Ville Hirsimäki

Versionhallinta ja parhaat käytänteet integraatiojärjestelmälle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

15.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ville Hirsimäki Versionhallinta ja parhaat käytänteet integraatiojärjestelmälle 37 sivua 13.4.2018
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmointi
Ohjaaja	yliopettaja Petri Vesikivi

Insinöörityössä selvitettiin, millainen työn tilaajayrityksen integraatiojärjestelmän käytössä oleva versionhallintakokonaisuus on ja miten sitä voitaisiin kehittää. Eri osa-alueiden parhaita sovellusvaihtoehtoja vertailtiin keskenään, ja niistä valittiin parhaat mahdolliset sovellukset tähän käyttötapaukseen. Työssä perehdyttiin käytössä olevaan versionhallintakokonaisuuteen ja tavoitteena oli löytää mahdollisia kohteita, joita voidaan parantaa.

Työssä havaittiin, että käytössä olevien sovellusten ja niiden vaihtoehtojen välillä ei ollut suuria eroja ja ne vaikuttivat tarjoavan samat päätoiminnallisuudet. Sovellusten väliset erot keskittyivät lähinnä tarjoamaan toiminnallisuuksia erityyppisille projekteille. Versionhallinnan eri osa-alueiden toteuttamisessa näitä eroja pitää verrata ja valita paras mahdollinen sovellus kyseiselle käyttötapaukselle. Tietyissä osa-alueissa nämä erot ovat kuitenkin sen verran pieniä, että käytännössä valinnalla ei olisi suurta vaikutusta projektille.

Insinöörityössä tutustuttiin käytössä olevaan versionhallintakokonaisuuteen ja löydettiin parhaat mahdolliset sovellukset tämäntyyppiselle projektille. Työn perusteella päädyttiin siihen lopputulokseen, että käytössä olevia sovelluksia ei ollut tarvetta lähteä muuttamaan. Työssä laadittiin parannusehdotuksia uusien versioiden tuotannonsiirtoon käyttämällä jatkuvan julkaisun käytänteitä. Lisäksi integraatiojärjestelmän puolelle voitaisiin lisätä toiminnallisuus, jolla voidaan siirtyä helpommin versioiden välillä. Näitä parannuksia voidaan lähteä toteuttamaan työn jälkeen, jos yritys siihen päätyy.

Avainsanat	Versionhallinta, Integraatiojärjestelmä

Author	Ville Hirsimäki
Title	Version control and its best practices for an integration platform
Number of Pages	37 pages
Date	13 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation Option	Programming
Instructor	Petri Vesikivi, Principal Lecturer

The goal of this thesis was to get to know the current version control elements of HiQ Finlands integration platform and find out possible improvements that can be done with either new software or practices. Different version control platforms and methods were compared with each other while trying to find the best candidates for this type of project.

It was found that most of the products offer the same core functionality as their competitors and provide different features that are aimed at a specific type of project. Choosing the right software was about finding the right features for the project in question. In some use cases the differences between various software are so small that it does not really matter which one you choose. For bigger projects it is still worth it trying to optimize the various elements of version control and development practices.

The current version control elements were found to be adequate for this type of project and did not need to be changed. Possible improvements were found in the form of implementing Continuous Delivery methods into the current production cycle, which would speed the deployment of new versions for customers. A new functionality could also be implemented on the user interface side that allows the user to switch between versions for different tasks. This was not currently implemented and using older versions of tasks provided to be tricky. These improvements can be implemented after the thesis if they are deemed appropriate by the company.

Keywords	Version control, Integration platform

Sisällys

Lyhenteet ja käsitteet

A2A	*Application to Application.* Sovellusten välinen kommunikointi.
API	*Application Programming Interface.* Rajapinta, jonka avulla kommunikoidaan sovelluksen kanssa.
B2B	*Business to Business.* Sovellusten välinen kommunikaatio yritysten välillä.
BPD	*Business Process Diagram.* BPMN-notaation diagrammi, jolla kuvataan liiketoiminnan prosesseja.
BPM	*Business Process Management.* Prosessipainotteinen johtamisoppi.
BPMN	*Business Process Management Initiative.* Graafinen tapa esittää liiketoiminnan prosesseja.
C#	.NET-pohjainen ohjelmointikieli.
CD	*Continuous Delivery.* Ohjelmistokehityksen tapa, jolla pyritään optimoimaan sovelluksen versioiden tuotantoon vientiä.
CI	*Continuous Integration.* Ohjelmistokehityksen tapa, jolla pyritään löytämään suurin osa virheistä sovelluksen kehityksen aikana.
EAI	*Enterprise Application Integration.* Sovelluksien yhdistämiseen suunniteltu kehys.
FTP	*File Transfer Protocol.* Standardoitu protokolla tiedostojen siirtämiseen tietokoneiden välillä.
MyGet	Erilaisten ohjelmistopakettien ylläpidon tarjoaja.
NuGet	Sovellus, jolla voidaan rakentaa .NET-pohjaisia ohjelmia paketteihin, joita voidaan käyttää muualla.
Repo	*Repository.* Kokonaisuus, johon Git tallentaa projektin versioiden historian.

SHA-1 *Secure Hash Algorithm 1.* Kryptograafinen tiivistefunktio.

SMTP *Simple Mail Transfer Protocol.* Internetstandardi sähköpostiviestien lähetykseen.

TFS *Team Foundation Server.* Microsoftin sovellus, suunniteltu helpottamaan ohjelmistokehityksen eri vaiheita.

VSTS *Visual Studio Team Services.* TFS:n pilvipohjainen versio.

1 Johdanto

Versionhallinta on tärkeä osa nykyaikaista ohjelmistokehitystä. Se on kehittynyt viime vuosikymmenen aikana yksinkertaisista paikallisista tietokannoista verkossa ylläpidettyihin versionhallintasovelluksiin, jotka tarjoavat käyttäjilleen versionhallinnan lisäksi myös muita ohjelmistokehityksen menetelmiä. Tämä nopea kehitys tarkoittaa myös sitä, että tarjolla on monia eri vaihtoehtoja erilaisille ohjelmistoprojekteille. Yritysten on pidettävä huoli siitä, että käytössä olevat versionhallintametodit ovat parhaita mahdollisia kulloiseenkin käyttötapaukseen.

Insinöörityön tarkoituksena oli tutkia ohjelmistoalalla toimivan HiQ Finland -yrityksen FRENDS-integraatiojärjestelmässä tällä hetkellä käytössä olevaa versionhallintakokonaisuutta raporttimielessä ja selvittää, ovatko käytössä olevat sovellukset parhaat mahdolliset tämäntyyppiselle projektille. Työn tarkoituksena ei ollut lähteä toteuttamaan parannuksia kyseiseen kokonaisuuteen, vaan esittää työn aikana havaitut parannusehdotukset, joita voidaan lähteä toteuttamaan työn jälkeen, jos näin halutaan.

Insinöörityöraportti on jaettu kolmeen osaan. Ensimmäisessä osassa käsitellään versionhallinnan kehittymistä ensimmäisistä metodeista nykyisin käytettyihin järjestelmiin sekä integraatiojärjestelmien teoriaa. Toisessa osassa käydään läpi HiQ:n integraatiojärjestelmän versionhallintakokonaisuus: sen osa-alueet, käytössä olevat sovellukset ja niille löytyvät vaihtoehdot. Kolmannessa osassa käydään läpi, mitä työn aikana havaittiin ja mitä parannusehdotuksia versionhallintakokonaisuudelle löydettiin.

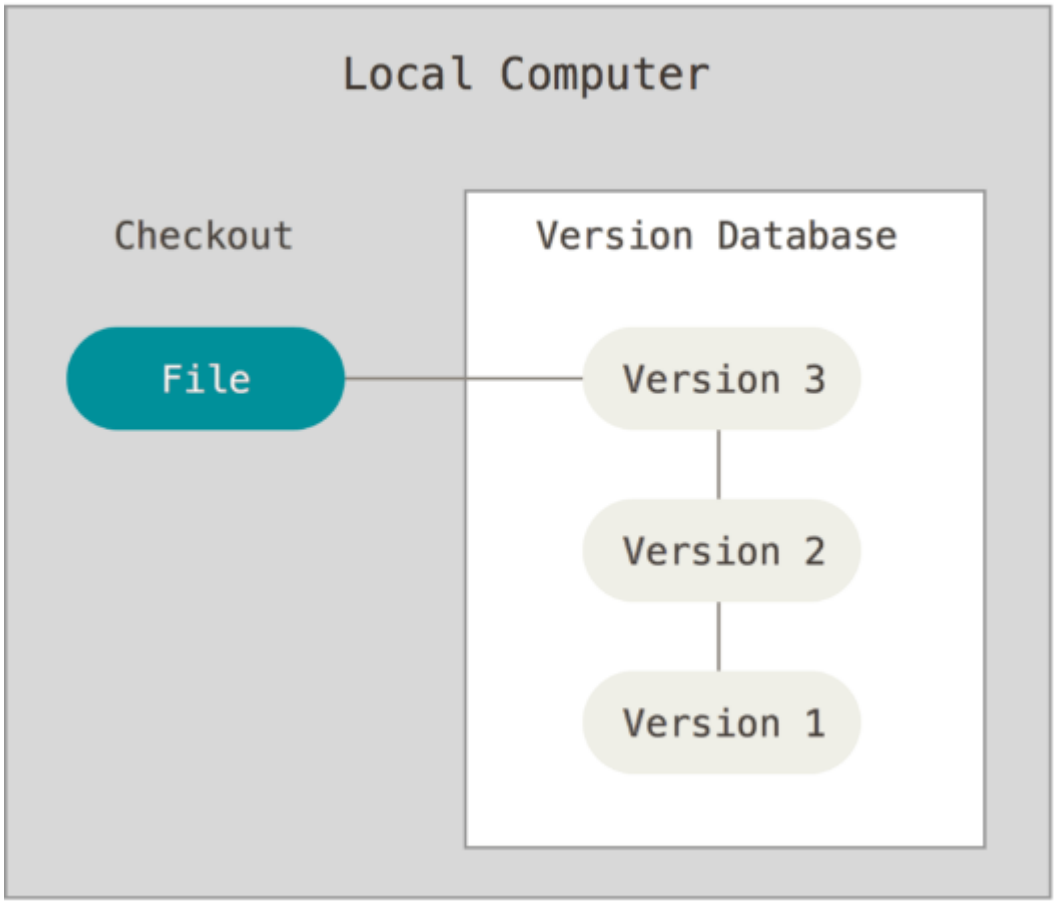
2 Versionhallinta ja versionhallintajärjestelmät

2.1 Versionhallinnan historia

Versionhallinnalla tarkoitetaan yleisesti jotain järjestelmää, jolla voidaan pitää kirjaa tiedostoon tai tiedostoihin tehdyistä muutoksista ja aiemmista versioista. Sen avulla voidaan siirtyä vaivattomasti tarkastelemaan aikaisempia versioita esimerkiksi ohjelman lähdekoodista. Versionhallintajärjestelmä pitää kirjaa tiedoston koko historiasta aina ensimmäisestä versiosta viimeisimpään ja mahdollistaa liikkumisen minkä tahansa versioiden välillä. (1, s. 30.)

Yleisin versionhallintatapa on omalla koneella tehty paikallinen versionhallinta. Käyttäjä tekee manuaalisesti kopioita tiedostoista työskentelyn yhteydessä joko uusiin kansioihin tai nimeämällä tiedoston uudestaan. Jokainen on varmasti käyttänyt aikaisemmin jonkinnäköistä manuaalista versionhallintaa esimerkiksi nimeämällä raportteja uudestaan aikaleimalla. Tämän on kuitenkin erittäin virhealtis versionhallinnan muoto, sillä se ei millään tavalla estä käyttäjää tekemästä virheitä. Käyttäjä voi vahingossa tallentaa tiedoston väärällä nimellä, tallentaa sen väärään paikkaan tai vahingossa poistaa vanhan version. Ensimmäiset versionhallintajärjestelmät kehitettiin estämään tämänlaisia virhetilanteita (2, s. 27).

Ensimmäiset versionhallintajärjestelmät olivat yksinkertaisia paikallisia tietokantoja, joihin tallennettiin jossain muodossa tiedostojen eri versiot. Yksi suosituimmista ensimmäisistä järjestelmistä on RCS (Revision Control System), joka tallentaa tietokantaan tiedoston muutokset verrattuna aikaisemmin tallennettuun tiedostoon. Tällöin aikaisempiin versioihin voidaan palata lisäämällä kaikki muutokset aikaisemmista versioista tiettyyn versioon asti. Tämä tietokanta versioiden muutoksista tallennettiin käyttäjän omalle kiintolevylle erityisformaatissa. RCS ei siis tallenna itse tiedostoja vaan tiedon muutoksista aikaisempaan versioon verrattuna. (1, s. 34.) Kuvassa 1 on havainnollistettu paikallisen versionhallinnan toimintaa.



Kuva 1. Paikallisen versionhallinnan toimintaperiaate (2, s. 28).

2.2 Versionhallinnan etuja

Versionhallintajärjestelmä mahdollistaa useiden henkilöiden työskentelyn samanaikaisesti projektin parissa. Ilman versionhallintajärjestelmää käyttäjät joutuisivat tekemään tiedostoon muutoksia peräkkäisesti. Ensimmäinen käyttäjä tekisi tiedostoon ensimmäisen muutoksen, minkä jälkeen seuraava käyttäjä saa oikeuden tiedostoon ja käy tekemässä muutokset. Versionhallintajärjestelmän avulla jokainen käyttäjä pystyy tekemään muutoksia samanaikaisesti tiedostoihin, minkä jälkeen muutokset pystytään kokoamaan yhteen ilman aikaisempien muutoksien hävittämistä. (3.)

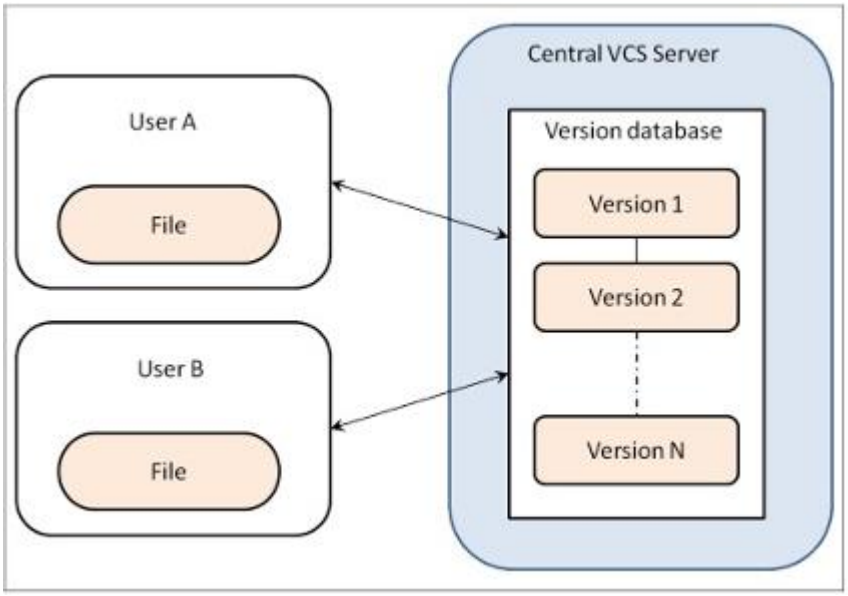
Versionhallintajärjestelmät arkistoivat kaikki muutokset, jotka tiedostoon tehdään. Tämän lisäksi ne myös pitävät listaa muutoksen ajankohdasta ja henkilöistä, jotka ovat tehneet muutoksen. Muutokselle on mahdollista myös lisätä kommentti, joka antaa käyttäjille kuvan tehdyistä asioista. Tämä on erittäin hyödyllistä esimerkiksi

ohjelmistoprojekteissa. Esimerkiksi kun yhteen tiedostoon ollaan tekemässä samanaikaisesti kahden eri henkilön muutoksia, joista toinen halutaan myöhemmin perua, versionhallintajärjestelmän tiedoista voidaan poimia helposti haluttu muutos ja mennä sitä aikaisempaan versioon. (3.)

2.3 Keskitetyt versionhallintajärjestelmät

Keskitetyt versionhallintajärjestelmät (**Centralized Version Control Systems**) kehitettiin helpottamaan työskentelyä projektin jäsenten kesken. Keskitetyssä versionhallinnassa käytetään keskitettyä tietokantapalvelinta paikallisen tietokannan sijasta. Jokainen projektin jäsen pystyy hakemaan palvelimelta projektin tiedostot ja tekemään niihin muutoksia. Projektin jäsenet pystyvät myös seuraamaan muiden jäsenten tekemiä muutoksia ja saamaan paremman käsityksen projektin etenemisestä. Järjestelmänvalvonnan kannalta keskitettyä versionhallintajärjestelmää on myös paljon helpompi hallita kuin yksittäisiä tietokantoja jokaisen projektin jäsenen koneelta. Tämänkaltaisia suosittuja järjestelmiä ovat esimerkiksi Subversion, CVS ja Perforce (2, s. 29).

Keskitetyn versionhallinnan etu on myös sen haittapuoli. Keskitetty tietokantapalvelin on yksittäinen piste, jonka toimintahäiriö keskeyttää koko projektin etenemisen. Käyttäjät tarvitsevat yhteyden palvelimelle tehdäkseen muutoksia tiedostoihin. Jos palvelin on kaatunut syystä tai toisesta, käyttäjät eivät pääse päivittämään muutoksiaan tietokantaan. Jos palvelin jostain syystä lakkaa toimimasta tai korruptoituu, koko projektin historia menetetään, jos varmuuskopioita ei ole tehty. Käyttäjien kiintolevyille tallentuu vain senhetkinen versio. Kaikki aikaisemmat versiot on tallennettu tietokantapalvelimelle (1, s. 36). Kuvassa 2 on havainnollistettu keskitetyn versionhallinnan toimintaa.



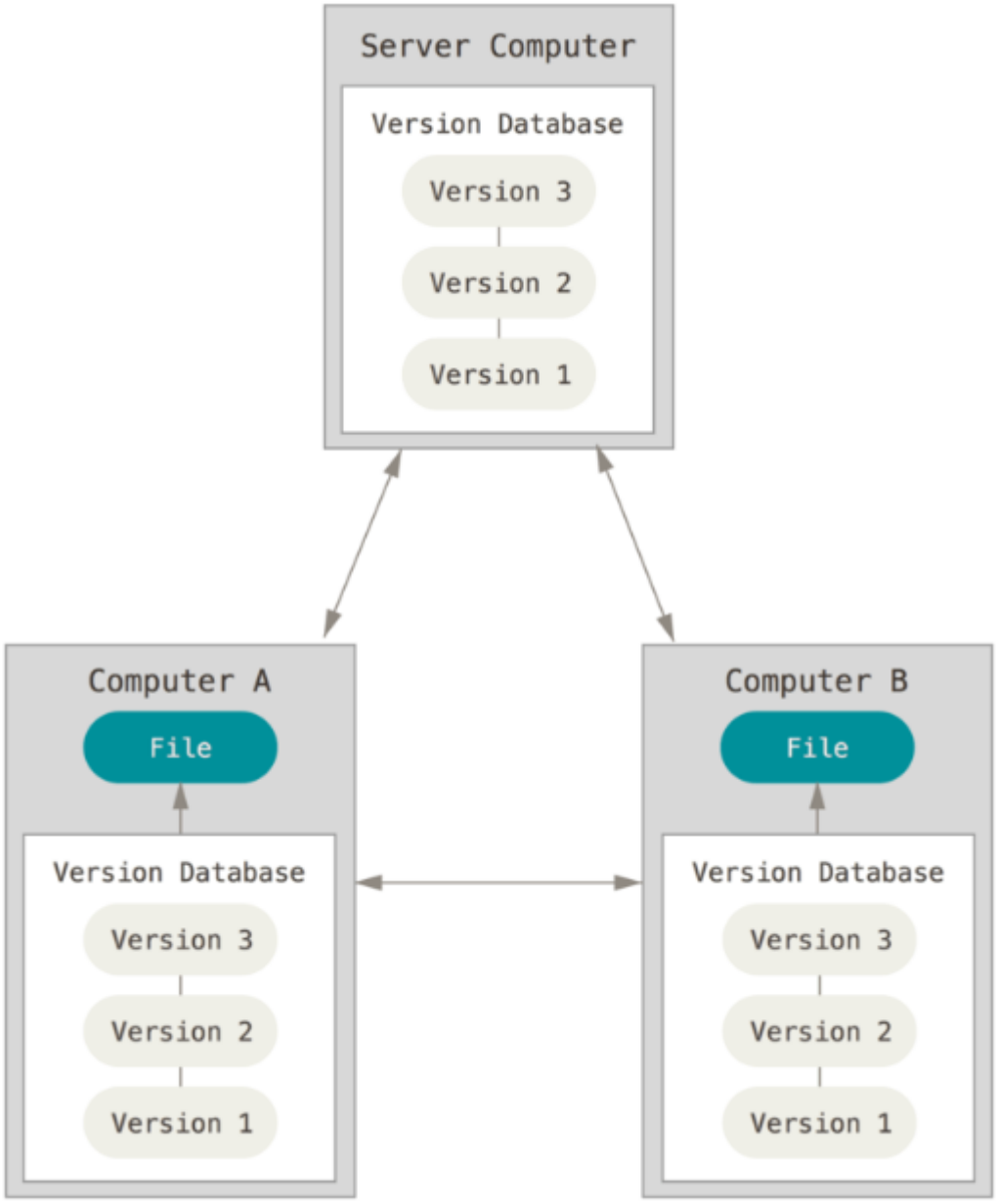
Kuva 2. Keskitetyn versionhallinnan toimintaperiaate (1, s. 35).

2.4 Hajautetut versionhallintajärjestelmät

Hajautetut versionhallintajärjestelmät (**Distributed Version Control Systems**) korjaavat keskitetyn versionhallinnan yksittäisen pisteen haittapuolen. Hajautetussa versionhallinnassa käyttäjät eivät hae palvelimelta pelkästään uusimpia tiedostoja, vaan ottavat kopion koko tietokannasta. Tehdyt muutokset viedään keskitettyyn tietokantaan, josta muut käyttäjät voivat hakea ne omiin tietokantoihinsa. Hajautettuja versionhallintajärjestelmiä ovat esimerkiksi Git, Mercurial, Bazaar ja Darcs, joista viime aikoina Git on siirtynyt yhdeksi suosituimmista vaihtoehdoista. (2, s. 30.)

Hajautetun versionhallinnan etuna on koko tietokannan kopioiminen jokaiselle käyttäjälle. Koska jokaisella käyttäjällä on oma tietokanta, sille pystytään tekemään mitä tahansa muista käyttäjistä riippumatta. Paikallisesti kopioitu tietokanta tarkoittaa myös sitä, että siihen kohdistuvat muutokset ovat huomattavasti nopeampia kuin keskitettyyn tietokantaan tehdyt muutokset. Muutokset ovat paikallisia, joten tietokantapalvelimeen ei tarvitse olla yhteydessä niitä tehdessä. Palvelimeen tarvitaan yhteys vain siinä vaiheessa, kun muutokset halutaan jakaa muille käyttäjille. Tämä mahdollistaa muutoksen tekemisen paikoissa, joissa ei ole mahdollisuutta internetyhteyteen. Koska hajautetussa versionhallinnassa kopioidaan koko tietokanta käyttäjille, keskitetyn tietokantapalvelimen toimintahäiriö ei tarkoita enää aikaisempien versioiden katoamista. (3.)

Hajautetussa versionhallinnassa on myös mahdollista käyttää enemmän kuin yhtä keskitettyä tietokantaa. Tällöin voidaan tehdä enemmän kuin yksi tietokanta eri projektin osa-alueille. Tätä voidaan hyödyntää esimerkiksi eri versioiden ylläpitämisessä asiakkaille tai projektin jäsenten työskentelemiseen kahdesta eri toimipisteestä, joissa molemmille on oma keskitetty tietokantapalvelin (3). Kuvassa 3 on havainnollistettu hajautetun versionhallinnan toimintaa.



Kuva 3. Hajautetun versionhallinnan toimintaperiaate (2, s. 30).

Hajautetun versionhallinnan toimintaperiaate voi myös olla sen haittapuoli, jos itse tietokanta on suuri eikä sitä pystytä pilkkomaan pienempiin osiin. Ison tietokannan kopioimisaika voi helposti nousta suureksi, jolloin jokainen käyttäjä joutuu turhaan odottamaan sen kopioimista. Järjestelmänvalvonnan kannalta hajautettu versionhallinta voi myös olla hankala ylläpitää. Jos tietylle kansiolle halutaan estää oikeudet, se pitää siirtää toiseen tietokantaan, jota kaikki käyttäjät eivät kopioi. Tiedostojen poistaminen tietokannasta on myös hankalaa. Esimerkiksi tapauksessa, jossa yhtiön on laillisesti poistettava kaikki tiedostot, jotka käsittelevät arkaluontoista tietoa, pitää koko tietokanta siirtää uuteen tietokantaan ja samalla poistaa kaikki arkaluontoinen materiaali. Kaikki versionhallintajärjestelmät eivät tue tiedostojen poistamista, ja tämä voi silti olla erittäin pitkä prosessi ison tietokannan tapauksessa, vaikka järjestelmä sitä tukisikin. (3.)

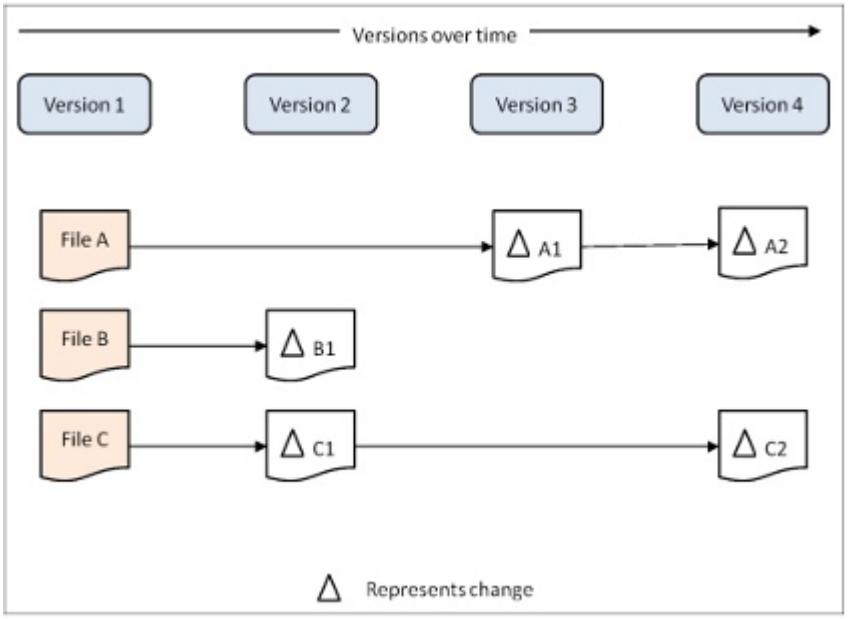
2.5 Git-versionhallintajärjestelmä

Git on Linus Torvaldsin ja Linux-kehittäjätiimin tekemä hajautettu versionhallintajärjestelmä Linux-ytimen kehittämisprojektille. Vuosien 1991 ja 2002 välillä Linux-ytimen ohjelmistopäivitykset siirtyivät eteenpäin lähinnä pakattuina tiedostoina. Vuonna 2002 Linux-ydin-projekti alkoi käyttää yksityistä hajautettua versionhallintajärjestelmää BitKeeperiä. Vuonna 2005 välit kuitenkin kiristyivät BitKeeperin yrityksen ja Linux-ytimen kehittäjätiimin välillä, jolloin BitKeeperiä ei enää haluttu käyttää lisenssiehtojen muututtua. Bitkeeperiä vastaavaa versionhallintajärjestelmää ei ollut saatavilla, joten tämä johti Gitin kehittämiseen BitKeeperin käytöstä opituilla asioilla. Tarkoituksena oli suunnitella täysin hajautettu versionhallintajärjestelmä, joka on yksinkertainen ja pystyy toimimaan nopeasti Linux-ytimen kaltaisissa isoissa ohjelmistoprojekteissa. (2, s. 31.)

Git on suunniteltu varmistamaan datan yhtenäisyys. Operaatioilla on vain kaksi mahdollista lopputulosta, ne joko onnistuvat tai epäonnistuvat. Git varmistaa, että operaatiot eivät päivitä vain osaa muutoksista virheen sattuessa. Tämä on erittäin tärkeää ympäristössä, jossa datan yhtenäisyydellä on merkitystä. Esimerkiksi pankkijärjestelmän siirrolla pitää olla vain kaksi mahdollista lopputulosta. Joko operaatiossa siirtyvät kaikki rahat, mitä pitkin, tai yhtään mitään ei siirry. Puolittaisten siirtojen tapauksissa datan yhtenäisyys häviää (1, s. 31).

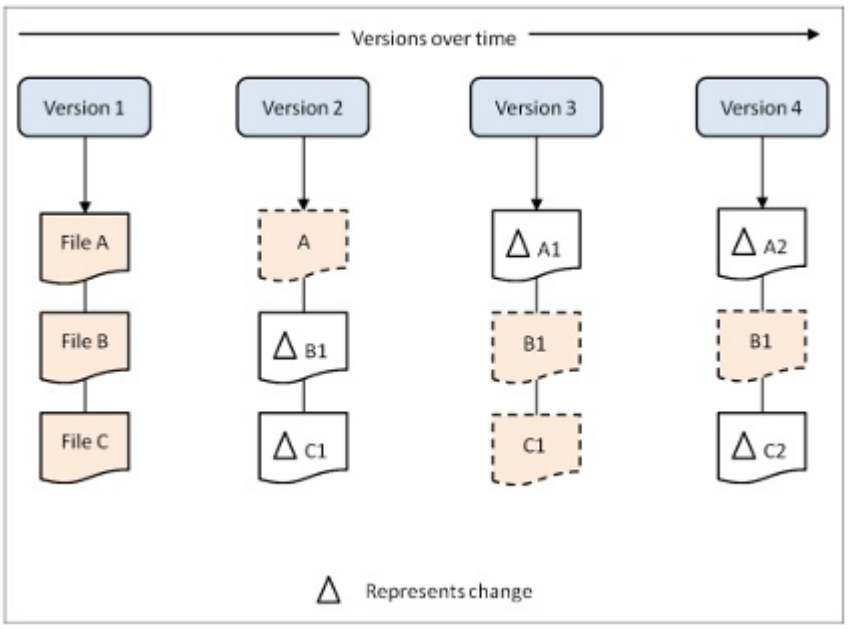
Verrattuna muihin hajautettuihin versionhallintajärjestelmiin Git on suunniteltu toimimaan nopeasti myös isojen tietokantojen yhteydessä. Muut versionhallintajärjestelmät pitävät

kirjaa tiedostoihin tehdyistä muutoksista tietyllä aikavälillä, kun taas Git ottaa tilannekatsauksen kaikista projektin tiedostoista ja tallentaa sen tietokantaan. Jos johonkin tiedostoon ei ole tehty muutoksia, Git osoittaa aikasemman tilannekatsauksen tiedostoon (2, s. 32). Tämä tarkoittaa, että Git käyttää mahdollisimman vähän tilaa projektin historian tallentamiseen. Tämän eron huomaa selvästi verrattuna toiseen hajautettuun versionhallintajärjestelmään Subversioniin, Projektit tallennetaan yksittäisiin kokonaisuuksiin, joita kutsutaan repoiksi (engl. repository). Testaamisessa Mozilla Firefox -internetselaimen lähdekoodi vei itsessään 2,7 gigatavua keskitetyssä versionhallintajärjestelmässä. Subversioniin siirrettäessä tämä koko kasvoi 8,2 gigatavuun, kun taas Gitiin siirrettäessä se pieneni vain 450 megatavuun. Tästä 450 megatavusta 350 kuului lähdekoodiin, jolloin koko projektin versioiden historia vei vain 100 megatavua. (1, s. 41.) Kuvasta 4 nähdään, kuinka muut versionhallintajärjestelmät pitävät listaa tiedostoihin tehdyistä muutoksista ja tallentavat vain nämä tehdyt muutokset tietokantaan. Git taas ottaa huomioon kaikki tiedostot ja niihin tehdyt muutokset.



