Tampereen yliopisto

Tommi Honkanen

AUTOMAATTISEN ARVIOINNIN KÄYTÄNTÖJEN YHTENÄISTÄMINEN KORKEAKOULUYHTEISÖSSÄ

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastajat: Yliopistonlehtori Terhi Kilamo
Yliopisto-opettaja Pia Niemelä
Marraskuu 2020

TIIVISTELMÄ

Tommi Honkanen: Automaattisen arvioinnin käytäntöjen yhtenäistäminen korkeakouluyhteisössä
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-tutkinto-ohjelma
Marraskuu 2020

Ohjelmoinnin opettamisessa yksi suosituimmista työkaluista on automaattinen arviointi. Tätä sovelletaan Tampereen yliopiston Tuni+ -palvelussa useilla ohjelmoinnin kursseilla. Kursseilla automaattisen arvioinnin työkalut ovat kuitenkin muotoutuneet hyvin erilaisiksi alkuperäisestä pohjastaan, ja tiedekunnalla on useita toisistaan teknisesti eroavia kurssitoteutuksia. Tämä vaikeuttaa kurssien hallintaa, työkalujen ylläpitoa ja uuden henkilökunnan työtä kursseilla.

Tässä diplomityössä vastaamme kysymyksiin:"Miten Tuni+:ssa olevien ohjelmointikurssien automaattisia arvioijia voidaan yhtenäistää? Millaisilla työkaluilla ja käytännöillä tätä voidaan edesauttaa?". Kysymyksiin haetaan vastaus konstruktiivisen tapaustutkimuksen avulla rakentamalla uusi työkalu pohjautuen neljään Tampereen yliopistossa opetettavaan ohjelmoinnin kurssiin. Työkalun lähtökohtana on, että se on yhteensopiva Tuni+:n kanssa. Aluksi esittelemme ohjelmoinnin arviointia ja sen automatisointia, sekä sen pedagogisia vaikutuksia. Tämän jälkeen käymme läpi Tuni+:n arkkitehtuurin, ja määrittelemme mitä ominaisuuksia työkalun tulee sisältää. Lisäksi tutkimme jo olemassa olevia komponentteja ohjelmointikursseilta, ja etsimme myös menetelmiä, joilla havaitut ongelmat kurssien yhtenäisyydessä saadaan korjattua, jotta toteutukset olisivat samanlaisempia aiempaan verrattuna.

Työn tuloksena syntyi uusi työkalu, jolla Tuni+:aan voidaan alustaa uusia ohjelmoinnin kursseja. Työkalu myös noudattaa uutta yhtenäistettyä toteutusta, jossa tehtävien arvioinnista vastaa yksi pääskripti, joka hoitaa eri tyyppisten tehtävien arvioinnin. Uusia kursseja ja tehtäviä varten on myös luotu uusi pohjatoteutus Docker-kuville, joka sisältää Tuni+:n kannalta välttämättömät osat ja jonka päälle arviointia suorittavan Docker-säiliön kuva voidaan täydentää. Uusi työkalu, Docker-kuvan pohjatoteutus sekä yhtenäistämiseen käytetyt menetelmät soveltuvat sellaisenaan Tuni+:aan käytettäviksi. Työkalua käyttämällä uusien kurssien toteutukset ovat valmiiksi yhdenmukaisia vanhojen kurssien kanssa, jolloin niiden käyttöönotto ja jatkokehittäminen on helpompaa henkilökunnalle.

Avainsanat: automaattinen arviointi, ohjelmoinnin opetus, koodikloonit

ABSTRACT

Tommi Honkanen: Uniforming the practices of automatic assessment in University setting
Master's thesis
Tampere University
Master's Programme in Computer Science
November 2020

Automatic assessment is one of the most popular tools in programming education. It is utilized on many programming courses on Tampere Universities' learning management system Tuni+. However, the tools used on the courses have become very different from course to course, and the faculty has many courses with different technical implementations. This makes the management of courses, tools and the work of the course personnel harder.

In this thesis we answer the following questions:"How the automatic graders of programming courses in Tuni+ can be uniformed? What kind of tools and practices can be used to achieve this?" The answers for the questions are sought via constructive research by creating a new tool based on four programming courses taught in Tampere University. The basis of the tool is that it is compatible with Tuni+. We start by introducing how programming can be assessed, how the assessment can be automated and its pedagogic effects. After this we examine the architecture of Tuni+, and define what properties the tool must contain. In addition we go through existing components from the programming courses, and look for methods which can be used to uniform the courses' implementations.

As a result of the thesis, we have a new tool which can be used to initialize a new programming course for Tuni+. The tool follows a new uniform implementation where a single main script handles the assessment process of all different exercises of a single course. A new Docker base image has also been created for the use of new courses and for new exercises, which contains all essential components required by Tuni+, and which can be used as a foundation for the actual Docker images used by the grading containers. The new tool, the Docker base image and the methods used in the uniforming can be used in Tuni+. Using the tool for creating new courses ensures that they are uniform with older courses, which makes the deployment and further development of the courses easier.

Keywords: automatic assessment, programming education, code clones

ALKUSANAT

Tämä diplomityö on tehty Tampereen yliopiston informaatioteknologian ja viestinnän tiedekunnan alaisuudessa syyskuusta 2019 marraskuuhun 2020. Työ on tehty yhteistyössä tiedekunnan järjestämien Ohjelmointi 1-3 -kurssien sekä Basic Web Applications -kurssin henkilökunnan kanssa. Tästä joukosta haluan kiittää Mikko Nurmista neuvonnasta Dockerin toiminnan suhteen. Haluan esittää suuret kiitokset lehtori Jyke Savialle, joka auttoi paljon Tuni+:n toiminnan selvittämisessä ja testaamisessa, sekä yliopistonlehtori Terhi Kilamolle työn ohjauksesta ja arvokkaista kommenteista.

Tampereella, 22. marraskuuta 2020

Tommi Honkanen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

OHJELMA- JA ALGORITMILUETTELO

LYHENTEET JA MERKINNÄT

AST	Abstraktit syntaksipuu (engl. Abstract syntax tree)
BWA	Tuni+:ssa opetettava verkko-ohjelmointikurssi (engl. Basic Web Applications)
CD	jatkuva julkaisu (engl. Continuous Deployment)
CI	jatkuva integraatio (engl. Continuous Integration)
GUI	graafinen käyttöliittymä (engl. Graphical User Interface)
HTML	merkintäkieli hypertekstille (engl. Hypertext Markup Language)
LMS	oppimisenhallintajärjestelmä (engl. Learning Management System)
MOOC	Kaikille avoimet verkkokurssit (engl. Massive Open Online Course)
OHJ3	Tuni+:ssa opetettava Ohjelmointi 3: Tekniikat -kurssi
PDG	Ohjelman riippuvuusgraafi (engl. Program Dependency graph)
PoC	idean toteuttamiskelpoisuutta esittelevä vajavainen toteutus (engl. Proof of Concept)
TAU	Tampereen yliopisto (engl. Tampere University)
TUNI	Tampereen korkeakouluyhteisö (engl. Tampere Universities)
UI	käyttöliittymä (engl. User Interface)
URL	verkkosivun osoite (engl. Uniform Resource Locator)
YAML	merkintäkieli (engl. Yeat Another Markup Language)

1. JOHDANTO

Ohjelmoinnin opettaminen on haastava tehtävä. Ohjelmoinnin opetus aloitetaan usein perusteista, kuten ohjelmointikielten perusrakenteista ja ohjelmointiin liittyvistä tehtävistä. Näiden tietojen varassa opiskelijoilla on kyky oppia ohjelmistokehityksen menetelmiä ja ratkaisuja sen yleisiin ongelmiin [1, s. 2-3]. Ohjelmointia oppiakseen opiskelijan täytyy kuitenkin saada ohjelmoida itse erilaisia tehtäviä ja ratkaista hänelle annettuja ongelmia oppimiensa menetelmien avulla . Nykyään ohjelmoinnin kursseille on tyypillistä suuret osallistujamäärät ja hyvin laaja valikoima erilaisia harjoitustehtäviä. Tehtävien läpikäymiseksi tarvitaan työkaluja, jotka voivat tarkistaa opiskelijoiden työn jälkiä automaattisesti. Tämä diplomityö keskittyy ohjelmoinnin opettamisessa käytettävän automaattisen arvioinnin ongelmiin ja koittaa luoda tilanteelle korjauksen. Aluksi kuitenkin paneudumme itse ohjelmoinnin opettamiseen nykyaikaisessa korkeakouluympäristössä.

Nykyaikaisessa ohjelmoinnin opettamisessa korostuvat erilaiset verkossa toimivat oppimisympäristöt sekä opetuksen ja arvioinnin automatisointi. Tampereen yliopiston tietotekniikan koulutusohjelmassa alkutason kurssit ovat jopa yli 1500 hengen massakursseja, ja jatkokursseilla osallistujia on useita satoja. MOOC:t (engl. Massive Open Online Course) ovat vakiintuneet ympäri maailman opetusmenetelmänä eri aloilla niin avoimina kuin oppilaitosten sisäisinä tutkinto-ohjelmien toteutuksina [2]. Ohjelmoijan taidot ovat yhdistelmä laajaa teoreettista pohjatietoa sekä runsasta käytännön kokemusta ohjelmoinnista hyviä käytäntöjä noudattaen. Palautteen rooli hyvien käytäntöjen ja ohjelmoinnin oppimisessa on kriittinen. Näin ollen jokaisen opiskelijan työnjäljen tarkastaminen manuaalisesti on kurssien resurssien kannalta täysin mahdoton tehtävä, jolloin kurssitöiden arviointia on täytynyt muuttaa automaattiseksi.

Tampereen yliopiston Tuni+ -oppimisympäristössä automaattinen arviointi on toteutettu ns. automaattisten arvioijien (engl. automatic grader) avulla. Opiskelija lukee tehtävänannon Tuni+:sta, kirjoittaa tehtävää varten ohjelman tai muuta koodia, siirtää työnsä kurssin tarjoamaan versionhallintaan ja palauttaa työnsä tehtävän alla olevalle sivulle antamalla linkin versionhallintaansa. Palautus kopioidaan opiskelijan versionhallinnasta erillisen arviointipalvelun haltuun, joka luo arviointia varten palvelun sisälle erillisen hiekkalaatikkoympäristön (engl. sandbox), jossa palautettu koodi käännetään, suoritetaan tai käydään muuten läpi. Arvioija kerää arvioinnin tulokset talteen ja palauttaa nämä opiskelijalle Tuni+:aan tehtäväsivulle, josta tehtävän onnistumisen tai mahdolliset virheet voi lukea suo-

raan. [3]

Tuni+ on Aalto-yliopiston Learning + Technology -tutkimusryhmän luomaa oppimisympäristöä A+:aa vastaava järjestelmä [4]. Tuni+ sisältää verkkomateriaalia mm. useille ohjelmoinnin sekä matematiikan kursseille, joissa opiskelijoiden suorittamat tehtävät ovat arvioitavissa verkon välityksellä. Opetettaviin ohjelmointikursseihin kuuluvat mm. Ohjelmointi 1: Johdatus, Ohjelmointi 2: Perusteet, Ohjelmointi 3: Tekniikat, sekä Basic Web Applications. Tämän diplomityön kohteena ovat näillä ohjelmoinnin kursseilla käytettävät arvioijat. Kurssien käyttämiin ohjelmointikieliin ja ympäristöihin kuuluvat Python, C++, HTML, JavaScript ja QtCreator-ohjelmointikehys, ja tämä vaikuttaa vahvasti kurssien käyttämiin arvioijiin ja niiden tarpeisiin.

Automaattisella arvioinnilla ja sen yhtenäistämisellä ja kehityksellä on useita organisatorisia syitä. Keskeisin näistä on yllä annettujen kurssiosallistujien valtava määrä kurssihenkilökuntaan nähden, mikä tekisi kurssien tehtävien arvioinnista mahdotonta ilman automaattisia palveluita. Lisäksi ohjelmointikurssien toteutukset ja henkilökunta ovat tasaisesti vaihtuvia, jolloin yhdenmukaisten ja hyvin dokumentoitujen kurssipohjien avulla kurssien järjestäminen ja ylläpito on mahdollisimman vaivatonta. Lisäksi kurssien ja opetustahojen, kuten vanhojen Tampereen yliopiston ja Tampereen teknillisen yliopiston fuusioituminen voidaan tehdä helpommaksi yhtenäistämällä kurssien käytäntöjä ja teknisiä toteutuksia.

Tuni+:ssa opetettavien ohjelmointikurssien automaattisten arvioijien suhteen on noussut esiin ongelmia arvioijien ylläpidossa, uuden materiaalin lisäämisessä, ja keskinäisessä yhdenmukaisuudessa [5, 6, 7, 8]. Usein kurssihenkilöt uutta kurssi-instanssia luodessaan kopioivat vanhojen toteutuskertojen arvioijia ja muokkaavat jokaista manuaalisesti uuden kurssin tarpeisiin, ja sama toistuu lukuvuosittain. Näin ollen arvioijien rakenne on hyvin sekava, mikä johtaa siihen, että uuden henkilökunnan työmäärä arvioijien käyttöönotossa ja opettelussa on valtava.

Edellä esitetyistä ongelmista voidaan muodostaa tutkimuskysymys: ”Miten Tuni+:ssa olevien ohjelmointikurssien automaattisia arvioijia voidaan yhtenäistää?” Millaisilla työkaluilla ja käytännöillä tätä voidaan edesauttaa?”. Tässä diplomityössä etsitään vastaus edellä oleviin kysymyksiin käyttäen konstruktiivisen tapaustutkimuksen menetelmiä. Konstruktiivinen tutkimus nojaa aikaisempaan tutkittuun tietoon, teorioihin ja kirjallisuuteen, joita sovelletaan uuden toimivan ratkaisun luonnissa. Ratkaisu voi olla tilanteesta riippuen kone, tietojärjestelmä, malli, ohjelma, algoritmi tai jokin muu vastaava kokonaisuus, jota käyttämällä esiintynyt ongelma saadaan ratkaistua. [9]

Työ aloitetaan esittelemällä ohjelmien arviointia ja arvioinnin automatisointia luvussa 2. tutkimalla olemassaolevia ohjelmointikurssien arvioijia sekä haastattelemalla kurssien henkilökuntaa arvioijien käytöstä. Näiden pohjalta luodaan vaatimusmäärittely uudelle työkalulle luvussa 3 ja kartoitetaan sen tarvitsemat resurssit. Tämän jälkeen luvussa 4 esitellään tarkemmin jo olemassa olevat komponentit ja tutkitaan miten niillä voidaan vastata

näihin tarpeisiin. Kurssihenkilökunnan haastatteluiden ja käsiteltyjen kurssien analysoinnin kanssa saadaan ideoitua työkalu, joka lopulta luodaan luvussa 5 esitettyjä yhtenäistämisen menetelmiä hyödyntäen. Lopuksi työkalua ja käytettyjä menetelmiä arvioidaan luvussa 6. Arviointiin perustuen esitetään työn yhteenveto ja ideoita työkalun jatkokehitykseen luvussa 7.

2. AUTOMAATTINEN ARVIOINTI JA LÄHDEKOODIN REFAKTOROINTI

Tässä luvussa avataan lukijalle automaattisen arvioinnin toimintaperiaatteita sekä Tampereen yliopiston käyttämän Tuni+ -palvelun arkkitehtuuria. Aloitamme tavallisesta ohjelmakoodin arvioinnista, josta siirrymme arvioinnin automatisointiin. Lopuksi kuvaamme automaattisen arvioinnin yhteydestä ohjelmoinnin opetukseen.

2.1 Ohjelmakoodin arviointi ja sen ongelmat

Ohjelmien testaamisen juurisyy löytyy yksinkertaisesta ajatuksesta ohjelmoinnin saralla: kaikki testaamaton koodi oletetaan toimimattomaksi. Ohjelmoinnin perimmäinen tavoite on tuottaa toimivia käskykokonaisuuksia eli ohjelmia, joten tämän tavoitteen täyttämiseksi ohjelmoijien tulee varmistua, että heidän tuotoksensa tekevät sitä mitä niiltä odotetaan. Ohjelmoinnin opetuksen kannalta testauksella on tärkeä tehtävä varmistaa, että opiskelijat ovat saaneet kirjoitettua ohjeistuksen mukaisesti toimivia ohjelmia, jolloin he ovat ymmärtäneet käsiteltävät aihealueet ohjelmoinnista, ja testien läpäisystä ei muodostu ongelmaa.

Ohjelmoinnin opetuksessa opiskelijoiden tehtävien arviointi on usein haastavaa. Cheang et al. [10, s.123] kuvaavat tyypillisen ohjelmointitehtävän arviointiin liityviä lähestymistapoja. Tehtävät voidaan jakaa arvioinnin kannalta kolmeen kategoriaan:

- **Oikeellisuus.** Ohjelma palauttaa oikeita arvoja, kun sille annetaan laillisiksi määriteltyjä syötteitä. Tämä kattaa ohjelman perustason vaatimukset.
- **Tehokkuus.** Ohjelma on tehokas jos se suoriutuu sille annetuista vaatimuksista tarpeeksi nopeasti sekä tarpeeksi pienellä muistinkäytöllä.
- **Ylläpidettävyys.** Ohjelman ylläpidettävyyteen vaikuttavat sen lukijaa auttavat seikat, kuten nimeämiskäytännöt, kommentit, modulaarisuus ja sisennykset.

Näistä kategorioista ohjelman oikeellisuus on painoarvoltaan suurin ohjelmoinnin arvioinnissa, sillä väärin toimiva ohjelma voidaan tulkita tehtävänannon vastaiseksi. Tampereen yliopiston ohjelmointikursseilla harjoituksien koodia arvioidaan tietyillä kursseilla useasta näkökulmasta, mutta varsinkin aloituskursseilla painopiste on oikeellisuudessa eli funk-

tionaalisessa testauksessa kurssisisällöllisistä syistä.

Ihmisen tekemänä ohjelmakoodin arviointi ohjelmoinnin opetuksessa on hyvin vaikeaa. Cheang et al. esittävät useita ongelmakohtia tähän liittyen [10, s. 124]:

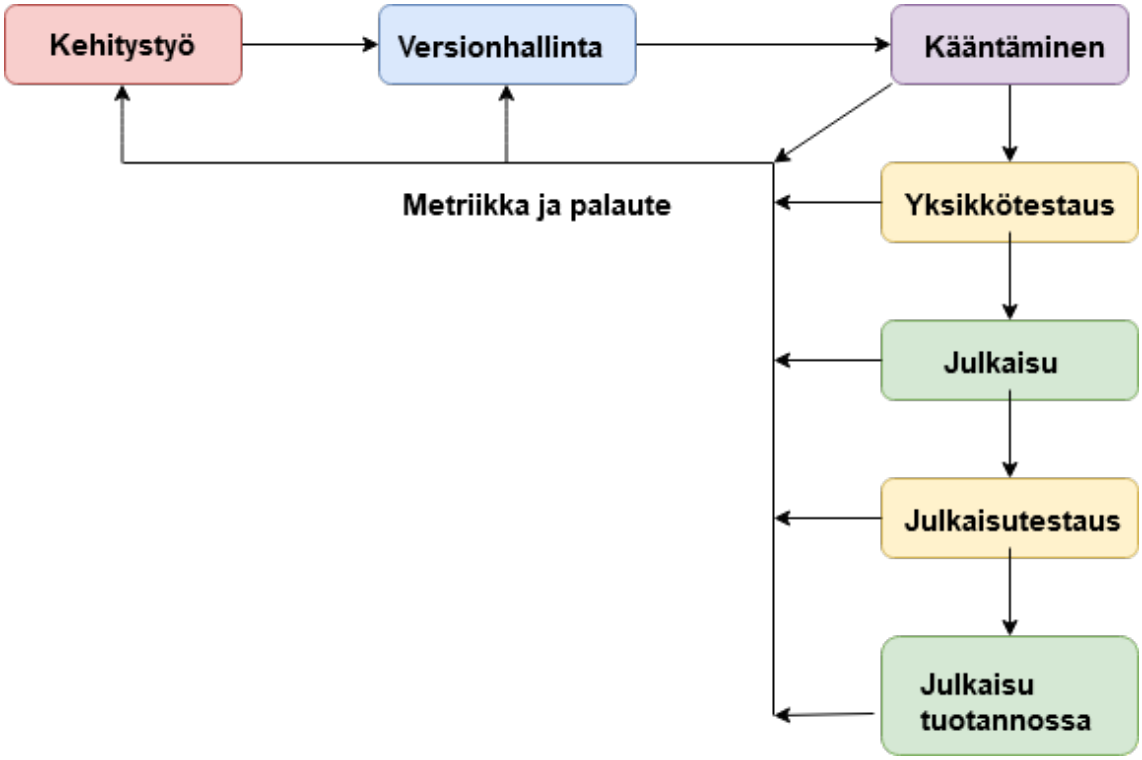
1. Oikeellisuuden ja tehokkuuden arvioinnissa virhetilanteen sattuessa ohjelmaa joudutaan käymään manuaalisesti läpi rivi riviltä, jotta esiintyneet ongelmat löydetään. Myös ohjelmien tarkastamisessa käytetty testidata vaikuttaa tuloksiin, sillä tietyllä syötejoukolla toimiva ohjelma saattaa kaatua toisella validilla syötejoukolla. Testidataongelman tulisi olla siis hyvin kattavaa, ja ongelman koko kasvaa ohjelman koon ja mahdollisten syötteiden määrän kanssa.
2. Samaan ongelmaan voi olla useita ratkaisuja, jotka ovat keskenään yhtä oikeellisia tehtävänannon kannalta. Usein annettuihin tehtäviin on olemassa mallivastaus, jota tarkastajat käyttävät tehtäviä arvioidessaan, mutta yksi malli vastaa vain yhtä mahdollista ratkaisua. Jos tarkastaja ei tunne muita vaihtoehtoja ongelman ratkaisuun, usein seurauksena on matalampi arvosana palautukselle.
3. Ohjelman ulkoasu korostuu arvioinnissa, kun edellä mainitut seikat vaikuttavat arvioijaan. Jos ylläpidettävyyteen liittyvät seikat vaikuttavat merkittävästi tehtävän arvosanaan, opiskelijoiden huomio siirtyy itse oikeellisuudesta ja tehokkuudesta muualle.
4. Ihmisten epäjohdonmukaisuus arvioinnissa on yleinen ongelma. Jos yhtä tehtävää arvioi useampi henkilö, lähes kaikki antavat eri arvosanan tulokseksi. Ihmisen tekemään arviointiin vaikuttavat mm. mieliala ja vireys, jolloin erityisesti ohjelmoinnin saralla yhden merkkivirheen huomiointi palautuksessa voi vaikuttaa arvosanaan merkittävästi.
5. Kurssiresurssien kannalta ihmisten tekemään arviointiin kuluu valtava määrä aikaa. Ongelmaa pahentaa se, että usein palaute yksittäisestä tehtävästä tulee tehtävän teon ja palautuksen jälkeen, jolloin palautteen hyödyllisyys laskee, kun käsiteltävä asia ei ole enää opiskelijoiden mielessä.

Yllä mainitut ongelmat ovat johtaneet siihen, että ohjelmoinnin opetuksessa ohjelmien arviointia on alettu suorittaa enenevissä määrin automaattisesti. Seuraavaksi paneudumme automaattiseen testaukseen, ja siihen miten ohjelmoinnin opetuksessa arviointi voidaan automatisoida.

2.2 Automaattinen testaus ja arvioinnin automatisointi

Automaattinen testaus kuvaa tilannetta, jossa ohjelmakoodin toiminnallisuus ja laatu arvioidaan toisen ohjelman tai tietoteknisen järjestelmän toimesta uusien muutosten jälkeen [11, s. 14]. Esimerkkinä tyypillisen ohjelmistoprojektin Git-repositorioon pusketulle koodille voidaan suorittaa heti sille kirjoitetut yksikkötestit, joilla varmistetaan, että käsiteltävän

luokan toiminta ei ole hajonnut. Usein automaattinen testaus on osa ns. jatkuvan integraation ja jatkuvan julkaisun (engl. CI/CD, Continuous Integration/Continuous Deployment) työnkulkua [12], jossa projektissa tuotettua uutta koodia ajetaan etukäteen määritetyn testisetin läpi aina, kun lähdekoodiin tehdään muutoksia. Tällä varmistetaan, että uusi koodi ei riko olemassaolevan kokonaisuuden toimintaa, vaan on toimiva lisä vanhaan koodiin.



Kuva 2.1. *Esimerkki jatkuvan julkaisun putkesta.*

Kuvassa 2.1 on esitetty yleinen esimerkki CI/CD-putkesta ohjelmistoprojektissa. Lähdekoodin muutokset käännetään ja sen yksikkötestit ajetaan, jonka jälkeen sovellus voidaan julkaista erillisessä ympäristössä, jossa sen kuuluisi toimia kuten tuotannossa. Testiympäristössä voidaan myös testata sovelluksen toimintaa ja varmistua, että sovellus on valmis tuotantoon. Tärkeänä osana CI/CD-putkea on eri vaiheista saatava palaute ja testitulokset, jotka ohjaavat työntekijöitä vikojen korjaamisessa.

Moses et al:n tutkimuksen [13] mukaan automaattisella testauksella on useita hyötyjä ohjelmistoprojektissa. Testaus parantaa tuottavuutta, sillä usein testit ovat helposti uudelleenkäytettäviä, ja testien helppo toistettavuus mahdollistaa suurempien testimäärien suorituksen. Automaattiset testit myös vähentävät testauksen ajankäyttöä ja kuluja projekteissa. Manuaaliseen testaukseen verrattuna automaattisen testauksen hinta on suurempi, mutta ajan kuluessa sillä on tuottavuuteen parantava vaikutus. Testauksen automatisointi ja ylläpito vaatii kuitenkin ylimääräistä työtä, eikä se korvaa täysin manuaalista testausta. [13]

Ohjelmoinnin opettamisen kontekstissa automaattista testausta voidaan soveltaa opiskelijoiden työn laadun arvioinnissa. Tällöin toiminnasta käytetään termiä automaattinen arviointi. Automaattinen arviointi on opiskelijoiden kirjoittaman koodin arvioimista valmiilla testeillä, ja näiden testien tulokset määrittävät opiskelijan onnistumisen ohjelmointitehtävässä [1, s. 27]. Tampereen yliopistossa ohjelmoinnin opettamisessa käytössä oleva Gitlab-palvelu [14] mahdollistaa helpon koodin hallinnan opiskelijoille ja kurssihenkilökunnalle eri kursseilla. Git on myös oleellinen osa automaattista arviointia, sillä se mahdollistaa opiskelijoiden harjoitusten siirtämisen näiden repositorioista testiympäristöihin, joissa työt voidaan arvioida ja opiskelijoiden saama palaute voidaan koota.

Automaattiseen testaukseen verrattuna automaattinen arviointi voidaan nähdä typistettynä versiona CI-putkesta, jossa opiskelijat kirjoittavat harjoitustehtäviä ja siirtävät niitä versionhallintaan, josta tehtävän koodi siirtyy automaattiseen testaukseen. Testauksesta lähetetään palautetta opiskelijalle, joka tarpeen mukaan tekee korjauksia tehtävään, ja palauttaa sen uudellen arvioitavaksi. Lopulta tuloksena on tehtävän vaatimukset täyttävä sovellus, joka on kehitetty jatkuvan integraation menetelmillä. Seuraavaksi esitellään automaattisen arvioinnin hyötyjä opetuksessa sekä sen kehittämistarpeita.

2.3 Automaattisen arvioinnin pedagogiset hyödyt ja kehittäminen

Glass et al. esittävät analyysissään [15] eri lähestymistapoja automaattisen arvioinnin tutkimiseen ohjelmoinnin opetuksessa. Nämä lähestymistavat voidaan jakaa kolmeen eri kategoriaan:

- **Formulatiivinen.** Tutkimista varten luodaan rakennetaan erilliset arviointiympäristöt sekä -työkalut. Palautettavien kurssitehtävien arviointiin luodaan myös oma prosessinsa. Tällä lähestymistavalla saadaan työkaluja ja menetelmiä joilla oppilaiden työn laatua voidaan tutkia ja heitä voidaan auttaa laadun parantamisessa.
- **Evaluatiivinen.** Edellä kuvatut arviointiympäristöt ja -prosessi arvioidaan tapaustutkimuksena yksittäisellä kurssilla. Tutkimuksessa keskitytään sekä oppilaiden että opettajien näkökulmaan, ja tavoitteena on löytää työkalujen ja prosessin vahvuuksia sekä parannuskohteita.
- **Deskriptiivinen.** Olemassa olevia tutkimuksia arvoinnin automatisoinnista, ohjelmoinnin opetuksesta ja oppimisesta analysoidaan. Kirjallisuuskatsauksia sekä aikaisempien kokemuksien synteesiä hyödynnetään tutkittaessa kuinka automaattinen arviointi tukee ohjelmoinnin opetusta. Tavoitteena on tunnistaa ja kuvailla tärkeitä käsitteitä evaluatiivista tutkimusta varten ja löytää uusia tutkimuskohteita.

Näistä kolmesta lähestymistavasta Glass et al:n mukaan formulatiivinen on tietojenkäsittelytieteissä ylivoimaisesti suosituin. Formulatiivinen tutkimustapa yhdistää automaattisen arvioinnin tutkimisen ja ohjelmoinnin opetuksen käytännöt useissa korkeakouluissa,

joissa automaattisesta arvioinnista on tullut osa kurssien vakiosisältöä.

Ala-Mutkan formulatiiviseen lähestymistapaan pohjautuvassa tutkimuksessa Tampereen teknillisessä yliopistossa Tietorakenteet ja algoritmit -kurssi käytti vanhan opetusassistentteihin perustuvan arvioinnin sijaan monivaiheista puoliautomaattista arviointia , jossa automaattinen arviointi toimii opetusassistenttien tukena kurssitöiden arvioinnissa. Tässä mallissa opiskelijat työskentelevät itsenäisesti, ja heillä on apunaan automaattisia arviointityökaluja. Tehtävien palautuksessa automaattinen arviointi varmistaa, että minimikriteerit palautukselle täyttyvät, ja opiskelijat saavat palautetta heti. Lopulta opetusassistentti käy palautukset läpi automaattisen arvioinnin kanssa, ja joko hyväksyy tai hylkää palautuksen. Ala-Mutka et al. toteavat , että mallin pedagoginen hyöty riippuu arvostelukriteerien selkeydestä opiskelijoille, arvioinnin työkalujen ja ominaisuuksien saatavuudesta ja niiden muokattavuudesta sekä harjoitustehtävän mallin ja arviointikriteerien laadusta. Vanhaan toteutukseen verrattuna uusi arviointitapa tuo monia etuja, kuten välittömän palautteen työstä opiskelijoille, kontrolloidut laatuvaatimukset palautuskoodille sekä laajemman arviointikattavuuden, johon sisältyy käännösvaroitukset, funktionaalisuus, muistinhallinta, ohjelmointityyli ja tehokkuus. [1, s. 36-37]

Ala-Mutka esittää, että mainittujen parannusten lisäksi automaattinen arviointi parantaa opetustyötä myös muilla tavoin opettajan kannalta. Tavoitteiden ja kurssisisällön suunnittelussa automaattista arviointia voidaan hyödyntää opiskelijoiden lähtötason mittaamisessa. Opiskelijoiden oppimisen ja työskentelyn tukemisessa automaattisella arvioinnilla voidaan ohjata opiskelijoita ajattelemaan tehtävien vaatimuksia sekä antaa suoraa palautetta formatiivisen analyysin avulla ja näin edistää hyviä ohjelmointikäytäntöjä. Arvioinnin ja kurssitoteutuksen kehityksen kannalta automaattinen arviointi tarjoaa tasavertaisen arvostelun opiskelijoille, ja sillä voidaan kerätä ajan tasalla olevaa tietoa opiskelijoiden tuloksista, ongelmista sekä työtavoista kurssilla. [1, s. 85]

Ala-Mutka myös huomauttaa, että automaattisen arvioinnin käyttämiä työkaluja tulee aktiivisesti kehittää eteenpäin, dokumentoida ja tutkia puutteiden ja parannuksien osalta. Yllä esitellyssä tutkimuksessa ilmeni ongelmia käytetyn arviointityökalun dokumentaation lukemisessa, kun vain 1 8:sta assistentista oli tutustunut päivitettyyn työkalun paikalliseen dokumentaatioon. Yksi assistentti kertoi haastattelussa, että hän ei halua lukea työkalun dokumentaatiota tai harjoituskonfiguraatiota, vaan haluaa vain tarvittavat komennot arviointia varten. Tämä asenne saattaa olla Ala-Mutkan mukaan yleinen kurssihenkilökunnan keskuudessa, jolla on paljon opetustehtäviä. Ala-Mutka esittää, että tämä tekee käytettyjen työkalujen keskitetystä ylläpidosta välttämätöntä ja hyvän ratkaisun, sillä se tarjoaa keskitetyn hallinnoinnin arviointijärjestelmille ja sen työkaluille. Ala-Mutka kuitenkin toteaa, että työkalujen tulee olla joustavia kurssien tehtävien suhteen, ja kyetä vastaamaan kurssien opetustavoitteisiin. [1, s. 57-58]

Ihantola et al. kuvaavat artikkelissaan [4] avoimen lähdekoodin ratkaisua ja arkkitehtuuria

sähköiselle oppimisenhallintajärjestelmälle (engl. LMS, Learning Management System), joka kykenee suorittamaan automaattista arviointia. A+ -nimisen järjestelmän arkkitehtuuri perustuu opiskelijoiden ja harjoituspisteiden tietojen sekä tehtävien tietojen ja arvioinnin jakamiseen erillisiin palveluihin, jotka kommunikoivat keskenään HTTP-protokollalla. Tehtävien tarkistamisesta vastaava palvelu on tilaton, eli aikaisemmat palautukset eivät vaikuta tuleviin palautuksiin, eikä palvelu tarvitse opiskelijoista tai kurssista tietoja tehtävän arviointiin. A+ tarjoaa myös mahdollisuuden staattiselle arvioinnille, jossa opiskelijan palautus tallennetaan A+:aan, jonka jälkeen kurssiassistentti voi noutaa palautuksia A+:sta ja arvostella ne. Kurssiassistentti lähettää tiedot harjoituspalvelulle, joka välittää ne eteenpäin A+:aan. A+:n keskeinen etu on LMS:n eri vastuualueiden selkeä erottelu, jossa harjoituspalvelu voidaan keskittää vain harjoitusten käsittelyyn, A+:n front endin hoitaessa mm. autentikoinnin ja kirjanpidon. [4]

Ala-Mutkan huomiot automaattisessa arvioinnissa käytettävien työkalujen ylläpidosta ovat osoittautuneet kurssihenkilökunnan haastattelujen perusteella oikeiksi. Yhtenäistämisen ja keskitetyn ylläpidon tarve näkyy tässä diplomityössä myöhemmin luvussa 5 esiteltävillä kursseilla pääosin työkalujen puutteellisena dokumentaationa ja tästä johtuvana ongelmatilanteiden korjaamisen vaikeutena. Nämä seikat toimivat motivaationa tälle diplomityölle, jolla haetaan parannusta nykyiseen tilanteeseen uusilla työkaluilla ja menetelmillä. Seuraavaksi esittelemme, miten lähdekoodia voidaan parantaa luettavuuden ja

2.4 Refaktorointi, yleistäminen ja parametrisointi

Refaktoroinnilla tarkoitetaan ohjelmakoodin mallin muokkaamista ilman, että ohjelman toiminnallisuus muuttuu. Sillä on suuri merkitys laajojen ohjelmistokokonaisuuksien laadun ylläpitämisessä ja parantamisessa. Refaktoroinnin avulla vanhentutta koodia voidaan muokata sopivaksi uusiin järjestelmiin ja selkeyttää ja tehdä helpommin luettavaksi. [16, luku 2]

Yleistäminen (engl. generalization) on yksi refaktoroinnissa käytetyistä tekniikoista. Siinä olemassa olevasta koodista etsitään ominaisuuksia, jotka voidaan yhdistää abstraktimmaksi kokonaisuudeksi, luoden "yleisemmän"ilmaisun aikaisemmista ominaisuuksista. Tämä lähestymistapa on hyvin yleinen olio-ohjelmoinnin parissa, jossa käsitteet superluokka ja alaluokka kuvaavat yleistämistä ja tämän vastakohtaa spesifiointia. Superluokalle voidaan antaa useiden eri alaluokkien ominaisuuksia ja toiminnallisuuksia, jolloin superluokka korvaa joukon alaluokkia ja näin yleistää ne. [17, s. 336]

Ohjelmoinnissa parametrisoinnilla kuvataan ohjelmakoodissa käytettävien arvojen muuttamista kovakoodatuista vakioista muuttujiksi, jotka voivat vaihdella arvoltaan [18]. Parametrisointi on yksi aikaisemmin kuvatun refaktoroinnin menetelmistä. Yleinen tapa ohjelmakoodin yleistämiselle on käytettyjen arvojen kysyminen käyttäjältä, ja näiden arvojen välittäminen eteenpäin ohjelmassa. Yllä esitetyssä yksinkertaisessa ohjelmassa 2.1 on

```
1  #Triviaali kovakoodattu summafunktio
2  def summation():
3          a = 12
4          b = 13
5          return a + b
6
7  result = summation()
8  print(result)
9
10 #Parametrisoitu summafunktio
11 print("Please enter the first integer:")
12 first = int(input())
13 print("Please enter the second integer:")
14 second = int(input())
15
16 def summation(a, b):
17         return a + b
18
19 result = summation(first , second)
20 print(result)
```

Ohjelma 2.1. *Esimerkki ohjelmakoodin parametrisoinnista.*

kaksi Python-kielellä toteutettua summafunktiota. Ylempi toteutus on kovakoodattu kokonaisluvuille 12 ja 13, ja tällöin funktio tuottaa joka kerta saman tuloksen 25, ellei lähdekoodin muuttujia a ja b muokata. Vaikka funktio toimiikin semanttisesti oikein, se on käytettävyydeltään erittäin rajoittunut ja käytännössä hyödytön, sillä se käsittelee vain tiettyä lukuparia. Alemmassa toteutuksessa käyttäjältä kysytään ensin kaksi arvoa, jotka funktio laskee yhteen ennen tulostusta. Muuttujat a ja b on parametrisoitu, jolloin funktion käytettävyys laajenee kokonaislukuihin yli- ja alivuotorajat huomioiden.

2.5 Koodikloonit

Joskus käsiteltävässä ohjelmakoodissa on useita kohtia, jotka on kirjoitettu joko täysin tai lähes samalla tavalla. Saini et al. ovat kuvanneet ilmiötä artikkelissaan [19], ja ilmiöstä käytetään termiä "koodiklooni"(engl. code clone). Koodikloonit voivat olla sekä tahallisia että tahattomia. Tahallinen koodin kloonaaminen on kustannus- ja aikatehokas tapa rakentaa ohjelmistoja tai uusia osia niihin. Työskentelevän henkilön taitotaso käytetyn kielten sekä ohjelmointikehysten suhteen on myös osasyy koodikloonien syntyyn, sillä toisaalla toimivaksi todettua kokonaisuutta on helppo käyttää kopioimalla se suoraan muualle joko ilman tai muutosten kanssa. Koodiklooneja voi syntyä myös tahattomasti, jos ohjelmistoihin tehdään uusia ominaisuuksia ja lisäyksiä ilman, että vanhoista olemassaolevista komponenteista ollaan tietoisia.

Saini et al. ovat esittäneet ohjelmistoissa esiintyvien kloonien jakoa 4 eri kategoriaan [19, s.723]:

1. **Sanatarkka klooni**. Kloonin koodi on samaa alkuperäiseen verrattuna, ainoana erona kommentit ja tyhjät rivit. Näitä klooneja voidaan tunnistaa merkkijonovertailulla.
2. **Parametrisoitu klooni**. Koodissa esiintyviä muuttujia ja avainsanoja on muokattu.
3. **"Läheltä piti"klooni**. Klooni on luotu alkuperäisestä muuttamalla, lisäämällä tai poistamalla lauseita [20].
4. **Semanttinen klooni**. Kloonin syntaksi poikkeaa alkuperäisestä, mutta sen käytös on säilynyt samana.

Näille kloonityyppein tunnistukseen on olemassa monia erilaisia menetelmiä. Saini et al. esittävät kloonien havaitsemiseen ja hallintaan yleistä algoritmia pseudokoodina [19, s.726]:

```
1  Vaihe 1: Syötä lähdekooditiedostot a ja b kloonintunnistus–
2           ja hallintajärjestelmään.
3  Vaihe 2: Käytä kloonintunnistusmenetelmää P kloonien tunnistukseen
4           lähdekooditiedostoista a ja b.
5  Vaihe 3: Jos lähdekooditiedostojen a ja b yhteneväisyys ylittää
6           raja–arvon T, siirry vaiheeseen 4, muussa tapauksessa
7           siirry vaiheeseen 8.
8  Vaihe 4: Siirrä havaitut kloonit tietokantaan ja päivitä tietokanta.
9  Vaihe 5: Määritä kloonien ylläpitokustannukset.
10 Vaihe 6: Vertaile ja järjestä kloonit ylläpitokustannusten
11          mukaiseen järjestykseen.
12 Vaihe 7: Refaktoroi tai poista kloonit.
13 Vaihe 8: Lopeta
```

Algoritmi 2.2. *Koodikloonien havaitsemiseen ja hallintaan käytettävä yleinen algoritmi.*

Kloonien tunnistukseen on olemassa useita lähestymistapoja. Näistä yleisimpiin kuuluvat alla listatut menetelmät:

- **Tekstipohjainen vertailu**: Lähdekooditiedoston rivejä verrataan itseensä ja toisten lähdekooditiedostojen riveihin. Samanlaiset rivit merkitään vertailumatriisiin, joka voidaan visualisoida pistekuvaajalla. Menetelmän ongelmana on herkkyys esimerkiksi tyhjien rivien tai merkkien muuttamiselle. [21]
- **Tokenien vertailu**: Lähdekoodista muodostetaan sarja tokeneita, joita verrataan toisiinsa kloonien löytämiseksi. Tämä menetelmä ottaa huomioon mahdolliset erot

koodirivien järjestyksessä eri koodilohkoissa, jotka toiminnaltaan ovat silti samanlaisia. [22]

- **Metriikkapohjainen vertailu**: Metriikkapohjaisessa vertailussa lähdekoodia itsessään ei tutkita, vaan siitä lasketaan erilaisia arvoja. Näitä arvoja voidaan vertaamalla saadaan tietoa lähdekoodissa olevista samankaltaisista osista. [23]
- **Abstraktit syntaksipuut**: Abstrakti syntaksipuu (AST, engl. Abstract syntax tree) on ohjelman lähdekoodista parsittu puurakenne. Puusta voidaan etsiä eri menetelmillä alipuita, jotka vastaavat toisiaan, mikä tarkoittaa kloonia lähdekoodissa. [24]
- **Ohjelman riippuvuusgraafit**: Ohjelman riippuvuusgraafi (PDG, engl. Program Dependency Graph) kuvaa ohjelman kontrolli- sekä datariippuvuuksia. Tektsipohjainen sekä tokeneihin perustuva vertailu riippuu tutkittavan koodin kirjoitusasusta ja eri osien järjestyksestä, jolloin osien paikkoja vaihtamalla tunnistuksen tulos muuttuu. Tämä ongelma voidaan ratkaista PDG:n avulla, joka abstraktoi ohjelman osien riippuvuuksia. [25]

Yllä esitettyihin menetelmiin lähdekoodin parantamiseksi palataan luvussa 5, jossa esitetään tässä työssä käsiteltäville ohjelmoinnin kursseille toteutettua yhtenäistämistä. Yhtenäistämisen keskeisimpänä ratkaisuna on sovellettu versio algoritmista 2.2, jolla joukko arviointiin käytettyjä skriptejä saadaan yhdistettyä yhdeksi pääskriptiksi.

3. JÄRJESTELMÄN VAATIMUSMÄÄRITTELYT

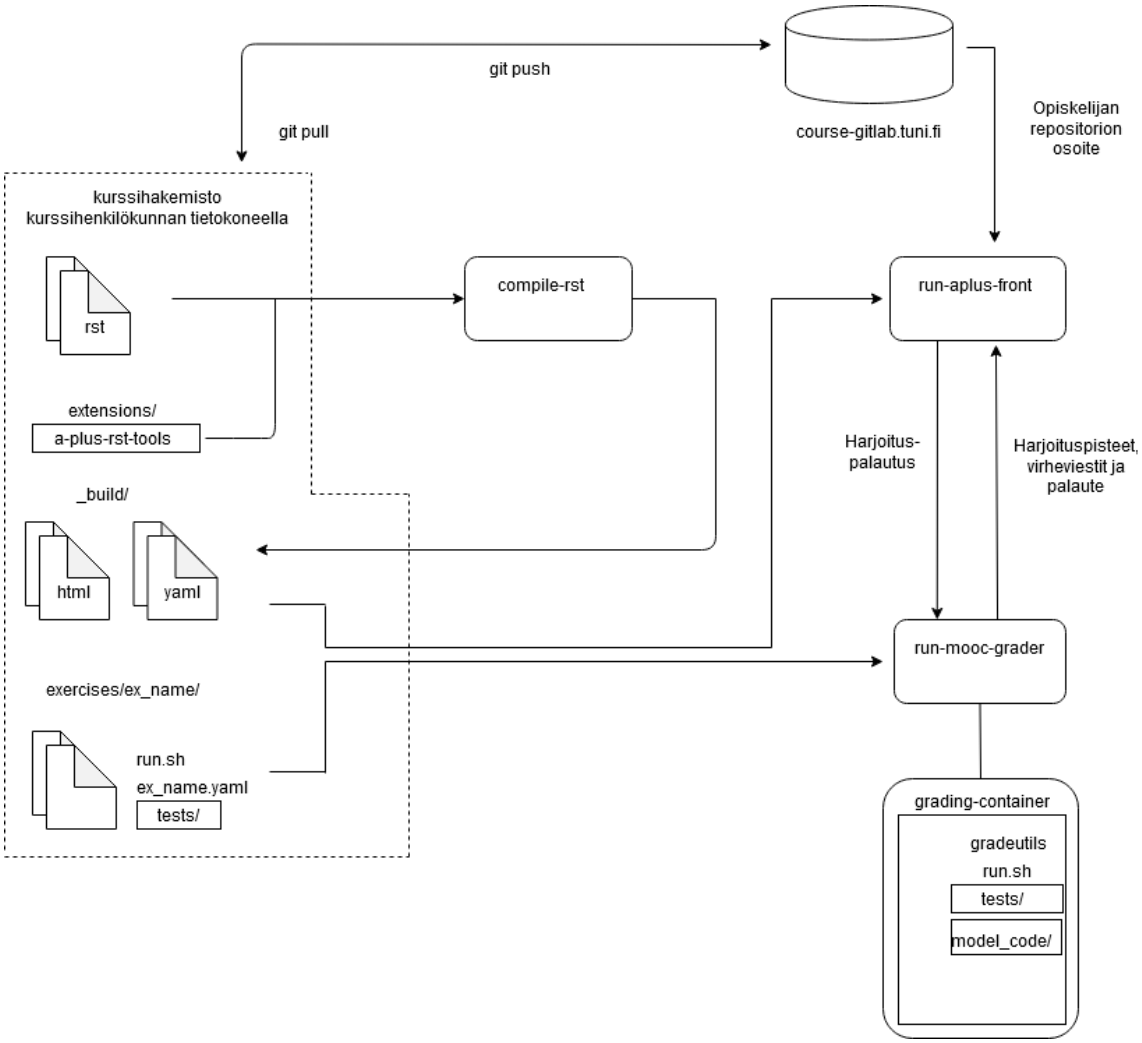
Tässä luvussa perehdytään Tuni+:n arkkitehtuuriin, sen käyttöön sekä työssä luotavan työkalun vaatimuksiin loppukäyttäjien eli opetushenkilökunnan kannalta. Aloitamme kuvaamalla Tuni+:n yksittäisen kurssin arkkitehtuurin, jonka pohjalta voimme määritellä käyttäjien tarpeet sekä vaatimukset uudelle työkalulle.

3.1 Tuni+ automaattisen arvioinnin alustana

Tampereen korkeakouluyhteisön Hervannan kampuksen tietotekniikan yksikkö on ottanut käyttöönsä Aalto-yliopiston Learning + Technology -tutkimusryhmän kehittämän A+-palvelun [4, 26] vuonna 2015. A+ on palvelupohjainen oppimisenhallintajärjestelmä, jossa kurssien sisältö, tehtävänannot ja tehtävien tarkastus on yhdistetty yhdeksi kokonaisuudeksi. Tampereen korkeakouluyhteisön käyttämä palvelu on saatavilla osoitteessa *plus.tuni.fi*, ja tässä diplomityössä siitä käytetään nimitystä Tuni+.

Kuvassa 3.1 on koottu Tuni+:n yksittäisen kurssin keskeiset osat kurssihenkilökunnan kannalta. Kurssihakemisto on git-repositorio, jossa kaikki kurssiin liittyvä sisältö on säilöttynä ja saatavilla niin henkilökunnalle muokattavaksi kuin myös kurssit tarjoavalle palvelimelle jaettavaksi [27]. Kurssihenkilökunta kirjoittaa kurssin opetussisällön ja rakenteen rst-tyyppisiin tiedostoihin, jotka käännetään A+ -kehitystiimin tuottamilla Sphinx-direktiiveillä [28] HTML- sekä YAML-tiedostoiksi compile-rst säiliössä Dockerissa [29]. Tässä työssä opetusmateriaalista käytetään termiä kurssisisältö, jonka lisäksi kursseihin kuuluu erilaisia ohjelmointitehtäviä, joihin annetaan ohjeita ja materiaalia kurssisisällön kautta. Tuni+:ssa esiintyvät harjoituspalautukset liitetään kurssisivuille viittaamalla rst-tiedostoissa erilliseen YAML-konfiguraatiotiedostoon, joka löytyy jokaisen harjoituksen kansiosta. Harjoituksen YAML-tiedostossa voidaan erikseen antaa tarkemmat ohjeet palautukselle, ja se myös sisältää tiedon mitä arviointeja vastaanottava Docker-säiliö tekee opiskelijan painaessa palautussivun submit-nappia.

Palautettavasta tehtävästä riippuen tehtävän YAML-tiedostossa käsketään arviointisäiliötä käynnistämään toinen erillinen säiliö, jossa varsinainen palautuksen tutkiminen tapahtuu. Näitä säiliöitä on kursseja varten useita erilaisia, ja joillakin kursseilla tarvitaan useita erilaisia säiliöitä tehtävien erilaisten ympäristöjen vuoksi. Tyypillisesti tehtävän arviointi alkaa opiskelijan palautuksessa antaman gitlab-repositorion kloonaamisella arviointisäi-



Kuva 3.1. *Tuni+:n arkkitehtuuri ja sen pääkomponentit [27].*

liöön, ja arvioitavan tehtävän testien ja muiden tarvittavien tiedostojen siirtämisellä oikeaan paikkaan. Tämän jälkeen palautukselle voidaan tehdä nopeat esitarkistukset mm. sallittujen include-komentojen ja muiden vastaavien kriteerien perusteella, jonka jälkeen itse palautuksen tarkistus voidaan suorittaa. Tarkistuksen tulokset tallennetaan erillisiin palautetiedostoihin, joista ne siirtyvät harjoituksen Tuni+ -sivulle opiskelijan luettavaksi.

Arviointiprosessissa suoritettavat komennot suoritetaan kurssien sisältämässä sandbox-hakemistossa sijaitsevalla run.sh-skriptillä. Skripti käynnistyy tehtävän YAML-tiedostossa viimeisenä olevalla komennolla, joka myös sisältää tehtävän tarkistukseen tarkoitettuja parametrejä, kuten tehtävän nimen. Run.sh-skriptit voidaan toteuttaa kahdella tavalla. Ensimmäisessä vaihtoehdossa jokaista kurssin tehtävää varten on olemassa oma run.sh-skriptinsä, johon tehtävän YAML-tiedosto viittaa tarkistuskomennossaan. Toisessa vaihtoehdossa kurssia varten on koottu yksi pääskripti, joka suoritetaan jokaisen tehtävän tarkistuksessa. YAML-tiedostossa on tällöin annettu parametreja pääskriptille, joiden avulla skripti suorittaa vain kyseiselle tehtävälle suunnatut testit ja muut komennot.

3.2 Kurssipohjan käyttötapaukset

Jotta henkilökunnan työnkulku uusien Tuni+ -kurssien luonnissa ja hallinnoinnissa pysyisi mahdollisimman paljon ennallaan ja LMS toimisi kuten aikaisemmin, Tuni+:n kurssipohjan arkkitehtuurin suhteen ei voida tehdä muutoksia tiettyjen ominaisuuksien kannalta. Näihin sisältyvät seuraavat ominaisuudet Tuni+:n arkkitehtuurista:

- Kurssipohja on itsessään Gitlab-repositorio
- Kurssisisältö on kirjoitettu rst-formaatissa
- Kurssin harjoitustehtävät tarkistetaan Docker-säiliöillä shell-skriptien kautta

Toisaalta tämä tarkoittaa, että luotavan työkalun kannattaa tukea tätä olemassa olevaa arkkitehtuuria mahdollisimman kattavasti, jotta sen käyttö Tuni+:n kanssa olisi mahdollisimman helppoa ja virheetöntä. Yllä esitetyt seikat vaikuttavat suunniteltuun työkaluun siten, että työkalun kannattaa tarjota käyttäjälle mahdollisimman vähän alustusta vaativa kurssipohja, joka on yhtenäistämismenetelmien avulla muutettu aikaisempiin kursseihin verrattuna mahdollisimman samanlaiseksi.

Kurssihenkilökunta suorittaa Tuni+:n yksittäiselle kurssipohjalle paljon toimenpiteitä. Henkilökunnalla tarkoitetaan yleisesti kurssin luennoitsijoita sekä mahdollisia opetusassistenttejä, joilla on pääsy kurssipohjaan ja kurssin materiaaleihin sen git-repositoriossa. Rajaamme nämä vain kaikkein yleisimpiin ja kurssin alustamisen ja ylläpidon kannalta krittisimpiin tapauksiin:

1. Henkilökunnan jäsen tarvitsee uuden kurssipohjan täysin uutta kurssia varten Tuni+:aan. Tätä varten Tampereen yliopiston Gitab-palveluun luodaan uusi repositorio, jossa on valmis kurssipohja perustuen Aallon vanhaan pohjaan. Tähän pohjaan kirjoitetaan erikseen kurssimateriaali rst-muodossa ja tehtäviä varten kopioidaan valmiit testit exercises-kansioon, kirjoitetaan tehtäville määrittelytiedostot ja luodaan tehtäviä varten run.sh-skripti, jonka kautta tarkistaminen tapahtuu. Kurssi testataan paikallisesti, Tuni+:n testipalvelimella ja lopuksi siirretään tuotantoon.
2. Henkilökunta lisää uuden tehtävän olemassa olevalle kurssille. Tätä varten tehtävän testitiedostot ja konfigurointitiedosto täytyy lisätä kurssipohjaan, ja tehtävän tarkistuskomennot tulee lisätä run.sh-skriptiin. Lisäysten jälkeen henkilökunta puskee ne Gitlabiin kurssipäivityksenä.
3. Henkilökunta joutuu tekemään korjauksia tehtävään. Tätä varten tehtävän määrittelytiedosto ja sitä vastaava yksittäinen run.sh-skripti tai tehtävälle kuuluva lohko kootussa run.sh-pääskriptissä tulee käydä läpi, löytyneet virheet korjata ja puskea Gitlabiin kurssipäivityksenä.
4. Henkilökunta lisää uutta kurssisisältöä olemassa olevalle kurssille. Henkilökunnan täytyy luoda uusi hakemisto kurssisisältöhakemiston sisälle, kirjoittaa uusi sisäl-

tö rst-tiedostoihin, ja listata nämä tietyn aihealueen index.rst-tiedostoon. Lisäykset pusketaan Gitlabiin kurssipäivityksenä.

