Thomas Gustafsson

Ohjelmointikurssin oppimisympäristön suunnittelu ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

26.4.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Thomas Gustafsson Ohjelmointikurssin oppimisympäristön suunnittelu ja toteutus 22 sivua + 1 liite 26.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Yliopettaja Erja Nikunen

Opinnäytetyön tarkoituksena oli toteuttaa uudenlainen työelämälähtöinen oppimisympäristö uudelle ohjelmointikurssille. Toteutuksen lähtökohtana oli työelämälähtöisyyden lisäksi helppokäyttöisyys ja tarkoituksenmukaisuus. Ympäristön toteutuksessa otettiin myös huomioon toteutuksen kiireellinen aikataulu ja järjestelmässä käytetiin hyväksi mahdollisimman paljon jo olemassa olevia toteutuksia.

Työssä käsitellään kahden vaihtoehtoisen toteutustavan valinta. Molemmat vaihtoehdot esitellään lyhyesti ja testataan käytännössä. Vaihtoehtoja vertaillaan ja lopullinen valinta perustellaan.

Varsinainen toteutus toteutettiin käyttäen Jenkins-sovellusta ja Subversion-versionhallintaa. Tämän lisäksi toteutuksessa toteutettiin Bash-skriptejä ympäristön konfiguroimiseen sekä Gradle-koonti toimitettujen tehtävien testaamiseen ja tulosten raportointiin. Työssä esitellään myös tuotantokäytössä toteutuksessa esiintyneet ongelmat ja niiden korjaukset.

Opinnäytetyö onnistui kohtalaisen hyvin. Aikataulutuksessa olisi kuitenkin ollut parannettavaa. Lisäksi joitakin työn osaksi suunniteltuja ominaisuuksia ei aikataulun puitteissa saatu toteutettua, vaan työ keskittyi pääominaisuuksien toteutukseen.

Avainsanat	oppimisympäristö, Jenkins, versionhallinta

Author(s) Title Number of Pages Date	Thomas Gustafsson Design and implementation of a learning environment for a programming course 22 pages + 1 appendice 26 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Erja Nikunen, Principal Lecturer

The purpose of this thesis was to implement a new work life -inspired studying environment for a programming course. The basis of this implementation was to produce an easy to use and fit to purpose environment for studying. Tight schedule had to be taken into consideration when implementing, and because of that the usage of existing implementations was preferred.

The selection process of the implementation design is described in this thesis. There were two alternative designs, which are both introduced and proof of concept -tested. There is also a comparison of the two designs included.

The actual implementation was done using Jenkins-software and Subversion version controlling system. Bash-scripts and Gradle build tool was also used in the implementation for configuring the environment-, testing- and reporting purposes. Actual problems and fixes in production-phase are also presented in this thesis.

The thesis and the project succeeded fairly well, though there could have been improvements in the schedules. Also, some parts of the implementation originally designed were left unimplemented, as due to schedules the implementation was focused on the main features only.

Keywords	studying environment, Jenkins, version control

Sisällys

Lyhenteitä ja määritelmiä

Apache	Avoimen lähdekoodin HTTP-palvelinohjelma.
Bash	Bourne again shell, Useimpien Linux-jakeluiden oletuskomentotulkki.
Eclipse	Ohjelmointiympäristö, joka tukee esimerkiksi Java-ohjelmointikieltä.
Git	Avoimen lähdekoodin hajautettu versionhallintaohjelmisto.
Gitlab	Verkkoselaimella käytettävä Git-versionhallinnan hallintaan tarkoitettu käyttöliittymä.
IDE	Integrated development environment. Ohjelmointiympäristö, jolla suunnitellaan ja toteutetaan ohjelmia.
Java	Oraclen kehittämä ohjelmointikieli ja ohjelmointialusta.
JUnit	Sovelluskehys yksikkötestien toteuttamiseen Java-ohjelmointikielellä.
LDAP	Lightweight directory access protocol. Käyttäjäntunnistukseen ja käyttöoikeuksien tarkistamiseen tarkoitettu verkkoprotokolla.
Moodle	Avoimen lähdekoodin oppimisalusta.
Ruby	Tulkittava skripti-tyylinen ohjelmointikieli.
Ruby on rails	Avoimen lähdekoodin Ruby-ohjelmointikieleen perustuva ohjelmistokehys.
TMC	TestMyCode, katso alta.
TestMyCode	Avoimen lähdekoodin ohjelmointikurssien tehtävien tarkistustyökalu.
Tuubi	Metropolia Ammattikorkeakoulun käyttämä tiedotus- ja työtilasivusto.

URL Uniform resource identifier. Merkkijono, jolla osoitetaan tiedon paikka, esimerkiksi verkkosivun osoite.

Ubuntu Linux-jakelu. Avoimen lähdekoodin tietokoneen käyttöjärjestelmä perustuen Linux-ytimeen.

1 Johdanto

Työn tavoitteena oli toteuttaa oppimisympäristö uudelle kurssille huomioiden opetussuunnitelman uusimmat linjaukset. Ympäristöstä haluttiin toteuttaa työelämälähtöinen, mikä tarkoittaa käytännössä työelämälähtöisten tekniikoiden käyttämistä. Lisäksi oleellista oli toteuttaa ympäristö mahdollisimman helppokäyttöisenä opiskelijoille ja keventää opettajien työtaakkaa tehtävien tarkistuksessa niin pitkälle kuin mahdollista.

Työssä esitellään toteutuksen matka suunnittelusta vaihtoehtojen kautta lopulliseksi tuotokseksi. Lisäksi tarkoituksena oli oppia projektityön toteutuksesta, haasteista matkan varrella sekä niiden ratkaisemisesta. Kaikki työn vaihteet on kerrottu ja valinnat perusteltu. Lopussa kerrataan vielä tulokset ja työn aikana opitut opetukset.

Lisäksi työn mukana toimitetaan liite käytännön ohjeistuksesta opiskelijoille, josta saa kuvan, millaista järjestelmää olisi käyttää opiskelijan näkökulmasta. Ohje ei ole lopullinen opiskelijoille tarkoitettu versio, vaan lähinnä kurssikohtaisesti pohjaksi tarkoitettu.

2 Ongelman esittely

2.1 Taustatiedot

2000-luvulle tultaessa lisääntynyt digitaalisuus on muovannut koulu- ja opetusmaailmaa perinteisistä luokkahuoneista ja luennoista yhä enemmän digitaalisempaan suuntaan. Digitaaliset työkalut antavat mahdollisuuden toisenlaiseen lähestymistapaan koko oppimisessa. Esimerkkeinä tästä ovat esimerkiksi yhä lisääntyvät virtuaalikurssit: sen sijaan, että opettaja opettaisi suoraan opetettavat asiat oppilaiden tai opiskelijoiden fyysisesti läsnäollessaan, useissa virtuaalikursseissa opettajan tehtävä on ennemminkin motivoida ja ohjata oppimista. [1.]

Digitaalisuus mahdollistaa monia asioita, jotka perinteisellä opetustavalla ovat olleet hankalia tai jopa mahdottomia. Esimerkiksi jo opetusmateriaalin siirtäminen fyysisestä oppikirjasta digitaaliseen materiaaliin mahdollistaa oppimateriaalin moninkertaistamisen ilman fyysisen kirjan rajoituksia. Oppimateriaali on myös digitaalisessa muodossa huomattavasti helpompi tarvittaessa muokata vastaamaan kurssin opetussuunnitelmia.

Tämä on useasti vain alkua: opetusmateriaaleja voidaan vapaasti hyödyntää useimmiten esimerkiksi suoraan internetistä. Tämä myös kouluttaa tarpeellisia ongelmanratkaisutaitoja. [1.]

Seuraava looginen mahdollisuus, jonka digitaalisuus tuo mukanaan on riippumattomuus ajasta ja paikasta. Digitaalinen opetus ja oppimateriaali mahdollistavat opiskelun hyvin pitkälti missä ja milloin tahansa – milloin tämä opetettavalle parhaiten sopii. Tämä antaa mahdollisuuden esimerkiksi käyttää perinteiset päivä- ja iltaopiskeluajat töiden tekemiseen tai muuhun asioiden hoitamiseen. Myös tiloista poistuminen antaa mahdollisesti mahdollisuuden koulutuskapasiteetin nostoon. [1.]

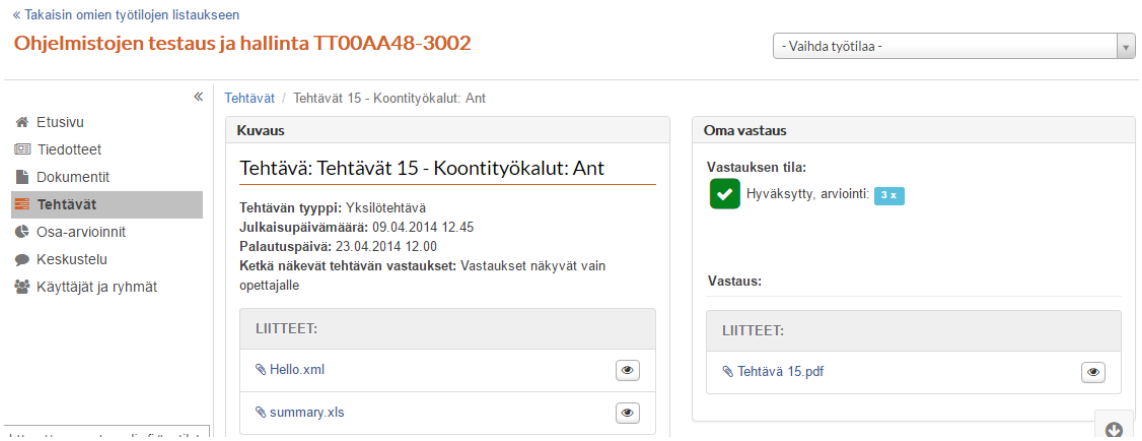
Tietenkin pitää ottaa huomioon, että kaikkialle ja kaikkiin paikkoihin eivät digitaaliset metodit välttämättä sovi. Esimerkiksi erilaisia fyysisiä taitoja on hankala opettaa etäopiskeluna. Tämä ei kuitenkaan tarkoita, etteikö tähänkin voitaisi erilaisia digitaalisia menetelmiä soveltaa esimerkiksi opetusmateriaalien muodossa. Kaikki erilainen tietotyö on usein mahdollista opettaa täysin digitaalisesti, kun mitään fyysistä taitoa ei tarvita.

Digitaalisessa oppimisessa on myös oleellista käytettävien työkalujen tarkoituksenmukaisuus. Hankalasti käytettävät työkalut voivat jopa pilata digitaalisuuden tuomat hyödyt. Työkalut ovat nykyään vielä useasti hieman keskeneräisiä tai tarpeettoman haastavia käyttää, eivätkä ne välttämättä vastaa työelämässä käytettäviä työkaluja. Tästä päästään tämän työn aiheeseen, jossa eräälle tietylle opetuskurssille suunniteltiin ja toteutettiin tarkoituksenmukainen oppimisympäristö oppimisen tueksi ja jopa opettamaan työelämässä käytettävien työkalujen käyttöä.

2.2 Ongelman kuvaus

Tietotekniikan insinöörikoulutuksen opetussuunnitelma muuttui vuonna 2015. Uuden ”Oppijan polku” -mallin mukaisesti koulutus on nyt jaettu isompiin 5, 10 tai 15 opintopisteen opintokokonaisuuksiin. Polut on suunniteltu joustaviksi, ketteriksi ja haastaviksi tavoiksi toteuttaa opetus. Käytännössä tämä tarkoittaa useimmiten vanhojen erillisten kurssien yhdistämistä teemojen mukaan yhdeksi isoksi kokonaisuudeksi. Lisäksi uuden mallin yksi kantavista teemoista on työelämälähtöisyys, joka piti ratkaisussa ottaa huomioon. [3.]

Yhdelle uusista kursseista haluttiin suunnitella uusi oppimisympäristö, joka vastaisi paremmin kyseisen kurssin tarpeita. Kurssin nimi oli ”Oliosovellukset ja tietokannat” tai lyhyemmin ”Olso”, ja se keskittyi nimensä mukaisesti olio-ohjelmointiin ja tietokantoihin. Kyseisellä kurssitoteutuksella käytettiin ohjelmointiin Java-ohjelmointikieltä. Kurssin ideana oli, että opiskelijat toteuttavat valmiiksi annetun tehtävänannon ja tehtäväpohjan mukaisen toteutuksen ja testaisivat itse oman koodinsa toimivuuden tehtäväpohjan mukana toimitettavilla JUnit-yksikkötesteillä. [2.]



Kuva 1. Kuvankaappaus Oma-työtilapalvelun tiedostotyyppisestä tehtävienpalautuksesta.

Oppilaitoksella oli ennestään tarjota oppimisympäristöiksi Tuubi-työtilapalvelua (nykyisin osa Oma-työpöytäpalvelua) ja Moodle-työtiloja, jotka molemmat mahdollistavat tehtävien palautuksen tekstimuodossa tai tiedostoina (kuva 1). Nämä eivät kuitenkaan ole optimaalisia ratkaisuja ohjelmointikurssille, koska ohjelmakoodin palauttaminen tekstimuodossa tai tiedostoina on hankalaa opiskelijalle sekä hankalaa opettajalle tarkastaa. Lisäksi ohjelmointikoodin palauttaminen tiedostoina ei vastaa työelämän käytäntöjä, joissa hyvin yleisesti on käytössä jokin versionhallintajärjestelmä ohjelmakoodin säilömiseen.

Näistä syistä päätettiin lähteä tutkimaan paremmin opintojaksoa tukevan oppimisympäristön toteutusta. Pääasiallisia kriteereitä uudelle oppimisympäristölle olivat tarkoituksenmukaisuus, helppokäyttöisyys sekä opiskelijalle että opettajalle, helpot tarkistustyökalut ja työelämälähtöisyys. Lisäksi mahdollisuuksien mukaan tulisi ratkaisun käyttää hyväksi jo olemassa olevia sovelluksia ja työkaluja työmäärän vähentämiseksi aikataulupaineiden takia.

2.3 Ongelman pohdintaa

Ongelman ratkaisua suunniteltaessa ensimmäinen lähtökohta oli yksinkertaisesti päättää, mikä filosofia ottaa suunnittelussa käyttöön: onko ympäristön tarkoitus olla vain tehtävien palautukseen mahdollisimman yksinkertainen ja helppokäyttöinen työkalu sekä opettajille että opiskelijoille? Vai kuuluisiko ympäristön opettaa jo itsessään jotain käytäntöjä opiskelijalle työelämästä? Ja jos, niin mitä ja kuinka paljon? Kuinka paljon työelämän käytäntöjä täytyy helpottaa opiskelijoille? Nämä ovat kaikki tärkeitä kysymyksiä, jotka tulee ottaa huomioon ratkaisuvaihtoehtoja mietittäessä.

Yksinkertaisen ohjelmointitehtävien palautustyökalun valitsemisessa lähtökohdaksi on monta etua: ensinnäkin tämänkaltainen työkalu lähes varmasti parantaisi nykytilannetta ja toisaalta olisi yksinkertainen toteuttaa, ja valmiitakin toteutuksia olisi tarjolla. Tällä lähestymistavalla saisi tarvittaessa toteutettua oppimisympäristön viikoissa käyttäen valmiita toteutuksia. Ongelmaksi kuitenkin muodostuu vaatimus työelämälähtöisyydestä: yksinkertainen tehtävienpalautusratkaisu on hyvin kaukana työelämästä eikä toisi opiskeluun käytännössä mitään lisäarvoa.

Lisäämällä työelämän käytäntöjä kuvaan päästään nopeasti tilanteeseen, jossa onkin järkevämpää käyttää pohjana työkaluja työelämästä ja soveltaa niitä opiskelukäyttöön kuin yrittää soveltaa tehtävienpalautusjärjestelmiä vastaamaan työelämäkäytäntöjä. Tässä voitaisiin jopa mennä niin pitkälle, että ympäristö vastaisi täysin yleisiä työelämäkäytäntöjä versionhallintoineen, yksikkötesteineen, jatkuvine koonteineen, koodianalytiikoineen ja testiympäristöineen. Mutta missä vaiheessa kuvaan astuu liikaa opittavaa aloitteleville opiskelijoille? Halutaanko ympäristöön suunnitella mahdollisuus käyttää samaa ympäristöä myöhemmän opiskeluvaiheen kursseilla?

Lisäksi viimeinen ja tärkeä huomioon otettava seikka on ratkaisun ylläpidettävyys. Mitä enemmän omaa toteutusta ympäristössä on, sitä enemmän työtä sen ylläpito vaatii. Ylläpito myös väistämättä siirtyy jossakin vaiheessa jollekin muulle taholle. Ratkaisussa on mietittävä dokumentaatio kuntoon, varsinkin täysin kustomoidussa toteutuksessa.

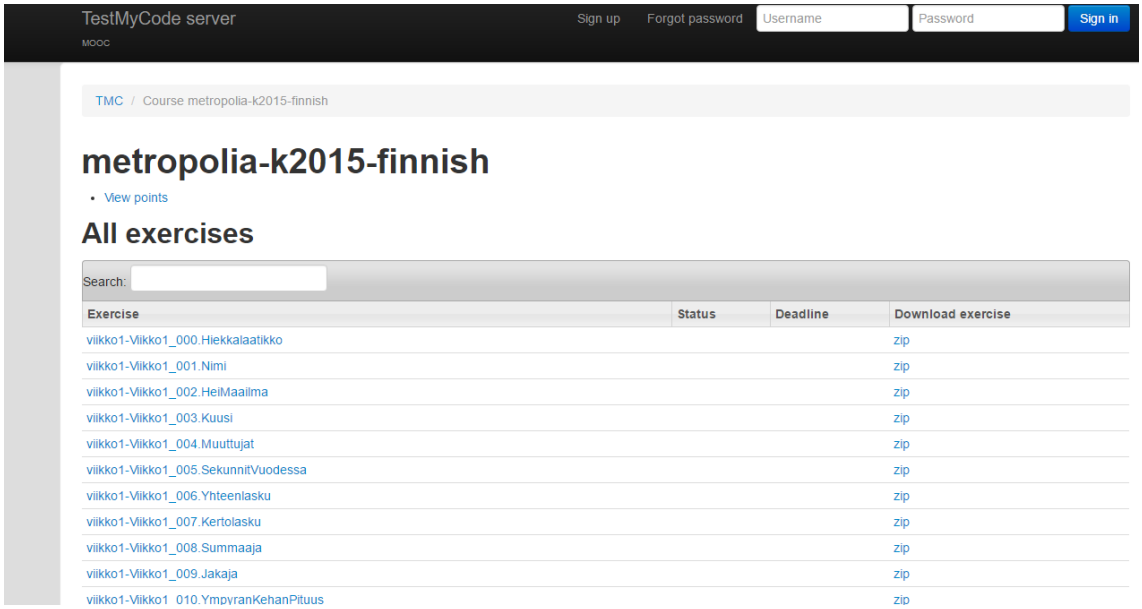
Näiden kysymysten osalta päätettiin yhdessä kurssin järjestäjien kanssa selvittää muutamat eri ratkaisuvaihtoehdot ja valita niiden väliltä.

3 Ratkaisuvaihtoehdot

3.1 TestMyCode

3.1.1 Esittely

TestMyCode (TMC) on alun perin Helsingin yliopiston Tietojenkäsittelytieteen laitoksella kehittämä peruskurssien tehtävien tarkistus- ja palautusjärjestelmä. Sovellus on tarkoitettu muun muassa MOOC (Massive Open Online Course) -verkkokurssien alustaksi. [4.] Sovellus on avointa lähdekoodia ja lisensoitu GPLv2-lisenssillä. Ohjelmiston osat on toteutettu pääosin käyttäen Ruby-ohjelmointikieltä ja Ruby On Rails -ohjelmistokehystä. Osia järjestelmästä on kuitenkin toteutettu Java-ohjelmointikielellä. [6.]



Kuva 2. TestMyCode-järjestelmän kurssinäkymä.

Ympäristössä ideana on, että opiskelija näkee järjestelmästä tehtävänsä (kuva 2). Tämän jälkeen opiskelija lataa tehtäväpohjan (jos on) ja toteuttaa tehtävät käyttäen valitsemaansa kehitystyökalua, esimerkiksi integroitua kehitysympäristöä (IDE) käyttäen. Kun tehtävä on valmis, opiskelija palauttaa toteuttamansa tehtävän joko käyttäen TestMyCoden internetkäyttöliittymää tai käyttäen valmiita liitännäisiä, joita on tarjolla yleisimpiin kehitysympäristöihin. TestMyCode ajaa tämän jälkeen palautettua koodia vastaan kurssinpitäjien ennalta määrittämät testit ja antaa opiskelijalle suoraan pa-

lautetta tehtävän suorituksesta. Jos tehtävä on suoritettu oikein, näkyy suoritusmerkintä heti opiskelijalla ja opettajalla tehtävälistauksessa.

TestMyCode on käytössä Suomessa esimerkiksi Aalto-yliopistolla ja Helsingin yliopistolla Tietojenkäsittelytieteen laitoksella. Lisäksi työkalua on käytetty MOOC-kursseilla, ja useita ammattikorkeakoulujakin on testannut sen käyttöä opetuksessa. [5.]

3.1.2 Testiympäristön käyttöönotto ja havainnot

TestMyCodelle on kirjoitettu hyvät asennusohjeet Ubuntu 14.04 -käyttöjärjestelmälle. Ongelmaksi testiympäristöä asentaessa muodostui kuitenkin, ettei testipalvelimella ollut käytössä Ubuntusta juuri 14.04-versio, vaan vanhempi pidempään tuettu 12.04-versio. Testiympäristöä asentaessa menikin huomattavasti oletettua pidempään, ja tästä syystä testaus jäi vähemmälle aikataulun painaessa päälle. Muilta osin asennusohje oli hyvä ja sisälsi tarvittavat välivaiheet ympäristön pystyttämiseksi. [6.]

Asennuksen ongelmaksi muodostuivat muuttuneet asennuspaketit Ubuntun versioiden välillä. Joidenkin pakettien kohdalla ongelma oli ainoastaan pakettien muuttuneet nimet, mutta joidenkin pakettien kanssa ongelmaksi muodostui, ettei Ubuntu suoraan tarjonnut riittävän uusia versioita tai kyseisiä paketteja ylipäätänsä. Tämä lisäsi tarvetta etsiä kolmansien osapuolien tarjoamia asennuspaketteja paikkaamaan puuttuvat paketit. Tämä luonnollisesti vaikutti aikatauluun. Voidaankin siis sanoa, että testiasennus olisi kannattanut asentaa Ubuntun 14.04-version päälle esimerkiksi virtuaalikoneita hyödyksi käyttäen.

Kuva 3. Helsingin yliopiston TestMyCode-järjestelmän responsiivinen näkymä.

Järjestelmän käyttöliittymä on pelkistetyn selkeä ja responsiivinen (kuva 3), mikä tarkoittaa, että käyttöliittymä mukautuu käytettävän laitteen näytön koon mukaan. Etusivulla käyttäjällä on mahdollisuus valita organisaatio, jonka kurssille on osallistumassa. Avaamalla organisaation saa esille kurssit, joille organisaatiolla on tehtäviä asetettuna, ja kurssin avaamalla näkee tehtävät. Tehtävälistasta on mahdollista ladata tehtävän pohja oikealta valitsemalla ”zip”-linkin, ja tehtävän voi palauttaa avaamalla tehtävän valitsemalla sen listasta.

Kun päästiin lopulta testaamaan itse järjestelmää, havaittiin, että ympäristö tekee juurikin sen, mitä lupaa: käyttöliittymä on helppokäyttöinen, tehtävien palautus on yksinkertaista ja järjestelmä keventää opettajien tehtävientarkistustaakkaa merkittävästi. Järjestelmä myös tukee Java-ohjelmointikieltä, joka oli vaatimus järjestelmältä, ja soveltuu hyvin ohjelmoinnin peruskurssien toteutukseen yksinkertaisuutensa vuoksi. Aikataulupaineiden takia kehitysympäristöjen liitännäiset TestMyCodeen jäivät testaamatta, mutta ne oletettiin toimiviksi. Lisäksi näiden käytöstä oli joillakin projektiin osallistuneilla opettajilla kokemusta.

3.1.3 Vaihtoehdon vahvuudet ja heikkoudet

TestMyCoden ehdoton vahvuus on sen helppokäyttöisyys. Järjestelmä on suunniteltu keventämään etenkin opettajien työsarkaa automaattisten testien osalta. Järjestelmä on suunniteltu peruskurssien toteutuksiin, joskin myös haastavampia tehtäviä on tietyissä rajoissa mahdollista toteuttaa. Se tukee myös kurssille valittua ohjelmointikieltä.

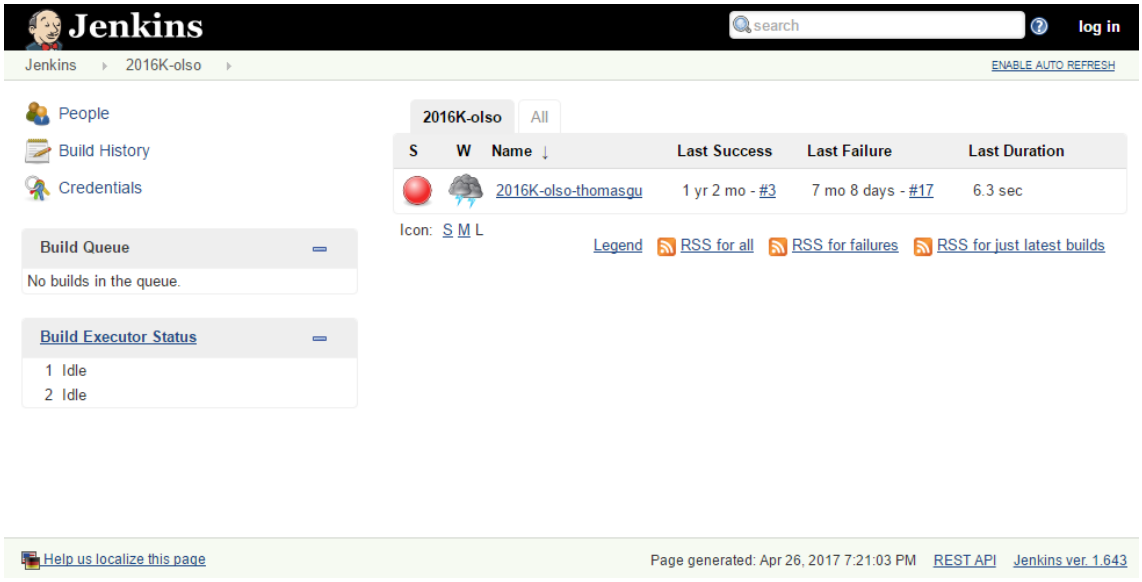
Järjestelmän heikkoutena on sen muunneltavuus ja jossain määrin harvinaisuus. Järjestelmä on käytössä pääosin Suomessa parilla yliopistolla, joten tuen saanti kohdatessa ongelmia jää kysymysmerkiksi. Lisäksi kyseessä on erittäin integroitu järjestelmä, jossa muutoksien tekeminen on aina iso työ. Järjestelmä on myös toteutettu käyttäen Ruby-ohjelmointikieltä, josta ei ollut ennestään kokemusta. Uuden ohjelmointikielen opettelu muutosten toteutusten aikana on iso riski.

Lisäksi heikkoudeksi voidaan laskea työelämälähtöisyys-aspektin puuttuminen ilman muokkauksia. Järjestelmä ei vastaa käytännössä mitään ohjelmistokehityksen prosesseja työelämässä, vaan järjestelmä on hyvin samankaltainen kuin olemassa olevat Moodle- ja Tuubi -oppimisympäristöt pienin muokkauksin ohjelmointikursseja silmällä pitäen. Järjestelmä on siis vahvoilla sellaisenaan, mutta heikoilla, jos toiminnallisuuteen halutaan muutoksia.