Kuva 4. Muiden versionhallintajärjestelmien tapa hallinnoida eri versioita ja niihin tehtyjä muutoksia (1, s. 40).

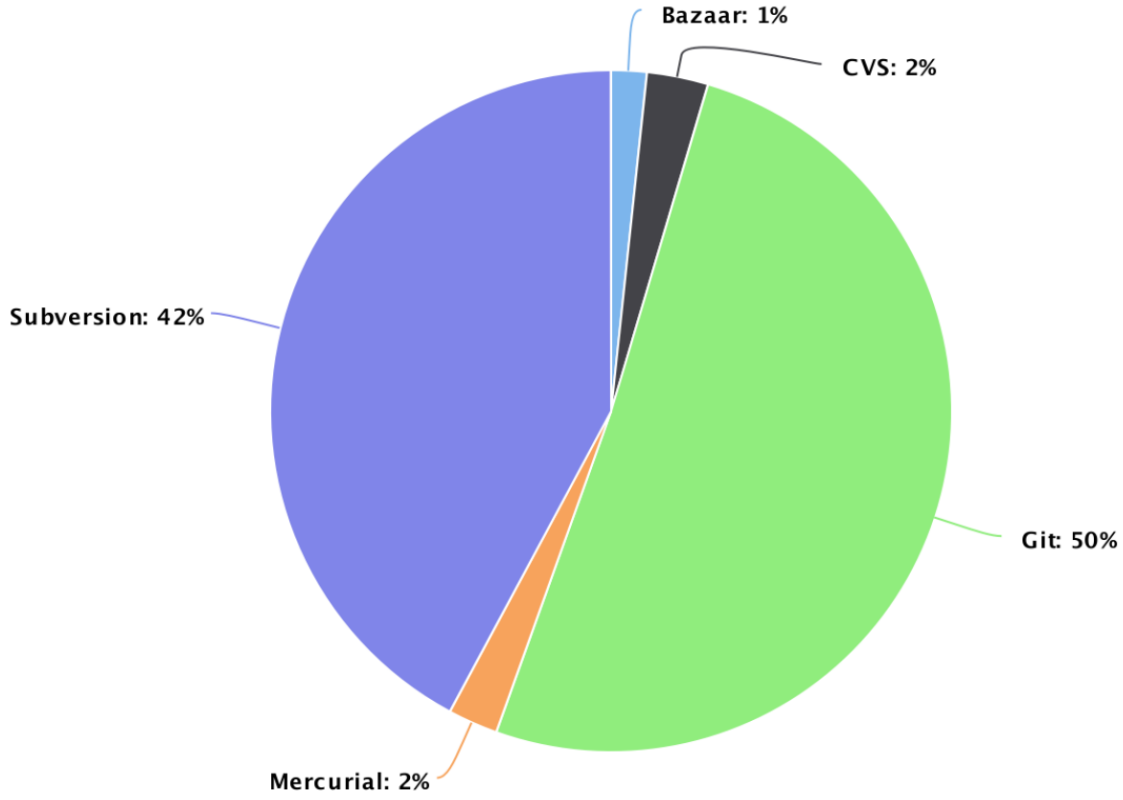
Jos uuden version tiedosto on identtinen aikaisemman version tiedoston kanssa, Git osoittaa aikaisempaan versioon eikä tallenna siitä uutta versiota. Tätä toimintaa nähdään tarkemmin kuvassa 5.



Kuva 5. Gitin tilannekatsaus kaikista tiedostoista eri versioiden välillä. Jos uuden version tiedosto vastaa aikaisempaa versiota, siitä ei tallenneta erikseen uutta tiedostoa. (1, s. 40.)

Jokaisen uuden muutoksen tapauksessa Git laskee sille tarkistussumman SHA-1 (Secure Hash Algorithm 1) -algoritmilla. Tämän jälkeen Git tunnistaa kyseisen version muodostetulla tarkistussummalla. Tämä tarkoittaa, että kukaan ei pysty tekemään muutoksia tiedostoon sen jälkeen, kun se on tallennettu, ilman että Git huomaa muutoksen. Kaikki tiedostot tallennetaan Gitiin tarkistussummalla eikä tiedostoon tehtyjen muutoksien perusteella, mikä varmistaa datan eheyden (2, s. 33).

Git on tällä hetkellä suosituin hajautettu versionhallintajärjestelmä, ja sen suosio on kasvanut viime vuosien aikana huomattavasti. Tällä hetkellä Gitin suosio on 50 % kaikista versionhallintajärjestelmistä ja seuraavana on Subversion 42 %:n osuudella (4). Kuvassa 6 on verrattu suosittuja versionhallintajärjestelmiä.



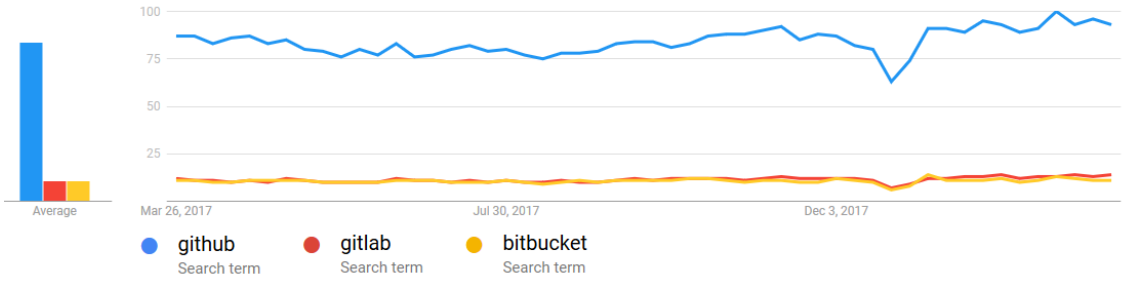
Kuva 6. Eri versionhallintajärjestelmien suosion vertailu OpenHub-sivuston mukaan (4).

2.6 Versionhallintapalvelut

Versionhallintapalvelut ovat webpohjaisia versionhallintajärjestelmien graafisia käyttöliittymiä. Ne auttavat ohjelmistoprojektin kehityksessä, ylläpitämisessä ja julkaisemisessa. Sen sijaan, että yritys pitäisi omaa palvelinta versionhallintajärjestelmälle, voidaan helposti käyttää kolmannen osapuolen palvelua, josta löytyy projektille tarpeelliset toiminnallisuudet. Niiden avulla pystytään myös reaaliaikaisesti seuraamaan projektin etenemistä helposti web-käyttöliittymien avulla. Osa versionhallintapalveluista tarjoaa myös muiden projektienhallintatyökalujen integraatiota suoraan versionhallintajärjestelmään. (5.)

Versionhallintapalvelua valitessa pitää verrata palvelun ominaisuuksia projektin tarpeisiin. Suurin osa versionhallintapalveluista tukee Gitin lisäksi myös muita versionhallintajärjestelmiä. Jotkut ovat avoimen lähdekoodin palveluita tai suosittuja avoimen lähdekoodien projekteissa, osa palveluista on suunniteltu enemmän yrityskäyttöön ja tarjoaa enemmän ominaisuuksia projektin ylläpitoon. Palveluista mikään ei ole niin sanotusti

toista parempi, vaan niiden hyödyllisyys riippuu projektista, jossa niitä käytetään. (6.) Tässä osuudessa käydään läpi kolme suosituinta Git-pohjaista versionhallintapalvelua, GitHub, GitLab ja Bitbucket, ja verrataan niiden ominaisuuksia erilaisissa käyttötapauksissa. Niiden perusominaisuudet ovat lähes identtiset, ja niiden erot tulevat käyttäjämääristä ja erikoistumisista eri projektityyppeihin. Näistä palveluista suosituin ja suurin on GitHub, jossa on yli 50 miljoonaa projektia (7). Kuvassa 7 on hahmotettu palveluiden suosiota Google-hakujen perusteella käyttäen Google Trends -palvelua.



Kuva 7. Google-hakujen suosio GitHubin, GitLabin ja Bitbucketin välillä (8).

2.6.1 GitHub

GitHub on vuonna 2007 aloitettu ja vuonna 2008 julkaistu Git-pohjainen versionhallintapalvelu, joka työllistää 752 henkilöä ja ylläpitää yli 77 miljoonaa projektia. Versionhallinnan kannalta projektit tukevat tavallisia Gitin komentoja. Niitä pystytään käyttämään joko komentoriviltä tai suurelta osalta kolmannen osapuolen palveluilta. Vaikka GitHub ei itsessään ole avoimen lähdekoodin palvelu, ominaisuuksiensa ansiosta se on tällä hetkellä suurin avoimien lähdekoodi-projektien palvelu. Se tukee ilmaisten julkisien projektien ylläpitoa ja tarjoaa käyttäjilleen mahdollisuuden katsoa ja ladata muiden käyttäjien julkisia projekteja. Rekisteröityneet käyttäjät pystyvät myös keskustelemaan projektista ja ehdottamaan siihen muutoksia. GitHub antaa mahdollisuudenylläpitää projektia joko GitHubin omilla palvelimilla tai yrityksen omalla palvelimella. (6; 9.)

GitHub on suunnattu enemmän julkisille projekteille, joissa lähdekoodi on kaikkien saatavilla. Käyttäjille annetaan rajaton määrä julkisia repoja ilmaiseksi, mutta yksityiset repot riippuvat lisenssihinnasta. Tämän takia GitHub soveltuu parhaiten projekteille, joissa lähdekoodi ei ole yrityksen sisäistä vaan se voidaan jakaa avoimesti kaikille GitHubin käyttäjille.

2.6.2 GitLab

GitLab vastaa päätoiminnoiltaan GitHubin ominaisuuksia, mutta se ei ole läheskään yhtä suosittu käyttäjien keskuudesta. Tämä nähdään jo kuvasta 7, jossa on verrattu kaikkien kolmen järjestelmän hakuja. GitLab on perustettu vuonna 2011, ja vuonna 2017 sillä oli yli puoli miljoonaa projektia. GitLab on suunnattu ominaisuuksiltaan enemmän suurille yrityksille kuin kilpailijansa. Se antaa mahdollisuuden käyttäjäryhmille, joiden perusteella voidaan antaa oikeudet eri projekteihin tai sen osa-alueisiin. GitHub taas antaa mahdollisuuden muuttaa käyttäjäoikeuksia käyttäjäkohtaisesti, mikä ei sovellu suurimmille yrityksille. Suuria yrityksiä, jotka käyttävät GitLabia, ovat esimerkiksi IBM, Sony ja NASA. GitLab tarjoaa rajattoman määrän julkisia sekä yksityisiä repoja, mutta lisenssihinta riippuu käyttäjämääristä. (10.)

2.6.3 Bitbucket

Bitbucket on vuonna 2008 julkaistu versionhallintapalvelu, joka alun perin tuki vain Mercurial-versionhallintajärjestelmää. Git-tuki lisättiin järjestelmään vuonna 2011, minkä jälkeen se on ollut palvelun pääkohde. Bitbucket suuntautuu ominaisuuksiltaan enemmän yrityksille kuin yksittäisten käyttäjien projekteihin. Toisin kuin GitHub ja GitLab, Bitbucket tukee myös ilmaisversioissaan suljettuja projekteja avoimien projektien lisäksi. Bitbucketin omistaa Atlassian-yritys, joka tekee muita projektinhallintaan liittyviä tuotteita, kuten Jira ja Confluence. Jira on Atlassianin kehittämä tikettien seurantajärjestelmä, joka tarjoaa erilaisia projektinhallintatyökaluja. Confluence mahdollistaa monen eri käyttäjän samanaikaisen työskentelemisen projektiin liittyvän dokumentaation kanssa. Koska yrityksen muut tuotteet on rakennettu suoraan integroitumaan Bitbucketin kanssa, ne tarjoavat huomattavasti paremmat projektinhallintamahdollisuudet kuin GitHub ja GitLab. (11.)

3 Integraatiojärjestelmät

3.1 Yleistä

Tietotekniikan alkuvaiheessa tiedon integraatiolle ei ollut olemassa mitään tarvetta. Yritykset ja niiden yksiköt kehittyivät itsekseen omissa haaroissaan kommunikoimatta muiden haarojen kanssa. Jos tuli jostain syystä tarvetta siirtää tietoa haarasta toiseen, se pystyttiin tekemään manuaalisesti siirtämällä tieto haarasta toiseen käsin. Esimerkiksi tiedostoja siirrettiin käsin sovelluksesta toiseen. Tietotekniikan kehittyessä myös sovellukset alkoivat kehittyä ja tiedon siirtoa sovelluksesta toiseen alettiin tarvita entistä enemmän. Tämä tarkoitti myös sitä, että tiedon syöttäminen käsin sovelluksesta toiseen ei enää ollut järkevää. (12, s. 6–8)

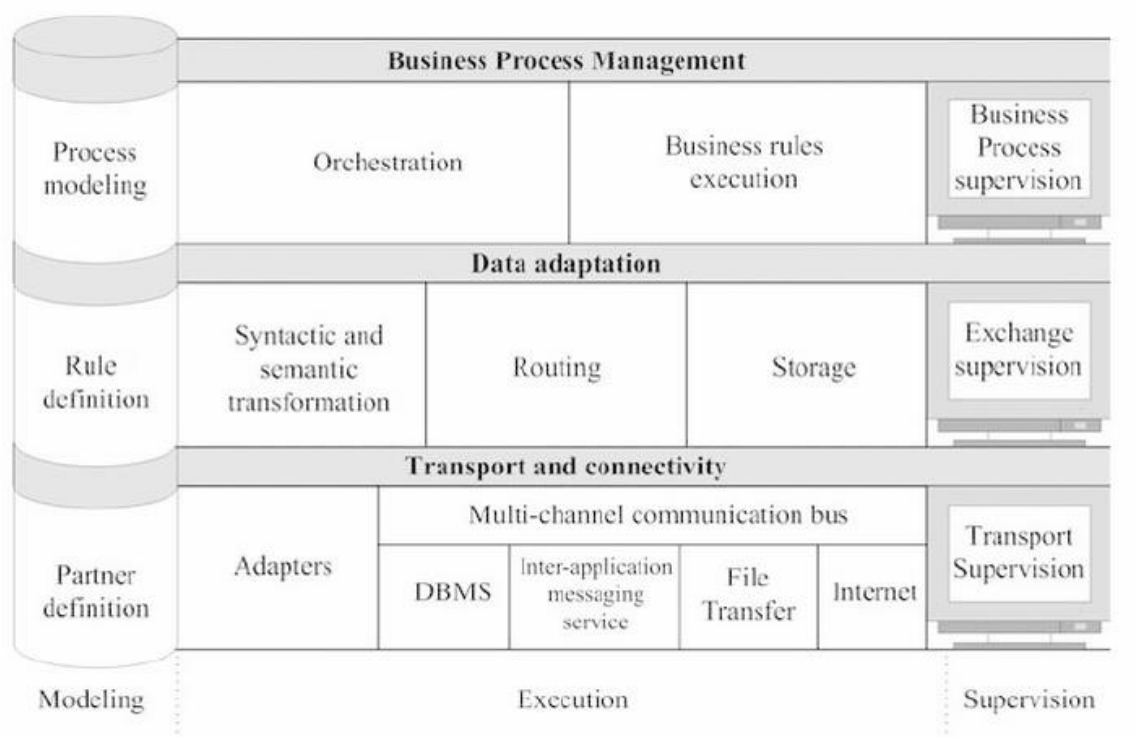
Sovelluksia alettiin yhdistää suoraan toisiin sovelluksiin, jotta tietoa ei tarvinnut enää syöttää käsiin. Tämän takana ei kuitenkaan ollut minkäännäköistä suunnitelmaa, vaan sovelluksia yhdistettiin keskenään tarpeen vaatiessa. Tämä johti yhteyksien kasvamiseen sovelluksien välillä, jotka eivät olleet suunniteltu toimimaan keskenään. Koska sovellukset vaativat suoria yhteyksiä kaikkien muiden sovelluksien välillä, yhteyksien määrät kasvoivat huomattavasti verrattuna sovelluksien määrään. (12, s. 12.) Kuvassa 8 on kaava, jolla lasketaan yhteyksien määrä verrattuna sovelluksien määrään, jos kaikki sovellukset tarvitsevat suoran yhteyden muihin sovelluksiin.

$$i = n\frac{n-1}{2}$$

Kuva 8. Yhteyksien määrä verrattuna sovelluksien määrään. i = yhteyksien määrä, n = sovelluksien määrä. (12, s. 12.)

