oma kuva

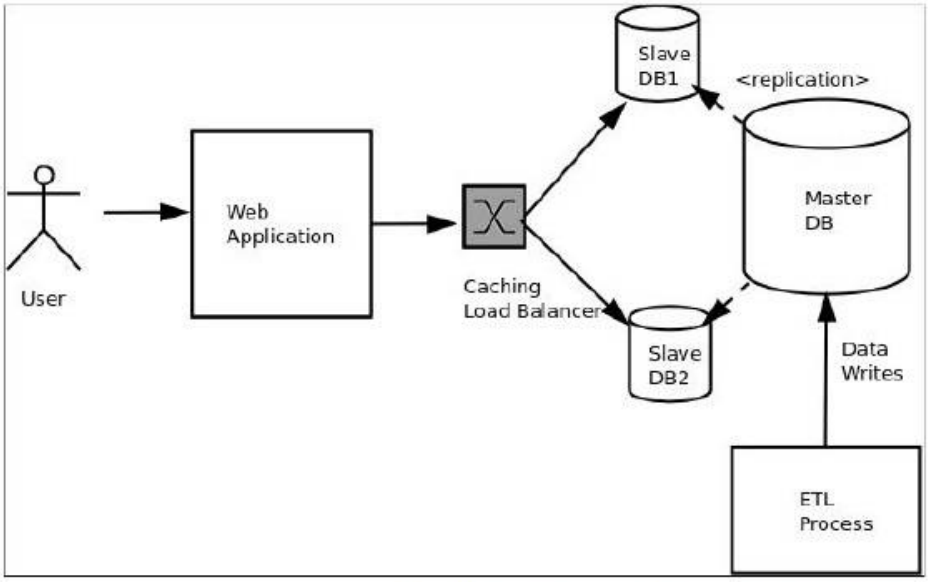
1.3. Yleiset vaatimukset

Selvitä alla olevat vaatimukset ohjelmalle projektin alussa.

- Terveyspuolen ohjelmilla on vaatimuksia, jos ohjelma vaikuttaa terveyteen. Esim. insuliinipumppu.
- Tietosuojakäytännöt
- Tiedon luotettava varastointi

2. Yleiskuva järjestelmästä

Kerrotaan lyhyesti yleiskuva, mitä laitteita järjestelmään kuuluu ja mitä tiedonvälistys tekniikkaa käytetään. Kuva OMT-menetelmä (Object modeling technique) tai teknisellä menetelmällä.

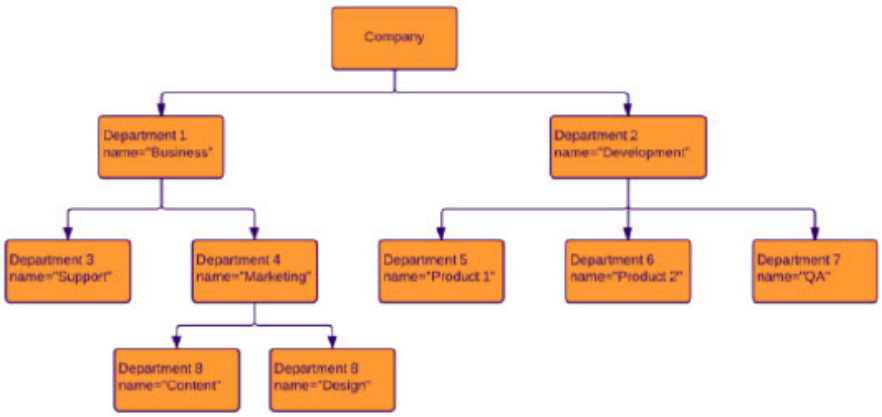


Kuva x. Kuvalla aukastaan järjestelmää. https://subscription.packtpub.com/book/application-development/9781786468529/1/ch01lvl1sec08/characteristics-of-software-architecture

3. XXXXX-mittalaite

3.1. Yleinen toiminta

- Mitä järjestelmän pitäisi tehdä?



Kuva x. Tämmöinen tai vastaava kaavio, mistä käy ilmi, mitä mistä ominaisuuksista ohjelma koostuu. https://www.lucidchart.com/pages/uml-object-diagram

3.2. Teknilliset kuvat

Lisätään tähän teknillisiä kuvia, mitä tietoja vaaditaan ohjelmistokehitykseen.

3.3. Ohjelmisto

Kerrotaan tässä ympäristöstä ja kuinka sitä hallitaan.

Avataan mittalaitteen toimintaa ja kuvataan mahdolliset rajapinnat.

3.4. Toiminnallisuudet/ohjelmistovaatimukset

Tässä kerrotaan kaikki ominaisuudet, mitä ohjelmistoon tarvitaan.

Parametrien mittaus

Parametrien lähetys kantaan.

Parametrien näyttäminen graafissa.

Käyttäjä pystyy näkemään graafit

3.5. Aikatavoitteet

- Milloin on tarkoitus, että seuraavat osiot ovat valmiita:

Osa 1 on valmis xx.xx.xxxx.

Osa 2 on valmis xx.xx.xxxx.

Osa 3 on valmis xx.xx.xxxx.

Osa 4 on valmis xx.xx.xxxx.

4. Järjestelmän ulkoiset liittymät

4.1. Käyttöliittymä

- Tähän kirjoitetaan mahdollisen prototyypin kuvaus, kuvaus tyypillisistä käyttöskenaarioista ja kuvia hahmotellusta käyttöliittymästä (prototyypistä saatuina tai piirrettyinä)

4.2. Alustustiedot

- Millaisia tietoja alustukseen vaaditaan / mitä jo on / mitä halutaan kertoa?

5. Toteutusratkaisut

5.1. Toteutuksen toimintaperiaate

- Rajapintojen toiminnallisuus.

- Hyvin lyhyt ohje ohjelman käyttämisestä.

5.2. Muut käytetyt aputyökalut

- Jos muitakin aputyökaluja on käytetty, kerro se tässä. Miksi niitä käytettiin?

5.3. Versionhallinta

- Mistä hakemistoissa ja tiedostoissa on mitkäkin moduulit?

6. Loppukäyttäjien palaute

6.1. Ohjelmassa hyviä asioita

- **Kuka sanoi mitä ja milloin?**

6.2. Kehitettäviä asioita

- **Kuka sanoi mitä ja milloin?**

Lähdeluettelo

Liitteet

- Käyttöohjeet loppukäyttäjälle
- Moduulien lähdekoodi
- Käytetyt kääntäjien optioasetukset
- Testiaineisto (testitapaukset, saadut tulokset, testilokit)

Hoshin oppimispohja dokumentti

1. Miksi. Mikä on päämäärä?

2. Miten. Mitkä ovat toimenpiteet?

3. Mitä. Mikä on lopputulos?

Kuva aiheesta.

OPINNÄYTETYÖSSÄ MAINITTUJA LINKKEJÄ

LIITE 3

Nimi	Linkki
GitLabin käyttöönotto	https://www.youtube.com/playlist?list=PL0b_f_eBMYdXwlEgcavttSkYfe5oc_8lX
Git:in käyttöönotto	https://vm.utu.fi/document/fi_pieni-git-opas.pdf
Git:in yleisimmät käskyt	https://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf