TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUKKA-PEKKA VENTTOLA

VERSIONHALLINTA OPETUSKÄYTÖSSÄ

Kandidaatintyö

Tarkastaja: Yliopistonlehtori Essi Isohanni
Jätetty tarkastettavaksi: 14.3.2018

TIIVISTELMÄ

JUKKA-PEKKA VENTTOLA: Versionhallinta opetuskäytössä
Tampereen teknillinen yliopisto
Kandidaatintyö, 27 sivua
Maaliskuu 2018
Tieto- ja sähkötekniikan TkK-tutkinto-ohjelma, Tietotekniikka
Pääaine: Ohjelmistotekniikka
Tarkastajat: Yliopistonlehtori Essi Isohanni
Avainsanat: versionhallinta, opetus, Git, GitLab

Tässä työssä luodaan katsaus Tampereen teknillisellä yliopistolla käytettyyn versionhallintajäjestelmään ja siihen, miten kyseistä järjestelmää käytetään opetuksen työkaluna. Lisäksi läpikäydään muita mahdollisia ratkaisuja käyttää versionhallintaa osana opetusta. Taustatietoina käsitellään versionhallintaan liittyvää terminologiaa sekä käytössä olevia järjestelmiä yleisesti.

Tärkeimpänä tavoitteina on selvittää, kuinka helppokäyttöisiä, tarkoituksenmukaisia ja ylläpidettäviä käytössä olevat ratkaisut ovat. Työ keskittyy erityisesti tutkimaan ratkaisuja yhteistyöskentelyn mahdollistamisessa, valmiin ohjelmakoodipohjan jakelussa sekä harjoitustöiden palauttamisessa.

Työssä on perehdytty työnkulkuun Tampereen teknillisen yliopiston tietotekniikan laboratoriolla sekä pohdittu muita mahdollisia vaihtoehtoja. Suorittamalla kirjallisuuskatsaus käytten aineistona aiheeseen liittyvää kirjallisuutta sekä käytössä olevien järjestelmien dokumentaatiosivuja.

Työssä todetaan, että Tampereen teknillisellä yliopistolla käytössä olevat järjestelmät pystyvät vastaamaan työssä tutkittuihin tarpeisiin. Ongelmia kuitenkin nousee siitä, että monien erilaisten järjestelmien ylläpitäminen aiheuttaa huomattavan paljon työkuormaa ja ylläpidon jatkuvuuden takaaminen on haastavaa.

SISÄLLYS

KUVAT

LYHENTEET JA MERKINNÄT

AD	Active Directory, Microsoftin kehittämä Windows-ympäristön käyttäjähallintajärjestelmä
API	Application programming interface, ohjelmointirajapinta
CSV	Comma Separated Values, tekstipohjainen taulukkoformaatti, jossa erotinmerkkinä käytetään pilkkua tai puolipistettä
Django	Python3-pohjainen web-ohjelmistokehityskehys
framework	Ohjelmistokehys, jonka avulla on helppo luoda ohjelmistoja tiettyyn tarjoitukseen
HTTP	HyperText Transfer Protocol. Selaimien ja palvelinten välillä usein käytetty tiedonsiirtoprotokolla
IDE	Integrated Development Environment. Ohjelmointiympäristö, työkalu joka on tarkoitettu yhdelle tai muutamalle ohjelmointikielelle
LDAP	Lightweight Directory Access Protocol, avoin, toimittajariippumaton pääsynhallinnan protokolla
REST	Representational state transfer, tilattomiin operaatioihin perustuva tiedonmanipulointimalli
SAAS	Software as a service, ohjelmistoliiketoimintamalli, jossa ohjelmisto tarjotaan palveluna
SHA-1	Secure Hashing Algorithm. Kryptografinen funktio, joka tuottaa 40 merkkiä pitkän heksadesimaalimuotoisen tiivisteen
Shibboleth	Avoimen lähdekoodin kertakirjautumis- ja autentikointijärjestelmä
skripti	Lyhyt, tulkattavalla kielellä toteutettu tietokoneohjelma
TTY	Tampereen teknillinen yliopisto
työkopio	Käyttäjällä olevasta, työstettävästä kopiosta käytetty nimitys

1. JOHDANTO

Nykyaikaisessa ohjelmistokehityksessä sujuva yhteistyöskentely on onnistumisen edellytys. Modernit versionhallintajärjestelmät tarjoavat muutoshistorian seuraamisen lisäksi työkaluja useamman ihmisen työpanoksen yhdistämiseksi. Versionhallintajärjestelmien käytön tueksi on olemassa erilaisia hallintajärjestelmiä, jotka tarjoavat lisää työkaluja projektin ja prosessinhallintaan. Tällaisten työkalujen käyttötaito on arvostettua ohjelmisteollisuuden puolella. Teknillisten oppilaitosten tehtävänä on kouluttaa työelämään hyvin valmistautuneita ohjelmistoalan ammattilaisia, joten pelkkä kyky tuottaa ohjelmakoodia ei riitä. Täten opetuksessa tulee käyttää työelämän kannalta merkityksellisiä työkaluja ja -tapoja. Lisäksi nämä teollisuuden käyttöön tarkoitetut järjestelmät voivat myös auttaa ratkaisemaan joitakin opetuksen järjestämiseen liittyviä ongelmia.

Ohjelmointikursseilla luentojen lisäksi viikkoittaisia harjoituksia, joissa kurssin kannalta tarpeellisia työkaluja ja -menetelmiä opetellaan käyttämään. Tämän lisäksi useilla kurssilla toteutetaan harjoitustyö. Molemmissa tapauksissa saattaa olla tarpeen tarjota joko malliksi tarkoitettu ohjelmakoodia tai valmiita pohjia, joita opiskelijat voivat lähteä työstämään. Lisäksi ryhmissä tehtävien töiden tapauksessa on tarpeen tarjota alusta, joka tulee tiimityöskentelyä. Mikäli näihin tarkoituksiin käytetään teollisuudessa käytössä olevia järjestelmiä, opiskelijat oppivat samalla työelämän kannalta arvokkaita taitoja.

Tämän työn tavoitteina on käydä läpi Tampereen teknillisen yliopiston Tietotekniikan laboratoriossa käytetyt versionhallintajärjestelmät, kuvailla niiden käyttötavat ja nostaa esille mahdollisia ongelmakohtia. Lisäksi tässä työssä kuvaillaan versionhallintajärjestelmien yleisperiaatteita sekä nostetaan esiin niitä erityistarpeita, joita suurten kurssien järjestäminen asettaa teollisuuden tarpeisiin suunnitelluille järjestelmille. Tämän työn tutkimuskysymyksinä on selvittää järjestelmien tarjoamia mahdollisuuksia yhteistyöskentelyn mahdollistamiseen, valmiiden ohjelmakoodipohjien jakeluun sekä harjoitustöiden palauttamiseen. Työssä arvioidaan käytössä olevien ratkaisujen tarkoituksenmukaisuutta, helppokäyttöisyyttä, ja ylläpidettävyyttä.

Tämän työn luvussa 2 käsitellään versionhallintajärjestelmiä yleisesti ja luvussa 3 tarkemmin Git-versionhallintajärjestelmää. Luvussa 4 kuvaillaan Gitin kanssa yhdessä käytettäväksi tarkoitettua GitLab-ohjelmistoa ja sen tarjoamia mahdollisuuksia. Luvussa 5 kuvaillaan TTY:n Tietotekniikan laboratiossa kehitettyä Repolainen-opetusjärjestelmää ja luvussa 6 kuvaillaan kurssihenkilökunnan työnkulku näiden järjestelmien käytössä. Luku 7 kuvailee muita mahdollisia vaihtoehtoja opetuksen järjestämiseen versionhallintajärjestelmiä hyödyntäen. Luvussa 8 arvioidaan käytössä olevien ratkaisuja. Luku 9 käy tulokset läpi ja tarjoaa ehdotuksia toimintamallien parantamiseksi.

2. VERSIONHALLINTA YLEISESTI

Versionhallinnalla tarkoitetaan järjestelmää, jota voidaan käyttää erilaisten tiedostopohjaisten projektien hallintaan. Erityisen paljon versionhallintajärjestelmiä käytetään ohjelmistoprojektien yhteydessä. Datan kahdentamisen lisäksi versionhallintajärjestelmät pitävät kirjaa muutoksista ja versiohistoriasta. Järjestelmään kuuluu käytännössä aina malli tietovarastosta, josta käyttäjät pystyvät hakemaan itselleen työkopioksi kutsutun version.

Versionhallintaa ei tule sekoittaa varmuuskopiointiin. Versionhallinnan avulla voidaan helposti palauttaa vahingossa poistettuja tiedostoja, sekä tutkia tiedostojen muutumista ajan saatossa, mutta versionhallintajärjestelmiä ei ole tarkoitettu korvaamaan varmuuskopioiden ottamista.

Versionhallintajärjestelmät voi jakaa karkeasti kolmeen pääryhmään, paikallisiin, keskitettyihin ja hajautettuihin järjestelmiin. Nämä pääryhmät on esitelty alla.

2.1 Paikallinen versionhallinta

Paikalliset versionhallintajärjestelmät tarjoavat yhdelle käyttäjälle mahdollisuuden tiedostojen historian hallintaan. Perustoiminnallisuuksiin kuuluu historiakirjanpito, mahdollisuus palauttaa aikaisempia versioita sekä eri versioiden yhdistäminen. Paikalliset versionhallintajärjestelmät eivät ohjelmistokehityksessä sovellu usean ihmisen projekteihin, sillä suoraa mahdollisuutta versioiden jakelemiseen ei ole toteutettu. Typillinen esimerkki paikallisesta versionhallintajärjestelmästä on vuonna 1982 julkaistu Revision Control System.

Periaatteessa myös tiedostojen eri versioiden tallentamista eri versionimillä voidaan pitää paikallisena versionhallintana, vaikka käytössä ei olisi mitään varsinaista järjestelmää versiointia varten. Tällainen järjestely on kuitenkin monimutkaisemmissa projekteissa liki mahdotonta.

2.2 Keskitetty versionhallinta

Keskitetyllä versionhallinnalla tarkoitetaan järjestelmää, joka on toteuttu siten, että tiedostot kerätään yhteen tietovarastoon. Keskitetyissä järjestelmissä käyttäjien työkopiot ovat suoraan yhteydessä keskustietovarastoon[2, p. 9-11]. Käyttäjät voivat ottaa tarpeen mukaan matalia kopioita, eli sen hetkisen version sekä viedä omia muutoksiaan keskustietovarastoon.

Keskustietovaraston voi jakaa useammalle käyttäjälle, jolloin yhteistyöskentely saman projektin parissa on mahdollista. Esimerkkinä keskitetystä versionhallintajärjestelmästä voidaan pitää 2000-luvun alkuun asti suosiossa ollutta Subversionia.

2.3 Hajautettu versionhallinta

Hajautettu versionhallinta on nykyään eniten ohjelmistoprojekteissa käytetty versionhallintatyyppi. Hajautetussa järjestelmässä jokainen työkopio on täydellinen versio projektista sisältäen täydelliset historiatiedot. Tämä johtaa myös siihen, että hajautetussa järjestelmässä projektin päähaara on olemassa vain käsitteellisellä tasolla, verrattuna keskistetyn ratkaisun tietovarastomalliin.

Yksittäisen tietovaraston korvaaminen usealla täydellisellä kopiolla lisää järjestelmän vikasietoisuutta, sillä mikäli tietovaraston sisältävä palvelinkone vikaantuu, voidaan tietosäiliö palauttaa mistä tahansa työkopiosta. Tässä työssä pääasiallisesti käsitelty Git edustaa tätä suunnittelumetodologiaa. Muita mainitsemisen arvoisia hajautettuja versionhallintajärjestelmiä ovat Bazaar ja Mercurial.

3. GIT

Vuonna 2005 julkaistu Git-versionhallintajärjestelmä on kehitetty alun perin Linux-käyttöjärjestelmän ytimen lähdekoodin ylläpitämistä varten korvaamaan Linux-projektissa aiemmin käytetty BitKeeper [1, p. 31]. Gitin suunnittelutavoitteet ovat nopeus, yksinkertainen rakenne, vahva tuki epälineaariselle ohjelmistokehitykselle, täysi hajautettavuus sekä kyky hallita suuriakin projekteja, josta esimerkkinä aiemmin mainittu Linuxin ydin.

Git on tyypiltään hajautettu versionhallintajärjestelmä, joten se ei vaadi erillistä keskustietovarastoa. Tästä huolimatta myös Gitin kanssa käytetään usein jonkinlaista keskustietovarastoratkaisua yhteistyöskentelyn helpottamiseksi.

3.1 Gitin suunnitteluperiaatteet

Git pitää kirjaa tiedostomuutoksista rivitasolla. Tämä johtaa käytännössä siihen, että Git pystyy tallentamaan ja palauttamaan tekstimuotoisten tiedostojen versioita nopeasti ja tehokkaasti. Ongelmia syntyy, jos Gitin versiohistoriaan säilöö binäärimuotoisia tiedostoja, sillä mikä tahansa muutos saa aikaan sen, että Gitin näkökulmasta koko tiedosto on muuttunut täysin. Tällöin tiedoston historiaan liittyvän metadatan määrä on huomattavan suuri. Gitin on suunniteltu alun perin ohjelmistoprojektien versionhallinnan tarpeisiin, tämä suunnitteluratkaisu hyväksyttävä vaihtokauppa.

Suurin osa usein käytetyistä Git-operaatioista on paikallisia, eli niiden suorittaminen ei vaadi verkkoyhteyttä. Työskentely on siis nopeampaa verrattuna keskitettyihin versionhallintajärjestelmiin, koska dataa ei tarvitse aina siirtää verkon ylitse. Lisäksi verkkokatkokset ja mahdollinen ruuhka verkoissa eivät pääse vaikuttamaan työskentelyyn. Tämä myös rohkaisee kehittäjiä tallentamaan muutoksensa usein paikallisesti, kun tallentaminen tapahtuu nopeasti ja onnistuu verkon ruuhkaisuudesta riippumatta.

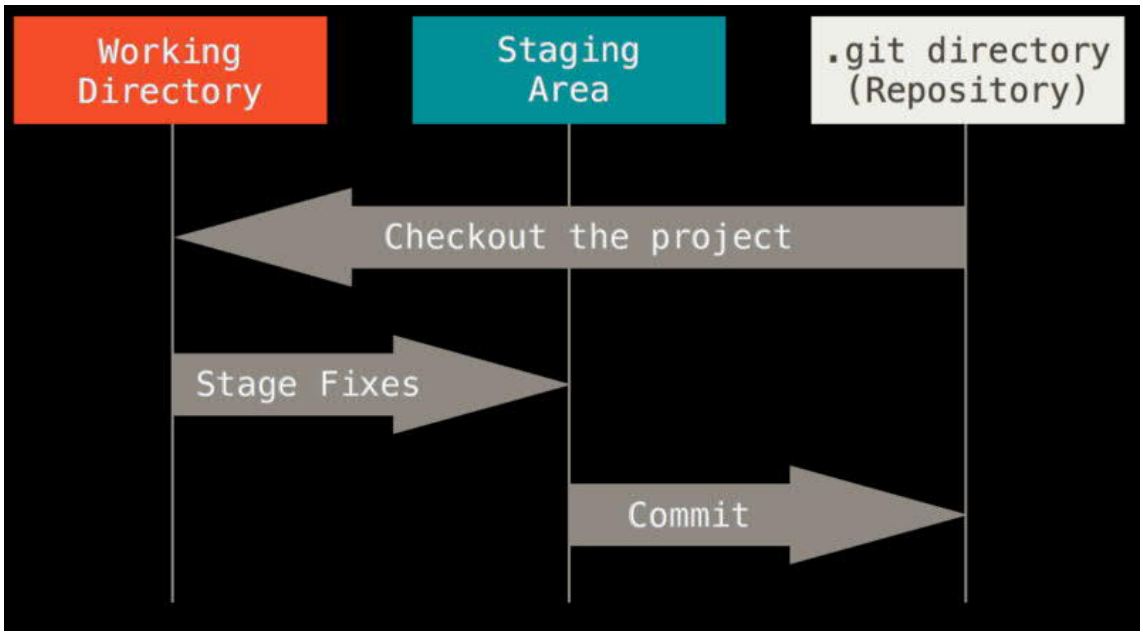
Git-tietovaraston (engl. repository) käsite kattaa versionhallinnan seuraamat (engl. tracked) tiedostot sekä versiohistorian metadatan, jonka avulla voidaan palata tar-

kastelemaan aikaisempia versioita tiedostoista, tiedon paikallisista haaroista, URL-osoitteet mahdollisiin etätietovarastoihin sekä tiedon näistä haetuista haaroista. Käytännössä tietovarasto on hakemisto, jossa versionhallinnan seuraamat tiedostot ja Gitin oma historiatietokanta sijaitsevat.

3.2 Gitin avulla työskentely

Tapoja käyttää Gitiä on useita, joko komentorivikäyttöliittymän, erillisen graafisen käyttöliittymän tai osana IDE:ä liitännäisen avulla. Jokaisessa tavassa on puolensa, mutta käyttöä komentoriviltä suositellaan, sillä se on usein ainoa tapa pystyä käyttämään kaikkia olemassa olevia komentoa. Toinen argumentti komentorivikäytön puolesta on se, että kaikki komennot ovat samoja kaikissa käyttöjärjestelmissä. Tämä saadaan aikaan sillä, että minkä tahansa Windowsille asennettavan ohjelmiston mukana asentuu myös Git Bash, joka on käytännössä Bash-emulaattori Windows-ympäristöön.

Graafisia Git-asiakasohjelmia on saatavilla runsaasti erilaisia. Suurimmat erot näiden ohjelmien välillä liittyvät siihen, että minkä ominaisuuksien helppokäyttöisyyttä ne painottavat, ja miten ne visualisoivat eri asioita. Lisäksi yleisimpiin IDEihin on saatavilla Git-liitännäisiä, joiden avulla versionhallinnan käyttö onnistuu saman työkalun avulla kuin varsinainen ohjelmistokehitystyö.



Kuva 3.1 *Kuvassa näkyy Gitin kolme paikallista tilaa sekä operaatiot niiden välillä. [1, p. 35]*

Gitissä on työskentelymallissa on kolme erillistä paikallista tilaa: muutettu, valmisteltu ja pysyvästi muutettu (engl. modified, staged ja committed)[1, p. 34]. Nämä tilat vat näkyvillä kuvassa 3.1. Työskentelyn alkuvaiheessa kaikki tiedosto ovat muuttamattomassa tilassa. Kun paikallisia tiedostoja muokataan esimerkiksi tekstiteditorilla, ne näkyvät muutetussa tilassa. Tämän jälkeen muutokset voi valmistella pysyvästi tallennettavaksi. Valmistelun jälkeen tiedostoihin tehdyt muutokset eivät ole mukana tehtävässä pysyvässä tallennuksessa. Kuten aiemmin todettu, Git pitää kirjaa tiedostojen muutoksista rivitasolla ja siten myös tarjoaa mahdollisuuden valita tiedostosta vain tietyt muuttuneet rivit sen sijaan, että valmistelisi kaikki tiedostoon tehdyt muutokset. Valmistellut muutokset voi edelleen merkata pysyvästi muutetuiksi. Kun muutokset on merkattu pysyvästi muutetuiksi, tietovaraston sen hetkisestä tiedosto- ja hakemistorakenteesta lasketaan SHA-1-tiivistesumma, jonka avulla voidaan varmistua tiedostojen muuttumattomuudesta ja eheydestä. Lisäksi tätä tiivistearvoa voidaan käyttää aikaisempiin versioihin viittaamiseen.

Aina kun tiedostoja muutetaan pysyvästi, muutosta tehdessä annetaan lyhyt, kuvaava viesti (engl. commit message). Viestissä tulee kuvailla tehtyjen muutosten syy ja tarkoitus. Lisäksi viestit toimivat lisädokumentaationa projektista ja niiden avulla voi helpottaa mahdollisesti syntyvien ongelmien selvittämistä. Pysyvän tiedostojen muuttamise yhteydessä talletetaan lisäksi tieto siitä, kuka muutoksen on tehnyt ja milloin muutos on tehty.

Vaikka tiettyyn versioon voi aina viitata SHA-1-tiivistesumman avulla, voi tämä pidemmän päälle olla työlästä. Tämän helpottamiseen Gitissä on mahdollita merkitä (engl. tag) tiettyjä versioita. Merkitsemällä voidaan tietylle versiolle antaa nimi, esimerkiksi versionumero. Merkitseminen ei kuitenkaan rajoitu versionumeroihin, vaan merkintä voi olla mikä tahansa merkkijono. Merkintöjen avulla eri versioiden välillä siirtyminen helpottuu, kun aina ei tarvitse selvittää tiivistesummaa erikseen.

3.3 Haarat

Gitin tärkeimpänä ominaisuutena voidaan pitää haaroja. Yleisesti ottaen haaroilla tarkoitetaan sitä, että projektin päälinjasta voidaan erkaantua, siten, että päälinja säilyy edelleen ehjänä. Täten haarojen avulla on mahdollista ylläpitää yhdestä projektista useanpaa samanaikaista versiota eri tarkoituksia varten. Samassa projektissa voi olla eri haaroissa esimerkiksi vakaa julkaisuversio, yleinen kehitysversio sekä useita erillisiä haaroja uusien ominaisuuksien kehittämistä varten. Gitissä oletushaara on nimeltään master. Uusille haaroille täytyy luomisen yhteydessä antaa nimi, jonka avulla kyseiseen haaraan voidaan myöhemmin viitata.

Gitissä haarat ovat enemmänkin peruslähtökohta, kuin päällerakennettu ominaisuus. Tämä johtaa siihen, että eri haarojen välillä siirtyminen on erittäin nopeaa verrattuna useisiin muihin versionhallintajärjestelmiin. Lisäksi haarojen toteutustavasta johtuen pitkänkin aikaa sitten erkaantuneiden haarojen yhdistäminen on nopeaa ja tehokasta.

3.4 Etätietovarastot

Tärkeimmät Gitin etäoperaatiot ovat kloonaus (engl. clone), työntäminen (engl. push) ja nyhtäminen (engl. pull). Kloonaamisella tarkoitetaan koko etätietovaston hakemista käyttäjän koneelle. Työntämällä paikalliset, pysyvät muutokset viedään etätietovastoon ja nyhtämällä on mahdollista hakea muiden käyttäjien työntämät muutokset mukaan paikalliseen tietovarastoon.

Vaikka Git ei hajautetun luonteensa vuoksi varsinaisesti tarvitse etätietovarastoa (engl. remote), yleensä yhteistyöskentelyn mahdollistamiseksi sellainen halutaan käyttöön. Yksinkertaisimmillaan etätietovarsto on vain Git-tietovarasto, joka on jollakin sellaisella palvelimella, johon kaikilla kehittäjillä on pääsy. Git ei tee eroa eri tietovarastojen välillä; mikään ei ole arvokkaampi kuin toinen.

Git ei itsessään rajoita lainkaan, kenellä on oikeuksia tehdä muutoksia versiohistoriaan. Tämä saattaa varsinkin kokemattomien käyttäjien tapauksessa johtaa tietojen häviämiseen, jos Git-komentoja käytetään ymmärtämättä komennon vaikutuksia. Tämän vuoksi halutaan käyttöön järjestelmä, joka tarjoaa lisää hallintamahdollisuuksia ja käyttöoikeuksien rajoittamista. Muutamia tällaisista järjestelmistä ja palveluista käsitellään myöhemmin tässä työssä.

4. GITLAB

Kuten aiemmassa luvussa 3 todettu, Gitin kanssa käytetään usein jonkinlaista keskustietovarastoa yhteistyöskentelyn helpottamiseksi. Tämä voisi käytännössä olla millä tahansa koneella sijaitseva tietovarasto, mutta yleensä käytetään ratkaisua, joka tarjoaa tietovaraston lisäksi muita työkaluja prosessin- ja projektinhallintaan. Tähän tarpeeseen vastaa GitLab, joka tarjoaa mahdollisuuden Git-keskustietovarastojen luomiseen, pääsyn- ja näkyvyydenhallintaan sekä runsaasti erilaisia työkaluja ohjelmistoprojektien hallinnointiin.

GitLab on avoimen lähdekoodin ohjelmisto, josta on saatavilla kaksi eri versiota: rajoitetummilla ominaisuuksilla varustettu, ilmainen Community Edition (CE) sekä laajemmilla ominaisuuksilla varustettu Enterprise Edition (EE). EE:stä josta on saatavilla kolme erilaista lisenssiä, joiden ominaisuudet sekä GitLab Inc.n tarjoamat tukitoiminnot vaihtelevat eri lisenssien välillä. GitLabin voi asentaa omalle palvelimella paikallisesti ylläpidettäväksi melko helposti, kunhan vain laitteistovaatimukset[6] täyttyvät. Tässä luvussa käsitellään paikallisesti asennettua GitLab-instanssia, GitLab Inc.n tarjoamaa GitLab.com-palvelua käsitellään myöhemmin.