5. Henkilökunta korjaa virheitä kurssisisällössä. Muutokset tehdään kurssisisällön rst-tiedostoihin tai niiden viittaamiin muihin materiaaleihin kuten kuviin. Muutokset pusketaan Gitlabiin kurssipäivityksenä.

Henkilökunnan toimenpiteet kurssipohjan suhteen keskittyvät pääasiassa kurssisisältöön rst-tiedostoissa sekä harjoitusten vaatimien tiedostojen käsittelyyn. Kurssien pääasialliset tekniset erot toisiinsa nähden ovatkin juuri rst-tiedostojen sisältö, kurssien sisältämät tehtävät sekä muutamat kurssikohtaiset asetukset Tuni+:aa varten, kuten kurssitunnuskoodit tai kurssin aukioloajat Tuni+:ssa. Uuden kurssin tapauksessa olisi hyödyksi, jos luotavalla työkalulla voitaisiin alustaa kurssisisällön runko, esimerkiksi luomalla tietty määrä valmiita hakemistoja eri aiheille. Tehtävien suhteen työkalulla voitaisiin luoda uudelle tehtävälle valmiiksi oikein määritetty YAML-tiedosto, johon tulisi enää lisätä tehtävänanto ja ohjeet. Tämä tosin edellyttää, että kurssia varten on olemassa run.sh-skripti, joka ymmärtää luotavan YAML-tiedoston tarkistuskomennon ja jolla tehtävien tarkistus voidaan hoitaa.

Edellä kuvattujen rajoitusten ja käyttötapausten pohjalta työkalulle voidaan hahmottaa selkeitä vaatimuksia sekä Tuni+:n että loppukäyttäjien näkökulmasta. Työkalun tulee:

- Tarjota rakenteeltaan valmis kurssipohja, joka voidaan liittää Tuni+:aan vähällä konfiguroinnilla
- Tarjota kurssipohjan kurssisisällölle runko, joka voidaan täydentää valmiiksi henkilökunnan toimesta
- Tarjota kurssin harjoituksia varten valmis run.sh-skripti ja muut tarkistusskriptit sekä kyky luoda valmiiksi konfiguroituja YAML-tiedostoja tehtäville

Nämä vaatimukset voidaan osittain täyttää jo olemassa olevalla kurssipohjalla, joka sisältää kaikki Tuni+:n tarvitsemat asetustiedostot, joilla kurssi saadaan siirrettyä LMS:ään. Ratkaistavaksi jää kurssisisällön rungon luonti sekä miten työkalulla voidaan luoda Tuni+:n ja yksittäisen kurssin kanssa toimiva harjoitustehtävän YAML-määrittelytiedosto. Tämän vuoksi käymme seuraavassa luvussa läpi Tuni+:n kurssipohjan komponentit, ja miten niitä voidaan soveltaa työkalun luonnissa.

4. OLEMASSAOLEVAT JÄRJESTELMÄT JA KOMPONENTIT

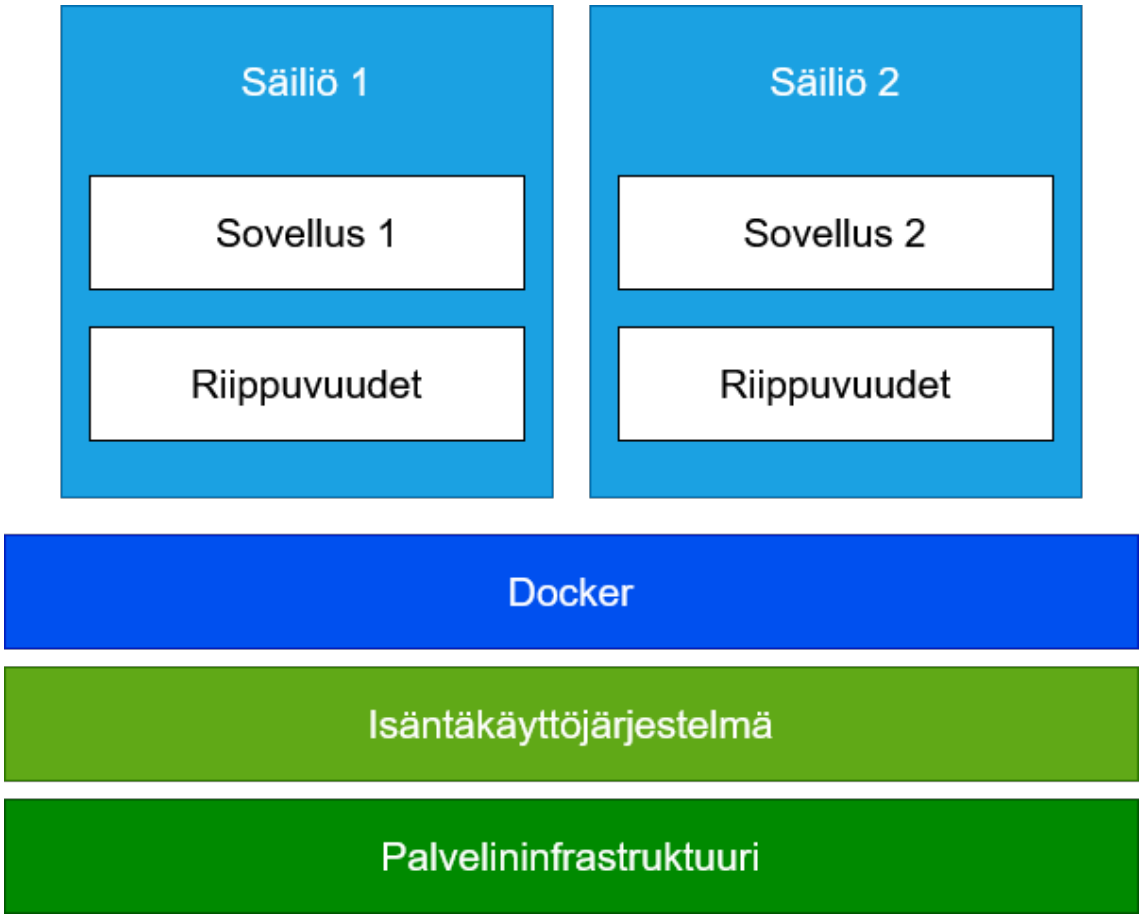
Tässä luvussa käydään läpi Tuni+:n kurssipohjaan kuuluvat valmiit komponentit, jotka mahdollistavat automaattisen tehtävien arvioinnin. Luvussa kuvataan myös komponenttien käyttötarkoitukset, sekä niiden roolit tässä diplomityössä luotaville uusille työkaluille.

4.1 Docker-säiliöteknologia

Tuni+ -ympäristöä suoritetaan Tampereen yliopiston palvelimilla Docker-säiliöteknologiaa käyttäen. Docker on -tyyppinen ohjelmisto, jolla olemassa olevia sovelluksia ja palveluita voidaan asettaa suoriutumaan säiliöiden (engl. container) sisällä. Useita ohjelmia ja palveluita voidaan kytkeä saman säiliön sisälle, ja useita erillisiä säiliöitä voidaan myös kytkeä toisiinsa verkkorajapinnan kautta. Dockerin suurimpiin etuihin palveluinfrastruktuurin suhteen kuuluvat laaja modulaarisuus sekä skaalautuvuus sekä säiliöiden pohjautuessa kustomoitavista kuvista ja niiden ollessa irrallisia kokonaisuuksia niitä ylläpitävistä palvelimista. [30]

Kuvassa 4.1 esitetään Dockerin arkkitehtuuri kerroksittain alkaen käytetystä palvelimesta. Säiliöön kuuluva sovellus ja sen riippuvuudet kuten kirjastot ja muut sovellukset kuuluvat säiliön kuvaan (engl. Docker image), joka on kooste säiliön sisällöstä [31]. Docker-kuva puolestaan rakennetaan erillisen Dockerfile-tiedoston avulla [32]. Dockerfile sisältää Dockerin rakennuskomentoja, jotka suoritetaan kerrallaan luoden joukon päällekkäisiä kerroksia. Yksi kerros vastaa yhtä Dockerfilessä esitettyä komentoa. Kaikki kerrokset paitsi Dockerfilen viimeiseen komentoon viittaava kerros on rajattu vain luku -oikeuksilla, ja viimeinen komento on oikeuksiltaan luku-kirjoitus. Tämä viittaa säiliön olevan interaktiivinen ja kykenevä ajamaan esimerkiksi tietyn ohjelman käynnistyessään, siinä missä muut kerrokset ovat vain lisäyksiä edelliseen kerrokseen [33].

Kuvassa 3.1 esitellään 4 Tuni+:n kannalta keskeistä Docker-säiliötä. Ensimmäinen järjestelmän suorituksessa on compile-rst, joka sisältää yksinkertaisen python-ympäristön kurssin rst-tiedostojen kääntämiseen HTML- ja YAML-tyyppisiksi. Run-aplus-front on Tuni+:n front endistä vastaava säiliö, joka vastaa mm. käyttäjien autentikoinnista sekä harjoituspistekirjauksesta. Run-mooc-grader on säiliö, joka toimii Tuni+:n ja itse tehtävää arvos-



Kuva 4.1. *Dockerin arkkitehtuuri kerroksittain.*

televan säiliön rajapintana. Se lukee compile-rst:n tuottamat tiedostot _build-kansiosta, ja esittää näin kurssin sisältöä sitä selaavalle opiskelijalle. Se myös ottaa vastaan palautuksia HTTP-muodossa, ja palauttaa niiden tulokset Tuni+:aan. Palautettaessa harjoitusta opiskelijan painaessa harjoitussivulla submit-nappia run-mooc-grader-säiliö käynnistää neljännen, tehtävässä määritellyn säiliön grading-container ja suorittaa sille määrätyn käskyn, yleensä run.sh-skriptin kurssin hiekkalaatikossa.

Dockerin suurin etu tulee sen mahdollistamista tehtäväkohtaisista säiliöistä, jotka voidaan rakentaa täysin uniikeiksi tehtävien arvioinnin tarvitsemien resurssien perusteella. Käytännössä yksi arviointiskriptejä suorittava säiliö koostuu käyttöjärjestelmästä, jonka sisällä arviointi suoritetaan, käyttöjärjestelmään asennetuista sovelluksista ja kirjastoista, kuten python-ohjelmontikielestä ja tietyistä python-kirjastopaketeista, sekä kurssirepositoriosta liitettävistä tarpeellisista tehtävä- ja testitiedostoista. Tuni+:ssa käytettävät arviointisäiliöt hyödyntävät Aalto-yliopiston luomia gradeutils-työkaluja, jotka muun muassa vastaavat arvioinnin pisteiden kirjaamisesta ja palautteen sekä pisteiden siirtämisestä oppilaan saamaan palautteeseen Tuni+:aan.

Luotavan työkalun kannalta Dockerin rooli on tarjota tehtäville säiliöt, joilla arviointi voidaan suorittaa. Uusien tehtävien kannalta vanhat jo käytössä olevat säiliöt eivät välttämät-

tä tarjoa sopivaa ratkaisua, joten työkalun lisäksi uusia tehtäviä varten tarvitaan toteutus, jolla niille voidaan luoda toimivat arviointisäiliöt.

Docker-säiliön rakentaminen on suoraviivaista. Säiliötä varten sille täytyy kirjoittaa erillinen Dockerfile-tiedosto, jossa määritellään kaikki säiliön rakennuksessa suoritettavat komennot. Docker hyödyntää rakentamisessa omaa komentoriviään, jonka syntaksiin kuuluu komentoja kuten RUN, USER, COPY ja WORKDIR [32]. Kurssien käyttämiin Docker-säiliöihin palataan myöhemmin luvussa 5, missä kuvataan uuden säiliön rakentaminen Ohjelmointi 3 -kurssin tarpeisiin.

4.2 Tuni+:n kurssipohja

Tuni+:ssa ohjelmointikursseilla tällä hetkellä käytettävät palautusautomaatit pohjautuvat A+:n kehittäjien tekemään kurssipohjaan [27], joka on valmiiksi rakennettu ympäristö kurssin tietosisällön ja tehtävien jakamiseen ja arvostelemiseen. Tässä luvussa kuvataan tarkemmin kurssipohjan sisältö ja sen kaksi pääosaa, kurssitehtävien tarkastukseen käytettävät skriptit sekä itse kurssisivuilla esitettävä sisältö. Luvussa 3 esitetyssä arkkitehtuurikuvassa 3.1 kurssipohja ja sen pääosat on esitetty katkoviivan sisällä.

Rakennettavan työkalun kannalta kurssipohja tarjoaa selkeän Tuni+:aan liittyvän kokonaisuuden, joka sisältää kaiken oleellisen kurssisisällön ja komponentit ohjelmointitehtävien julkaisuun ja arviointiin. Kurssipohjaa voidaan siis soveltaa työkalussa mallina, johon työkalun käyttäjä tekee työkalulla tarvittavia lisäyksiä kurssia varten.

4.2.1 Run.sh-skriptit ja palautuksille ajettavat testit

Jokainen automaattisesti arvioitava ohjelmointitehtävä Tuni+:n kursseilla hyödyntää run.sh-skriptejä. Käsiteltävillä kursseilla on kahdenlaisia toteutuksia skripteille: yhdessä kurssin hiekkalaatikkoympäristössä on yksi pääskripti, jonka kautta kaikki palautustehtävät siirtyvät arvioitavaksi, ja toisessa jokaiselle palautettavalle tehtävälle on erillinen skripti, jossa arviointi suoritetaan.

Kuvassa 4.2 on esitetty yksi Basic Web Applications -kurssilla käytettävistä run.sh-skripteistä, joka käy läpi Django-verkko-ohjelmointikehyksellä tehdyn projektin tietomalleja. Skripti alkaa opiskelijan palautuksen kloonaamisella arviointisäiliöön, jonka jälkeen arvioitavat tiedostot, tässä tapauksessa models.py, siirretään oikeaan paikkaan ja tehtävän Django-projektille kirjoitetut testit ajetaan `python manage.py test` -komennolla. Komentoa edeltävät `capture pre` -apukomennot tallentavat testien tulosteet HTML-käärittynä palautetiedostoihin, jotka `grade` -apukomento palauttaa käyttäjälle tehtäväsivulle.

Kriittisenä osana jokaista ohjelmointikurssia ovat niille suunnitellut harjoitukset ja niiden testit. Käsiteltävät kurssit perustuvat eri ohjelmointikielille ja -kehyksille, jolloin opiske-

```
#!/bin/bash
  if sudo-capture pre git-clone-submission /exercise/wsd_rsa \
    exercises/05_django/firstdjango/webshop/models.py
  then
    cd /submission/user
    cp -r /exercise/firstdjango /exercise/webshop /exercise/manage.py .
    mv models.py webshop/
    capture pre python3 manage.py test -v 2 webshop
    if [ $? -eq 0 ]; then
      echo "1/1" > /feedback/points
    else
      echo "0/1" > /feedback/points
    fi
    > /feedback/out
    err-to-out
  fi
grade
```

Kuva 4.2. *Esimerkki Basic Web Applications-kurssilla käytettävästä run.sh-skriptistä.*

lijoiden tekemät ja palauttamat tehtävät vaihtelevat valtavasti kurssien välillä. Jokainen palautettu tehtävä arvioidaan tehtävätyypin mukaan, esimerkiksi ohjelmoinnin johdatuskurssin ensimmäiset aloitustehtävät ovat Python-kielen print-komennon käyttöä ja tämän tulosteen vertailua oletettuihin tulosteisiin. Toisaalta harjoituksien arvioijilla voidaan myös esimerkiksi vain kääntää opiskelijoiden palauttama harjoitustyöprojekti ja varmistaa että projekti on saatu käyntiin, jolloin voidaan seurata ja varmistua osallistujien työskentelystä, ja näin aikatauluttaa kurssitöitä helposti aikarajojen muodossa.

Diplomityössä toteutettavan uuden työkalun kannalta run.sh-skriptien merkitys on suuri, sillä ne vastaavat Docker-säiliössä tapahtuvasta tehtävän arvioinnista. Näin ollen niiden toimivuus ja helppokäyttöisyys tulee saada mahdollisimman hyväksi. Yhtenäistämisen kannalta tarvitaan menetelmä, jolla nykyiset kaksi eri toteutustapaa skripteille voidaan muuttaa yhdeksi, joka on samanlainen kurssien kesken. Yhtenäistämisestä ja käytetystä menetelmästä kerrotaan lisää luvussa 5.

