Tampereen yliopisto

Samuli Kohomäki

DEVOPS-KÄYTÄNNÖT OPETUSJÄRJESTELMIEN KEHITYKSESSÄ JA YLLÄPIDOSSA

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Helmikuu 2019

TIIVISTELMÄ

SAMULI KOHOMÄKI: DevOps-käytännöt opetusjärjestelmien kehityksessä ja ylläpidossa
Tampereen yliopisto
Diplomityö, 65 sivua, 8 liitesivua
Helmikuu 2019
Diplomi-insinöörin tutkinto, Tietotekniikka, DI
Pääaine: Ohjelmistotuotanto
Tarkastajat: Yliopistonlehtori Terhi Kilamo ja Associate Professor (tenure track) Kari Systä

Avainsanat: DevOps, opetusjärjestelmät, opetuskäyttö

DevOps on kasvattanut suosiotaan viimeisen viiden vuoden aikana varsin tasaisesti Google Trends -palvelun perusteella. Myös tieteellinen kiinnostus termiä kohtaan on kasvanut: esimerkiksi IEEE:n Xplore-hakupalvelussa suoritettujen hakujen perusteella IEEE on julkaissut yli tuplaten DevOpsiin liittyviä artikkeleita vuonna 2017 verrattuna vuoteen 2015. DevOpsin käsite on kuitenkin vielä monitulkintainen.

DevOpsia on myös opetettu. Esimerkiksi Japanin Saga Cityn yliopisto on hyödyntänyt siihen liittyviä työkaluja opetuksessaan, ja Moroccon Cadi Ayyad University on järjestänyt aiheesta kurssin. Myös Tampereen teknillisessä yliopistossa ja nykyisessä Tampereen yliopistossa opetetaan ilmiöön liittyviä käytäntöjä.

On kuitenkin epäselvää, tehdäänkö niin kuin opetetaan eli hyödynnetäänkö itsekin DevOpsiin liittyviä käytäntöjä. Tämän diplomityön tutkimustavoitteena onkin selvittää, hyödynnetäänkö opetusjärjestelmien kehittämisessä ja ylläpidossa näitä käytäntöjä, ja miten niiden hyödyntämistä voisi kehittää.

Koska DevOpsiin liittyy käytäntöjä, joita on mahdollista suorittaa erilaisilla työkaluilla, on tarpeen selvittää sellaiset työkalut, joita yliopistolla on jo tarjota käytettäväksi. Koska ohjelmistojen suoritusympäristöjen esittäminen ohjelmakoodina liittyy DevOpsiin, on myös saatavilla olevien suoritusympäristöjen selvittäminen oleellista. Tällä tavalla voidaan saada tietoa, onko yliopistossa edellytykset DevOpsin hyödyntämiselle.

Opetusjärjestelmien DevOps-käytäntöjen hyödyntämistä on selvitetty niihin perustuvalla tapaustutkimuksella. Työkalut ja suoritusympäristöt on selvitetty sähköpostiviestein ja yliopiston intrasivustoa tutkimalla.

Tutkimuksen tulokset osoittavat, että yliopistolla on tarjota työkaluja DevOps-käytäntöjen hyödyntämiseen. Suoritusympäristöjen dokumentaation osalta on kuitenkin kehitettävää. Opetusjärjestelmistä Repolainen noudattaa parhaiten DevOpsiin liittyviä käytäntöjä, mutta kaikissa tutkituissa opetusjärjestelmissä on parannettavaa sovelluksen monitoroinnin, suunnittelun ja IT-infrastruktuurin ohjelmakoodina esittämisen suhteen.

Repolaisen osalta toimenpiteet työssä esitettäviin kehitysideoihin perustuen on aloitettu, mutta Tietotekniikan yksikölle suositellaan myös muiden ideoiden kokeilemista soveltuvuusselvityksien muodossa. Myös ideoiden jatkokehitys ryhmissä ja aivoriihien järjestäminen voivat olla hyödyllisiä.

ABSTRACT

SAMULI KOHOMÄKI: DevOps practices in development and maintenance of educational systems
Tampere University
Master of Science (Technology), 65 pages, 8 Appendix pages
February 2019
Information Technology, MSc
Major: Software Engineering
Examiners: University Lecturer Terhi Kilamo and Associate Professor (tenure track) Kari Systä

Keywords: DevOps, educational systems, educational use

DevOps has gained popularity during the last five years based on Google Trends. Interest in the science community has also increased. For example searches performed at the IEEE Xplore Digital Library indicate that the IEEE published double the amount of research papers on DevOps in 2017 compared to 2015. However the term DevOps is still ambiguous.

DevOps has also been teached. For example Saga University in Japan has utilized an education system which took advantage on tools related to DevOps and Cadi Ayyad University in Morocco has organized a course about DevOps. Tampere University of Technology which is nowadays Tampere University has also teached DevOps practices.

It is though unclear whether we follow the guidelines we teach in other words, whether we utilize DevOps practices ourselves. The aim of this research is therefore to find out do the development and maintenance processes of IT systems used in teaching utilize DevOps practices and could that utilizing be developed in some way.

Because certain practices are related to DevOps and they can be implemented with various technologies it is worthwhile to discover if the university offers some of these technologies. Since infrastructure as code is a part of DevOps figuring out the available software execution environments is also essential. By investigating these aspects it is possible to discover whether the university provides prerequisites for utilizing DevOps practices.

Study on the utilization of DevOps practices in the development and maintenance of educational systems is performed by a case study. Technologies and execution environments available are discovered by email messages and searching them from the intranet website of the university.

The results of the study indicate that the university does offer tools to utilize DevOps practices but the documentation on execution environments is lacking. Of the educational systems Repolainen is best at utilizing DevOps practices but all educational systems studied in this thesis have rough edges on continuous monitoring and planning and sharing the IT infrastructure as code.

Actions based on the development ideas presented in this thesis considering Repolainen have been started but proof of concept implementations based on the other ideas as well are recommended for the Computing unit. Getting together to develop the ideas further and arranging brainstorms could also be beneficial.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston Tieto- ja sähkötekniikan tiedekunnan Tietotekniikan laboratoriossa syys-joulukuussa 2018 ja Tampereen yliopiston Informaatioteknologian ja viestinnän tiedekunnan Tietotekniikan yksikössä tammi-helmikuussa 2019. Työtä ovat tukeneet OKM:n hanke Älykkäät oppimisympäristöt ja niiden sisällöntuotanto sekä yliopiston sisäinen kehittämishanke Plussaa oppimiseen.

Haluan kiittää työni tarkastajaa Terhi Kilamoa hänen antamistaan kommenteista, Terhiä sekä työn toista tarkastajaa Kari Systää työn tarkastamisesta ja kaikkia niitä ihmisiä, joita lähestyin työhön liittyneen tutkimuksen merkeissä - olitte avoimia ja avuliaita. Kiitoksia mukavalle E1- ja F1-käytävien väelle sekä eritoten F105-huoneen kollegoille vertaistuesta. Kiitos myös naisystävälleni Tiialle tuesta koko diplomityön teon aikana.

Tampereella, 6.2.2019

Samuli Kohomäki

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

A+	Web-perustainen opetusjärjestelmä
ALECSS	DevOpsia hyödyntävä opetusjärjestelmä
Apache	HTTP-palvelinohjelma
C	Yleiskäyttöinen, käännettävä imperatiivinen ohjelmointikieli
C++	Yleiskäyttöinen, käännettävä olio-ohjelmointikieli
Caddy	HTTP-palvelinsovellus, joka ottaa HTTPS:n automaattisesti käyttöön verkkosivustolla
CI	Continuous integration, jatkuva integraatio
CiBoT	Moodlen automaattinen ohjelmakoodin tarkastin
CGI	Common Gateway Interface, protokolla sovellusten suorittamiseen web-palvelimilla
commit	Komento, jolla muutokset voidaan tallentaa Git-versionhallinnan tietovarastoon
Devopsfile	Tiedostomuoto DevOpSlangin määrittämiseen
DevOpSlang	Sovellusaluekieli DevOpsin toteuttamiseen
Django	Web-ohjelmointikehys Python-ohjelmointikielelle
DNS	Domain Name System, nimipalvelujärjestelmä
Docker	Työkalu ohjelmistojen virtualisointiin konteiksi käyttöjärjestelmätasolla
Docker Compose	Työkalu Docker-konttien hallitsemiseen ja suorittamiseen
Docker Hub	Tietovarasto Docker-levykuvien tallentamiseen
Dockerfile	Docker-levykuvan ja sen perusteella luotavan kontin määritelmä, sinikopio
Elasticsearch	Hakukone
Elastic Stack	Kokonaisuus, joka koostuu Elasticsearchista, Logstashista ja Kibanasta
EU	Euroopan unioni
Fabric	Työkalu komentorivikomentojen suorittamiseen SSH-yhteyden yli Python-ohjelmointikielellä
geckodriver	Työkalu Geckoon (selainmoottori) perustuvien verkkoselainten, esimerkiksi Firefoxin, kanssa kommunikointiin ohjelmallisesti
Git	Hajautetun versionhallinnan työkalu
GitHub	Git-perustainen etätietovarasto webissä
GitLab	Monipuolinen Git-etätietovarasto
GitLab CI/CD	GitLabin osa, jolla voi toteuttaa jatkuvan integraation, toimituksen ja käyttöönoton
GDPR	General Data Protection Regulation, EU:n yleinen tietosuoja-asetus
Google Groups	Sähköpostilista
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
HTTPS	Hypertext Transfer Protocol Secure, HTTP ja TLS-protokollan yhdistelmä
IBM	International Business Machines Corporation, informaatioteknologiayritys
IEEE	Institute of Electrical and Electronics Engineers, kansainvälinen tekniikan alan järjestö
IPv4	Internet-protokollan neljäs versio
IPv6	Internet-protokollan kuudes versio

IT	Informaatioteknologia
Jenkins	Jatkuvan integraation työkalu
Jenkinsfile	Tiedostomuoto ohjelmiston toimitusputken määrittämiseen Jenkinsillä
Jenkins Pipeline	Jenkinsillä suoritettava toimitusputki
Kanban	Projektinhallintamenetelmä, joka painottaa työnkulun visualisointia, työn alla olevien tehtävien määrän rajoittamista sekä syklien keston mittaamista
Kibana	Työkalu Elasticsearch-datan visualisointiin
localhost	Isäntänimi, jolla voidaan yhdistää isäntäkoneesta kyseiselle (samalle) isäntäkoneelle
Logstash	Työkalu sovelluslokien keräämiseen ja käsittelyyn
Mattermost	Pikaviestin
Microsoft Windows	Microsoftin käyttöjärjestelmä
Moodle	Web-perustainen opetusjärjestelmä
MySQL	Relaatiotietokantajärjestelmä
OpenStack	Alusta pilvilaskentaan
Perl	Yleiskäyttöinen tulkattava ohjelmointikieli
PHP	Tulkattava ohjelmointikieli skriptaukseen web-palvelimilla
pip	Sovellus Python-moduulien hallintaan
PostgreSQL	Relaatiotietokantajärjestelmä
Prospector	Työkalu Python-ohjelmointikielellä tehdyn ohjelmakoodin staattiseen analyysiin
Puppet	Suoritusympäristöjen provisiointi- ja hallintatyökalu
Python	Yleiskäyttöinen, tulkattava ohjelmointikieli
Redhat Enterprise Linux	Linux-käyttöjärjestelmä
Redis	Tietokoneen keskusmuistiin tallennettu avain-arvo-tietokanta
Repolainen	Web-perustainen GitLabin kanssa integroituva opetusjärjestelmä
Robot Framework	Työkalu yleiseen hyväksyntätestaukseen
Selenium	Työkalu web-sovellusten testaamiseen verkkoselaimissa
Sentry	Monitorointityökalu sovellusten virheiden seuraamiseen
Shibboleth	Kertakirjautumisjärjestelmä
SSD	Solid-state drive, tallennusväline
SSH	Secure Shell, salatun verkkoliikenteen protokolla
SLA	Service Level Agreement, sopimus palvelun tasosta
Slack	Pikaviestin
SQLite	Relaatiotietokantajärjestelmä
SonarQube	Työkalu ohjelmakoodin staattiseen analysointiin
TeamCity	JetBrainsin jatkuvan integraation työkalu
TLS	Transport Layer Security, salausprotokolla
TOSCA	Topology and Orchestration Specification for Cloud Applications, OASIS-konsortion avoin standardi
Triton DataCenter	Kokonaisvaltainen ratkaisu pilvilaskennan hallintaan, esimerkiksi virtuaalikoneiden virtualisointiin
Travis	Jatkuvan integraation työkalu
Trello	Pilvipalvelu Kanban-menetelmää käyttävän projektin hallintaan
Ubuntu	Linux-käyttöjärjestelmä
Vagrant	Suoritusympäristöjen virtualisointityökalu

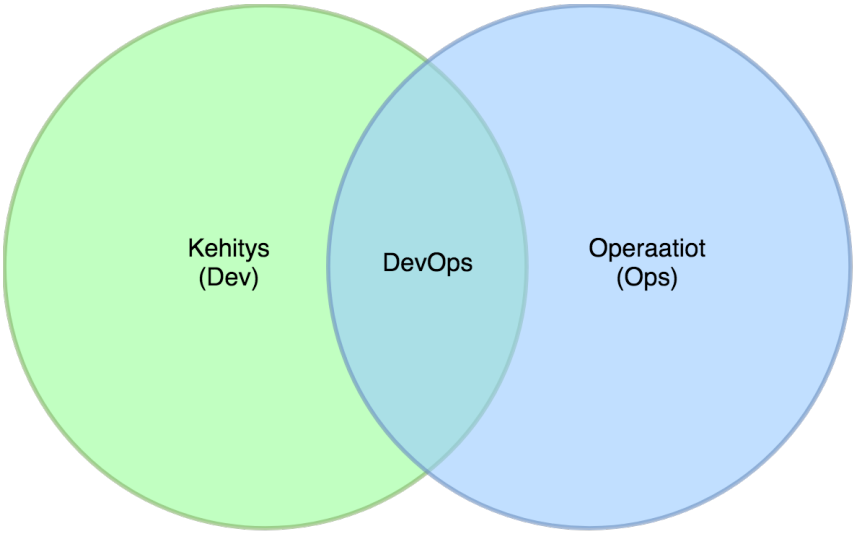
Venn-diagrammi	Matemaatikko John Vennin mukaan nimetty joukko-opin diagrammi
venv	Python-moduuli virtuaalisten Python-suoritusympäristöjen luomiseen
virtualenv	Työkalu eristettyjen Python-ympäristöjen luomiseen
VMware	Yritys, joka valmistaa ohjelmistoja esimerkiksi virtualisointiin
Webropol	Verkkokyselypalvelu
Wordpress	PHP:lla ohjelmoitu sisällönhallintajärjestelmä
XML	Extensible Markup Language, merkintäkieli
Zabbix	Kokonaisvaltainen monitorointityökalu
YAML	YAML Ain't Markup Language, standardi tiedon sarjana esittämiseen

1. JOHDANTO

DevOpsin suosio on viimeisten vuosien aikana kasvanut selvästi. Vuonna 2009 alkunsa saaneen ilmiön [27] verkkohakujen määrä on Googlen hakukoneen osalta lisääntynyt varssin tasaisesti viimeisen noin viiden vuoden ajan [26]. Tällä hetkellä kyseisellä hakusanalla tehtyjä Google-hakuja on enemmän kuin koskaan aiemmin [26].

Myös tieteellinen kiinnostus DevOpsia kohtaan on kasvanut viimeisten kolmen vuoden ajan. Esimerkiksi *IEEE*:n (Institute of Electrical and Electronics Engineers, kansainvälinen tekniikan alan järjestö) digitaalisessa Xplore-kirjastossa suoritettujen hakujen perusteella vuonna 2017 aiheeseen liittyviä artikkeleita julkaistiin 77 kappaletta [53], kun vastaavat luvut vuonna 2016 ja 2015 olivat 64 [54] ja 34 [55] kappaletta.

Suosiostaan huolimatta DevOps on vieläkin moniselitteinen käsite. Sille ei ole muodostunut tarkkaa määritelmää [61]. Sana itsessään muodostuu englanninkielisistä sanoista ”Development”, kehittäminen, ja ”Operations”, operaatiot tai operatiivinen toiminta [19]. Sen tavoitteena onkin ollut poistaa esteet kehityksen ja operaatiivisen toiminnan väliltä [25], sillä näiden on nähty sijaitsevan erillisissä yksiköissään IT-organisaatioissa [52]. DevOpsia havainnollistaa kuvan 1.1 Venn-diagrammi. Suomenkielistä käännöstä termille ei ole, joten jatkossa tässä diplomityössä käytetään sen englanninkielistä versiota.



Kuva 1.1. *Venn-diagrammi DevOpsista.*

Kehittävän ja operoivan henkilöstön sijoittumisen erillisiin yksiköihin on havaittu aiheuttavan ongelmia [52]. Koska Tampereen teknillisellä yliopistolla Tieto- ja sähkötekniikan tiedekunnan Tietotekniikan laboratiossa eli nykyisen Tampereen yliopiston Informaatioteknologian ja viestinnän tiedekunnan Tietotekniikan yksikössä suoritetaan ohjelmistoke-

hitystä opetusta varten, ja koska IT-palveluiden ylläpidosta vastaava taho ei ole sijoittunut samaan yksikköön Tietotekniikan yksikön kanssa, on mahdollista, että edellä mainittu jakautuminen aiheuttaa ongelmia myös kyseisessä organisaatiossa.

Yliopisto-organisaatiossa henkiöstön vaihtuvuus voi myös olla suurehkoa: esimerkiksi diplomitutkintoaan suorittavat tutkimusapulaiset eivät välttämättä jatka yliopiston palveluksessa tutkinnon suoritettuaan. On mahdollista, että tutkimusapulainen on esimerkiksi diplomityönään kehittänyt tietyn järjestelmän, joka on jäänyt yliopiston käyttöön. Diplomityö tehdään yleensä yksin, jolloin kyseisellä tutkimusapulaisella on voinut olla paljon järjestelmään liittyvää tietoa, joka on kuitenkin saattanut jäädä dokumentoimatta, jos diplomityön fokus ei ole ollut siinä.

Itse tutkittava ilmiö saattaa myös asettaa esteitä eri roolien eriyttämiseen. Esimerkiksi Bayser et al. huomauttavat, että tutkimus saattaa vaatia alan täsmällistä tietämystä, jolloin tutkijan ja ohjelmoijan roolien eriyttäminen voi olla hankalaa [19]. Täten voidaan myös ajatella, että jos ohjelmiston ylläpitäjäksi valikoituu jokin muu taho kuin sen kehittäjä, saattaa tämän olla vaikeaa suorittaa tehtäväänsä, jos ohjelmiston dokumentaatiossa on oletettu sitä tulkitsevan henkilön ymmärtävän järjestelmään liittyvä tausta.

Myös muutokset yliopisto-organisaatiossa saattavat vaikeuttaa siellä suoritettavaa opetusjärjestelmiin liittyvää ohjelmistokehitystä. Jotta molemmat jatkuisivat sulavasti myös muutosten jälkeen, on riittävä dokumentaatio tärkeää. Tampereen teknillisellä yliopistolla tapahtui 2010-luvulla niin tiedekuntamuutos kuin laitosten uudelleennimeäminenkin: Tietotekniikan ja Sähkötekniikan tiedekunnat yhdistyivät vuonna 2013 osana yliopiston rakennemuutosta [71], ja Tietotekniikan laitos muuttui Tietotekniikan laboratorioksi vuoden 2017 alussa [17]. Vielä suurempi muutos on kuitenkin tapahtunut vasta hiljattain: Tampereen yliopisto ja Tampereen teknillinen yliopisto yhdistyivät 1.1.2019 Tampereen yliopistoksi, ja tämä muodostaa Tampereen ammattikorkeakoulun kanssa Tampereen korkeakouluyhteisön [35]. Edellä mainitut näkökulmat huomioon ottaen opetusjärjestelmien kehityksen ja ylläpidon kestävyyden ja jatkuvuuden kannalta olisi tärkeää, että hyödynnettäisiin niitä tukevia käytäntöjä.

DevOps voisi mahdollisesti olla yksi tällainen menetelmä. Tämän diplomityön tutkimuskysymyksenä onkin selvittää, miten DevOpsiin liittyviä käytäntöjä on tällä hetkellä hyödynnetty opetusjärjestelmien kehityksessä ja ylläpidossa, ja miten niiden hyödyntämistä voisi kehittää. Työ rajautuu kyseisten järjestelmin ohjelmistokehitykseen ja -ylläpitoon, joten esimerkiksi DevOpsin opettaminen on tämän rajauksen ulkopuolella.

Koska DevOps kuitenkin vaikuttaa kasvattavan suosiotaan koko ajan, ja Suomessakin on tällä hetkellä työelämän sosiaalisen median palvelun LinkedInin perusteella lähemmäs kaksi sataa avointa DevOpsiin liittyvää työpaikkailmoitusta [2], saattaisi DevOpsille olla myös opetuksellista kysyntää. Japanin Saga University onkin hyödyntänyt DevOps-työkaluja opetuksessa [78] ja Moroccon Cadi Ayyad University on järjestänyt kurssin, jossa on käsitelty DevOpsia [11]. Myös Tampereen teknillisellä yliopistolla on opetettu

DevOps-käytäntöjä, esimerkiksi jatkuvaa integraatiota.

Työn tekijä uskoo kuitenkin ”syö oma koiranruokasi” [30] -ajatukseen, joka tässä kontekstissa tarkottaisi, että DevOpsin opetuksen laatu voisi parantua, jos sitä hyödynnettäisiin myös itse. Täten saatettaisiin ainakin välttyä huonon esimerkin näyttämisestä opiskelijoille.

Aiempaa tutkimusta DevOpsin hyödyntämiseen opetusjärjestelmien kehittämisessä ja ylläpitämisessä ei tämän diplomityön puitteissa löytynyt, mutta tieteellisestä käytöstä on julkaistu artikkeleita. Esimerkiksi Bayser et. al ovat hyödyntäneet DevOps-käytäntöjä tutkimuksessaan [19], samoin Edinburghin yliopiston astronominen observatorio [75]. Vastaavasti esimerkiksi Wettinger et al. ovat julkaisseet tutkimuksessaan sovellusaluekielen ja tiedostoformaatin, jolla ohjelmistoprojektin DevOps-toteutus voitaisiin määritellä [104]. Tämä voidaan nähdä myös DevOpsin hyödyntämisen kehittämisenä. Tähän työhön liittyvää tutkimusta esittelee enemmän luku 3.3.

Työn tutkimuskysymykseen vastaamiseksi hyödynnetään seuraavaa tutkimusmenetelmää: Tampereen teknillisellä yliopistolla ja Tieto- ja sähkötekniikan tiedekunnan Tietotekniikan laboratoriolla käytössä ja saatavilla olevat ohjelmistokehityksen työkalut sekä ohjelmistojen suoritusympäristöt selvitetään sähköpostilla. Asennetut työkalut saattavat olla jo tuttuja opetushenkilöstölle, ja saatavilla olevien suoritusympäristöjen tekniset tiedot saattavat mahdollistaa tulevien järjestelmien tuotantoympäristöjen kahdentamisen aikaisemmin kehitys- ja testausympäristöissä.

Vastaavasti opetusjärjestelmiin tutustutaan laadullisen tapaustutkimuksen avulla. Tähän sisältyy kaksi vaihetta: omatoiminen tutustuminen ja kysely. Omatoimisessa tutustumisessa voidaan versionhallintaprojektien ja järjestelmien lähdekoodin perusteella havaita, mitä DevOps-käytäntöjä niissä on hyödynnetty. Kyselyllä voidaan saada lisätietoa tästä, tukea työn tekijän tekemiin havaintoihin sekä tietoa siitä, kuinka hyvin esimerkiksi järjestelmien kehitys- ja ylläpitotiimit kommunikoivat keskenään. Tutkimusmenetelmää avaa enemmän luku 3.

Tässä työssä tehdyn tutkimuksen perusteella on havaittavissa, että tutkitut opetusjärjestelmät hyödyntävät DevOps-käytäntöjä joiltain osin. Hyödyntämisen tasossa on kuitenkin myös kehitettävää. Myös DevOps-käytäntöjen hyödyntämisen mahdollistavia työkaluja on saatavilla. Tutkimuksen havainnot ovat luettavissa luvuista 4 ja 5 ja kehitysideat luvusta 6.

Tämä diplomityö jatkuu seuraavasti: luku 2 avaa DevOpsin käsitettä, siihen liittyviä kulttuurellisia näkökulmia, käytäntöjä, sillä saavutettavia mahdollisia hyötyjä sekä sen mahdollisia ongelmia ja käyttöönoton esteitä. Luvussa 3 avataan enemmän työn tutkimusmenetelmää ja esitellään sen aiheeseen liittyvää tutkimusta. Luku 4 kertoo Tampereen teknillisellä yliopistolla saatavilla olevat työkalut ja ohjelmistojen suoritusympäristöt, ja luku 5 esittää, miten DevOps-käytäntöjä ja -työkaluja on hyödynnetty Tampereen teknillisen

yliopiston Tieto- ja sähkötekniikan tiedekunnan Tietotekniikan laboratorion opetusjärjestelmissä. Diplomityön lopuksi luku 6 esittää kehitysideoita, ja luku 7 tekee yhteenvedon työn havainnoista ja tuloksista.

Tämän työn osalta huomioitavaa on se, että siihen liittyvä tutkimus on aloitettu silloin, kun Tampereen teknillinen yliopisto oli vielä oma yliopistonsa. Täten tässä työssä viitataan kyseisen yliopiston Tieto- ja sähkötekniikan Tietotekniikan laboratorioon. Tämän laboratorion tutkimus ja opetus kuitenkin jatkavat osana aiemmin mainitun uuden yliopiston Tietotekniikan yksikköä.

2. DEVOPS

DevOps on käsitteenä verrattain nuori. Vuonna 2009 Belgian Ghentissa järjestettiin ensimmäinen ”Devopsdays”-tapahtuma [27], ”DevOps-päivät”, joita on sittemmin pidetty ympäri maailmaa useamman kerran vuodessa [9]. Devopsdays on maailmanlaajuinen tekninen konferenssi, jossa käsitellään ohjelmistokehitystä, informaatioteknologian operatiivista toimintaa sekä näiden leikkausta. Termi sai pohjustusta jo hieman aiemmin Agile 2008 -konferenssissa. Siellä Devopsdays-konferenssin perustaja Patrick Debois [9] piti esityksen aiheenaan ketterät menetelmät IT-infrastruktuureissa ja operatiivisessa toiminnassa [24]. Hänen mukaansa kehittäminen ja IT-infrastruktuuri on nähtävä yhtenäisenä kokonaisuutena eikä erillisinä osina.

DevOps on lyhenne sanoista ”Development”, kehittäminen, ja ”Operations”, operatiivinen toiminta [19]. Vaikka termi koostuu selkeästi kahdesta sanasta, ei sille ole kuitenkaan vakiintunut tarkkaa määritelmää. Tutkijat ja asiantuntijat ovat antaneet DevOpsille toisistaan poikkeavia kuvauksia. Esimerkiksi Jabbari et al. toteavat, ettei sen määritelmä ole vakiintunut [61]. Vastaavasti Erich et al. ovat suositelleet taksonomian luomista käsitteestä [33].

Tämän diplomityön tavoitteena ei ole muodostaa yhtä DevOpsin määritelmää vaan sitä on tarkoitus tutkia eri näkökulmista. Seuraavassa alaluvussa 2.1 käsitelläänkin erilaisia määritelmiä, joita siitä on tieteellisissä artikkeleissa ja kirjallisuudessa esitetty. 2.2 esittelee DevOpsiin liittyviä kulttuurellisia näkökulmia, 2.3 siihen liittyviä käytäntöjä, ja 2.4 sekä 2.5 kertovat luvun lopuksi sillä saavutettavia hyötyjä sekä sen ongelmia ja mahdollisia esteitä sen hyödyntämiselle.

2.1 Määritelmiä

DevOps on käsitteenä monitulkintainen. Esimerkiksi Smeds et al. [95] määrittävät vuonna 2015 tekemässään kirjallisuuskatsauksessaan DevOpsin joukoksi tuotannollisen prosessin kykyjä tai potentiaaleja, joita tukevat kulttuurilliset ja tekniset mahdollistajat. Mahdollistajien tarkoitus on tehdä potentiaalien määrittelemien prosessien toteuttamisesta sulavaa, joustavaa ja tehokasta.