4.1 GitLabin tärkeimmät ominaisuudet

GitLab tarjoaa Git-tietovaraston lisäksi projektin käsitteen. Projektin käsitteeseen kuuluu Git-tietovarasto, ongelmienseurantatyökalu (engl. issue tracker), haarojen yhdistämispyynnöt (engl. merge request), wikisivut, versionhallinnan ulkopuolella olevat koodipätkät (engl. code snippets). Lisäksi projektissa voi olla jatkuvaan integraatioon ja jatkuvaan tuotantoonviemiseen (continous integration, continous deployment) liittyviä asioita. Projektiin voi myös lisätä jäseniä erilaisilla pääsyoikeustasoilla. Lisäksi projektien näkyvyysasetuksia on mahdollista määrittää kolmelle eri tasolle, yksityinen, sisäinen ja julkinen. Yksityiset projetit ovat vain ja ainoastaan niiden henkilöiden nähtävillä, joille on eksplisiittisesti annettu pääsyoikeus. Sisäiset projektit ovat näkyvissä ainoastaan sisäänkirjautuneille käyttäjille ja julkisia projeteja voi tarkastella kuka tahansa, myös kirjautumattomat käyttäjät.

Projektien lisäksi GitLabissa on ryhmän käsite. Ryhmää voidaan käyttää ryhmitte-

lemään ja eristämään organisaation eri osioita toisistaan. Käyttäjiä voidaan lisätä ryhmiin samoilla pääsyoikeustasoilla kuin projekteihin ja nämä pääsyoikeudet välittyvät ryhmän alle luoduille projekteille. Ryhmän jäsenten pääsyoikeudet siirtyvät kaikkiin ryhmän alaisuudessa oleviin aliryhmiin ja projekteihin. Ryhmiin on lisäksi mahdollista luoda aliryhmiä, jotka tarjoavat hienojakoisemman pääsyoikeuksien säätämismahdollisuuden. Pääryhmän jäsenet näkevät kaiken, mitä aliryhmissä on, mutta aliryhmien jäsenet vain oman aliryhmänsä alla olevat aliryhmät ja projektit. GitLab-ryhmän aliryhmiä voi olla korkeintaan 20 tasoa (21 mukaanlukien alkuperäinen pääryhmä) [4].

Ryhmien alla olevien projektien lisäksi käyttäjien on mahdollista luoda henkilökohtaisia projekteja GitLabiin. Näihin projekteihin on pääsy oletuksena vain kyseisen projektin luojalla, mutta projektiin on mahdollista lisätä jäseniä vapaasti.

Pääsynhallintaan GitLab tarjoaa hyvin laajat mahdollisuudet. GitLabiin voi joko rekisteröidä paikallisia käyttäjiä tai GitLabin voi yhdistää olemassa olevaan käyttäjänhallintajärjestelmään, esimerkiksi AD/LDAP-järjestelmään. Tällöin ensikirjautumisen yhteydessä luodaan paikallinen käyttäjä GitLabiin kirjautumisjärjestelmän tarjoamilla tiedoilla. Käyttäjätunnuksia GitLabissa on kahdenlaisia, peruskäyttäjiä sekä pääkäyttäjiä(admin). Pääkäyttäjien vastuulla on GitLabin hallinnointi selainkäyttöliittymän kautta, johon kuuluu erityisesti ryhmien luonti, käyttäjien asettaminen ryhmien omistajiksi. Tämän jälkeen vastuu ryhmän hallinnasta on ryhmän omistajille. Omistajat voivat vapaasti hallita kaikkia ryhmänsä sisäisiä toimintoja, lisätä henkilöitä ryhmään, luoda ja hallinnoita aliryhmiä sekä projekteja.

4.2 GitLabin hallinnointi

Selainkäyttöliittymän lisäksi GitLab tarjoaa hyvin laajan REST API-rajapinnan [5], jonka avulla on mahdollista käskyttää GitLabia HTTP-kyselyjen avulla. API:n käyttö on erityisen hyödyllistä kun halutaan automatisoida operaatioita. API-kutsujen avulla voidaan hakea tietojaa GitLabista, joiden avulla sitten suoritetaan käskyjä, jotka muokkaavaat GitLabin tilaa. Kun näitä käskyjä ajetaan skripteissä, niin sellaiset toimenpiteet, jotka olisivat monimutkaisia ja aikaavieviä voidaan hoitaa hyvin nopeasti ja tehokkaasti. GitLabin suosion myötä saatavilla on kirjastoja monille ohjelmontikielille helpottamaan omien hallinnointiskriptien ja -sovellusten kehittämistä.

GitLabin hallinnointi REST API:n kautta vaatii käyttäjäkohtaisen pääsytunnuksen (engl. access token) luomisen. Tämän tunnuksen avulla tunnistetaan kyselyn tehnyt

käyttäjä ilman, että hänen tarvitsee syöttää kirjautumistietoja kyselyn yhteydessä. Pääsytunnuksen pystyy luomaan GitLabin selainkäyttöliittymässä. Tunnus tulee ottaa itselle talteen, sillä sitä ei ole selkokielisenä enää nähtävillä luomisen jälkeen.

Paikallisena asennuksena GitLab vaatii ylläpitoresursseja, palvelinkoneen sekä GitLab-asennuksen päivittämiseen riittävän usein. Tämä aiheuttaa huomattavan ylläpidollisen kuorman ja on yksi syy siihen, että ohjelmistoyrityksissä ollaan siirtymässä itse hallitusta versionhallinnan keskustietovarastosta palvelupohjaiseen malliin. GitLab Inc. julkaisee uuden version GitLabista kuukausittain ja suosittaa seuraamaan tätä julkaisutahtia, mikäli mahdollista. Nämä kuukausittaiset päivitykset on laadittu siten, että ne ovat taaksepäin yhteensopivia edellisen version kanssa, jotta jatkuva päivittäminen olisi helpompaa.

5. REPOLAINEN

Tampereen teknillisessä yliopistossa vuonna 2018 suurimmalla GitLabia käyttävällä kurssilla oli 550 opiskelijaa. Näiden opiskelijaprojektien luominen selainkäyttöliittymän kautta olisi ollut valtava työmäärä käsin tehtynä, joten tämän kaltaisen prosessit kannattaa automatisoida.

Repolainen on Tampereen teknillisellä yliopistolla käytössä oleva, selainkäyttöinen GitLabin ja harjoitustöiden palautusten hallintatyökalu. Se on toteutettu osana Mika Mäenpään diplomityötä Tietotekniikan laitokselle vuosina 2014-2016. Repolainen luotiin vastaamaan tarpeeseen saada hallintatyökaluja korvaamaan Subversion-versionhallintajärjestelmän kanssa yhdessä käytössä ollut KurssiSVN-järjestelmä, kun Ohjelmistekniikan laitos siirtyi käyttämään Git-versionhallintajärjestelmää yhdessä GitLabin kanssa. Repolainen on toteutettu Python 3.3 -ohjelmointikieltä ja web-ohjelmistokehitykseen tarkoitettua Django-ohjelmistokehystä.

Repolaisen avulla kurssin vastuuhenkilö voi automatisoidusti luoda opiskelijoille kurssikohtaisia GitLab-projekteja. Lisäksi Repolainen tarjoaa kehyksen palautusten tekemiseen Gitin avulla sekä mahdollisuuden palautusten tarkasteluun helposti.

5.1 Tärkeimmät käsitteet

Tärkeimmät repolaiseen liittyvät käsitteet avataan tässä aliluvussa. Käsitteiden avaaminen auttaa ymmärtämään luvussa 6 kuvattua työnkulkua. Tällä hetkellä Repolaisen ainoa käyttöliittymäkieli on englanti ja termien kuvailun yhdessä mainitaan myös englanninkielinen termi. Kaikkia Repolaiseen liittyviä termejä ei avata tässä luvussa, ainoastaan tämän työn kannalta tärkeimmät.

5.1.1 Kurssi ja toteutuskerta

Repolaisessa kurssi (engl. course) vastaa opinto-oppaasta löytyvää opintojaksoa. Kurssin käsitteeseen kuluu nimi, sekä siitä vastuussa olevat opettajat, sekä lisäksi siihen liittyvät kurssin toteutuskerrat.

Kurssin toteutuskerta (engl. course implementation) vastaa tiettynä ajankohtana järjestettyä kurssia. Toteutuskertaan liitty nimi, tunniste, järjestämisvuosi, sekä toteuskerralle osallistuvat opetusassistentit. Suurin osa Repolaisen toiminnallisuudesta toimii toteutuskertojen ympärillä, sillä toteutuskertaan liittyy myös opiskelijat, tehtävät sekä ryhmät, mikäli kurssin harjoitustyö on ryhmätyö.