Jos 6 sovellusta tarvitsee jo 15 yhteyttä, niin 150 sovellusta tarvitsisi yhteensä 11 175 yhteyttä. Tämä johtaa erittäin nopeasti ns. ”spagetti”systeemeihin, joihin on erittäin vaikeaa tehdä mitään muutoksia, koska ei olla perillä siitä, mitkä muutokset vaikuttaisivat mihinkin sovelluksiin (11, s. 12). Yhteyksien hallitsemiseen on suunniteltu useita kehyksiä, kuten EAI (Enterprise Application Integration) ja A2A (Application to Application). Integraatiojärjestelmät voidaan jakaa kolmeen tasoon niiden toiminnallisuuksien perusteella (kuva 9), ja ne voidaan jakaa vielä omiin alitasoihinsa. Päätasoihin kuuluu

tiedostojen haku ja siirto kohdejärjestelmiin, tietojen muokkaus sovellusten vaatimalla tavalla sekä koko prosessin automatisointi. (12, s. 29.)



Kuva 9. Integraatiojärjestelmien toiminnallisuudet jaettuna tasoihin ja alitasoihin (12, s. 30).

3.2 Tietojen haku ja siirto

Tietojen haun ja siirron taso keskittyy tiedon siirtoon lähdesovelluksesta eteenpäin kohdesovelluksiin. Lähdesovelluksesta odotetaan tietoa tai tapahtumia, joista haluttu tieto generoidaan. Halutut tiedot siirretään eteenpäin kohdesovelluksiin, jotka prosessoivat tiedot eteenpäin. Tämän jälkeen voidaan vielä mahdollisesti siirtää tietojen prosessoimisesta syntyvät tapahtumat takaisin lähdesovelluksiin. Jotta tietoja pystytään siirtämään sovelluksesta toiseen, tarvitaan kaikkien sovellusten välisten parametrien määrittämiset etukäteen tietokantaan.

Parametreihin sisältyvät muun muassa sovellusten väliseen topografiaan liittyvät tiedot: Miten sovelluksiin saadaan yhteydet? Missä kyseistä sovellusta ylläpidetään? Onko se saman yrityksen sisällä vai onko kyseessä B2B (Business to Business) -integraatio yritykseltä toiselle? Jos kyseessä on B2B-integraatio, niin tarvitaan myös osoitteet

yrityksen ympäristöön. Mitä protokollia käytetään yhteyden muodostamiseen? Protokollat voivat vaihdella esimerkiksi HTTP (Hypertext Transfer Protocol) -yhteydestä SMTP (Simple Mail Transfer Protocol) -yhteyteen tai tiedostojen siirtämiseen FTP:n (File Transfer Protocol) avulla. Jos sovelluksessa otetaan yhteys tietokantaan, mitä tietokantaa käytetään ja mikä on siihen liittyvä yhteydenottolause? Mitkä ovat tietokannassa olevien taulujen nimet ja rakenteet? Mitkä ovat siirrettävien tiedostojen nimet ja polut niiden osoitteisiin? (12, s. 31.)

Tarvitaan myös tieto siitä, miten haluttu tieto siirretään sovelluksesta toiseen. Käytetäänkö ajastusta, jolloin jokin tiedosto siirretään sovelluksesta toiseen tai käydään hakemassa kohdesovelluksesta aina samalla aikavälillä? Katsooko integraatio aina kyseistä kansiota ja aktivoituu, kun kansioon on siirretty tiedosto, jota halutaan käsitellä? Pitää myös ottaa huomioon tietoihin liittyvät turvallisuuspuolet. Jos integraatiossa käsitellään salasanoja tai arkoja tietoja, niiden pitää olla suojattuna samalla tasolla kuin lähde- ja kohdesovelluksissa. Sovelluksien välisissä yhteyksissä voidaan käyttää autentikaatiota, ja tiedostot voidaan salata siirron yhteydessä. Integraation pitää tukea monia eri tapoja siirtää tiedostoja kohteesta toiseen ja ottaa huomioon niiden turvallisuus integraation tarpeisiin verrattuna. (12, s. 32–36.)

Sovellukset on yleensä rakennettu vastaanottamaan tai lähettämään tietoja tiettyihin osoitteisiin, jotka eivät yleensä ole suunniteltu suoraan integraatiojärjestelmälle. Integraatiojärjestelmän pitää vastaanottaa sovelluksen tiedot samassa muodossa, kuin ne on lähetetty. Tiedot eivät saa muuttua ei-halutulla tavalla sovelluksien välisessä siirrossa. Integraation pitää myös tarjota tiedot kohdejärjestelmälle sen haluamassa muodossa. Integraatiojärjestelmän tavoite on yhdistää sovellukset keskenään siten, että sovelluksia ei tarvitse muokata, jotta niiden kommunikoiminen keskenään on mahdollista. Tämä toteutetaan adaptereilla eri sovellusten väliin, jotta ne pystyvät kommunikoimaan keskenään, esimerkiksi API (Application Programming Interface) -rajapinnoilla, joiden avulla sovellukset pystyvät ottamaan yhteyden ennalta määritettyyn rajapintaan. (12, s. 42-43.)

Esimerkiksi sovelluksella, joka muodostaa tiedostoja, ei välttämättä ole toiminnallisuutta siirtää tiedostoja eteenpäin. Tiedostot muodostetaan tiettyyn kansioon, mutta ne pitäisi siirtää sen jälkeen toisen sovelluksen tietokantaan, josta ne prosessoidaan eteenpäin. Tähän väliin voidaan sijoittaa adapteri, joka valvoo kansiota. Jos kansioon siirtyy määritetyn niminen tai muotoinen tiedosto, se käydään hakemassa kansiosta ja siirretään eteenpäin tietokantaan. Tässä välissä voidaan myös tehdä muutoksia tiedoston tyyppiin

tai tietoihin, jotta ne sopivat kohdejärjestelmään. Nämä muutokset tehdään ennalta määritettyjen sääntöjen perusteella. Samanlainen tapahtuma voi myös olla kohdistettu tietokantaan, josta käydään hakemassa tauluun tehdyt muutokset ja siirretään lisätyt rivit toiseen sovellukseen tai jopa päivitetään tietokannan muutettuja rivejä. Adapterin pitää myös pystyä vertaamaan haettua tietoa aikaisemmin määritettyihin sääntöihin ja validoimaan sitä. Jos tieto on vääräntyyppistä, voidaan sen perusteella generoida haluttu virhe. (12, s. 43–49.)

Sovellusten välisiä siirtoja pitää myös pystyä seuraamaan mahdollisten virheiden takia. Jos sovellukset ovat yhteydessä toisiinsa ja toiselle osapuolelle tulee aikakatkaisu esimerkiksi tietokantakyselyn yhteydessä, pitää integraation osata tehdä tästä oikeanlainen virheviesti. Virheviestin perusteella pystytään selvittämään, mistä virhe johtui ja joko ajamaan siirto uudestaan tai korjaamaan virhe. Tähän pitää olla jonkinnäköinen graafinen käyttöliittymä, josta siirtojen statusta pystytään seuraamaan. (12, s. 50–51.)

3.3 Tietojen muokkaus

Integraation kannalta pelkkä tietojen hakeminen ja siirto sovelluksesta toiselle ei ole riittävää. Pitää olla myös mahdollista tehdä halutut muutokset tietoon, jotta se pystytään hyväksymään kohdejärjestelmässä. Muokkauksen lisäksi pitää ottaa huomioon tapahtumista muodostuneet tiedot, jotka halutaan siirtää eteenpäin tietyllä aikavälillä. Integraation pitää osata päätellä lähdejärjestelmästä saatujen tietojen perusteella, mitä muutoksia tietoon pitää tehdä ja missä muodossa se pitää lähettää eteenpäin kohdejärjestelmään. Sen pitää myös osata päätellä, milloin lähdejärjestelmä haluaa tiedot.

Todennäköisesti yhdistettävät sovellukset eivät molemmat tuota tietoa samassa muodossa. Lähdejärjestelmä lähettää siirrettävät tiedot ulos omassa muodossaan, minkä jälkeen integraatio muokkaa ne vastaamaan kohdejärjestelmän tarvitsemaa tietoa ennalta määritettyjen sääntöjen perusteella. Jos sääntöjä ei ole määritelty, integraatio ei osaa tehdä haluttuja muutoksia. Tiedot voivat liikkua sovellusten välillä monilla eri tavoilla. Yksinkertaisin ja vanhin muotoilu tiedon esittämiselle on ns. tasainen tai muotoilematon tiedosto (engl. flat file). Tässä tavassa tieto esitetään yhdessä putkessa, jossa jokainen yksittäinen tieto on aina samanpituinen jokaisessa tiedostossa. Nämä pituudet ovat ennalta määritettyjä, jotta integraatio osaa poimia tiedot oikeasta kohtaa. Jos tieto ei vastaa ennalta määritettyä pituutta tai sillä ei ole arvoa, sen pituutta lisätään sovitulla merkillä.

Tämä voidaan tehdä esimerkiksi välilyönneillä tai nollilla. (12, s. 53–54.) Kuvassa 10 on esimerkki flat file -tiedostosta, jonka arvot on eroteltu välilyönneillä.

```
0001    11001    Etunimi    Sukunimi          Osoite
0002    10011    Etunimi    Sukunimi          Osoite
0003    01111    Etunimi    Sukunimi          Osoite
```

Kuva 10. Flat file -hahmotelma, jossa arvot on eroteltu välilyönneillä.

Tästä muotoilusta on myös erilaisia versioita, jotka erottelevat tietoa hieman eri tavoilla. Jos tiedon pituus vaihtelee eikä haluta käyttää täytemerkkejä, voidaan tiedon pituus antaa aina ennalta määritetyllä tavalla ennen tietoa. Tällöin integraatio lukee ensin tiedon pituuden ja määrittää sen avulla, mistä kohtaa tieto luetaan. Hieman enemmän käytetty tapa olisi erottaa tiedot jollain tietyllä merkillä, kuten esimerkiksi pilkulla tai puolipisteellä. Tällöin tiedot erotellaan merkin perusteella ja integraation pitää tietää, kuinka monennesta kohdasta haluttu arvo löytyy. Tiedon siirrossa voidaan myös käyttää XML (eXtensible Markup Language) -muotoa, jos halutaan käyttää tarkempaa rakennetta tietojen välillä. (12, s. 54–57.)

Tietojen muotoiluvalinta vaikuttaa myös integraation tehokkuuteen. Mitä monimutkaisempi muotoilu, sitä hankalampaa on määrittää, mistä kohtaa halutut tiedot pitää hakea, minkä takia suoritusaika lisääntyy. Mitä yksinkertaisempi muotoilutapa, sitä nopeampaa on tiedon hakeminen. Tiedon muokkaamisen lisäksi integraation pitää myös osata päätellä, mihin kohdejärjestelmään nämä muokatut tiedot halutaan lähettää. Jos sovellukset olisi yhdistetty suoraan keskenään, tästä ei tulisi ongelmaa, koska tiedettäisiin automaattisesti, mihin osoitteeseen tiedot lähetetään. Koska integraatio voi yhdistää yhden sovelluksen tiedot moneen eri sovellukseen lähetettyjen tietojen perusteella, tarvitaan jonkinnäköinen sääntö, jonka perusteella päätellään mihin tieto lähetetään.

Lähdesovellustapahtuma siirtyy integraatiolle. Tämä voi olla esimerkiksi tiedoston luonti kansioon jollakin nimellä, jolla integraatio tunnistaa sen. Tietokannassa on ennalta määritetyt säännöt, joiden perusteella tiedostoon tehdään halutut muutokset ja siirretään eteenpäin kohdesovellukseen. Jos sovellus lähettää vain yhdenkaltaisia tietoja, voidaan tapahtumaa automaattisesti käsitellä tiettyyn kohdesovellukseen. Jos taas sovelluksesta lähtee eteenpäin monia eri tapahtumia, tarvitaan tarkempia sääntöjä, joiden perusteella

tapahtumat eritellään eri kohdejärjestelmiin. Tämä voi tapahtua esimerkiksi jonkin tietyn kentän arvon perusteella. Integraatio päättelee lähdesovelluksen ja tietojen perusteella, mihin kohdejärjestelmään tiedot lähetetään. (12, s. 62–65.)

Jos esimerkiksi lähdesovelluksen tiedoissa on kenttä, jonka perusteella määritetään, mihin kohdesovellukseen muutetut tiedot lähetetään, kenttä voi esimerkiksi saada arvot 1, 2 tai 3. Integraatio lukee kentän arvon ja sen perusteella haarautuu kohdesovelluksen vaatimaan logiikkaan. Tällöin lähdesovellus pystyy lähettämään monen eri sovelluksen yhteydet samaan integraatioon, minkä jälkeen integraation sisällä haaraudutaan kyseisen kentän perusteella. Suoria yhteyksiä jokaisen sovelluksen välillä ei enää tarvita. Integraation pitää myös tarjota jonkinnäköinen tapa tallentaa tietoa väliaikaisesti sovellusten välisellä siirrolla, jos esimerkiksi kohdejärjestelmä on ajastettu hakemaan kaikki päivittyneet tiedot tietyllä aikavälillä. Tämä voidaan toteuttaa esimerkiksi väliaikaisesti tiedostoon tai tietokantaan tallentamalla (12, s. 65).

3.4 Prosessien automatisointi

Integraation eri osa-alueet sovellusten välisille yhteyksille voidaan koostaa yhteen prosessiin. Prosessi kattaa tapahtuman vastaanottamisen lähdesovellukselta, kaikki muutokset tietoihin ja sen perusteella päätelmät tapahtuman lähettämisestä kohdesovellukseen. Tätä ajattelua kutsutaan prosessijohtamiseksi (engl. Business Process Management, BPM).

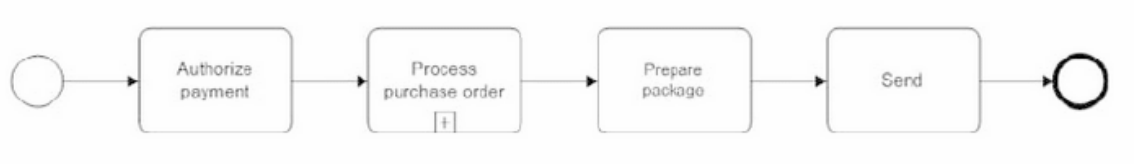
WFMC (Workflow Management Coalition) määrittää BPM:n seuraavanlaisesti:

> Business Process Management (BPM) is a discipline involving any combination of modeling, automation, execution, control, measurement and optimization of business activity flows, in support of enterprise goals, spanning systems, employees, customers and partners within and beyond the enterprise boundaries (13).

Esimerkiksi pankin lainajärjestelmä voidaan sisällyttää yhteen prosessiin. Prosessissa kutsutaan eri alijärjestelmiä, joista haetaan lainan antamiseen tarvittavat tiedot. Prosessi voi olla myös yhteydessä moneen eri yritykseen, joista haetaan prosessin kuluessa erilaisia tietoja. Prosessin lopussa nämä tiedot yhdistetään ja lähetetään lähdesovellukselle.