Kulttuurillisiin mahdollistajiin lukeutuu esimekiksi kokeilukulttuuri: yksilöille on annettava mahdollisuus kokeilla, jotta he voivat oppia onnistumisista ja hyväksyä myös epäonnistumiset tilaisuuksina oppia. Tekniset mahdollistajat vastaavasti koostuvat työkaluista, joissa painotetaan ohjelmistokehityksen eri työvaiheiden automatisointia. Smeds et al. huomauttavat myös, että vaikka keskittyminen on pääasiassa potentiaaleissa, on DevOpsin tehokkaan käytön kannalta olennaista, että mahdollistajat tukevat niitä. [95]

Myös Humblen ja Moleskyn määritelmässä automaatiolla on roolinsa. He esittävät DevOpsin neljän osa-alueen kokonaisuutena: kulttuuri, automaatio, mittaaminen ja jakaminen [50]. Kulttuurillista näkökulmaa ja jakamista käsitellään myöhemmin alaluvussa 2.2, ja automaatio liittyy alaluvussa 2.3 käsiteltäviin käytäntöihin. Mittaamisella Humble ja Molesky tarkoittavat korkean tason liiketoiminnan mittarien, esimerkiksi liikevaihdon, seuraamista [50].

Jabbari et al. [61] puolestaan määrittävät vuonna 2016 tekemässään systemaattisessa kartoitustutkimuksessaan DevOpsin siten, että sen tarkoitus on kaventaa kuilua kehittävän ja operatiivisen tiimin välillä. DevOps painottaa heidän kuvauksessaan kommunikaatiota ja yhteistyötä, jatkuvaa integraatiota, laadunvalvontaa sekä automatisoitua toimitusta ja käyttöönottoa hyödyntäen ohjelmistokehitykseen liittyviä erilaisia käytäntöjä. Näihin käytäntöihin he listaavat esimerkiksi jatkuvan suunnittelun, jatkuvan monitoroinnin, jatkuvan integraation, jatkuvan ja automaattisen käyttöönoton, jatkuvan toimituksen, jatkuvan ja automaattisen testauksen sekä infrastruktuurin ohjelmakoodina, jotka on kerätty listaukseksi useammasta lähteestä. Näitä käytäntöjä käsitellään syvällisemmin alaluvussa 2.3.

Jabbari et al. esittävät myös lähdemateriaalistaan tekemiään huomioita DevOpsin suhteesta ketteriin menetelmiin [61]. Esimerkiksi Kilamo et al. mainitsevat, ettei DevOps palvele kaikilta osin ketteriä menetelmiä vaan, että se on enemmän työkaluja ja prosesseja kuin sosiaalista kanssakäymistä [68]. Ketterän ohjelmistokehityksen julistus arvostaa yksilöitä ja sosiaalista kanssakäymistä enemmän kuin prosesseja ja työkaluja [73].

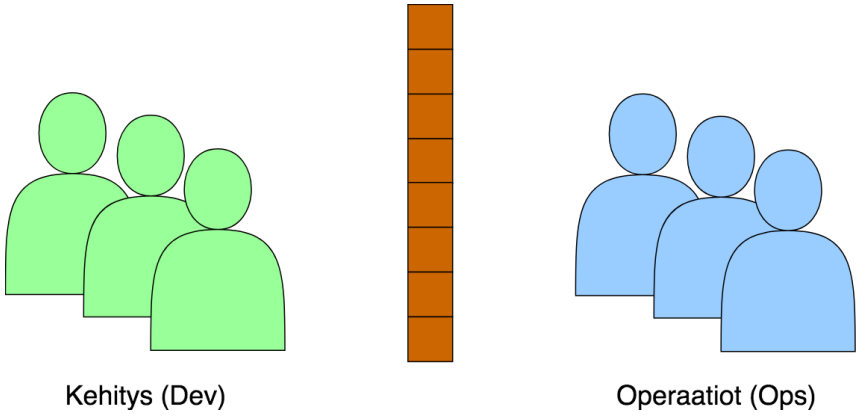
Toisaalta B.S. ja D.L. Farroha [37] kirjoittavat, että DevOps laajentaa ketteriä menetelmiä tuomalla operatiivisen IT-toiminnan lähemmäs näiden periaatteita. Heidän mukaansa painotus on enemmän tiimiytymisessä ja kommunikaatiossa kuin työkaluissa ja prosesseissa. Myös Virmani toteaa DevOpsin laajentavan ketterien menetelmien käytön koko ohjelmistotoimituksen alueelle [101]. Debois kommentoi, että DevOps sai alkunsa liikkeestä, jonka tavoitteena oli poistaa esteet kehityksen ja operaatioiden väliltä [25].

Vastaavasti Erich et al. [33] määrittävät vuonna 2014 tekemässään kartoitustutkimuksessaan DevOpsin konseptuaaliseksi viitekehykseksi, jonka tavoitteena on uudelleenintegroida kehitys ja operaatiivinen toiminta. Samalla he kuitenkin huomauttavat, että DevOps on nähty myös työnimikkeenä tai kokoelmana erilaisia osaamisalueita.

Vaikka työnimikemääritelmän perusteella voisi olla mahdollista ymmärtää, että DevOps on samalla tavalla toteutettavissa kuin esimerkiksi ohjelmistoprojekti, on toisenlaisiakin kannanottoja tehty. Esimerkiksi McCarthy et al. kommentoivat, ettei DevOpsia voi ostaa tai toteuttaa eikä sitä voi väkisin pakottaa organisaation käyttöön [74]. DevOpsia ei myöskään voi Smeds et al. [95] mukaan yleistää vaan sen käyttöönotto on orgaanisaatiokohtainen. Käyttöönotossa havaituista haasteista voivat kuitenkin heidän mukaansa oppia myös muut muuttaessaan organisaatiotaan sen suuntaan.

2.2 Kulttuuri

IT-organisaatioiden kehittävän ja operatiivisen toiminnan on kuvattu olevan omia yksiköitään, joiden välillä vallitsee erilainen kulttuuri: kehittäjät luovat uutta ja operoiva henkilöstö valvoo vanhaa [52]. Tämä kuilu saattaa vaikuttaa negatiivisesti näiden kahden yksikön väliseen toimintaan [52], ja esimerkiksi Debois onkin sitä mieltä, että nämä kaksi osaa olisi nähtävä yhtenä kokonaisuutena [24]. Kuva 2.1 esittää IT-organisaatiota, jossa kehittävä ja operoiva tiimi ovat erillisinä yksiköinään.



Kuva 2.1. *Perinteinen IT-organisaatio, jossa kehitys ja operaatiot ovat erillisinä yksiköinään.*

Walls näkee DevOps-kulttuurin neljän näkökulman kokonaisuutena: avoin kommunikaatio, kannustimien ja vastuiden tasaaminen, kunnioitus ja luottamus. DevOpsia hyödyntävän tiimin jäsenet keskustelevat enemmän, ovat valmiita päivystämään töissä ongelmien varalta, omaavat yhteiset päämäärät, kunnioittavat toisiaan ja luottavat toisiinsa. [102]

DevOpsiin liittyvää kulttuurillista muutosta ei voi kuitenkaan pakottaa organisaation käyttöön [91]. Ei siis voida ajatella, että organisaatio voisi tehdä DevOpsiin liittyvän kulttuurisen muutoksen nappia painamalla. On kuitenkin keinoja, joilla kulttuurellista siirtymää voidaan edistää.

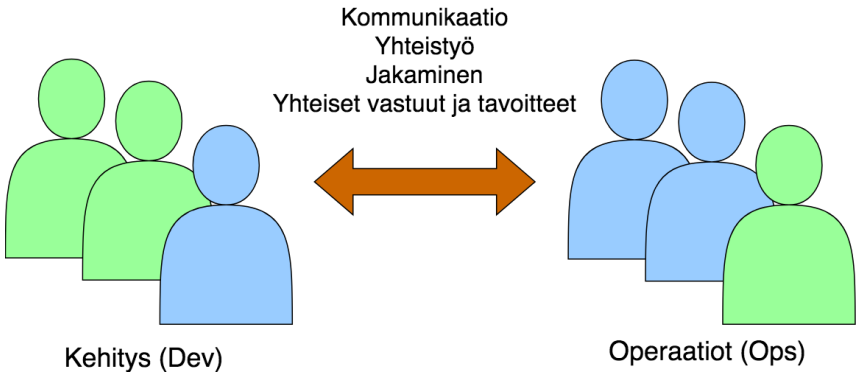
Yhteistyö kehittävän ja operatiivisen tiimin välillä on tärkeää. Tätä voidaan parantaa esimerkiksi siten, että operatiivinen yksikkö on mukana ohjelmiston suunnittelussa ja sen siirtymävaiheessa käyttöönottoon. Operaatiivisen henkilöstön pitäisi osallistua ohjelmistoprojektiin liittyviin selvityksiin, retrospektiiveihin, suunnittelutapaamisiin ja projektitiimien esittelytilaisuuksiin. Samaan pitäisi pyrkiä myös toisinpäin: kehittäjiä pitäisi kierrättää operatiivisessa tiimissä, ja projektitiimien edustajien pitäisi osallistua säännöllisesti operatiivisen yksikön tapaamisiin. Kehittäjien pitäisi myös olla tavoitettavissa auttamaan ongelmien ratkaisemisessa niiden ilmetessä. [50]

Myös jakaminen voidaan ajatella kulttuurisena näkökulmana kehittävän ja operatiivisen tiimin kuilun kaventamisessa. Jakamista voi tapahtua monella tasolla [50]. Esimerkiksi

tiivistynyt yhteistyö lisää tietämyksen ja kokemuksen jakamista [91]. Jakaminen voi olla kehitystyökalujen ja -teknologioiden jakamista, mutta myös toisenlaisia aktiviteetteja, esimerkiksi onnistuneiden julkaisujen yhteistä juhlimista [50].

Organisaation pitäisi lisäksi pyrkiä rohkaisemaan työntekijöitään kokeilemaan, sillä sekä onnistumisista että virheistä voi oppia. Muiden syyttäminen virheistä, kunnioituksen puute kollegoita kohtaan ja vain omaan työpanokseen keskittyminen vaikuttavat negatiivisesti DevOps-henkiseen kulttuuriin. [95]

Organisaatio voi myös Feitelson et al. [38] mukan tukea työntekijöitään olemaan henkilökohtaisesti vastuussa tekemästään ohjelmakoodista. He kertovat, että esimerkiksi Facebook on tehnyt näin erillisen laadunvalvontatiimin puuttuessa hyödyntäessään jatkuvaa käyttöönottoa. Heidän mukaansa vastuu oman ohjelmakoodin laadusta myös tukee ohjelmistokehittäjää tekemään ohjelmakoodista operoitavuusystävällistä, ja on yksi yhteisvastuullisen kulttuurin näkökulmista. Täten se saattaa myös edistää henkilöstön pyrkimyksiä kohti yhteisiä tavotteita ja kannustaa sitä luopumaan Humblen ja Moleskyn mainitsemasta ”seinän yli operaatiiviselle toiminnalle” [50] -toiminnnasta. Kuva 2.2 esittää IT-organisaatiota DevOpsiin liittyvän kulttuurimuutoksen jälkeen.



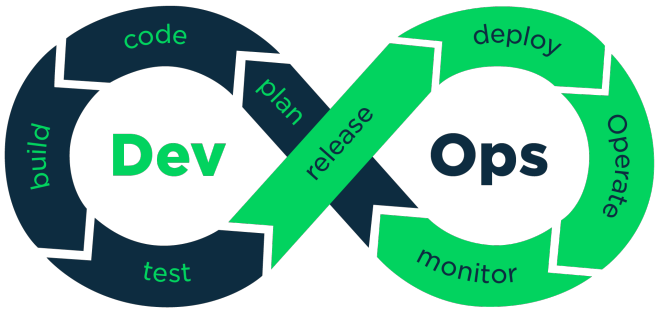
Kuva 2.2. *DevOps-henkinen organisaatio.*

Humble ja Molesky esittävät, että tuotetiimien olisi lisäksi hyvä koostua henkilöistä, joiden osaaminen risteää [50]. Tämä voidaan nähdä yhtenä kulttuurimuutoksen edistäjänä, sillä täten kehittävälläkin tiimillä on mahdollisesti tietämystä ohjelmistojen operoimisesta. Myös kehittävän ja operatiivisen tiimin väliset mahdolliset kommunikaatioesteet pitäisi poistaa ja sallia yhteistyö yhteisten päämäärien saavuttamiseksi.

Liittyen edellä mainittuihin osaamisalueiltaan risteäviin tiimeihin Balalaie et al. [18] esittävät, että DevOps suosittelee projektin jäsenten vertikaalista tiimeihin jakoa horisontaalisen jaon sijaan. Tällä tarkoitetaan sitä, että on suositeltavampaa luoda useampi pieni tiimi, jolla on osaamista esimerkiksi sekä kehittämisestä että operoinnista, sen sijaan, että koottaisiin erilliset tiimit kehitykseen ja operointiin. Tämä on nähty myös ”NoOps” eli ”No Operations” -bisnesmallina, jossa IT-järjestelmä on täysin automatisoitu ja itse itseään hallinnoiva, ja jonka ongelmat korjataan välittömästi ilman, että ihmisen pitää puuttua niihin [109].

2.3 Käytäntöjä

DevOpsiin liittyy erilaisia käytäntöjä ja prosesseja. Näitä ovat esimerkiksi jatkuva suunnittelu, jatkuva integraatio, jatkuva testaus, jatkuva toimitus, jatkuva käyttöönotto ja jatkuva monitorointi [61][101] sekä infrastruktuuri ohjelmakoodina [61]. DevOpsiin liittyvien käytäntöjen yhteydessä toistuvat sanat jatkuva ja automaattinen. DevOps-suuntautuneessa työskentelyssä painotetaankin eri työvaiheiden automatisointia [95]. Jatkuvuutta DevOpsissa kuvastaa kuva 2.3.



Kuva 2.3. *Jatkuvuus DevOps-henkisessä ohjelmistotoimituksessa [70].*

Virmani [101] esittää, että erinäiset käytännöt yhdistämällä on mahdollista luoda ohjelmistokehityksen toimituksesta putki, joka on mahdollisimman automatisoitu. Tämä putken ideana on, että tietyn ohjelmiston kaikki julkaisut kulkevat sen läpi. Toimitusputki voidaan nähdä jatkuvana, sillä kehitysvaiheeseen palataan uudestaan jatkuvan integraation, jatkuvan toimituksen, jatkuvan testauksen, jatkuvan käyttöönoton, jatkuvan asiakaspalautteen ja jatkuvan suunnittelun jälkeen. Seuraavissa alaluvuissa kuvataan DevOps-käytäntöjä tarkemmin. On kuitenkin syytä huomauttaa, että DevOpsiin saattaa näkemyksestä riippuen liittyä myös muita kuin seuraavaksi esitettäviä käytäntöjä.

2.3.1 Infrastruktuuri ohjelmakoodina

Ohjelmistoprojektin infrastruktuurinen suunnitelma määrittelee, millaisia resursseja kehitettävä ohjelmisto tulee tarvitsemaan [15]. Näitä voivat olla esimerkiksi virtuaalitietokoneiden määrä [15] tai tietokantatyyppi ja -moottori. Infrastruktuurisen suunnitelman perusteella luodaan yleensä monia asennus- ja konfiguraatioskriptejä, joilla alustetaan ja yhdistetään käytettävät fyysiset tai virtuaaliset tietokoneet, asennetaan niihin tarvittavat ohjelmistot, sekä alustetaan ja käynnistetään täydentävät palvelut kehitettävän ohjelmiston suorittamiseksi kyseisessä suoritusympäristössä [15].

Kaiken muun paitsi kehitettävän ohjelmiston ohjelmakoodin katsotaan olevan infrastruktuurista ohjelmakoodia [47]. Tällaisen ohjelmakoodin säilyttäminen ja saatavilla olo tarvittaville tahoille on välttämätöntä. Jos se ei ole hyvin dokumentoitua, voi infrastruktuurin uudelleenpystyttäminen olla hankalaa [47]. DevOps rohkaiseekin käyttämään infrastruktuuriselle ohjelmakoodille samanlaista merkintätapaa kuin itse sovelluksen ohjelmakoo-

dille [15]. Kuten ohjelmakoodin suhteen on myös infrastruktuurisen lähdekoodin osalta tärkeää valita tilanteeseen sopiva ohjelmointikieli tai -työkalu [47].

Jotta infrastruktuuria ei tarvitsisi yrittää luoda manuaalisesti esimerkiksi skriptien avustuksella jokaiseen uuteen suoritusympäristöön kuten ohjelmistokehittäjän omalle tietokoneelle, esittää Hutterman [47], että siihen liittyvät vaiheet voidaan määrittää suoritettavina spesifikaatioina virtualisointia avuksi käyttäen. Hän lisää, että tällaiset spesifikaatiot olisi mahdollista tallentaa versionhallintaan, ja täten ne olisivat helposti saatavilla esimerkiksi uudelle ohjelmistokehittäjälle. Hutterman jatkaa, että jos infrastruktuurin määrityksiä muutettaisiin, jakaisi kehittävä tiimi samalla muutokset myös operatiiviselle tiimille, ja tämä voisi ottaa ne käyttöön muissa suoritusympäristöissä. Jos muutokset aiheuttaisivat ongelmia esimerkiksi tuotantoympäristössä, olisi aiemman, toimivan, määrityksen uudelleenkäyttöönotto helppoa versionhallinan ansiosta.

Infrastruktuuri on mahdollista esittää ohjelmakoodina esimerkiksi virtualisointityökalu *Vagrantin* ja provisiointityökalu *Puppetin* konfigurointitiedostojen avulla. Vagrantilla voidaan luodan esimerkiksi ohjelmiston kehitys- ja testausympäristöt virtuaalitietokoneina, ja Puppet:lla voidaan hallita näiden suoritusympäristöjä, esimerkiksi määrittää niiden sisältämät resurssit. [47]

Muita esimerkkejä infrastruktuurisesta ohjelmakoodista ovat muun muassa käyttöjärjestelmätason virtualisoinnin mahdollistaman *Dockerin Dockerfile*-tiedostot. Niiden avulla on mahdollista esittää tiedostona käskyt, jotka suorittamalla Docker luo levykuvat [1]. Myös muiden käytäntöjen kuten jatkuvan integraation mahdollistamien työkalujen konfigurointitiedostot ovat infrastruktuurista ohjelmakoodia. Esimerkkinä mainittakoon jatkuvan integraation työkaluun *Jenkinsiin* liittyvät *Jenkinsfile*-tiedostot, joilla on mahdollista määrittää *Jenkins Pipeline* [83] eli operaatiot, jotka Jenkinsin on tarkoitus suorittaa aina tietyn työn yhteydessä.

2.3.2 Jatkuva integraatio

Jatkuva integraatio viittaa Virmanin [101] mukaan ajatukseen integroida eli ottaa osaksi vanhaa mahdollisimman usein. Hän avaa ajatusta kirjoittamalla, että uutta ohjelmakoodia ei olisi syytä pitää omassa paikallisessa työympäristössä kovin pitkään vaan se kannattaa jakaa tiimin kesken, ja sen toimivuutta aiemman ohjelmakoodin kanssa kannattaa varmistaa jatkuvasti. Integraation tiheys on tärkeää, jotta ohjelmistokehittäjät saisivat nopeasti palautetta kehittämästään ohjelmakoodista [39]. Virmani suosittelee jatkuvaan integraatioon koko tuotteen tasolla komponenttien lisäksi [101].

Jatkuva integraatio on nähty prosessina, joka koostuu pienemmistä osavaiheista. Näitä vaiheita voivat olla esimerkiksi ohjelmakoodin kääntäminen, yksikkö- ja hyväksyntätestien suorittaminen, testien ohjelmakoodin kattavuuden selvittäminen, ohjelmakoodistandardien ja -tyyliohjeiden noudattaminen sekä sovelluksen käyttöönottopakettien luominen. Tyypillisesti nämä vaiheet on automatisoitu jollakin tavalla. [39]

Jatkuvan integraation aikana esintyneet ongelmat ovat tärkeitä, ja niihin voi liittyä esimerkiksi hyvin visuaalisia artefakteja. Näiden tarkoitus on varmistaa, että ilmenneiden ongelmien ratkaiseminen priorisoidaan korkealle ja osoitetaan taholle, jonka katsotaan olevan siitä vastuussa. [39]

Ståhl ja Bosch tiivistävät jatkuvasta integraatiosta vuonna 2014 tekemässään systemaattisessa kirjallisuuskatsauksessaan monien lähteiden perusteella, että jatkuvan integraation prosessin käynnistävät muutokset ohjelmiston lähdekoodissa [97]. He kuitenkin huomauttavat, että jatkuvan integraation voi asettaa käynnistymään myös muilla tavoilla. Prosessi voidaan aloittaa esimerkiksi tiettynä kellonaikana [48], sekä ohjelmakoodin muutosten myötä että tiettyyn kellonaikaan [29], tai useamman toiminnon sarjana [97], jolloin ensimmäistä vaihetta seuraa toinen ja niin edelleen.

Jatkuvalla integraatiolla on etuja. Näitä ovat esimerkiksi korkeampi ohjelmiston julkaisutiheys ja ennakoitavuus, ohjelmistokehittäjien tuottavuuden parantuminen sekä parantunut kommunikaatio. DevOps luo pohjaa jatkuvalle integraatiolle, sillä se vaatii kehityksen ja operaatioiden välistä yhteistyötä. [39]

Jatkuvan integraation toteuttamiseen on olemassa monia vaihtoehtoja. Esimerkiksi aiemmin mainittua Jenkinsia voi käyttää jatkuvaan integraatioon [62], mutta muitakin vaihtoehtoja on: *GitLabin* osana oleva *GitLab CI/CD* [44], *Travis CI*, joka integroituu *GitHub*-versionhallintasivuston kanssa, sekä JetBrainsin *TeamCity*.

2.3.3 Jatkuva testaus

Jatkuvan testauksen yksi mahdollistajista on jatkuva integraatio. Zimmerer huomauttaa, että se on jatkuvan testauksen esiehto [110]. Kuten alaluvussa 2.3.2 esitetään, saattaa jatkuvan integraation yksi osavaihe olla esimerkiksi yksikkötestien suorittaminen. Tällöin jatkuvasti integroitaessa myös testataan jatkuvasti. Testatakseen jatkuvasti pitää testien olla täysin automatisoituja [101]. Virmani jatkaa, että ohjelmaa pitäisi olla mahdollista testata joka kerta ilman käyttäjän interaktiota samalla, kun ohjelmistosta luodaan automaattisesti ajettava versio [101].

Jatkuvan testauksen pyrkimys on integroida testaaminen mahdollisimman lähelle ohjelmistokehitysprosessia [39]. Sen ideana on, että ohjelmistoa testattaisiin koko sen elinkaaren ajan alusta loppuun asti, ja että arvittaessa testejä voidaan myös soveltaa ja suorittaa vain tarpeellinen osa niistä [110].

Jatkuva testaus tuo ohjelmiston käyttöönottokanavaan mukaan automatisoidut testit ohjelmiston toiminnallisuudelle, eheydelle, konsistenssille, suorituskyvylle ja turvallisuudelle sekä myös sen ei-toiminnallisille osille. Testata voidaan myös esimerkiksi infrastruktuurista ohjelmakoodia ja itse ohjelmiston toimitusputkea. Lisäksi testausta on mahdollista suorittaa useammassa ympäristössä, esimerkiksi tuotantoympäristössä. [110]

Jatkuvan testauksen etuna on esimerkiksi parantunut reagointi virheisiin. Koska ongelmaan liittyvä konteksti on mahdollisesti paremmin ohjelmistokehittäjän muistissa, saattaa itse ongelman ratkaiseminen olla nopeampaa, ja myös testien epäonnistumisen juurisyy on mahdollista selvittää ja poistaa [39]. Koska testit ovat automatisoituja [110], ei niitä tarvitse ajaa manuaalisesti, ja täten säästyy aikaa itse ohjelmistokehitykselle.

Ohjelmistokieliin ja -kehyksiin liittyy runsaasti työkaluja, joilla niillä tehtyjä ohjelmistoja voi testata automatisoidusti ja täten jatkuvasti. Vastaavasti yleistä hyväksyntätestausta voi tehdä esimerkiksi *Robot Frameworkilla* [92] ja web-sovellusten testausta selaimessa esimerkiksi *Seleniumilla* [93]. Ohjelmakoodin staattiseen analyysiin voi käyttää esimerkiksi *SonarQubea* [79].

2.3.4 Jatkuva toimitus

Jatkuva toimitus tarkoittaa toimivien ohjelmistoversioiden toimittamista tiettyyn suoritusympäristöön, mutta ei välttämättä niiden välitöntä käyttäjien käytettäväksi tarjoamaista [40]. Tälle analogiana voidaan ajatella esimerkiksi työpöytä- ja mobiilisovellusten toimittaminen: uusia versioita niistä voidaan toimittaa automaattisesti esimerkiksi sovelluskauppaan tai sovelluksen valmistaneen tai sitä jakavan tahon web-sivustolle, mutta niitä ei automaattisesti asenneta tai päivitetä käyttäjän päätelaitteille.

Jatkuva toimitus onkin aiemman perusteella koettu myös kyvykkyytenä tehdä ohjelmiston käyttöönotto silloin, kun halutaan. Tämä saattaa tarkoittaa sitä, että jokainen lisäys ohjelmiston lähdekoodin versionhallintaan siirretään myös suoraan tuotantoympäristöön käyttöön. Käyttöönottojen tiheys ei ole oleellista vaan kyky niiden tekemiseen. [76]

Jatkuvan toimituksen etuna on esimerkiksi pienempi määrä ongelmia tuotantoympäristössä: kun muutosten sarja on pienempi, on mahdollisia ongelmia vähemmän, ja niiden selvittäminen helpompaa. Jatkuva toimitus pitää myös palautesyklin lyhyenä ja mahdollistaa suunnittelun tahdin ylläpitämisen ohjelmiston kehityksessä. [76]

Neelyn ja Stoltin [76] mukaan jatkuva toimitus on kiinnostava konsepti myös liiketoiminnan näkökulmasta. He perustelevat tätä sillä, että verrattuna ajanjaksojen mukaan rajattuihin julkaisuihin, esimerkiksi kahden kuukauden välein, julkaisujen määräajoista myöhästyminen ei haittaa, koska seuraavaa julkaisua ei tarvitse odottaa seuraavaan määräaikaan asti. Tämä on positiivista asiakastyytyväisyyden kannalta, sillä esimerkiksi täyttämättä jääneet lupaukset on mahdollista paikata huomattavasti nopeammin.

Toimituksen tekninen kompleksisuus ei myöskään ehdi kasvaa pienemmissä julkaisuissa isojen julkaisujen tasolle. Liiallinen kompleksisuus aiheuttaa esimerkiksi aikaa vieviä ohjelmakoodin yhdistämisprosesseja sekä datan migraatioita, jotka ovat myös virheherkkiä vaiheita. Jatkuva toimitus saattaa myös vähentää työntekijödien stressiä, sillä tietyin väliajoin julkaistaessa siihen pitää valmistautua hyvin, ja se saattaa myös vaatia kaikkien osanottoa. [76]

Jatkuvaa toimitusta voi tehdä esimerkiksi Jenkinsilla [62], joka mainittiin jo jatkuvan integraation yhteydessä alaluvussa 2.3.2. Myös esimerkiksi GitLab CI/CD soveltuu siihen [45].

2.3.5 Jatkuva käyttöönotto

Jatkuvan käyttöönoton esiehto on jatkuva toimitus ja sen ideana on, että ohjelmisto on jatkuvasti valmiina julkaistavaksi, ja että uudet julkaisut myös viedään automaattisesti todellisten käyttäjien kättöön [40]. Jatkuva käyttöönotto tähtää ohjelmiston käyttöönottoon välittömästi, kun uutta ohjelmistokoodia on kehitetty [21].

Jatkuvan käyttöönoton etuja ovat esimerkiksi uudet liiketoiminnalliset mahdollisuudet, julkaisujen riskien vähentyminen sekä tarpeettoman ohjelmakoodin kehittämiseltä välttyminen [21]. Koska ohjelmisto tulee jatkuvasti myös käyttäjien käyttöön verrattuna jatkuvaan toimitukseen, jonka kuvattiin luvussa 2.3.4 koskevan vain ohjelmiston vientiä tiettyyn käyttöympäristöön, on palautesykli vieläkin lyhyempi. Tämä mahdollistaa jatkuvan käyttöönoton etuna mainitun tarpeettomien ominaisuuksien kehittämisen vähenemisen [21].

Jatkuva käyttöönotto tekee käyttöönotosta myös yhdenmukaisen, tasaisen, prosessin [101]. Täten tuntemattomien muuttujien määrä käyttöönottoa kohden on minimoitu, ja prosessi on ennakoitavissa. Käyttöönottoprosessi on täysin jäljitettävissä taaksepäin ohjelmiston lähdekoodiin ja vaatimuksiin [50].