5.1.2 Kurssihenkilökunta

Repolaisessa kurssihenkilökuntaa on kahta eri tasoa. Opettajat liittyvät aina kurssiin ja opetusassistentit vain kurssin toteutuskertaan. Vain opettajilla on mahdollisuus hallinnoida itse kurssia ja lisätä sille muita opettajia ja muokata opiskelijoiden muodostamia. Lisäksi opettajilla ja opetusassistenteilla on erilaiset mahdollisuudet hallinnoida kurssin toteutuskertaa.

Opettajilla ja opetusassistenteilla on lisäksi erilaiset oikeudet kurssin toteutuskertaa vastaavaan GitLab-ryhmään. Opettajilla on omistajatason (engl. owner) oikeudet, siinä missä taas assistenteille on vain mestaritason (engl. master) oikeudet. Omistajilla on täydet oikeudet ryhmän hallinnointiin, kun taas mestarilla hieman rajatummat.

5.1.3 Opiskelijat ja ryhmät

Opiskelija (engl. student) vastaa kurssin osallistujaa. Opiskelijaan liittyy etunimi, sukunimi, sähköpostiosoite, intranet-tunnus sekä opiskelijanumero. Kurssin henkilökunta lisää opiskelijat kurssille, eli opiskelijat eivät ilmoittaudu itse. Mikäli opiskelijoiden halutaan tekevän kurssin harjoitustyö ryhmätyönä, voidaan kurssin toteutuskerralle määritellä myös vapaasti valittava määrä ryhmiä. Kurssiin liittyvät opiskelijat ja ryhmät voi joko luoda yksi kerrallaan tai tuoda sopivan tiedostoformaatin avulla. Tuomisen työnkulku on kuvattu luvussa 6.

5.1.4 Tehtävä ja palautus

Tehtävä (engl. excercise) opettajan määrittämä tehtävä, johon opiskelijan tulee tehdä palautus (engl. submisson). Tehtävään liittyy aina nimi, sekä aukeamisajankohta ja sulkeutumistakaraja (engl. hard deadline). Lisäksi on mahdollista määrittää niin sanottu pehmeä takaraja (engl. soft deadline). Pehmeän takarajan avulla voidaan mahdollistaa se, että harjoitustyön voi palauttaa vielä palautuksen takarajan jälkeen, mutta myöhästyneen palautuksen voi ottaa huomioon arvostelussa. Palautuksen tarkastuksen tekevä henkilö pystyy antamaan palautetta tehdystä palautuksesta

sekä antamaan sille pistemääräisen arvioinnin.

5.2 Kommunikaatio GitLabin ja Repolaisen välillä

Repolainen hallinnoi GitLabia pääasiassa REST APIn kautta. Repolaista varten luodaan GitLabiin paikallinen käyttäjätunnus, jolla on pääkäyttäjäoikeudet kyseiseen GitLab-asennukseen. Tämän jälkeen kyseiselle käyttäjälle luodaan API key, jonka avulla Repolainen pystyy autorisoimaan GitLabin REST APIin tehdyt kyselyt.

Kun Repolaiseen luodaan uusi kurssin toteuskerta, Repolainen luo GitLabiin toteutuskertaa vastaavan ryhmän. Kurssin henkilökunta lisätään jäseniksi tähän ryhmään aiemmin mainituilla pääsyoikeusasetuksilla ja ryhmään luodaan opiskelijoiden GitLab projektien pohjana toimiva template_project. Tämän jälkeen kurssin vastuuhenkilö voi lisätä opiskelijoille jaettavaksi tarkoitetut tiedostot ja niiden muutoshistorian template_projectin tietovarastoon.

Repolainen luo opiskelijoille tai opiskelijaryhmille tarkoitetut GitLab-projektit REST API:n kautta ja lisää ryhmään kuuluvat opiskelijat kehittäjätasoisilla pääsyoikeuksilla jäseniksi tähän ryhmään. Mikäli opiskelijoiden tietoja ei löydy GitLabista, lisätään heidät GitLabiin Repolaisessa olevien tietojen perusteella. GitLab-kirjautumisen tapahtuessa intranet-tunnuksen avulla, luotu käyttäjä linkittyy automaattisesti intranet-tunnusta käyttävään henkilöön.

Repolainen hakee GitLabista juuri luodun tyhjän Git-tietovaraston. Tämän jälkeen Repolainen hakee toteutuskerran template_projectista tiedostot ja historian juuri haettuun tyhjään tietovarastoon. Lopuksi Repolainen puskee muutokset ryhmän etätietovarastoon. Tämän seurauksena template_projectilla ja opiskelijoiden projekteilla on yhteinen historian alkupiste. Opiskelijoiden on mahdollista hakea template_projectiin puskettuja muutoksia ja yhdistää ne osaksi omaa paikallista tietovarastoaan myös harjoitustyön toteutuksen aloittamisen jälkeen.

5.3 Tehtävien palauttaminen

Repolainen käyttää palautusten hallinnointiin Gitin merkintöjä (engl. tag). Opiskelijoiden tulee luoda merkintä paikallisessa tietovarastossaan ja puskea se GitLabiin. Tehtävän palauttamisen yhteydessä Repolainen osaa hakea opiskelijoiden GitLab-projektista sieltä löytyvät merkinnät, jonka jälkeen Repolainen osaa tarjota niitä omassa käyttöliittymässään palautuksen tekemiseen.

Palautuksen pystyisi tekemään myös pelkän SHA-1-tiivistesumman avulla, mutta merkintöjen käyttäminen helpottaa palautusten hallintaa niissä tapauksissa, kun kurssin aikana palautetaan enemmän kuin yksi tehtävä. Myös uusintapalautusten tekeminen helpottuu, kun palautettavan version etsimiseen voi käyttää selkokielistä nimeä.

6. VERSIONHALLINTAJÄRJESTELMIEN KÄYTTÖ TTY:N OHJELMOINTIKURSSEILLA

Tässä luvussa kuvaillaan TTY:n Tietotekniikan laboratoriolla käytössä olevat versionhallintajärjestelmät, käytännöt sekä nostetaan esiin muutamia tämänhetkisillä järjestelmillä toistaiseksi ratkaisemattomia ongelmia. Tietotekniikan laboratoriolla on opetuskäyttöä varten TTY:n tietohallinnon tarjoama GitLab-asennus. Tämän lisäksi käytössä on tähän GitLab-asennukseen yhdistetty Repolainen. Molemmat järjestelmät on yhdistetty TTY:n käyttäjähallintajärjestelmään, joten tarvetta paikallisille käyttäjätunnuksille ei ole. Lisäksi autentikointi hoidetaan Shibboleth-kertakirjautumisjärjestelmää hyväksi käyttäen.

6.1 Työnkulku yksittäisen kurssin vastuuhenkilön osalta

Mikäli kurssi haluaa alkaa käyttämään Repolaista ja sen avulla tarjota opiskelijoille GitLab-projektit harjoitustöiden tekemisen sekä palauttamiseen, ensimmäisenä pitää ottaa yhteyttä Repolaisen ylläpitäjään. Ylläpitäjää pyydetään luomaan Repolaiseen kurssi ja lisäämään vastuuhenkilö tällä kurssille opettajaksi.

Tämän jälkeen kurssiin vastuuhenkilö pystyy luomaan kurssille toteutuskerran. Toteutuskerran luomisen yhteydessä lisätään kurssin henkilökunta assistenteiksi toteutuskerralle sekä määritellään kurssin harjoitustöiden ryhmänmuodostukseen liittyvät asetukset.

Vastuuhenkilön tulee valita kurssin toteutuskerralle nimi. Oletuksena nimeksi tarjotaan yhdistelmää, jossa on kurssin nimi ja kuluva lukukausi. Vastuuhenkilön tulee lisäksi valita toteutuskerran järjestämisvuosi. Tärkein nimeämiseen liittyvä asia on kurssin kurssitunnus (engl. course identification). Tätä kurssitunnusta käytetään GitLab-ryhmän nimenä, joten se tulee näkymään myös Repolaisen ulkopuolelle. On suotavaa käyttää mahdollisimman lyhyttä kurssitunnistetta helpon muistettavuuden ja siisteyden vuoksi. Oletuskurssitunnus muodostuu kurssin nimestä ja lukuvuodesta alaviivalla erotettuna.

Oletuksena toteutuskerralla käytetään ryhmiä ja opiskelija voi kuulua vain yhteen

ryhmään. Asetusten avulla voi mahdollistaa opiskelijan kuuluminen useampaan ryhmään. Lisäksi on mahdollista valita, että kurssilla ryhmät eivät ole käytössä lainkaan, jolloin jokaisella opiskelijalle luodaan oma GitLab-projekti opiskelijanumeron perusteella.

