

Kalle Kurki

Julkaisuun liittyvän automaation kehittäminen

Insinööri

Tieto- ja viestintätekniikka

Kevät 2023



**KAMK • University
of Applied Sciences**

Tiivistelmä

Tekijä(t): Kurki Kalle

Työn nimi: Julkaisuun liittyvän automaation kehittäminen

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: ohjelmistojulkaisu, automaatio, ohjelmistokehitys, Jenkins, Jira, Bitbucket

Opinnäytetyö toteutettiin toimeksiantona Raute Oyj:lle. Työn aiheena oli tutkia toimeksiantajan julkaisuprosessia ja kartoittaa keinoja, joilla kyseistä prosessia voitaisiin kehittää. Tämän lisäksi tavoitteena oli luoda etenemissuunnitelma, jonka mukaisesti kartoituksessa löydetty kehitysratkaisut voitaisiin vaiheittain implementoida kehityskohteena olevaan julkaisuprosessiin.

Työn aikana tutustuttiin ohjelmistojulkaisuun ja käytiin vaihe vaiheelta läpi kehityskohteena oleva julkaisuprosessi. Tämän perusteella kartoitettiin alalla yleisesti käytössä olevia menetelmiä ja työkaluja jäljellä olevien manuaalivaiheiden automatisoimiseksi ja tutkittiin niiden soveltuvuutta toimeksiantajan käyttötar-koitukseen.

Työn lopuksi laadittiin etenemissuunnitelma, jossa työssä esitetyt kehityskeinot ja niiden toteuttamiseksi vaaditut toimenpiteet järjestettiin tiekarttaan, huomioon ottaen kehityskeinojen vaatimukset sekä niillä saavutettavat hyödyt. Laadittu etenemissuunnitelma tiivistää hyvin työssä tehdyn kartoituksen, tarjoten toimeksiantajalle selkeän suunnitelman, jonka avulla se pystyy lähteä kehittämään julkaisuprosessiaan yksi vaihe kerrallaan.

Abstract

Author(s): Kurki Kalle

Title of the Publication: Improving the Automation of Software Deployment

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: software deployment, automation, software development, Jenkins, Jira, Bitbucket

The thesis was commissioned by Raute Oyj. The purpose of the thesis was to study the client's deployment process and find ways to improve it. The aim was to develop a roadmap, that would allow the client to develop their deployment process step by step.

The thesis examined the software deployment process in general and explained the client's deployment process in detail. Based on this, commonly used methods and tools to automate the remaining manual steps were mapped and their suitability for the client's deployment process was investigated.

Finally, a roadmap was created to organise the methods presented in the study according to their requirements and benefits. The resulting roadmap illustrates well the presented development methods and their required actions and serves as a good guideline for implementing these steps to the client's deployment process.

Sisällys

1	Johdanto	1
2	Lähtökohdat.....	2
2.1	Tarpeellisuus ja tavoitteet	2
2.2	Kohdeorganisaation esittely	2
3	Ohjelmistojulkaisuprosessi	4
3.1	Lähdekoodi	5
3.2	Koostaminen.....	5
3.3	Testaus.....	6
3.4	Julkaisu ja käyttöönotto	7
4	Toimeksiantajan ohjelmistokehitysprosessi.....	8
4.1	Tehtävienhallinta	8
4.2	Lähdekoodin hallinta	9
4.3	Ohjelmistootomaatio	9
4.4	Julkaiseminen	11
4.4.1	Komponenttikohtainen versiointi ja muutostiedot.....	11
4.4.2	Julkaisupakettien siirtäminen	12
4.4.3	Julkaisu- ja muutostietojen täydentäminen.....	13
5	Julkaisuprosessin kehittämiskeinot.....	15
5.1	Julkaisupakettien siirtäminen	15
5.2	Muutostietojen automatisointi.....	17
6	Toteutussuunnitelma.....	22
7	Yhteenveto.....	24
	Lähteet	25

Termit ja lyhenteet

Automaatioputki	Määritelty joukko automaattisia tehtäviä, jotka suoritetaan yksi kerrallaan ohjelmistoautomaatiossa.
Bitbucket	Atlassian-tuoteperheeseen kuuluva lähdekoodin hallinnointiin tarkoitettu palvelu.
Confluence	Atlassian-tuoteperheeseen kuuluva wikialusta.
Git	Avoimeen lähdekoodiin perustuva hajautettu versionhallintajärjestelmä.
HTML	Nettisivujen rakenteen ja sisällön kuvaamiseen tarkoitettu merkintäkieli.
Jenkins	Avoimeen lähdekoodiin perustuva ohjelmistoautomaatioalusta.
Jira	Atlassian-tuoteperheeseen kuuluva projektihallintaohjelmisto.
Markdown	Yksinkertainen tekstipohjainen merkintäkieli.
Merkintäkieli	Tietokonekieli, joka koostuu helposti ymmärrettävistä avainsanoista nimistä tai tunnisteista.
Subversion	Avoimeen lähdekoodiin perustuva keskitetty versionhallintajärjestelmä.

1 Johdanto

Ohjelmistojulkaisuautomaatio on tärkeä osa nykyaikaista ohjelmistokehitystä, sillä se mahdollistaa nopean ja tehokkaan ohjelmiston julkaisun sekä toimittamisen käyttäjille. Sen avulla voidaan automatisoida manuaalisia ja usein toistuvia vaiheita, mikä vähentää inhimillisten virheiden mahdollisuutta ja vapauttaa vaiheisiin kulutetun ajan luovimpia tehtäviä varten.

Opinnäytetyön aiheena on tutkia työn tilaajan Raute Oyj:n Kajaanin yksikön julkaisuprosessia ja kartoittaa keinoja, joilla kyseistä prosessia voitaisiin kehittää. Työn tavoitteena on hakea moderneja automatisointimenetelmiä sekä työkaluja prosessin kehittämiseksi ja luoda niiden perusteella etenemissuunnitelma, jonka mukaan työn tilaaja voisi lähteä kehittämään prosessiaan yksi vaihe kerrallaan.

Työssä käsitellään ensimmäiseksi ohjelmistojulkaisua yleisellä tasolla ja käydään läpi vaiheet, joita se yleisimmin pitää sisällään. Tämän jälkeen käydään läpi vaihe vaiheelta kehitystyön kohteena oleva prosessi ja siihen liittyvät ohjelmistokehitystyökalut. Teoriaosuuden jälkeen esitellään kartoituksessa löydetty kehitysratkaisut. Valitut ratkaisut käydään läpi yksi kerrallaan, esittäen niiden ominaisuudet, rajoitteet sekä vaatimukset. Työn lopuksi esitellään etenemissuunnitelma, jonka avulla esitetyt kehitysratkaisut voidaan implementoida osaksi toimeksiantajan prosessia vaihe kerrallaan.

2 Lähtökohdat

2.1 Tarpeellisuus ja tavoitteet

Ohjelmistokehitys on aikaa vievä prosessi, sillä se sisältää useita eri työvaiheita. Ensimmäiseksi on ymmärrettävä ongelma, jota ollaan ratkaisemassa, suunnitella arkkitehtuuri, valita käytettävät työkalut sekä toteuttaa tarvittavat koodimuutokset. Tämä voi kestää tapauksen riippuen muutamasta tunnista jopa kuukausiin. Lisäksi näissä tehtävissä vaaditaan luovuutta, minkä ansiosta ne pitävät työntekijän mielenkiintoa yllä tarjoamalla työhön sopivasti haastetta sekä vaihtelevuutta. [1.]

Varsinaisen kehitystyön jälkeen muutokset täytyy vielä viedä julkaisuprosessin läpi, mikä sisältää useita vaiheita, kuten ohjelmiston koostamisen lähdekoodista, testaamisen sekä ohjelmiston paketoinnin ja toimittamisen tuotantoon [2]. Prosessin vaiheet toistuvat rutiininomaisesti kerta toisensa jälkeen, mistä syystä nämä voidaan mieltää useimmiten tylsiksi ja henkisesti kuluttaviksi. Julkaisuprosessin vaiheita voidaan automatisoida, jolloin manuaalisiin vaiheisiin käytetty aika vapautuu muille tehtäville. Näin työn kuormittavuus pienenee, minkä lisäksi myös ohjelmiston laatu paranee, sillä automatisointi poistaa ihmisten tekemien virheiden mahdollisuuden sekä takaa, että julkaisuprosessi pysyy samanlaisena kerta toisensa jälkeen. [1.]

Työn toimeksiantajan julkaisuprosessi on jo osittain automatisoitu, mutta sisältää vielä useita manuaalisesti tehtäviä vaiheita, jotka kuluttavat turhaan kehittäjien aikaa ja tekevät prosessin alttiiksi virheille. Työn tarkoituksena on kartoittaa useimmiten toistuvat ja eniten aikaa vievät vaiheet, etsiä keinoja niiden automatisoimiseksi ja tarjota toimeksiantajalle etenemissuunnitelma, jonka mukaan julkaisun automatisointia voidaan lähteä lisäämään vaihe kerrallaan.