Jatkuvan käyttöönoton seurauksena ohjelmistokehittäjä on vastuussa myös käyttöönotoista. Täten laadunvalvontatiimi ei enää tarkista ohjelmakoodin laatua vaan se on samaisten ohjelmistokehittäjien vastuulla. Tämä vastaavasti aiheuttaa stressiä ohjelmistokehittäjille, jolloin kehittäjien ja managerien kommunikaation parantaminen on tärkeää. [21]

Jatkuvassa käyttöönotossa voi olla myös riskejä. Automaattisiin testeihin luottaminen saattaa aiheuttaa ohjelmointivirheiden esiintymistä käyttäjien käytössä, jos testien laatu ei ole käyttäjille julkaistavan ohjelmiston tasolla. Tätä riskiä voi vähentää mahdollistamalla esimerkiksi takaisinkierron eli paluun edelliseen toimivaan versioon. Jatkuvan käyttöönoton toteuttaminen saattaa myös olla aikaa vievää ja monimutkaista. [21]

Jatkuvaan käyttöönottoon soveltuvat esimerkiksi aiemmin mainittu GitLab CI/CD [45]. Myös Travis CI:ta voidaan käyttää siihen. [99].

2.3.6 Jatkuva monitorointi

Jatkuva monitorointi perustuu tarpeeseen ennakoida ohjelmiston ajonaikaista käyttäytymistä ja tämän pysymistä sallitulla rajatulla alueella sekä varmistaa luotettavuusominaisuuksien, esimerkiksi *SLA*:n [49] (Service Level Agreement, sopimus palvelutasosta), toteutuminen ja käyttäjäodotusten täyttyminen [21]. Jatkuvan monitoroinnin voidaan ajatel-

la olevan myös sitä, että ohjelmiston toiminnallisuutta seurataan koko sen elinkaaren aikana. Testaaminen aikaisin ja tuotantoympäristöä vastaavissa suoritusympäristöissä mahdollistaa tämän [101].

Jatkuvaa monitorointia on mahdollista suorittaa järjestelmän kaikilla tasoilla, esimerkiksi fyysisen laitteiston, käyttöjärjestelmän, väliohjelmistojen sekä itse ohjelmiston tilan osalta. [49] Jälkimmäisestä esimerkkinä on tässä alaluvussa aiemmin mainittu ohjelmiston ajonaikaisen käyttäytymisen seuraaminen.

Jatkuvalla monitoroinnilla on etuja. Se mahdollistaa esimerkiksi palvelunlaadun ongelmien kuten suoritushyvyn heikkenemisen aikaisen havaitsemisen. Sen avulla on myös mahdollista kerätä tietoa ohjelmiston käytöstä. [49]

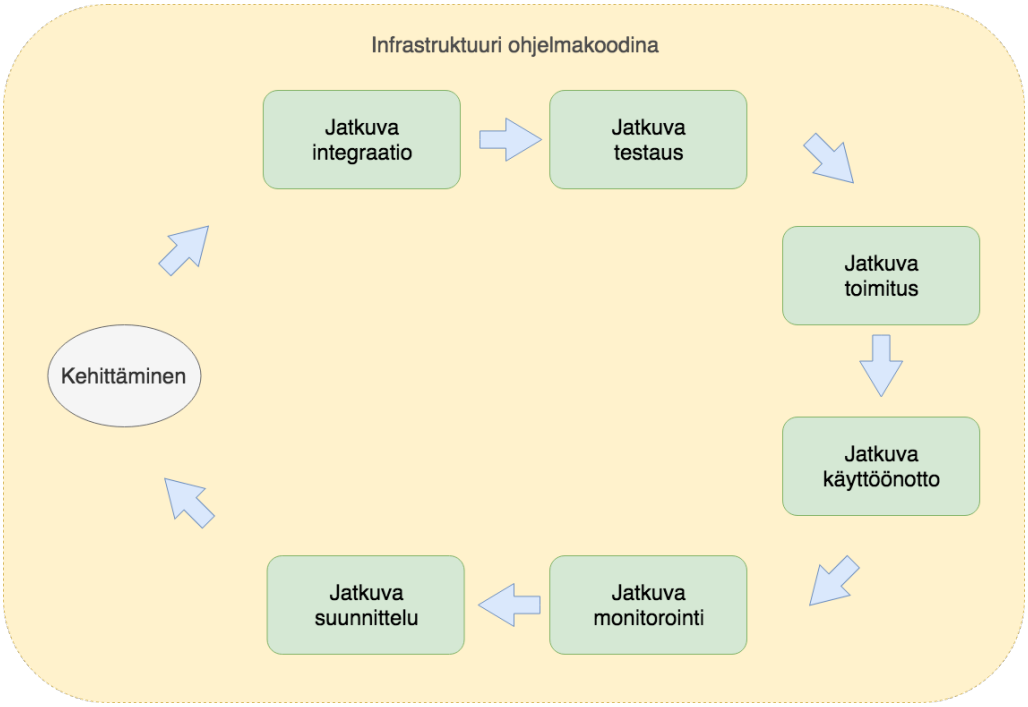
Jatkuvaa monitorointia voi suorittaa kokonaisvaltaisesti esimerkiksi *Zabbix*:n avulla. Se soveltuu muun muassa verkkoyhteyksien, palvelimien, tallennusvälineiden, tietokantojen ja sovellusten monitorointiin [108]. Vastaavasti sovellusten virheiden esiintymistä voi monitoroida *Sentrylla* [94]. Sovellusten lokeja voi monitoroida esimerkiksi *Elastic Stackilla*: lokit kerätään *Logstash*:lla, ne tallennetaan *Elasticsearchiin*, ja ne visualisoidaan *Kibanalla* [32].

2.3.7 Jatkuva suunnittelu

Ohjelmistotuotannon liiketoiminnallisten suunnitelmien on oltava ketteriä, koska markkinoiden olosuhteet saattavat vaihtua nopeastikin. DevOps mahdollistaa nopean reagoinnin muutoksiin priorisoidun ominaisuusluettelon, jatkuvan asiakaspalautteen, ja kyvyn vaihtaa ominaisuusluettelon prioriteetteja, avulla. DevOps-henkisessä suunnitteluprosessissa suunnittellaan pieni osa, suoritetaan se, kerätään palautetta ja reagoidaan siihen. Tämä sykli voi toistua useamman kerran. [101]

Fitzgerald ja Stol määrittelevät jatkuvan suunnittelun holistisena pyrkimyksenä, jossa on mukana useampia sidosryhmän jäseniä liiketoiminnasta ja ohjelmistokehityksestä, ja jossa suunnitelmat kehittyvät markkinatilanteen mukaan ollen täten dynaamisia ja avoimia [39]. Täten jatkuvassa suunnittelussa suunnittelu ja toteuttaminen ovat tiukemmin toisiinsa integroituneina [39]. Fitzgerald ja Stol esittävätkin ilmiön ”BizDev”, joka on laajennus DevOpsiin, pyrkimyksenään lähentää liiketoiminnallista strategiaa ja ohjelmistokehitystä [39]. Kuva 2.4 visualisoi, miten erilaiset DevOps-käytännöt suhteutuvat toisiinsa.

Jatkuvaa suunnittelua tukevat monet erilaiset työkalut. Esimerkiksi pikaviestimillä voidaan kommunikoida nopeasti ja vaivattomasti. Vastaavasti *Trello*-pilvipalvelulla on mahdollista hallita *Kanban*-menetelmää noudattavaa projektia [64]. Kanban painottaa työnkulun visualisointia, työn alla olevien tehtävien määrän rajoittamista sekä syklien keston mittaamista eli sen mittaamista, kuinka kauan yhden tehtävän suorittaminen keskimäärin kestää [69].



***Kuva 2.4**. DevOps-käytännöt toisiinsa suhteutettuina. Idea perustuu osittain [101].*

2.4 Hyötyjä

Kuten alaluvussa 2.2 esitetään IT-organisaatioissa kehittävän ja operatiivisen henkilöstön on nähty sijaitsevan eri yksiköissä, mistä johtuen näiden ammattilaisten välillä saattaa vaikuttaa erilainen kulttuuri, mikä vastaavasti saattaa vaikuttaa negatiivisesti kehittävän ja operatiivisen tiimin väliseen kanssakäymiseen. Koska DevOpsin on esimerkiksi ajateltu kaventavan kuilua kehityksen ja operaatioiden välillä [61], voidaan sen yhtenä hyötynä väittää olevan kuilun aiheuttavien ongelmien poistuminen tai ainakin vähentyminen.

DevOpsin etuina ovat myös alaluvussa 2.3 esitettyjen käytäntöjen hyödyt. Esimerkiksi Riungu-Kalliosaari et al. listaavat näitä vuonna 2016 julkaistussa tutkimuksessaan. DevOpsin etuna he mainitsevat isomman määrän toteutettuja ominaisuuksia ja tiheämmät julkaisut: automatisoitujen käännös-, testaus- ja käyttöönottoprosessien ansiosta aikaa jää enemmän ohjelmiston uusien ominaisuuksien tekemiseen, ja niitä kyetään siirtämään nopeammin ohjelmiston tuotantoympäristöön asiakkaan käyttöön [91].

Julkaisuketjun automatisointi parantaa myös laadunvalvontaa, sillä se helpottaa ohjelmiston toiminnan varmistamista ennen sen siirtämistä tuotantoympäristöön [91]. Ongelmia ja vääriä hälytyksiä ilmenee vähemmän. Erich et. al kertovat yhden yrityksen havainneen myös ohjelmiston laadun parantuneen DevOpsin käyttöönoton myötä [34].

Automatisoinnin ansiosta on lisäksi mahdollista tehdä paljon kooltaan pieniä julkaisuja, jolloin DevOpsin hyödyntäminen saattaa näyttäytyä positiivisesti myös asiakkaalle: pienemmissä julkaisuissa uudet ominaisuudet ovat mahdollisesti helpommin havaittavissa kuin isommissa. Koska julkaisuja tehdään tiheämmin, myös palautetta saadaan ohjel-

miston loppukäyttäjiltä useammin. Tällöin loppukäyttäjät testaavat sovellusta, jolloin sitä voidaan palautteen perusteella kehittää heidän tarpeitansa ja toiveitansa paremmin tyydyttävään suuntaan. [91]

DevOps myös pakottaa kehittävän ja operoivan tiimin keskustelemaan keskenään enemmän, jolloin molemmat osapuolet myös jakavat enemmän osaamistaan ja tietämystään toisilleen. Myös työntekijöiden hyvinvointi saattaa parantua, sillä stressaavia, paljon lisättyjä ominaisuuksia sisältäviä, julkaisuja ei ole, vaan ohjelmisto kehittyy ja rakentuu pienemmissä osissa myös julkaisujen näkökulmasta. [91]

Erich et. al raportoivat myös ongelmanratkaisukykyjen parantuneen eräässä yrityksessä ja ohjelmistojen suoritusympäristöjen pystyttämiseen käytetyn ajan vähentyneen toisessa. Yksi yritys havaitsi testaamisen parantuneen, ja toinen koki DevOpsin tukevan esimerkiksi jatkuvan integraation ja toimituksen käyttöönottoa. [34]

On kuitenkin huomioitava, että Erich et al. [34] ja Riungu-Kalliosaari et al. [91] havainnot perustuvat haastatteluihin, jolloin edellä esitetyt hyödyt eivät välttämättä ole yleistettävissä. Kyseiset tutkijat mainitsevat tämän näkökulman myös itse arvioidessaan tutkimustensa ulkoista kelpoisuutta. Riungu-Kalliosaari et al. kuitenkin huomauttavat, että jokin muukin DevOpsin käyttöönottava organisaatio saattaa havaita heidän havaitsemiaan hyötyjä [91].

2.5 Ongelmia ja esteitä

DevOpsiin on myös havaittu liittyvän ongelmia. Varsinkin sen käyttöönotossa saattaa esiintyä esteitä. DevOpsin monitulkintainen määritelmä saattaa jättää auki, mitä sitä käyttöönottaessa on todellisuudessa tarkoitus tehdä ja saavuttaa. Adoption päämäärät saattavat täten jäädä epäselviksi [95]. DevOps määritelmänä sekä siihen liittyvät käytännöt ja työkalut myös muuttuvat ja kehittyvät koko ajan [91]. Organisaation rakennekin saattaa asettaa esteitä DevOpsin hyödyntämiselle [95] kuten myös siihen syvälle painautuneet käytännöt [91]. Kuten hyötyjen suhteen myös ongelmien ja esteiden osalta on huomioitava, että Riungu-Kalliosaari et al. [91] ja Smeds et al. [95] havainnot perustuvat haastatteluihin.

Alaluvussa 2.3 esitettyjen DevOps-käytäntöjen riskit saattavat myös asettua esteiksi DevOpsin hyödyntämiselle. Esimerkiksi jatkuvaan käyttöönottoon saattaa liittyä pelko siitä, että automaattisiin testeihin luottaminen aiheuttaisi ohjelmointivirheiden esiintymisen kasvua todellisessa käyttöympäristössä [21]. Jatkuva käyttöönotto saattaa myös olla aikaavievää ja monimutkaista [21], mikä vastaavasti saattaa tarkoittaa rahan kulumista ja keskittymisen siirtymistä pois tuottavasta työstä.

Smeds et al. listaavat vuoden 2015 kirjallisuuskatsauksensa yhteydessä tehtyjen haastattelujen perusteella havaittuja DevOpsin kulttuurillisia ja teknisiä esteitä. Kulttuurillisia esteitä ovat työntekijöiden geograafinen jakautuminen, kyllästyminen muotisanaan sekä

pelko, että DevOps aiheuttaa lisää työtä kehittävälle tiimille. Työntekijöiden työskennellessä kaukana toisistaan ympäri maailmaa on esimerkiksi keskustelu kasvostusten hankalaa. Myös se, että DevOps vaatii osaamista sekä kehittävän että operatiivisen alueen osalta, nähdään esteenä. Tämä siksi, että näiden kahden alueen nähdään eroavan toisistaan niin paljon, ettei työntekijä välttämättä kykene tekemään molempia tehtäviä tehokkaasti. [95]

Kehittäjät eivät välttämättä myöskään ole kiinnostuneista operatiivisesta toiminnasta ja päinvastoin. [95]. Kommunikaatio saattaa olla puutteellista ja keskittyminen suuntautunut eri asioihin [91]. Riungu-Kalliosaari et al. huomauttavatkin, ettei kulttuurimuutosta voi pakottaa, vaan organisaation jäsenten on oltava myötämielisiä muutokseen, esimerkiksi oman roolin ja työnkuvan muuttumiseen [91]. Organisaation operatiivinen tiimi myös tietää muutosten mahdollistavan ongelmatilanteet ja saattaa täten pyrkiä minimalisoimaan niiden tekemisen tuotantoympäristöön [24] eli esimerkiksi rajoittamaan uusien ominaisuuksien käyttöönottoa.

Teknisiin esteisiin lukeutuvat esimerkiksi ohjelmiston massiivinen arkkitehtuuri. Tällainen arkkitehtuuri vaikuttaa siihen, miten järjestelmää kehitetään ja testataan, ja miten se on mahdollista ottaa käyttöön. Myös erilaisten ohjelmistojen suoritusympäristöjen erot saattavat muodostua esteeksi: kehitys-, testaus- ja tuotantoympäristöt eivät välttämättä vastaa tuotantoympäristöä. Jos tuotantoympäristöä ei voida simuloida muissa ympäristöissä, on riskinä, ettei ohjelmiston toimintaa pystytä kunnolla varmistamaan ennen kuin se otetaan käyttöön. [95]

Toisaalta tuotantoympäristö voi olla esimerkiksi tietokannan osalta niin monimutkainen, että sen kahdentaminen kehitys- tai testiympäristössä olisi huomattavan haastavaa [91]. Myös tuotantoympäristöjen moninaisuus voi aiheuttaa vaikeuksia DevOpsin adoptiossa ja kehitysprosessin eri vaiheiden yleistäminen ja automatisointi saattaa olla vaikeaa tällaisessa tilanteessa esimerkiksi erilaisten pääsyoikeuksien takia [95].

Tuotantoympäristön pääsyoikeuksien rajoituksia voivat asettaa myös lait ja tehdyt sopimukset [91]. Säännöksetkin saattavat esimerkiksi lääketieteellisissä ohjelmistoissa Laukkarinen et al. [72] mukaan estää DevOps-käytäntöjen hyödyntämisen. He kirjoittavat olevan mahdollista, että kaikilta ohjelmistossa käytetyiltä ohjelmointikieliltä ja -kirjastoilta vaaditaan säännösten noudattamista, mikä rajoittaa mahdollisten käytettävien työkalujen valikoimaa.

Laukkarinen et al. huomauttavat myös, että heidän tietääkseen sellaista työkaluketjua ei ole, joka pystyisi kaikissa ohjelmistokehityksen vaiheissa automaattisesti pitämään kirjaa tietyn vaatimuksen etenemisestä. Säännösten alaisessa ohjelmistokehityksessä tällaisen jäljittämisen pitää kuitenkin olla mahdollista, mikä tarkottaisi DevOps-työkaluja käytettäessä manuaalista asiayhteyksien tekemistä, mikä on vastaavasti pois tuottavasta kehittämisestä. [72]

Debois kommentoi, että projektien nopea siirtyminen kehityksestä eteenpäin saattaa aiheuttaa niiden kertymistä operatiivisen tiimin hallintaan [24]. Täten osa projekteista saattaa viivästyä enemmän kuin toiset. Tämä saattaa aiheuttaa eripuraa eri projektien välillä. Debois huomauttaakin, että kyseisen ilmiön mahdollisuus pitäisi selvittää eri projekteille, ja että näiden managerien pitäisi sopia yhteisesti projektien prioriteeteista operatiivisen toiminnan osalta [24]. Deboisin kommentit ovat ajalta ennen DevOpsin syntyä, mutta saattavat olla edelleen käypiä, jos esimerkiksi operatiivisen tiimin resurssit ovat liian alhaisen hallinnoitaviin projekteihin nähden.

3. TUTKIMUSMENETELMÄ

Kuten luvussa 1 esitetään, on tämän diplomityön tutkimustavoitteena selvittää, miten DevOpsiin liittyviä käytäntöjä hyödynnetään Tampereen teknillisen yliopiston Tieto- ja sähkötekniikan tiedekunnan Tietotekniikan laboratorion opetusjärjestelmien kehittämisessä ja ylläpidossa, ja miten niiden hyödyntämistä voisi näissä kehittää.

Edellä esitettyyn tutkimuskysymykseen ei ole numeraalista vastausta. Vastaus ei esimerkiksi voi olla prosentuaalinen luku. Määrällisessä tutkimuksessa on välttämätöntä hyödyntää standardoituja mittajärjestelmiä [81, s. 14], jollainen on esimerkiksi aika ja sen yksi yksikkö sekunti. Jos tässä tutkimuksessa mitattaisiin esimerkiksi mahdollisten olemassa olevien DevOps-toimitusputkien kestoaikoja tai sitä, kuinka kauan kehitysympäristöjen valmistelu kestää, olisi kyse määrällisestä tutkimuksesta.

Työn tutkimuskysymykseen vastaamiseksi on tutkittava laadullista tietoa. Tällaista tietoa ovat haastattelut, tarkkailu ja dokumentit, ja sitä hankitaan yleensä kenttätyön avulla [81, s. 4]. Tieto käytettävissä olevista työkaluista ja ohjelmistojen suroitusympäristöistä hanktiaan sähköpostihaastattelujen ja -kyselyjen avulla. Näillä saatava data lukeutuu haastatteluihin ja dokumentteihin [81, s. 4]. Alaluku 3.1 kertoo enemmän, miten edellä esitettyä tietoa on tarkoitus kerätä.

Tutkimusdatan kerääminen opetusjärjestelmien kehityksen ja ylläpidon osalta suoritetaan laadullisena tapaustutkimuksena. Tapaustutkimuksella viitataan tutkimukseen, jossa selvitetään nykyhetken ilmiötä syvällisesti todellisen maailman kontekstissa [107, s. 237]. DevOpsin voidaan ajatella olevan tällainen ilmiö, ja opetusjärjestelmien kehityksen sekä ylläpidon todellisen maailman konteksti. Täten laadullinen tapaustutkimus soveltuu tämän työn tutkimuskysymykseen vastaamiseen. Tapaustutkimukseen liittyvät tapaukset voivat olla konkreettisia tai abstrakteja [107, s. 237]. Tämän työn tapaustutkimuksen tapaukset ovat opetusjärjestelmiä ja ne ovat konkreettisia.

Tapaustutkimus voi tähdätä esimeriksi teorian luomiseen, olemassa olevan teorian testaamiseen tai sen laajentamiseen [67]. Tämän työn tapaustutkimuksen voidaan näkökulmasta riippuen ajatella olevan sekä olemassa olevan teorian testaamista että sen laajentamista. DevOpsiin liittyy luvussa 2 esitettyä teoriaa, joten sen toteutumisen selvittäminen opetusjärjestelmien kehityksessä ja ylläpidossa on tämän teorian testaamista. Toisaalta kuten luvuissa 1 ja 3.3 esitetään, ei DevOpsin hyödyntämisestä tässä kontekstissa ole olemassa tutkimusta, joten tämän työn tutkimus on tietyllä tavalla myös teorian laajentamista. Molemmat edellä esitetyt näkökulmat ovat mahdollisia, sillä tapaustutkimuksen eri lähestymistavat eivät ole toisiaan poissulkevia [67].

Seuraavissa alaluvuissa kuvataan, miten edellä esitetyt laadullinen tutkimus ja laadullinen tapaustutkimus aiotaan toteuttaa. 3.1 esittää, miten käytettävissä olevat työkalut ja suoritusympäristöt on tarkoitus kartoittaa, ja 3.2 kertoo, miten DevOps-käytäntöjen hyödyntämistä opetusjärjestelmissä on tavoite selvittää. 3.3 esittää luvun lopuksi aiheeseen liittyvää aiempaa tutkimusta.

3.1 Käytettävissä olevien työkalujen ja suoritusympäristöjen kartoitus

DevOps-käytäntöjen hyödyntämiseen on olemassa monia työkaluja, mistä osviittaa tarjoaa XebiaLabs-yrityksen DevOps-työkalujen jaksollisen järjestelmän taulukko [82]. Osa työkaluista on ilmasia ja vapaasti käytettävissä, osa ei. Koska Tampereen teknillisellä yliopistolla ja Tietotekniikan laboratoriolla saattaa jo olla joitakin työkaluja käytössä, on ensin syytä selvittää, mitä ne ovat. Tämä siksi, että käytettävissä olevia työkaluja on jo saatettu käyttää tai ainakin kokeilla, jolloin kynnys niiden käyttämiselle saattaa olla matalampi.

Samalla on myös hyödyllistä kartoittaa ohjelmistojen mahdolliset suoritusympäristöt ja niiden tekniset ominaisuudet. Tämä siksi, että täten voidaan selvittää, onko esimerkiksi ohjelmiston kehittäjän mahdollista saada pääsyoikeus suoritusympäristöön, jotta hän voi monitoroida ohjelmistoa sen suoritusaikana ja esimerkiksi päivittää sen uusimpaan versioon. Mahdollisten suoritusympäristöjen tekniset tiedot voivat myös tulevaisuudessa auttaa ohjelmistokehitystä opetusjärjestelmissä, sillä täten ohjelmiston infrastruktuuri on mahdollisesti jo aiemmin kahdennettavissa. Koska DevOps syntyi verkkoperustaisten järjestelmien toimittamisen nopeuttamiseen [19], keskitytään työssä vain web-sovellusten suoritusympäristöihin.

Saatavilla olevat työkalut ja suoritusympäristöt on tarkoitus selvittää kysymällä asiaa sähköpostilla Tampereen teknillisen yliopiston IT-helpdeskiltä ja Tietotekniikan laboratorion IT-yhteyshenkilöltä. Saatavilla olevia työkaluja ja suoritusympäristöjä etsitään myös Tampereen teknillisen yliopiston intrasivustolta Tutkasta. Edellä esitetty tieto lukeutuu haastatteluihin ja dokumentteihin ja on täten laadullista tietoa.

Sähköpostikyselyiden osalta työkaluihin liittyvä kysymys muotoillaan siten, että se kysyy, mitä ohjelmistokehityksen apuvälineitä on saatavilla käyttöön, ja annetaan myös esimerkki yhdestä sellaisesta. Jos vastaanottaja ei tunnista, mistä on kyse, avataan asiaa tälle enemmän. Suoritusympäristöjen osalta kysytään, millaisia ympäristöjä on saatavilla teknisiltä ominaisuuksiiltaan, ja millaiset pääsyoikeudet ohjelmiston kehittäjän on mahdollista saada ympäristöihin.

Ensimmäisessä sähköpostiviestissä vastaanottajaa pyydetään raportoimaan toinen henkilö, jos hänellä itsellään ei ole tietämystä saatavilla olevista työkaluista tai suoritusympäristöistä. Tätä menetelmää noudatetaan niin kauan kunnes joko työkaluista ja suoritusympäristöistä saadaan tietoa tai sähköpostiviestin vastaanottaja ei tiedä, kuka tietäisi niistä

enemmän. Jos tässä alaluvussa esitetyllä menetelmällä ei saada selvitettyä saatavilla olevia työkaluja ja ohjelmistojen suoritusympäristöjä, selviävät käytetyt ja täten siltä osin saatavilla olevat työkalut ja suoritusympäristöt tutkimuksen myöhemmässä vaiheessa.

3.2 Laadullinen tapaustutkimus opetusjärjestelmiin

Opetusjärjestelmien osalta suoritetaan laadullinen tapaustutkimus. Siinä keskitytään sellaisiin järjestelmiin, jotka ovat käytössä Tampereen teknillisen yliopiston Tietotekniikan laboratorion opetuksessa, ja joiden kehitykseen laboratorion henkilökunta on osallistunut. DevOps syntyi web-järjestelmien toimitusten ripeyttämiseen [19], joten tässä työssä keskitytään vain web-perusteisiin järjestelmiin.

Opetusjärjestelmien ohjelmistokehitys- ja ylläpitovaiheissa hyödynnetyt DevOps-käytännöt ja -työkalut on tarkoitus kartoittaa omatoimisen tutustumisen ja kyselyn avulla. Lisäksi näiden avulla tehtyjä havaintoja on tarkoitus vertailla. Seuraavissa alaluvuissa näitä tapaustutkimuksen eri vaiheita kuvataan tarkemmin.

3.2.1 Omatoiminen tutustuminen

Omatoimisessa tutustumisessa työn tekijän tarkoitus on tutustua opetusjärjestelmiin liittyvään ohjelmakoodiin itse. Tämä voidaan tehdä tutkimalla näihin liittyviä versionhallintaprojekteja. Tapaustutkimuksen tapaukset eli tutkittavat projektit ja järjestelmät voidaan hankkia kyselemällä näihin tutustumisen mahdollisuutta laboratorion työntekijöiltä esimerkiksi laboratorion käytävillä, Slack-pikaviestimessä ja lähettämällä sähköpostia laboratorion sähköpostilistalle.

Omatoimisessa tutustumisessa työn tekijä voi versionhallintaprojektien sisältöä tutkimalla selvittää, mitä DevOps-käytäntöjä ja -työkaluja järjestelmissä on käytetty etsimällä näihin liittyviä tiedostoja kuten jatkuvan integraation konfigurointeja, esimerkiksi Jenkinsiin liittyviä Jenkinsfileja. Nämä ovat dokumentteja eli laadullista tietoa. Työn tekijä yrittää myös pystyttää ohjelmiston suoritusympäristön omalle tietokoneelleen, millä on mahdollista selvittää, kuinka hyvin kyseisten järjestelmien infrastruktuuri on dokumentoitu ja määritelty ohjelmakoodiksi, ja kuinka helposti se on kahdennettavissa. Myös tämä tieto on laadullista, tarkemmin tarkkailua ja havaintoja.

3.2.2 Kysely

Tutkimuksen kyselyosassa työn tekijän tavoitteena on selvittää, mitä DevOps-käytäntöjä ja -työkaluja laboratorion opetusjärjestelmien kehittämisessä ja ylläpitämisessä on hyödynnetty. Kysely suoritetaan verkkokyselynä *Webropol*-palvelussa. Ensin työn tekijä luo kyselyn. Jotta kysely noudattaisi *GDRP*-asetusta (EU:n eli Euroopan unionin yleinen tietosuoja-asetus), kysytään kyselyn vastaanottajilta etukäteen, sallivatko he kyselylinkin

lähettämisen heille. Webropol noudattaa GDRP-asetusta, sillä kaikki tieto on tallennettuna EU-alueella eikä sitä luovuteta sen ulkopuolelle tai käsitellä sen ulkopuolella [98].

Kysely suoritetaan oletuksena nimettömänä. Vastaajalle tarjotaan kuitenkin mahdollisuus kertoa nimensä ja sähköpostiosoiteeseensa, jos hän on halukas vastaamaan työn tekijän mahdollisiin lisäkysymyksiin. Vastaajalle annetaan myös mahdollisuus keskusteluhetken toivomiseen, jolloin nimi ja sähköpostiosoite on täytettävä, ja jolloin työn tekijä ottaa vastaajaan yhteyttä sähköpostilla sopiakseen keskustelun ajankohdan. Vaikka vastaaja kertoisi kyselyyn vastatessaan nimensä ja sähköpostiosoitteensa, käsitellään vastauksia tässä työssä täysin anonyymisti.