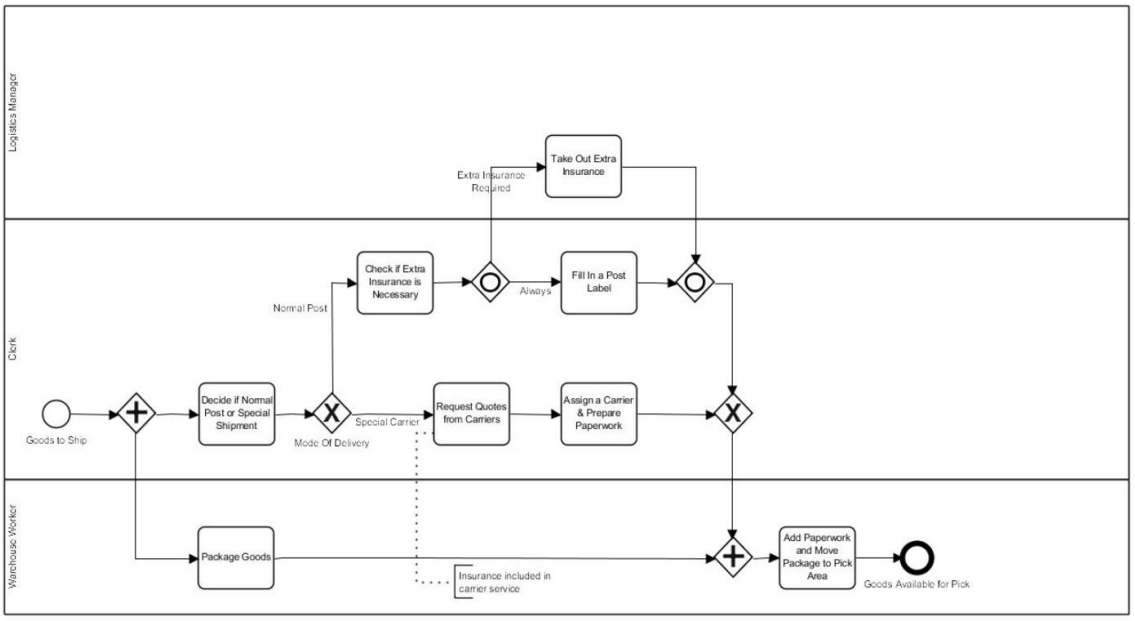
Prosessi pitää kaavoittaa siten, että se pystytään automatisoimaan integraation toteutuksessa. Kaavoitus määrittää prosessin eri toiminnallisuudet, niiden välisen logiikan, prosessin alku- ja loppupisteet sekä jokaisen vaiheen sisältämät tiedot. Prosessien vaiheiden tietoihin voi sisältyä esimerkiksi siirrettävä data ja se, mihin järjestelmään se kohdistuu. Vuonna 2004 BPMI (Business Process Management Initiative) julkaisi ensimmäisen version graafisesta notaatiosta nimeltä BPMN (Business Process Management Notation), jolla pystytään kuvailemaan prosessia BPD-diagrammeilla (Business Process Diagram). BPMN ei ota kantaa prosessin toteutuksessa käytettävään kieleen. Sen tarkoituksena on havainnollistaa, miten prosessi menee eteenpäin vaiheesta vaiheeseen. Tämä toteutetaan BPEL-kielellä (Business Process Execution Language), joka määrittää logiikan tiettyjen notaatioiden välillä. (12, s. 73–75.)

Kuvassa 11 on esimerkki yksinkertaisesta BPMN-tyyppisestä prosessista, joka käsittelee jonkinnäköistä maksutapahtumaa. Se ei ota yhteyttä muihin sovelluksiin, vaan kaikki prosessissa tapahtuva logiikka suoritetaan sen sisällä. Prosessilla on määriteltynä alku- ja loppukohdat. Se siirtyy vaiheesta vaiheeseen ilman suurempaa logiikkaa.



Kuva 11. Esimerkki prosessista, jossa käsitellään maksutapahtuma ja lähetetään se eteenpäin seuraavalle sovellukselle (12, s. 82).

Prosessissa voitaisiin myös käsitellä monimutkaisempaa logiikkaa, jonka perusteella se haarautuu eri vaiheisiin. Tätä on kuvattu tarkemmin kuvassa 12. Tarkempaa tietoa kaikista BPMN:n ominaisuuksista on sen kotisivulla bpmn.org (14).



Kuva 12. Esimerkki BPMN-notaatiosta, joka haarautuu eri osiin logiikan perusteella (16).

Prosessin kaavoittaminen olisi parasta tehdä graafisessa käyttöliittymässä. Graafisen käyttöliittymän avulla myös muut kuin koodaajat pystyvät olemaan osallisena prosessin määrittelemisessä. Tällöin prosessin arkkitehtuuri ja itse alla oleva koodaus pystytään erottelemaan eri asiantuntijoille. Prosessin logiikka pystytään määrittelemään etukäteen niillä henkilöillä, jotka tuntevat prosessin tarpeet ja määritykset, mutta eivät välttämättä osaa koodata itse sovelluksen toimintoja. BPMN-notaatiolla piirretyn kaavion jälkeen jokainen vaihe voidaan jälkeenpäin tehdä kaavion määrittämällä tavalla. Tämä mahdollistaa tarkempien määrittelyjen antamista ennen itse toteutuksen aloitusta. Kun prosessi on kaavoitettu ja sen vaiheet on tehty, prosessia voidaan ajaa läpi. Tällöin kaavoituksen mukaista logiikkaa seurataan vaihe vaiheelta. (12, s. 85–86.)

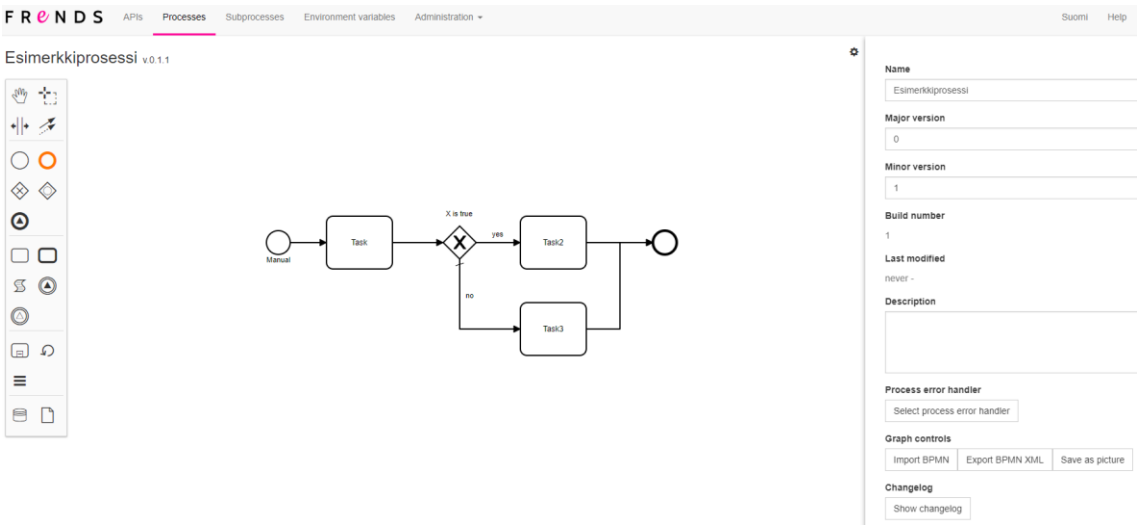
Prosessien automatisointiin liittyy myös prosessien valvominen ja monitorointi. Graafisen liittymän avulla pystytään valvomaan prosessien eri ominaisuuksia: menevätkö prosessit kaavoituksen mukaisesti läpi ilman virheitä sekä prosessin eri vaiheet suorituksen yhteydessä ja noudattaako prosessi annettuja aikamääreitä eri yhteyksien muodostamisessa. Virheiden tapauksessa hälytetään monitoroinnin valvojille, jotka pystyvät tarkistamaan virheen ja korjaamaan sen. Tätä kutsutaan BAM (Business Activity Monitoring) -menetelmäksi, jossa seurataan reaaliaikaisesti prosessien tehokkuutta ja ilmoitetaan käyttäjille, jos huomataan jotain normaalista toteutuksesta poikkeavaa. (12, s. 88–90.)

4 FRENDS-integraatiojärjestelmä

4.1 Yleistä

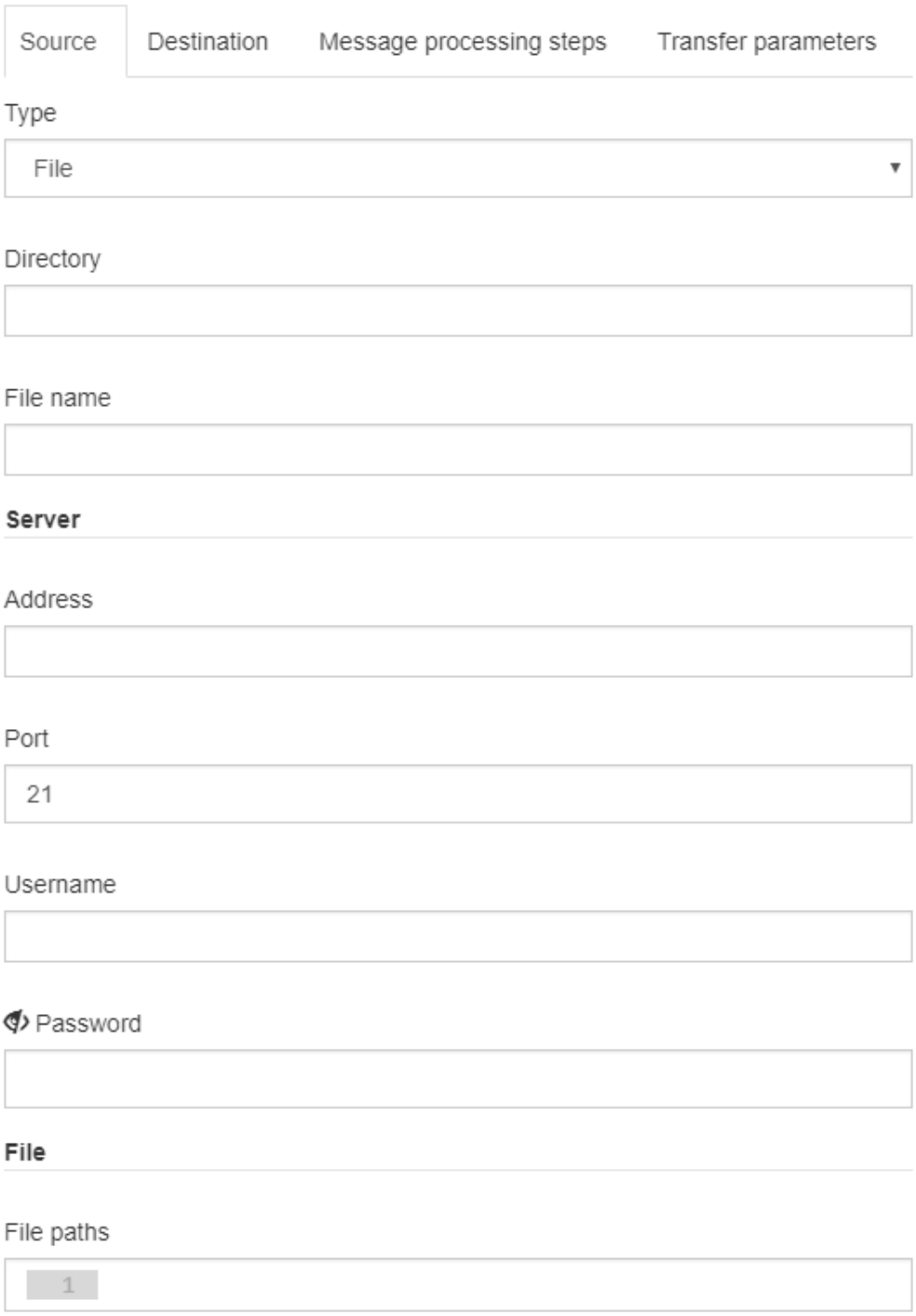
FRENDS on HiQ Finlandin kehittämä ja ylläpitämä integraatiojärjestelmä, joka on suunniteltu mahdollisimman helposti käytettäväksi. Integraatioiden toteutus tehdään graafisella käyttöliittymällä, joka noudattaa luvussa 3.4 läpikäytyä BPMN-notaatiota. Integraatiojärjestelmään on myös sisäänrakennettu valvonta ja monitorointi, jolla pystytään seuraamaan prosessien suoritusta suoraan graafisesta käyttöliittymästä. Suomessa integraatiojärjestelmää käyttäviä yrityksiä ovat muun muassa S-pankki, Mtv3, Rovio, Helen ja Destia. (15.)

Kuvassa 13 on kuvattu FRENDSin käyttöliittymää prosessin kaavion tekovaiheesta. Koska kaaviot tehdään BPMN-notaatiolla, kaikki BPMN:n tukemat operaatiot ovat saatavilla. Kuvan laatikot kuvaavat prosessin eri vaiheissa suoritettuja operaatioita. Ne voivat olla esimerkiksi tiedostojen siirtoja, tietokannan taulujen lukemista tai viestien lähettämistä toisiin rajapintoihin. Nämä laatikot ovat C#-kielellä toteutettuja operaatioita, joita kutsutaan FRENDSin taskeiksi (engl. task, suom. yksittäinen tehtävä tai toiminnallisuus). FRENDSiin sisältyy jo valmiiksi toteutettuja taskeja, joiden avulla voidaan toteuttaa suurin osa yksinkertaisimmista prosesseista ilman minkäänlaista koodausosaamista. FRENDSin käyttöliittymä antaa tekijälle ennalta määritetyt parametrit, joille voidaan antaa arvoja. Prosessin suorituksen yhteydessä arvot luetaan taskin C#-koodin sisään, minkä jälkeen se toteutetaan. Tämän avulla prosessien luominen on erittäin helppoa, eikä käyttäjä tarvitse aikaisempaa C#-koodin osaamista.



Kuva 13. FRENDS-käyttöliittymä prosessin luomiselle.

Mikäli FRENDSin valmiiksi rakennettujen taskien perusteella ei pystytä toteuttamaan yrityksen tarvitsemaa prosessia, pystytään C#:n avulla tekemään oma task, jossa on tarvittavat ominaisuudet, ja lisäämään se helposti FRENDSin käyttöliittymään. Kuvassa 14 on esitetty task graafisen käyttöliittymän puolelta. Siinä näkyy taskin nimi, sille annettavat parametrit ja se, missä muodossa parametrit annetaan.



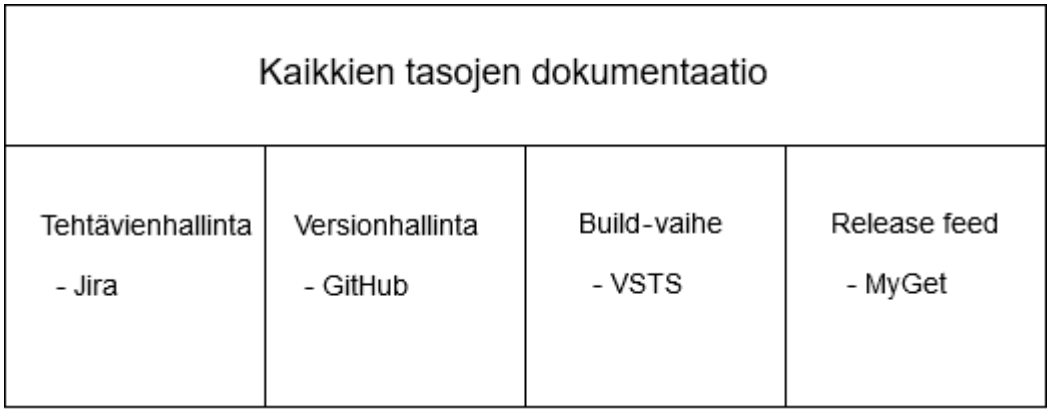
Kuva 14. FRENDS-task käyttöliittymän puolelta.