2.2 Kohdeorganisaation esittely

Työn tilaajana toimii Raute Oyj:n Kajaanin yksikkö. Raute on suomalainen teknologia- ja palveluyritys, joka tuottaa koko tuotantoprosessin kattavia ratkaisuja LVL-, vaneri- sekä viiluteollisuudelle. Raute on maailmanlaajuinen markkinajohtaja vaneriteollisuudessa, minkä lisäksi yrityksellä on myös vahva asema LVL-teollisuudessa, sillä yli puolet maailmalla tuotetusta LVL:sta tuotetaan yrityksen toimittamilla koneilla. [3.]

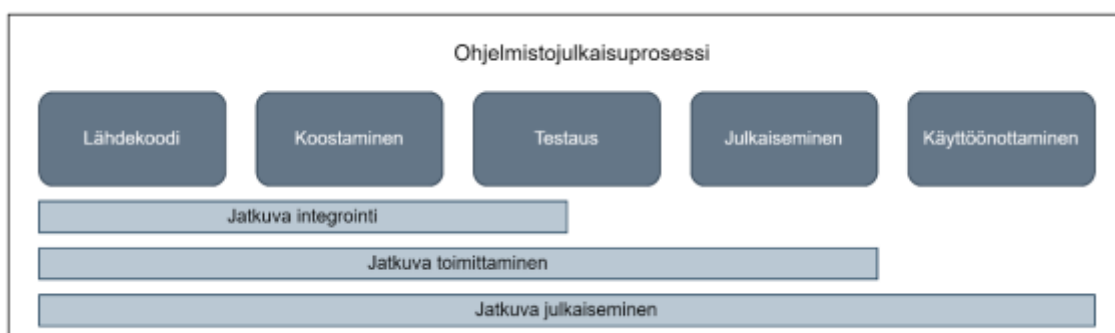
Raute-konsernin päätoimipiste sijaitsee Lahdessa, ja lisäksi sillä on Suomessa toimipisteitä Kajaanissa sekä Oulussa. Näiden lisäksi yrityksellä on tuotantoyksiköitä sekä huolto- ja palvelupisteitä eri puolilla maailmaa. Kokonaisuudessaan yritys työllistää maailmanlaajuisesti yli 800 työntekijää. Kajaanissa toiminta keskittyy älykkäiden konenäkö- ja kosteusanalysaattoreiden kehittämiseen ja valmistamiseen. [3.]

3 Ohjelmistojulkaisuprosessi

Ohjelmistojulkaisu on yksi tärkeimmistä ohjelmistokehitysprosessin osa-alueista, sillä sen avulla tehdyt ohjelmistomuutokset saadaan vietyä kehittäjiltä ohjelmiston käyttäjille. Lisäksi käytettävät menetelmät vaikuttavat suoraan siihen, kuinka nopeasti tuote pystyy vastaamaan asiakkaan mieltymyksiin sekä vaatimuksiin. [4.]

Julkaisuprosessin ensimmäisessä vaiheessa lähdekoodi käännetään konekieliseksi ohjelmistoksi ja paketoidaan yhdessä muiden ohjelmiston toiminnan kannalta kriittisten tiedostojen kanssa julkaisupaketiksi [4]. Tämän jälkeen julkaisupaketille suoritetaan ennalta määritetyt testit, joilla varmistetaan sen oikeaoppinen toimivuus. Kun ohjelmistopaketti on läpäissyt testit hyväksytysti, siirytään lopulta julkaisuvaiheeseen, jossa määritetään julkaisulle versionumero ja täydennetään julkaisutiedot. Tämän jälkeen julkaisu on valmis otettavaksi käyttöön ohjelmiston kohdeympäristöissä. [5.]

Julkaisuprosessin vaiheet voidaan suorittaa manuaalisesti tai automatisoidusti. Useimmat IT-organisaatiot käyttävät julkaisemisprosessissaan automaattisten ja manuaalisten vaiheiden yhdistelmää [4]. Julkaisun automatisointiasteen voi jakaa kolmeen eri luokkaan automatisoitujen vaiheiden perusteella. Kuva 1 esittää näitä kolmea eri automaatioastetta ja niissä automatisoitavia julkaisuprosessin vaiheita.



Kuva 1. Ohjelmistojulkaisuprosessin vaiheet [6].

Jatkuvan integroinnin mallissa kehittäjät yhdistävät tekemänsä ohjelmistomuutokset päähaaraan päivittäin, minkä jälkeen niille suoritetaan automaattisesti koostamis- ja testausvaiheet. Tämä nopeuttaa kehitystyötä, sillä muutosten vaikutusta ohjelman toimintaan testataan jatkuvasti, minkä ansiosta ongelmat saadaan paikannettua ja korjattua aikaisemmassa vaiheessa, jolloin se on vielä helpompaa kuin jos yhdistäminen tehtäisiin vasta kehitystyön loppuun. [7.]

Jatkuvan toimituksen mallissa lähdekoodi pyritään pitämään koko ajan julkaisukelpoisena. Käytännössä malli on jatkuvan integroinnin laajennus, missä julkaisupaketti valmistellaan automaattisesti käyttöönottoa varten. Jatkuva toimitus takaa, että ohjelmistosta on jatkuvasti ajantasainen ohjelmistojulkaisupaketti, joka on milloin tahansa otettavissa käyttöön. [2.]

Jatkuvan julkaisun mallissa julkaisuprosessi on täysin automatisoitu. Tämä tarkoittaa, että kaikki lähdekoodimuutokset koostetaan, testataan, julkaistaan sekä viedään tuotantoon täysin automaattisesti. Tämä on nopein mahdollinen julkaisumalli, mutta virheet ohjelmistossa voivat aiheuttaa ongelmia ja mahdollisia käyttökatkoksia, minkä takia sen käyttäminen vaatii erittäin kehittynyttä valvontakulttuuria, päivitysvalmiutta sekä kykyä palauttaa järjestelmä aikaisempaan tilaan. [2.]

3.1 Lähdekoodi

Ohjelmiston lähdekoodia säilytetään versiohallintajärjestelmässä, mikä mahdollistaa muun muassa versioinnin, muutosten seurannan sekä liitettävyyden muiden työkalujen kanssa. Versiohallinnan liitettävyyden merkitys on automatisoinnin kannalta suuri, sillä sen avulla julkaisuprosessi voidaan asettaa alkamaan automaattisesti, kun lähdekoodissa havaitaan muutoksia. Tämä mahdollistaa julkaisuvälin kaventumisen, jolloin tehdyt muutokset saadaan nopeammin tuotantoon. Myös ohjelmointivirheiden paikantaminen helpottuu, kun uusien muutosten määrä testiajoissa vähenee. [9.]

Kehityksen aikana muutokset tehdään useimmiten versiohallinnan päähaarasta erotettuun kehityshaaraan, mikä mahdollistaa ominaisuuksien kehittämisen ja testaamisen ilman pelkoa toimivan ohjelman rikkomisesta [9]. Kun muutokset on todettu toimiviksi, liitetään ne myös päähaaraan, mikä aloittaa yleensä varsinaisen ohjelmistojulkaisuprosessin [8].

3.2 Koostaminen

Koostaminen on julkaisuprosessin ensimmäinen vaihe, jossa versiohallinnassa olevasta lähdekoodista muodostetaan ohjelmistopaketti. Yksi koostamisen olennaisimmista vaiheista on ohjelmiston kääntäminen, jossa lähdekoodimuodossa oleva ohjelmisto muunnetaan kohdejärjestelmässä

suoritettavissa olevaksi ohjelmistoksi. Jotta ohjelmistoa voitaisiin käyttää sellaisenaan kohdeympäristössä, on siihen sisällytettävä kaikki ohjelmiston ajamiseen tarvittavat tiedostot. Näitä ovat itse sovellusten lisäksi esimerkiksi erilaiset asennusskriptit sekä sovellusten metatiedot. Välttämättömien tiedostojen lisäksi pakettiin voidaan sisällyttää myös esimerkiksi ohjelmiston käyttöön liittyvää dokumentaatiota sekä lisenssitietoja. [10.]

Ohjelmistopakettien muodostamisen lisäksi koostamisvaiheessa tehdään usein myös erilaisia toimenpiteitä ohjelmistolaadun varmistamiseksi. Näitä ovat esimerkiksi ohjelmiston testaaminen yksikkötestien avulla sekä koodin turvallisuuden ja hyvien ohjelmointikäytänteiden toteutumisen valvominen erilaisten koodianalysointityökalujen avulla. [10.]

3.3 Testaus

Testauksella pyritään ylläpitämään julkaistavan ohjelmiston laatua varmistamalla, että tehdyt muutokset toimivat odotetulla tavalla eivätkä ne aiheuta virheitä ohjelmiston toiminnalle [11]. Ohjelmiston testaus sisältää usein sekä manuaalisesti että automaattisesti suoritettavia testejä. Manuaalitestauksessa testaaja suorittaa käsin eri ohjelmiston toiminnallisuuden tarkistuksia ja tarkistaa, että ohjelmiston toiminta vastaa odotuksia. Automaattinen testaus puolestaan toteutetaan erilaisten ohjelmistojen ja työkalujen avulla, jotka suorittavat skriptimuodossa määritetyt testit täysin automaattisesti. Automaattisten testaaminen on nopeampaa ja tehokkaampaa kuin manuaalisten, sillä tietokone kykenee suorittamaan testitapauksia ihmistä nopeammin. [11.] Kaikkia testejä ei kuitenkaan voida automatisoida, sillä tiettyjen toimintojen testaaminen vaatii ihmissilmää. Lisäksi kaikkia testejä ei ole taloudellisesti kannattava automatisoida. Tällaisia ovat esimerkiksi testitapaukset, jotka testaavat ominaisuuksia, jotka ovat aikaisessa kehitysvaiheessa tai joiden vaatimusmäärittelyt muuttuvat usein. [12.]

Automaattitestit voidaan jakaa kolmeen eri pääalueeseen: yksikkötestaus, integraatiotestaus ja hyväksyntätestaus [9]. Yksikkötestien tarkoituksena on varmistaa, että sovelluksen jokainen pienin osa, yksikkö, toimii suunnitellulla tavalla. Tämä yksikkö voi olla esimerkiksi yksittäinen komponentti, luokka tai funktio. Yksikön toimintaa testataan syöttämällä sille testidataa ja varmistamalla, että palautettu arvo on halutunlainen. Useimmiten yksikkötestit suoritetaan jo ohjelmiston koostamisvaiheessa. Integraatiotestaus puolestaan testaa komponenttien välistä yhteistoimintaa. Sen avulla pyritään varmistamaan, että eri osat ohjelmasta toimivat yhdessä odotetulla tavalla ja että niiden väliset rajapinnat toimivat suunnitellusti. Hyväksyntätestaus puolestaan testaa

ohjelmiston kokonaisuutta ja sen toimintaa ohjelmiston loppukäyttäjän näkökulmasta. Tämä auttaa varmistamaan, että ohjelmisto vastaa asetettuja vaatimuksia ja että se toimii odotetulla tavalla käytännössä. [11.]

3.4 Julkaisu ja käyttöönotto

Ohjelmistojulkaisupaketti on valmis julkaistavaksi, kun se on koostettu ja testattu hyväksytysti. Paketille määritetään yksilöllinen versionumero, jonka avulla se pystytään erottamaan toisista versioista [13]. Tämän lisäksi julkaisun muutokset dokumentoidaan muutos- ja julkaisutietoihin [2].

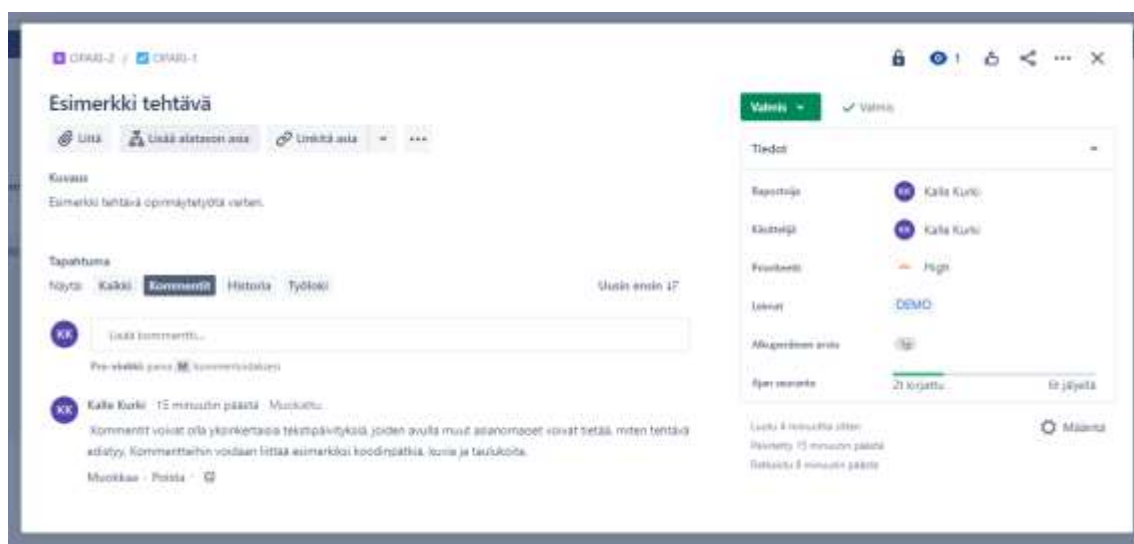
Kun kaikki ennalta määritetyt ohjelmistojulkaisuprosessin vaiheet on suoritettu, voidaan julkaisu toteuttaa käytettävän automatisointimallin mukaisesti. Jatkuva julkaisua käyttäessä, julkaisu viedään tuotantoympäristöön automaattisesti. Kun taas jatkuvan toimituksen mallissa julkaisupaketti ainoastaan valmistellaan manuaalisesti tehtävää käyttöönottoa varten [2].

4 Toimeksiantajan ohjelmistokehitysprosessi

4.1 Tehtävienhallinta

Toimeksiantajalla on käytössään Atlassian-yhtiön tuoteperheeseen kuuluva Jira-projektinhallintajärjestelmä, joka sisältää monipuolisesti ominaisuuksia aina vaatimusten ja testitapausten hallinnoinnista ketterään ohjelmistokehitykseen [14]. Ohjelmistojulkaisun kannalta projektinhallintajärjestelmän olennaisin ominaisuus on tehtävienhallinta, sillä sen avulla määritellään ja seurataan kaikkea tehtävää työtä.

Jokaiselle yksittäiselle tehtävälle, kuten kehitettävälle ominaisuudelle tai korjattavalle ohjelmistovirheelle, luodaan tehtävisivu, jonka avulla sen etenemistä voidaan seurata. Jira luo automaattisesti jokaiselle tehtävälle uniikin avaimen, jota käyttämällä tehtävään pystytään helposti viittamaan ohjelmistokehitysprosessin eri vaiheissa. Tämän lisäksi tehtävisivulle kirjataan myös erilaisia tehtävään liittyvää tietoa, kuten tehtävän kuvaus, prioriteetti sekä arvio tehtävään kuluva ajasta. [15.] Kuva 2 havainnollistaa tehtävisivun näkymää.



Kuva 2. Kuvakaappaus Jira-tehtävästä.

Jira-ohjelmisto voidaan yhdistää useiden eri kehitystyökalujen kanssa, minkä avulla kehitystyön etenemistä pystytään seuraamaan suoraan tehtävisivun kautta. Ohjelmistoon voidaan liittää esimerkiksi versiohallinnan tai ohjelmistootomaation työkalut, jolloin tehtävisivulle päivittyy automaattisesti tietoa esimerkiksi lähdekoodiin tehdyistä muutoksista sekä julkaisuprosessin eri vaiheista. [16.]

4.2 Lähdekoodin hallinta

Toimeksiantajalla on käytössä tällä hetkellä Subversion-versionhallintajärjestelmä eli SVN, mutta tarkoituksena on tulevaisuudessa siirtyä kokonaan monipuolisemman Git:n käyttöön. Git ja SVN ovat molemmat avoimeen lähdekoodiin perustuvia versiohallintajärjestelmiä. Keskeisin eroavaisuus näiden kahden välillä on niiden lähestymistapa lähdekoodin hallintaan. SVN käyttää keskitettyä arkkitehtuurimallia, kun taas Git hajautettua. Keskitetyssä versionhallinnassa palvelimelta kopioidaan kehittäjälle lähdetiedostot ilman versiohistoriaa ja kehitystyön jälkeen muutokset yhdistetään suoraan palvelimella sijaitsevaan versioon, minkä vuoksi muutosten integrointi pääversioon vaatii aina verkkoyhteyden palvelimeen. Hajautetussa mallissa jokaiselle käyttäjälle kopioidaan täydellinen kopio versiohallinnan tietovarastosta, mikä sisältää myös tämän versiohistorian kokonaisuudessaan. Tämä poistaa tarpeen muodostaa jatkuvasti yhteys palvelimeen ja mahdollistaa muutosten integroimisen pääversioon lokaalisti ennen muutosten kopioimista palvelimelle. [17.]

Git-versiohallintajärjestelmään ollaan siirtymässä sen paremman integroituvuuden ja monipuolimpien ominaisuuksien takia. Siirtyminen mahdollistaa esimerkiksi Atlassianin Bitbucket -palvelun käyttämisen, mikä avaa useita hyödyllisiä ominaisuuksia, kuten muutosten katselmoinnin ja kehityshaarojen luonnin suoraan Jiran käyttöliittymän kautta. Versiohallinnan vaihtamista puoltaa myös ilmenneet ohjelmistoautomaation ongelmat Subversionin tukemiseen liittyen.