Kyselyssä käytetään valinta-, monivalinta- ja matriisikysymyksiä. Näitä täydentämään käytetään myös avoimia kysymyksiä. Valinta-, monivalinta- ja matriisikysymykset ovat pakollisia, ja niissä tarjotaan myös vastausvaihtoehdot *"En osaa sanoa"* tai *"Muu: <vastaajan syöte>"*. Avoimet kysymykset eivät ole pakollisia, koska vastaaja voi kokea, ettei osaa vastata niihin. Kysely toteutetaan suomen kielellä.

Kyselyn alustava kohdeyleisö on verrattain pieni, joten kyselyn aukioloaika on noin kolme tai neljä viikkoa. Vastaajia ohjeistetaan vastaamaan kyselyyn yhtä monta kertaa kuin on olemassa sellaisia opetusjärjestelmiä, joiden kehityksessä, ylläpidossa tai hallinnoinnissa he kokevat jollakin tavalla olevansa osallisina. Näin siksi, ettei vastaaja joudu samaan aikaan vastaamaan useamman järjestelmän osalta. Myös kysely on täten mahdollista pitää selkeämpänä.

Kyselyssä kysytään aluksi, minkä opetusjärjestelmän osalta vastaaja vastaa. Tämän jälkeen kysytään vastaajan omasta roolista ja järjestelmään liittyvästä tiimistä. Käyttäjää pyydetään kuvaamaan järjestelmän julkaisuketjua tai -käytäntöjä sekä kertomaan, hyödyntääkö järjestelmä luvussa 2.3 esitettyjä käytäntöjä. Näiden osalta vastaajia pyydetään myös raportoimaan käytetyt työkalut. Vastaajalta kysytään myös, mitä hyvää tai mitä ongelmia kyseessä olevassa järjestelmässä on sen kehityksen ja ylläpidon näkökulmasta. Nämä kysymykset ovat laadullisia ja niiden vastaukset vastaavasti dokumentteja.

Kyselyn vastaajakandidaatit valitaan sen perusteella, ketkä ovat jäseninä niissä versionhallintaprojekteissa, jotka liittyvät niihin opetusjärjestelmiin, joihin työn tekijä on itse tutustunut, tai joiden kanssa työntekijä on keskustellut opetusjärjestelmistä. Myös Tampereen teknillisen yliopiston IT-helpdeskiltä kysytään, vastaako joku henkilö tai jotkut henkilöt IT-palveluissa jonkin järjestelmän ylläpidosta, ja olisivatko nämä henkilöt halukkaita vastaamaan kyselyyn. Tarvittaessa kysely voidaan lähettää myös muille henkilöille, jos selviää, että he ovat muulla tavalla osallisina jossakin opetusjärjestelmässä, tai jakaa laboratorion Slack-pikaviestimen opetusaiheisilla keskustelukanavilla.

3.2.3 Havaintojen vertailu

Kysely tukee omatoimista tutustumista tämän työn tutkimuskysymykseen vastaamisessa. Sen avulla on mahdollista pyrkiä vahvistamaan työn tekijän tekemiä havaintoja. Tämä on

tärkeää, sillä omatoimisen tutustumisen suorittaa yksi henkilö, jolloin menetelmän riskinä on liiaallinen subjektiivisuus sekä kyseisen henkilön oman DevOps-tietämyksen mahdollinen puutteellisuus. Työn tekijä on pyrkinyt vähentämään jälkimmäistä riskiä tekemällä kirjallisuusselvityksen DevOpsista luvussa 2.

Omatoimisen tutustumisen ja kyselyn perusteella on myös mahdollista suorittaa vertailua. Vaikka jälkimmäisen menetelmän perusteella kerätyn tiedon avulla työn tekijän omia havaintoja pyritään vahvistamaan, saatta myös havaituilla eroilla mahdollisesti olla merkitystä.

Opetusjärjestelmien kehityksessä, ylläpidossa tai hallinnoinnissa mukana oleville henkilöille osoitetun kyselyn perusteella voi myös olla mahdollista selvittää, millaisia eroja saman projektin tai järjestelmän jäsenten näkemyksissä on. Voi olla mahdollista esimerkiksi selvittää, miten ohjelmiston kehittäjän ja sitä operoivan eli ylläpitävän henkilön näkemykset eroavat. Voi myös olla mahdollista selvittää, miten itse järjestelmän tekijän ja vastuuhenkilön näkemykset aiheesta eroavat.

Näiden vertailujen perusteella saatetaan saada tietoa siitä, kuinka hyvin erilaiset käytännöt on dokumentoitu, ja kuinka hyvin opetusjärjestelmien jäsenet kommunikoivat keskenään. Edellä mainittu vertailun vaatimuksena on, että opetusjärjestelmän tiimin koko on isompi kuin yksi, ja että kyselyyn vastaa useampi kuin yksi henkilö siitä.

Vertailussa tehdyt havainnot ovat laadullista tietoa. Ylipäätään on hyödyllistä tapaustutkimuksen laadun kannalta, että siinä käytetään useampia tietolähteitä, tämän tutkimuksen osalta erilaisia havaintoja ja dokumentteja, ja esimerkiksi Yin listaa tämän näkökulman tapaustutkimuksen tiedonkeruun neljän periaatteen joukkoon [107, s. 118-120]. Yin myös kirjoittaa, että tapaustutkimuksen havainnot ja päätelmät ovat vakuuttavampia ja virheettömämpiä, jos ne perustuvat useampaan tietolähteeseen.

3.3 Liittyvä tutkimus

DevOpsia on tutkittu eri näkökulmista. Esimerkiksi Ericht et al. ovat tehneet vuonna 2014 systemaattisen kartoitustutkimuksen käyttäen ”DevOps”:ia hakusanana [33]. Myös Jabbari et al. ovat suorittaneet systemaattisen kartoitustutkimuksen vuonna 2016 tavoitteenaan selvittää, miten DevOps on määritetty kirjallisuudessa [61]. Vastaavasti Smeds et al. ovat suorittaneet vuonna 2015 kirjallisuuskatsauksen tavoitteenaan DevOpsin määrittelevien ominaisuuksien selvittäminen [95]. Näiden tutkimuksien esittämät määritelmät DevOpsista on esitetty luvussa 2.1.

Smeds et al. vuoden 2015 tutkimus selvittää myös haastatteluihin perustuen DevOpsin omaksumiseen liittyviä esteitä ja sen ongelmia [95]. Näitä havaintoja on esitetty luvussa 2.5. Myös Riungu-Kalliosaari et al. esittävät vuonna 2016 julkaistussa tutkimuksessaan yritysten työntekijöille tehtyjen haastattelujen pohjalta tällaisia esteitä, mutta myös

DevOpsin omaksumisen hyötyjä [91]. Näitä hyötyjä on kuvattuna luvussa 2.4 ja esteitä luvussa 2.5.

Opetusjärjestelmiä ovat esittäneet tutkimuksessa esimerkiksi Karavirta et al. ja Ohtsuki et al. Karavirta et al. esittävät vuonna 2013 julkaistussa konferenssijulkaisussaan *A+*-opetusjärjestelmän [66], ja vastaavasti Ohtsuki et al. vuonna 2016 julkaistussa konferenssijulkaisussaan *ALECSS*-opetusjärjestelmän [78]. ALECSS hyödyntää DevOps-käytäntöjä kuten jatkuvaa integraatiota Jenkinsin avulla opetuksessa [78], mutta siihen liittyvä julkaisu ei kerro, onko näitä käytäntöjä hyödynnetty myös kyseisen järjestelmän kehittämisessä ja ylläpidossa.

DevOpsia on myös opetettu yliopistossa. Esimerkiksi Moroccon Cadi Ayyad University on järjestänyt pilvilaskentakurssin, jonka oppimistavoitteisiin on lukeutunut DevOps. Kurssi oli jaettuna kahteen osaan: luentoihin, joissa käsitteltiin DevOpsia, sekä projektityöhön, jossa luennoilla esitettyjä konsepteja hyödynnettiin käytännössä. Kurssin tavoitteena oli, että opiskelijat tutustuisivat DevOps-kulttuuriin ja -työkaluihin. [11]

Tämän diplomityön puitteissa ei havaittu aiempaa tutkimusta koskien DevOps-käytäntöjen hyödyntämistä opetusjärjestelmien kehityksessä ja ylläpidossa, mutta DevOpsia on hyödynnetty tieteellisessä tutkimuksessa. Esimerkiksi Bayser et al. ovat hyödyntäneet siihen liittyviä käytäntöjä ja prosesseja *IBM*:n Brasilian tutkimuskeskuksessa nimeten lähestymistapansa ”ResearchOps”:ksi [19]. Koska opetusjärjestelmä voi olla myös tutkimusprojekti, on edellä mainittu oleellista myös tämän diplomityön kannalta.

Bayser et al. tavoitteena on ollut, että kaikilla tutkimusprojektissa mukana olevilla tutkijoilla olisi aina käytettävissään viimeisin prototyyppi siihen liittyen, ja että he kykenisivät itsekin osallistumaan sen kehitykseen. He perustelevat DevOpsin hyödyntämistä esimerkiksi sillä, että heidän tutkimuksensa on täsmällistä tutkimusalan tietämystä vaatiivaa toimintaa, jolloin tutkijan ja ohjelmoijan eriyttäminen olisi hankalaa. [19]

Bayser et al. myös esittävät, että DevOpsista saattaisi olla hyötyä tutkimukselle. Tutkijalle on esimerkiksi saatettu asettaa rajoituksia käytettävän käyttöjärjestelmän osalta, mikä saattaa rajoittaa käytettävien teknologioiden valikoimaa yhteensopivuusongelmien takia. On myös heidän mukaansa yleistä, että esimerkiksi tohtorikoulutettavat ottavat kehitettäväkseen ohjelmakoodin, jota ovat aiemmin kehittäneet heidän ohjaajansa sekä muut opiskelijat, ja joka saattaa olla puutteellisesti dokumentoitua tai kommentoitua. Edellä mainittuihin ongelmiin he esittävät mahdollisina ratkaisuina virtualisoinnin ja infrastruktuurin jakamisen ohjelmakoodina versionhallinnan välityksellä. [19]

Economou et al. ovat esittäneet, että DevOpsista saattaisi olla hyötyä myös esimerkiksi astronomisille observatorioille [31]. Myöhemmin tällaiset observatoriot ovatkin hyödyntäneet DevOps-käytäntöjä projekteissaan. Esimerkiksi Firethorn-projekti on hyödyntänyt infrastruktuurista ohjelmakoodia Dockerin avulla: sillä virtualisoitiin projektin sovelluksen osia kontteihin, ja näiden konttien määritykset jaettiin versionhallinnassa Dockerfile-tiedostoina [75].

Wettinger et al. ovat puolestaan suorittaneet useita tutkimuksia, jotka liittyvät jollakin tavalla DevOps-menetelmien tai -käytäntöjen kehittämiseen. Esimerkiksi vuonna 2014 julkaistussa konferenssijulkaisussa he esittävät *DevOpSlang*-sovellusaluekielen sekä *Devopsfile*-tiedostomuodon, jota noudattavien tiedostojen avulla määritettäisiin ohjelmiston DevOps-toteutus. [104]

Vuoden 2014 konferenssijulkaisussaan Wettinger et al. [105] kategorisoivat eri työkaluihin liittyvät DevOps-artefaktit tietoelementtikohtaisiksi eli esimerkiksi tiettyyn virtuaalikoneeseen liittyviksi sekä ympäristökohtaisiksi eli koko sovelluksen suoritusympäristöön liittyviksi. Koska erimuotoisten artefaktien yhtäaikainen käyttö on haastavaa, he muunsivat ne *TOSCA*-tyypeiksi (Topology and Orchestration Specification for Cloud Applications -tyypeiksi), jotta niitä voisi käyttää saumattomasti ja yhteentoimivasti. TOSCA on OASIS-konsortion avoin standardi, joka määrittää esimerkiksi pilvipalveluissa suoritettavien palveluiden ja ohjelmistojen yhteentoimivat komponentit, yhteydet, riippuvuudet, vaatimukset ja kyvyt mahdollistaen täten tällaisen palvelun tai ohjelmiston siirretävyyden [77].

Wettinger et al. myös esittävät vuonna 2017 julkaistussa artikkelissaan ratkaisutietovaraston, johon voitaisiin kerätä uudelleenkäytettäviä DevOps-ratkaisuja tai viittauksia niihin. Näille ratkaisuille määritettäisiin malli, ja ne esitettäisiin jollakin merkintäkielellä, esimerkiksi *YAML*:lla. Erilaiset DevOps-artefaktit ja -ratkaisut olisi mahdollista kerätä automatisoidusti ja ne voitaisiin tarvittavan prosessoinnin jälkeen lisätä tietopankkiin, joka voitaisiin julkaista halutunlaisena teknisenä toteutuksena. [103]

4. SAATAVILLA OLEVAT TYÖKALUT JA SUORITUSYMPÄRISTÖT

Saatavilla olevat työkalut ja suoritusympäristöt selvitettiin luvussa 3.1 esitetyn tutkimusmenetelmän mukaisesti: niitä etsittiin Tampereen teknillisen yliopiston intrasivustolta, Tutkasta, sekä lähettämällä sähköpostia yliopiston IT-helpdeskille ja Tietotekniikan laboratorion IT-yhteyshenkilölle. Diplomityön tekijä lähestyi sähköpostin välityksellä näiden yhteydenottojen sekä ammattitiedon jakamisen ja keskustelujen seurauksena myös muuta Tietotekniikan laboratorion henkilöstöä, jolla olisi mahdollisesti tietämystä näihin aiheisiin liittyen. Alaluku 4.1 esittelee työkalut ja 4.2 suoritusympäristöt.

4.1 Työkalut

Tampereen teknillisellä on käytössä GitLab-järjestelmän Community Edition -versio. Kyseisessä versiossa ovat Core-tilaustason ominaisuudet [56], jotka ovat ilmaisia [46]. Core-ominaisuuksiin lukeutuvat esimerkiksi sisäänrakennettu tuki jatkuvalle integraatiolle, toimitukselle ja käyttöönotolle sekä Issue Board [46]. Täten saatavilla on työkalu, jolla on mahdollista toteuttaa jatkuva integraatio, toimitus sekä käyttöönotto järjestelmään, ja josta on mahdollista saada tukea jatkuvaan suunnitteluun.

Tampereen teknillisellä yliopistolla on käytössä kaksi erillistä GitLab-asennusta. Yksi on opetukseen, ja toinen on tiedekuntien, laboratorioiden sekä yliopiston henkilökunnan ja opiskelijoiden käyttöön. Jälkimmäiseen luotuja henkilökohtaisia projekteja ei saa käyttää töissä vaan tiedekunnan tai laboratorion GitLab-yhteyshenkilöä on pyydettävä luomaan projekti. Syy tähän on se, että henkilökohtaiset projektit poistuvat suljettaessa käyttäjän Tampereen teknillisen yliopiston käyttäjätunnukset.

Tampereen teknillisen yliopiston Tietotekniikan laboratoriossa on opetuksessa kokeilukäytössä myös SonarQube opiskelijoiden harjoitustöiden ohjelmakoodin staattiseen analyysiin ja laadunseurantaan. Sellaista asennusta, jota voisi käyttää myös opetusjärjestelmien kehittämiseen, ei kuitenkaan ole vielä työn tekijän tiedon mukaan konfiguroituna.

Tutkimuksen aikana laboratoriolla havaittiin myös Jenkins-palvelin. Sillä ei kuitenkaan ole tällä hetkellä ylläpitäjää, ja myös sen versio on vanhentunut. Jenkinsin uusin pitkän aikavälin tuen versio on kirjoitushetkellä 2.138.3 [63] ja laboratoriolla on asennettuna versio 1.563. Palvelimelta puuttuu myös `TLS`-sertifikaatti (Transport Layer Security, salausprotokolla).

Diplomityön teon ohessa havaittiin laboratorion käytössä myös Slackin työympäristö. Sitä

on mahdollista hyödyntää jatkuvan suunnittelun tukena, sillä pikaviestimet mahdollistavat nopean ja joustavan keskustelun internetin välityksellä.

Vuoden 2018 lopussa Slackin sijaan Tietotekniikan laboratoriossa siirryttiin käyttämään *Mattermost*-pikaviestintä. Tiimi kyseiseen järjestelmään on käytössä myös Tampereen yliopiston Informaatioteknologian ja viestinnän tiedekunnan Tietotekniikan yksikössä 2019 alkaen. Lisäksi entisellä Tampereen teknillisen yliopiston henkilöstöllä säilyy työ- ja opetuskäytössä aiemmin tässä luvussa esitetyt GitLab-asennukset ainakin vuoden 2019 alun ajan. Kirjoitushetkellä on epäselvää, miten GitLabin osalta toimitaan tulevaisuudessa.

4.2 Suoritusympäristöt

Tampereen teknillisellä yliopistolla on käytössä tutkimusverkko, joka on tuotantoverkon ulkopuolella, ja jonka rajoitukset esimerkiksi palomuurin osalta ovat lievemmät kuin tässä. Tutkijan tai opetusjärjestelmän kehittäjän on mahdollista kytkeä oma laitteensa, esimerkiksi verkkopalvelin, tutkimusverkkoon ottamalla yhteyttä IT-helpdeskiin. Myös virtuaalikoneiden kytkeminen on mahdollista. Yliopiston IT-palvelut ylläpitää verkkoa, mutta siihen kytkettyjen laitteiden ylläpitovastuu on laitteen omistajalla.

Tutkimusverkon laitteille on saatavissa *IPv4* ja *IPv6*-osoitteet sekä *DNS*-nimi (Domain Name System, nimipalvelujärjestelmä). Tutkimusverkon laitteet ja palvelimet voivat käyttää sellaisia Tampereen teknillisen yliopiston sisäverkon palveluita, jotka ovat saatavissa ulkoverkon kautta. Myös muut palvelut ovat niin sovittaessa yleensä mahdollisia.

Tampereen teknilliseltä yliopistolta on saatavissa myös virtuaalikoneita. Näihin koneen tilaaja saa pääkäyttäjän oikeudet ja etäyhteysmahdollisuuden. Mahdollisia käyttöjärjestelmiä ovat *Microsoft Windows* ja *Redhat Enterprise Linux*. IT-palvelut asentaa käyttöjärjestelmän, mutta muiden ohjelmien asentaminen sekä virtuaalikoneen ylläpito on sen omistajan vastuulla. Tietyissä tilanteissa myös IT-palveluiden ylläpito on mahdollista. Virtualisointi suoritetaan *VMware*n ohjelmistoilla.

Virtuaalipalvelinkoneet on jaettu kolmeen ryhmään: perus, ylläpidetty ja ylläpidetty vaativa palvelinympäristö. Peruspalvelin ei sisällä ylläpitotyötä tai varmuuskopiointia, ja siihen on mahdollista saada yksi tai kaksi suoritinydintä, nelgä gigabittiä keskusmuistia, viisikymmentä gigabittiä levytilaa sekä Microsoft Windows tai Redhat Enterprise Linux-asennus. Keskusmuistia ja suoritintehoa on mahdollista tilata enemmän lisähintaan.

Ylläpidetty ja ylläpidetty vaativa palvelinympäristö ovat kalliimpia vuositasolla, mutta sisältävät ylläpidon, ja virtuaalikoneiden tekniset vaatimukset ovat sovittavissa tapauskohtaisesti. Yliopiston IT-palveluilla on myös tarjota sekä *SSD*- (Solid State Drive, tallennusväline) että kovalevytilaa. Palvelu ei sisällä varmuuskopiointia.

Tampereen teknillisellä yliopistolla on saatavilla myös niin kutsuttu webhotel-palvelu. Siellä on mahdollista suorittaa esimerkiksi *CGI*-protokollaa tai *Python*-ohjelmointikieltä

hyödyntävää web-sovellusta tai -sivustoa. Tuettuja teknologioita ja saatavilla olevia palveluita ovat *Apache*-palvelinohjelma, CGI sekä ohjelmointikielet *PHP*, Python ja *Perl*. Myös *MySQL*-relaatiotietokanta on mahdollinen.

Saatuaan tilaamansa webhotellin käyttäjän pitää vielä erikseen pyytää käyttöoikeus siihen Tampereen teknillisen yliopiston tunnuksenhallinnan kautta. Webhotellien käyttöjärjestelmien ja niihin valmiiksi asennettujen ohjelmistojen ylläpidosta vastaa IT-palvelut. Verkkosivustoon liittyvien ohjelmistojen, esimerkiksi *Wordpress*-sisällönhallintajärjestelmän, ylläpidosta vastaa kuitenkin verkkosivuston omistaja. Jos palvelimen ohjelmistopäivitys aiheuttaa ongelmia verkkosivuston käytössä, on ongelman korjaaminen sen omistajan vastuulla.

Tampereen teknillisellä yliopistolla havaittiin olevan tarjolla myös *Microsoft Azure* -pilvilaskentapalvelu. Se tarjoaa monenlaisia pilvipalveluita, esimerkiksi tekoälyä, konelaskentaa, tietokantoja, tallennustilaa ja suoritusympäristöjä web-sovelluksille [28]. Palvelun käyttöön yliopistolla ei löydetty ohjeistusta.

IT-helpdeskiltä saadun tiedon mukaan Azuren käyttö on mahdollista henkilökunnan osalta. Käyttö ei ole kuitenkaan täysin vapaata vaan käyttöoikeuksia on hallinnoitava ja laskutustileille on asettettava jonkinlaiset käyttörajat. IT-palveluilla on vuonna 2018 kapasiteettia palveluun, mutta suuremmat käyttökulut kohdistetaan kullekin palvelua käyttävälle yksikölle.

Vastaavasti tietotekniikan laboratoriolla on saatavilla *OpenStack*-klusteri, johon on mahdollista pystyttää virtuaalikone. Klusterin resurssit ovat kuitenkin sen ylläpitäjän mukaan rajalliset, mikä pitää ottaa huomioon virtuaalikonetta luotaessa. Virtuaalikoneen tilaajan kanssa keskustellaan esimerkiksi käyttöjärjestelmäversiosta, ja hän saa koneelle ylläpitäjän oikeudet. Tilaaja on täten myös vastuussa kaikesta, mitä hänen virtuaalikoneellaan tapahtuu.

OpenStack-klusterin palomuuri estää oletuksena kaiken sisään tulevan liikenteen, mutta sen ylläpitäjä avaa tarvittaessa portteja. Klusterin virtuaalikoneiden julkisille IP-osoitteille luodaan tietyntyyliset DNS-nimet, mutta koneen omistajan toiveiden mukaisten DNS-nimien asettamisesta vastaa yliopiston IT-palvelut.

Tietotekniikan laboratoriossa on myös yksityisessä verkossa testikäytössä *Triton DataCenter* -klusteri. Lisäksi on mahdollista saada VMWarella virtualisoitu virtuaalikone esimerkiksi tutkimusprojektin käyttöön. Triton DataCenterin osalta tekniset tiedot ovat avoimet, sillä se ei ole kirjoitushetkellä vielä kaikkien käytettävissä.

Suoritusympäristöjen tekniset tiedot ja käytännöt Tampereen yliopiston osalta ovat avoimet kirjoitushetkellä. Esimerkiksi uutta intrasivustoa ei ole vielä julkaistu, joten tietoja ei voi selvittää sieltä. Microsoft Azure vaikuttaisi kuitenkin olevan käytettävissä myös uudessa yliopistossa. Siihen liittyvissä tiedusteluissa kannattanee kuitenkin lähestyä yliopiston IT-helpdeskiä.

5. DEVOPS-KÄYTÄNNÖT JA -TYÖKALUT OPETUSJÄRJESTELMISSÄ

Luvussa 3.2 esitetyn tutkimusmenetelmän perusteella tehtiin tapaustutkimus opetusjärjestelmiin liittyen. Mahdolliset eroavaisuudet esitettyyn tutkimusmenetelmään on esitetty alaluvuissa.

Kandidaatit tutkittaviksi opetusjärjestelmiksi selvisivät ammattitiedon jakamisen ja työntekoon liittyvän kommunikoinnin myötä. Näiden kandidaattien kehityksessä mukana oleville henkilöille lähetettiin sähköpostia liittyen mahdollisuuteen tutustua kyseiseen järjestelmään. Yhden järjestelmän osalta kehittäjää lähestyttiin Slackin avulla sekä kasvotusten.

DevOps-käytäntöjen ja -työkalujen hyödyntämistä opeutsjärjestelmissä tutkittiin omatoimisen tutustumisen, kyselyn sekä näiden perusteella suoritetun vertailun avulla. 5.1 esittää omatoimisen tutustumisen tulokset, 5.2 kyselyn tulokset ja 5.3 vertailun tulokset.

5.1 Omatoiminen tutustuminen

Omatoimisessa tutustumisessa työn tekijä tutki järjestelmiin liittyviä versionhallintaprojekteja sekä lähdekoodia ja pystytti kehitysympäristön. Tutustuminen suoritettiin tietokoneella, jonka käyttöjärjestelmä oli 64-bittinen *Ubuntu* 18.04.1 LTS. Kyseisessä käyttöjärjestelmässä olivat ennen tutustumisen aloittamista asennettuna myös Python-versiot 2.7.15rc1 ja 3.6.6. sekä *SQLite* 3.22.0 -tietokanta.

Tutustumiseen käytettyjen *Git*-versionhallinnan commitien tiivisteet ovat seuraavat:

- Repolainen: `a0a347e10db1b5f8d7456a67d76204d33adb2afb`
- A+: `f5037dcf6eabc3a406edd86d3f3fb55ae748b194`
- Versionhallinta Gitillä: `180cddee62fddebde353d7c62377fd248932ce2e` ja `7b5ada09403ce9814f2a24260cbad5e99e354600`
- prplatform: `cc39e3afb7cddc7c6e56bca6e9fae02ccae062a1`

Seuraavat alaluvut esittävät omatoimisen tutustumisen tulokset: 5.1.1 analysoi Repolaista, 5.1.2 A+-järjeselmää, 5.1.3 Versionhallintaa Gitillä -verkkokurssia ja 5.1.4 prplatformia.

5.1.1 Repolainen

Repolainen on web-perustainen opetusjärjestelmä, joka mahdollistaa opiskelijoiden hyödyntää nykyaikaisia ohjelmistokehityksen käytäntöjä suorittaessaan opintojaksoa [90].

Se on Tietotekniikan laboratorion (silloisen Tietotekniikan laitoksen) opiskelijan Mika Mäenpään diplomityön ohessa tehty työkalu, jonka tarkoitus on helpottaa versionhallinnan käyttöä sekä opintojakson henkilökunnan että opiskelijoiden näkökulmasta [58]. Sittemmin Repolaista on jatkokehitetty. Mäenpään diplomityö ei ole saatavilla, joten siihen ei voida viitata tässä diplomityössä.

Repolainen integroituu GitLabiin. Opiskelija voi palauttaa opintojaksoon liittyvän suoritteen sen käyttöliittymän kautta Git-versionhallinan avulla, tarkistaa palautukseen liittyvän arvosanan ja opettajan palautteen sekä tarkistaa automaattisten testien tulokset, jos ne ovat opintojaksolla käytössä [90]. Sittemin tehtävien palautus Repolaisen kautta on kuitenkin poistunut käytöstä, koska kyseinen osa järjestelmästä ei ole enää yhteensopiva GitLabin kanssa.

Jukka-Pekka Venttola mainitsee kandidaatintyössään, että Repolaisella on myös mahdollista hyödyntää GitLabin osana olevaa jatkuvan integraation työkalua, ja että se on mahdollista ottaa käyttöön opiskelijoiden projekteissa tämän avulla [100]. Tutustuttaessa järjestelmään havaittiin, että edellä mainittu on mahdollista ainakin järjestelmän käyttöliittymän ja lähdekoodin perusteella.

Edellä mainitun perusteella on perusteltua väittää, että Repolaisen ja GitLab CI/CD:n avulla Tampereen teknillisen yliopiston opiskelijoille on jo mahdollista opettaa DevOps-käytäntöjä, tässä tapauksessa esimerkiksi jatkuvaa integraatiota. Koska GitLab CI/CD mahdollistaa myös esimerkiksi jatkuvan käyttöönoton [45], on sitäkin teoriassa mahdollista opettaa opintojaksoilla.