FRENDSissä on sisäänrakennettu versionhallinta jokaiselle prosessille. Kaikki aikaisemmat versiot tallennetaan sisäiseen tietokantaan, josta niihin voidaan palata missä vaiheessa tahansa. Tämä mahdollistaa eri versioiden ylläpidon eri ympäristöissä. Valmiiksi rakennettuja taskeja FRENDSiin taas on olemassa kohtuullinen määrä. Taskeja ylläpidetään NuGet-paketeilla, joiden avulla voidaan jakaa taskien koodipaketteja. NuGet-

paketit ladataan FRENDSin ympäristöön, jolloin taskia pystytään käyttämään prosessin tekovaiheessa.

Tästä seuraa kuitenkin versionhallinnallisia ongelmia, joihin tässä työssä yritetään etsiä mahdollisia vastauksia. Miten pystytään hallinnoimaan kaikkia eri taskeja ja niiden kaikkia versioita? Miten pidetään huoli taskien dokumentaatiosta? Miten niille saadaan automatisoitua testit ja päivitykset? Miten asiakkaat saavat haettua taskeista päivitetyt versiot tai kokonaan uudet taskit?

Versionhallinnan kokonaisratkaisua on yrityksessä lähdetty selvittämään kuvan 15 hahmottamalla tavalla. Kuvassa näkyvät FRENDSissä käytetyt versionhallinnalliset vaiheet. Seuraavaksi käydään vaiheet yksitellen läpi ja selvitetään tämänhetkiset FRENDSin käyttämät tavat ja se, miten ne voitaisiin mahdollisesti tehdä toisella tavalla.



Kuva 15. FRENDS-versionhallintakokonaisuuden hahmottelua.

4.2 Dokumentaatiot

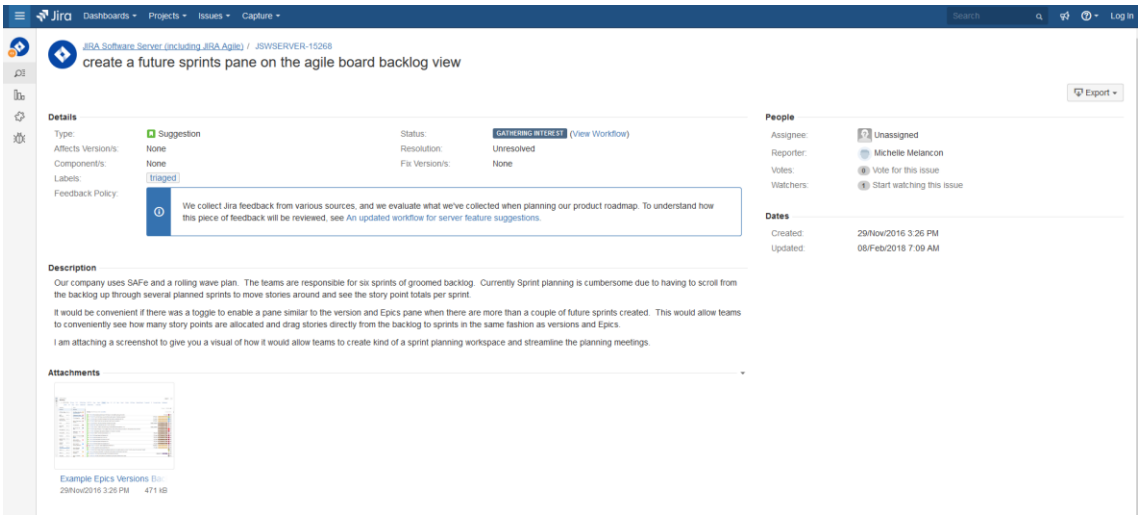
Kaikkiin FRENDSiin liittyviin dokumentaatioihin käytetään Atlassianin Confluence-ohjelmistoa. Se on tiimipohjainen dokumentaatio-ohjelmisto, jota jokainen tiimin jäsen pystyy muokkaamaan suoraan webselaimella, jos käyttäjälle on annettu oikeudet sivuun. Se vastaa toiminnoltaan esimerkiksi Wikipedian rakennetta, jota kaikki käyttäjät pääsevät muokkaamaan ja lisäämään siihen materiaalia. Ainoa poikkeus dokumentaatiossa ovat itse taskien dokumentaatiot, jotka on hallinnoitu suoraan versionhallinnan puolella.

Confluence tarjoaa käyttäjilleen valmiiksi rakennettuja dokumentaatiopohjia uusien sivujen luomisen helpottamiseksi. Tekstipohjaisen dokumentaation lisäksi Confluencesta löytyy valmiita mediapohjaisia työkaluja esimerkiksi kaavioiden piirtämiseen. Confluencessa on valmis versionhallinta jokaiselle sivulle ja sivuilla esiintyville liitteille. Koska FRENDSissä käytettävien versionhallinnallisten työkalujen mukana on myös muita Atlassianin tekemiä ohjelmistoja, todennäköisesti Confluence on tässä käyttötapauksessa paras dokumentaatiotyökalu. Se tarjoaa integraatiomahdollisuuksia esimerkiksi Jiran kanssa.

4.3 Tehtävienhallinta

Tehtävienhallinnalla tarkoitetaan tässä tapauksessa jotain Issue Tracking -seurantajärjestelmää, jolla pystytään hallinnoimaan projektin eri vaiheita ja seuramaan projektissa ilmeneviä ongelmia. Tällä hetkellä FRENDSin eri projekteissa käytetään Atlassianin Jira-tehtävienhallintaohjelmistoa. Atlassian julkaisi Jiran vuonna 2002, ja se on yrityksen ensimmäinen tuote. Jira tarjoaa käyttäjilleen erilaisia seurantajärjestelmän ominaisuuksia, joilla pystytään seuraamaan projektin eri vaiheita. (16.)

Projektit on jaettu pienempiin kokonaisuuksiin, jotka keskittyvät yhden asian seuraamiseen. Näitä kokonaisuuksia kutsutaan tiketeiksi. Yhdellä tiketillä voi olla esimerkiksi löydetyn virheen seuraamista sen korjaukseen asti, uuden toiminnallisuuden toteutusta tai uuden muutoksen seuraamista. Tiketillä pystytään määrittämään eri vaiheet, jotka kuvaavat tiketin kulkemista alkutilanteesta lopputilanteeseen. Näihin voivat kuulua esimerkiksi toteutus, testaus ja tuotantoon siirto. Tiketeille pystytään lisäämään Confluencen tyyliin dokumentaatiota ja liitteitä. Vaiheiden siirtymisen yhteydessä pystytään myös dokumentoimaan, kuinka paljon aikaa tiettyyn vaiheeseen kului toteutuksessa. Tiketeille pystytään myös kirjoittamaan kommentteja, joilla voidaan tarkemmin selvittää tiketin eri vaiheita. Kuvassa 16 on esimerkki Jiran käyttöliittymästä.



Kuva 16. Esimerkki Jiran käyttöliittymästä (17).

Jira tarjoaa myös erilaisia työkaluja projektien eri tikettien järjestämiseen halutuilla tavoilla. Näillä pystytään seuraamaan projektin eri vaiheita esimerkiksi kaikkien ongelmatikettien näyttämisellä yhdessä paikkaa. Tikettejä pystytään myös etsimään nimen perusteella tai tekemään omia suodattimia, joiden perusteella etsitään vain tietynlaisia tikettejä.

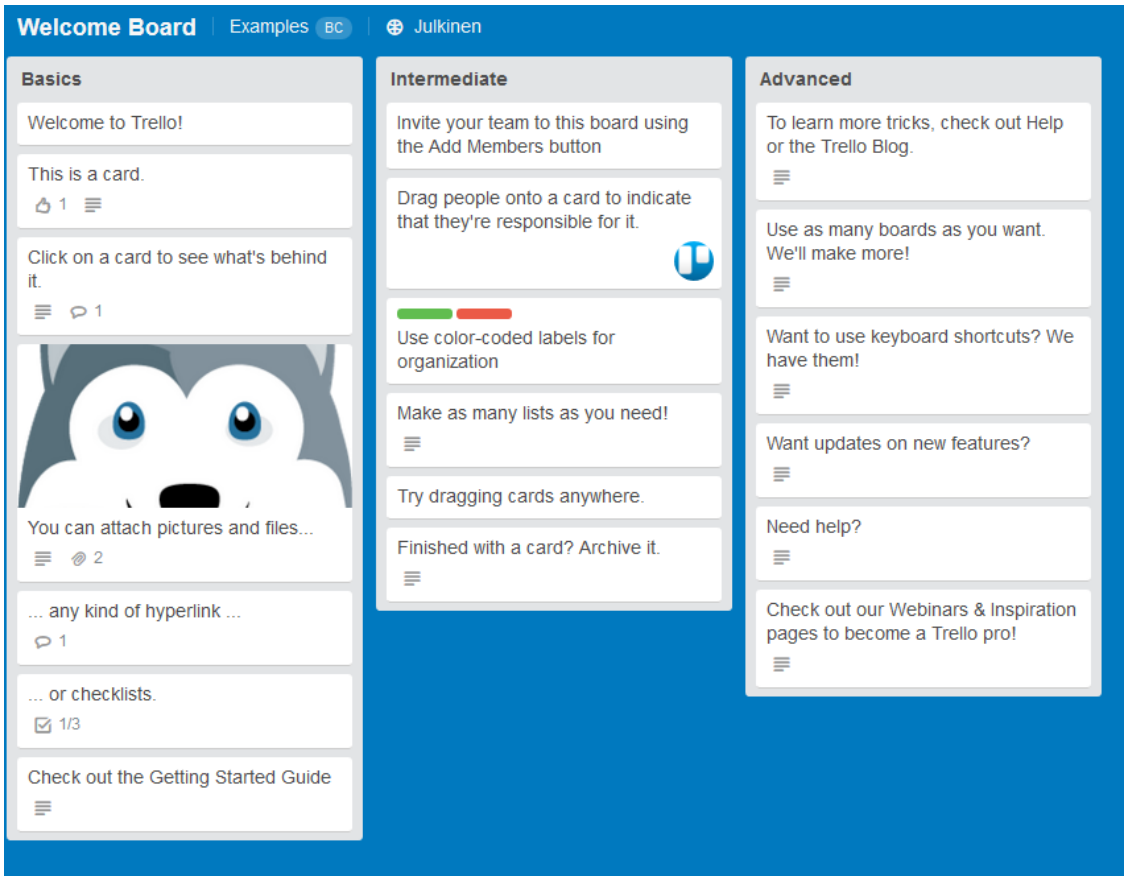
Koska Jira on Atlassianin tekemä tuote, se integroituu kätevästi myös Confluencen dokumentaation kanssa. Confluencen projektin määritysdokumenteista pystytään esimerkiksi tekemään suoraan Jira-tikettejä, jotka kuvaavat tarvittavat muutokset tai toteutukset. Jiran tiketeiltä pystytään myös päivittämään automaattisesti Confluencen dokumentaatiosivuja esimerkiksi tuotantoon siirrettyjen projektien julkaisuilmoituksilla.

Jira ei ole pelkästään virheiden ja tikettien seuraamisjärjestelmä, vaan se tukee myös muita mukautettuja tarpeita eri projekteille. Jiralle on olemassa yli 3 000 erilaista sovellusta, joilla voidaan lisätä sen toiminnallisuutta, jos sitä ei ole valmiiksi Jiran tarjoamissa ominaisuuksissa. Eri vaiheita pystytään myös automatisoimaan Jiran sisällä erinäisten API:en avulla (16). Tämä voi olla kuitenkin melko hankalaa, sillä Jira on alun perin suunniteltu tikettien seuraamiseen. Erilaisia ominaisuuksia on paljon, ja ne saattavat olla yhteydessä keskenään, vaikka se ei olisikaan itsestään selvää. Tällöin ominaisuuksien muuttaminen saattaa vaikuttaa asioita, joita ei alun perin otettu huomioon. (15.)

Ominaisuuksien määrä voi olla myös Jiran huono puoli, sillä niiden määrä ei tee ohjelmistosta helppokäyttöistä uusille käyttäjille. Tämän huomaa etenkin, jos Jiraa käytetään

mukautettuihin käyttötapauksiin, jossa alkuperäistä toiminnallisuutta on muutettu sovelluksilla. Tämä tarkoittaa, että Jira vaatii uusilta käyttäjiltä enemmän aikaa oppia käyttämään ohjelmistoa verrattuna kilpaileviin tuotteisiin. Myös Jiran käyttöönotto voi viedä enemmän aikaa kuin muiden ohjelmistojen, jos alkuperäistä toiminnallisuutta halutaan lisätä aiemmin mainituilla sovelluksilla (18).

Ominaisuuksia saattaa myös olla liian paljon pienille yrityksille, jotka tarvitsevat vain pientä osaa Jiran tarjoamista ominaisuuksista. Vaihtoehtoja Jiralle ovat esimerkiksi myös Atlassianin omistama Trello, Countersoftin Gemini tai Mozillan Bugzilla. Trello on erittäin nopea ja yksinkertainen projektinhallintaohjelmisto. Kun Jira on suunniteltu enemmän ohjelmistokehittäjille, Trello tukee kaikenlaisia projekteja. Trello on käyttäjille helpompi oppia ja saattaa olla parempi vaihtoehto pienemmille yrityksille, jos Jiran kaikille ominaisuuksille ei ole tarvetta. Molempien ohjelmistojen käyttöliittymät ovat webpohjaisia. Kuvassa 17 on esimerkki Trellon käyttöliittymästä.



Kuva 17. Trellon käyttöliittymä (19).

4.4 Versionhallintajärjestelmä

FRENDSin eri projekteissa on ollut aikaisemmin käytössä kaikkia kolmea luvussa 2.6 läpi käytyä versionhallintajärjestelmää. Projektista riippuen käytössä on ollut joko GitHub, GitLab tai Bitbucket. Nyt uusissa projekteissa ollaan kuitenkin siirtymässä käyttämään pelkästään GitHubia. Kaikki taskit siirretään omiin avoimen lähdekoodin projekteihin, joiden lähdekoodi on vapaasti haettavissa. Tällöin FRENDSin asiakkaat voivat hakea taskeista lähdekoodin ja tehdä niistä omia versioita, jotka sopivat paremmin heidän ympäristöönsä. Jokaisen taskin dokumentaatio siirretään suoraan sitä vastaavaan GitHub-projektiin. Aikaisemmin taskien dokumentaatio oli Confluencen puolella eri projekteissa, jolloin käyttöohjeiden löytäminen saattoi olla hankalaa. Tämä mahdollistaa ohjeiden löytämisen aina samasta paikasta.

Kaikki FRENDSin taskit ovat omissa julkisissa projekteissa, jotta ne ovat mahdollisimman helppoja jakaa asiakkaille. GitHub antaa käyttäjien tehdä rajoittamattoman määrän julkisia projekteja. Nämä projektit ovat ilmaisia, ja niiden käyttäjämäärät ovat rajattomia. Jos kyseessä olisivat yrityksen sisäiset projektit, ne pitäisi todennäköisesti pitää yksityisinä. GitHubin ilmaislisenssi ei anna käyttäjien ylläpitää yhtäkään yksityistä projektia. GitHubin tapauksessa yrityslisenssien hinnat nousevat ylöspäin projektien määrän mukaan (20). FRENDSin käyttötapauksessa projektien määrä todennäköisesti nousisi korkeaksi, joten GitHub ei olisi hyvä vaihtoehto, jos projektit haluttaisiin pitää yksityisinä.

Atlassianin Bitbucket tarjoaa käyttäjille rajattoman määrän julkisia ja yksityisiä projekteja, mutta ilmaisversio tukee vain viittä käyttäjää. Lisenssin hinta vaihtelee käyttäjien määrän mukaan ja voi nousta melko suuriin summiin, jos käyttäjiä tarvitaan paljon. Bitbucketin hyötypuolena on sen helppo integroiminen Jiran kanssa. Järjestelmät ovat saman yhtiön tuotteita, ja ne on alun perin suunniteltu käytettäväksi keskenään (21). GitHub on nykyään myös mahdollista linkittää Jiran kanssa, mutta tämä toiminnallisuus on lisätty sovellukseen jälkeenpäin. Tämä tarkoittaa, että se ei aina välttämättä toimi halutulla tavalla niin kuin muut Jiraan myöhemmin lisätyistä toiminnoista. Molemmat versiot tukevat normaalia Git-käytänteitä ja ovat toiminnallisuuksiltaan lähestulkoon samanlaiset.