Prosessi versiohallintajärjestelmän vaihtamiseksi on jo aloitettu. Henkilöstöä on koulutettu uuden järjestelmän käytöstä ja sen integrointi osaksi muuta ohjelmistokehitysympäristöä on aloitettu. Toimeksiantajalla on jo käytössä BitBucket-palvelu, jonne nykyinen tietovarasto synkronoidaan automaattisesti ohjelmistoautomaatiolla. Tämän ansiosta tikettiä koskevat muutokset ovat jo nähtävissä Jiran tehtävisivulta, mikäli muutosviesteissä käytetään tehtävän yksilöllistä tunnusta. Vaikka työ versiohallintajärjestelmän vaihtamiseksi on jo aloitettu, niin ei siirtyminen ole vielä aivan ajankohtaista, sillä uuden versiohallintajärjestelmän integroiminen ohjelmistoautomaatioon ja Jira-ohjelmistoon vaatii vielä reilusti työtä.

4.3 Ohjelmistoautomaatio

Kehitystyön kohteena olevan julkaisuprosessin automatisoidut vaiheet ovat toteutettu Jenkins-ohjelmistoautomaatioalustalla. Jenkins on avoimen lähdekoodin ohjelmistoautomaatiopalvelin,

jolla pystytään automatisoimaan kaikki julkaisuprosessin vaiheet, kuten koostaminen, testaaminen sekä julkaisu. Siihen on saatavilla satoja eri lisäosia, joita käyttämällä automaatiopalvelimen toiminta pystytään räätälöimään automatisoitavan projektin vaatimusten mukaiseksi. [18.] Lisäosien avulla Jenkins pystytään myös liittämään muihin jatkuvan integraation ja toimituksen työkaluihin, mikä mahdollistaa esimerkiksi tehtävien julkaisutilan seuraamisen suoraan Jiran tehtäväisivun kautta [19].

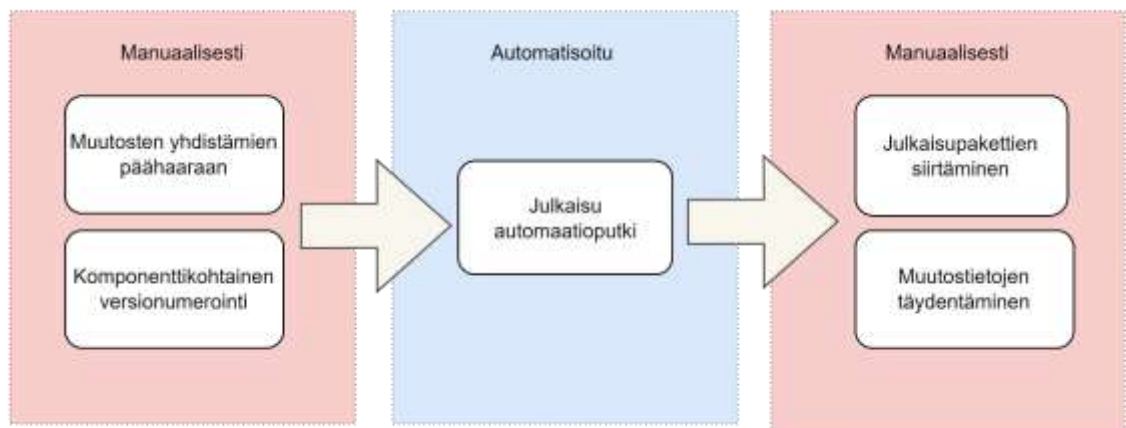
Toimeksiantajan ohjelmistoautomaatiossa on kahta erilaista automaatioputkea, joiden läpi lähdekoodiin tehdyt muutokset viedään. Toiset näistä ovat integrointiputkia, jotka suoritetaan välittömästi, kun muutoksia yhdistetään kehityshaarasta julkaisu- tai päähaaraan. Näissä putkissa varmistetaan ohjelmiston oikeaoppinen toiminta koostamalla ja testaamalla lähdekoodi. Kehittäjät yhdistävät haaroihin muutoksia useita kertoja päivässä, mikä tarkoittaa sitä, että myös integrointiputkia ajetaan useita kertoja vuorokaudessa. Integraatioputkien lisäksi on julkaisuputkia, jotka ajetaan ajastetusti yöaikaan, jolloin automaatiopalvelimella on vähiten kuormaa. Tällä tavoin ohjelmistojulkaisu ei häiritse muuta ohjelmistokehitystyötä hidastamalla koodin integroimista.

Julkaisuputket eroavat integrointiputkesta ainoastaan muutamalla lisätoimenpiteellä, joilla varmistetaan, että lopputuloksena saadaan julkaisua varten täysin puhdas ohjelmistopaketti. Koostamisvaiheessa lähdekoodi ladataan versiohallinnan tietovarastosta kokonaisuudessaan automaatiopalvelimelle pelkän päivittämisen sijasta ja kääntäminen suoritetaan lisäparametrein. Näillä toimenpiteillä varmistetaan, ettei koostettava lähdekoodi sisällä ylimääräisiä tiedostoja aikaisemmilta suorituskerroilta ja ohjelmistopaketti koostetaan joka kerta täysin samalla tavalla. Lopuksi molemmat automaatioputket tallentavat hyväksytysti koostetut ja testatut ohjelmistopaketit putkikohtaiseen hakemistoon, josta ne on julkaistavissa manuaalisesti tehtävillä toimenpiteillä.

Toimeksiantajan julkaisuprosessin automaatioaste täyttää jatkuvan integroinnin mukaiset kriteerit, sillä pääversioon yhdistetään muutoksia päivittäin, minkä lisäksi ohjelmiston oikeaoppinen toiminta testataan aina muutosten yhdistämisen jälkeen ohjelmistoautomaation avulla. Julkaisuprosessi ei kuitenkaan täytä vielä jatkuvan toimituksen määritelmää, sillä julkaisu sisältää manuaalisesti tehtäviä vaiheita. Toimeksiantajan tavoitteena on kuitenkin tulevaisuudessa automatisoida kaikki nämä vaiheet, jolloin ohjelmiston julkaisu tapahtuu täysin jatkuvan toimittamisen mallin mukaisesti.

4.4 Julkaiseminen

Vaikka ohjelmistojulkaisuprosessin monimutkaisimmat ja työläimmät vaiheet eli koostaminen ja testaaminen on jo automatisoitu, niin siinä on vielä manuaalisia vaiheita, jotka käsin tehtynä vievät kehittäjien aikaa tärkeämmiltä tehtäviltä ja altistavat prosessin virheille. Kyseiset vaiheet sijoittuvat sekä ennen että jälkeen julkaisuautomaatiota. Ennen julkaisuputkea tapahtuu komponenttikohtaisten versionumeroiden sekä muutostietojen päivittäminen, ja automaation jälkeen julkaisutietojen täydentäminen sekä automaatioputkessa koostetun ohjelmiston julkaiseminen. Kuva 3 havainnollistaa manuaalivaiheiden sijoittumista suhteessa julkaisuputken suoritukseen.



Kuva 3. Toimeksiantajan manuaalivaiheet julkaisuprosessissa

4.4.1 Komponenttikohtainen versiointi ja muutostiedot

Komponenttikohtaiset versionumerot ja muutostiedot päivitetään muutosten päähaaraan yhdistämisen yhteydessä. Kyseisessä vaiheessa jokaisen muutosta koskevan komponentin versionumeroa nostetaan ja komponenttiin kohdistuneet muutokset kirjataan lähdekoodissa oleviin tiedostoihin, jotka sijaitsevat komponenttikohtaisissa juurihakemistoissa. Tämän vaiheen jälkeen lähdekoodimuutokset yhdistetään päähaaraan, minkä jälkeen se käy läpi integraatio- ja julkaisu-putket ennen muita manuaalisia vaiheita.

Komponenttikohtainen versiointi mahdollistaa muutosten seuraamisen komponenttitasolla ja helpottaa eri komponenttiversioiden yhteensopimattomuusongelmista johtavien virheiden paikantamista. Komponenttikohtaisesta versioinnista on hyötyä etenkin silloin, kun ohjelmiston osat

ovat löyhästi kytkettyinä toisiinsa ja ohjelmisto tukee eri versioisten komponenttien yhdistelmistä. [20.] Tämä edellyttää kuitenkin komponenttien kattavaa yksikkö- ja integraatiotestauksista, jolla varmistetaan, ettei tehdyillä muutoksilla ole vaikutusta muiden komponenttien toimintaan.