3.2 Jenkins-pohjainen ympäristötoteutus

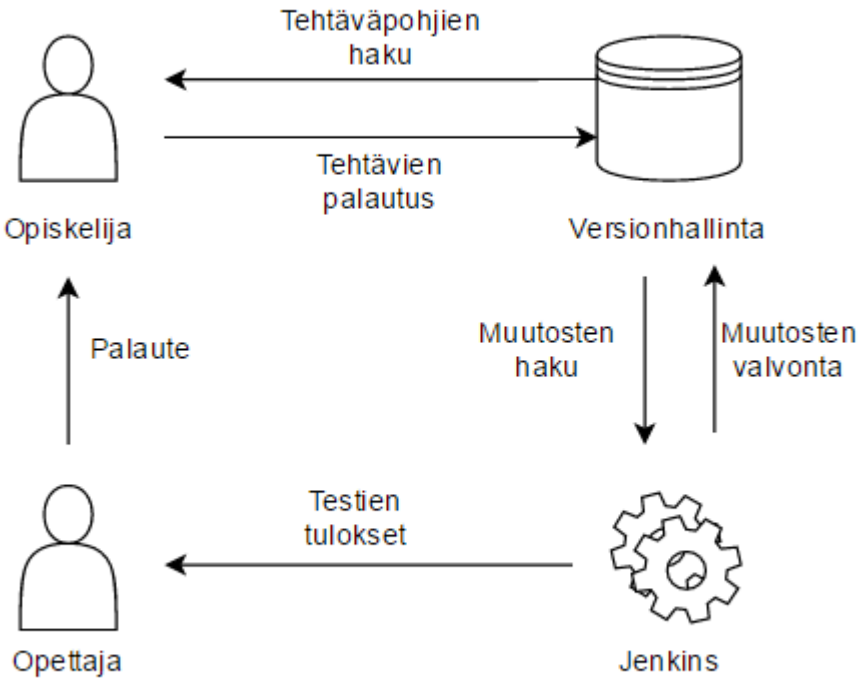
3.2.1 Esittely

Jenkins on avoimen lähdekoodin alun perin automaattisiin koonteihin tarkoitettu palvelinsovellus. Yksinkertaisimmillaan sovellusta käytetään esimerkiksi yhdistettynä johonkin versionhallintaan: Jenkins havaitessaan koodimuutoksen versionhallinnassa (commit) automaattisesti käynnistyy ja kääntää ohjelmakoodin, ajaa ohjelmakoodin mukana toimitetut yksikkötestit, ajaa koodianalytiikkatyökalut ja muodostaa tuloksista koonnin. Tämän jälkeen koonnissa muodostunut paketti voidaan automaattisesti siirtää säiliöön (repository) ja/tai viedä esimerkiksi testipalvelimelle (deploy).



Kuva 4. Jenkins-sovelluksen selainkäyttöliittymä.

Jenkinsille löytyy lukemattomia valmiita lisäosia, jotka lisäävät sen toimintoja ja kykyjä. Jenkins on hyvin yleisesti käytetty työkalu jatkuvien koontien tekemiseen [8]. Jenkins on toteutettu Javalla, Servlet-palvelintekniikkaa käyttäen ja sitä hallitaan selainkäyttöliittymällä (kuva 4). [7.]



Kuva 5. Jenkins-pohjaisen toteutuksen toimintamalli.

Ideana tässä ympäristötoteutuksessa on käyttää Jenkinsiä tehtävien tarkistukseen ja palautteen antamiseen, ja itse tehtävien palautus tapahtuisi käyttäen jotakin versionhallintaa. Ratkaisun idea perustuu juuri Jenkinsin tuomiin mahdollisuuksiin ja joustavuuteen: ratkaisu on helposti muokattavissa eri käyttötilanteisiin jopa kirjoittamatta riviäkään koodia. Idean mukaisesti opiskelijalla on oma versionhallinnan säiliö, johon opiskelija hakee tehtäväpohjat kurssin yhteisestä tietovarastosta. Opiskelija tämän jälkeen toteuttaa tehtävät ja lähettää ne versionhallinnan palvelimelle. Jenkins havaitsee muutokset versionhallinnassa ja käynnistää tehtäväpohjien mukana toimitettujen yksikkötestien ajon toimitettua tehtäväkoodia vasten. Tämän jälkeen Jenkins julkistaa testien tulokset, joiden perusteella opiskelija voi korjata koodiaan, ja opettaja voi pisteyttää tehtävän. Malli (kuva 5) ei ota kantaa käytettäviin tekniikoihin, vaan tekniikat voidaan päättää toteutusvaiheessa.

3.2.2 Testiympäristön käyttöönotto ja havainnot

Jenkinsin asennus tapahtui nopeasti ja helposti. Yleisimmille käyttöjärjestelmille löytyy valmiit ohjelmistopaketit, joiden avulla käyttöönotto tapahtui minuuteissa. Asennuksen jälkeen Jenkinsiin asennettiin tarvittavat liitännäiset suoraan verkkohallintaliittymän kautta. Testiympäristön tapauksessa tarvittavat liitännäiset olivat tuki JUnit-testeille, Git-versionhallinnalle ja Gradle-koontikielelle. [9.]

Testiympäristöä varten tehtiin yksinkertainen koonti käyttäen valmista Gradle-esimerkkiä ja Java-ohjelmakoodia, joka ainoastaan tulostaa tekstin ”Hello World!”. Koodi siirrettiin Git-versionhallintaan entuudestaan olemassa olevalle palvelimelle. Tämän jälkeen Jenkinsiin konfiguroitiin yksinkertainen työ (englanniksi Job), joka haki Git:stä muuttuneet ohjelmakoodit, ajoi Gradle-esimerkin koodia vasten ja julkaisi Gradle-esimerkin muodostaman testiraportin.

Kaikki tämä onnistui suhteellisen nopeasti ja oli nopeasti testattavissa. Tietenkään esimerkiksi Gradle-koonti ei vastannut oikeasti tarvittavaa, jos ympäristö valitaan, mutta antoi esimerkin, miten järjestelmä voisi toimia. Tämän lisäksi lopulliseen toteutukseen olisi mietittävä tarkemmin, miten versionhallinnan tehtävärakenne toteutetaan.

3.2.3 Vaihtoehdon vahvuudet ja heikkoudet

Vaihtoehdon ehdoton vahvuus on muunneltavuus. Kaikki osat ovat vaihdettavissa helposti, ja käytettävät tekniikat voi valita vapaasti. Esimerkiksi käytettävä versionhallinta voidaan helposti vaihtaa. Lisäksi vaihtoehdon kanssa pääsee pitkälti ilman skriptejä kummallisempaa koodia, ja tästä johtuen toteutus on hyvin ylläpidettävä. Myös tehtävien tulokset ovat luettavissa suhteellisen helposti.

Heikkoutena nousee esiin mahdollinen oppimiskynnys: kurssi on jatkokurssi peruskurssille eikä opiskelijoilla välttämättä ole osaamista versionhallinnasta etukäteen. Miten varmistetaan, että kaikki opiskelijat saavat palautettua tehtävänsä? Lisäksi heikkoudeksi voidaan nostaa epäselvyydet Jenkinsin skaalautuvuudesta, jos opiskelijoita on paljon, tai Gradlen skaalautuvuudesta, jos tehtäviä on paljon.

Vaihtoehdon puolesta puhuu työelämälähtöisyys: Jenkins ja erilaiset versionhallinnat ovat hyvin yleisessä käytössä ohjelmistoyrityksissä kautta maailman. Tämä toisi opiskelijoille käytännön kokemusta versionhallinnasta jo peruskursseista lähtien.

3.3 Vaihtoehtojen vertailu ja valinta

Rajallisen ajan vuoksi vertailu tapahtui kahden potentiaalisen vaihtoehdon välillä: TestMyCode- ja Jenkins-pohjaiset ratkaisut esiteltiin. Koska molempien vaihtoehtojen lähestymistapa ongelman ratkaisuun oli hyvin erilainen, päätös tehtiin ennemmin lähestymistapojen perusteella kuin itse ratkaisujen teknisillä perusteilla. Pohjimmaiseksi kysymykseksi muodostui kysymys: ”Pitäisikö oppimisympäristön käytön olla oppikokemus jo itsessään työelämää silmällä pitäen vai pitäisikö työkalun olla mahdollisimman yksinkertainen käyttää?”

On päivänselvää, että TestMyCode-ratkaisu olisi yksinkertaisin ratkaisu. Asennusvaikeuksista huolimatta kyseessä olisi valmis nopeasti käyttöönotettu ohjelmisto, joka tekisi kaiken tarvittavan – mutta ei mitään muuta. Jenkins-pohjainen ratkaisu on enemmän kustomointia vaativa, joskin helpommin asennettava. Loppukädessä Jenkins-pohjainen ratkaisu vastasi parhaiten alkuperäisiä vaatimuksia, pääosin työelämälähtöisyyden osalta.

Päätös oli yksimielinen: kyseisen kurssin oppimisympäristöä lähdettäisiin rakentamaan Jenkins-pohjaisen ratkaisun ympärille. Helppokäyttöisyyden varmistamiseksi hajautettu Git-versionhallinta vaihdettiin Subversion (SVN) -versionhallintaan, joka perustuu selkeämmin omaksuttavaan palvelinpohjaiseen malliin. Versionhallinnan valinta myös vastasi paremmin vanhan opetussuunnitelman vastaavan kurssin sisältöä, sillä vanhan opetussuunnitelman mukaisessa kurssissa versionhallinta opetettiin myös aluksi SVN-versionhallintaa käyttäen. Lisäksi päätettiin, että opiskelijoille on laadittava selkeä ohje tehtävien lähetyksestä versionhallintaan sotkujen välttämiseksi. Vaarana nähtiin, että opiskelijat sijoittaisivat tehtävänsä versionhallintaan ihan omiin kansiorakenteisiinsa, jolloin Jenkins ei pääse suorittamaan testejään oikein.