Mikäli toteutuskerralla halutaan käyttää GitLabin Continous Integration-palvelua, sen voi halutessaan ottaa käyttöön toteutuskertaa luodessa. Repolaisen avulla pystyy määrittelemään kaikki tarpeelliset Gitlab Continous Integration-palveluun liittyvät asetukset, jotta opiskelijat saavat palvelun käyttöönsä projekteissaan.

Toteutuskertaa luodessa voi myös valita, että lähettääkö Repolainen opiskelijoille sähköpostiin ilmoituksen GitLab-projektin luomisesta. Lisäksi oletussähköpostipohjaa voi halutessaan muuttaa. Tämä tapahtuu toteutuskerran luomisen asetusten kohdasta Email.

6.1.1 Opiskelijoiden lisääminen toteutuskerralle

Toteutuskerran luomisen jälkeen vastuuhenkilön tulee lisätä opiskelijat toteutuskerralle. Opiskelijoiden lisääminen tapahtuu toteutuskertasivun Students-välilehdeltä. Tämä onnistuu joko manuaalisesti lisäämällä jokainen opiskelija yksi kerrallaan tai tuomalla tiedot CSV-tiedostomuodossa. Tuettuja CSV-formaatteja on kaksi. Toinen on Repolaisen oma yleisempään käyttöön tarkoitettu formaatti. Toisena vaihtoehtona on käyttää TTY:lle räätälöityä, opiskelijatietojärjestelmä POP/ROCKin kanssa yhteensopivaa formaattia.

Repolaisen omassa formaatissa tulee olla otsikkorivi sisältäen sarakkeet Username(opiskelijan intranet-tunnus), First name(etunimi), Last name(sukunimi), Email (sähköpostiosoite) sekä Student number(opiskelijanumero). Erottimena käytetään pilkkua. Mikäli listassa olevaa opiskelijaa ei löydy Repolaisen tietokannasta, lisätään opiskelijan tiedot tietokantaan. Mikäli opiskelijat varmasti löytyisivät Repolaisen omasta tietokannasta, olisi mahdollista käyttää tietojen tuomiseen vain intranet-tunnusta.

Täydellisen tietojen antaminen on turvallisempaa, sillä Repolainen pitää huolen siitä, että data ei pääse turhaan kahdentumaan ja operaatio onnistuu aina. Mikäli taas annetaan vain lista opiskelijanumeroista, ja kaikkia opiskelijoita ei Repolaisen tietokannasta löydy, operaatio epäonnistuu, eikä opiskelijoita lisätä kurssille lainkaan.

POP/ROCK-yhteensopivassa tiedostoformaatissa tulee olla otsikkorivi joka sisältää sarakkeet Opnro(opiskelijanumero), Nimi(etunimi ja sukunimi) sekä Email (sähkö-

postisoite). Erottimena käytetään puolipistettä. Tässä tapauksessa Repolainen hakee käyttäjän intranet-tunnuksen ja muut tiedot LDAP-palvelimelta sähköpostiosoitteen perusteella ja lisää opiskelijan tiedot tietokantaan. Lisäksi on mahdollista antaa vain opiskelijanumerot, mutta kuten Repolaisen oman formaatin tapauksessa, täydellisten tietojen syöttäminen on varmempaa.

Opiskelijoiden tuominen noudattaa lisäys-filosofiaa [7] (Design/Importers), eli opiskelijoita tuodessa opiskelijat, joita ei ole kurssille lisätty, lisätään, mutta opiskelijoita, joita ei enää ole mukana tuotavassa tiedossa, ei poisteta kurssilta. Opiskelijat voi myös tuoda kurssille ryhmien tuomisen yhteydessä, mikäli kurssilla käytetään ryhmiä. Tämä kuvaillaan tarkemmin seuraavassa aliluvussa.

6.1.2 Ryhmien luominen

Ryhmiä voi Repolaisessa luoda käsin samaan tapaan kuin opiskelijoita, mutta tähänkin on tarjolla mahdollisuus tuoda ryhmien tiedot. Ryhmien lisääminen tapahtuu toteutuskertasivun Groups-välilehdeltä. Kurssilta puuttuvat opiskelijat lisätään kurssille ryhmien luomisen yhteydessä. Tällöin tulee käyttää Moodlen Group self-selection-lisäkkeen [3] kanssa yhteensopivaa CSV-formaattia. Tällöin tiedoston tulee olla alla olevan esimerkin kaltainen.

```
1 Group Name,Member 1 Username,Member 2 Username,Member 3 Username
2 Group1,username1,username2,username3
3 Group2,username3,username4,
```

Esimerkki 6.1 *Ryhmien tuomiseen käytettävän tiedoston rakenne. Esimerkissä ryhmän maksimikoko on 3 henkeä.*

Kuten aliluvussa 6.1.1 todettiin, voi opiskelijoiden tietoja lisätä Repolaiseen myös ryhmien luomisen yhteydessä. Tällöin CSV-tiedoston tulee olla kuuten esimerkissä 6.1, mutta tämän lisäksi jokaista opiskelijaa kohden tulee olla lisäksi sarakkeet Member n Firstname, Member n Lastname, Member n Email, Member n ID Number, jossa n on kyseisen opiskelijan järjestysnumero. Niiden opiskelijoiden tiedot, joita Repolaisessa ei vielä ole, tulee syöttää oikeisiin sarakkeisiin. Tällöin opiskelijoiden tiedot ensin lisätään Repolaiseen, sen jälkeen kurssille ja lopulta ryhmiin.

Molemmissa tapauksissa ryhmässä voi olla jäseniä alle maksimimäärän, eli osassa ryhmistä voi olla vähemmän jäseniä. Ryhmien lisääminen noudattaa korvaus-filosofiaa [7]. Käytännössä tämä tarkoittaa siihen, että toisin kuin opiskelijoita tapauksessa, ryhmiä uudelleen tuotaessa CSV-tiedoston tulee sisältää kaikkien kurssilla jo olevien ryhmien tiedot. Mikäli olemassa olevien ryhmien tietoja ei tiedostosta löydy, niitä yritetään siis poistaa Repolaisesta. Poistaminen onnistuu vain, jos

ryhmä ei ole tehnyt yhtään palautusta. Repolainen ei kuitenkaan tee poistamista suoraan vaan näyttää käyttäjälle, millaisia muutoksia on tapahtumassa ja pyytää varmistusta.

6.1.3 GitLab-projektien luominen ja hallinnointi

Ryhmien luomisen jälkeen on edessä GitLab-projektien luominen opiskelijoille. Se tapahtuu GitLab itegration-välilehdeltä. Kuten luvussa 5.2 todettu, Repolainen tarjoaa tähän automaation. Create-painike saa aikaan sen, että Repolainen alkaa selvittää, minkä verran ryhmiä tarvitsee luoda ja näyttää tulokset käyttäjälle. Tämän jälkeen käyttäjän tulee vahvistaa opiskelijoiden GitLab-projektien luominen.

Luomisen jälkeen Create-painikkeen tilalle tulee Update-painike, jonka avulla on mahdollista päivittää Repolaiseen tehdyt muutokset myös GitLabiin. Näitä muutoksia ovat uusien ryhmien lisääminen, ryhmien kokoonpanon muuttaminen, sekä ryhmien poistaminen. Lisäksi muutokset toteutuskerran Continous Integration -asetuksiin saadaan ajettua opiskelijoiden GitLab-projekteihin tätä kautta.

6.1.4 Toteutuskerran hallinnointi

Toteutuskerran asetuksia on mahdollista muuttaa Settings-välilehdellä. Muuttaa voi kaikkia kurssin toteutuskerran luomisen yhteydessä määriteltyjä asetuksia. Joidenkin asetuksien muuttaminen, esimerkiksi kurssitunnisteen vaihtaminen aiheuttaa ongelmia, sillä opiskelijoiden tietovarastojen polku vaihtuu myös. Tästä syystä Repolainen kysyy varmistuksen ennen kurssitunnisteen muuttamista.

6.1.5 Tehtävien lisääminen ja muokkaaminen

Opiskelijoille tarkoitettujen tehtävien lisääminen tapahtuu Repolaisen Exercises-välilehdeltä. Tehtävien laatimiseen Repolainen tarjoaa todella paljon asetusmahdollisuuksia.