Tutustuttaessa Repolaisen GitLab-projektiin ja lähdekoodin on havaittavissa, että siinä itsessäänkin on hyödynnetty DevOps-käytäntöjä osittain. Tietovaraston juuressa on `.gitlab-ci.yml`-tiedosto, jolla projektiin on määritetty GitLab CI/CD -toimitusputki. Projekti hyödyntää jatkuvaa integraatiota, sillä jokaisen haaran viimeisimmälle commitille suoritetaan automatisoidut testit. `master`-haaran commitit otetaan käyttöön järjestelmän testiympäristössä, joka ei kuitenkaan ollut saavutettavissa tutkimushetkellä.

Projekti hyödyntää myös jatkuvaa toimitusta ja käyttöönottoa, sillä jokainen versionhallinan tagi otetaan automaattisesti käyttöön myös järjestelmän tuotantoympäristössä. Kun tagin avulla julkaistaan uusi versio, tulee se likimmiten heti myös järjestelmän loppukäyttäjien käyttöön. Järjestelmässä ei kuitenkaan havaittu olevan automaattista edelliseen toimivaan versioon palaamista, jos uudessa versiossa havaittaisiin ongelma vasta sen käyttöönoton jälkeen. Jatkuvan käyttöönoton avuksi järjestelmässä on käytössä myös *Fabric*. Sillä on mahdollista suorittaa komentorivikomentoja *SSH*-yhteyden (Secure Shell, salatun verkkoliikenteen protokolla) välityksellä Python-ohjelmointikielellä [41].

Projektissa on havaittavissa myös jatkuvaa testausta. Tutkimushetkellä järjestelmässä oli 1101 yksikkötestiä. Siitä, kuinka hyvin ne kattavat järjestelmän ohjelmakoodin, ei kuitenkaan ole tietoa, sillä järjestelmässä ei ole käytössä tätä mittaavaa työkalua. Järjestelmässä

ei myöskään havaittu olevan esimerkiksi integraatio-, hyväksyntä- tai toimitusputkitestejä. Järjestelmässä on konfigurointi Python-ohjelmointikielen staattisen analyysin *Prospector*-työkalulle, mutta tätä ei ole hyödynnetty sen toimitusputkessa.

Järjestelmän GitLab-projektin ja lähdekoodin perusteella ei havaittu sen hyödyntävän jatkuvaa monitorointia tai suunnittelua. Esimerkiksi GitLab tarjoaa suunnittelun avuksi Issue Boardin, jolla voi hallita projektin ongelmia [59] tai vaatimuksia, mutta tutkimushetkellä järjestelmän taulu oli tyhjä. On kuitenkin mahdollista, ettei järjestelmässä ollut tutkimushetkellä avoimia vaatimuksia tai ongelmia. Toisaalta taulun suljetut-lista oli myös tyhjä.

Tutkittaessa järjestelmän infrastruktuurista ohjelmakoodia havaittiin siinä hyödynnettävän Pythonin *venv*-moduulia virtuaalisten Python-suoritusympäristöjen luomiseen. Tämän riippuvuudet on määritelty `requirements.txt`-tiedostoilla. Virtuaalisessa ympäristössä käytettävä Python-versio riippuu kuitenkin käytettyyn käyttöjärjestelmään asennetusta versiosta eikä sitä ole määritetty ohjelmallisesti tässä projektissa. Lisäksi esimerkiksi *Apache*-verkkopalvelimen säätämiseen on tarjolla esikonfiguroituja tiedostoja.

Järjestelmän suoritusympäristöä ei ole kuitenkaan määritetty ohjelmallisesti tämän tarkemmin. Ohjelmakoodin muodossa ei ole määritetty esimerkiksi, mitä versiota *PostgreSQL*-tietokannasta on tarkoitus järjestelmän tuotantoympäristössä käyttää. Dokumentaation perusteella suositeltava versio on 9.2. Koska suoritusympäristöä ei ole määritetty ohjelmakoodina, voi sen kahdentaminen olla työlästä.

Liittyen suoritusympäristön kahdentamiseen järjestelmässä käytetään kehitysympäristössä eri tietokantaa kuin tuotantoympäristössä: kehitysympäristössä käytössä on SQLite ja tuotantoympäristössä PostgreSQL. Nämä ovat erilaisia, sillä esimerkiksi SQLiten osalta vierasavaimet on erikseen kytkettävä käyttöön jokaisen tietokantayhteyden osalta [96]. Eroavaisuduet kehitys- ja tuotantoympäristöissä saattavat aiheuttaa ongelmia esimerkiksi jatkuvassa toimituksessa ja käyttöönotossa, ja kehittäjien voi olla vaikeaa sopeutua tuotantoympäristöön tilanteen niin vaatiessa [95].

Projektissa oli myös havaittavissa Dockerin hyödyntämistä. Ilmeisesti toimitusputkessa suoritettavien testien nopeuttamiseksi järjestelmää varten on luotu Dockerfile ja sen perusteella levykuva, joka sisältää käytettävän Python-version lisäksi `requirements-test.txt`-tiedoston määrittämät riippuvuudet. Docker ei ole kuitenkaan muutoin projektissa käytössä.

Pystytettäessä kehitysympäristöä on havaittavissa siiheen liittyvässä dokumentaatiossa joitakin puutteita. Ohjeissa käytetty Python-versio 3.3 ei ollut yhteensopiva joidenkin `requirements.txt`-tiedostossa määritettyjen riippuvuuksien suhteen. Käytettäessä esimerkiksi versiota 3.4.9 ongelma poistui. Koska järjestelmä vaatii tietyn Python-version, ja kehittäjän valmiiksi asennettu versio saattaa erota tästä, on kehittäjän käytettävä esimerkiksi *pyenv*-työkalua eri versioiden hallintaan. Se mahdollistaa esimerkiksi globaalin ja tietyn hakemiston käyttämän Python-version määrittämisen [88].

Ohjeissa mainittiin myös, että `submissionsystem`-hakemistossa olisi `develop-data.json`-tiedosto, joka sisältäisi esitietoa, jolla tietokannan voisi alustaa. Kyseistä tiedostoa ei kuitenkaan enää ole projektin versionhallinnassa. Tutkittaessa projektin versiohistoriaa tiedosto oli kuitenkin mahdollista löytää. Se tosin sisälsi vain käyttäjäolioita eikä esimerkiksi kursseja. Käyttäjien luontiin on ohjeistus kehitysympäristön pystytysohjeessa.

Ohjeiden mukaan yksikkötestit suoritettaessa kaksi testiä päättyi virheeseen *mock*-riippuvuuden puuttumisen takia. Vain 1074 testiä 1101:sta suoritettiin. Asentamalla `requirements-test.txt`-tiedoston esittämät riippuvuudet ongelmat korjaantuivat. Vastaavasti suoritettaessa Prospector kehitysympäristössä teki se 1367 huomautusta. Työkalun suorittaminen epäonnistui annettujen ohjeiden mukaan tehtynä, koska kaksi riippuvuutta puuttui. Kyseisten riippuvuuksien asentamisen jälkeen analyysi onnistui.

Kehitysympäristön pystytysohjeistuksesta puuttui GitLab-integraation tekeminen, mutta ohjeistus löytyi järjestelmän asennusohjeista. Kehitysympäristöä varten tarvittiin toimiva Python 3.4.9, Python-pakettienhallintasovellus *pip*, pyvenv-moduuli, GitLab-instanssi ja avain-arvo-tietokanta *Redis*. Koska kehitysympäristössä on käytössä SQLite, vältettiin kehitysympäristön pystytyksessä tietokantaserverin konfiguroinnilta. Tämä siksi, että SQLite käyttää tiedostoja tiedon tallettamiseen [10].

5.1.2 A+

A+ on avoimen lähdekoodin palvelusuuntautuneeseen arkkitehtuuriin perustuva oppimisenhallintajärjestelmä, johon on mahdollista yhdistää lisäosia [66]. A+:n käyttämien palveluiden avulla opettajien on mahdollista muunnella verkko-opetusmateriaaliansa ja yhdistää niitä joustavasti [3]. A+-järjestelmässä voi olla opiskelijan suoritettavaksi tarkoitettuja tehtäviä, jotka voidaan arvioida synkronisesti, asynkronisesti tai staattisesti: synkronisessa ja asynkronisessa arvioinnissa arvioinnin suorittaa tehtävään liittyvä palvelu ja staattisessa arvioinnissa opettaja [66].

Järjestelmää ovat kehittäneet Aalto-yliopiston ja Tampereen teknillisen yliopiston henkilöstö, se soveltuu esimerkiksi tietojenkäsittelytieteiden opetukseen ja se on käytössä Aalto-yliopistolla ja Tampereen teknillisellä yliopistolla [3]. A+:n GitHub-etätietovarasto on julkinen ja siihen on mahdollista tehdä vetopyyntöjä. Järjestelmä säilyy käytössä Tampereen teknillisen yliopiston opetussuunnitelman mukaisilla kursseilla Tampereen yliopistossa vuoden 2019 alussa ja uusi asennus tehdään myöhemmin.

Tutustuttaessa A+:n GitHub-etätietovarastoon on havaittavissa, että sen kehityksessä on hyödynnetty jatkuvaa integraatiota. Projektissa on siihen käytössä Jenkins, jonka suorittamien testien tulokset sekä itse järjestelmän testisivusto ovat julkisia [7]. Tutkimushetkellä testisivusto ei kuitenkaan toiminut vaan palautti `Bad Gateway` -virheen. Tämä virhe viittaa siihen, että yhdysväylänä tai välityspalvelimena toiminut verkkopalvelin sai sen takana olevalta palvelimelta epäkelvon vastauksen [51].

Tutkittaessa projektin vetopyyntöjä oli huomattavissa, että sellaista ohjelmakoodia, jonka testit ovat epäonnistuneet, on yhdistetty `master`-haaraan [87]. Kyseinen haara on A+-projektin oletushaara [20]. Projektissa on kuitenkin myös erikseen `production`-haara [20], joka on luultavasti tarkoitettu ohjelmakoodille, joka on hyväksytty otettavaksi käyttöön järjestelmän tuotantoympäristössä.

Järjestelmän viimeisin julkaistu versio on v1.3 [89], jossa on kuitenkin GitHub-etätietovaraston perusteella sellaista ohjelmakoodia, jonka testit ovat epäonnistuneet Jenkinsissa [4][36]. On kuitenkin huomioitava, että nämä ongelmat on saatettu korjata myöhemmin pikakorjauksina, ja että Jenkinsin järjestelmän testauskonfiguraatiossa saattaa olla ongelmia. Tutkimushetkellä avoimena olevassa vetopyynnössä `#374` on mainittu, että Selenium-testit ovat rikki [12]. Testien epäonnistumiset eivät siis välttämättä johdu ongelmista ohjelmakoodissa.

Projektissa on myös havaittavissa jatkuvaa testausta. Järjestelmässä on 125 Python-yksikkötestiä ja 23 Selenium-verkkoselaintestiä. Jälkimmäisten voidaan ajatella olevan järjestelmätestejä, sillä ne testaavat koko A+-järjestelmää verkkoselaimen kautta. Projektissa ei vaikuta olevan käytössä työkalua, joka mittaisi testien ohjelmakoodin kattavuuden, joten tämä jää epäselväksi. Myöskään esimerkiksi ohjelmakoodin staattiseen analyysiin ei vaikuta olevan käytössä työkalua.

Projekti hyödyntää jatkuvaa toimitusta ja käyttöönottoa siltä osin, että Jenkins julkaisee `master`-haaran testisivustolle [5]. Kuten edellä mainitaan testisivusto ei kuitenkaan vaikuttanut toimivan tutkimushetkellä. Projekti ei varsinaisesti kuitenkaan hyödynnä jatkuvaa toimitusta ja käyttöönottoa, sillä siinä ei ole havaittavissa automatisointia asennuspakettien tuotantoympäristöön toimittamiseen ja siellä käyttöönottoon.

Projektissa ei ole myöskään viitteitä siitä, että siinä hyödynnettäisiin jatkuvaa monitorointia. Jatkuvaa suunnittelua hyödynnetään mahdollisesti siten, että projektissa on avoimille ongelmille kirjaus, ja nämä voidaan myös halutessa nimikoida, josta esimerkki on ”bug” (bugi) [60]. Toisaalta projektin GitHub-etätietovaraston ”Projects”-näkymä, jossa on mahdollista esimerkiksi järjestellä tehtäviä sekä seurata edistymistä, on tyhjä [84]. Projektissa on myös käytössä Slack-pikaviestintiimi sekä *Google Groups* -sähköpostilista [3], joita saatetaan käyttää apuna jatkuvassa suunnittelussa.

Infrastruktuurisena ohjelmakoodina järjestelmässä on määritetty Python-työkalu *virtualenv*:n avulla luotavan eristetyn suoritusympäristön riippuvuudet `requirements.txt`-tiedostona. Suoritusympäristöä ei ole kuitenkaan määritetty ohjelmallisesti järjestelmässä tämän tarkemmin. Esimerkiksi käytettävää Python-versiota ei ole määritetty ohjelmallisesti. Kuten luvussa 5.1.1 esitetään virtuaalisen suoritusympäristön Python-versio riippuu käyttöjärjestelmään asennetuista Python-versioista. Versioita voi kuitenkin hallita pyenv:n avulla [88]. Myös Jenkinsin tehtäväkuvaukset on jaettu *XML*-muodossa, ja GitHub-etätietovarastossa ovat myös tarjolla *Shibboleth*:n konfiguraatiotiedostot.

Muilta osin järjestelmän infrastruktuuria ei vaikuta olevan määritetty ohjelmallisesti. Järjestelmän kehitysympäristö ei myöskään täysin vastaa sen tuotantoympäristöä, sillä esimerkiksi ensimmäisessä käytetään SQLite-tietokantaa ja jälkimmäisessä PostgreSQL-tietokantaa. Kuten luvussa 5.1.1 mainitaan eroavaisuudet suoritusympäristöissä saattavat aiheuttaa ongelmia.

Pystytyttäessä kehitysympäristöä on havaittavissa lieviä ongelmia. Järjestelmän käyttöönotto-ohjeistuksessa mainitaan, että se olisi riippuvuuksien takia pakotettu käyttämään Python-versiota 3.4.3 ja *Django*-versiota 1.7 [6], mutta toisaalta `requirements.txt`-tiedosto määrittää Djangon versioksi suuremman tai yhtäsuuren kuin 1.10.8 ja pienemmän kuin 1.11 [8], jotka eivät kehitysympäristön luontivaiheessa olleet yhteensopivia pienemmän kuin Python-versio 3.5:n kanssa. Kehitysympäristön asennusohjeissa mainitaan tarvittavaksi Python-versioksi 3.4+ ja komentoesimerkeissä on käytetty versiota 3.4.3 [8].

Kun käytettäväksi Python-versioksi valittiin 3.5.6, järjestelmä saatiin asennettua `doc`-hakemiston `create_test_environment.sh` -komentoriviskriptin avulla. Järjestelmän käynnistys antoi virheilmoituksen puuttuneesta `BASE_URL`-asetuksesta, josta ei ollut mainintaa asennus- tai kehitysympäristöohjeissa, mutta johon oli mahdollista löytää esimerkki esimerkkiasetustiedostosta `aplus/local_settings.example.py`. `doc/example_grader.py`-tiedoston esimerkkiarviointiohjelma käynnistyi annettujen ohjeiden mukaan. Python-version 3.5.6 lisäksi kehitysympäristön pystytykseen vaadittiin Python-pakettienhallintasovellus pip ja virtualenv moduuli.

Suoritettaessa yksikkötestit yksi testi 125:stä päättyi virheeseen, sillä Python-tulkki ei pystynyt kirjoittamaan tiedostoa `test.png` hakemistoon `media`, koska kyseistä hakemistoa ei ollut versionhallinnassa. Lisäämällä kyseinen hakemisto onnistuivat kaikki 125 yksikkötestiä. Selenium-testien riippuvuuksien asennusohjeet olivat hieman puutteelliset, mutta puuttunut *geckodriver*-riippuvuus asentamalla 23:stä testistä 22 onnistui. Yksi epäonnistui, koska oletettu arvo ei vastannut vastaanotettua arvoa.

5.1.3 Versionhallinta Gitillä

Versionhallinta Gitillä on verkkokurssi Tampereen teknillisellä yliopistolla käytössä olevassa A+-järjestelmässä. Sen avulla opiskelijan on mahdollista opetella Gitin ominaisuuksia eri vaatimustasoilla. Kurssi sisältää sekä opetusmateriaalia että tehtäviä siihen perustuen.

Kyseinen kurssi ei varsinaisesti ole oma opetusjärjestelmänsä, mutta se saattaa tarjota tietoa siitä, millaista A+:n kurssimateriaalia ja -tehtäviä on kehittää ja ylläpitää. Projektissa on myös sellaisia vaiheita, esimerkiksi sisällön julkaisu, johon voisi teoriassa olla mahdollista hyödyntää DevOps-käytäntöjä.

Tutustuttaessa kurssin GitLab-etätietovarastoon havaittiin, että projektissa on hyödynnetty infrastruktuurista ohjelmakoodia jossain määrin. Esimerkiksi ilmeisesti sisällön luomiseen tai virheenjäljitykseen tarvittava ympäristö on määritetty Python-riippuvuudet

esittävällä `requirements.txt`-tiedostolla. Ympäristön virtualisointiin hyödynnetään virtualenv-moduulia. Järjestelmän suoritusympäristöä ei ole kuitenkaan määritetty ohjelmallisesti tämän tarkemmin. Esimerkiksi ympäristön Python-versiota ei ole määritetty ohjelmallisesti vaan se riippuu käyttöjärjestelmään asennetuista versioista.

Kurssin sisällön testaamiseen paikallisessa kehitysympäristössä tarvitaan toimiva A+-asennus. Tätä varten projektissa on määritys *Docker Compose* -työkalulle. Se on työkalu useamman Docker-kontin sisältämän sovelluksen määrittämiseen ja suorittamiseen [80]. Docker Compose -konfiguraation lisäksi projektissa on komentoriviskripti, jolla tarvittavat Docker-kontit voi käynnistää. Projektissa ei havaittu muita DevOps-käytäntöjä infrastruktuurisen ohjelmakoodin lisäksi.

Projektin ohjeet järjestelmän kehitysympäristön pystyttämiselle ovat hieman ongelmalliset, sillä oletushaara `master`in ohjeilla julkaisuhaaraan `html-files-in-git` siirtyminen ei onnistunut, sillä Git jätti siirtymisen suorittamatta seuraamattomien tiedostojen ylikirjoittumisen takia. Jos Git-alamoduuleja ei alustanut ennen haaran vaihtoa, vaihto onnistui. `html-files-in-git`-haara sisältää ohjeet ympäristön pystyttämiseen ja kurssin materiaalin tarkasteluun. Materiaalin tarkastelu verkkoselaimella onnistui myös ilman virtualenvilla luotua Python-ympäristöä. Siihen tarvittiin Docker ja Docker Compose sekä `docker-up.sh`-skriptin suorittaminen.

Kurssin sisällön kääntämisen on tarjottu komentoriviskripti. Kääntäminen onnistui `master`-haarassa, mutta ei `html-files-in-git`-haarassa, johon ohjeistetaan siirtymään ennen käännöksen suorittamista. Jos kyseisen haaran ohjeistuksella luotu virtuaalinen Python-ympäristö oli aktivoituna, saatiin virheilmoitus, ettei `aplus_setup`-moduulin lataaminen onnistunut. Jos virtuaalinen ympäristö ei ollut aktivoituna, saatiin virheilmoitus `sphinx-build`-komennon puuttumisesta.

5.1.4 prplatform

prplatform on avoimen lähdekoodin web-sovellus opiskelijoiden opintosuoritteiden vertaisarviointiin esimerkiksi yliopistojen kursseille. Opettaja voi luoda järjestelmään tehtäviä, opiskelijat voivat tehdä näihin palautuksia, ja opettajat voivat arvioida näitä. prplatform tukee myös opiskelijoiden välistä vertaisarviointia sekä opiskelijaryhmiä. Se on myös mahdollista integroida alaluvussa 5.1.2 esitettyyn A+-järjestelmään. [86]

Tutustuttaessa prplatformin GitHub-etätietovarastoon havaittiin, että järjestelmässä on hyödynnetty infrastruktuurista ohjelmakoodia. Järjestelmässä on esimerkiksi erilliset Dockerfilet ja Docker Compose -konfiguraatiot kehitys- ja tuotantoympäristöille. Myös Python-riippuvuudet ja Django-sovellukseen liittyvät asetukset ovat määriteltynä erikseen kehitys-, testi- ja tuotantoympäristöille. Lisäksi tuotantoympäristöön on määritettynä esimerkiksi PostgreSQL-ylläpitotehtäviä.

Järjestelmässä on myös käytössä sama tietokanta, PostgreSQL:n versio 10.3, kehitys- ja tuotantoympäristöissä. Tuotantoympäristössä ovat myös käytössä Redis ja *Caddy*. Caddy

on *HTTP*-palvelinsovellus (Hypertext Transfer Protocol, hypertekstin siirtoprotokolla), joka automaattisesti ottaa *HTTPS*:n (Hypertext Transfer Protocol Secure, HTTP ja TLS-protokollan yhdistelmä) käyttöön verkkosivustolla [16]. Redis ja Caddy eivät ole käytössä kehitysympäristössä. Caddyn käyttö ei kuitenkaan välttämättä ole perusteltua kehitysympäristössä. Tämä siksi, että isäntänimi ei saa olla esimerkiksi `localhost` tai IP-osoite [16].

Projektissa vaikuttaa myös olevan hyödynnetty jatkuvaa suunnittelua jossain määrin, sillä sen GitHub-etätietovaraston ”Projects”-näkymässä on taulu, jossa on listat tekemättömille ja tehdyille tehtäville [85]. Kyseistä taulua on kuitenkin päivitetty viimeksi 4. heinäkuuta 2018 [85], vaikka järjestelmään on tehty committeja tämän ajankohdan jälkeenkin [22].

Järjestelmässä on myös testejä, mutta viitteitä jatkuvasta, automatisoidusta, testauksesta ei projektin GitHub-etätietovaraston perusteella havaittu. Järjestelmän kehittäjä kertoo kuitenkin, että testit suoritetaan Gitin *pre-commit*-koukussa, ja ettei committia hyväksytä, jos testit epäonnistuvat. Git suorittaa kyseisen koukun ennen kuin commit tehdään [43]. Projektin Git-tietovarastossa ei kuitenkaan ole näitä koukkuja. Tämä siksi, että ne ovat asiakaspuolen koukkuja, jotka eivät kopioidu, kun Git-tietovarasto kloonataan [43].

Muita DevOps-käytäntöjä ei havaittu projektissa. Pystytettäessä kehitysympäristöä huomattiin, että järjestelmän dokumentaatio on puutteellinen tältä osin. Koska Docker ja Docker Compose ovat tuttuja työkaluja työn tekijälle, oli hänen mahdollista näihin liittyvän tietämyksen sekä tietovarastosta löytyvien `local.yml` ja `production.yml`-tiedostojen perusteella päätellä, että järjestelmän suoritusympäristöjen suorittamiseen käytettäisiin edellä mainittuja työkaluja.

Kehitysympäristön pystyttäminen oli edellä mainittu huomioon ottaen kokeilemista. Ympäristön pystyttäminen estyi ensin riippuvuuksien, jotka eivät olleet yhteensopivia keskenään, takia. Ongelma poistui, kun `compose/local/django/Dockerfile`-tiedostosta riviltä 12 poistettiin paketti *openssl-dev*, jolloin sitä ei asenneta käyttöjärjestelmään, johon kyseinen Docker-levykuva perustuu. On kuitenkin mahdollista, että tämä aiheuttaa ongelmia järjestelmän joissakin käyttötilanteissa.

Seuraava ongelma kehitysympäristön pystytyksessä oli ympäristömuuttujatiedostojen puuttuminen. Näistä ei ollut mainintaa järjestelmän dokumentaatiossa. Kun puuttuneet tiedostot lisättiin, kehitysympäristön pystyttäminen eteni. Tämän jälkeen ongelmana olivat kuitenkin puuttuneet ympäristömuuttujat. Tarvittavia ympäristömuuttujia ei oltu dokumentoitu. Lisäämällä nämä muuttujat ympäristömuuttujatiedostoihin oli kehitysympäristö mahdollista saada toimimaan komennolla

```
docker-compose -f local.yml up -d,
```
(5.1)

jossa *f*-parametrilla syötetään Docker-Compose-konfiguraation sisältävän tiedoston polku, ja jossa *d*-lipulla järjestelmänä liittyvät kontit on mahdollista käynnistää taustalle suoritukseen. Kehitysympäristön luomiseen tarvittiin Docker ja Docker Compose.

Järjestelmään piti vielä luoda pääkäyttäjä komennolla

```
python manage.py createsuperuser
```
(5.2)

Docker-kontin sisällä, jolloin ongelmaksi ilmeni ympäristöstä puuttuva `DATABASE_URL`-muuttuja. Suorittamalla hakemiston `compose/production/django/` tiedoston `entrypoint.sh` rivin 17 komento ennen pääkäyttäjän luontia oli käyttäjä mahdollista luoda onnistuneesti.

Myös järjestelmän testaaminen kehitysympäristössä oli hieman epäselvää. Koska järjestelmää suoritettaisiin Docker-konteissa, ei järjestelmän `README.rst`-tiedoston testausohjeiden komentoja voisi suoraan suorittaa komentorivikehotteella projektin paikallisessa textttGit-tietovarastossa.

Testit oli kuitenkin mahdollista suorittaa Docker-kontissa avaamalla ensin *sh*-komentorivikehote kontissa komennolla

```
docker exec -i -t <kontin nimi> sh,
```
(5.3)

jossa *exec* on Docker-komento komentojen suorittamiseen kontissa, $-i$ lippu, joka pitää standardisyötteen auki, vaikka kontti ei olisi kiinnitetyssä tilassa, $-t$ lippu, jolla saadaan pseudonyymi komentorivikehote käyttöön, <*kontin nimi*> korvattuna prplatformin sisältävän kontin nimellä, ja *sh* komento, joka halutaan suorittaa kyseisessä kontissa.

Tämän jälkeen suorittamalla tiedoston `README.rst` sisältämä `py.test`-komento, testit suoritettiin. Testejä suoritettiin neljäkymmentä, joista yksi epäonnistui. Suorittamalla testit

```
python manage.py test
```
(5.4)

-komennolla kaikki neljäkymmentä testiä suoritettiin onnistuneesti. Projektissa on myös työkalu testien kattavuuden mittaamiseen. Kehitysympäristössä kattavuudeksi mittautui 71 prosenttia.

Taulukko 5.1 tekee yhteenvedon opetusjärjestelmissä hyödynnetyistä DevOps-käytännöistä omatoimisen tutustumisen perusteella. ✓-merkki tarkoittaa, että kyseistä käytäntöä on hyödynnetty järjestelmässä, ja vastaavasti x, että tätä käytäntöä ei ole hyödynnetty.

Taulukko 5.2 tekee yhteenvedon DevOps-käytäntöjen hyödyntämiseen käytetyistä työkaluista opetusjärjestelmäkohtaisesti omatoimisen tutustumisen perusteella. Merkki - tarkoittaa, ettei järjestelmä hyödynnä kyseistä käytäntöä tai ettei sen hyödyntämiseen käytettyjä työkaluja tunnistettu. Suluissa oleva solun arvo ilmaisee, että opetusjärjestelmä hyödyntää joltain osin, mutta ei täysin, kyseessä olevaa käytäntöä.

Taulukko 5.1. *DevOps-käytäntöjen hyödyntäminen opetusjärjestelmissä omatoimisen tutustumisen perusteella.*

Käytäntö	Repolainen	A+	Versionhallinta Gitillä	prplatform
Jatkuva integraatio	✓	✓	x	x
Jatkuva testaus	✓	✓	x	x
Jatkuva toimitus	✓	x	x	x
Jatkuva käyttöönotto	✓	x	x	x
Jatkuva monitorointi	x	x	x	x
Jatkuva suunnittelu	x	✓	x	✓
Infrastruktuuri ohjelmakoodina	x	x	x	✓

Taulukko 5.2. *DevOps-käytäntöjen hyödyntämiseen käytetyt työkalut opetusjärjestelmäkohtaisesti omatoimisen tutustumisen perusteella.*

Käytäntö	Repolainen	A+	Versionhallinta Gitillä	prplatform
Jatkuva integraatio	GitLab CI/CD	Jenkins	-	-
Jatkuva testaus	GitLab CI/CD, Docker, Python: unittest + django.test	Jenkins, Selenium, Python: unittest + django.test	-	-
Jatkuva toimitus	GitLab CI/CD	-	-	-
Jatkuva käyttöönotto	GitLab CI/CD, Fabric	-	-	-
Jatkuva monitorointi	-	-	-	-
Jatkuva suunnittelu	-	GitHub: Issues, Slack, Google Groups	-	GitHub: Projects
Infrastruktuuri ohjelmakoodina	(Python: venv)	(Python: virtualenv)	(Docker, Docker Compose, Python: virtualenv)	Docker, Docker Compose

5.2 Kysely

Verkkokysely suoritettiin Webropol-kyselypalvelussa. Kyselyn kysymykset ovat luettavissa liitteestä A. Kyselyn mahdolliset vastaanottajat selvisivät opetusjärjestelmiin liittyvien GitLab ja GitHub-versionhallintaprojektien jäsenlistausten perusteella. Tämän selvityksen perusteeella tunnistettiin 12 henkilöä, jotka voisivat liittyä opetusjärjestelmien kehitykseen tai ylläpitoon jollakin tavalla. Näiltä kaikilta kysyttiin sähköpostilla halukkuutta vastata kyselyyn. Näistä 12:sta henkilöstä neljä henkilöä olivat halukkaita vastaamaan kyselyyn.