Toimeksiantaja on lähitulevaisuudessa luopumassa kokonaan komponenttikohtaisesta versioinnista, sillä se on todettu tarpeettomaksi. Vaikka yrityksen ohjelmistot onkin jaettu pienempiin komponentteihin, niin julkaisun kannalta ohjelmistoja käsitellään aina suurempina kokonaisuuksina ja ne koostetaan ja testataan kokonaisuuksina ohjelmistootomaatioissa. Koska testaamista ei toteuteta juurikaan integraatio- ja yksikkötasolla, niin ei voida olla varmoja ohjelmiston oikeaoppisesta toiminnasta eri komponenttiversioyhdistelmillä. Tämän takia ohjelmistojulkaisupaketit viedään aina tuotantolaitteisiin kokonaisuudessaan, eikä yksittäisten komponenttien päivittämistä tueta. Sen tukeminen vaatisi testauksen laajentamista merkittävästi, eikä tämä toisi merkittävää hyötyä vaivaansa nähden.

4.4.2 Julkaisupakettien siirtäminen

Julkaisuautomaatio koostaa ohjelmistojulkaisupaketin jokaista ohjelmistoa ja sen kohdeympäristön käyttöjärjestelmää kohden. Nämä paketit tallennetaan palvelimen hakemistoon, jossa säilytetään ohjelmistopaketteja useilta eri automaatioputken suorituskerroilta. Julkaisuvaiheessa julkaisuputken koostamat ohjelmistopakettit siirretään varsinaiseen julkaisuhakemistoon, jossa säilytetään vain viimeisintä julkaistua versiosta ohjelmistosta. Manuaalisesti tehtynä tämä vaihe on hyvin altis virheille, sillä se sisältää paljon toistoa. Siirrettäviä tiedostoja on monia, jolloin osasta niistä voi tekijän epähuomiossa päätyä väärään kansioon tai jopa unohtua kokonaan.

Jukkapekka Lalu on opinnäytetyönsä esimerkkiprojektina automatisoinut tämän vaiheen. Automaatiota varten on luotu Jenkins-skripti, joka kutsuttaessaan siirtää parametreina määritellyt tiedostot julkaisuhakemistoihin. Kyseistä skriptiä kutsutaan julkaisuputken aikana, mikäli automaatio havaitsee ohjelmistossa muutoksia [21]. Ongelmien takia kyseistä automatisointia ei ole vielä pystytty ottamaan käyttöön. Ilmenneet ongelmat liittyvät testausautomaation tulosten tulkin-
taan ja sääntöihin, joiden perusteella kyseistä automaatiota kutsutaan. Tämänhetkinen toteutus ei tunnista kaikkia testiautomaation tilanteita, minkä takia julkaisupaketteja ei siirretä aina kun tarvittaisiin. Jotta automaatio voitaisiin ottaa käyttöön, täytyisi edellä mainittu ongelma korjata.

4.4.3 Julkaisu- ja muutostietojen täydentäminen

Muutos- ja julkaisutiedot ovat dokumentteja, joihin kirjataan tietoja ohjelmistoon tehdyistä ominaisuuksista ja korjauksista. Kyseisten dokumenttien avulla tieto ohjelmiston muutoksista saadaan viestittyä kohdeyleisölle. Muutos- ja julkaisutiedot yhdistetään usein virheellisesti yhdeksi samaksi asiaksi, ja vaikka niiden pääperiaate onkin hyvin samankaltainen, niin eroavat ne kuitenkin sisällöllisesti toisistaan. Julkaisutiedot on tarkoitettu eritoten ohjelmiston loppukäyttäjiä varten ja ne sisältävät useimmiten pelkästään sanallisen kuvauksen päivityksen tuomista muutoksista. Muutostiedot sen sijaan on tarkoitettu enemmänkin ohjelmistoa kehittäväälle tiimille ja ne ovat useimmiten kirjoitettu vähemmän verbaalisessa muodossa julkaisutietoihin verrattuna. Perustietojen lisäksi muutostietoihin sisällytetään usein myös sellaisia teknisiä yksityiskohtia, joilla ei ole ohjelmiston loppukäyttäjän näkökulmasta mitään merkitystä. [22.]



Toimeksiantaja koostaa versiokohtaiset julkaisutiedot automaattisesti osana julkaisuautomaatioita. Julkaisutietoihin sisällytetään lista kyseiseen versioon linkitetyistä tehtävistä ja se tallennetaan samaan hakemistoon ohjelmistojulkaisupakettien kanssa. Muutostiedot puolestaan täydennetään täysin manuaalisesti. Ohjelmistomuutosten tiedot kirjataan Atlassianin Confluence -palvelussa sijaitsevalle wikisivulle, jonne jokaista julkaisua kohden kirjataan muutoksia koskeva Jira-tehtävä, julkaisupäivämäärä, muutoksia koskettavat komponentit sekä muutoksista vastaava henkilö. Kuva 4 havainnollistaa julkaisun yhteydessä tehtävää muutostietomerkinä. Ensimmäiselle riville merkitään julkaisupäivämäärä ja muutosten tekijä. Tämän jälkeen listataan komponentit versionumeroineen, joihin muutokset kohdistuvat. Komponenttien versionumerot koostuvat kolmesta eri osasta: Pääversionumerosta, komponenttikohtaisesta versionumerosta sekä julkaisupaketit erottavasta koostamisnumerosta. Esimerkiksi versionumeron ollessa 1.20.3_42420, pääversion osuutta vastaa luku 1.20, komponenttikohtaista versionumeroa numero kolme ja koostamisnumeron osuutta alaviivalla erotettu 42420.

1. jouluk. 2022

@Kalle Kurki

komponentti_A (Rel 1.20.3_42420)

komponentti_B (Rel 1.20.4_42420)

-  OPARI-1: Esimerkki tehtävä VALMIS
-  OPARI-3: Toinen esimerkki tehtävä VALMIS

Kuva 4. Esimerkki toimeksiantajan käyttämästä muutostietojen formaatista.

Yleensä kehittäjät aloittavat muutostietojen päivittämisen samalla, kun yhdistävät lähdekoodi-muutokset päähaaraan. Tässä vaiheessa koodimuutokset tarkastetaan vielä viimeisen kerran ennen julkaisemista mahdollisten yhdistämisristiriitojen varalta. Tällöin kehittäjän on myös helpoin listata komponentit, joihin muutokset ovat kohdistuneet. Muutostietojen päivittämistä ei kuitenkaan voida saattaa loppuun ennen ohjelmistojulkaisupaketin valmistumista, sillä versionumeroiden koosteosuus määräytyy vasta julkaisuautomaation aikana. Tämän takia kehittäjän on vielä palattava päivittämään komponenttien versionumerot muutosten käytyä julkaisuputken lävitse.

5 Julkaisuprosessin kehittämiskeinot

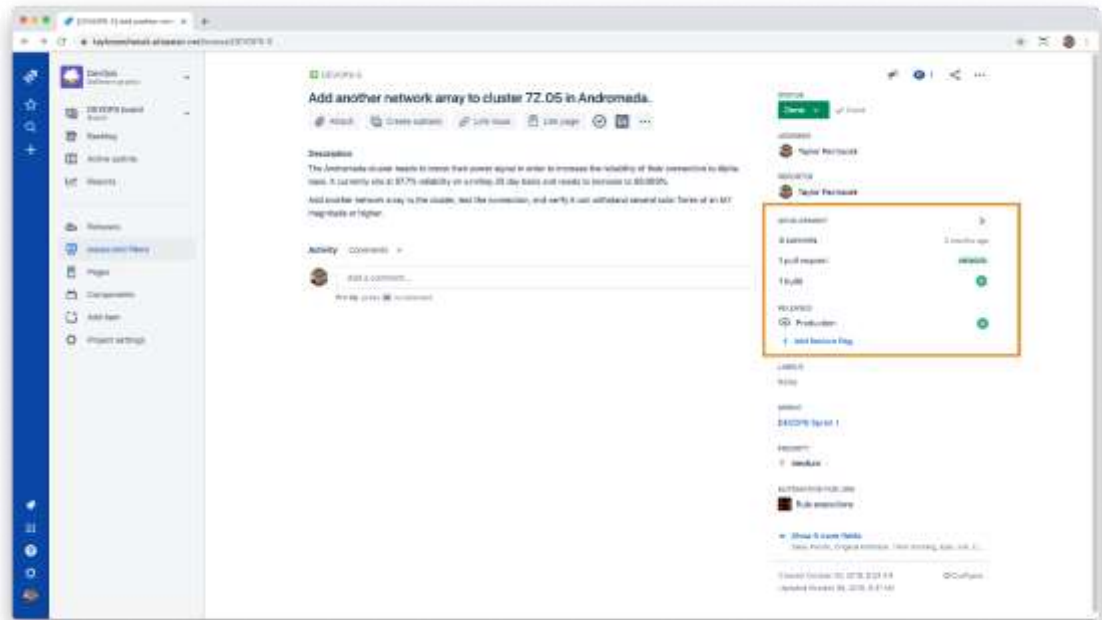
Tässä osiossa käsitellään ratkaisuja, joilla julkaisuprosessin jäljellä olevat manuaalivaiheet voitaisiin automatisoida ja näin saada prosessi jatkuvan toimituksen -mallin mukaiseksi. Manuaalivaiheiden automatisointia koskevien ratkaisujen lisäksi esitellään myös yksi kehitysratkaisu, jolla käytettyjen ohjelmistotyökalujen välistä integroituvuutta pystyttäisiin parantamaan, liittämällä julkaisua koskevia tietoja tehtävähallintajärjestelmään.