Lisäksi päätettiin, ettei TestMyCode-testiympäristöä poisteta, vaan se jätettäisiin odottamaan mahdollista käyttöä muilla kursseilla. TestMyCode-ympäristö koettiin hyödylliseksi esimerkiksi mahdollisten verkkokurssien järjestämiseen. TestMyCode-

ympäristöön ei kuitenkaan tämän opinnäytetyön puitteissa tehtäisi enää mitään, vaan painopiste siirtyy puhtaasti Jenkins-pohjaisen ratkaisun toteutukseen.

4 Toteutus

4.1 Lähtötilanne

Vaikka toteutettavaksi valittiinkin Jenkins-pohjainen ratkaisu, ei ratkaisua tarvinnut toteuttaa täysin tyhjästä: Osia toteutuksesta voitiin pienin muokkauksin uudelleenkäyttää vanhoista oppimisympäristötoteutuksista usealle aikaisemmalle kurssille. Näitä olivat esimerkiksi Ohjelmistojen testaus ja hallinta -kurssin sekä Ohjelmistotuotantoprojektin ympäristöt, joista voitiin uudelleenkäyttää osittain versionhallinnan varaston (repository) luontiskriptit, Jenkins-ympäristöä sekä palvelinympäristö. [10.]

Ohjelmistojen testaus ja hallinta -kurssilla opetetaan versionhallinnan ja koontien lisäksi nimensä mukaisesti testausta [10]. Kyseisellä kurssilla on käytössä jokaiselle opiskelijalle oma tietovarasto Subversion-versionhallinnassa, johon suurin osa tehtävistä palautetaan. Näille ei kuitenkaan ajeta automaattisia testejä, vaan tehtäville oli Tuubissa teksti-tyyppinen palautuskenttä, johon palautettiin opiskelijan versionhallinnan osoite (URL). Opettaja käy siis manuaalisesti tarkistamassa ko. osoitteesta tehtävien suorituksen, ja pisteyttämässä tehtävät. Lisäksi kurssilla oli yksittäisillä oppitunneilla käytössä Git-versionhallinta toteutettuna GitLab-ohjelmistoa hyödyntäen sekä Jenkins-ympäristö. Jenkins oli käytössä lähinnä opiskelijoille pintaraapaisuksi koonneista, jossa opiskelijoiden piti toteuttaa koonti olemassa olevalle koodille. Tässäkään siis ei automaattisia testejä suoranaisesti ollut toteutettu.

Ohjelmistotuotantoprojektissa käytettävät ympäristöt mukautettiin kunkin projektin mukaan [10]. Osalle projekteista tarjottiin ainoastaan projektinhallinta, kun taas osalla projekteista oli käytössä täysi ohjelmistotuotannon ympäristö sisältäen testipalvelimet, versionhallinnan, jatkuvat koonnit ja automaattiset testipalvelimelle viennit. Projekteissa käytetyt ohjelmointikielet ja tekniikat vaihtelevat, joten erilaisia koonteja löytyy uudelleenkäytettäväksi ja mahdollisesti hyödynnettäväksi tässä ja mahdollisissa myöhemmissä toteutuksissa.

4.2 Käytännön toteutus

4.2.1 Versionhallinta

Kuten aiemmin on todettu, sovittiin, että versionhallintana käytetään Subversionia (SVN). Subversion on jo valmiiksi käytössä Ohjelmistojen testaus ja hallinta -kurssilla, joten versionhallinnan osalta löytyi jo toimiva ympäristö. Ympäristöä haluttiin kuitenkin muokata uutta mallia varten, jossa opiskelijalla olisi yksi varasto kaikille kursseille kurssikohtaisten varastojen sijasta. Lisäksi haluttiin luopua vuosi ja lukukausi -yhdistelmästä versionhallinnan URL-osoitteissa, ja käyttää puhtaasti opiskelijan käyttäjänimeä ainoana tunnistavana tekijänä. Tämän tarkoituksena oli selkeyttää ja helpottaa versionhallinnan käyttöä. Lisäksi sovittiin, että kursseille tehdään kurssikohtaiset varastot, joita käytetään kurssin kaikissa toteutuksissa.

```
printf ' - luodaan hakemistoja\n'
mkdir --parents $APACHE_SVN_REPO_CONF_DIR;
mkdir --parents $REPO_DIR;

printf ' - luodaan tietovarasto\n'
svnadmin create $REPO_DIR;

chown -R www-data:www-data $SVN_REPO_DIR/*;
chmod -R 0770 $SVN_REPO_DIR/*;

printf ' - määritetään Apachea\n'
```

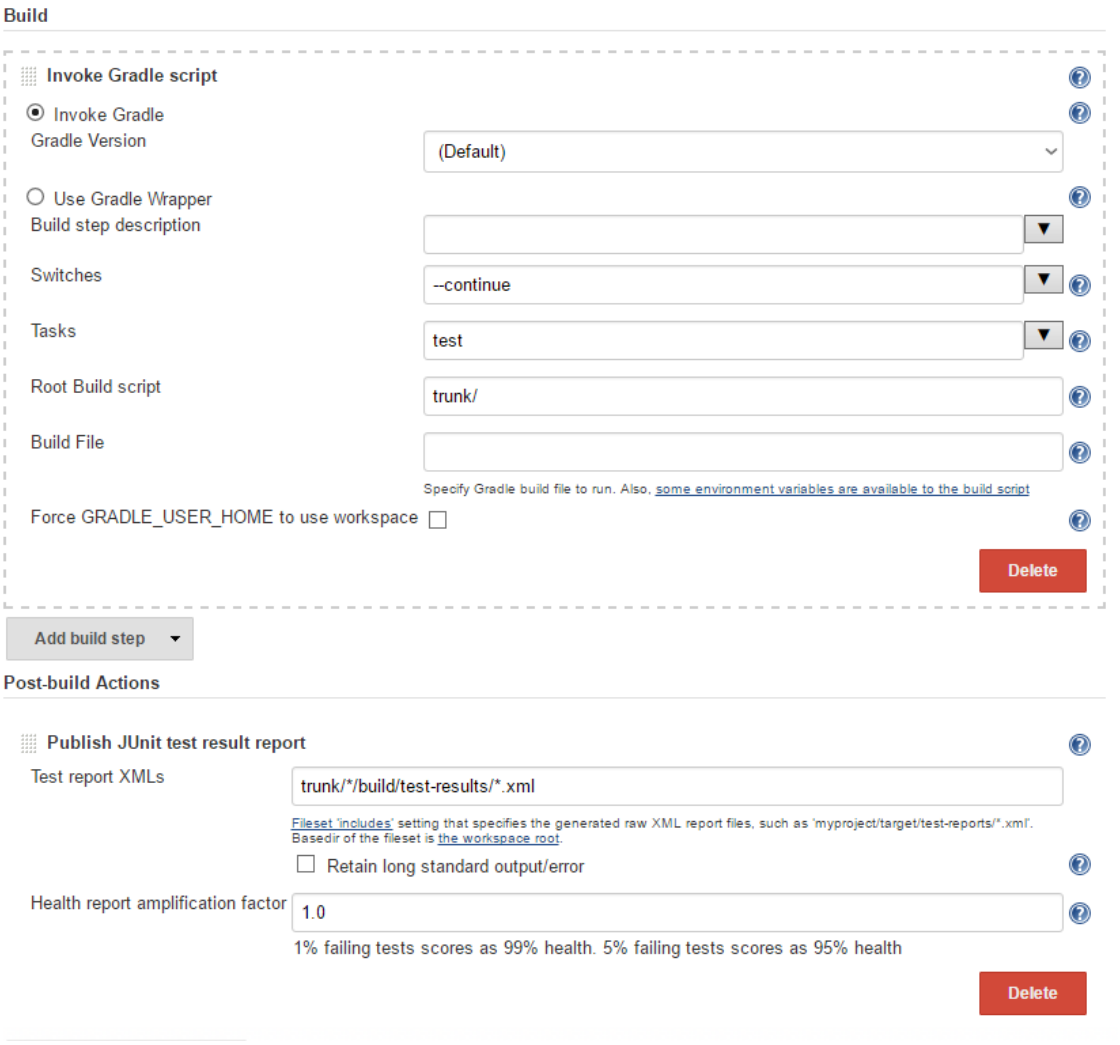
Esimerkkikoodi 1. Ote Subversion-tietovarastojen Bash-luontiskriptistä.

Olemassa olevassa järjestelmässä vanhaa kurssia varten oli jo toteutettu Subversion-versionhallinta käyttäen Apachea. Versionhallintaa pystyi käyttämään HTTP- tai HTTPS -yhteyksien yli. Lisäksi tarjottiin yksinkertainen web-käyttöliittymä versionhallinnassa olevien tiedostojen selaamiseen. Nämä osat voitiin käyttää suoraan hyödyksi. Lisäksi aiempaa kurssia varten oli toteutettu joukko Bash-skrpitejä Subversion-varastojen luontiin. Nämä skriptit annettiin olla sellaisenaan jatkokäyttöä varten, mutta skriptejä käytettiin pohjana käyttäjäkohtaisten varastojen luontia varten (esimerkkikoodi 1). Tarvittavat muutokset koskivat lähinnä kansiopolkuja, Apachen asetuksia kansiopolkujen muutoksista johtuen sekä skriptin tarvitsemien parametrien käsittelyä. Versionhallinnan autentikointijärjestelmänä käytetiin LDAP-palvelua, kuten aikaisemmassakin toteutuksessa.