4.2.2 Kurssisisältö kurssipohjassa

Tuni+:n yksi ohjelmointikurssi vastaa käytännössä yhtä kurssipohjaa, jossa kurssin sisältö on ensin kirjoitettu rst-merkintäkielellä (engl. reStructuredText) [34]. Kurssihenkilökunta kirjoittaa kurssin sisällön erillisiksi osiksi ja listaa osat index.rst-tiedostoon, joista ne käännetään Sphinx-työkalun [35] avulla HTML-muotoon (engl. Hypertext Markup Language), jotka näkyvät Tuni+:ssa normaaleina verkkosivuina. Sphinx on alun perin dokumentaation kirjoittamiseen laadittu työkalu, mutta siihen on mahdollista kirjoittaa omia direktiivejä, jotka määräävät kuinka esim. käännös YAML:sta HTML:ksi tapahtuu. Aallon Plussa-kehitysryhmä on kirjoittanut kurssipohjia varten joukon direktiivejä, joilla kurssisisällön saa

helposti Plussaan näkyväksi HTML-muodossa.

```
1  Kurssin käytännönasiat
2  ==============================
3
4  .. aplusmeta::
5     :open-time: 2019-08-19
6     :close-time: 2019-12-31 23:59
7
8  .. toctree::
9     kaytanto_fi.rst
10    gdpr_fi.rst
11    harjoitustyo_yleinen_fi.rst
12    tentti_fi.rst
13    harjoitustyö_fi.rst
14
```

Kuva 4.3. *Esimerkki Ohjelmointi 3 -kurssin yksittäisen aihealueen index.rst-tiedostosta.*

Kääntämistä varten kurssisisältö tulee indeksoida kurssipohjan sisällä. Tähän käytetään index.rst-nimisiä tiedostoja, joihin listataan hierarkkisesti kurssin sivut alkaen kurssipohjan juuresta, jossa kaksikielisillä kursseilla listataan suomen- sekä englanninkielisten sivujen indeksit. Kurssin aihealueet ja sisältö on jaettu tyypillisesti rounds- tai modulesnimisen hakemiston sisään erillisiin hakemistoihin, jotka on myös indeksoitu tässä hakemistossa olevaan index.rst-tiedostoon . Yksittäisen aihealueen hakemistossa on kaikki aiheeseen liittyvä materiaali omina rst-tiedostoinaan, jotka on aihehakemiston sisällä indeksoitu kuvan 4.3 mukaisesti. Hierarkkinen esitys on mahdollista Sphinxin Toctree-direktiivin ansiosta, joka määrittää, mitä kurssisivuvalikossa Tuni+:ssa näkyy eri merkintöjen alla [36, 37].

Luotavan työkalun kannalta kurssisisällön rungon alustus voi toimia apuna uusien kurssien luonnissa. Tätä varten työkalun tulee kyetä rakentamaan hakemistorakenne kurssisisällölle, ja sijoittamaan tarvittavia rst-tiedostopohjia sekä index.rst-tiedostoja erillisten aihealue-hakemistojen sisälle. Lisäksi työkalun tulee sisältää kattava dokumentaatio rst- ja index-tiedostojen kirjoittamiseen, jotta ne näkyvät oikein valmiissa kurssissa.

4.3 Oppilaan saama palaute testeistä

Vaikka itse ohjelmatestit suoritetaan oppilaan kannalta erillään, niillä on tärkeä tehtävä opiskelijan työhön liittyen. Yleisesti testit tutkivat ohjelman toimintaa vertaamalla ohjelman ulostuloja oikeisiin arvoihin, suorittamalla ohjelmalle yksikkötestejä tai vertailemalla ohjelman graafisia elementtejä. Vertailuihin kuuluu oleellisesti oletetusta poikkeavien tulosten eli havaittujen virheiden huomiointi ja tiedon välittäminen palautuksen tekijälle. Tuni+:n arkkitehtuurista johtuen opiskelija ei pääse itse käsiksi testien ajoympäristöön, jolloin virhetilanteet täytyy välittää opiskelijalle verkon yli. Tämä on toteutettu käytössä olevissa arvioijissa kirjoittamalla mahdolliset kääntäjien ja testitapausten tulosteet erilliseen error-output-tiedostoon, jonka sisältö kiedotaan (engl. wrapping) HTML-muotoiseksi

ja tämä HTML-palaute lähetetään tehtäväsivulle. Opiskelija näkee palautteen erillisessä popup-ikkunassa tehtäväsivulla, jolla tehtävä alun perin palauttiin.

Yksityiskohtainen palaute on tärkeä asia monimutkaisemmissa harjoituksissa, sillä erilaisten virhetilanteiden määrä kasvaa ohjelman koon ja kompleksisuuden kanssa. Laajat arviointitestit sekä virhetapausten yksilöinti ja kuvaus antavat opiskelijalle tietoa siitä, missä tehtävään liittyvät ongelmat ovat, ja näin nopeuttavat asian korjaamista ja samalla vähentävät tarvittuja työtunteja ongelman parissa. Mikäli kurssihenkilökunta joutuisi selventämään kaikki virhetilanteet opiskelijoiden kanssa, se tarkoittaisi päivittäistä neuvontaa ja suurta lisätöiden määrää muiden töiden ohella. Myös itse palautettavaan tehtävään liittymättömät virheet arviointijärjestelmässä on tärkeää saada välitettyä eteenpäin, jolloin mahdolliset virheet arviointiskripteissä ja järjestelmässä saadaan korjattua nopeasti.

Tehtävien tarvitsemat palautteen muotoilukomennot löytyvät valmiiksi arviointia suorittavista Docker-säiliöistä, joiden sisällä run.sh-skriptit kutsuvat niitä testien jälkeen. Nämä komennot täytyy kuitenkin dokumentoida huolellisesti työkaluun, jotta kurssihenkilökunta saa välitettyä tarpeelliset tiedot tehtävistä opiskelijoille.

5. TOTEUTETTU RATKAISU JA TUTKIMUKSEN ETENEMINEN

Tässä luvussa kuvataan syntyneen ratkaisun rakenne ja kuinka ratkaisun rakentaminen eteni. Tutkimuskysymykseen ”Miten Tuni+:ssa olevien ohjelmointikurssien automaattisia arvioijia voidaan yhtenäistää?” Millaisilla työkaluilla ja käytännöillä tätä voidaan edesauttaa?” on haettu vastausta luomalla kursseilla käytettäville kurssirepositorioille oma rakennustyökalu, ja kursseilla käytettäville arviointi- säiliöille on luotu uudet pohjat tiedekunnan Gitlab-palveluun. Työkalua ja nykyisiä kursseja varten on myös sovellettu menetelmiä, joilla run.sh -skriptien toteutukset on saatu yhtenäisiksi kurssien kesken. Menetelmät on kuvattu luvussa 2.4. Lisäksi käytössä olevia arviointisäiliöitä on korvattu uusilla paremmin rakennetuilla säiliöillä. Tästä kerrotaan lisää luvussa 5.4.

5.1 Työhön liittyvät ohjelmointikurssit

Jotta tämän työn tuloksia voitaisiin esitellä ja perustella tarkemmin, täytyy aluksi luoda katsaus työn lähdemateriaaliin, eli Tuni+:ssa opetettaviin ohjelmoinnin kursseihin. Opetettaviin ohjelmointikursseihin kuuluvat mm. Ohjelmointi 1: Johdatus, Ohjelmointi 2: Perusteet, Ohjelmointi 3: Tekniikat, sekä Basic Web Applications. Tämän diplomityön kohteena ovat näillä ohjelmoinnin kursseilla käytettävät arvioijat. Kurssien käyttämiin ohjelmointikieliin ja ympäristöihin kuuluvat mm. Python, C++, HTML, JavaScript ja QtCreator-ohjelmointikehys, ja tämä vaikuttaa vahvasti kurssien käyttämiin arvioijiin ja niiden tarpeisiin.

Ohjelmointi 1: Johdatus on Tampereen yliopiston Hervannan kampuksen tarjoama ohjelmoinnin alkeiskurssi, jossa opiskelijat perehdytetään ohjelmoinnin käytäntöihin ja perusmenetelmiin. Kurssilla käytetään Python-kieltä PyCharm-nimisen IDE:n (engl. Integrated Development Environment) kanssa, ja tehtävät kurssilla etenevät yksinkertaisista tulostuksista laskureihin ja erilaisiin graafisiin työkaluihin. Aihesisällöltään kurssi on melko yksinkertainen: kurssilla käydään läpi perinteiset syöte- ja ulostulotoiminnot, matemaattiset operaatiot koodissa, ohjelman jako funktioihin, yksinkertaiset UI-elementit (engl. User Interface) ja perustason tietorakenteet. Kurssi on käsiteltävistä kursseista osallistujamäärältään suurin sen tutkinto-ohjelmissa olevan pakollisuutensa sekä alan kiinnostavuuden vuoksi, ja vuonna 2018 Tampereen yliopiston ja Tampereen teknillisen yliopiston kurssin

aloittajien yhteenlaskettu määrä oli 1610 henkeä.

Ohjelmointi 2: Perusteet on suora jatkokurssi edellä mainitulle johdantokurssille, ja nyt kieli vaihtuu C++:aan. Editorina ja työympäristönä kurssilla on käytössä QtCreator niminen IDE, joka on suunniteltu C++-ohjelmointiin ja sisältää omat graafisten komponenttien sekä testien kirjastot. Kurssin aihepiiri jatkaa peruskurssin asioista ja tarjoaa syventävän katsauksen mm. C++:n tietorakenteisiin, STL-kirjastoon, muistinhallintaan, rekursioon ja Qt:n graafisten komponenttien käyttöön. Kurssilla on parityönä tehtävä harjoitustyö, jossa kurssilla opittuja menetelmiä sovelletaan. Vastaava yllä esitetty kurssilla aloittaneiden määrä vuonna 2018 oli 945 henkeä.

Ohjelmointi 3: Tekniikat (OHJ3) on myös C++-pohjainen jatkokurssi edelliselle kurssille, jossa fokuksessa on ohjelmien rakenteet ja yleiset ohjelmointityöhön liittyvät tekniikat, kuten versionhallinta monen ihmisen projektissa, testaus yksikkötasolla, modulaarisuus ohjelmoinnissa, dynaaminen sitominen ja virtuaalifunktiot, muistinhallinta sekä poikkeukset. Kurssin sisältöä sovelletaan kurssin loppupuolella parityönä toteutettavaan harjoitustyöhön. Kurssilla oli vuonna 2018 aloittaneita yhteensä 746 henkeä.

Basic Web Applications (BWA) on diplomivaiheen kurssi, jossa tutustutaan verkkosovellusten ohjelmointiin. Kurssi esittelee HTML-, CSS- ja JavaScript-työkalut, verkkosivujen toimintaperiaatteita, verkkopalveluiden perusteet ja yhden web-ohjelmointikehyksen, nykyään node.js:n. Kurssilla on harjoitustyö, jossa 3 hengen ryhmä tuottaa ohjeistuksen mukaisen verkkopalvelun käyttäen opetettuja menetelmiä ja työkaluja. Kurssin koko on kasvanut aivan viime vuosina osallistujamäärän ollessa uusimmalla toteutuskerralla noin 300.

Edellä esitetyn kuvauksen perusteella on selvää, että kurssien sisältämien harjoitustehtävien ja -töiden arviointiin tarvitaan hyvin monipuolinen ja vaihteleva määrä testejä, kirjastoja sekä skriptejä. Tämä luo rajoitteita yhdenmukaistukselle, sillä samaan aikaan toteutuksen tulisi sallia uudentyyppisten tehtävien lisääminen kursseille ja mahdollistaa olemassaolevien tehtävien käyttämisen.

5.2 Tutkimuksen eteneminen

Tässä kappaleessa kuvataan tutkimuksen päävaiheet ja niihin sisältyvät toimenpiteet. Työn varsinainen alku oli käsiteltävien arvioijien lähdekoodin tutkiminen ja kurssien henkilökunnan haastattelu arvioijien käytöstä. Haastattelujen pohjalta koottiin lista nykyisistä käyttöongelmista ja työkaluun halutuista ominaisuuksista.

Keskeisimmät esille nousseet ongelmakohdat kurssien toteutuksissa on listattu alla:

- Kaikilla kursseilla useita kirjoittajia, paljon dokumentoimatonta koodia
- Kurssit käyttävät kahta eri toteutusta run.sh -skripteille

- Uusien testien lisääminen arvioijaan on haastavaa
- Arviointisäiliöiden toiminta osin epäselvää, arvioijan tilan tutkiminen ja debuggaaminen on haastavaa
- Arvioinnin tulosten muotoilu palautteessa

Vuosien kuluessa käsiteltäviä kursseja on ollut luomassa ja päivittämässä lukuisia henkilökunnan jäseniä, ja usein luodut muutokset ovat korjauksia sekä pieniä lisäyksiä olemassa olevaan toteutukseen. Näitä muutoksia ei olla niiden luomishetkellä dokumentoitu, ja ne ovat jääneet kurssien toteutuksiin toimivana koodina. Näin ollen vanhan koodin päivittämisessä nousee esiin ongelma, jossa vanhan toteutuksen logiikkaa ja käyttöohjeita ei olla kirjattu minnekään, jolloin uusien käyttäjien perehtyminen toteutuksiin ja niiden jatkokehitys vaikeutuu.

Käsiteltävillä kursseilla opiskelijoiden tekemät palautukset tarkistetaan run.sh -skriptien kautta. Näille on olemassa 2 erilaista toteutusta, jolloin uusia kursseja luodessa harjoitusten pohjaksi voitaisiin valita kumpi tahansa. Tämä kasvattaisi entisestään kurssien eroja ajan kuluessa, jos uudet kurssit käyttävät satunnaisesti toista näistä rakenteista. Uusista run.sh -skripteistä kerrotaan lisää luvussa 5.3.2.

Kurssihenkilökunnan haastatteluissa [5, 7] ilmeni ongelmia uusien tehtävien ja testien lisäämisessä kurssitoteutukseen. Yleisesti uutta tehtävää kurssille lisättäessä sillä tulee olla valmis toimiva testitoteutus, joka sisältää halutun joukko testejä, testien tarvitsemat tiedostot ja kirjastot, sekä toimivat tarkistuskomennot run.sh -skriptissä. Testit ja niiden tiedostot siirretään kurssipohjan exercises-kansion sisälle omaan kansioonsa, ja tehtävän tarkistusta varten run.sh -skriptin pitää osata ajaa palautetulle koodille testit ja tallentaa niiden tulokset. Lisäksi testien tarvitsemat kirjastot tulee lisätä tarkistussäiliöön asennettaviin kirjastoihin.

Arviointisäiliöt itsessään ovat tarkistuspalvelimella ajettavia Docker-säiliöitä, jotka ovat palvelimesta eristettyjä erillisiä instansseja pohjautuen säiliön "kuvaan"(engl. Docker image) [31]. Tästä seuraa, että kaikki säiliön sisällä tapahtuvat operaatiot ja niiden ulostulot, kuten virhetulosteet, jäävät säiliön sisälle, ja lopulta katoavat kun säiliö itsessään lopetetaan. Säiliöiden debuggaamisen kannalta tämä on haastavaa, sillä säiliössä tapahtuvia operaatioita ei voi tarkastella säiliön ulkopuolelta, ja väärin toimivien tai säiliön kaatavien testien viat jäävät piiloon. Säiliöiden toiminnan tutkimiseen palataan myöhemmin tässä luvussa kohdassa 5.4.2.

Yksittäisen tehtävän arviointiprosessin lopussa tehtävälle suoritettujen testien tulokset palautetaan säiliöstä Tuni+:aan. Tuni+:n palauteikkuna on tyypillinen HTML-popup-ikkuna, joka tukee CSS-muotoilua. Arviointisäiliöstä tulevan palautteen tulee kuitenkin olla HTML-muodossa, eli kaikki muotoilut, kuten värikorostukset tai fonttikoot joudutaan liittämään palautetekstiin arviointisäiliössä ennen sen lähettämistä Tuni+:aan.

Edellä listattujen ongelmien pohjalta ideoitiin työkalu, joka varmistaa että uudet Tuni+ -kurssit noudattavat yhtenäistä arkkitehtuuria, ja että kurssin luontiin ja päivittämiseen tarvittavat toiminnot ja työkalut on selkeästi dokumentoitu. Luotavalle työkalulle ideoitiin seuraavia ominaisuuksia:

- Heti valmis pohja kursseille, jonka voisi siirtää sellaisenaan suoraan Plussaan
- Testipohjat erilaisille tehtäville
- Tehtäväpankki olemassaoleville ja uusille tehtäville
- Uudet yhtenäiset toteutukset run.sh -skripteille OHJ3- sekä BWA-kursseille
- Itsedokumentoiva työkalu, ohjeistus tehtäville ja run.sh -skripteille
- Uusi pohjakuva tehtävien käyttämille Docker-säiliöille, jonka päälle ne voidaan rakentaa ja kustomoida

Työn edetessä ominaisuuksia jouduttiin karsimaan, ja uusia tarpeita ilmeni loppukäyttäjien kanssa käytävissä keskusteluissa. Kurssien tehtäville luotava tehtäväpankki todettiin tarpeettomaksi, sillä se sisältäisi vain tarpeettomasti kopioitua koodia, eivätkä toisten kurssien tehtävät olisi sellaisenaan helposti siirrettävissä muille kursseille. Kursseilla käytettäviin tehtävien arviointisäiliöihin liittyen nousi esiin ongelmia ylläpidettävyydessä ja toteutuksen iässä, joihin palataan myöhemmin luvussa 5.4.

Ohjelmointi 3 - sekä Basic Web Applications -kurssien kurssipohjat analysoitiin diplomityön alussa, ja havaittiin, että käytetyt arviointiskriptit käyttävät paljon samaa koodia. Diplomityön konstruktiivisessa osassa algoritmia 2.2 on hyödynnetty muokattuna versiona. Molempien kurssien arviointiskriptit prosessoitiin seuraavalla diplomityön tarkoituksiin sovelletulla algoritmilla:

```
1 Vaihe 1: Kokoa yhden kurssin kaikki run.sh-skriptit
2          vertailtaviksi keskenään.
3 Vaihe 2: Käytä tekstipohjaista tunnistusta skriptien kesken.
4 Vaihe 3: Jos skripteissä on tarpeeksi suuria yhteneväisyyksiä
5          toisiinsa nähden, siirry vaiheeseen 4, muussa
6          tapauksessa siirry vaiheeseen 6.
7 Vaihe 4: Kirjaa havaitut kloonit ylös ja kategorisoi ne.
8 Vaihe 5: Refaktoroi tai poista kloonit.
9 Vaihe 6: Lopeta
```

Algoritmi 5.1. *Kurssien arviointiskripteihin sovellettu algoritmi.*

Algoritmia sovellettaessa tekstipohjainen tunnistus suoritettiin manuaalisesti vertaamalla skripteissä käytettyjä komentoja toisiinsa. Tämä johtuu siitä, että skriptien käyttämälle shell-kielelle ei löytynyt valmiita klooninhavaitsemisratkaisuja, joihin skriptitiedostot olisi voitu syöttää. Lisäksi vertailtavan koodin määrä ja laatu oli sopiva ihmisen tekemälle ver-

tailulle. Manuaalisesti tehtynä skripteistä tunnistettiin myös riippuvuuksia tiettyjen arvojen suhteen, ja näitä riippuvuuksia on hyödynnetty skriptien refaktoroinnissa.