Tämän lisäksi erästä henkilöä lähestyttiin erikseen, koska työn tekijä sai tietää, että kyseinen henkilö voisi mahdollisesti olla erään tässä työssä esiintymättömän järjestelmän vastuuhenkilö tai ylläpitäjä. Lisäksi tässä työssä esiintyneiden järjestelmien GitLab ja GitHub-etätietovarastoprojektien jäsenet tarkastettiin uudestaan torstaina 13. päivä joulukuuta 2018 ja tämän perusteella yhtä henkilöä lähestyttiin sähköpostilla kyselyyn liittyen.

Nämä edellä mainitut kaksi henkilöä eivät kuitenkaan osoittaneet kiinnostusta kyselyyn vastaamiseen.

Myös Tampereen teknillisen yliopiston IT-helpdeskiltä kysyttiin sähköpostilla, onko joku IT-palveluiden työntekijä tai jotkut työntekijät vastuussa joidenkin tässä työssä esiintyneiden opetusjärjestelmien ylläpidosta, ja kysyttiin halukkuutta vastata kyselyyn. IT-helpdeskiltä saadun tiedon mukaan yliopiston IT-palvelut ei kuitenkaan vastaa tässä työssä esiintyneiden opetusjärjestelmien ylläpidosta, joten kyselylinkkiä ei toimitettu kyseiseen yksikköön.

Webropol-verkkokysely avattiin perjantaina 23. päivä marraskuuta 2018 kello 12:23, kun ensimmäinen kyselylinkki lähetettiin vastaajalle. Kysely sulkeutui automatisoidusti perjantaina 21. päivä joulukuuta 2018 kello 12:00 eli se oli auki 23 minuuttia alle neljä viikkoa. Kyselyyn kertyi yhteensä kolme vastausta. Tämä on 75 prosenttia kaikista kyselylinkin saaneista henkilöistä ja noin 21,43 prosenttia kaikista niistä, joilta kysyttiin halukkuutta kyselyyn vastaamiseen. Vastauksien jakautuminen opetusjärjestelmiä kohden on esitetty taulukossa 5.3.

Taulukko 5.3. *Kyselyn vastaukset opetusjärjestelmää kohden.*

Järjestelmä	**Määrä**	**Prosenttiosuus (%)**
Repolainen	1	33,33
A+	1	33,33
Versionhallinta Gitillä	1	33,33
prplatform	0	0,00

Kyselyyn vastanneista henkilöistä yksi oli halukas varaamaan keskuteluhetken työn tekijän kanssa. Kyseistä henkilöä lähestyttiin luvussa 3 esitetystä tutkimusmenetelmästä poiketen Slackin avulla. Keskustelua varten työn tekijä valmisteli tämän omatoimisen tutustumisen sekä edellä mainitun henkilön kyselyvastausten perusteelta viisi kysymystä. Keskustelua ei äänitetty, mutta sen aikana työn tekijä kirjasi haastateltavan vastaukset paperille.

Repolaisen osalta kysely saavutti koko kyseisen järjestelmän tiimin, sillä kyselyn vastausten perusteella tiimin koko on yksi henkilö, ja kyselyyn vastasi yksi Repolaiseen liittyvä henkilö. A+:n osalta kyselyn kattavuus jäi auki. Kyselyyn vastannut A+:ssa mukana oleva henkilö kertoi, että kyseistä järjestelmää kehitetään Aalto-yliopistolla eikä ohjelmistokehitystä ole juurikaan tehty Tampereen teknillisellä yliopistolla.

Versionhallinta Gitillä -kurssin tiimin koko jäi epäselväksi, sillä kyselyyn vastannut henkilö kertoi tiimin kattavan kaikki Gittia kursseillaan käyttävät opettajat. Projektissa ovat hänen mukaansa GitLabin osalta mukana myös yksi tutkimusapulainen sekä yliopiston IT-palvelut. Kuten taulukko 5.3 osoittaa prplatformin osalta kyselyyn ei kertynyt vastauksia, joten siihen liittyvän tiimin koosta ei saatu tietoa. Taulukko 5.4 esittää vastausprosentit opetusjärjestelmien tiimien kokoja kohden. Merkillä - tarkoitetaan, ettei kyseinen tieto selvinnyt kyselyn avulla.

Taulukko 5.4. *Kyselyn vastausten määrä suhteessa opetusjärjestelmän tiimin kokoon.*

Järjestelmä	**Vastausten määrä / tiimin koko**	**Vastausprosentti (%)**
Repolainen	1/1	100,00
A+	1/-	-
Versionhallinta Gitillä	1/-	-
prplatform	0/0	-

Vastaavasti taulukko 5.5 esittää jokaista opetusjärjestelmää kohden roolit, jotka kysely kattoi. Siinä ✓-merkki tarkoittaa, että kyseisellä roolilla järjestelmässä operoiva henkilö vastasi kyselyyn. Vastaavasti - tarkoittaa, ettei vastaajilla ollut kyseistä roolia.

Taulukko 5.5. *Kyselyyn vastanneet roolit opetusjärjestelmää kohden.*

Järjestelmä	**Ohjelmisto-suunnittelija**	**Ohjelmisto-kehittäjä**	**Testaaja / laadun-valvoja**	**Ylläpitäjä**	**Vastuu-henkilö**
Repolainen	-	-	-	✓	✓
A+	-	✓	-	✓	✓
Versionhallinta Gitillä	-	-	-	-	✓
prplatform	-	-	-	-	-

Repolaisen osalta kyselyyn vastannut henkilö vastaa kyseisen järjestelmän ylläpidosta ja yliopiston IT-palvelut sen tietokoneen, jolla Repolaista suoritetaan, ylläpidosta. A+:aa ylläpitää kyselyyn sen osalta vastannut henkilö. Versionhallinta Gitillä -kurssia ylläpitää kyselyn perusteella yliopiston IT-palvelut. A+:n osalta keskustelua ohjelmistokehittäjien ja ylläpitäjien välillä käydään Slackin avulla. Kyselyyn vastanneen henkilön mukaan kommunikointi toimii riittävän hyvin. Muiden järjestelmien osalta tietoa keskustelun muodosta tai tasosta ei saatu.

Järjestelmien ylläpidossa on havaittavissa ongelmia. Esimerkiksi A+:n dokumentaatio on vanhentunut. Versionhallinta Gitillä -kurssilla ongelmana on, miten todentaa sen uusin versio, kun samaan aikaan kurssin pitäisi toimia koko ajan ja materiaalin pitäisi olla ajantasalla. Repolaisen osalta GitLabin päivitykset ovat aiheuttaneet ongelmia, sillä sen rajapinnan toimintoja on uusien versioiden myötä poistettu ja Repolainen on ollut riippuvainen näistä.

Repolaisessa on ongelmia myös sen kehityksen näkökulmasta. Projektissa ei ole ohjelmistokehittäjiä, minkä vastaaja kokee suureksi ongelmaksi. Kysyttäessä häneltä, mitä ongelman ratkaisemiseksi on tehty, kertoi hän, että Tampereen TietoTeekkarikillalle on hänen tietääkseen välitetty rekrytointi-ilmoitus, ja että Tietotekniikan laboratorio kykeni allokoimaan vähän erään työntekijän työaikaa Repolaiseen. Kyselyyn vastanneen henkilön mukaan tämä aikamäärä on kuitenkin liian vähän ja hänen tietääkseen myös Repolaiseen allokoidun työntekijän työsopimus päättyi vuoden 2018 loppuun.

Repolaisen julkaisuketjussa uusi versio viedään automatisoidusti ensin testiympäristöön ja tämän jälkeen loppukäyttäjien käyttöön tuotantoympäristöön. A+:n julkaisuketju ei ole

automatisoitu. Molemmissa järjestelmissä edellisen toimivan version palautus on tehtävä manuaalisesti.

Versionhallinta Gitillä -kurssin ja prplatformin julkaisuketjuista ei saatu kyselyn avulla tietoa. Selvitettäessä mahdollisia kandidaatteja tutkittaviksi järjestelmiksi prplatformin kehittäjältä saatiin kuitenkin tieto, että kyseisen järjestelmän käyttöönottoprosessi on manuaalinen ja sisältää SSH:lla tuotantoympäristöön kirjautumisen sekä `git pull` -komennolla uuden version hakemisen.

Omatoimisessa tutustumisessa havaittiin, ettei Repolaisen testiympäristö toimi. Kyselyvastausten seurauksena pidetty keskustelu Repolaisen osalta vastanneen henkilön kanssa kuitenkin selvensi, että pääsy kyseiseen ympäristöön on vain tietystä IP-osoitteesta. Samalla selvisi, että IT-palvelut vastaa virtuaalikoneen porttien hallinnoinnista sekä verkosta tulevien yhteyksien sallimisesta eli IP-osoitteiden asettamisesta valkoiselle listalle. Kyseinen henkilö piti kuitenkin mahdollisena, että testiympäristön versio on kaatunut ja kertoi kyselyvastauksessaan, ettei kyseistä suoritusympäristöä juurikaan hyödynnetä nykyisin.

Repolaisen osalta automatisoitu julkaisuketju on kyselyyn vastanneen henkilön mukaan toimiva, sillä se on vähentänyt käsin tehtävän työn määrää. Myös A+:n julkaisuketju on kyselyyn vastanneen henkilön mukaan toimiva, sillä hän ei halua mitään siirrettävän tuotantoympäristöön automaattisesti, koska muut henkilöt kuin hän itse kehittävät järjestelmää.

Repolaisen osalta kysely osoittaa, että kyseisessä järjestelmässä hyödynnetään jatkuvaa integraatiota, testausta, toimitusta ja käyttöönottoa. Kyselyn tulosten perusteella A+:ssa hyödynnetään jatkuvaa testausta ja suunnittelua. Versionhallinta Gitillä -kurssin osalta vastannut henkilö ei osannut sanoa, hyödynnetäänkö siinä jotakin kysytyistä DevOps-käytännöistä.

Kysytyt käytännöt ja niiden hyödyntämisen yhteenveto ovat nähtävissä taulukossa 5.6. Siinä ✓-merkki tarkoittaa, että järjestelmä hyödyntää kyseistä käytäntöä, x, ettei käytäntöä hyödynnetä, ja -, ettei tietoa hyödyntämisestä saatu kyselyn avulla.

Taulukko 5.6. *DevOps-käytäntöjen hyödyntäminen opetusjärjestelmissä kyselyn perusteella.*

Käytäntö	**Repolainen**	**A+**	**Versionhallinta Gitillä**	**prplatform**
Jatkuva integraatio	✓	x	-	-
Jatkuva testaus	✓	✓	-	-
Jatkuva toimitus	✓	x	-	-
Jatkuva käyttöönotto	✓	x	-	-
Jatkuva monitorointi	x	x	-	-
Jatkuva suunnittelu	x	✓	-	-
Infrastruktuuri ohjelmakoodina	x	x	-	-

Repolaisessa jatkuvaan integraatioon, testaukseen, toimitukseen ja käyttöönottoon käytetään kyselyn perusteella GitLab CI/CD:ta ja Dockeria. Jatkuvassa testauksessa on käy-

tössä myös SonarQube ohjelmakoodin staattiseen analyysiin. Jatkuvaan käyttöönottoon hyödynnetään lisäksi Python-kirjasto Fabricia.

A+:ssa käytetään kyselyn perusteella CircleCI:ta jatkuvaan integraatioon. Järjestelmän kehitysympäristö hyödyntää Dockeria, vaikka vastausten perusteella A+:ssa ei kuitenkaan hyödynnetä infrasktruktuurista ohjelmakoodia. Versionhallinta Gitillä -kurssin ja prplatformin osalta ei saatu tietoa myöskään käytäntöihin käytetyistä työkaluista.

Repolaista testataan kyselyn perusteella yksikkötestien ja ohjelmakoodin staattisen analyysin avulla. A+:aa testataan vastaavasti yksikkötestein ja päästä päähän -testein. Näiden osalta vastaaja on kuitenkin hieman epävarma. Päästä päähän -testit ovat testejä, jotka testaavat järjestelmän toimintaa kaikkine integraatioineen todellisessa suoritusympäristössä [106]. Versionhallinta Gitillä -kurssin ja prplatformin osalta tietoa järjestelmän testauksen tasosta ei saatu.

Taulukko 5.7 esittää yhteenvetona opetusjärjestelmäkohtaisesti työkalut, joilla niissä hyödynnetään kutakin kyselyssä kysyttyä käytäntöä. Merkki - tarkoittaa, että käytetyistä työkaluista ei saatu tietoa kyselyn avulla. Suluissa oleva solun arvo ilmaisee, että opetusjärjestelmä hyödyntää joltain osin, mutta ei täysin, kyseessä olevaa käytäntöä. Selkeyden vuoksi Versionhallinta Gitillä -kurssin ja prplatformin tiedot eivät ole taulukossa, koska niitä ei saatu.

***Taulukko 5.7**. DevOps-käytäntöjen hyödyntämiseen käytetyt työkalut opetusjärjestelmäkohtaisesti kyselyn perusteella.*

Käytäntö	**Repolainen**	**A+**
Jatkuva integraatio	GitLab CI/CD, Docker	-
Jatkuva testaus	GitLab CI/CD, Docker, SonarQube	CircleCI
Jatkuva toimitus	GitLab CI/CD, Docker	-
Jatkuva käyttöönotto	GitLab CI/CD, Docker, Fabric	-
Jatkuva monitorointi	-	-
Jatkuva suunnittelu	-	-
Infrastruktuuri ohjelmakoodina	-	(Docker)

5.3 Vertailu

Kyselyvastausten perusteella vertailua omatoimiseen tutustumiseen on mahdollista suorittaa Repolaisen, A+:n ja Versionhallinta Gitillä -kurssin osalta. Kuten taulukko 5.3 osoittaa prplatformin osalta ei saatu vastauksia, joten vertailu ei ole mahdollista. Myös Versionhallinta Gitillä -kurssin osalta vertailu jää kevyeksi, sillä tietoa DevOps-käytäntöjen hyödyntämisestä siinä ei saatu kuten taulukko 5.6 osoittaa.

Repolaisen osalta sekä omatoimisessa tutustumisessa että kyselyn vastauksissa on havaittavissa samat hyödynnetyt DevOps-käytännöt. Nämä ovat jatkuva integraatio, testaus, toimitus ja käyttöönotto. Omatoimisen tutustumisen perusteella järjestelmä hyödyntää jossain määrin infrastruktuurista ohjelmakoodia, mutta samaa havaintoa ei ole mahdollista

tehdä kyselyn perusteella. Toisaalta kyseisen käytännön hyödyntäminen on omatoimisen tutustumisen perusteellakin vain osittaista.

Taulukko 5.8 tekee yhteenvedon Repolaisessa hyödynnetyistä DevOps-käytännöistä omatoimisen tutustumisen ja kyselyn perusteella. ✓-merkki ilmaisee, että järjestelmä hyödyntää kyseistä käytäntöä, ja x-merkki, ettei se hyödynnä sitä.

Taulukko 5.8. *DevOps-käytäntöjen hyödyntäminen Repolaisessa omatoimisen tutustumisen ja kyselyn perusteella.*

Käytäntö	**Omatoiminen tutustuminen**	**Kysely**
Jatkuva integraatio	✓	✓
Jatkuva testaus	✓	✓
Jatkuva toimitus	✓	✓
Jatkuva käyttöönotto	✓	✓
Jatkuva monitorointi	x	x
Jatkuva suunnittelu	x	x
Infrastruktuuri ohjelmakoodina	x	x

Käytäntöihin käytetyissä työkaluissa on puolestaan havaittavissa eroa. Kyselyn perusteella Repolaisessa hyödynnetään Dockeria jatkuvaan integraatioon, toimitukseen ja käyttöönottoon, mutta samaa havaintoa ei tehty omatoimisessa tutustumisessa. Tutkimalla järjestelmään liittyvää GitLab-etätietovarastoa, on sen CI/CD-putkien lokeista mahdollista havaita teksti *"Using Docker executor with image ..."*, mikä viitannee siihen, että GitLab CI/CD suorittaa Repolaiseen liittyvät tehtävät Dockerin avulla. Tämä selittänee havaitun eron.

Myös jatkuvan testauksen osalta on havaittavissa eroa. Omatoimisessa tutustumisessa Repolaisessa havaittiin yksikkötestien suorittaminen osana jatkuvaa integraatiota. Yksikkötesteihin havaittiin hyödynnettävän kirjastoja *unittest* ja *django.test*. Yksikkötestit on mainittu Repolaisen kyselyvastauksessa, mutta edellä mainittuja työkaluja ei.

Projektissa on myös työkalu ohjelmakoodin staattiseen analyysiin, Prospector, mutta tämä ei ollut käytössä projektin toimitusputkessa. Kyselyn vastaus puolestaan väittää, että Repolainen hyödyntäisi jatkuvassa testauksessa ohjelmakoodin staattiseen analyysiin SonarQubea.

Tarkistettaessa uudestaan Repolaisen GitLab-etätietovarasto, on mahdollista havaita `sonarqube`-haara, jonka commitien toimitusputkitehtävissä SonarQube on tehnyt analyysinsa. Kyseistä haaraa ei kuitenkaan ole vielä liitetty `master` tai `develop`-haaroihin. Ensimmäinen `sonarqube`-haaran commit on tehty 26. päivä marraskuuta 2018, mikä on myöhemmin kuin jolloin omatoiminen tutustuminen suoritettiin. Tämä selittää havaintojen eron.

SonarQube-analyysi on tarkoitus Repolaisen vastuuhenkilön mukaan integroida osaksi järjestelmää sitten, kun siitä on järjestelmän kehitykselle hyötyä. Hänen mukaansa siitä on hyötyä tälle sitten, kun joku henkilö kykenee kehittämään järjestelmää.

Tutustuttaessa SonarQuben Repolaisesta tekemään raporttiin on havaittavissa nolla bugia, nolla haavoittuvuutta, kolme päivää teknistä velkaa, kaksi sataa *"code smell"*:ia sekä 48,4 prosentin testien ohjelmakoodin kattavuus. Fowlerin mukaan "code smell" on yleensä pintapuolen merkki syvemmästä ongelmasta järjestelmässä [42], joten niiden tunnistaminen voi olla hyödyllistä. Vertailtaessa SonarQuben ja Prospectorin havaintoja on huomattavissa, että molempien raporteissa on huomioita, joita toinen analyysityökalu ei tehnyt. Ero saattaa johtua esimerkiksi työkalujen erilaisista konfiguroinneista.

Taulukko 5.9 tekee yhteenvedon Repolaisessa käytetyistä työkaluista DevOps-käytäntöjä kohden omatoimisen tutustumisen ja kyselyn perusteella. Merkki - tarkoittaa, ettei järjestelmässä havaittu käytettävän omatoimisen tutustumisen tai kyselyn perusteella työkaluja kyseiseen käytäntöön. Vastaavasti eroavaisuudet työkaluissa kunkin käytännön osalta ovat lihavoidulla tekstillä.

Taulukko 5.9. *DevOps-käytäntöjen hyödyntämiseen käytetyt työkalut Repolaisessa omatoimisen tutustumisen ja kyselyn perusteella.*

Käytäntö	**Omatoiminen tutustuminen**	**Kysely**
Jatkuva integraatio	GitLab CI/CD	GitLab CI/CD, **Docker**
Jatkuva testaus	GitLab CI/CD, Docker, **Python: unittest + django.test**	GitLab CI/CD, Docker, **SonarQube**
Jatkuva toimitus	GitLab CI/CD	GitLab CI/CD, **Docker**
Jatkuva käyttöönotto	GitLab CI/CD, Fabric	GitLab CI/CD, **Docker**, Fabric
Jatkuva monitorointi	-	-
Jatkuva suunnittelu	-	-
Infrastruktuuri ohjelmakoodina	-	-

A+:n osalta hyödynnetyissä DevOps-käytännöissä on havaittavissa eroa. Sen osalta vastannut henkilö vastasi, ettei järjestelmässä hyödynnetä jatkuvaa integraatiota, vaikka omatoimisessa tutustumisessa tätä havaittiin toteutettavan Jenkinsilla. Vastaajan havainto saattaa perustua esimerkiksi kokemukseen hitaasti hyväksytyistä vetopyynnöistä.

Muilta osin hyödynnetyt käytännöt vastaavat omatoimisessa tutustumisessa havaittuja. Taulukko 5.10 tekee yhteenvedon A+:ssa hyödynnetyistä DevOps-käytännöistä omatoimisen tutustumisen ja kyselyn perusteella. Merkinnöistä ✓ilmaisee, että järjestelmä hyödyntää kyseistä käytäntöä, ja x päinvastoin.

Taulukko 5.10. *DevOps-käytäntöjen hyödyntäminen A+:ssa omatoimisen tutustumisen ja kyselyn perusteella.*

Käytäntö	**Omatoiminen tutustuminen**	**Kysely**
Jatkuva integraatio	✓	x
Jatkuva testaus	✓	✓
Jatkuva toimitus	x	x
Jatkuva käyttöönotto	x	x
Jatkuva monitorointi	x	x
Jatkuva suunnittelu	✓	✓
Infrastruktuuri ohjelmakoodina	x	x

Myös käytäntöjen hyödyntämiseen käytetyissä työkaluissa on eroa. Koska A+:n osalta kyselyyn vastannut henkilö ei merkannut jatkuvaa integraatiota käytettäväksi järjestelmässä, voitaneen sen osalta puuttuvat työkalut selittää tällä. Jatkuvaan testaukseen puolestaan käytetään kyselyn perusteella CircleCI:ta. Toisaalta vastajaa ei ollut työkalun käytöstä täysin varma.

A+:n dokumentaatiossa on Jenkins-osio [7], mutta toisaalta vastaaja mainitsi myös ylläpidon ongelmana vanhentuneen dokumentaation. Ilmeisemmin järjestelmä kuitenkin edelleen käyttää Jenkinsia, koska tällä hetkellä viimeisin *"A-plus-test-MASTER"*-testiajo, järjestyksessään 94., on suoritettu 3. päivä tammikuuta 2019 kello 14:36 [57].

Omatoimisessa tutustumisessa jatkuvaan testaukseen havaittiin käytettävän unittest ja django.test-kirjastoja yksikkötestaukseen sekä Seleniumia järjestelmätesteihin. A+:n osalta kyselyyn vastannut henkilö ei maininnut näitä työkaluja, mutta testaamisen tasoksi hän kertoi yksikkö- ja päästä päähän -testit. Riippuen järjestelmätesteistä ne saattaa olla mahdollista kokea myös päästä päähän -testeiksi, mikä voi selittää eron.

Kyselyn perusteella A+:n kehitysympäristö hyödyntää Dockeria. Omatoimisen tutustumisen perusteella näin ei kuitenkaan ole. A+:n kurssipohja kuitenkin hyödyntää Dockeria kehitysympäristössä [23]. A+:lle on myös olemassa Docker-levykuvaversio [13][14], mutta se on erillisessä Git-tietovarastossa, A+:n kehitysympäristö ei hyödynnä sitä eikä sen tuotantokäyttökelpoisuudesta ole tietoa. Edellä mainitut näkökulmat saattavat kuitenkin selittää havaitun eron omatoimisen tutustumisen ja kyselyn välillä.

Taulukko 5.11 kokoaa A+:ssa käytetyt työkalut DevOps-käytäntöjä kohden omatoimiseen tutustumiseen ja kyselyyn perustuen. Merkki - tarkoittaa, ettei järjestelmässä havaittu käytettävän työkaluja kyseiseen käytäntöön. Suluissa olevat työkalut ilmaisevat, että opetusjärjestelmä hyödyntää niiden avulla joltain osin, mutta ei täysin, kyseessä olevaa käytäntöä. Eroavaisuudet työkaluissa kunkin käytännön osalta ovat lihavoidulla tekstillä.

Taulukko 5.11. *DevOps-käytäntöjen hyödyntämiseen käytetyt työkalut A+:ssa omatoimisen tutustumisen ja kyselyn perusteella.*

Käytäntö	Omatoiminen tutustuminen	Kysely
Jatkuva integraatio	**Jenkins**	-
Jatkuva testaus	**Jenkins, Selenium, Python: unittest + django.test**	**CircleCI**
Jatkuva toimitus	-	-
Jatkuva käyttöönotto	-	-
Jatkuva monitorointi	-	-
Jatkuva suunnittelu	**GitHub: Issues, Slack, Google Groups**	-
Infrastruktuuri ohjelmakoodina	-	**(Docker)**

Versionhallinta Gitillä -kurssin osalta kyselyn perusteella jäi epäselväksi, hyödyntääkö kyseinen järjestelmä DevOps-käytäntöjä. Sen osalta kyselyyn vastannut vastuuhenkilö vastasi käytäntöjen osalta *"en osaa sanoa"*. Omatoimisen tutustumisen perusteella kysei-

nen kurssi ei hyödynnä osittaista infrastruktuurista ohjelmakoodia lukuunottamtta DevOps-käytäntöjä. Ero omatoimisen tutustumisen ja kyselyn välillä voi johtua esimerkiksi siitä, että vastuuhenkilö ei tiedä, hyödynnetäänkö järjestelmässä kysyttäjä käytäntöjä, tai siitä, että kysytyt käytännöt jäivät kyselyyn vastattaessa epäselviksi.

Koska Versionhallinta Gitillä -kurssin osalta ei saatu vertailutietoa hyödynnettyjen DevOps-käytäntöjen osalta, ei kyseisiin käytäntöihin käytettyjen työkalujen vertailu ole relevanttia. Taulukko 5.12 tekee yhteenvedon kyseisellä kurssilla hyödynnettyjen DevOps-käytäntöjen osalta omatoimisen tutustumisen ja kyselyn perusteella. Merkki x tarkoittaa, ettei järjestelmä hyödynnä kyseistä käytäntöjä, ja -, ettei tietoa hyödyntämisestä saatu.

Taulukko 5.12. *DevOps-käytäntöjen hyödyntäminen Versionhallinta Gitillä -kurssin omatoimisen tutustumisen ja kyselyn perusteella.*

Käytäntö	**Omatoiminen tutustuminen**	**Kysely**
Jatkuva integraatio	x	-
Jatkuva testaus	x	-
Jatkuva toimitus	x	-
Jatkuva käyttöönotto	x	-
Jatkuva monitorointi	x	-
Jatkuva suunnittelu	x	-
Infrastruktuuri ohjelmakoodina	x	-

6. DEVOPS-KÄYTÄNTÖJEN JA -TYÖKALUJEN HYÖDYNTÄMISEN KEHITTÄMINEN

Tässä luvussa esitellään luvuissa 4 ja 5 esitettyjen tutkimuksen tulosten perusteella havaittuja opetusjärjestelmien kehitykseen ja ylläpitoon sekä työkaluihin ja suoritusympäristöihin liittyviä ongelmia. Näiden pohjalta on tarkoitus luoda kehitysideoita, jotka olisivat myös mahdollisesti yleistettävissä esimerkiksi uusiin opetusjärjestelmiin ja tutkimuksessa suoritettavaan ohjelmistokehitykseen. Alaluku 6.1 esittää kehitysideat työkalujen sekä suoritusympäristöjen ja 6.2 opetusjärjestelmien osalta.

6.1 Työkalut ja suoritusympäristöt

Työkalujen osalta kaikki Tampereen teknillisellä yliopistolla käytössä olleet työkalut olivat dokumentoituna yliopiston intrasivustolla. Suoritusympäristöjen osalta tieto ei kuitenkaan ollut yhtä kattavaa. IT-palveluilla havaittiin esimerkiksi olevan sisäinen dokumentaatio palveluiden hinnastoista. Koska pääsy intrasivustolle on rajattu nykyisin entisen Tampereen teknillisen yliopiston jäsenille, olisi kyseisen tiedon jakaminen täällä toivottavaa ja todennäköisesti jopa mahdollista.

Työn tekijä löysi myös sattumalta Microsoft Azuren yliopiston saatavilla olevista palveluista. Yliopiston intrasivustolta ei kuitenkaan löytynyt ohjeistusta tämän käyttöön. On luonnollista, että IT-palveluilla voi olla työtehtäviä, joiden prioriteetti on korkeampi kuin vähäisellä kiinnostuksella olleen Azuren ohjeistukset.