Skriptimuutosta testatessa ilmeni, että skriptissä tiedosto-oikeuksien käsittelyssä sattui virhe. Virheen takia skripti kävi muuntamassa kaikkien yläkansioidenkin oikeudet Apachelle ja Subversionille sopiviksi. Mikä pahinta, ilmeni virhe vasta palvelimella jossa versionhallintaa käytetään tuotantokäytössä. Skripti kävi siis esimerkiksi muokkaamassa GitLab-asennuksen tiedosto-oikeuksia ja rikkoi asennuksen. Itse skriptin muutos oli yksinkertainen toteuttaa, mutta virheen takia tarvitut tiedostojen oikeuksien palautukset nostivat työsarkaa työn tässä vaiheessa merkittävästi.

4.2.2 Jenkins

Jenkinsin asennus toteutettiin Jenkinsin asennusohjeiden mukaisesti [9]. Apache asetettiin käänteiseksi välityspalvelimeksi (reverse proxy) Jenkinsin eteen vastaamaan verkosta saapuviin pyyntöihin. Asennuksen ainoa haaste oli ns. ”reverse proxy” -asetusten asettaminen oikein, koska Jenkinsin vaatimukset käänteiselle välityspalvelimelle olivat vastikään muuttuneet eivätkä verkosta löytyneet asennusohjeet vastanneet uusia vaatimuksia. Lopulta Nginx-palvelinsovelluksen asetuksia soveltamalla Apacheen saatiin Jenkinsin asennus toimimaan välityspalvelimen kautta.



Kuva 6. Kuvankaappaus Jenkins-töiden konfiguraatiosta.

Jenkinsin konfiguroinnissa käytettiin hyväksi testiympäristöä varten tehtyä asennusta, ja tarvittavat liitännäiset asennettiin testiympäristössä havaittujen tarpeiden mukaisesti. Opiskelijakohtaisten testitöiden ideana oli alun perin, että ne toteutettaisiin skriptillä valmista pohjatyötä käyttäen. Aikataulusyistä skriptiä ei kuitenkaan saatu valmiiksi, vaan lopullisessa ratkaisussa työt kopioitiin pohjatyöstä ja muokattiin tarvittavilta osin käsin. Tämäkään ei osoittautunut tarpeettoman monimutkaiseksi toimenpiteeksi; pohjatyöhön tarvitsi vain kahteen kohtaan asettaa opiskelijan käyttäjätunnus. Muutoin pohjassa käytettiin testiympäristössä jo testattuja yksikkötestien julkistamistyökaluja (kuva 6).

Versionhallinnan osalta huomattiin, ettei testejä voi eikä kannata ajaa ajastetusti opiskelijoiden tietovarastoja vasten suorituskykysyistä. Testiympäristössä käytettiin alun

perin versionhallinta Git:iä GitLab:n kautta, ja tämä ratkaisu tarjosi mahdollisuuden niin sanottuihin ”build hook”:ihin, jota kautta testaus voitiin kutsua heti tietovaraston muutoksien yhteydessä. Subversionista tällaista toimintoa ei kuitenkaan löytynyt, vaan tämä jouduttiin ratkaisemaan toisella tavalla. Ratkaisuksi lopulta valittiin niin kutsuttu ”poll”-tyyppinen ratkaisu, jossa Jenkins käynnistyy väliajoin tarkistamaan, onko versionhallintaan ilmestynyt muutoksia. Tämä havaittiin myös palvelimen suorituskyvyn kannalta riittäväksi kompromissiksi; palvelin pysyi hyvin käytettävänä kyselyiden lisäämistä kuormista huolimatta.

4.2.3 Gradle-koonti

Koonnin tekoon eli käytännössä ensivaiheessa testien ajamiseen valittiin käytettäväksi tekniikaksi testiympäristön mukaisesti Gradle. Gradle on käytäntöihin perustuva koontityökalu, jossa kaiken koko koonnin prosessien määrittämisen sijasta koontiedostossa lähinnä kerrotaan käytännöistä poikkeamisesta. Jos poikkeuksista ei mainita koontitiedostossa, oletus on, että nämä asiat ovat toteutettu käytänteiden mukaisesti. Gradlessä koontietiedoston muodostamiseen käytetään Java-pohjaista Groovy-skriptikieltä. [11.]

```
include
":OlsoTehtSarja1",":OlsoTehtSarja2",":OlsoTehtSarja3",":OlsoTehtSarja4
",":OlsoTehtSarja5",":OlsoTehtSarja6",":OlsoTehtSarja7",":OlsoTehtSarj
a8",":OlsoTehtSarja9",":OlsoTehtSarja10",":OlsoTehtSarja11"
```

Esimerkkikoodi 2. Aliprojektien (tehtävien) esittely settings.gradle-tiedostossa.

Tehtävien pohjina käytettiin Eclipse-kehitysympäristön projektirakennetta. Tämä sopi hyvin Gradle-koontien tekoon, sillä Eclipse käyttää pitkälti valitsevien Java-käytäntöjen mukaisia kansiorakenteita. Tältä osin koontiin ei tarvinnut tehdä isoja muutoksia. Ongelmaksi kuitenkin muodostui, että tehtävät oli toteutettu erillisinä Eclipse-projekteina, Gradle-koonnin olettaessa niiden olevan yksittäinen hakemisto. Tämä ratkaistiin aliprojekteja käyttäen, eli Gradleen toteutettiin niin kutsuttu vale-pääprojekti (dummy), jonka aliprojekteja yksittäiset tehtävät olivat (esimerkkikoodi 2).

```
subprojects {
  apply plugin: "java"
  compileJava.options.encoding = 'UTF-8'
  tasks.withType(JavaCompile) {
        options.encoding = 'UTF-8'
  }

sourceSets {
  main {
                java {
                        srcDir 'src'
                }
        }
        test {
                java {
                        srcDir 'tests'
                }
        }
        }

  repositories {
      mavenCentral()
  }

  dependencies {
    testCompile "junit:junit:4.11"
}
}
```

Koodiesimerkki 3. Ote Gradlen koontitiedostosta build.gradle.

Koontien itsensä määrittelyssä ei isoja muutoksia tarvittu, vaan tarvittavat toiminnallisuudet saatiin pitkälti Java-liitännäisen kautta. Tarvittavat muutokset koskivat muun muassa Eclipsen hiukan erilaista kansiorakennetta, jonka johdosta Gradle-koontietiedostoon jouduttiin määrittämään lähdekoodi ja testi-kansioiden polut (Koodiesimerkki 3). Tämän lisäksi Gradle-koontitiedostoon määritettiin yksikkötestejä varten vaatimukseksi Junit-kirjasto sekä tarvittavien kirjastojen hakua varten käytettävä tietovarasto, tässä tapauksessa MavenCentral. MavenCentral on Java-ohjelmointikielen kanssa useasti käytettävä tietovarasto, joka on alun perin kehitetty Maven-koontityökalua varten, mutta tuettu useamman koontityökalun toimesta.

5 Kokemukset käytöstä

Ensimmäistä kurssitoteutusta varten valmistui erinäisten syiden takia vain versionhallinta-osuus, mikä sisältää itse versionhallinnan toteutuksen, versionhallinnan kansiorakenteen ja ohjeiden toteutuksen. Ratkaisussa ei ilmennyt isompia ongelmia, vaan versionhallinnan osalta ratkaisu toimi lähes moitteettomasti. Ongelmia aiheuttivat lähinnä oppilaitoksen muuttuneet kirjautumispalvelut, joiden kautta opiskelijan käyttäjätunnuksen selvittäminen vaikeutui huomattavasti. Ongelmat kulminoituivat usein opiskelijoiden puuttuviin versionhallinnan tietovarastoihin, sillä järjestelmistä saatu käyttäjätieto ei ollut luotettava. Tämä myöhemmin ratkaistiin uusilla palveluilla, joista opettajat pystyivät tulostamaan listan kurssille ilmoittautuneista opiskelijoista sisältäen kyseisen opiskelijan käyttäjätunnuksen.

Seuraavaa toteutusta varten valmistui myös Jenkins-pohjainen automaattisten testien ajoympäristö. Käytön aikana tässä havaittiin kuitenkin lukuisia ongelmia, joita testiympäristössä ei havaittu. Yksi merkittävimmistä ongelmista oli käyttäjäoikeuksien hallinta, sillä testeissä oli käytetty kehittäjän omia käyttäjätunnuksia. Tuotantokäytössä näin ei voitu kuitenkaan toimia puuttuvien oikeuksien ja tietoturvan takia. Ongelma ratkaistiin kiertämällä koko ongelma: koska Jenkins-palvelin oli sama palvelin, jolla versionhallintakin sijaitsi, asetettiin Jenkins lukemaan versionhallinnan sisältö suoraan tiedostojärjestelmästä. Tämä ei ollut suoraan tuettu toimintatapa, mutta yhtä kaikki toimiva, ja näin saatiin käyttäjätunnusongelma ratkaistua.

Seuraavaksi ongelmia aiheuttivat lähdekoodit: Java-käytänteiden mukaisesti lähdekoodien oletettiin olevan UTF-8-merkistökoodauksella toteutettuja, mutta myöhemmin huomattiin, että Eclipse käyttää Windows-ympäristössä oletuksena ISO-8859-1-merkistökoodausta. Koska opiskelijat käyttivät erilaisia käyttöjärjestelmiä, osa koodeista oli toteutettu toisella merkistökoodauksella ja osa toisella. Ongelma myös ilmeni vain, jos opiskelija oli käyttänyt esimerkiksi muuttujien nimissä skandinaavisia merkkejä ä, å tai ö. Kyseisen toteutuksen kohdalla sovittiin lopulta käytettäväksi ISO-8859-1-merkistökoodausta, sillä tämä oli valtaosalla opiskelijoita käytössä.