Esitettävien ratkaisujen kartoituksessa on pyritty ottamaan huomioon tulevat julkaisuprosessin muutokset. Välttääkseen ylimääräisen työn tekemisen kaikki esitettävät ratkaisukeinot on valittu tulevan Git-versiohallinnan perusteella. Lisäksi kartoituksesta rajattiin pois komponenttikohtaista versionumerointia koskevien vaiheiden automatisointi. Koska komponenttikohtaisesta versionumeroinnista sekä muutostiedoista ollaan luopumassa kokonaan, ei niitä siksi ole järkevä lähteä automatisoimaan.

5.1 Julkaisupakettien siirtäminen

Ohjelmistojulkaisupakettien siirtämisen automatisointi on jo hyvässä vaiheessa, eikä sen käyttöön ottaminen vaadi kuin testauskattavuuden tulkintaan liittyvien ongelmien korjaamisen. Todennäköisesti tämä vaatii ainoastaan pieniä skriptillisiä muutoksia, eikä se näin vaadi merkittävää ajallista panostusta.

Automatisoinnin lisäksi kyseistä julkaisuprosessin vaihetta voitaisiin jatkokehittää liittämällä kyseisen tapahtuman tiedot käyttöönottotietoina Jiraan. Tämä mahdollistaisi tehtävien julkaisutilan seurannan tehtäväsivun kautta, mikä parantaisi tehtävien tilan seurantaa ja edesauttaisi tiedon välitystä tehtävää koskevien henkilöiden välillä. Käyttöönottoja koskevat tiedot liitetään Jiraan sisäänrakennetun Open Devops -ominaisuuden avulla, joka mahdollistaa kehitystietojen siirtämisen kolmannen osapuolen työkaluista Jiraan. Kyseisen ominaisuuden avulla Jira-tehtäviin pystytään liittämään monenlaista tietoa kehitystyöstä. Näitä ovat käyttöönottotietojen lisäksi esimerkiksi tiedot tehtävään liittyvistä muutoksista sekä kehityshaaroista. Kyseiset tiedot ovat nähtävissä Jira-tehtäviin ilmestyvien kehitys- ja julkaisutietokenttien kautta kuvan 5 mukaisesti. Tehtäväsivun kentät esittävät perustietoja kehitystyön tilasta ja niitä klikkaamalla avautuu paneeli, jonka kautta pystytään tarkastelemaan kehitystietojen tarkempia yksityiskohtia. [23.]



Kuva 5. Jira-tehtävän kehitystietoikkuna [23].

Bitbucket-palvelua käyttäessä versiohallinnassa sijaitsevat kehitystiedot päivittyvät automaattisesti Jiraan, kun tehtävän yksilöivä tunnus sisällytetään muutosten viestikenttiin, kehityshaarojen nimiin tai vetopyyntöihin [24]. Bitbucket-palvelun käyttäminen mahdollistaa myös hyvät työkalut tämän toteuttamiseksi. Tehtävää varten voidaan luoda kehityshaara suoraan tehtävisivun linkin kautta, jolloin siihen lisätään viittaus tehtävästä automaattisesti. Tämän lisäksi Bitbuckettiin voidaan määrittää asetus, joka estää linkittämättömien muutosten lisäämisen versiohallintaan.

Käyttöönottotietojen liittämiseksi Jiraan täytyy tehtäviin viitata versiohallinnan muutoksissa edellä mainitulla tavalla. Lisäksi Jira täytyy integroida Jenkinsin kanssa sitä varten tarkoitettujen liitännäisten avulla. Edellä mainittujen toimenpiteiden jälkeen käyttöönottotietojen lähettäminen on mahdollista määritellä julkaisuputken skriptiin. [19.]

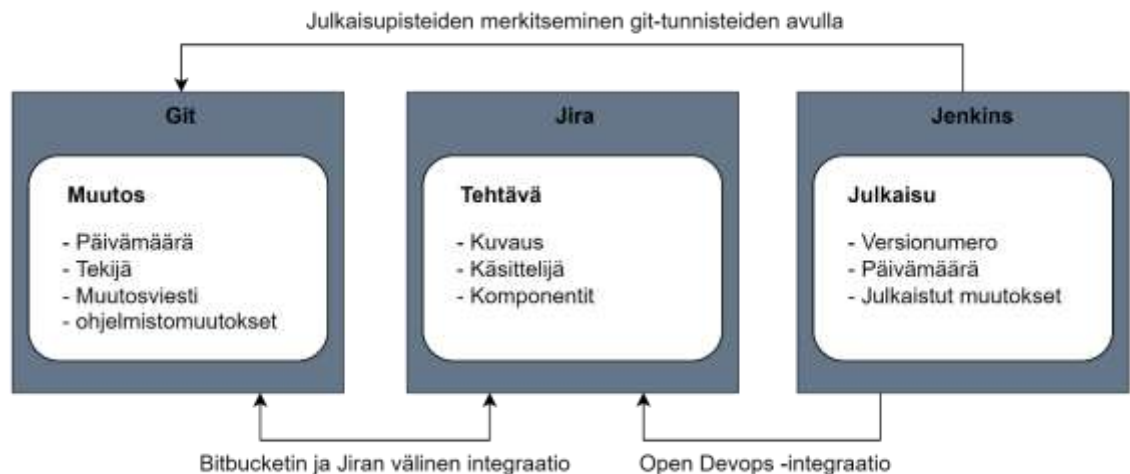
5.2 Muutostietojen automatisointi

Kartoitin työni aikana erilaisia ratkaisuja muutostietojen automatisoimiseksi ja tutkin niiden soveltuvuutta toimeksiantajan käyttötarkoitukseen. Toteutustavaksi haluttiin löytää mahdollisimman valmis ja yksinkertainen ratkaisu, jolla automatisointi pystyttäisiin toteuttamaan mahdollisimman vähäisellä työmäärällä. Ensisijaisesti haluttiin kartoittaa nykyisten työkalujen ja varsinkin Jiran tarjoamia mahdollisuuksia. Tätä toivottiin, jotta automatisoinnin kehittämis- ja ylläpitokustannukset pysyisivät mahdollisimman alhaisina ja automatisoinnilla saavutettaisiin suurin mahdollinen hyöty. Koostettavan muutostietojen sisällön osalta oltiin avoimia, eikä toteutusratkaisun tarvitse välttämättä tukea kaikkia tämänhetkisen muutostietodokumentin sisältämien tietojen esittämistä. Tärkeintä on, että valittava ratkaisu havainnollistaa selkeästi eri ohjelmistojulkaisujen sisältämät muutokset.

Automatisoinnin kannalta olennaisinta ovat lähteet, joista muutostietojen koostamiseen tarvittavat tiedot ovat saatavilla. Toimeksiantajan tapauksessa tämä sijoittautuu kolmeen eri lähteeseen: Jiraan, Jenkinsiin sekä Git:iin. Näistä Jira on kaiken tehtävän työn keskiössä, sillä sen avulla sekä määritetään että seurataan kaikkea tehtävää työtä. Jira-tehtävät sisältävätkin olennaisimmat tiedot tehdyistä muutoksista. Näitä ovat esimerkiksi tehtävän kuvaus, käsittelijä sekä ohjelmistokomponentit, joihin muutokset ovat kohdistuneet. Git-versiohallintajärjestelmän tiedot ovat puolestaan Jiran tietoja teknisempiä. Git sisältääkin tarkan historian kaikista ohjelmistoon tehdyistä muutoksista, mukaan lukien tiedot muutosten päivämäärästä, tekijästä sekä varsinaisista muutoksista lähdekoodissa. Jira-tehtävät voidaan helposti yhdistää versiohallinnan muutoksiin Bitbucketin ja Jiran välisen integraation avulla, jossa versiohallinnan muutosten viesteihin liitetään sitä koskevien tehtävien tunnukset. Tämän ansiosta muutostietoihin voidaan sisällyttää molempien lähteiden tietoja ja näin kuvailla paremmin ohjelmistoon kohdistuneita muutoksia.

Koska automatisoitavien muutoslokien tärkein tehtävä on havainnollistaa eri ohjelmistoversioiden sisältämät muutokset, tarvitaan niiden koostamiseksi myös ohjelmistojulkaisuun liittyviä tietoja, kuten julkaisupäivämäärä ja ohjelmistojulkaisun yksilöivä versionumero. Kyseiset tiedot sijoittautuvat Jenkinsin julkaisuautomaatioon, josta niitä ei vielä toistaisesti tallenneta minnekään. Edellisessä luvussa esitetyn Jira-integraation lisäksi nämä tiedot on mahdollista liittää myös versiohallintajärjestelmään. Tämä tapahtuu Git:n tunnisteiden avulla, jotka ovat versiohallintaan liitettäviä merkintöjä, joilla pystytään merkitsemään versiohallinnan historiaan erilaisia ohjelmistoa koskettavia muutoskohtia. Tunnisteita käytetäänkin tyypillisesti juuri julkaisupisteiden merkitsemiseen. [25.] Tunnisteisiin voidaan sisällyttää yksilöivänä nimenä toimivan versionumeron lisäksi

myös kuvaus, tunnisteiden lisääjä sekä julkaisupäivämäärä. Julkaisupisteiden merkitseminen voitaisiin sisällyttää julkaisuautomaatioon, jolloin julkaisujen tiedot lisättäisiin versiohallintaan täysin automaattisesti. Kuva 6 havainnollistaa muutoksia koskevia tietolähteitä sekä keinoja, joiden avulla eri lähteiden tiedot voidaan yhdistää toisiinsa.