FRENDS-taskien muutokset hoidetaan pull requestien avulla. Taskin reposta otetaan oma haara eli branch, johon tehdään uudet muutokset. Niiden puskemisen jälkeen tehdään repolle pull request, joka ilmoittaa uudesta muutoksesta muille käyttäjille. Muutokset käydään läpi code review -tyylillä, johon on ohjeet aikaisemmin mainitussa

Confluencen dokumentaatiossa. Tässä vaiheessa tarkistetaan taskiin tehdyt muutokset ja varmistetaan, että aikaisempien versioiden korjattuja virheitä ei tule takaisin uusiin versioihin.

4.5 Version rakentaminen

FRENDSin taskiin tehtyyn muutoksen tehdään pull request, ja se käydään läpi code review -tyylillä. Taskin pull request käydään läpi manuaalisesti ja muutokset testataan jossain FRENDS-ympäristössä. Kun pull request on tarkistettu ja hyväksytty, se pitää rakentaa oikeaan muotoon, jotta sitä voidaan käyttää FRENDSin puolella. FRENDS hallitsee taskien rakennetta NuGet package managerin avulla. Se on .NET-pohjainen sovellus, joka mahdollistaa koodin paketoimisen NuGet-paketteihin. NuGet-pakettien avulla koodit pystytään siirtämään ja ottamaan käyttöön sovellukselle. FRENDS hoitaa taskien rakentamisen NuGet-paketteihin tällä hetkellä VSTS:n (Visual Studio Team Services) build-automatisaatiolla.

Visual Studio Team Services on Microsoftin pilvipohjainen sovellus, joka tarjoaa käyttäjälle joukon erilaisia toimintoja koodin kehittämiseen. VSTS vastaa Microsoftin TFS (Team Foundation Server) -sovellusta, jota on mahdollista käyttää pelkästään paikallisesti asennettuna. VSTS on TFS palvelun vastaava pilvipohjainen vaihtoehto. VSTS toimii Microsoftin Azure -palvelun avulla, joka tarjoaa käyttäjille erilaisia pilvipalveluja sovellusten ylläpitämiseen ja rakentamiseen. VSTS tarjoaa käyttäjille erilaisia toimintoja sovellusten kehittämiseen, joihin kuuluvat muun muassa sisäänrakennettu versionhallintajärjestelmä, sovellusten monitorointijärjestelmä, sovellusten automaattiset build- ja release-toiminnallisuudet sekä dokumentaatiomahdollisuudet. (22.) FRENDS käyttää tällä hetkellä VSTS:n toiminnoista vain automaattista build-toiminnallisuutta.

Continuous Integration (CI) tarkoittaa käytäntöä, jolla automatisoidaan koodin testausta ja yhdistämistä alkuperäiseen haaraan. Automatisoinnilla pystytään huomaamaan koodin muutoksista ilmeneviä virheitä ja muita ongelmia ja korjaamaan ne, ennen kuin ne menevät eteenpäin. Mitä aikaisemmin muutoksen virheet löydetään, sitä helpompi ne on korjata (23). VSTS:n build-toiminnallisuudella pystytään noudattamaan CI-käytänteitä automatisoimalla FRENDSin taskien testaus ja rakentaminen NuGet-paketteihin. Kun muutoksen pull request on hyväksytty, VSTS käy automaattisesti hakemassa koodin ja yrittää rakentaa ja testata sitä määritetyllä tavalla.

VSTS:n build-vaiheet pitää erikseen määritellä jokaiselle taskille: mistä taskin lähdekoodi löytyy versionhallintajärjestelmästä, mitä parametreja task saa rakennusvaiheessa, mitä testejä taskille ajetaan sekä mihin ja miten se rakennetaan. Uusille taskeille pitää aina määritellä build-vaiheen asetukset, muuten automaattista rakennusta ei tapahdu. Rakennuksen yhteydessä muutoksen koodi yhdistetään aikaisempaan haaraan, ja jos kaikki menee läpi ilman virheitä, rakennettu NuGet-paketti viedään eteenpäin. Build-vaiheeseen voidaan myös lisätä erilaisia Visual Studion arkkitehtuurisääntöjä, jotka käydään läpi rakennuksen yhteydessä. Jos sääntöjä ei ole noudatettu tai rakennuksesta tulee muita virheitä, automaattista buildia ei tehdä. Tällä hetkellä FRENDSissä ei ole käytössä näitä lisättyjä sääntöjä.

VSTS antaa myös mahdollisuuden Continuous Delivery (CD)-käytäntöihin. Tällöin muutosten koodit rakennettaisiin, testattaisiin ja vietäisiin automaattisesti eteenpäin joko testi- tai tuotantoympäristöihin. Tällöin kaikki hyväksytyt muutokset vietäisiin automaattisesti suoraan FRENDSin puolelle, ja ne olisivat valmiit käyttöön asiakkailla. CD käytänteitä ei kuitenkaan tällä hetkellä käytetä FRENDSin taskien versionhallinnassa, vaan valmiiksi rakennetut NuGet-paketit siirretään odottamaan eri feediin, josta ne siirretään myöhemmin manuaalisesti eteenpäin.

Jos CD-käytänteitä ruvettaisiin käyttämään taskeissa, asiakkaat saisivat tehdyt muutokset huomattavan paljon nopeammin käyttöönsä. Tämä kuitenkin poistaisi yhden vaiheen muutosten siirtämisestä versionhallintajärjestelmästä tuotantoympäristöihin, jolloin mahdolliset virheet, joita ei aikaisemmissa vaiheissa huomattu, voisivat mennä helpommin läpi. NuGet-pakettien manuaalinen siirtäminen kohdeympäristöihin hidastaa muutosten käyttöönottoa, mutta tarjoaa vielä viimeisen kerroksen virheiden huomaamiseen muutoksille. Jos build-vaiheessa taskeille asettaisiin enemmän tarkistussääntöjä, voitaisiin mahdollisesti estää suurin osa virheistä ja automatisoida pakettien vieminen kohdeympäristöihin. Tämä kuitenkin riippuu käyttötarkoituksesta: onko kehityksessä suurempi tarve minimoida mahdolliset virheet vai saada uudet muutokset mahdollisimman nopeasti ulos asiakkaille.

VSTS vastaa melkein kaikilta ominaisuuksiltaan TFS:n paikallisen asennuksen versiota. VSTS on yksinkertaisempi ja helppokäyttöisempi versio, jonka ominaisuuksissa on pieniä eroavaisuuksia käytettävyyden kannalta verrattuna TFS:ään. Suurin syy TFS:n käyttämiseen VSTS:n sijaan olisi yrityksissä, joiden pitää varmistaa datan pysyminen

yrityksen sisällä. FRENDSin tapauksessa tähän ei ole tarvetta, sillä kaikki taskit ovat muutenkin julkisissa repoissa kaikkien saatavilla.

VSTS:n huonona puolena voisi pitää sen lisenssin hinnoittelua, joka nousevat ylöspäin käyttäjämäärän mukaan. Se voidaan myös hankkia MSDN (Microsoft Developer Network) -lisenssin yhteydessä, johon sisältyvät Visual Studion lisenssit ja muita Microsoftin tarjoamia palveluita. Yksittäinen VSTS-lisenssi on ilmainen pienille tiimeille, joiden käyttäjämäärä ei ylitä viittä käyttäjää. Sen jälkeen lisenssihinta vaihtelee 30 dollarista 10 käyttäjälle aina 6 150 dollariin 1 000 käyttäjälle. VSTS tarjoaa myös liitännäisiä, jotka laajentavat sovelluksen ominaisuuksia, ja osa näistä liitännäisistä on maksullisia. (24.) Muita suosittuja sovelluksia, jotka tarjoavat Continuous Integration -toimintoja, ovat esimerkiksi Jenkins sekä Atlassianin Bamboo- ja Bitbucket Pipelines-sovellukset.

Jenkins on yksi tällä hetkellä suosituimmista koodin automatisointisovelluksista. Se on rakennettu Java-ohjelmointikielellä, joten se on alun perin suunnattu Java-projekteille. Sen suosio perustuu suurilta osin sen avoimeen lähdekoodiin, minkä takia sille on rakennettu erittäin paljon erilaisia liitäntöjä. Näiden liitäntöjen avulla sitä pystytään käyttämään myös muiden ohjelmointikielien projekteissa ja tekemään erittäin paljon eri asioita, joita siihen ei ollut alun perin suunniteltu. (25.)

Liitäntäpohjainen rakennustapa on tosin myös sen huono puoli, sillä liitäntöjen laatu ei ole aina taattu ja todennäköisesti se on huonompi verrattuna sovelluksiin, joissa vastaava toiminto on jo valmiiksi sisäänrakennettuna. Tämän vuoksi uusia liitäntöjä joudutaan testaamaan tarkemmin niiden käyttöönotossa eivätkä ne välttämättä ole edes sopivia kyseiseen projektiin. Jenkins on kuitenkin ilmainen sovellus, ja kaikki sen liitännät ovat ilmaisia. Liitäntöjä on myös helppo tehdä itse, koska sovelluksen lähdekoodi on avoin. Sovellus asennetaan paikallisesti yrityksen palvelimille, joten sen käytettävyys ei ole samaa luokkaa kuin VSTS:n pilvipohjainen asennus, jota voidaan käyttää mistä tahansa. (26; 27.)

Bamboo ja Bitbucket Pipelines ovat Atlassianin tarjoamat CI-sovellukset. Bamboo tarjosi käyttäjille aluksi sekä paikallista että pilvipohjaista versiota, jotka olivat nimeltään Bamboo ja Bamboo Cloud. Bamboo Cloudin ylläpito kuitenkin lopetettiin melko nopeasti julkaisun jälkeen. Syynä oli todennäköisesti tuotteen huono kysyntä. Samanaikaisesti Atlassian ilmoitti uudesta pilvipohjaisesta Bitbucket Pipeline -sovelluksesta, joka vastaa osittain Bamboo Cloudia (28). Molemmat sovellukset tarjoavat erittäin hyvät

integraatiomahdollisuudet muiden Atlassianin sovelluksien kanssa, kuten aikaisemmin on mainittu esimerkiksi Bitbucketin yhteydessä.

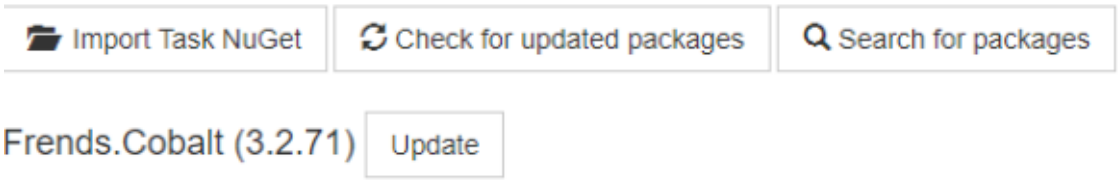
Sovellukset ovat kuitenkin ominaisuuksiltaan erilaisia, eivätkä ne sovellu samanlaisiin projekteihin. Bitbucket Pipelines käyttää Docker-sovellusta, joka toimii eri tavalla Gitiä käyttäviin sovelluksiin verrattuna. Docker on pääasiassa Linux-pohjainen sovellus, josta on erittäin rajoitettu Windows-versio. Docker lukee omista Docker-repoista Dockerfile-muotoisia tiedostoja, joiden perusteella suoritetaan taskin rakennus (27). Dockerin takia Bitbucket Pipelines ei ole suositeltava vaihtoehto Windows-pohjaisille projekteille.

Bamboon paikallinen versio vastaa toiminnoltaan VSTS:n ja Jenkinsin mallia ja tukee Windows-pohjaisia ohjelmointikieliä. Se on myös rakennettu Javalla Jenkinsin tapaan ja tukee ladattavia liitäntöjä. Bamboon liitäntöjen määrä on tosin suppea verrattuna Jenkinsin tarjoamiin liitäntöihin, ja osa niistä on maksullisia. Jenkinsin tarjoaa yli 1 000 liitäntää verrattuna Bambo:n noin 150 liitäntään (29; 30). Atlassian tarjoaa Bamboon lisenssihintaa buildien ja ajojen mukaan. Halvimmalla versiolla voi asettaa vain 10 buildia, eikä se anna mahdollisuutta suorittaa taskien rakennusta muilla kuin palvelimella, jolle Bamboo on asennettu (31). Bamboo ja Jenkins ovat toiminnoiltaan melko samanlaisia. Bamboon suurin etu Jenkinsiin verrattuna on sisäänrakennettu integraatio Jiran ja Bitbucketin kanssa. FRENDSin käyttötapauksessa VSTS on näistä vaihtoehdoista käytännöllisin, mutta Jiraa ja Bitbucketia jo valmiiksi käyttävät yritykset voivat harkita Bamboo-sovellukseen siirtymistä.

4.6 Pakettien vieminen tuotantoympäristöihin

FRENDS-taskin lähdekoodin muutos rakennetaan VSTS:n automaattisella build-vaiheella NuGet-paketiksi. Tämä NuGet-paketti pitää siirtää VSTS:n puolelta eri asiakkaiden FRENDS-ympäristöihin. Tämä hoidetaan yrityksessä tällä hetkellä kaksivaiheisesti käyttämällä Myget-feedejä. Feedeihin pystytään lataamaan NuGet-paketteja, joista paketit voidaan viedä eteenpäin kohdejärjestelmien tuotantoympäristöihin. FRENDSin käyttämille NuGet-paketeille on tällä hetkellä käytössä kaksi feediä, staging feed ja release feed. VSTS:n rakentamat paketit siirretään automaattisesti staging feediin. Nämä paketit käydään manuaalisesti siirtämässä tietyin väliajoin release feediin. Eri FRENDS-ympäristöt ovat linkitettynä release feediin ja monitoroivat sitä. Jos release feediin on viety uudempi versio paketista, joka ympäristössä on käytössä, antaa ympäristö

käyttäjälle mahdollisuuden päivittää task uudempaan versioon. Tätä on havainnollistettu kuvassa 18.



Kuva 18. FRENDS-taskin päivitys käyttöliittymässä.

Vaikka release feediin on viety taskin uudempi versio, uudempia versioita ei viedä automaattisesti asiakkaiden ympäristöihin. Tämä antaa asiakkaille mahdollisuuden pysyä vanhassa versiossa esimerkiksi vanhalla integraatiolla, jota ei haluta päivittää tai joutua muuttamaan millään tavalla. Käytetty tapa lisää vielä viimeisen kerroksen NuGet-pakettien rakennuksesta julkaisuun, jossa voidaan havaita mahdolliset virheet esimerkiksi paketin versionumeroissa. Voitaisiin myös käyttää yhden feedin ratkaisua, jossa rakennetut paketit viedään suoraan release feediin, josta asiakkaat voivat ladata niitä. Tällöin minimoitaisiin manuaalista työtä pakettien siirtämisessä, mutta menetettäisiin viimeinen mahdollisuus tarkistaa paketti virheiltä.

FRENDSin versionhallintakokonaisuudessa ei tällä hetkellä noudateta Continuous Delivery-käytänteitä. VSTS tarjoaa mahdollisuuden siirtää testatun taskin rakennettu NuGet-paketti suoraan FRENDSin kohdeympäristöihin. Asiakkaiden FRENDS-ympäristöjen taskit päivittyisivät automaattisesti, eikä heillä olisi mahdollisuutta pysyä vanhoissa versioissa. Tämä myös mahdollistaisi virheiden päätymisen asiakkaiden ympäristöihin, jos taskin lähdekoodia ei ole käyty läpi riittävän tarkasti code review-vaiheessa.

Tällä hetkellä Myget-feedejä käytetään nykyisessä versionhallintaratkaisussa, koska niitä oli jo aikaisemmassa käytössä eikä niitä haluttu viedä esimerkiksi yleisempään Nuget.org-feediin. Feedeille on kuitenkin myös muita vaihtoehtoja kuin Myget, kuten VSTS:n omat feedit, ProGet tai oman NuGet Server -feedin ylläpito, jos paketit halutaan pitää yksityisessä jakelussa yrityksen sisällä. Feedin valinnassa ei tosin näyttäisi olevan suuria eroavaisuuksia eri valintojen välillä, vaan suurin valinta NuGet-pakettien jakelussa on valinta joko feedien tai CD-käytäntöjen käyttö, jolloin ei käytetä ollenkaan feedejä.