Viimeisen ongelman aiheutti jälleen kerran Java-käytänteet: alkuperäisessä käyttötarkoituksessaan yksikkötesteissä tapahtuneen virheen jälkeen koontitiedoston ajo keskeytetään, eikä seuraavaa projektia enää käännetä tai testata. Tämä ei tietenkään sopinut oppilaitoskäyttöön, jossa kaikki tehtävät tulee tarkistaa. Lopulta Gradlestä löytyi

tarvittava vipu ”--continue” (kuva 6), jolla Gradle saatiin ajamaan loppuun asti mahdollisista virheistä koodissa huolimatta. Kaikki nämä korjaukset eivät kuitenkaan kerineet valmistua toisen toteutuksen loppuun mennessä, vaan tehtävien tarkistus jouduttiin toteuttamaan perinteisin menetelmin.

Myöhemmissä toteutuksissa järjestelmä on toiminut pieniä takkuiluja, esimerkiksi asetustiedostojen kanssa, lukuunottamatta suhteellisen vakaasti.

6 Jatkokehityskohteet

Vaikka järjestelmä saatiin toimivaan tilaan, olisi järjestelmässä paljon jatkokehityksen varaa. Yksi isoimmista puutteista, joka järjestelmään jäi, oli yhtenäisen käyttöliittymän toteutus, josta opiskelijat ja opettajat voisivat helposti tarkistaa tehtävien suorituksen tilanteen ja mahdollisten virheiden laadut. Nykyinen puhtaasti Jenkinsin kautta toteutettu tulosten raportointi palveli lähinnä opettajien tarpeita, eikä Jenkinsin olemassaolosta kerrottu opiskelijoille. Tämä tehtiin muun muassa siksi, että Jenkinsin osalta ei ollut helposti toteutettavissa autentikointia opiskelijoille niin, että opiskelijat näkisivät vain omat tehtävänsä. Nykyisessä järjestelmässä näkee kaikkien kurssille osallistuneiden opiskelijoiden tulokset.

Toinen mahdollinen jatkokehityskohde olisi eri toimintojen automatisointi esimerkiksi skriptejä käyttäen tai parhaimmassa tapauksessa hallintaliittymää hyväksikäyttäen. Esimerkiksi versionhallinnan tietovaraston ja Jenkins-töiden luonnin voisi toteuttaa yhtenäiseen skriptiin. Parhaimmillaan tästä voisi työstää myös hallintaliittymän, josta opettaja voi esimerkiksi opiskelijat listaamalla luoda itse tarvittavat versionhallinnat ja tarkistustyöt. Hallintaliittymään voitaisiin toteuttaa myös esimerkiksi tulosten vientimahdollisuus esimerkiksi Excel-formaatissa.

Kumpikaan kehitysvaihtoehdoista ei olisi mahdottoman vaikeita toteuttaa, koska kaikista tarvittavista komponenteista löytyvät rajapinnat toteutusten tekemiseen. Tästä voisi löytyä esimerkiksi toiselle opiskelijalle insinöörityön mahdollisuus.

7 Loppusanat

Vaikka lopputulos onnistuikin kohtuullisen hyvin, olisi prosessissa ollut paljon parantamisen varaa. Suurin ongelma projektin toteutuksessa oli ajan hallinta: alun perin projektia suunniteltiin tehtävän iltaisin ja viikonloppuisin päivätyön ohella, mutta käytännössä jälkikäteen katsoen tämä oli liian objektiivinen tavoite. Käytännössä työtä edistettiin viikonloppuisin ajan salliessa sekä lomilla. Projekti venyi alun perin suunnitellusta puolen vuoden tai vuoden toteutuksesta lähes kahden vuoden toteutukseksi.

Tilanteet huomioiden loppuratkaisu onnistui kohtalaisen hyvin täyttämään vaatimukset: opiskelijoille saatiin kurssia varten työelämälähtöinen opiskeluympäristö, joka myös helpottaa opettajien työtaakkaa. Vaatimus automaattisista testeistä saatiin toteutettua, mutta toivomuksia hallintaliittymästä tai testitulosten viemisestä arvostelua varten ei saatu ajan puitteissa toteutettua.

Myös raportin kirjoitus jäi erinäisistä syistä johtuen viime hetkeen. Tästä ei tietenkään voi syyttää muuta kuin suunnitelmallisuuden puutetta. Alun perin projekti suunniteltiin toteutettavan joustavasti ajan salliessa. Käytännössä voidaan kuitenkin todeta, ettei tämä toimi. Projekti opetti kuitenkin paljon käytetyistä tekniikoista, joita voidaan hyödyntää työelämässä. Lisäksi projekti tuotti toimivan ympäristön, jota voidaan tarpeen vaatiessa jatkokehittää. Tärkeimpänä opetuksena työ opetti järjestelmällisyyden ja aikataulujen tärkeyden työn ja projektin onnistumisen kannalta.

Lähteet

1 Kaisla, Maija; Kutvonen-Lappi, Titta; Kankaanranta, Marja. 2015. Digitaalinen oppimateriaali koulun arjessa. Jyväskylä: Jyväskylän yliopistopaino. <https://ktl.jyu.fi/julkaisut/julkaisuluettelo/julkaisut/2015/d115.pdf>. Luettu 17.4.2017.

2 Opinto-opas, Tieto- ja viestintätekniikka (Helsinki), Ohjelmistotuotannon opinto-polku. 2014. Verkkodokumentti. Metropolia ammattikorkeakoulu. <http://opinto-opas-ops.metropolia.fi/index.php/fi/16183/fi/70361/TXK14S1H/1388/year/2014>. Luettu 19.4.2017.

3 Oppijan polku 2015. 2015. Verkkodokumentti. Metropolia ammattikorkeakoulu. <https://www.slideshare.net/metropolia_uas/tuotoropiskelijan-opas-2015-48431867>. Luettu 19.4.2017.

4 TestMyCode. 2015. Verkkodokumentti. <http://testmycode.github.io>. Luettu 22.4.2017.

5 Haasio, Ari; Zechner, Minna; Päällysaho, Selina. 2015. Internet, verkkopalvelut ja tietotekniset ratikaisut opetuksessa ja tutkimuksessa. Verkkodokumentti. <https://www.theseus.fi/bitstream/handle/10024/103048/A22.pdf?sequence=1>. Luettu 22.4.2017.

6 Setup. 2016. Verkkodokumentti. <https://github.com/testmycode/tmc-server/blob/master/Installation.md>. Luettu 22.4.2017.

7 Jenkins. 2017. Verkkodokumentti. <https://jenkins.io>. Luettu 23.4.2017.

8 Some statistics on the usage of Jenkins. 2017. Verkkodokumentti. <http://stats.jenkins.io/jenkins-stats/svg/svgs.html>. Luettu 23.4.2017.

9 Installing Jenkins. 2016. Verkkodokumentti. <https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins>. Luettu 23.4.2017.

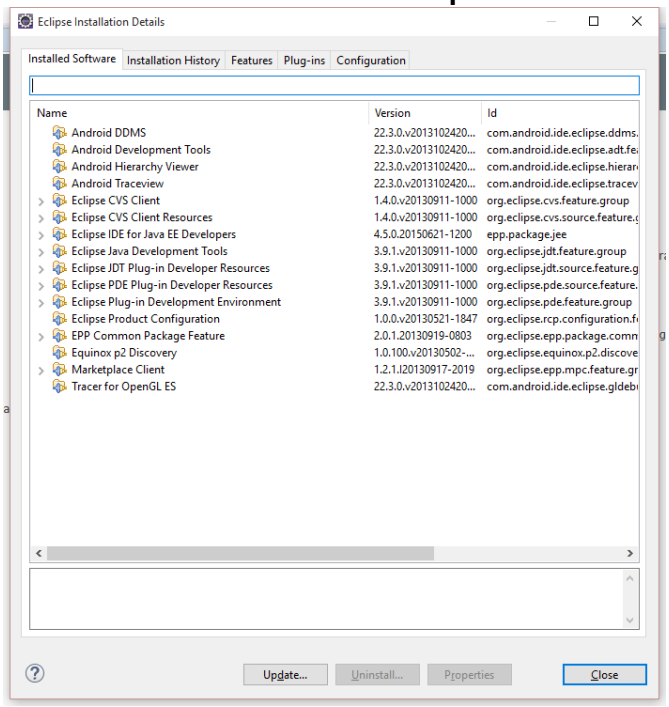
10 Opinto-opas, Tietotekniikka (Helsinki), Ohjelmistotekniikka. 2012. Verkkodokumentti. Metropolia ammattikorkeakoulu. <http://opinto-opas-ops.metropolia.fi/index.php/fi/16183/fi/76/TT12S1H/969/year/2012>. Luettu 24.4.2017.

11 Kasari, Topi. 2012. Jatkuvan integroinnin koontityökalut. Verkkodokumentti. <https://www.theseus.fi/bitstream/handle/10024/41201/Kasari_Topi.pdf?sequence=1>. Luettu 25.4.2017.