Kuva 6. Muutosten tietolähteet.

Kun julkaisutiedot on yhdistetty Jiraan, on projektia koskevien tehtävien käyttöönottopahtumia mahdollista tarkastella kuvan 7 mukaisen Käyttöönotot-näkymän avulla. Näkymä listaa eri ympäristöihin julkaistut tehtävät ja sijoittaa niitä vastaavat käyttöönottopahtumat aikajanelle. Esitettäviä tietoja on mahdollista rajata erilaisten suodattimien avulla esimerkiksi julkaisu ympäristön, tehtävätyypin tai versionumeron perusteella. Tarkempia tietoja esitetyistä käyttöönotoista on mahdollista nähdä klikkaamalla aikajanelle sijoitettuja tapahtumia. Tällä tavoin on nähtävissä muun muassa tiedot automaatioputkesta, jossa käyttöönotto on toteutettu sekä käyttöönoton yksilöivä nimi.

Kuva 7. Jiran käyttönotot-näkymä.

Vaikka käyttöönotot-näkymän avulla pystytäänkin tehokkaasti seuraamaan projektin tehtävien etenemistä eri julkaisuputkien läpi, niin ei siitä ole korvaajaa manuaalisesti ylläpidettäville muutostiedoille. Näkymästä on hankalaa hahmottaa, missä käyttöönottopahtumassa esitetyt tehtävät on julkaistu, sillä se ei esitetä käyttöönottoja yksilöiviä tietoja ilman erillisiä klikkauksia. Käyttöönottopahtumia ei myöskään voida suoraan yhdistää koostettuun julkaisupakettiin, sillä käyttöönoton yksilöivä nimi määritetään automaattisesti, eikä se vastaa ohjelmistojulkaisupaketeissa käytettävää koostenumeroa.

Kartoitukseni perusteella asetettujen vaatimusten mukaiset muutostiedot koostetaan yleisesti versiohallinnan tietojen perusteella ja käytetyt toteutustavat ovat riippuvaisia tiedoista, joita koostettavaan dokumenttiin halutaan sisällyttää. Yksinkertaisimmillaan koostaminen voidaan toteuttaa pelkästään versiohallintajärjestelmän omien lokikomentojen avulla, jolloin muutostiedot muodostetaan pelkästään versiohallinnan tiedoilla. Kehittyneempien muutostietojen koostamiseksi on saatavilla erillisiä työkaluja, joiden avulla muutostiedoissa voidaan esittää myös versiohallinnan muutoksiin linkitettyjen tehtävien sisältämiä tietoja. Jiran toimiessa kaiken kehitystyön keskiössä olisi tärkeä, että valittava ratkaisu integroituisi siihen mahdollisimman hyvin. Tämän takia ensisijaisesti haettiin ratkaisuja, joilla muutostietoihin pystyttäisiin sisällyttämään mahdollisimman laajasti myös Jira-tehtävien sisältämiä tietoja.

Saatavilla olevista ratkaisuksista parhaiten toimeksiantajan tarpeita vastasi Jenkinsiin asennettava Git Changelog -lisäosa, jota käyttäen pystytään koostamaan kuvan 8 mukaiset muutostiedot,

joista on selkeästi hahmotettavissa eri ohjelmistojulkaisujen sisältämät muutokset. Kyseisen ratkaisun avulla muutostietoihin on mahdollista sisällyttää versiohallinnan tietojen lisäksi myös muutoksiin linkitettyjen Jira-tehtävien tietoja. Tätä tuetaan kuitenkin ainoastaan rajallisesti, eikä esimerkiksi nykyisten muutostietojen sisältämien komponenttitietojen esittämistä tueta. Tuki kattaa kuitenkin tehtävien perustiedot, minkä lisäksi muutosten tietoihin on mahdollista sisällyttää linkki tehtävän Jira-sivulle, jonka kautta lukijalla on helppo pääsy muihin tehtävää koskettaviin tietoihin.

Changelog Release 1.20



Luonut Kalle Kurki

Viimeksi päivitetty: 1 minuutti sitten • 1 henkilö katsoi

v1.20-77 (2023-02-02)

OPARI-2201: Kolmas esimerkki tehtävä VALMIS

Kalle Kurki: OPARI-2201 Kolmas esimerkki muutos

v1.20-76 (2023-02-02)

OPARI-1: Esimerkki tehtävä VALMIS

Kalle Kurki: OPARI-1 Esimerkki muutos

OPARI-3: Toinen esimerkki tehtävä VALMIS

Kalle Kurki: OPARI-3 Toinen esimerkki muutos

Kuva 8. Git Changelog -lisäosalla koostetut muutostiedot.

Ratkaisu koostaa muutostietodokumentin helposti konfiguroitavan mallipohjan perusteella, johon dokumentin rakenne voidaan määrittää esimerkiksi HTML- tai Markdown-merkintäkielten avulla. Esitettävät tiedot määritellään mallipohjaan lisäosan määrittämien tunnisteiden avulla,

jotka korvataan koostamisen aikana niitä vastaavilla muutostiedoilla. Kuva 9 esittää mallipohjaa, jonka perusteella myös aikaisemmin esitetyt muutostiedot on koostettu.

```

{{#tags}}
  <!-- Julkaisun tiedot -->
  <h2> {{name}} ({{tagDate}}) </h2>
  {{#issues}}
    <p>
      <!-- Linkki JIRA-tehtävään -->
      <a href="https://your-domain.atlassian.net/browse/{{issue}}"
        data-card-appearance="inline"
        class="external-link"
        rel="nofollow">
        {{issue}}
      </a>
    </p>
  {{#commits}}
    <!-- Versiohallinnan muutosten tiedot -->
    <p>{{authorName}}: {{messageTitle}}</p>
  {{/commits}}
{{/issues}}
<hr></hr>
{{/tags}}

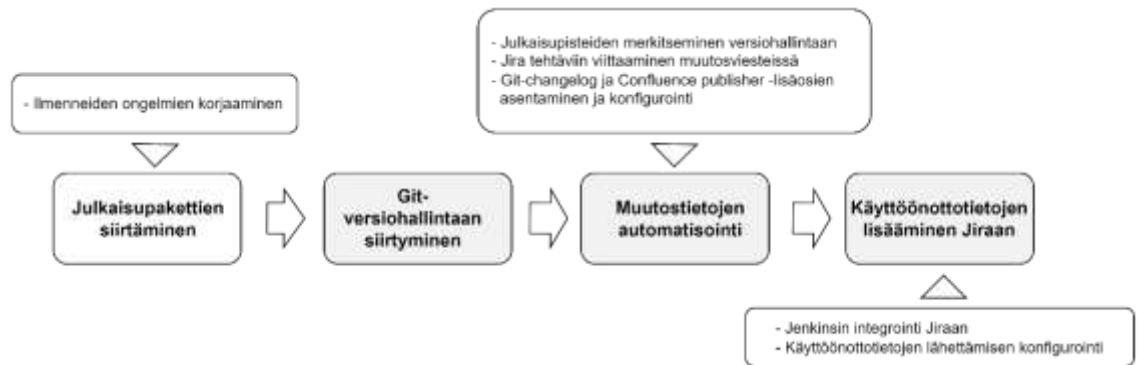
```

Kuva 9. mallipohja.

Git Changelog -lisäosan avulla koostetut muutostiedot on mahdollista julkaista eri alustoille erikseen asennettavien lisäosien avulla. Toimeksiantajan tapauksessa julkaisu olisi paras toteuttaa Confluenceen, jonne myös muu ohjelmistoa koskeva dokumentaatio on keskitetty. Tämä voitaisiin toteuttaa Confluence-sivujen luomisen ja muokkauksen mahdollistavan Confluence publisher -lisäosan avulla.

6 Toteutussuunnitelma

Yhtenä työn tavoitteista oli laatia etenemissuunnitelma, jota noudattaen työn tilaaja voisi lähteä kehittämään julkaisuprosessiaan yksi vaihe kerrallaan. Tätä tarkoitusta varten työssä kehiteltiin kuvan 9 mukainen tiekartta, jossa esitetään vaihe vaiheelta, kuinka luvussa 5 esitetyt kehitysratkaisut voitaisiin implementoida osaksi toimeksiantajan prosessia.