Jos paketit halutaan siirtää mahdollisimman nopeasti asiakkaiden käyttöön, CD-käytänteiden avulla ne voidaan siirtää suoraan ympäristöihin esimerkiksi VSTS:n avulla. Jos taas halutaan tarkempaa hallintaa pakettien jakelussa, voidaan käyttää feedejä. Feedien avulla paketit voidaan siirtää suoraan valmiiksi asiakkaiden päivitettäväksi tai odottamaan vielä viimeistä tarkistusta, ennen kuin niitä tarjotaan asiakkaille.

5 Parannusehdotukset ja yhteenveto

Insinöörityön tarkoituksena oli analysoida nykyisin käytössä oleva versionhallintakokonaisuus HiQ Finlandin FRENDS-integraatiojärjestelmässä ja selvittää, miten käytänteitä ja käytössä olevia sovelluksia voitaisiin lähteä kehittämään. Tähän kokonaisuuteen kuuluivat integraatiojärjestelmän ja sen komponenttien dokumentaatiot, uusien muutoksien seurantajärjestelmät, käytetyt versionhallintajärjestelmät ja muutoksien tuotantoon vientiin käytetyt sovellukset. Työssä käytiin läpi jokaisessa vaiheessa käytetyt sovellukset ja niille mahdolliset vaihtoehdot. Tämän yhteydessä havaittiin, että suurimmalle osalle käytössä olevista sovelluksista ei ole niin sanottua parasta vaihtoehtoa, vaan sovelluksen sopivuus riippuu projektin käyttötapauksesta. Esimerkiksi versionhallintajärjestelmien tapauksessa eri vaihtoehdoilla on suurimmaksi osaksi samat toiminnallisuudet ja niiden erot tulevat julkisten ja yksityisten projektien muodossa ja käyttäjämäärissä projektia kohden. Eroja huomattiin usein myös lisenssihinnoissa, joissa osa järjestelmistä tarjosi joko ilmaisia projekteja pienelle määrälle käyttäjiä tai pelkästään julkisia projekteja ilman käyttäjärajaa.

Työssä havaittiin käytössä olevien sovelluksien olevan FRENDSin tyyppiselle projektille sopivia. Suurimmassa osassa vaihtoehdoista ei ole uusia ominaisuuksia, jotka antaisivat syyn vaihtaa niiden käyttöön. Osa käytössä olevista sovelluksista, kuten dokumentaatiosta vastaava Confluence ja muutosten seurannasta vastaava Jira, ovat myös saman yhtiön tekemiä ja tarjoavat integraatiomahdollisuuksia, joita muut vaihtoehdot eivät pysty parantamaan. Työssä käytiin kuitenkin läpi vaihtoehtoja eri vaiheiden sovelluksille, jotka voisivat sopia erityyppisille projekteille. Vaikka suoria parannusehdotuksia ei löytynyt-kään uusista sovelluksista, niitä löydettiin kuitenkin jo käytössä olevien sovelluksien toiminnoista, jotka eivät ole tällä hetkellä käytössä. Nämä kohdistuivat lähinnä uusien

muutoksien tuotantoon viennin automatisoimiseen Visual Studio Team Services (VSTS)-sovelluksen toimintojen avulla.

Sovellusten kehityksessä yleisin pullonkaula tulee versioiden julkaisemisen yhteydessä. Uusien versioiden muutokset joudutaan tarkistamaan yksitellen, jolloin julkaisut voivat pitkittyä. Tämä myös lisää mahdollisten virheiden korjausaikaa, joita ei pystytä huomaamaan manuaalisissa ja automaattisissa tarkistuksissa. Continuous Delivery-käytänteet pyrkivät lyhentämään aikaa muutoksen testaamisesta tuotantoon viemiseen. CD on ketterän kehityksen (engl. Lean manufacturing/production) käytäntö, jolla pyritään optimoimaan tuotantoon vientiä ja minimoimaan tuhlattua aikaa automatisaation avulla.

Koska CD-käytänteitä ei ole toteutettu FRENDSin kokonaisuudessa, niiden käyttöönotto on merkittävin parannusehdotus, joka tämän työn perusteella tehtiin. Se voidaan toteuttaa esimerkiksi ryhmillä: ylläpidetään kahta ryhmää, jossa toisessa on toimivaksi todettu vanha versio, joka on todettu toimivaksi, ja toisessa ryhmässä on uuden muutoksen versio. Asiakkaille annetaan mahdollisuus liittyä ryhmään, joka saa uudet muutokset suoraan testattavaksi. Jos uuden muutoksen toiminnallisuudessa havaitaan virheitä, voidaan siirtyä käyttämään aikaisempaa versiota eikä tämä virheellinen versio ole siirtynyt kaikille asiakkaille. Testausryhmässä on myös mukana kehittäjien testausympäristö, jossa testataan uutta versiota samalla, kun se on käytössä asiakkailla. Jos uusi versio toimii halutulla tavalla eivätkä testausryhmän asiakkaat ole löytäneet virheitä, se voidaan hyväksyä testausryhmän puolelta ja siirtää jakeluun kaikille asiakkaille. VSTS voi myös pitää listaa henkilöistä, jotka ovat hyväksyneet tietyn muutoksen viemisen eteenpäin kaikille asiakkaille. (32.) Tällöin asiakkaat, jotka haluavat saada uudet versiot käyttöön mahdollisimman nopeasti saavat siihen mahdollisuuden. Tämä myös mahdollistaa uuden versioiden julkaisun kaikille asiakkaille mahdollisimman nopeasti.

FRENDSin kannalta uudet muutokset saadaan testattua mahdollisimman nopeasti, mikä tarkoittaa myös mahdollisten virheiden huomaamista ja korjaamista nopeammin kuin nykyisellä tavalla. Kaikkia virheitä ei ole mahdollista huomata koodin katselmoinnissa ja valmiissa testeissä, vaan ne yleensä huomataan massakäytön yhteydessä. Enemmän testaajia tarkoittaa myös enemmän käyttötapauksia, joissa mahdolliset virheet tulevat esiin. Halukkaiden asiakkaiden lisääminen testausryhmään lisää käyttötapauksien määrää. Tämä vastaa esimerkiksi pelikehityksen alfa- ja beeta-käytänteitä, joissa pelaajat saavat mahdollisuuden testata uutta sisältöä ja raportoida löytämistään bugeista kehittäjille.

Jos CD-käytänteitä otetaan käyttöön, olisi myös hyvä lisätä FRENDSin puolelle toiminnallisuus, jolla voidaan vaihtaa laajennusosa eli task käyttämään vanhaa versiota. Jos esimerkiksi taskin uudessa versiossa ei havaita mitään virheitä, se viedään tuotannon puolelle kaikille asiakkaille. FRENDSiin viedään taskit NuGet-paketeilla ja jokaiselle uudelle versiolle on uusi NuGet paketti, joka erotellaan sen versionumeron perusteella. Uusi toiminnallisuus voisi säilyttää FRENDSin asennuspalvelimella jokaisesta taskista tietyn määrän aikaisempien versioiden paketteja. Esimerkiksi palvelimella voisi olla tallessa viiden viimeisen version paketit ja asiakkaat voivat suoraan käyttöliittymän puolelta valita aikaisemman version paketin ja siirtyä käyttämään aikaisempaa versiota. Tällöin mahdolliset virheet huomataan aikaisemmin kuin nykyisessä toteutuksessa ja asiakkaat voivat siirtyä käyttämään vanhaa toimivaa versiota. Tämä parannusehdotus olisi myös hyvä toteuttaa, vaikka CD-käytänteitä ei otettaisikaan käyttöön, sillä uusista versioista saattaa muutenkin löytyä virheitä, joita ei ole aikaisemmin huomattu. Tällä hetkellä aikaisempiin versioihin siirtyminen on vaivalloista.

On myös olemassa asiakkaita, jotka eivät enää tee uusia integraatioita FRENDSin puolella. Heillä on käytössä olemassa olevat prosessit, joiden toiminnallisuutta ei haluta muuttaa millään tavalla. Uudet taskien versiot voivat muuttaa toiminnallisuutta jollain tietyllä tavalla, jolloin itse prosesseja jouduttaisiin myös päivittämään. Näissä tapauksissa CD-käytänteillä taskien automaattiset päivitykset eivät ole suotavia. VSTS:n asetuksissa on mahdollista antaa eri käyttäjille eri toimintoja ryhmien perusteella. Nämä asiakkaat voitaisiin luoda ryhmä, joille ei päivitetä uusia taskeja. Tämän lisäksi voidaan myös pitää yllä vanhan toteutuksen kaltaista Myget-feediä, josta paketit voidaan hakea manuaalisesti tarvittaessa.

VSTS tukee myös muita CD-tapoja, joilla uudet versiot päivitetään asteittain asiakkaille. Näihin kuuluu muun muassa kahden version ylläpitäminen samanaikaisesti. Ensimmäisellä versiolla on vanha toimiva versio taskista. Toisessa versiossa on uusi muutos, jota halutaan viedä tuotantoon. Asiakkaita ruvetaan vähitellen ohjaamaan uudempaan versioon kuormituksen tasapainottamisen (engl. Load Balancing) avulla. Tätä prosessia monitoroidaan, ja jos uudessa versiossa huomataan virhe, ohjataan kaikki asiakkaat käyttämään ensimmäistä versiota.

Joissain käyttötapauksissa voi myös olla parasta viedä muutokset hitaammalla vauhdilla tuotantoon ja tarkistaa versiot huolellisesti, jotta vanhoja virheitä ei pääse vahingossa siirtymään takaisin tuotannon puolelle. Näissä tapauksissa voidaan myös käyttää

ryhmiä, joille uusia muutoksia ei päivitetä samalla nopeudella. Käyttötapauksista huolimatta olisi hyvä lisätä FRENDSin puolelle taskien version vaihtamisen toiminnallisuus. Tällä hetkellä tätä mahdollisuutta ei ole käyttöliittymän puolella, vaan se joudutaan poistamaan kokonaan ja lataamaan vanha versio uudelleen tyhjästä. Vaikka kaikilla asiakkuuksilla ei tule olemaan tarvetta tälle toiminnallisuudelle, se olisi hyvä olla mahdollista varmuuden vuoksi.

Insinöörityössä käytiin läpi yrityksen versionhallintakokonaisuus, ja siihen esitettyjä parannusehdotuksia voidaan lähteä toteuttamaan yrityksen kanssa jälkeenpäin, jos niille katsotaan olevan tarvetta.

Lähteet

1 Somasundaram, Ravishankar. 2013. Git: Version Control for Everyone. E-kirja. Packt Publishing.

2 Chacon, Scott & Straub, Ben. 2014. Pro Git. E-Kirja. Apress.

3 Sink, Erik. 2011. Version Control by Example. E-Kirja. Erik Sink.

4 Compare Repositories. 2018. Verkkoaineisto. BlackDuck Software, Inc. <https://www.openhub.net/repositories/compare>. Luettu 20.3.2018

5 Hong, Neil Chue. 2018. Choosing a repository for your software project. Verkkoaineisto. Software Sustainability Institute. <https://software.ac.uk/resources/guides/choosing-repository-your-software-project>. Luettu 5.2.2018.

6 GitHub vs. Bitbucket vs. GitLab vs. Coding. 2016. Verkkoaineisto. Medium. <https://medium.com/flow-ci/github-vs-bitbucket-vs-gitlab-vs-coding-7cf2b43888a1>. Luettu 7.2.2018.

7 Kharchenko, Nataliia. 2017. GitHub vs. GitLab vs. Bitbucket: Which code repo should you choose? Verkkoaineisto.<https://jaxenter.com/github-gitlab-bitbucket-code-repo-138308.html>. Luettu 7.2.2018

8 Google trends. 2018. Verkkoaineisto. Google. <https://trends.google.com/trends/explore?q=github,gitlab,bitbucket>. Luettu 7.2.2018.

9 GitHub. 2018. Verkkoaineisto. GitHub. <https://github.com/>. Luettu 8.2.2018.

10 Peham, Thomas. GitLab vs GitHub: Key differences & similarities. Verkkoaineisto. <https://usersnap.com/blog/gitlab-github/>. Luettu 15.2.2018.

11 Bitbucket. 2018. Verkkoaineisto. Atlassian. < https://bitbucket.org/product>. Luettu 8.2.2018.

12 Manouvrier, Bernard & Menard, Laurent. 2009. Application Integration: EAI B2B BPM and SOA. E-kirja. John Wiley & Song.

13 What is BPM? Verkkoaineisto. Workflow Management Coalition. <https://www.wfmc.org/what-is-bpm>. Luettu 15.2.2018.

14 Business Process Model and Notation. Verkkoaineisto. Object Management Group. <http://www.bpmn.org/>. Luettu 15.2.2018.

15 Shipment Process of a Hardware Retailer. Verkkoaineisto. Business Process Incubator. <https://www.businessprocessincubator.com/content/shipment-process-of-a-hardware-retailer/>. Luettu 15.2.2018.

16 HiQ Finland. Verkkoaineisto. HiQ Finland. <https://hiqfinland.fi/>. Luettu 15.2.2018.

17 Jira. Verkkoaineisto. Atlassian. <https://www.atlassian.com/software/jira>. Luettu 20.2.2018.

18 Jira esimerkkiprojekti. Verkkoaineisto. Atlassian. <https://jira.atlassian.com/browse/JSWSERVER-15268?attachmentViewMode=list>. Luettu 20.2.2018

19 Trello Welcome Board. Verkkoaineisto. Trello. <https://trello.com/b/bKbdmCKB/welcome-board>. Luettu 20.2.2018

20 5 reasons NOT to choose Atlassian JIRA in custom software development. Verkkoaineisto. Sensinum. <https://sensinum.com/jira-project-management/>. Luettu 20.2.2018.

21 Github vs Bitbucket. 2017. Verkkoaineisto. UpGuard. <https://www.upguard.com/articles/github-vs-bitbucket>. Luettu 20.2.2018.

22 JIRA and Bitbucket integration. 2018. Verkkoaineisto. Atlassian. <https://www.atlassian.com/software/jira/bitbucket-integration>. Luettu 25.2.2018.

23 Visual Studio Team Services features. 2018. Verkkoaineisto. Microsoft. <https://www.visualstudio.com/team-services/features/>. Luettu 28.2.2018.

24 Build and Release in VSTS and TFS. 2016. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/vsts/build-release/overview>. Luettu 28.2.2018.

25 Visual Studio Team Services pricing. 2018. Verkkoaineisto. Microsoft. <https://www.visualstudio.com/team-services/pricing/>. Luettu 28.2.2018.

26 Jenkins. Verkkoaineisto. <https://jenkins.io/>. Luettu 16.2.2018.

27 Johnson, Patricia. 2017. Is One Continuous Integration Tool Better than All the Rest? Verkkoaineisto. WhiteSource. <https://www.whitesourcesoftware.com/whitesource-blog/top-continuous-integration-tools/>. Luettu 29.2.2018.

28 Bamboo vs Bitbucket Pipelines. 2016. Verkkoaineisto. Jack Graves. <http://www.jackgraves.co.uk/blog/post/bamboo-vs-bitbucket-pipelines>. Luettu 29.2.2018.

29 Bamboo apps. 2018. Verkkoaineisto. Atlassian. <https://marketplace.atlassian.com/addons/app/bamboo/newest>. Luettu 15.2.2018.

30 Jenkins plugins. 2018. Verkkoaineisto. Jenkins. <https://plugins.jenkins.io/>. Luettu 15.2.2018.

31 Bamboo Licensing and Pricing. 2018. Verkkoaineisto. Atlassian. >https://www.atlassian.com/licensing/bamboo>. Luettu 29.2.2018.

32 What is Continuous Delivery? 2018. Verkkoaineisto. Microsoft. <https://www.visualstudio.com/learn/what-is-continuous-delivery/>. Luettu 25.3.2018