Käytetyn algoritmin 5.1 tuloksena havaittiin kaksi selkeää kloonikokonaisuutta: arvioitavan palautuksen hakeva git clone-lohko, sekä tietyille samantyyppisille tehtäville suoritettavien arviointikomentojen lohkot. Kuvassa 5.1 on Ohjelmointi 3 -kurssin vanhojen run.sh -skriptien käyttämä ohjelmalohko, joka hoitaa opiskelijan antaman git-repositorion kloonaamisen arviointisäiliöön. Ohjelmalohkon koko on 23 riviä, ja se toistuu kaikissa kurssin 11 erillisessä arviointiskriptissä joko kokonaan tai osittain.

```
submissionurl=`cat /submission/user/gitsource` # The student's submitted string is in this file
cloneurl=`sed 's/https:\/\/course-gitlab.tuni.fi\//git@course-gitlab.tuni.fi:/g' <<< $submissionurl`

export GIT_SSH_COMMAND="ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -i /exercise/sandbox/secrets/id_rsa";
git clone $cloneurl /palautushakemisto/

if [ $? -ne 0 ]; then
    echo "Cannot clone $cloneurl " >> /feedback/err
    echo "Are you sure the repository exists *and* is correct for this course?" >> /feedback/err
    grade
fi

cd /palautushakemisto

echo "All tags in your repository:" >> /feedback/out
pre git tag -l >> /feedback/out

git tag -l $CURRENT_TAG --sort=creatordate > matching_tags
if [ ! -s matching_tags ]; then
    echo "Couldn't find matching tags from your repository. Tried to find $CURRENT_TAG. Aborting." >> /feedback/out
    grade
fi

echo "All matching tags for this submission in your repository:" >> /feedback/out
pre cat matching_tags >> /feedback/out


latest_matching_tag=`tail -n 1 matching_tags`
echo "Checking out tag $latest_matching_tag:" >> /feedback/out

git checkout $latest_matching_tag 1>& git_checkout_msg
pre tail -n 1 git_checkout_msg >> /feedback/out
```

Kuva 5.1. *Ohjelmointi 3 -kurssin run.sh -skriptien git clone -lohko.*

Löydetyistä koodiklooneista skriptien käyttämä git clone -lohko edustaa sanatarkkaa kloonia, sillä sen sisältämät komennot ovat sellaisenaan useassa skriptissä. Skripteistä löytyneet samanlaiset arviointilohkot on toteutettu muuttamalla tiettyjä komentojen käyttämiä arvoja, eli ne kuuluvat parametrisoituihin klooneihin. Molempien kurssien skriptit sisälsivät näitä klooneja. Kloonit refaktoroitiin molempien kurssien tapauksessa yhdistämällä niitä käyttävät skriptit yhteen run.sh -tiedostoon. Yhdistäminen tapahtui aloittamalla kumpikin skripti kurssin kaikkien tehtävien käyttämällä git clone -lohkolla, jonka jälkeen eri tehtävien tarkistuskomennot on asetettu lohkoina tämän perään. Eri tehtävien käyttämät lohkot on eritelty shell-skriptikielen if-rakenteella niin, että jokainen lohko suoritetaan vain, jos tehtävään kuuluvasta *config_tests.txt*-tiedostosta löytyy kyseistä lohkoa kuvaava merkkijono. Tämä takaa sen, että vain tehtävälle kuuluvat komennot ajetaan, ja että tehtävälle voidaan toisaalta antaa myös useampi eri tyyppi merkkijonoilla, jolloin tehtävät voidaan luoda aiempaa monimuotoisemmiksi lisäämättä tarkistukseen kuuluvien tiedostojen määrää.

5.3 CourseCreator -ohjelma

Kurssilla käytettävien arvioijien tutkimisen tuloksena syntyi CourseCreator-niminen Python-ohjelma, jolla käyttäjä voi luoda haluamansa mukaisen arvioijan ja tehtäviä kursseille. Ohjelma käyttää hyödyksi kurssimateriaalien rst-tiedostoja sekä tehtävien yaml-määrittelytiedostoja, jotka konfiguroidaan suoraan ajettaviksi arviointisäiliössä.

CourseCreatorin tekninen toteutus on yksinkertainen graafisella tkinter-rajapintakirjastolla [38] rakennettu asennusohjelma, joka kysyy käyttäjältä arvioijan ja tehtävien parametreja dialogina tkinterin frame-komponenttien sisällä. Käyttäjän syöttämät valinnat tallennetaan ja niiden avulla luodaan käyttäjän varmistuksen jälkeen kurssirunko tai harjoitustehtävän konfiguraatio. Käyttöliittymä on rakennettu siten, että käyttäjä voi liikkua dialogi-ikkunoissa taaksepäin esim. virheen tai muutosten takia, ja valmiin asennusprosessin jälkeen käyttäjä voi luoda uusia harjoituksia tarvitsemansa määrän.

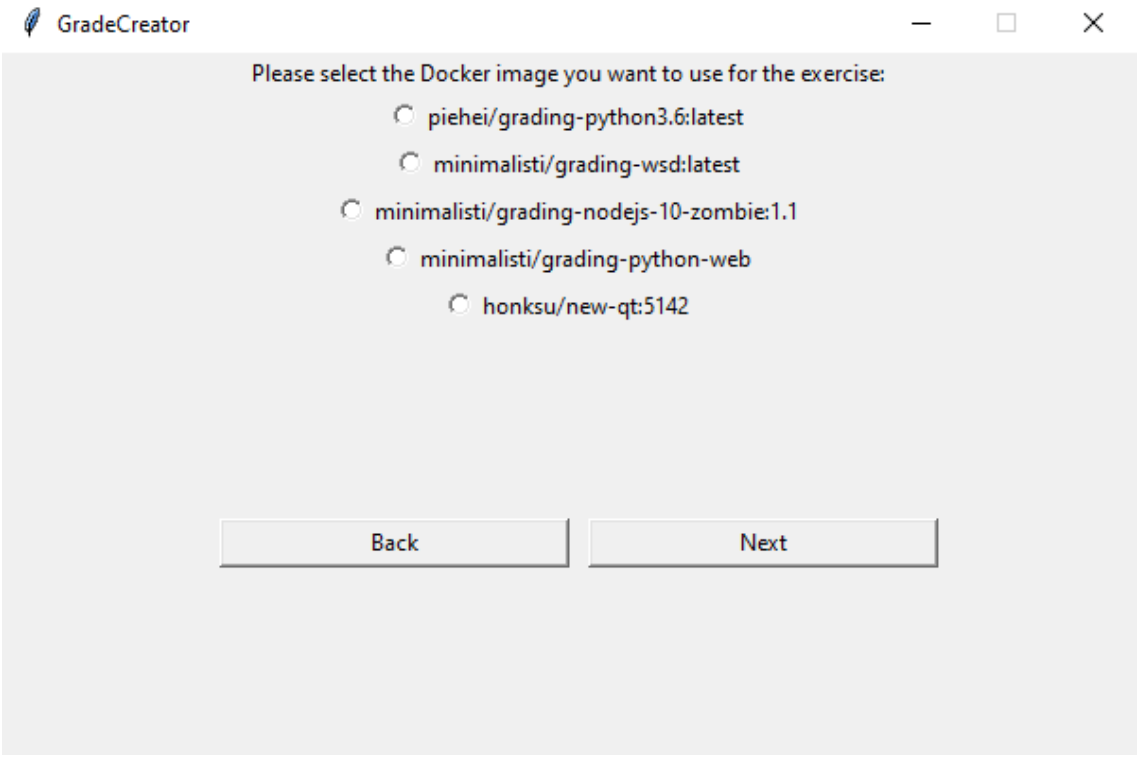
Toteutuksen alustaksi valittiin Python, koska käytetty tkinter-rajapintakirjasto on saatavilla Pythonin vakioasennuksessa, ja sen käyttämä graafisten komponenttien Tcl/Tk-kirjasto [39] on saatavilla yleisimmille käyttöjärjestelmille. Näin ollen työkalun käyttöönotto on mahdollisimman yksinkertaista sen vaatiessa vain Git-versionhallinnan. Python on myös ohjelmointikielenä rakenteensa ja yleisyytensä vuoksi helposti luettavissa ja opeteltavissa [40, s.3-4], ja työkalun jatkokehitys on näin mahdollista useimmille tietotekniikan henkilökuntaan kuuluville.

Coursecreator on koottu erilliseksi Git-repositorioksi Tampereen yliopiston Gitlab-palveluun, josta työkalun voi käynnistää create.sh-skriptillä. Itse työkalun lähdekoodi on jaettu päätiedostoon sekä erilliseen frames.py -tiedostoon, jossa ovat kaikki työkalun käyttöliittymäikkunat ja niiden toiminnallisuus. Työkalun tarvitsemat pohjatiedostot on sijoitettu *Installation_files* -kansioon, josta ne kopioidaan oikeaan paikkaan ohjelmaa käytettäessä ja täydennetään käyttäjän valintojen mukaan.

5.3.1 Kurssisisällön ja harjoitusten alustus CourseCreatorilla

CourseCreatorin ensimmäinen päätoiminnallisuus on kurssilla käytettävän materiaalin rungon luominen. Käyttäjä voi valita työkalun avulla haluttujen "kurssikierrosten"määrän, ja työkalu luo arvioijan rounds-hakemistoon valitun määrän valmiita rst-tiedostopohjia. Coursecreatorin dokumentaatio sisältää ohjeet referenssilinkkien kirjoittamiseen rst-sisältöön sekä kuvien liittämiseen Tuni+:ssa näkyviin sivuihin.

Toinen päätoiminnallisuus CourseCreatorissa on harjoituksen rungon luonti arvioijaan. Kuvissa 5.2, 5.3 ja 5.4 esitellään GraderCreatorin ikkunoita uutta harjoitusta luotaessa. Käyttäjä antaa harjoitukselle haluamansa nimen, palautustyypin, arvioijan käyttämän Dockerkuvan, sekä tarvittavat polut mikäli palautettavaa koodia käännetään, jonka jälkeen

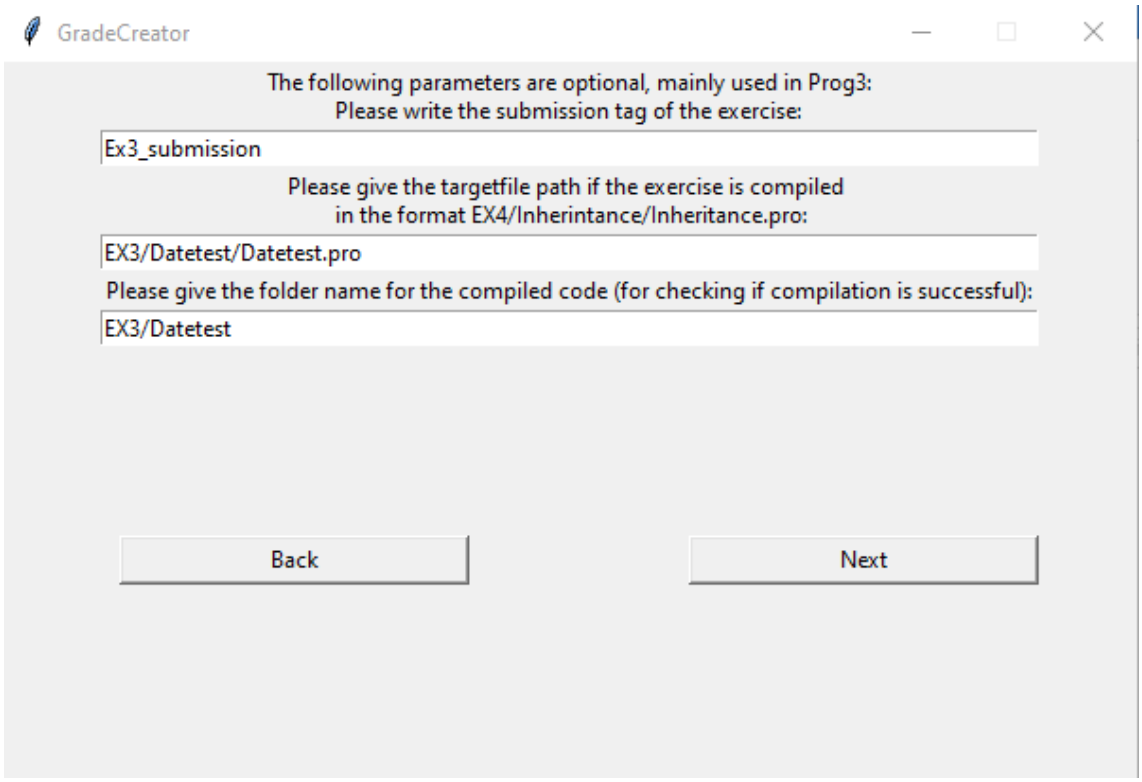


Kuva 5.2. *Uuden tehtävän luonnissa valittavat ohjelmointikurssien käyttämät Docker-kuvat.*

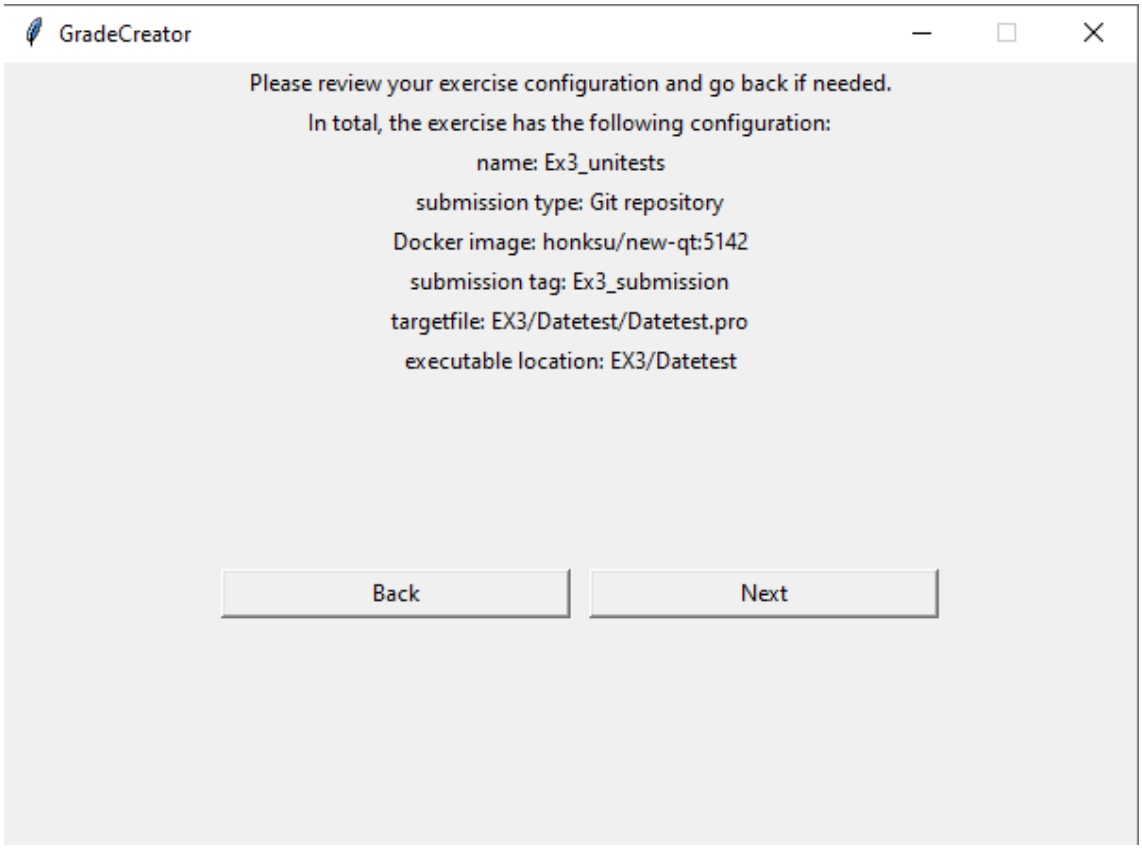
exercises-hakemistoon luodaan nimeä ja asetuksia vastaava yaml-tiedosto. Kuvan 5.3 parametrit on tarkoitettu pääasiassa Ohjelmointi 3 -kurssille, jossa palautettavaa koodia voidaan joutua kääntämään. Nämä parametrit on koostettu edellä esiteltyjen koodikloonien tutkimismenetelmien perusteella.

Kuvassa 5.2 on listattu aikaisemmin esiteltyjen ohjelmointikurssien käyttämät Docker-kuvat. Tämän tarkoituksena on tarjota helppo valinta uusia tehtäviä luodessa, kun henkilökunta tietää, mitä ohjelmointikieltä tehtävässä käytetään, ja minkä tyyppinen tehtävä on. Tätä listaa tulee kuitenkin ylläpitää aktiivisesti, sillä käytettyihin Docker-kuviin saattaa tulla uusia versioita, ja uusia tehtäviä ja kursseja varten voidaan joutua rakentamaan kokonaan uusia Docker-kuvia.

Kuvassa 5.4 on GraderCreatorin tehtäväluonnin viimeinen ikkuna, jossa esitetään käyttäjän antamat tiedot tehtävälle. Ikkunan tarkoitus on antaa mahdollisuus tarkistaa, että kaikki tiedot ovat varmasti oikein, ja ikkunasta pääsee palaamaan takaisin edellisiin valintaikkunoihin tarpeen vaatiessa. Mikäli asetukset ovat oikein, käyttäjä jatkaa tehtävän luontiin, jolloin Basegrader-hakemiston sisällä olevaan exercises-hakemistoon luodaan uusi hakemisto tehtävän nimellä, ja kuvan 5.5 esittämä yaml-tiedosto kirjoitetaan sinne. Tämän jälkeen kurssihenkilökunta lisää harjoituksen hakemistoon tarvittavat testi- ja muut kooditiedostot, joita run.sh -skripti kutsuu. Tähän kuuluu myös *config_tests.txt*-tiedoston täydennys, jossa määritellään minkä tyyppinen luotu harjoitus on, eli mitä testejä run.sh



Kuva 5.3. *Uuden tehtävän luonnissa valittavat käännösparametrit.*



Kuva 5.4. *Uuden tehtävän vahvistusikkuna GraderCreatorissa.*

-skripti suorittaa palautuksille.

```yaml
---
# Author: Tommi Honkanen
# Course instances: Example exercise yaml file

template: ./exercises/exercise.html

title: The title of the exercise

learninggoals: |

  Write the learning goals of the exercise here in plain text.

instructions: |
# This is a section where the instructions for the student are written. The actual format is HTML.

view_type: access.types.stdasync.acceptGitAddress
require_gitlab: course-gitlab.tuni.fi

container:
  image: honksu/new-qt:5142
  mount: exercises
  cmd: /exercise/sandbox/run.sh Ex3_unitests Ex3_submission EX3/Datetest/Datetest.pro EX3/Datetest

```

Kuva 5.5. *GraderCreatorilla luotu uuden tehtävän yaml-konfiguraatiotiedosto.*

Coursecreatorilla luotu ja tarvittavilla kurssisällöllä ja tehtävien testitiedostoilla täydennetty valmis kurssipohja toimii kuten tämänhetkiset Tuni+:ssa olevat kurssit. BaseCourse-kansion sisältö voidaan siirtää olemassaolevaan kurssirepositorioon Tampereen yliopiston Gitlab-palveluun, josta se siirtyy puskuoperaation jälkeen julkiseksi kurssiksi Tuni+:aan.