Osa henkilöstöstä ei välttämättä kuitenkaan edes tiedä kyseisestä mahdollisuudesta, joten jonkinasteisen ohjeistuksen sisäinen jakaminen voisi olla suositeltavaa. Yliopistojen yhdistyminen voisi olla kannustin tälle, sillä ainakin pääsy Azure-portaaliin on sallittu.

TTY:lla (Tampereen teknillisellä yliopistolla) tarjotun webhotel-palvelun ohjeistuksessa mainittiin, että mahdolliset järjestelmien päivityksistä ohjelmistoille aiheutuneet ongelmat ovat näiden hallinnoijien vastuulla. Olisi kuitenkin tarpeen ilmoittaa tehtävistä päivityksistä etukäteen, jos näin ei jo tehdä. Tällöin ohjelmiston hallinnoija voisi ehkä paremmin varautua mahdollisiin ongelmatilanteisiin.

Entisen Tietotekniikan laboratorion, jonka toiminnot ja henkilöstö ovat nykyisin osa Tampereen yliopiston Informaatioteknologian ja viestinnän tiedekunnan Tietotekniikan yksikköä, suoritusympäristöt olivat kuitenkin vielä enemmän pimennossa. Käytännössä niiden selvittäminen vaati ammattitiedon jakamisen seurauksena saadun tiedon hyödyntämistä.

On välttämätöntä, että joku henkilö tietää jonkun toisen henkilön, joka ylläpitää jotakin järjestelmää, mutta varsinkin isoissa muutoksissa kuten Tampereen yliopiston ja Tampereen teknillisen yliopiston yhdistymisessä suoritusympäristöjen ja työkalujen dokumentaatio olisi ajankohtaista. Voisihan olla esimerkiksi niin, että entisellä Tietotekniikan laboratoriolla olisi joitakin sellaisia palveluita ja järjestelmiä, jotka voitaisiin ottaa myös uudessa laajemmassa Tietotekniikan yksikössä käyttöön. Keskitetyn tiedon puutteen takia on myös mahdollista, että on olemassa muitakin käytössä olevia työkaluja ja suoritusympäristöjä kuin tässä työssä havaitut.

Tämän diplomityön puitteissa Tietotekniikan laboratorion Triton-hankkeen teknisiä tietoja ei kyetty selvittämään. On epäselvää, missä vaiheessa hanke on. Voisi kuitenkin olla hyödyllistä jakaa enemmän tilannetietoa projektista. Jakaminen on esimerkiksi Humblen ja Moleskyn mukaan yksi DevOpsin osa-alueista [50].

Tietämyksen ja osaamisen jakaminen muillakin tavoilla voisi olla hyödyllistä. Yksi vaihtoehto voisi olla esimerkiksi Wettinger et al. vuonna 2017 esittelemän DevOps-ratkaisutietovaraston [103] tyylinen järjestelmä, johon voitaisiin kerätä tietoa teemaan liittyvistä aihealueista kuten esimerkiksi automaattisista toimitusputkista, staattisen analyysin toteuttamisesta, sovelluksen monitoroinnista ja suoritusympäristöjen automaattisesta provisioinnista. Kuten Wettinger et al.:n ratkaisu myös tällainen voisi olla jollakin tavalla automatisoitavissa, esimerkiksi käymällä läpi GitLab-etätietovarastoprojektien `.gitlab-ci.yml`-tiedostoja.

Humblen ja Moleskyn mainitsemista osa-alueista automaatio jo toteutuukin yliopistolla käytössä olevien GitLab-asennusten johdosta. Kyseisen järjestelmän ansiosta yliopiston tietotekninen infrastruktuuri myös mahdollistaa luvussa 2.3 esitettyjen DevOps-käytäntöjen hyödyntämisen ainakin jatkuvan integraation, testaamisen, toimituksen ja käyttöönoton osalta.

6.2 Opetusjärjestelmien kehitys ja ylläpito

Luvussa 5 tutkituista opetusjärjestelmistä Repolainen ja Versionhallinta Gitillä -verkkokurssi käyttävät luvussa 4.1 esitettyjä GitLab-asennuksia versionhallintaan. Näistä Repolainen hyödyntää GitLab CI/CD:ta, mutta ei esimerkiksi Issue Boardia. Tehdyn tutkimuksen perusteella voidaan perustellusti esittää, että jokaisessa järjestelmässä on kehitettävää DevOps-käytäntöjen hyödyntämisessä.

Esimerkiksi Repolaisen testaaminen rajoittuu tällä hetkellä yksikkötesteihin. SonarQube-integraatio on kuitenkin jo ainakin suunnilleen tehty, joten sen viimeistely ja käyttöönotto mahdollisimman pian voisi olla hyödyllistä: näin esimerkiksi joku henkilö saattaisi helpommin ohjautua tekemään korjauksia, jos ongelmaraportit olisivat saatavilla. SonarQube-analyysin valmistelu on kuitenkin osoitus siitä, että käytäntöjen hyödyntäminen on kehittymässä ainakin jossain määrin. Myös muut opetusjärjestelmät, esimerkiksi A+, ja tutkimushankkeet laajemminkin voisivat hyötyä staattisen analyysin käyttöönotosta.

Repolaisessa on käytössä automaattinen toimitusputki, mutta ei automaattista edelliseen versioon palaamista. Uuden version käyttöönotossa voisi olla hyödyllistä suorittaa testi järjestelmän toimitusputkelle. Yksi mahdollinen ratkaisu voisi olla ensin tehdä käyttöönotto testiympäristössä, tarkistaa tietyn verkko-osoitteen saatavuus, tehdä käyttöönotto tuotantoympäristöön ja tarkistaa tietyn verkko-osoitteen saatavuus myös siellä. Täten saataisiin ainakin tieto siitä, että järjestelmä on valmis käsittelemään HTTP-pyyntöjä myös uuden version käyttöönoton jälkeen. Jos ongelmia ilmenisi, saatettaisiin tarve vanhaan versioon palamiseen havaita aiemmin, vaikka prosessi manuaalinen edelleen olisikin.

Opetusjärjestelmistä vain prplatform hyödyntää infrastruktuurista ohjelmakoodia laajemmin. Repolaisen ja A+:n kehitysympäristöjen pystytyksessä ongelmia voivat aiheuttaa esimerkiksi ristiriidat vaaditun ja kehitysympäristöön asennetun Python-version välillä. Molemmissa järjestelmissä on myös kehitysympäristössä käytössä esimerkiksi eri tietokanta kuin tuotantoympäristössä.

Näihin eri komponenttien jakaminen esimerkiksi Docker-konteiksi voisi mahdollisesti auttaa. Näitä kontteja olisi mahdollista operoida esimerkiksi Docker Composen avulla. Dockerin käyttöönotto saattaisi nopeuttaa kehitysympäristön luontia sekä helpottaa tuotantoympäristön luontia ja sen kahdentamista. Tällöin ohjelman toiminnan varmistaminen voisi mahdollisesti olla laadukkaampaa.

Kuten luvussa 5.3 mainitaan, A+:sta on jo olemassa Docker-versio. A+:n järjestelmän kehityksen sisältävä GitHub-etätietovarasto voisi ottaa tämän käyttöön. Repolaisen osalta Docker-kontitus saattaisi myös helpottaa ainakin kehitysympäristöissä tehtävää GitLab-asennusta, koska kyseisen riippuvuuden kehitysympäristön asennusohjeet ovat varsin monimutkaiset.

Sekä A+:n että Repolaisen dokumentaatio on vanhentunutta. Järjestelmien Docker-kontitus olisi myös hyvä mahdollisuus uuden ajantasaisen dokumentaation tekemiselle. Jos järjestelmien käyttöönoton ja asennuksen monimutkaisuutta onnistuttaisiin vähentämään, saattaisivat myös näiden vaiheiden dokumentaatiot olla yksinkertaisempia ja täten helpommin ajantasalla pidettäviä.

Dockeria ja Docker Composea hyödyntävän prplatformin dokumentaatio on puutteellinen, mutta kehitysympäristön luominen oli varsin itsensä selittävä prosessi edellä mainittujen työkalujen johdosta. Tämä toki olettaen, että ympäristön luomista yrittävällä henkilöllä on jonkin tason tietämys kyseisistä työkaluista. Tämä näkökulma kuitenkin todistaa, että järjestelmien hallinnointi voisi olla yksinkertaisempaa esimerkiksi Dockerin avulla. Toivottavaa prplatforminkin osalta olisi kuitenkin lisätä dokumentaatio kehitysympäristön luomiseen, koska prosessi ei ollut kovin monimutkainen, ja täten sen ajantasalla pitämisen ei pitäisi olla kovin vaativaa.

Repolaisen osalta työn tekijä on kuullut joistakin ongelmista ja kehitysideoista, mutta kuten luvussa 5.1.1, on kyseisen projektin GitLab-etätietovaraston Issue Board tyhjä. Tiket-

tien luonti voisi edistää korjausten ja uusien ominaisuuksien tekoa. Se voisi myös parantaa järjestelmän jatkokehitystä ja mahdollistaa jatkuvan suunnittelun.

Tämän työn teon loppuvaiheella Repolaisen osalta selvisi myös Tampereen yliopistoon liittyvän tietojärjestelmien migraation seurauksena syntynyt ongelma. Koska kaikkia järjestelmiä ei migratoitu, ovat ongelmat migratoitujen ja migratoimattomien järjestelmien välillä mahdollisia. Jos Repolaista hallinnoi yksi henkilö, on mahdollista, ettei kaikkia asioita huomata. Täten olisi hyödyllistä, jos useammalla henkilöllä olisi parempi tietämys siitä. Kommunikaation ja jakamisen merkitys näkyy tässäkin tilanteessa.

Rajalliset resurssit Repolaisen hallinnoinnissa näkyvät myös esimerkiksi sen suoritusympäristöjen pääsynhallinnassa: vain järjestelmän ylläpitäjän tietyltä tietokoneelta on pääsy sen testiympäristöön. Jonkin tasoinen rajaus lienee ajankohtainen, etteivät esimerkiksi opiskelijat pääse käyttämään testiympäristöä, mutta sen saatavuus yhdelle henkilölle myös rajaa alhaiseksi mahdollisuuden testata järjestelmää sen tuotantoympäristöä vastaavassa ympäristössä. Voisi olla järjestelmän laadunvarmistuksen kannalta hyödyllistä, jos jokaisella Repolaista käyttävällä opettajalla olisi pääsy sen testiympäristöön.

Tutkimuksen aikana havaittiin, että pääsyoikeus Repolaisen GitLab-projektiin on vain tietyillä henkilöillä. Jakamista voisi olla sekin, että kaikilla Tietotekniikan yksikön työntekijöilllä tai vähintään järjestelmää käyttävillä opettajilla olisi raportoijatason oikeudet kyseiseen etätietovarastoon. Tämä mahdollistaisi esimerkiksi bugien ilmoittamisen suoraan Issue Boardilla, ja voisihan olla, että joku sellainen henkilö, joka ei ole aktiivisesti mukana järjestelmän kehityksessä tai ylläpidossa, korjaisi jonkun pienen bugin, jos hänellä olisi jostain syystä hieman vapaata aikaa.

Kyselyyn A+:n osalta vastannut henkilö oli epävarma testaukseen käytetyistä työkaluista. Kommunikoinnin ja jakamisen tehostaminen voisi olla relevanttia myös kyseisen järjestelmän osalta, sillä kehityskäytännöt ja niihin lukeutuvan jatkuvan integraation prosessi olisi tarpeen tietää jokaisen, joka on järjestelmää edes jollakin tavalla kehittänyt.

A+:aa ylläpitävä henkilö kokee kyselyn mukaan, ettei automaattinen käyttöönotto olisi toivottavaa, koska järjestelmää kehittää eri taho kuin hän. Tutkimuksen perusteella A+:n jatkuvan integraation ja testauksen prosessi on rikki, ja sellaista ohjelmakoodia, joka ei läpäise testejä tässä prosessissa, on mahdollisesti julkaistu. Tämä voi luonnollisesti aiheuttaa luottamuksen puutteen järjestelmää kohtaan.

Ideaalisessa tilanteessa järjestelmän Jenkinsputki ja Selenium-testit korjattaisiin ennen kuin uutta ohjelmakoodia julkaistaan. Rajalliset resurssit ja aikatauluvaatimukset voivat kuitenkin hankaloittaa tätä. Siltikin järjestelmän laadunvalvonnan tilanne on pidemmällä aikavälillä kestämätön, koska sitä jatkokehitetään jatkuvasti. Myös järjestelmän testisivuston korjaaminen voisi olla hyödyllistä, sillä täten sitä ehkä kokeilisi joku, mikä saattaisi parantaa laadunvarmistusta, jos jokin ongelma havaittaisiin jo siellä.

Nykyisen Tampereen yliopiston osalta A+:lle voisi olla mahdollista tehdä automaattinen

toimitusputki. Tässä voitaisiin hyödyntää esimerkiksi GitLabia ja sen jatkuvan toimituksen ja käyttöönoton ominaisuuksia, jos kyseinen palvelu tulisi myöhemmin myös Tampereen yliopiston käyttöön. Kuten luvussa 5.3 mainittiin, on A+:lle olemassa jo Docker-versio. Tuotantokäytössä olevaa A+-asennusta voisi mahdollisesti operoida Docker Composen avulla.

Kun uusi versio A+:sta julkaistaisiin, voitaisiin Docker Compose ja GitLab CI/CD -konfiguraatioissa päivittää Docker-levykuvan tagi, tehdä tästä Git-commit, työntää tämä GitLab-etätietovarastoon, antaa GitLab CI/CD:n suorittaa järjestelmän testit, ja jos ne onnistuvat, tehdä automaattinen käyttöönotto ensin testiympäristössä ja tämän jälkeen tuotantoympäristössä sisältäen testit sille, onnistuivatko käyttöönotot. On kuitenkin epäselvää, onko A+:n Docker-levykuva valmis tuotantokäyttöön, joten tätä ideaa ei välttämättä ole mahdollista toteuttaa luotettavalla tavalla.

Jatkuva käyttöönotto ei välttämättä sovellu itse A+:n `GitHub`-etätietovarastoon, koska järjestelmä on käytössä myös muualla kuin Aalto-yliopistolla. Koska järjestelmästä on kuitenkin olemassa jo Docker-levykuva, voisi jatkuvaa toimitusta soveltaa esimerkiksi projektissa jo käytössä olevan Jenkinsin avulla siten, että tämä tekee Docker-levykuvat ja työntää ne Docker Hub -artefaktitietovarastoon.

Tällä hetkellä A+:n julkaisusykli on kuitenkin puolivuosittainen: uusi versio julkaistaan tammikuussa ja elokuussa [65]. Tässä tilanteessa automatisoidusta toimituksesta ei välttämättä ole hyötyä. Järjestelmän kehityksessä voitaisiin kuitenkin miettiä, voitaisiinko julkaisujen määrää tihentää ja tiukoista aikarajoista luopua. Jatkuvalla toimituksella saatettaisiin saavuttaa luvussa 2.3.4 esitettyjä hyötyä. Tiukkojen aikataulujen poistuminen saattaisi myös vähentää henkistä kuormitusta, jos kehityksen resurssit ovat rajalliset.

A+:n tulevaisuuden suunnitelmassa järjestelmän käyttöä ollaan helpottamassa opettajan näkökulmasta. Myös sen arkkitehtuuria ollaan muovaamassa paremmin palvelusuuntautunutta arkkitehtuuria noudattavaksi. [65] Tämä on positiivista. Suositeltavaa olisi kuitenkin huomioida myös järjestelmän kehitysnäkökulma, johon voisi hyödyntää edellä mainittuja kehitysideoita.

A+-kurssien kuten Versionhallinta Gitillä opetusmateriaalien testaaminen saattaisi olla hankalaa ohjelmallisesti, mutta materiaalin tuotantokelpoiseksi kääntämiseen voisi olla mahdollista hyödyntää esimerkiksi GitLab CI/CD:ta. Opettaja joutuisi vielä itse tarkastamaan materiaalin oikeudellisuuden omassa kehitysympäristössään, mutta jos sen kääntäminen onnistuu myös GitLab CI/CD:lla, voisi se mahdollisesti riittää tämän jälkeen takeeksi siitä, että materiaali voidaan siirtää myös tuotantokäyttöön.

Tällöin myöskään käännösartefakteja ei olisi välttämätöntä pitää Git-tietovarastossa ja opettajan riittäisi tietää vain GitLab-etätietovarasto eikä tämän lisäksi tuotantopalvelimen kyseisen kurssin Git-etätietovarastoa. Tämä voisi hieman yksinkertaistaa julkaisuprosessia opettajan näkökulmasta.

prplatformin osalta käytäntöjä voisi kehittää esimerkiksi jatkuvan integraation mahdollistavan CircleCI:n avulla, koska tämä integroituu GitHubiin. Tällä hetkellä testaaminen tapahtuu ennen commitia Gitin koukkujen avulla, mutta nämä paikalliset koukut eivät ole etätietovarastossa. Sekin olisi kehitysaskel, jos koukut ja ohjeet niiden käyttöönottamiselle olisivat jossakin muodossa GitHub-etätietovarastossa.

Jokaisen opetusjärjestelmän osalta jonkinlainen monitorointi olisi suositeltavaa. Kuten luvussa 2.3.6 mainitaan, se mahdollistaa esimerkiksi suorituskykyongelmien aikaisemman havaitsemisen ja palvelun käytön seurannan. Edellä mainitun avulla sovelluksen laatu voi parantua nopeammin. On myös parempi havaita ongelma itse kuin että loppukäyttäjä havaitsee sen.

Tämän työn teon aikana työn tekijä havaitsi myös tilanteita, joissa joku henkilö oli tehnyt tai löytänyt arkistosta työkalun johonkin tiettyyn opetukseen liittyvään tarkoitukseen ja tämän jälkeen selvisi, että joku toinenkin oli tehnyt vastaavan työkalun. On toisaalta tutkimushenkistä tehdä ja kokeilla itse, mutta jonkinlainen kordinaatio voisi kuitenkin olla hyödyllistä. Ehkä yhdellä yhdessä tehdyllä työkalulla voisi olla paremmat edellytykset olla laadukas ja yleiskäyttöinen kuin kourallisella yksin tehdyillä? Taulukko 6.1 tekee yhteenvedon kehitysideoista.

Taulukko 6.1. *Yhteenveto tutkimuksen tulosten perusteella esitetyistä kehitysideoita.*

Kehitysidea	**Kohde**
Dokumentaation ja ohjeistuksen parantaminen	Yliopiston ja yksikön suoritusympäristöt sekä opetusjärjestelmät
Kommunikaation ja jakamisen tehostaminen	Opetusjärjestelmien sisäinen sekä niiden ja IT-palveluiden ylläpidon välinen
Jatkuvan suunnittelun tehostaminen	Opetusjärjestelmät
Jatkuvan monitoroinnin käyttöönotto	Opetusjärjestelmät
Infrastruktuurin ohjelmakoodina esittämisen tehostaminen	Opetusjärjestelmät

7. YHTEENVETO

Tässä diplomityössä on selvitetty DevOps-käsitettä sekä siihen liittyvää kulttuurellista näkökulmaa, käytäntöjä sekä sen ongelmia ja hyötyjä, ja laadittu tutkimusmenetelmä DevOps-käytäntöjen hyödyntämisen selvittämiseksi opetusjärjestelmien kehityksessä ja ylläpidossa. Tämän perusteella on suoritettu tutkimus sekä analysoitu ja esitetty sen tulokset. Näiden perusteella on esitetty kehitysideat.

DevOps on käsitteenä monitulkintainen ja jatkuvasti kehittyvä, mutta samalla kuitenkin suosiotaan tasaisesti kasvattava ilmiö. Ohjelmistotuotannon ammattilaisten ja yritysten on välttämätöntä seurata trendejä, jotta he pysyvät kehityksessä mukana ja turvaavat elinkeinonsa ja liiketoimintansa myös tulevaisuudessa. Tämä asettaa haasteita myös alan opetukselle ja tutkimukselle.

Tähän haasteeseen on vastattu. Niin kansainvälisesti kuin nykyisessä Tampereen yliopistossakin opetetaaan jo DevOpsiin liittyviä käytäntöjä, esimerkiksi jatkuvaa integraatiota. Tämä huomioiden onkin merkityksellistä selvittää, noudatetaanko näitä käytäntöjä myös itse.

Työn tutkimustavoitteena oli selvittää, miten DevOps-käytäntöjä hyödynnetään opetusjärjestelmien kehittämisessä ja ylläpidossa, ja miten tätä voisi kehittää. Tähän tutkimuskysymykseen vastattiin laadullisen tapaustutkimuksen avulla. Tämän yhteydessä pyrittiin myös selvittämään, onko yliopisto-organisaatiolla tarjota edellytyksiä eli tässä työssä työkaluja ja sopivia suoritusympäristöjä näiden käytäntöjen hyödyntämiseen.

Tutkimuksen tulokset osoittavat, että työkaluja ja suoritusympäristöjä on saatavilla, mutta dokumentaatio ja ohjeitus ovat puutteelliset jälkimmäisten osalta. Myös opetusjärjestelmien osalta DevOps-käytäntöjen hyödyntämisessä on kehitettävää. On kuitenkin huomioitava, että esimerkiksi Repolainen hyödyntää jo yliopiston GitLab-asennusta jatkuvan integraation, testauksen, toimituksen ja tuotannon osalta, ja kehitystäkin on havaittavissa, sillä järjestelmään on ollut valmisteilla ohjelmakoodin staattinen analyysi SonarQuben avulla.

Keskeisimmiksi luvussa 6 esitetyistä kehitysideoista voidaan työkalujen ja suoritusympäristöjen osalta ottaa esille dokumentaation ja ohjeistuksen parantaminen. Vastaavasti opetusjärjestelmien kehityksen ja ylläpidon osalta nämä ovat jonkinlaisen monitoroinnin käyttöönottaminen sekä infrastruktuurin ohjelmakoodina esittämisen ja jatkuvan suunnittelun tehostaminen. Tietotekniikan yksikön sisäisen sekä tämän ja IT-palveluiden välisen kommunikaation ja jakamisen merkitystä ei myöskään voi aliarvioida varsinkaan, kun yliopistojen yhdistyminen on kirjoitushetkellä juuri tapahtunut. Taulukkossa 6.1 on yhteenveto kehitysideoista.

Repolaisen osalta työn tekijä ja Repolaisen vastuuhenkilö ovat kirjoitushetkellä jo aloittaneet toimenpiteet luvussa 6.2 esitettyihin kehitysideoihin perustuen: kehitysympäristö on kontitettu Dockerin ja Docker Composen avulla, millä on pyritty sen asentamisprosessin yksinkertaistamiseen sekä hallinnettavuuden helpottamiseen, ja GitLabin Issue Boardille on aloitettu tikettien luonti.

Konkreettisena toimenpiteenä Tampereen yliopiston Informaatioteknologian ja viestinnän tiedekunnan Tietotekniikan yksikölle suositellaan lisäksi muiden tässä työssä esitettyjen kehitysideoiden kelpoisuuden selvittämistä soveltuvuusselvityksen avulla. Esimerkiksi toimenpiteet A+:n uuden asennuksen sekä käyttöönoton osalta ovat mahdollisia. Voi myös olla hyödyllistä jatkokehittää kyseisiä kehitysideoita isommalla ryhmällä sekä järjestää esimerkiksi aivoriihiä, sillä ryhmässä saattaa olla mahdollista kehittää enemmän ideoita kuin yksin. Ryhmätapaamiset ja aivoriihet olisivat myös jakamiseen kannustava toimenpide.

Yliopiston IT-palveluista vastaavalle yksikölle suositellaan kattavan suoritusympäristöjen dokumentaation koostamista yhteen sijaintiin esimerkiksi uudelle intrasivustolle. Tälle ja Tietotekniikan yksikölle suositellaan näiden välisen kommunikaation ja jakamisen tehostamista. Tähän esimerkiksi jo käytössä oleva Mattermost tarjoaa mahdollisuuden.

Työ onnistui hyvin ja se toteuttiin aikataulussa Kanban-menetelmää hyödyntäen. Tutkimuksen toistettavuus on pyritty mahdollistamaan kirjaamalla siihen liittyvät yksityiskohdat, menetelmät ja tulokset mahdollisimman tarkasti. Opetusjärjestelmien kehityksen ja ylläpidon osalta tehdyn laadullisen tapaustutkimuksen omatoimisen tutustumisen tulosten mahdollisina luotettavuusongelmina ovat työn tekijän mahdollinen subjektiivisuus sekä DevOps-tietämyksen puute. Ensimmäisen vaikutusta pyrittiin vähentämään kyselyn avulla ja jälkimmäistä lukua 2 varten tehdyn DevOps-selvityksen avulla.

Kysely tarjosi osittain vahvistusta omatoimisessa tutustumisessa tehdyille havainnoille, mutta myös eroilla havaittiin olevan merkityksensä. Kysely saavutti erään opetusjärjestelmän osalta koko kohdeyleisön, mutta muiden osalta tämä jäi auki. Koska kyseisen järjestelmän kohdeyleisö osoittautui pieneksi, olisi esimerkiksi puolistruktuurisella haastattelulla saatettu onnistua keräämään yksityiskohtaisempaa tietoa siitä.

Toisaalta kysely suoritettiin nimettömänä ja vastaajille tarjottiin mahdollisuus vastata lisäkysymyksiin ja keskustella kasvokkain. Kyselyn valintaa on mahdollista perustella myös sillä, että haastatteluja olisi tulkinnut yksi henkilö. Kyselyvastaukset ovat valmiiksi kirjallisessa ja digitaalisessa muodossa, joten niiden tallennusmuoto ei aiheuta tulkintavaikeuksia toisin kuin esimerkiksi haastattelun epäonnistunut äänitys tai haastattelumerkintöjen niiden tekijälle ominainen käsiala. Verkkokyselyissä asiaankuulumattomat vastaukset ovat myös mahdollinen riski. Tätä riskiä pyrittiin hallitsemaan henkilökohtaisten kyselylinkkien avulla.

Työkalujen ja suoritusympäristöjen osalta olisi voinut olla hyödyllistä suorittaa esimerkiksi Tietotekniikan laboratorion henkilöstölle kysely, mitä työkaluja ja suoritusympäris-

töjä he tietävät olevan käytössä, tai mitä sellaisia he ovat kokeilleet yliopistolla. Kyselyn tulosten perusteella olisi voitu vahvistaa havaittujen työkalujen ja suoritusympäristöjen saatavuus. Sähköpostiviestein suoritettu selvitys ei välttämättä saavuttanut kaikkia niitä henkilöitä, joilla olisi ollut tietämystä aiheesta.

Työssä on pyritty yleiskäyttöisten kehitysideoiden tekemiseen. Ideoiden toteutuskelpoisuus jää kuitenkin tämän työn osalta selvittämättä. On kuitenkin teoriassa mahdollista, että muutkin kuin tässä työssä mainitut opetusjärjestelmät tai yleisemmin tutkimukseen liittyvä ohjelmistokehitys voisivat hyötyä kyseisistä kehityssuunnista.

Tässä luvussa aiemmin mainittu kehitysideoiden jatkokehittäminen ja tarkentaminen voisi olla mahdollinen jatkotutkimuksen aihe. Toinen tutkimusongelma voisi olla ottaa kehitysideat käyttöön ja tutkia, saavutetaanko niillä hyötyä, millaista, ja missä määrin. Tämän perusteella voisi puolestaan olla mahdollista tutkia, vaikuttaako DevOps-käytäntöjen kehittäminen opetuksen työkalujen teossa ja hallinnoinnissa myös opetuksen laatuun saman aihealueen osalta.

LÄHTEET

[1] | Docker Documentation, Docker Inc., verkkosivu. Saatavissa (viitattu 26.10.2018): https://docs.docker.com/engine/reference/builder/

[2] 182 DevOps Jobs | LinkedIn, LinkedIn, verkkosivu, 2018. Saatavissa (viitattu 18.10.2018): https://www.linkedin.com/jobs/search?keywords=DevOps&location=Finland&locationId=fi:0

[3] A+ | A+ LMS, GitHub Pages, verkkosivu. Saatavissa (viitattu 9.11.2018): https://apluslms.github.io/

[4] A-plus-test-PR/builds/169/log, plustest.cs.hut.fi, verkkosivu. Saatavissa (viitattu 9.11.2018): http://plustest.cs.hut.fi/jobs/A-plus-test-PR/builds/169/log

[5] a-plus/A-plustest-publish.xml at f5037dcf6eabc3a406edd86d3f3fb55ae748b194 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/blob/f5037dcf6eabc3a406edd86d3f3fb55ae748b194/doc/jenkins/A-plustest-publish.xml

[6] a-plus/DEPLOYMENT.md at f5037dcf6eabc3a406edd86d3f3fb55ae748b194 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/blob/f5037dcf6eabc3a406edd86d3f3fb55ae748b194/doc/DEPLOYMENT.md

[7] a-plus/README.md at f5037dcf6eabc3a406edd86d3f3fb55ae748b194 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/blob/f5037dcf6eabc3a406edd86d3f3fb55ae748b194/doc/jenkins/README.md

[8] a-plus/README.md at f5037dcf6eabc3a406edd86d3f3fb55ae748b194 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/blob/f5037dcf6eabc3a406edd86d3f3fb55ae748b194/doc/README.md

[9] About devopsdays, devopsdays.org, verkkosivu, 2018. Saatavissa (viitattu 24.9.2018): https://www.devopsdays.org/about/

[10] About SQLite, SQLite, verkkosivu. Saatavissa (viitattu 8.11.2018): https://www.sqlite.org/about.html

[11] M. Airaj, Enable cloud DevOps approach for industry and higher education, Concurrency and Computation: Practice and Experience, Vol. 29, Iss. 5, March 10, 2017.