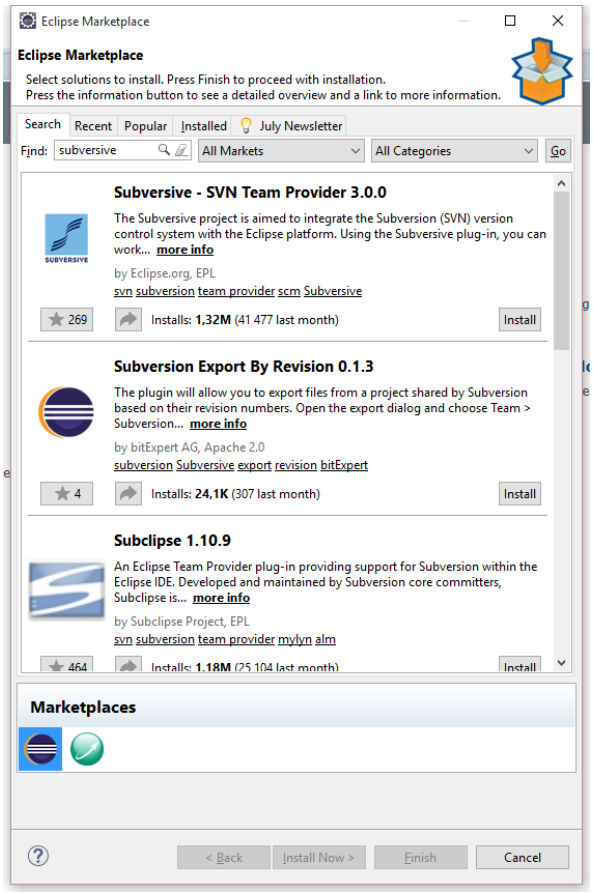
Liite 1. Opiskelijan ohjeistuspohja tehtävien tekoon

Ensimmäisellä käyttökerralla / uudella koneella

1. Jos sinulla ei ole Subversive-pluginia asennettuna, asenna se. Voit tarkistaa onko se asennettuna valitsemalla **Help → Installation details.**

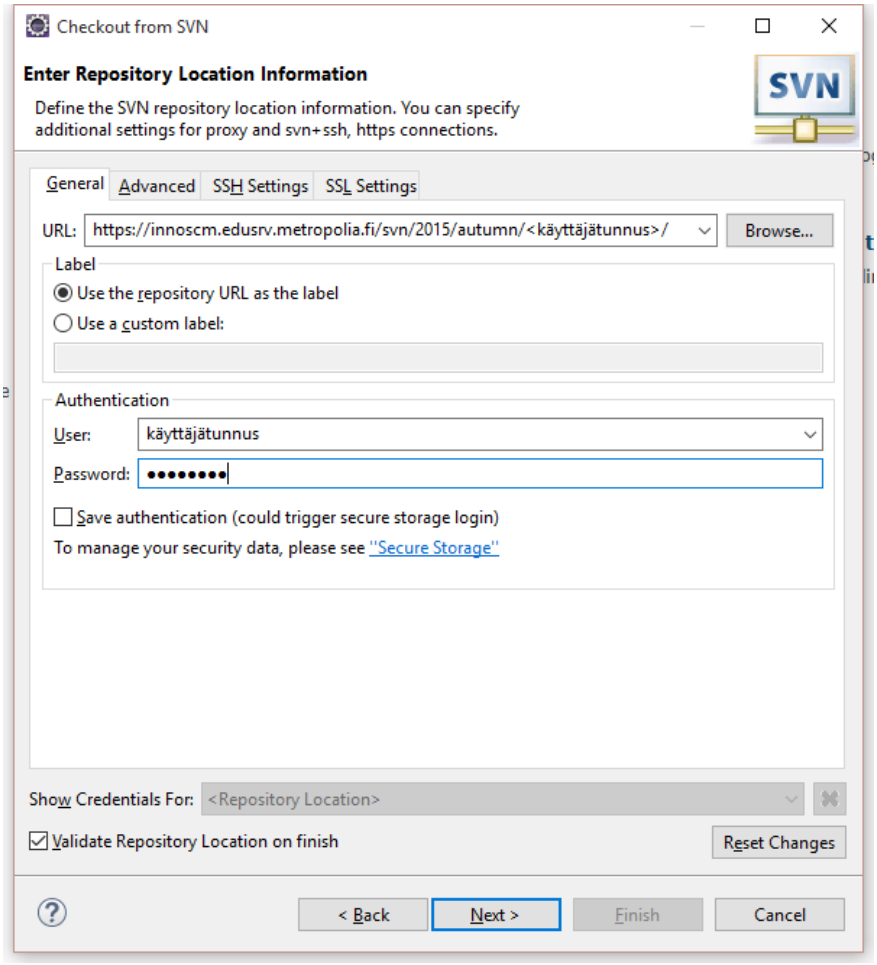


Jos listassa ei ole Subversivea asennettuna (kuten yllä), sulje ikkuna ja valitse **Help → Eclipse Marketplace...**. Hae **”subversive”** ja klikkaa **Install** Subversive-pluginin kohdalla.



Tämän jälleen valitse **Confirm >**, hyväksy lisenssiehdot ja valitse **Finish**.

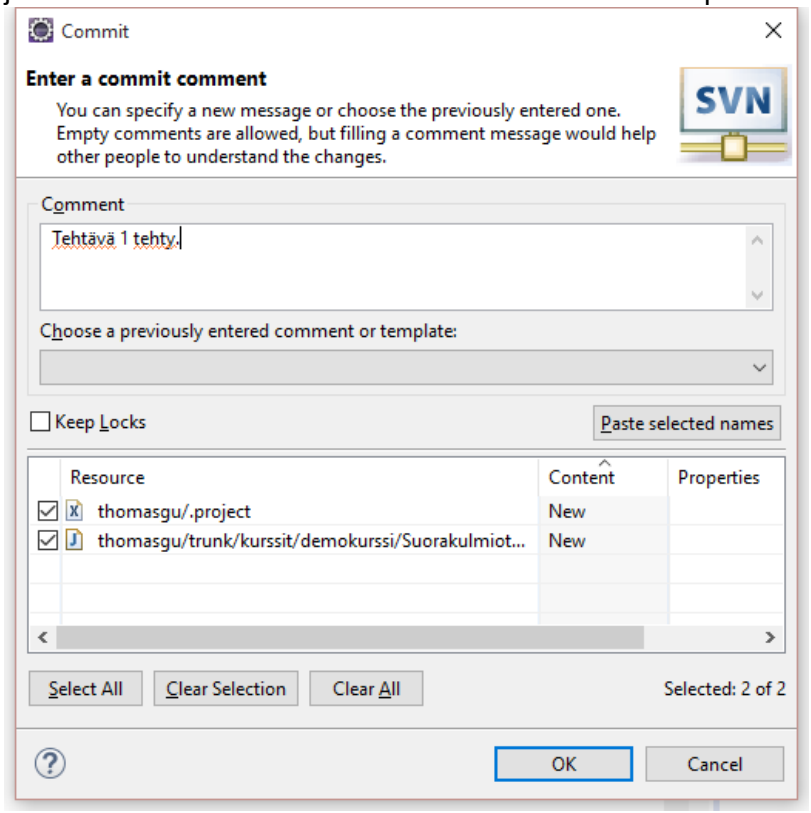
2. Valitse **File → Import**, aukeavasta ikkunasta **SVN → Project from SVN** ja valitse **Next >**
3. Jos sinulta kysytään Subversive Connectoria, **valitse listasta uusin versio** ja paina **Finish**. Paina **Next > → Next > → Hyväksy lisenssiehdot** ja paina **Finish.** Jos sinulta kysytään allekirjoittamattomasta sisällöstä, paina **OK**. Käynnistä tarvittaessa Eclipse uudestaan ja tee **Vaihe 2. uudestaan.**
4. Laita kohtaan URL **https://innoscm.edusrv.metropolia.fi/svn/2015/autumn/<käyttäjätunnus>/** jossa **<käyttäjätunnus> korvataan Metropolian käyttäjätunnuksellasi.** Authentication-kohtaan laita **Metropolian käyttäjätunnuksesi ja salasanasi.** Halutessasi voit valita kohdan **Save authentication.** Lopuksi valitse **Next >.**



5. Jos Eclipse kysyy sertifikaatista, valitse **Trust.**
6. Lopuksi paina **Finish.**
7. Uudesta ikkunasta valitse **Check out as a project with the name specified**. Paina **Finish.**
8. Tämän jälkeen tee vaihe **2 uudestaan.**
9. Valitse **Create a new repository location** ja **Next >.**
10. Laita osoitteeksi tällä kertaa **https://innoscm.edusrv.metropolia.fi/svn/2015/autumn/olso/** ja käyttäjätunnukseksi/salasanaksi **Metropolian tunnuksesi**. Valitse **Next >**.
11. Valitse taas **Check out as a project with the name specified.** Tämä on kurssin yhteinen repository, sen voi nimetä niin, että muistaa olla tekemättä muutoksia tähän projektiin. Lopuksi valitse **Finish**.
12. Ensimmäisellä kerralla **kopioi koko kansiorakenne kurssin yhteisestä repositorysta (”olso”) omaan repositoryysi**. Muutoin kopioi vain tarvitsemasi tiedostot (yksittäisen tehtävän tiedostot).
13. Tee tehtävät!

Kun tehtävä on valmis tai haluat välitallentaa

1. Klikkaa repositoryasi hiiren oikealla ja valitse **Team → Commit...**
2. Valitse alta tiedostot jotka haluat viedä repositoryyn ja laita kommentti-kenttään joku kuvaava komentti mitä teit tiedostoissa. Valitse lopuksi **OK.**



3. Jos et tallentanut käyttäjätunnuksiasi, Eclipse kysyy käyttäjätunnuksiasi. Muutoin muutoksesi on tallennettu.

Kun haluat hakea repositoryn muutokset

1. Klikkaa repositorya hiiren oikealla ja valitse **Team → Update**