Keskeisin yhtenäistämisen kohde diplomityössä oli arvioijien käyttämät run.sh -skriptit, jotka vastaavat tehtävien arvioinnin kulusta. Käsiteltävien kurssien toteutukset analysoitiin, ja niiden pohjalta päädyttiin luomaan kaksi uutta pääskriptiä Ohjelmointi 3- sekä Basic Web Applications-kursseille. Aikaisempaan verrattuna tehtävien tarvitsemat skriptit löytyvät nyt yhdestä paikasta, ja tehtävän arviointiin tarvittavat tiedot voi lisätä suoraan tehtävän yaml-tiedostoon työkalun kautta.

5.3.2 Uudet run.sh -skriptit

Kun Ohjelmointi 3 -kurssin sekä Basic Web Applications -kurssin run.sh -skriptejä analysoitiin, huomattiin että skripteissä esiintyy ohjelman 2.1 tavoin kovakoodattuja arvoja, jotka voidaan muuttaa muuttujiksi.

Ohjelmointi 3- sekä Basic Web Applications -kurssien vanhat run.sh -skriptit ja niille kuuluvat yksittäiset tehtävät voidaan ajatella erillisinä luokkina, joilla on spesifit arvot, kuten tehtävän nimi ja palautukseen liittyvä git-tagi. Uusia yhtenäisiä run.sh -skriptejä varten tehtävät tulee yleistää niin, että arvioinnin tarvitsemat arvot välitetään tarkistusprosessin alussa skripteille, jolloin ne voivat arvojen perusteella päätellä, mikä tehtävä on kyseessä ja suorittaa arvioinnin.

Taulukossa 5.1 on listattu Ohjelmointi 3 -kurssin uudessa run.sh -skriptissä käytetyt uudet parametrit. Listatut esimerkkiarvot liittyvät vanhan toteutuksen *run_ex3.sh*-tiedostoon, jossa skripti suorittaa kolmannen viikkoharjoituksen tarkistuksen käyttämällä näitä ar-

Taulukko 5.1. *Ohjelmointi 3 -kurssin parametrisoidut muuttujat uudessa run.sh -skriptissä.*

parametrin nimi	esimerkkiarvo(merkkijono)	muuttujan tarkoitus
EXERCISE	EX3	käsiteltävän harjoituksen hakemiston nimi
CURRENT_TAG	EX3_submission	käsiteltävän harjoituksen käyttämä git-tagi
TARGETPATH	EX3/Datetest/Datetest.pro	polku harjoituksen .pro-tiedostoon kääntämistä varten
COMPILEDFOLDER	EX3/Datetest/Datetest	polku käännettyyn ohjelmaan

voja sellaisenaan skriptissä. Nämä arvot voidaan kuitenkin välittää skriptille tehtävän YAML-konfiguraatiotiedoston kautta, jolloin ne voidaan skriptin sisällä ottaa talteen shell-syntaksilla `EXERCISE=$1`, eli ensimmäinen skriptille annettu parametri sijoittuu muuttujaan *EXERCISE*. Tästä seuraa, että muuttamalla vanhan skriptin vanhoja kovakoodattuja arvoja käyttävät komennot käyttämään tällaisia muuttujia, skripti kykenee tarkistamaan minkä tahansa nimisen ja minkä tahansa kansio- ja tiedostorakenteen sisältävän tehtävän, kunhan nämä tiedot ovat tehtävän YAML-konfiguraatiotiedostoa kirjoittavan tiedossa. Basic Web Applications -kurssi käyttää myös *EXERCISE*-parametria uudessa run.sh -skriptissään.

Työtä käsittelevien ohjelmointikurssien repositorioita analysoitaessa huomattiin, että Ohjelmointi 1 ja Ohjelmointi 2 -kurssit noudattavat samanlaista arkkitehtuuria harjoitusten ja run.sh -skriptin suhteen. Molemmilla kursseilla on käytössä yksi run.sh -skripti, joka käsittelee kaikki kurssilla arvioitavat tehtävät. Tämä vähentää tarvittavien run.sh -skriptien määrää riippuen kurssin tehtävätyyppien määrästä, ja siirtää kaikkien tehtävien tarkistukseen tarvittavan logiikan yhteen paikkaan.

Tämän pohjalta päädyttiin luomaan kaksi uutta pääskriptiä Ohjelmointi 3 - sekä Basic Web Applications -kursseille. Näiden kurssien tehtävien tarkistuskomennot erillisissä run.sh -skripteissä on siirretty yhteen tiedostoon, josta ne voidaan ajaa tarkistussäiliössä. Vanhojen skriptien koodikloonit on refaktoroitu siten, että ne alkavat kaikille tehtäville yhteisellä git clone -lohkolla, jonka jälkeen eri tehtävien arviointikomennot on eritelty erikseen lohkoiksi. Skriptin tulee kuitenkin tietää, minkä tyyppistä tehtävää ollaan arvioimassa palautuksen yhteydessä, joten tehtäviin on liitetty uusi *config_tests.txt*-tiedosto, jossa määritetään tehtävälle jokin tyyppi yksinkertaisena merkkijonomuuttujana. Esimerkkinä *GITLAB_CI* on yksi Ohjelmointi 3 -kurssin tehtävistä. *Config_tests.txt*-tiedostot ovat käytössä laajemmin Ohjelmointi 1 -kurssilla, jossa tiedostoihin voidaan määrittää myös mm. tehtävistä saatavat pisteet ja muita parametrejä tarkistusta varten.

Taulukko 5.2. *Ohjelmointi 3 - sekä Basic Web Applications -kurssien tehtävätyypit.*

Tehtävätyyppi(merkkijono)	tehtävätyypin kurssi	tehtävän sisältö
ASYNC	OHJ3	asynkroninen harojitustyön palautus
SOLO	OHJ3	henkilökohtainen C++-tehtävä
GROUP	OHJ3	ryhmätyönä suoritettava C++-tehtävä
GITLAB_CI	OHJ3	Gitlabin CI-putken toiminta
GIT	OHJ3	Gitin toiminta ja ominaisuudet
DJANGO	BWA	Django-ohjelmointikehyksen toiminta
CSS	BWA	CSS-määrittelyn toiminta
HTML	BWA	HTML-sivujen toiminta
JS	BWA	Javascriptin toiminta
JS_OBJECTS	BWA	objektit Javascriptissä
NODE_ZOMBIE	BWA	node.js:n toiminta
JSONP_SEARCH	BWA	JSON-formaatin käyttö
PYTHON_KEYWORDS	BWA	Python-funktiot
PYTHON_CIRCLE	BWA	matemaattiset operaatiot pythonilla
PYTHON_SONG	BWA	suurten datamäärien käsittely Pythonilla

Jokainen tehtävätyyppi uusissa skripteissä on eritelty oman nimimuuttujansa alle, jolloin run.sh -skripti tarkistaa, mitkä tyypit tehtävälle on annettu *config_tests.txt*-tiedostossa, ja suorittaa vain kyseisille tehtävätyypeille määrättyjä komentoja. Tehtävätyypit on listat-

tu taulukossa 5.2. Tehtävätyypit on johdettu kurssien tehtävien nimistä sekä yleisestä sisällöstä. Tehtävien selkeä erittely tällaisella muuttujalla on pakollista, jotta tietylle palautetulle tehtävälle saadaan ajettua juuri kyseisen tehtävän komennot. Mikäli kursseille luodaan uusia tehtäviä, jotka vaativat uusia komentoja run.sh -skriptiin, ne voidaan lisätä omana lohkonaan skriptin sisälle ja erottaa muista shell-skriptikielen `if grep -q <tehtävätyyppi>` -muotoisella ehdolla, joka etsii annettua merkkijonoa tietystä tiedostosta, tässä tapauksessa tehtävän tyyppiä *config_tests.txt*-tiedostosta.

5.4 Uudet tehtävien arviointisäiliöt

Työn edetessä keskeneräisen työkalun esittelyssä nousi esiin tarve uusille Docker-säiliöille tehtävien arviointia varten. Nykyiset säiliöt on rakennettu hyvin nopeasti PoC (engl. Proof of Concept) -tasolla, ja ne on otettu sellaisenaan käyttöön niiden toimiessa tarpeeksi kattavasti. Kuitenkin esim. Ohjelmointi 3- kurssin käyttämä säiliö on kooltaan yli 10 gigatavua, mikä tarkoittaa usean sadan hengen kurssilla valtavaa palvelinkuormitusta opiskelijoiden palauttaessa tehtäviä, kun näitä säiliöitä ajetaan suuria määriä rinnakkain. Lisäksiaikaisemmin kurssien tarvitsemat Docker-säiliöt oli luotu käytännössä henkilökunnan yksityisiin Dockerhub-repositorioihin, jotka mahdollistavat Docker-säiliöiden käytön, mutta niiden rakentaminen uudestaan tai muokkaaminen Dockerfile-tiedostojen [32] kautta ei ole mahdollista, sillä nämä tiedostot on asetettu yksityisiksi.

Ratkaisuna ilmenneeseen ongelmaan luotiin uusi pohja Tuni+:ssa käytettäville tehtävien arviointi- säiliöille, jota voidaan käyttää uusien säiliöiden pohjana. Pohjasäiliö on käytännössä Tuni+:aa varten muokattu versio Aallon omasta grading-base-säiliöstä [41], joka sisältää Aallon luomat gradeutils-työkalut [42]. Tämän pohjan päälle voidaan Dockerilla rakentaa uusi säiliö, johon asennetaan tarvittavat kirjastot ja komponentit vanhoille ja uusille tehtäville. Uudesta arviointisäiliöiden pohjasta kerrotaan lisää tämän luvun kohdassa "Arviointisäiliön rakentaminen".

Merkittävin uudistus kurssien säiliöiden suhteen on Ohjelmointi 3 -kurssille rakennettu Qt-kehyksen versio 5.14.2:n [43] sisältävä säiliö, jonka koko tuotanto-olosuhteissa on 2,4 gigatavua. Tämä on yli 75 prosentin pienennys säiliön tarvitsemaan kiintolevytilaan, mikä helpottaa Tuni+:n resurssitarpeita, jotka aikaisemmin saattoivat kuormittua liikaa kurssitehtävien palautuksien suhteen.

Muutos säiliöiden koossa on selitettävissä Qt:n asennuksessa valittavilla asennuslipuilla, sekä asennuksen jäljiltä säiliöön jäävillä asennustiedostoilla. Qt:n vakioasennus sisältää komponentteja kuten testiesimerkit, joita ei tarvita itse arviointiprosessissa tehtävän ollessa tarkistuksessa säiliössä, ja asennukseen tarvittat mutta itse Qt:n toiminnan kannalta tarpeettomien asennustiedostot jäävät normaalisti säiliön sisälle. Nämä poistamalla tarpeettomina uusi säiliö on paremmin skaalautuva Tuni+:n palvelimille, kun arvioitavia palautuksia käsitellään paljon yhtäaikaisesti.

5.4.1 Arviointisäiliön rakentaminen

Tampereen yliopiston Tuni+-kursseja varten päätettiin rakentaa uusi säiliö, jota voidaan käyttää uusien kurssien ja vanhojen kurssien päivittämistä varten pohjana niitä varten tehtäville varsinaisille tarkistussäiliöille. Docker-kuvan kirjoitus alkaa FROM-komennolla, joka määrittää mitä jo olemassa olevaa kuvaa uudessa kuvassa käytetään alustana. Tämän jälkeen käytetyn pohjakuvan päälle voidaan tehdä muutoksia, kuten uusien kirjastojen tai ohjelmien asennus, ja luoda näin uusi säiliökuva jossa yhdistyvät vanhan kuvan ominaisuudet sekä uudet muokkaukset.

```
1  FROM ubuntu:bionic
2
3  ENV LANG C.UTF-8
4
5  COPY rootfs /
6  COPY bin /usr/local/bin
7
8  RUN apt-get -y upgrade
9  RUN apt-get -y update
10
11 RUN apt-get -y install software-properties-common apt-utils
12 RUN add-apt-repository main
13 RUN add-apt-repository universe
14 RUN add-apt-repository restricted
15 RUN add-apt-repository multiverse
16
17 RUN apt-get -y upgrade
18 RUN apt-get -y update
19
20 RUN add-apt-repository -y ppa:deadsnakes/ppa
21 RUN apt-get -y install python3.8 python3-pip
22
23 # Build prerequisites
24 RUN apt-get -y install g++ xz-utils make perl gcc binutils pkg-config zip git vim tmux
25
26 [redacted]
27
28
29 RUN mkdir -p /feedback /submission /exercise
30 RUN chmod 0770 /feedback
31 RUN usermod -d /tmp nobody
32
33 WORKDIR /submission
34 ENTRYPOINT ["/gw"]
35 CMD ["bash"]
```

Kuva 5.6. *Uusi arviointisäiliöiden pohjan Dockerfile.*

Kuvassa 5.6 on uusi Tuni+:ssa käytettävien arviointisäiliöiden pohja. Dockerfilen alussa olevat kaksi COPY-komentoa siirtävät Tuni+:n käyttämät apukomennot ja niiden tarvitsemat tiedostot säiliöön oikeille paikoille. Tällä varmistetaan, että run.sh -skripteissä hyödynnettävät komennot kuten *grade* ovat skriptien käytettävissä säiliön sisällä tarkistuksen aikana. Tiedostossa olevat `RUN apt-get -y install` -komennot ovat yleishyödyllisten kirjastojen asennusta varten, joita monet kurssien tehtävät vaativat. Kuvasta on sensuroitu pääkäyttäjän salasanan asettaminen, jolla säiliötä pystyy hallitsemaan tarvittaessa kuten Linux-pääkäyttäjä.

Uusi Ohjelmointi 3 -kurssin arviointisäiliö pohjautuu Tampereen yliopiston vanhojen työntekijöiden Heinon [44] ja Venttolan [45] luomille säiliöille, joita on käytetty aikaisemmin kurssilla. Kurssin tärkeimmät komponentit arviointisäiliössä ovat Aallon luomat gradeutils-

työkalut sekä itse Qt-ohjelmointikehys. Uusi arviointisäiliöiden pohjakuva sisältää työkalut jo valmiiksi, joten jäljelle jää itse Qt:n konfigurointi ja asennus. Tämä tapahtuu täysin samalla tavalla kuin asennuksen suoritus Ubuntu-käyttöjärjestelmän komentoriviltä, sillä uusi pohjasäiliö on rakennettu Ubuntu 18.0.4 -version päälle, joka pohjautuu Debian Linux-järjestelmälle. Esimerkiksi Qt-ohjelmistopaketin lataaminen onnistuu asentamalla säiliöön ensin wget-paketti [46], jolla voidaan ladata verkossa jaettavia paketteja tietokoneen kiintolevylle, tässä tapauksessa arviointisäiliöön. Dockerfilessa tämä tapahtuu komennolla `RUN wget <paketin_osoite>`, jonka jälkeen ladattu linux-paketti voidaan purkaa tar-työkalun avulla komennolla `RUN tar xf <ladatun_paketin_nimi>`.

5.4.2 Arviointisäiliön virheenjäljitys

Ohjelmointikurssien harjoitustehtäviä luotaessa ja valmista kurssia testattaessa kurssihenkilökunnan on varmistuttava siitä, että käyettävät arviointisäiliöt toimivat oikein sekä itse tehtävän arvioinnin että Tuni+:n kannalta. Arviointisäiliön pitää pystyä vastaanottamaan opiskelijan palautus joko yksittäisenä tiedostona tai git-repositoriona annetusta osoitteesta, ja ajamaan tehtävälle kuuluvat testit onnistuneesti. Näiden testien tulokset tulee myös saada oikeassa muodossa takaisin Tuni+:an, josta opiskelija näkee suorituksensa tilan. Tämän varmistamiseksi arvointisäiliöille täytyy pystyä suorittamaan virheenjäljitystä, jotta ongelmatilanteet ja säiliöiden toiminta voidaan korjata tarvittaessa.

Tuni+:n käyttämiä Docker-säiliöitä voidaan ajaa sellaisenaan paikallisesti komennolla `docker run -it -entrypoint sh <kuvan_nimi>`, joka käynnistää tietyn nimiseen Docker-kuvaan pohjautuvan säiliön ja sen sisällä bash-komentorivin. Säiliötä voi tämän jälkeen operoida kuten Linux-käyttöjärjestelmällä varustettua tietokonetta, ja siellä voi esimerkiksi testata Gitin toimintaa kloonaamalla tunnetun repositorion säiliöön tai käynnistämällä tiettyjä säiliössä olevia sovelluksia. Automaattisen arvioinnin testauksessa tehtävästä saatava palaute voidaan myös ottaa käyttöön arviointiprosessin virheenjäljitystä varten, sillä palautteen koostamisessa käytettyihin tiedostoihin voidaan kirjoittaa myös mm. komentojen paluuarvoja tai välitulostuksia run.sh -skriptin sisällä. Näin voidaan selvittää, mikä virhetilanne tietyn komennon suhteen ilmenee ja suorittaako skripti tiettyjä komentoja ollenkaan.

6. TOTEUTETUN JÄRJESTELMÄN ARVIOINTI

Tässä luvussa tutkimme, kuinka hyvin toteutus vastaa luvussa 1 esitettyyn tutkimuskysymykseen: ”Miten Tuni+:ssa olevien ohjelmointikurssien automaattisia arvioijia voidaan yhtenäistää?” Millaisilla työkaluilla ja käytännöillä tätä voidaan edesauttaa?”. Tässä luvussa arvioimme seuraavia seikkoja:

1. Miten automaattisia arvioijia voidaan yhtenäistää?
2. Millaisilla työkaluilla arvioinnin yhtenäistämistä voidaan edesauttaa?
3. Millaisilla käytännöillä arvioinnin yhtenäistämistä voidaan edesauttaa?