General-kohdasta voi määritellä tehtävän yleiset asetukset, eli nimen, onko tehtävä näkyvissä opiskelijoille heti luomisen jälkeen, antaako opetushenkilökunta arvosanan tehtävästä sekä tehdäänkö palautukselle automaattitarkastuksia. Tältä välilehdeltä asetaan myös tehtävän aukeamisajankohta sekä takarajat. Tehtävälle voi määrittää niin sanotun kovan takarajan lisäksi pehmeän takarajan. Kovan takarajan jälkeen

palautuksia ei oteta enää vastaan, eli pehmeää takarajaa voi käyttää tehtävän varsinaisena takarajana, jonka jälkeen on vielä mahdollisuus palauttaa tehtävä, mutta arviointiin voi tulla esimerkiksi pistevähennyksiä.

Grading-välilehdeltä määritellään tehtävän arviointiin liittyvät asetukset. Opiskelijoille on mahdollista näyttää joko arvosana tai hyväksytty/hylätty. Lisäksi on mahdollista määrittää pisteiden minimi- ja maksimimäärät sekä kuinka paljon pisteitä tarvitaan läpipääsyyn. Pisteiden määrä voidaan määrittää vapaasti kokonaislukujen joukosta, eli pistemääräksi voi määrittää esimerkiksi arvosana väliltä 0-5 tai jokin muu lukuväli.

Submissions-välilehdeltä voi halutessaan rajoitaa palautuskertojen maksimimäärää, sekä valita, halutaanko opiskelijalle halutaan näyttä vain viimeisin palautus. Lisäksi voi valita, näytetäänkö automaattitestauksen tuloksia ennenkuin opettaja on arvostellut palautuksen. Myös uusintapalautusten tekemistä voi halutessaan rajoittaa. Kaikkia tässä aliluvussa mainittuja asetuksia on mahdollista muuttaa tehtävän luomisen jälkeen, myös vaikka tehtävä olisi jo näkyvissä opiskelijoille.

6.1.6 Uusien ohjelmakoodipohjien ja muutosten julkaisu

Kurssin vastuuhenkilön on mahdollista uutta materiaalia opiskelijoille template_projectin avulla. Opettaja voi tehdä kehitystyötä itse kloonaamassaan template_projectissa ja tämän jälkeen työntää muutokset opiskelijoiden saataville GitLabiin. Tässä tapauksessa opiskelijat tulee ohjeistaa lisäämään template_projectin osoite etätietovarastoksi omaan paikalliseen tietovarastoon ja hakemaan opettajan työntämät muutokset template_projectista Gitin avulla.

Vaihtoehtoisesti voidaan käyttää erillistä GitLab-projektia päivitysten jakamiseen ja käyttää template_project vain opiskelijoiden tietovarastojen alustamiseen. Tällöin myös tämän erillisen projektin tulee pohjautua template_projectiin, jotta versiohistorioiden yhdistäminen opiskelijoiden tietovarastojen kanssa olisi ongelmatonta.

7. MUUT VAIHTOEHDOT

Tässä kappaleessa käsitellään muita mahdollisuuksia versionhallinnan käyttämiseen opetuksen työkaluna. Nämäkin ratkaisut keskittyvät Gitin ympärille, johtuen sen käytön yleisyydestä ja saatavilla olevien järjestelmien runsaudesta.

7.1 GitLab ilman Repolaista

GitLab itesessään on tarkoitettu yhteistyöskentelyn mahdollistamiseen, joten sen avulla pystyy mahdollistamaan opiskelijoiden yhteistyöskentelyn tarpeen vaatiessa. Lisäksi GitLabiin on mahdollista luoda henkilökohtaisia projekteja, joten myös yksin tehtävien harjoitustöiden tekeminen on mahdollista. Tämä vaatisi opiskelijoilta enemmän perehtyneisyyttä Gitin ja GitLabin käyttöön, kun suuri osa versionhallintajärjestelmän käyttöön liittyvästä työstä jäisi heidän harteilleen.

Toinen vaihtoehto olisi hyväksikäyttää GitLabin tarjoamaa REST APIa ja sen avulla luoda opiskelijoille projektit. Tässä tapauksessa esivaatimuksena olisi, että kaikki opiskelijat ovat käyneet kirjautumassa kertaalleen GitLabiin, jotta paikallinen käyttäjätunnus on olemassa. Repolaisen sijaan olisi mahdollista käyttää komentorivis-kriptejä, jotka käyttäisivät GitLabin tarjoamaa REST APIa opiskelijoiden projektien luomiseen. Näitä skriptejä olisi mahdollista jakaa kurssien vastuuhenkilöiden kesken ja kukin voisi ajaa skriptit oman pääsytunnuksensa avulla. Ilman tällaisten skriptien kirjoittamista varsinkin suurien kurssin hallinnointi GitLabissa olisi huomattavan työlästä ja vaatisi suuren määrän käsin tehtyä työtä.

7.2 GitHub.com

GitHub on vuodesta 2007 asti toiminut, palvelumuotoinen etätietovarastojärjestelmä, joka tarjoaa GitLabin tavoin Git-etätietovastaston sekä projektin käsitteen. Github.com tarjoaa GitLabin tapaan projektin käsitteen. GitHub-projekti sisältää hyvin samankaltaiset ominaisuudet kuin GitLab-projekti, mutta termistössä on hieman eroja. GitLab-projektin jäsentä vastaava termi on GitHubissa työtoveri (engl. collaborator) ja haarojen yhdistämispyyntöjen sijaan GitHubissa on nyhtöpyynnöt (engl.

pull request). Pääpiirteissään GitHubia voidaan pitää ominaisuuksiltaan GitLabin kanssa samalla tasolla olevana.

GitHubia käytetään useisiin avoimen lähdekoodin projekteihin. GitHubiin voi rekisteröityä kuka tahansa ja projekteihin on helppo osallistua luomalla oma niin kutsuttu haarukka (engl. fork) projektista ja alkaa työstämään sitä. Lopulta oman työpanoksensa voi pyytää yhdistettäväksi pääprojektiin äsken mainitun nyhtöpyynnön avulla. GitHub on myös teollisuuden kannalta relevantti järjestelmä, sillä monet avoimen lähdekoodin kehitystä tekevät yritykset käyttävät GitHubia kehitys- ja julkaisualustana.

Etuna tässä ratkaisussa itse hallinnoituun GitLab-asennukseen on se, että järjestelmän käytöstä ei synny ylläpitokustannuksia. Toisaalta käyttäjähallintajärjestelmän yhdistäminen GitHub.com-palveluun ei ole mahdollistam joten opiskelijat ja henkilökunta joutuisivat luomaan erilliset käyttäjätunnukset GitHubiin.

7.2.1 GitHub Education

GitHub Education on GitHub Inc.:n tarjoama palvelupaketti, joka tarjoaa työkaluja opetuksen järjestämiseen Gitiä hyödyntäen. Palvelupaketti on saatavissa maksullisena. Kiinteää hinnoittelua ei ole, vaan paketin hinnoittelu räätälöidään aina tarpeen mukaan.

GitHub Classroom on GitHub Education -pakettiin kuuluva osa, jonka avulla opettajat voivat automatisoida opiskelijoiden GitHub-projektien luomisen ja jakaa valmista ohjelmakoodia opiskelijoille. Lisäksi GitHub Classroomin selainkäyttöliittymässä on mahdollista tarkastella ja analysoida opiskelijoiden harjoitustöiden etenemistä.

Classroomin avulla on mahdollista hallinnoida sekä ryhmässä, että yksin tehtäviä harjoitustöitä. Tämän lisäksi Classroomin avulla voidaan tuoda harjoitustyön pohjakoodi suoraa linkin avulla mistä tahansa GitHub-projektista.

GitHub Educationin käyttäminen vaatii luonnollisesti rekisteröitymisen GitHubiin sekä opiskelijoilta että opetushenkilökunnalta. Opiskelijoiden lisääminen Classroomiin onnistuu kuitenkin myös kutsulinkin avulla, joten kaikkien opiskelijoiden GitHub-tunnuksia ei ole tarpeen tietää tai kerätä etukäteen.

GitHub Education on tällä hetkellä ominaisuuksiltaan vielä hieman rajallinen. Esimerkiksi mekanismia palautusten määräaikojen määrittämiseen ei ole. Opiskelijoiden tekemiä palautuksia on toki mahdollista seurata niiden SHA-1 tunnisteiden avulla ja katsoa koska pysyvät muutokset on tuotu etätietovarastoon. Lisäksi palautteen

välittämiseen opiskelijoille ei ole olemassa muuta mekanismia kuin lisätä tiedostoja opiskelijoiden tietovarastoihin tai kommentoida suoraan lähdekooditiedostoihin.