[12] API: allow POST by teacher to create a new submission by piehei · Pull Request #374 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/pull/374

[13] apluslms/run-aplus-front - Docker Hub, Docker Inc., verkkosivu. Saatavissa (viitattu 9.1.2019): https://hub.docker.com/r/apluslms/run-aplus-front

[14] apluslms/run-aplus-front at 5809d09433d228a6a69fee48f25fdbce845b4963, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.1.2019): https://github.com/apluslms/run-aplus-front/tree/5809d09433d228a6a69fee48f25fdbce845b4963

[15] M. Artac, T. Borovssak, E.D. Nitto, M. Guerriero, D.A. Tamburri, DevOps: Introducing Infrastructure-as-Code, in: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), May 20–28, 2017, IEEE, pp. 497–498.

[16] Automatic HTTPS - Caddy, Light Code Labs, LLC., verkkosivu. Saatavissa (viitattu 10.12.2018): https://caddyserver.com/docs/automatic-https

[17] K. Ayres, Tutustu uusiin dekaaneihin! | Sidosryhmälehti Rajapinta | Tampereen teknillinen yliopisto, Tampereen teknillinen yliopisto, verkkosivu, 2016. Saatavissa (viitattu 18.10.2018): http://www.tut.fi/rajapinta/artikkelit/2016/3/tutustu-uusiin-dekaaneihin%21

[18] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, IEEE Software, Vol. 33, Iss. 3, March 18, 2016, pp. 42–52.

[19] M.d. Bayser, L.G. Azevedo, R. Cerqueira, ResearchOps: The Case for DevOps in Scientific Applications, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), May 11–15, 2015, IEEE, Ottawa, ON, Canada, pp. 1398–1404.

[20] Branches · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/branches

[21] G.G. Claps, R.B. Svensson, A. Aurum, On the journey to continuous deployment: Technical and social challenges along the way, Information and Software Technology, Vol. 57, January, 2015, pp. 21–31.

[22] Commits · piehei/prplatform, GitHub, Inc., verkkosivu. Saatavissa (viitattu 10.12.2018): https://github.com/piehei/prplatform/commits/cc39e3afb7cddc7c6e56bca6e9fae02ccae062a1

[23] course-templates/05_docker.rst at 1723df3b336cb1e2d31216925f3e5e3809d03f10 · apluslms/course-templates, GitHub, Inc., verkkosivu. Saatavissa (viitattu 4.1.2019): https://github.com/apluslms/course-templates/blob/1723df3b336cb1e2d31216925f3e5e3809d03f10/m01_introduction/05_docker.rst

[24] P. Debois, Agile Infrastructure and Operations: How Infra-gile are You?, in: Agile 2008 Conference, Aug 4–8, 2008, IEEE, Ottawa, ON, Canada, pp. 202–207.

[25] P. Debois, Opening Statement, Cutter IT Journal, Vol. 24, Iss. 8, August, 2011, pp. 3–5.

[26] DevOps - Tutki - Google Trends, Google, verkkosivu. Saatavissa (viitattu 17.1.2019): https://trends.google.fi/trends/explore?date=2014-01-17%202019-01-17&q=DevOps

[27] Devopsdays Ghent 2009, devopsdays.org, verkkosivu, 2009. Saatavissa (viitattu 24.9.2018): https://legacy.devopsdays.org/events/2009-ghent/

[28] Directory of Azure Cloud Services | Microsoft Azure, Microsoft, verkkosivu. Saatavissa (viitattu 29.11.2018): https://azure.microsoft.com/en-us/services/

[29] J. Downs, J. Hosking, B. Plimmer, Status Communication in Agile Software Teams: A Case Study, in: 2010 Fifth International Conference on Software Engineering Advances, August 22–27, 2010, IEEE, Nice, France, pp. 82–87.

[30] Eat Your Own Dog Food, Investopedia, LLC., verkkosivu. Saatavissa (viitattu 18.10.2018): https://www.investopedia.com/terms/e/eatyourowndogfood.asp

[31] F. Economou, J.C.H. ands Pat Norris, Your data is your dogfood: DevOps in the astronomical observatory, in: Software and Cyberinfrastructure for Astronomy III, SPIE Astronomical Telescopes and Instrumentation conference, June, 2014, arXiv.org.

[32] ELK Stack: Elasticsearch, Logstash, Kibana | Elastic, Elasticsearch BV, verkkosivu. Saatavissa (viitattu 29.10.2018): https://www.elastic.co/elk-stack

[33] F. Erich, C. Amrit, M. Daneva, A Mapping Study on Cooperation between Information System Development and Operations, in: PROFES 2014: Product-Focused Software Process Improvement, 2014, Springer International Publishing, pp. 277–280.

[34] F.M.A. Erich, C. Amrit, M. Daneva, A qualitative study of DevOps usage in practice, Journal of Software: Evolution and Process, Vol. 29, Iss. 9, June, 2017.

[35] Etusivu | Tampereen korkeakouluyhteisö, Tampereen korkeakouluyhteisö, verkkosivu, 2019. Saatavissa (viitattu 11.1.2019): https://www.tuni.fi/fi

[36] Excel csv by jrp6 · Pull Request #344 · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/pull/344

[37] B.S. Farroha, D.L. Farroha, A Framework for Managing Mission Needs, Compliance, and Trust in the DevOps Environment, in: 2014 IEEE Military Communications Conference, October 6–8, 2014, IEEE, Baltimore, MD, USA, pp. 288–293.

[38] D.G. Feitelson, E. Frachtenberg, K.L. Beck, Development and Deployment at Facebook, IEEE Internet Computing, Vol. 17, July-August, 2013, pp. 8–17.

[39] B. Fitzgerald, K.J. Stol, Continuous Software Engineering and Beyond: Trends and Challenges, in: RCoSE 2014 Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, June 03, 2014, ACM, Hyderabad, India, pp. 1–9.

[40] B. Fitzgerald, K.J. Stol, Continuous software engineering: A roadmap and agenda, Journal of Systems and Software, Vol. 123, January, 2017, pp. 176–189.

[41] J. Forcier, Welcome to Fabric! — Fabric documentation, Fabric, verkkosivu. Saatavissa (viitattu 8.11.2018): http://www.fabfile.org/

[42] M. Fowler, CodeSmell, martinfowler.com, verkkosivu, February 9, 2006. Saatavissa (viitattu 4.1.2019): https://martinfowler.com/bliki/CodeSmell.html

[43] Git - Git Hooks, Git, verkkosivu. Saatavissa (viitattu 13.12.2018): https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

[44] GitLab Continuous Integration & Deployment | GitLab, GitLab, verkkosivu. Saatavissa (viitattu 26.10.2018): https://about.gitlab.com/product/continuous-integration/

[45] GitLab Continuous Integration & Deployment | GitLab, GitLab, verkkosivu. Saatavissa (viitattu 26.10.2018): https://about.gitlab.com/product/continuous-delivery/

[46] GitLab Pricing | GitLab, GitLab, verkkosivu. Saatavissa (viitattu 15.11.2018): https://about.gitlab.com/pricing/#self-managed

[47] M. Hüttermann, DevOps for Developers, Chapter 9: Infrastructure as Code, Apress, 2012.

[48] J. Holck, N. Jørgensen, Continuous Integration and Quality Assurance: a case study of two open source projects, Australasian Journal of Information Systems, Vol. 11, Iss. 1, 2003, pp. 40–53.

[49] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, D. Kieselhorst, Continuous Monitoring of Software Services: Design and Application of the Kieker Framework, Department of Computer Science, Kiel University, Germany, Technical Reports by Department of Computer Science TR-0921, Germany, 2009, 27 p. Saatavissa: http://oceanrep.geomar.de/id/eprint/14459

[50] J. Humble, J. Molesky, Why Enterprises Must Adopt Devops to Enable Continuous Delivery, Cutter IT Journal, Vol. 24, Iss. 8, August, 2011, pp. 6–12.

[51] Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Internet Engineering Task Force (IETF), Standards Track RFC 7231, 2014, 101 p. Saatavissa: https://tools.ietf.org/html/rfc7231

[52] J. Iden, B. Bygstad, The social interaction of developers and IT operations staff in software development projects, International Journal of Project Management, Vol. 36, Iss. 3, April, 2018, pp. 485–497.

[53] IEEE Xplore Search Results, IEEE, verkkosivu, 2018. Saatavissa (viitattu 18.10.2018): https://ieeexplore.ieee.org/search/searchresult.jsp?action=search&newsearch=true&searchField=Search_All&matchBoolean=true&queryText=(DevOps)&ranges=2017_2017_Year

[54] IEEE Xplore Search Results, IEEE, verkkosivu, 2018. Saatavissa (viitattu 18.10.2018): https://ieeexplore.ieee.org/search/searchresult.jsp?action=search&newsearch=true&searchField=Search_All&matchBoolean=true&queryText=(DevOps)&ranges=2016_2016_Year

[55] IEEE Xplore Search Results, IEEE, verkkosivu, 2018. Saatavissa (viitattu 18.10.2018): https://ieeexplore.ieee.org/search/searchresult.jsp?action=search&newsearch=true&searchField=Search_All&matchBoolean=true&queryText=(DevOps)&ranges=2015_2015_Year

[56] Index · Administration · Help · GitLab, Tampereen teknillinen yliopisto, verkkosivu. Saatavissa (viitattu 15.11.2018): https://gitlab.tut.fi/help/administration/index.md

[57] Index of /jobs/A-plus-test-MASTER/builds/94, plustest.cs.hut.fi, verkkosivu. Saatavissa (viitattu 4.1.2019): http://plustest.cs.hut.fi/jobs/A-plus-test-MASTER/builds/94/

[58] E. Isohanni, Uusi vuosi, uudet systeemit | TUT Pervasive Computing Blog, TUT Pervasive Computing Blog, verkkosivu, January 22, 2015. Saatavissa (viitattu 8.11.2018): https://pervasive.cs.tut.fi/?p=625

[59] Issue Board | GitLab, GitLab, verkkosivu. Saatavissa (viitattu 8.11.2018): https://about.gitlab.com/product/issueboard/

[60] Issues · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/issues

[61] R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer, What is DevOps?: A Systematic Mapping Study on Definitions and Practices, in: XP '16 Workshops, May 24, 2016, ACM, Edinburgh, Scotland, UK, pp. 12:1–12:11.

[62] Jenkins, Jenkins, verkkosivu. Saatavissa (viitattu 26.10.2018): https://jenkins.io/

[63] Jenkins installation and setup, Jenkins, verkkosivu. Saatavissa (viitattu 21.11.2018): https://jenkins.io/download/

[64] H.A. Johnson, Resource Review: Trello, Journal of the Medical Library Association, Vol. 105, Iss. 2, April, 2017, pp. 209–211.

[65] J. Kantojärvi, Future and roadmap of A+ LMS, Aalto yliopisto, verkkosivu, November 7, 2018. Saatavissa (viitattu 15.1.2019): https://apluslms.github.io/roadmap/presentation/

[66] V. Karavirta, P. Ihantola, T. Koskinen, Service-Oriented Approach to Improve Interoperability of e-Learning Systems, in: 2013 IEEE 13th International Conference on Advanced Learning Technologies, July 15–18, 2013, IEEE, pp. 341–345.

[67] M. Ketokivi, T. Choi, Renaissance of case research as a scientific method, Journal of Operations Management, Vol. 32, Iss. 5, July 14, 2014, pp. 232–240.

[68] T. Kilamo, M. Leppänen, T. Mikkonen, The Social Developer – Now, Then, and Tomorrrow, in: Proceedings of the 7th International Workshop on Social Software Engineering, September 1, 2015, ACM, Bergamo, Italy, pp. 41–48.

[69] H. Kniberg, M. Skarin, Kanban and Scrum - Making the Most of Both, Chapter 1: What is Scrum and Kanban anyway?, C4Media Inc., 2010, pp. 3–5.

[70] I. Kornilova, DevOps is a culture, not a role! – Irma Kornilova – Medium, Medium, verkkosivu, April 28, 2017. Saatavissa (viitattu 15.10.2018): https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

[71] T. Laukkanen, TTY:lle uusi tuotantotalouden ja rakentamisen tiedekunta | Sidosryhmälehti Rajapinta | Tampereen teknillinen yliopisto, Tampereen teknillinen yliopisto, verkkosivu, 2012. Saatavissa (viitattu 18.10.2018): http://www.tut.fi/rajapinta/artikkelit/2012/4/tty_lle-uusi-tuotantotalouden-ja-rakentamisen-tiedekunta

[72] T. Laukkarinen, K. Kuusinen, T. Mikkonen, Regulated software meets DevOps, Information and Software Technology, Vol. 97, May, 2018, pp. 176–178.

[73] Manifesto for Agile Software Development, agilemanifesto.org, verkkosivu, 2001. Saatavissa (viitattu 24.9.2018): http://agilemanifesto.org/

[74] M.A. McCarthy, L.M. Herger, S.M. Khan, B.M. Belgodere, Composable DevOps: Automated Ontology Based DevOps Maturity Analysis, in: 2015 IEEE International Conference on Services Computing, June 27 – July 2, 2015, IEEE, New York, NY, USA, pp. 600–607.

[75] D. Morris, S. Voutsinas, N. Hambly, R. Mann, Use of Docker for deployment and testing of astronomy software, Astronomy and Computing, Vol. 20, July, 2017, pp. 105–119.

[76] S. Neely, S. Stolt, Continuous Delivery? Easy! Just Change Everything (well, maybe it is not that easy), in: 2013 Agile Conference, August 5–9, 2013, IEEE, Nashville, TN, USA, pp. 121–128.

[77] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC | FAQ, OASIS, verkkosivu. Saatavissa (viitattu 23.11.2018): https://www.oasis-open.org/committees/tosca/faq.php

[78] M. Ohtsuki, K. Ohta, T. Kakeshita, Software Engineer Education Support System ALECSS Utilizing DevOps Tools, in: iiWAS '16 Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, November 28–30, 2016, ACM, pp. 209–213.

[79] Open Source Static Code Analysis | SonarQube, SonarSource S.A, verkkosivu. Saatavissa (viitattu 26.10.2018): https://www.sonarqube.org/about/

[80] Overview of Docker Compose | Docker Documentation, Docker Inc., verkkosivu. Saatavissa (viitattu 14.11.2018): https://docs.docker.com/compose/overview/

[81] M.Q. Patton, Qualitative Research & Evaluation Methods, 2nd edition, Sage Publications, Inc., 2002, 598 p.

[82] Periodic Table of DevOps Tools, XebiaLabs, Inc., verkkosivu, 2019. Saatavissa (viitattu 5.2.2019): https://xebialabs.com/periodic-table-of-devops-tools/

[83] Pipeline, Jenkins, verkkosivu. Saatavissa (viitattu 26.10.2018): https://jenkins.io/doc/book/pipeline/

[84] Projects · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/projects

[85] prp, GitHub, Inc., verkkosivu. Saatavissa (viitattu 10.12.2018): https://github.com/piehei/prplatform/projects/1

[86] prplatform/README.rst at cc39e3afb7cddc7c6e56bca6e9fae02ccae062a1 · piehei/prplatform, GitHub, Inc., verkkosivu. Saatavissa (viitattu 10.12.2018): https://github.com/piehei/prplatform/blob/cc39e3afb7cddc7c6e56bca6e9fae02ccae062a1/README.rst

[87] Pull Requests · Aalto-LeTech/a-plus, GitHub, Inc., verkkosivu. Saatavissa (viitattu 9.11.2018): https://github.com/Aalto-LeTech/a-plus/pulls?utf8=%E2%9C%93&q=is%3Apr+is%3Aclosed+base%3Amaster

[88] pyenv/pyenv: Simple Python version management, GitHub, Inc., verkkosivu. Saatavissa (viitattu 8.11.2018): https://github.com/pyenv/pyenv

[89] Releases | A+ LMS, GitHub Pages, verkkosivu. Saatavissa (viitattu 9.11.2018): https://apluslms.github.io/releases/

[90] Repolainen & GitLab – Repolainen User Manual For Students 1.1 documentation, Tampere University of Technology, verkkosivu. Saatavissa (viitattu 8.11.2018): https://tie-repolainen.rd.tut.fi/docs/student_manual/overview.html

[91] L. Riungu-Kalliosaari, S. Mäkinen, L.E. Lwakatare, J. Tiihonen, T. Männistö, DevOps Adoption Benefits and Challenges in Practice: A Case Study, in: PROFES 2016: Product-Focused Software Process Improvement, 2016, Springer International Publishing, pp. 590–597.

[92] Robot Framework, Robot Framework, verkkosivu. Saatavissa (viitattu 26.10.2018): http://robotframework.org/

[93] Selenium - Web Browser Automation, SeleniumHQ, verkkosivu. Saatavissa (viitattu 26.10.2018): https://www.seleniumhq.org/

[94] Sentry | Error Tracking Software — JavaScript, Python, PHP, Ruby, more, Functional Software, Inc., verkkosivu. Saatavissa (viitattu 29.10.2018): https://sentry.io/welcome/

[95] J. Smeds, K. Nybom, I. Porres, DevOps: A Definition and Perceived Adoption Impediments, in: XP 2015: Agile Processes in Software Engineering and Extreme Programming, 2015, Springer International Publishing, pp. 166–177.

[96] SQLite Foreign Key Support, SQLite, verkkosivu. Saatavissa (viitattu 8.11.2018): https://www.sqlite.org/foreignkeys.html

[97] D. Ståhl, J. Bosch, Modeling continuous integration practice differences in industry software development, Journal of Systems and Software, Vol. 87, January, 2014, pp. 48–59.

[98] Tietoturvakuvaus: Webropol-kyselypalvelut, Webropoll, Valkoinen paperi, May, 2017, 11 s. Saatavissa: https://webropol.fi/gdpr/Webropol_Tietoturvakuvaus_Whitepaper_FIN-20171207.pdf

[99] Travis CI - Test and Deploy with Confidence, TRAVIS CI, GMBH, verkkosivu. Saatavissa (viitattu 26.10.2018): https://travis-ci.com/

[100] J. P. Venttola, Versionhallinta opetuskäytössä, kandidaatintyö, Tampereen teknillinen yliopisto, Tieto- ja sähkötekniikan tiedekunta, Tietotekniikka, Tampere, April 4, 2018, 34 s. Saatavissa: https://dspace.cc.tut.fi/dpub/handle/123456789/25749

[101] M. Virmani, Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery, in: Fifth International Conference on the Innovative Computing Technology (INTECH 2015), May 20–22, 2015, IEEE, pp. 78–82.

[102] M. Walls, Building a DevOps Culture, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2013, 28 p.

[103] J. Wettinger, U. Breitenbücher, M. Falkenthal, F. Leymann, Collaborative gathering and continuous delivery of DevOps solutions through repositories, Computer Science - Research and Development, Vol. 32, Iss. 3–4, July, 2017, pp. 281–290.

[104] J. Wettinger, U. Breitenbücher, F. Leymann, DevOpSlang – Bridging the Gap between Development and Operations, in: ESOCC 2014: Service-Oriented and Cloud Computing, 2014, Springer, Berlin, Heidelberg, pp. 108–122.

[105] J. Wettinger, U. Breitenbücher, F. Leymann, Standards-Based DevOps Automation and Integration Using TOSCA, in: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, December 8–11, 2014, IEEE, London, UK, pp. 59–68.

[106] What is an End-to-End Test? - Definition from Techopedia, Techopedia Inc., verkkosivu. Saatavissa (viitattu 2.1.2019): https://www.techopedia.com/definition/7035/end-to-end-test

[107] R.K. Yin, Case Study Research: Design and Methods, 5th edition, SAGE Publications, Inc., 2014, 282 p.

[108] Zabbix Solutions, Zabbix LLC., verkkosivu. Saatavissa (viitattu 29.10.2018): https://www.zabbix.com/solutions

[109] M. Zasadziński, M. Solé, A. Brandon, V. Muntés-Mulero, D. Carrera, Next Stop "NoOps": Enabling Cross-System Diagnostics Through Graph-Based Composition of Logs and Metrics, in: 2018 IEEE International Conference on Cluster

Computing (CLUSTER), September 10–13, 2018, IEEE, Belfast, UK, pp. 212–222.

[110] P. Zimmerer, Strategy for continuous testing in iDevOps, in: ICSE '18 Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, May 27 – June 03, 2018, ACM, Gothenburg, Sweden, pp. 82–87.

LIITE A: KYSELYN KYSYMYKSET

DevOps-käytäntöjen hyödyntämisen kehittäminen opetusjärjestelmien kehityksessä ja ylläpidossa

Hei! Teen diplomityötä DevOps-käytäntöjen hyödyntämisen kehittämisestä opetus-/oppimisjärjestelmien kehityksessä ja ylläpidossa. Siihen liittyen teen kyselyn, jonka kohdeyleisönä ovat Tietotekniikan laboratorion henkilöt, jotka ovat jollakin tavalla mukana tällaisten järjestelmien (esimerkiksi A+) kehittämisessä, hallinnoinnissa tai ylläpidossa.

Kysely on avoinna perjantaihin 21.12.2018 kello 12:00 asti.

Vastaa erikseen jokaisen järjestelmän osalta, mutta vain kerran kutakin järjestelmää kohti.

1. Järjestelmä, jonka osalta vastaat *

- ◯ Repolainen
- ◯ A+ / Plussa
- ◯ Versionhallinta Gitillä
- ◯ Peer-Review / PRP
- ◯ Muu _____

2. Mikä seuraavista on roolisi järjestelmässä? *

- ☐ Ohjelmistosuunnittelija
- ☐ Ohjelmistokehittäjä
- ☐ Testaaja/laadunvalvoja
- ☐ Ylläpitäjä
- ☐ Vastuuhenkilö
- ☐ Muu _____

***Kuva A.1**. Kyselyn kysymykset, osa 1.*

3. Millainen tiimi tämän järjestelmän kehittämisessä ja ylläpidossa näkemyksesi mukaan on? (Esimerkiksi tiimin koko, jäsenten roolit)

4. Mikä seuraavista tahoista vastaa järjestelmän ylläpidosta? *

- ◯ Minä
- ◯ Joku muu tiimin jäsen
- ◯ Tiimi yhdessä
- ◯ Yliopiston IT-palvelut
- ◯ Ei kukaan/mikään
- ◯ Muu _____

5. Miten kehittäjien ja ylläpidon välinen kommunikaatio mielestäsi toimii?

6. Millainen julkaisuketju/-käytäntö järjestelmässä mielestäsi on? (Esimerkiksi manuaalinen komentoriviskriptin suoritus, automatisoitu CI-putki)?

Kuva A.2. *Kyselyn kysymykset, osa 2.*

7. Onko tämä mielestäsi toimiva ratkaisu?

- ◯ Kyllä
- ◯ Ei
- ◯ En osaa sanoa

8. Miksi ratkaisu on/ei ole toimiva?

9. Jos järjestelmän uudessa julkaisussa havaitaan ongelma sen käyttöönoton jälkeen, millainen mekanismi järjestelmässä on edellisen toimivan version uudelleenkäyttöönottamiseksi?

Kuva A.3. *Kyselyn kysymykset, osa 3.*

10. Mitä seuraavista käytännöistä järjestelmä hyödyntää? *

	Kyllä	Ei	En osaa sanoa
Jatkuva integraatio *	◯	◯	◯
Jatkuva testaus (esimerkiksi automatisoidut yksikkötestit, staattinen analyysi) *	◯	◯	◯
Jatkuva toimitus (toisin sanoen järjestelmän uusin asennuspaketti toimitetaan jatkuvasti ja automatisoidusti tuotantoympäristöön) *	◯	◯	◯
Jatkuva käyttöönotto (toisin sanoen järjestelmän uusin asennuspaketti otetaan jatkuvasti ja automatisoidusti käyttöön tuotantoympäristössä) *	◯	◯	◯
Jatkuva monitorointi/seuranta (toisin sanoen järjestelmän suoritusta valvotaan erinäisin apuvälinein, esimerkiksi lokeihin perustuvan visualisoinnin avulla) *	◯	◯	◯
Jatkuva suunnittelu (toisin sanoen järjestelmää suunnitellaan jatkuvasti esimerkiksi edellä mainittujen käytäntöjen hyödyntämisen ja käyttäjäpalautteen pohjalta) *	◯	◯	◯
Infrastruktruuri ohjelmakoodina (Infrastructure as Code, esimerkiksi järjestelmän suoritusympäristöjen määritykset koodina) *	◯	◯	◯

11. Jos järjestelmässä hyödynnetään muita käytäntöjä, mitä ne ovat?

_____

_____

_____

_____

_____

***Kuva A.4**. Kyselyn kysymykset, osa 4.*

12. Onko tällä sivulla listattujen käytäntöjen hyödyntämisessä mielestäsi kehitettävää järjestelmässä? *

- ◯ Kyllä
- ◯ Ei
- ◯ En osaa sanoa

13. Mitä työkaluja järjestelmässä käytetään jatkuvaan integraatioon?

14. Millä tasolla järjestelmää testataan (esimerkiksi yksikkötestit, staattinen analyysi)?

15. Mitä työkaluja järjestelmässä käytetään jatkuvaan testaukseen?

16. Mitä työkaluja järjestelmässä käytetään jatkuvaan toimitukseen?

***Kuva A.5**. Kyselyn kysymykset, osa 5.*

17. Mitä työkaluja järjestelmässä käytetään jatkuvaan käyttöönottoon?

18. Mitä työkaluja järjestelmässä käytetään jatkuvaan monitorointiin/seurantaan?

19. Mitä työkaluja järjestelmässä käytetään jatkuvaan suunnitteluun?

20. Mitä työkaluja järjestelmässä käytetään infrastruktuuriseen ohjelmakoodiin (esimerkiksi Docker ja Dockerfile)?

21. Jos vastasit edellisellä sivulla käytäntöihin Muu, mitä työkaluja tähän/näihin käytäntöön/käytäntöihin käytetään?

***Kuva A.6**. Kyselyn kysymykset, osa 6.*

22. Mikä järjestelmässä on mielestäsi hyvää sen kehittämisen kannalta?

23. Mikä järjestelmässä on mielestäsi hyvää sen ylläpidon kannalta?

24. Onko järjestelmän kehittämisessä mielestäsi ongelmia?

25. Onko järjestelmän ylläpidossa mielestäsi ongelmia?

26. Haluan varata juttutuokion *

Keskustelen aiheesta mielelläni kanssasi, jos haluat. Työpisteeni sijaitsee huoneessa TF105, jonne voit halutessasi tulla rupattelemaan. Myös samuli.kohomaki@tut.fi tavoittaa. Voit myös vastata alla olevaan kysymykseen Kyllä, jos haluat, että sovimme juttutuokion.

- ◯ Kyllä
- ◯ Ei

***Kuva A.7**. Kyselyn kysymykset, osa 7.*

Jos vastasit edelliseen kysymykseen Kyllä, täydennä nimesi ja sähköpostiosoitteesi, jotta voin ottaa sinuun yhteyttä. Voit antaa sähköpostiosoitteesi myös, jos olet tarvittaessa valmis vastaamaan mahdollisiin lisäkysymyksiin (diplomityön näkökulmasta).

Nimeä ja sähköpostiosoitetta ei käytetä muihin kuin edellä mainittuihin tarkoituksiin. Vastaajan nimi ja sähköpostiosoite säilytetään niin kauan kuin se on tarpeellista diplomityön kannalta. Vastauksia käsitellään tutkimuksessa anonyymisti. Täydentämällä nimesi ja sähköpostiosoitteesi ja painamalla "Lähetä" hyväksyt nimesi ja sähköpostiosoitteesi tallentamiseen.

Webropol tallentaa kyselyn vastaukset EU-alueelle eikä niitä luovuteta sen ulkopuolelle tai käsitellä sen ulkopuolella: https://webropol.fi/gdpr/Webropol_Tietoturvakuvaus_Whitepaper_FIN-20171207.pdf

27. Nimi ja sähköpostiosoite

Nimi	
Sähköposti	

***Kuva A.8**. Kyselyn kysymykset, osa 8.*