Kuva 10. Tiekartta.

Kaiken kaikkiaan kehityskeinoja käsittelevässä luvussa esiteltiin kolme ratkaisua, joilla toimeksiantajan julkaisuprosessia voitaisiin kehittää. Näistä julkaisupakettien siirtäminen ja muutostietojen automatisointi koskettivat olemassa olevien vaiheiden automatisointia. Niiden avulla saavutetaan konkreettisia säästöjä vapauttamalla vaiheisiin käytetty aika muita tehtäviä varten sekä pienentämällä prosessin virhealttiutta. Lisäksi ne poistaisivat kehitystyöstä yksitoikkoisia ja toistuvia vaiheita, millä on positiivisia vaikutuksia työn mielekkyyteen. Automatisointien lisäksi kapaleessa esitettiin myös kehitysratkaisu, jolla käyttöönottojen tiedot voitaisiin lisätä toimeksiantajan tehtävähallintajärjestelmään. Tämä puolestaan mahdollistaisi käyttöönottoja koskevien tietojen seuraamisen Jiran kautta ja parantaisi julkaisua koskevan tiedon välitystä.

Koska automatisointia koskevat kehitysratkaisut tarjoavat konkreettisempiä hyötyjä, tulisi ne implementoida ensimmäiseksi. Näistä ensimmäiseksi voitaisiin automatisoida julkaisupakettien siirtäminen, sillä sen toteuttaminen on jo aloitettu, eikä toteutuksen loppuun saattaminen vaatisi kuin ilmenneiden ongelmien korjaamisen. Lisäksi sen automatisoimisella saavutettaisiin kaikista suurin hyöty, koska kyseisen vaihe on manuaalisesti toteutettuna kaikista työläin sekä alttein virheille. Seuraavaksi automatisoitavat muutostiedot voitaisiin toteuttaa, kun sen edellyttämä Git-versiohallintajärjestelmä on otettu käyttöön. Tämä on kaikista työläin vaihe automatisoida, sillä

se edellyttää sekä julkaisupisteiden merkitsemisen versiohallintaan että Jira-tehtäviin viittaamisen versiohallinnan muutoksissa. Lisäksi varsinaisen automatisoinnin lisääminen vaatii kahden eri liitännäisen asentamisen ja konfiguroimisen toimeksiantajan Jenkins-ympäristöön. Viimeiseksi implementoitavaksi jää käyttöönottotietojen lisääminen Jiraan. Kun Jira-tehtävien linkitys versiohallintaan on toteutettu jo edellistä ratkaisua varten, ei tämän toteuttaminen vaadi kuin Jenkinsin integroimisen Jiraan ja käyttöönottotietojen lähettämisen konfiguroinnin julkaisuputkeen.

7 Yhteenveto

Opinnäytetyön tarkoituksena oli kartoittaa toimeksiantajan julkaisuprosessista useimmiten toistuvat sekä eniten aikaa vievät manuaalivaiheet ja etsiä keinoja niiden automatisoimiseksi. Lisäksi tavoitteena oli kehittää etenemissuunnitelma, jota käyttäen työn tilannut yritys voisi lähteä kehittämään julkaisuprosessiaan vaihe kerrallaan.

Kaikkiaan kartoituksen kohteena oleva julkaisuprosessi sisälsi kolme manuaalisesti tehtävää vaihetta. Näistä komponenttikohtainen versionumerointi on poistumassa käytöstä lähitulevaisuudessa, eikä sitä siksi ollut järkevä lähteä automatisoimaan. Tämän lisäksi julkaisupakettien siirtämisen automatisointi oli jo aloitettu, jonka takia muutostietojen koostaminen oli lopulta ainoa vaihe, minkä automatisoimiseksi haettiin ratkaisua. Tähän ratkaisua haettiin kartoittamalla muutostietojen koostamiseen yleisesti käytettyjä menetelmiä ja kyseiseen käyttötarkoitukseen saatavilla olevia työkaluja. Tämän kartoituksen perusteella työssä esiteltiin toimeksiantajan tarpeita parhaiten vastaava ratkaisu, jolla muutoksia koskevat tiedot voitaisiin esittää asetettujen vaatimusten mukaisesti. Julkaisuprosessin vaiheita koskevien automatisointien lisäksi työssä esiteltiin myös ratkaisu, jolla julkaisuprosessia voitaisiin kehittää liittämällä julkaisua koskevat tiedot tehtävänhallintajärjestelmään. Tällä tavoin Jiran kautta pystyttäisiin seuraamaan tehtävien etenemistä myös julkaisun osalta.

Esitettyjen kehityskeinojen perusteella laadittiin tiekartta, jossa kuvaillaan vaihe vaiheelta, kuinka esitetyt ratkaisut voitaisiin implementoida toimeksiantajan julkaisuprosessiin. Tämä tiekartta tiivistää hyvin työssä kartoitetut asiat ja tarjoaa työn tilaajalle selkeän etenemissuunnitelman, jolla lähteä kehittämään julkaisuprosessiaan vaihe kerrallaan.

Lähteet

1. Nazar A. Hidden benefits of automated deployment you should know about. [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://deployplace.com/blog/hidden-benefits-of-automated-deployment/>
2. Developer experience knowledge base. Automated deployment. [Internet]. 2019 [viitattu 16.2.2023] Saatavilla: <https://developerexperience.io/articles/automated-deployment>
3. Raute oyj. Vuosikertomus 2021. [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://www.raute.fi/vuosikertomus-2021/>
4. Sumo logic. Software deployment – definition & overview [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://www.sumologic.com/glossary/software-deployment>
5. Chandani P. Secure and scalable CI/CD Pipeline With AWS. [Internet]. 2019 [viitattu 16.2.2023] Saatavilla: <https://dzone.com/articles/secure-and-scalable-cicd-pipeline-with-aws>
6. Rossel S. Continuous Integration, Delivery and Deployment: Getting Started with the Processes and the Tools to Continuously Deliver High-Quality Software. Brimingham. Packt Publishing, Limited: 2017.
7. Developer experience knowledge base. Continuous integration. [Internet]. 2019 [viitattu 16.2.2023] Saatavilla: <https://developerexperience.io/practices/continuous-integration>
8. Hari's Technical Space. Concepts: The stages of the CI/CD pipeline. [Internet]. 2020 [viitattu 16.2.2023] Saatavilla: <https://haritechworld.com/2020/08/13/concepts-the-4-stages-of-the-ci-cd-pipeline/>
9. Helsingin yliopisto. Tietokone työvälineenä. [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://tkl-lapio.github.io/>
10. Fernandez M. Design an Effective Build Stage for Continuous Integration. [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://semaphoreci.com/blog/build-stage>
11. The Startup Lab. What is Automated Testing. [Video]. 2019 [viitattu 16.2.2023] Saatavilla: https://www.youtube.com/watch?v=Nd31XiSGJLw&t=256s&ab_channel=TheStartupLab
12. Triare. Quality assurance: Automated vs manual testing vs unit testing. [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://triare.net/insights/automated-vs-manual-testing/>
13. Pathania N. Learning Continuous Integration With Jenkins : Beginner's Guide to Implementing Continuous Integration and Continuous Delivery Using Jenkins. Brimingham. Packt Publishing, Limited: 2016.

14. Atlassian. Who uses Jira Software. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#jira-for-agile-teams>
15. Atlassian. Create and configure your issues. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://support.atlassian.com/jira-software-cloud/docs/create-and-configure-your-issues/>
16. Atlassian. Configure development tools. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://support.atlassian.com/jira-software-cloud/docs/configure-development-tools/>
17. Open Data Science. Git vs SVN: What's the Difference. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://odsc.medium.com/git-vs-svn-whats-the-difference-2c7072f7679f>
18. Jenkins handbook: Managing Plugins. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://www.jenkins.io/doc/book/managing/plugins/>
19. Atlassian. Integrate with Jenkins. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://support.atlassian.com/jira-cloud-administration/docs/integrate-with-jenkins/>
20. Code With Engineering Playbook: Component Versioning. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://microsoft.github.io/code-with-engineering-playbook/source-control/component-versioning/>
21. Lalu J. Ohjelmistoautomaation lisääminen ohjelmistoprosessiin ja ylläpitoon [Internet]. 2022 [viitattu 16.2.2023] Saatavilla: <https://urn.fi/URN:NBN:fi:amk-202204255887>
22. Amoeboids. Changelog example, format and how to write a changelog? [Internet]. 2020 [viitattu 16.2.2023] Saatavilla: <https://amoeboids.com/blog/changelog-how-to-write-good-one/>
23. Atlassian. Open DevOps experiences with Jira. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://developer.atlassian.com/cloud/jira/software/open-devops>
24. Atlassian. View development information for an issue. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://support.atlassian.com/jira-software-cloud/docs/view-development-information-for-an-issue/>
25. Git. Git Basics - Tagging. [Internet]. 2023 [viitattu 16.2.2023] Saatavilla: <https://git-scm.com/book/en/v2/Git-Basics-Tagging>