8. ARVIOINTI

Tässä työssä on käsitelty TTY:n ratkaisua opetuksen järjestämiseen versionhallintaa hyväksi käyttäen, sekä pintapuoleisesti pohdittua muutamia muita vaihtoehtoja. Tässä luvussa käydään osa läpi esitellyistä vaihtoehdoista ja vertaillaan niiden hyviä ja huonoja puolia TTY:n Tietotekniikan laboratorion ratkaisuun.

8.1 GitLab vs. GitHub.com

Itse hallinnoidun etätietovarastojärjestelmän etuna on mahdollisuus yhdistää se käytössä olevaan pääsyn- ja käyttäjähallintaan. TTY:llä, kuten myös useilla muilla on paljon erilaisia opetus- ja oppilastietojärjestelmiä, joita käytetään samoilla tunnuksilla. Mahdollisuus yhdistää versionhallintajärjestelmä osaksi tätä kokonaisuutta helpottaa työskentelyä, kun ei tarvitse muistaa erillisiä käyttäjätunnuksia ja salasanoja.

Opiskelijaprojektien säilöminen GitHub.comissa olisi siinäkin mielessä ongelmallista, että TTY:n ohjeiden mukaan opiskelijoiden harjoitustöitä on säilytettävä tietty aika eikä niitä saa turhaan säilyttää liian kauaa. GitHub.comia käytettäessä säilytysaikojen hallinta olisi haastavaa.

Kuten kaikkien itsehallinnoitujen tietojärjestelmien tapauksessa, myös GitLabin hallinnointi vaatii joissain määrin ylläpidollista työtä. Lisäksi rajanvedon tekeminen siitä, mitä järjestelmiä laboratorio itse ylläpitää ja mitä taas yliopiston tietohallinto tarjoaa, saattaa olla vaikeata. TTY:llä tämä on kuitenkin saatu hoidettua melko kivuttomasti tekemällä tiivistä yhteistyötä Tietotekniikan laboratorion ja TTY:n Tietohallinnon välillä.

8.2 Repolainen vs. GitHub Education

Repolaisessa on monipuoliset mahdollisuudet tehtävien palautusten käsittelemiselle. Erityisesti kahden eri määräajan määrittelemisen helppous ja sen tarjoama joustavuus antavat vastuuhenkilölle uusia tapoja määritellä harjoitustehtävien pisteytystä.

GitHub Educationin tarjoamat hallinnointimahdollisuudet ovat huomattavasti rajoittuneemmat kuin Repolaisessa. GitHub Educationin avulla luodut projetit säilötään GitHubissa, jolloin opintojen ohessa tehtyjen harjoitustöiden käyttäminen referensseinä työnhaussa helpottuu, kun tehdyn työn saa helpommin julkisesti näkyville.

8.3 GitLab Repolaisen kanssa vs. ilman Repolaista

Repolainen tarjoaa mahdollisuuden automatisoida opiskelijoiden GitLab-projektien luomisen sekä valmiin ohjelmakoodin jakamisen. Lisäksi Repolainen helpottaa uusien materiaalien julkaisemista myös projektien luomisen jälkeen, sillä pohjana toimivassa projektissa ja opiskelijoiden projekteissa oleva historia on yhteinen.

Kurssin opetushenkilökunnalla on automaattinen pääsy kaikkiin opiskelijoiden GitLab-projekteihin ilman, että opiskelijat näkisivät suoraan toistensa projekteja. Tällainen pääsynhallinnan hienojakoisuuden toteuttaminen olisi työlästä ilman Repolaisen tarjoamaa automaatiota. Mi

Huonona puolena Repolaisesta voi mainita sen, että opiskelijat joutuvat käyttämään vielä yhtä ylimääräistä järjestelmää. Repolaisen käyttöliittymä ei myöskään ole aina kaikkein intuitiivisin, eikä järjestelmän käyttöliittymän kehitykseen ole juurikaan käytetty resursseja.

Repolaisen ylläpidon ja jatkokehittämisen kanssa on ollut huomattavia vaikeuksia. Mika Mäenpään työsuhteen päätyttyä Repolaista ei ylläpitänyt eikä jatkokehittänyt kukaan. Myöhemmin ylläpito sälytetiin sellaisen henkilön harteille, joka ei ollut osallistunut Repolaisen kehitystyöhön eikä ollut kovinkaan kokenut Repolaisen toteutusteknologioiden kanssa. Kesällä 2017 ylläpitoon saatiin hieman paremmin resursseja, mutta tilanne ei vieläkään ole ideaalinen.

GitLabista julkaistaan uusi versio kuukausittain ja vaikka nämä julkaisut pyritään pitämään taaksepäin yhteensopivina, saattaa ongelmia esiintyä inhimillisten virheiden vuoksi. Tämän vuoksi Repolaisen toiminta uusimman GitLab-version kanssa tulee varmistaa aina ennen uusimpaan versioon päivittämistä. Lisäksi taaksepäin yhteensopimattomien päivitysten yhteydessä saatetaan tarvita laajempaa refaktorointi- ja päivitystyötä.

Harjoitustöiden palautuksen työnkulussa on Repolaisen kohdalla parantamisen varaa. Vaikka palautuksen ajoissa olemisesta ja muuttumattomuudesta voidaan varmistua, on tämänhetkinen työnkulku aiheuttanut ongelmia joillakin kursseilla, sillä

esimerkiksi Gitin merkintöjen luominen ja niiden etätietovarastoon viemisen muistaminen. Lisäksi varsinkin opintojen alkuvaiheessa versionhallintajärjestelmien käyttäminen on ollut haparoivaa. Näitä ongelmia voisi välttää versionhallintajärjestelmien käytön ja perusperiaatteiden opetusta lisäämällä peruskurssien yhteydessä.

9. YHTEENVETO

Tässä työssä käytiin läpi mahdollisuuksia hyödyntää erilaisia versionhallintajärjestelmiä opetuksen tukena. Läpikäydyt järjestelmät olivat GitLab, Repolainen, GitHub.com sekä GitHub Education, joista TTY:n ohjelmointikursseilla on käytössä GitLab tuettuna Repolaisella.

TTY:n Tietotekniikan laboratoriolla käytössä olevan toimintatavan voidaan todeta mahdollistavan yhteistyöskentelyn, valmiin ohjelmakoodin jakelun tehokkaasti ja helposti sekä tarjoavan mahdollisuuden harjoitustöiden palautusten hallinnointiin tarkoituksenmukaisella tavalla. Työläät ja käsin tehtynä aikaa vievät toimenpiteet on automatisoitu ja niistä on tehty niin helppokäyttöisiä, että versionhallintajärjestelmän käyttöön ottaminen opetuksessa onnistuu myös ilman syvällistä ymmärtämistä käytettyjen järjestelmien toimintaperiaatteista.

Suurimmaksi ongelmaksi havaittiin, että nykyistä paremmin pitäisi pitää huolta siitä, että mikäli Repolaista halutaan käyttää jatkossakin ohjelmontikurssien järjestämien tukena, sen ylläpidolle ja jatkokehitykselle varataan riittävästi resursseja. Tämänhetkinen tilanne, jossa kyseistä järjestelmää ylläpitää ja kehittää vain yksi henkilö äärimmäisen riskialtis. Mikäli tämä kyseinen henkilö poistuu organisaatiosta jostain syystä, täytyy järjestelmä hylätä ellei löydy riittävällä aikataululla henkilöä, jolla on aikaa aloittaa järjestelmään perehtyminen tyhjästä.

LÄHTEET

[1] S. Chacon and B. Straub, *Pro Git - Everything you need to know about Git*. Apress, 2014. [Online]. Available: https://progit2.s3.amazonaws.com/en/2015-09-04-9cc92/progit-en.812.pdf

[2] B. Collins-Sussman, B. W.Fitzpatric, and C. Pilato, *Version Control with Subversion*. O'Reilly Media, 2011. [Online]. Available: http://svnbook.red-bean.com/en/1.7/svn-book.pdf

[3] M. Glancy, P. Škoda, H. Foster, and P. Pyykkönen, "Group self-selection module documentation," 2016. [Online]. Available: https://docs.moodle.org/34/en/Group_self-selection_module

[4] G. Inc., "Gitlab community edition documentation: Subgroups," 2018. [Online]. Available: https://docs.gitlab.com/ce/user/group/subgroups/index.html

[5] ——, "Gitlab community edition documentation: Subgroups," 2018. [Online]. Available: https://docs.gitlab.com/ee/api/README.html

[6] ——, "Gitlab community edition documentation: System requirements," 2018. [Online]. Available: https://docs.gitlab.com/ce/install/requirements.html

[7] M. Mäenpää, "Repolainen manual," 2016.