Baxter et al. ovat laatineet ISO/IEC 9126-1:2001 -standardiin [47] perustuvan arviointikriteeristön [48], jolla ohjelmistojen eri osa-alueiden laatua voidaan arvioida. Tämän diplomityön tuloksiin eli rakennettuun työkaluun ja sen osiin sovelletaan seuraavia alaosia kriteeristöstä:

- **Dokumentaatio**: Kuinka kattava ja hyvin koostettu dokumentaatio työkalulla ja sen osilla on?
- **Opittavuus**: Kuinka helposti työkalun toiminta ja sen osat ovat opittavissa?
- **Testattavuus**: Kuinka helposti työkalun ja sen osien toiminnan oikeellisuus on testattavissa?
- **Muokattavuus**: Kuinka helposti työkalua ja sen osia voidaan muokata?

Työkalun osilla tarkoitetaan tässä tapauksessa sen hyödyntämiä uusia run.sh -skriptejä sekä automaattisen arvioinnin suorittavan Docker-säiliön pohjakuvaa. Nämä osat eivät suoranaisesti kuulu työkalun lähdekoodiin, mutta ovat tärkeitä työkalun hyödyntämiä toteutuksia, jotka kuuluvat kiinteästi ohjelmointikurssin Tuni+ -kurssipohjan toimintaan.

6.1 Automaattisten arvioijien yhtenäistäminen

Tässä diplomityössä luotiin uudet run.sh-skriptit Ohjelmointi 3- sekä Basic Web Applications -kursseille. Toteutuksissa on sovellettu luvuissa 2.4 ja 2.5 esitettyjä menetelmiä, joilla vanhat toteutukset on muokattu yhtenäisiksi kahden muun kurssin suhteen. Näitä toteutuksia on testattu integraatiotestauksella paikallisesti Linux-virtuaalikoneella niin, että kurssit ovat saatavilla localhost-osoitteessa virtuaalikoneella ja käyttävät uusia run.sh-

skriptejä. Kurssien tehtäviä on palautettu Tampereen yliopiston Gitlab-palvelussa olevasta testirepositoriosta kurssien sivuilla. Molemmilla kursseilla palautettiin tehtäviä testiympäristöön 3 kappaletta, ja tehtävien arviointi onnistui kaikissa tapauksissa. Täysin kattavan testauksen kannalta kurssien kaikki käytössä olevat tehtävät tulisi testata aluksi paikallisesti ja lopulta itse tuotantoympäristössä oikeana kurssina Tuni+:ssa. Tätä ei kuitenkaan tehty, sillä kurssien tehtävistä ei ollut saatavilla valmiita toimiviksi todettuja mallitoteutuksia. Tämän sijaan osassa tehtävistä käytettiin vanhoja omia toteutuksia, jos sellainen löytyi, sekä muutamaa uutta itse kirjoitettua toteutusta.

Testien perusteella voidaan todeta, että yhtenäistämisprosessi ohjelmointikurssien arvioijiin on ainakin osittain onnistunut. Uudet run.sh-skriptit pystyivät testeissä suorittamaan niille tarkoitetut tehtävät kuten aikaisemmat arvioijat, ja vanhoihin toteutuksiin verrattuna kaikki tarvittavat arviointikomennot löytyvät nyt yhdestä paikasta. Uudet run.sh -skriptit on dokumentoitu niin, että niiden toiminta ja tarvitsemat parametrit on esitelty skriptissä. Uusien skriptien käytön opetteleminen vaatii kuitenkin tietämystä shell-skriptikielestä, jolla kaikki arviointiin liittyvät toiminnot kuten testien suoritus ja palautteen kirjaus on toteutettu. Skriptien testattavuus puolestaan vaatii Tuni+:ssa toimivaksi todetun kurssipohjan, joka voidaan siirtää Tuni+:n testialueelle ja jonka kautta kurssin tehtäviä voidaan palauttaa kuten opiskelijat tekevät oikealla kurssilla tuotannossa. Testaus riippuu myös käytettävästä Docker-säiliöstä, jonka tulee sisältää kaikki tarpeelliset kirjastot ja muut komponentit tehtäviä varten. Skriptien muokattavuus on mahdollistettu selkeällä rakenteella ja osiin jaolla, ja uusien tehtävien lisäämiseksi kurssille voidaan otta mallia olemassa olevista tehtävistä.

6.2 Arvioinnin yhtenäistämisen työkalut

Tämän diplomityön tuloksena luotu CourseCreator-työkalu pyrkii tarjoamaan uuden yhtenäisen tavan Tuni+-kurssien rakentamiseen ja kurssien tehtävien alustamiseen. Työkalua on testattu Tampereen yliopiston Gitlab-palvelun avulla Tuni+:n testialueella, jossa kursseja voi testata ja tarkistaa ennen muutosten julkaisemista virallisessa Tuni+:ssa. CourseCreatorilla luotu kurssipohja ja tehtävien YAML-konfiguraatiotiedostot toimivat oletetusti Tuni+:ssa, eli työkalua voidaan soveltaa uusien kurssien kanssa, joita ollaan luomassa Tuni+:aan.

Työkalun käyttö on dokumentoitu ja jaettu selkeisiin osiin sen repositorion README.md -tiedostossa [49]. Työkalun käyttö on suoraviivaista, mutta sen opettelu vaatii tietämystä Tuni+:n arkkitehtuurista ja kurssipohjan toiminnasta. Työkalun testattavuus edellyttää, että käyttäjällä on mahdollisuus täydentää työkalulla luotava kurssipohja toimivaksi Tuni+ -kurssiksi, joka voidaan siirtä Tuni+:aan. Työkaluun voidaan lisätä uusia toiminnallisuuksia lisäämällä tarvittavia uusia kurssimuuttujia gradercreator.py -tiedostoon, ja lisäämällä näiden syöttämiseen tarvittavat käyttöliittymäikkunat frames.py -tiedostoon. Uudet toiminnal-

lisuudet voidaan lisätä harjoitusten ja kurssisisällön alustamisesta vastaaviin funktioihin. Työkalun jatkokehitys vaatii tietämystä Python-kielestä ja tkinter-kirjaston toiminnasta.

CourseCreatorin suurimmat ongelmat ovat täysin uusien kurssien kanssa, sillä ne vaativat lähes täysin alusta asti kirjoitetun run.sh-skriptin tehtävilleen, sekä testatut ja toimivat Docker-säiliöt tehtäville. Tällä hetkellä CourseCreatorilla voi luoda työssä luvussa 5 esiteltyjä 4 ohjelmointikurssia vastaavia kurssipohjia, jotka käyttävät kurssikohtaisia run.sh-skriptejä. Täysin uutta kurssia luodessa sen tarvitsemaa run.sh-skriptiä varten voidaan ottaa mallia näistä toteutuksista, mutta uutta kurssia varten on tarpeen käydä läpi dokumentaatio tarkistusprosessista ja tarvittavista skripteistä ja tiedostoista.

Toisena osana työkaluja luotiin uusi pohja Tampereen ylopiston ohjelmointikurssien käyttämille Docker-kuville, jotka vastaavat itse tehtäväpalautusten tarkistamisesta. Tätä pohjaa sovellettiin Ohjelmointi 3 -kurssin kanssa uuden tarkistussäiliön luonnissa, jossa Qt-ohjelmointikehyksen versio päivitettiin 5.9.6:sta 5.14.2:een. Uutta säiliötä on käytetty kurssin toteutuksella syksyllä 2020 onnistuneesti tehtävien arvioinnissa. Toistaiseksi käytetty Docker-kuva on diplomityön tekijän henkilökohtaisen Docker-tilin alla, mutta kuva on tarkoitus siirtää laitoksen yhteiselle Docker-tilille.

Uuden Docker-pohjakuvan dokumentaatio on hyvin rajallinen, mutta itse pohjakuvaa on voidaan hyödyntää uusissa arviointisäiliöiden kuvissa pelkällä Dockerin FROM-komennolla uuden kuvan Dockerfile-tiedostossa. Pohjakuvan käyttö ja muokattavuus edellyttävät tietämystä Dockerin sekä pohjakuvan perustana toimivan Linux-käyttöjärjestelmän toiminnasta. Uusien pohjakuvaan perustuvien arviointisäiliöiden testaukseen tarvitaan vähintään testattavan tehtävän lähdekoodi sekä arvioinnissa käytetyt testit, jotka voidaan siirtää säiliöön ja suorittaa sen sisällä.

6.3 Arvioinnin yhtenäistämisen käytännöt

Diplomityön luvussa 2.5 esiteltyjen menetelmien soveltaminen luvussa 5.2 esitetyin tavoin osoittautui toimivaksi käsiteltävien kurssien yhtenäistämisessä. Koodikloonien tutkiminen ja refaktorointi uusiin run.sh-skripteihin tarjosi uuden yhtenäisen mallin 4 ohjelmointikurssin kesken. Toisaalta uusien run.sh-skriptien käyttämät tehtävätyypit voitaisiin välittää skriptin sisälle myös tehtävän määrittelytiedoston kautta parametrina. Tämä selkeyttäisi toteutusta, ja vähentäisi tehtäville kuuluvien tiedostojen määrää. Lisäksi uusissa run.sh-skripteissä on vielä komentoja, jotka sisältävät parametrisoitavia arvoja. Näin ollen koodiklooneista ei olla päästy täysin eroon, mutta vanhaan verrattuna samanlaisten koodilohkojen määrä on vähentynyt.

Oleellisena osana uuden työkalun ja sen osien käytössä on niille tarjottu dokumentaatio. Tässä on paikoitellen puutteita, jotka vaikeuttavat niiden käyttöönottoa ja jatkokehitystä. Dokumentaation parantaminen on yksi yhtenäistämisen käytännöistä, johon tulisi kiinnit-

tää parempaa huomiota tulevaisuudessa.

6.4 Yhteenveto

Tässä luvussa esitettiin arvio siitä, kuinka hyvin yhtenäistetyt run.sh -skriptit sekä uusi CourseCreator-työkalu ja Docker-pohjakuva vastaavat tutkimuskysymyksiin. Arviointi perustuu Baxter et al:n laatimaan arviointikriteeristöön, jota sovelletaan toteutusten dokumentaatioon, opittavuuteen, testattavuuteen sekä mmuokattavuuteen.

Työkalulle ja sen osille suoritettujen testien perusteella voidaan todeta, että käytetyt menetelmät soveltuvat automaattisen arvioinnin yhtenäistämiseen. CourseCreator-työkalu kykenee luomaan käytettävän pohjan Tuni+:n ohjelmointikurssille, ja uudet yhtenäistetyt run.sh -skriptit tarjoavat ainakin osittain saman toiminnallisuuden kuin vanhat toteutukset. Uusi Docker-pohjakuva tehtävien arviointia varten voidaan todeta toimivaksi sen ollessa aktiivisesti käytössä Ohjelmointi 3 -kurssin tämänhetkisellä toteutuskerralla. Koska uusia run.sh -skriptejä ei olla testattu täysin kattavasti kaikilla nykyisillä tehtävillä, voidaan ne todeta PoC -tason toteutuksiksi, joilla on testattuna ja parannettuna edellytykset korvata vanhat käytössä olevat skriptit.

7. YHTEENVETO JA JATKOKEHITYS

Tämän diplomityön tarkoituksena oli etsiä vastaus tutkimuskysymyksiin: ”Miten Tuni+:ssa olevien ohjelmointikurssien automaattisia arvioijia voidaan yhtenäistää?” Millaisilla työkaluilla ja käytännöillä tätä voidaan edesauttaa?”. Näihin kysymyksiin etsittiin vastauksia konstruktiivisen tapaustutkimuksen kautta, jonka seurauksena luotiin GraderCreator-niminen työkalu Tuni+:ssa opetettavien kurssien luontiin. Ohjelmointi 3 - sekä Basic Web Applications -kursseille luotiin uudet run.sh-skriptit, joiden kautta palautettujen harjoitustehtävien tarkistus tapahtuu. Lisäksi uusia arvointisäiliöitä varten luotiin uusi pohja, ja tätä pohjaa hyödynnettiin onnistuneesti Ohjelmonti 3 -kurssilla uuden arviointisäiliön muodossa.

Tutkimus alkoi luvussa 3 kuvattujen vaatimusten kartoittamisella, jota varten Tuni+:n arkkitehtuuri ja käyttötapaukset täytyi hahmotella ja mahdolliset rajoitteet luotavalle työkalulle saatiin näin selville. Tämän jälkeen sovellettiin luvuissa 2.4 ja 5.2 kuvattuja menetelmiä Ohjelmointi 3- sekä Basic Web Applications -kurssien arviointiskripteihin. tämän tuloksena molempien kurssien tehtäväkohtaiset skriptit yhdistettiin yhdeksi pääskriptiksi, jossa tehtävien tarvitsemat arvot kuten git-tagi, nimi sekä tiedostopolut parametrisoitiin. Nämä arvot annetaan uusille skripteille niiden ajokomennossa jokaisen tehtävän määrittelytiedoston kautta.

Kun työkalun vaatimukset ja rajoitteet oli selvitetty ja uudet arviointiskriptit luotu, uusia kursseja varten luotiin GraderCreator-työkalu, jolla voidaan alustaa nopeasti uusia Tuni+:aan luotavia kursseja, ja jolla voidaan kirjoittaa suoraan konfiguroituja tehtävien määrittelytiedostoja. Työkalu tukee tällä hetkellä neljää diplomityössä käsiteltyä ohjelmointikurssia siten, että tehtävien tarkistusta varten käytettävä sandbox-hakemisto voidaan kurssia luodessa liittää kurssin exercises-hakemistoon tehtävien arviointia varten. Uudet arviointiskriptit todettiin toimiviksi PoC-tasolla, mutta ne vaativat tarkempaa testausta ja mahdollisia korjauksia. Lisäksi uusi Docker-pohjakuva ohjelmointitehtävien tarkistussäiliöitä varten todettiin toimivaksi ratkaisuksi, ja sitä on käytetty onnistuneesti uuden arviointisäiliön luomisessa Ohjelmointi 3 -kurssille, jossa säiliö on tällä hetkellä käytössä.

Jotta työn koko olisi pysynyt tarpeeksi rajattuna, karsittiin aluksi määritellyistä vaatimuksista palautteen muotoilun muuttaminen, tehtäväpankki sekä valmiit testipohjat tehtäville. Palautteen muotoilu nykyisellään toimii riittävissä määrin, mutta sen ymmärtäminen vaa-

tii selkeämpää dokumentaatiota Tuni+:aan. Uusien testipohjien luonti olemassa oleville tehtäville ei auttaisi uusien kurssien kehitystä, eikä toisaalta parantaisi olemassa olevia toteutuksia merkittävästi. Tehtäväpankki jo olemassa oleville tehtäville ei myöskään auttaisi nykyisiä kursseja, ja se loisi tarpeettomia kopioita tehtävien koodista. Lisäksi tällaisen tehtäväpankin ylläpito loisi ylimääräistä työtä henkilökunnalle.

Suurin jatkokehityskohde työn pohjalta on koodikloonien ratkaisun ja yleistämisen hyödyntäminen laajemmin Tuni+:n ohjelmointikursseilla. Käsitellyistä Mooc-gradereista käytännössä vain Ohjelmointi 3 - sekä Basic Web Applications -kurssien toteutukset muuttuivat, kun niiden run.sh -skriptien rakenne yhtenäistettiin Ohjelmointi 1 - ja Ohjelmointi 2 -kurssien mukaiseksi. Tämä ei kuitenkaan tarkoita, ettei muiden kurssien toteutuksissa olisi mahdollisia koodiklooneja arviointiskripteissä tai tehtävien testitiedostoissa. Lisäksi uudessa toteutuksessa on itsessään vielä samanlaisia komentoja, jotka voitaisiin parametrisoida vähentäen kloonien määrää. Tätä varten jokaista Tuni+:ssa julkaistua kurssia tulisi tutkia luvussa 2.5 esitellyin menetelmin, ja kursseille tulisi suorittaa enemmän laadunvalvontaa staattisen analyysin keinoin. Lisäksi arvioinnissa käytettyjen työkalujen dokumentaatiota tulisi parantaa, jotta Tuni+:aa voitaisiin hyödyntää laajemmin Tampereen yliopiston kurssien alustana.

LÄHTEET

[1] Ala-Mutka, K. Automatic assessment tools in learning and teaching programming. Publication / Tampere University of Technology, 559. Tohtorinväitöskirja. Tampere: Tampere University of Technology, 2005.

[2] Pappano, L. *The Year of the MOOC*. The New York Times. 2. marraskuuta 2012. URL: https://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html (viitattu 22. 11. 2020).

[3] *Automatic assessment framework compatible with A-plus LMS*. URL: https://github.com/Aalto-LeTech/mooc-grader (viitattu 10. 12. 2019).

[4] Karavirta, Ville, Ihantola, Petri ja Koskinen, Teemu. Service-Oriented Approach to Improve Interoperability of E-Learning Systems. eng. IEEE, 2013, s. 341–345. ISBN: 9780769550091.

[5] Honkanen, T. Suntioinen Ari 2019. [Haastattelu] Haastattelijana Tommi Honkanen. 26. lokakuuta 2019.

[6] Honkanen, T. Harsu Maarit, Färm Minna 2019. [Haastattelu] Haastattelijana Tommi Honkanen. 22. lokakuuta 2019.

[7] Honkanen, T. Nurminen Mikko 2019. [Haastattelu] Haastattelijana Tommi Honkanen. 22. lokakuuta 2019.

[8] Honkanen, T. Niskanen Aku 2019. [Haastattelu] Haastattelijana Tommi Honkanen. 29. lokakuuta 2019.

[9] Lukka, K. The Constructive Research Approach. *Case Study Research in Logistics, Publications of the Turku School of Economics and Business Administration, Series B* (tammikuu 2003), s. 83–101.

[10] Cheang, B., Kurnia, A., Lim, A. ja Oon, W.-C. On automated grading of programming assignments in an academic institution. *Computers & Education* 41.2 (2003), s. 121–131. ISSN: 0360-1315. DOI: https://doi.org/10.1016/S0360-1315(03)00030-7. URL: http://www.sciencedirect.com/science/article/pii/S0360131503000307.

[11] Virtanen, T. Literature review of test automation models in Agile testing. eng. Tuotantotalous ja tietojohtaminen – Industrial and Information Management. diplomityö. Tampere University of Technology, maaliskuu 2018.

[12] Sacolick, I. What is CI/CD? Continuous integration and continuous delivery explained. eng. *InfoWorld.com* (17. tammikuuta 2020).

[13] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Petersen, K. ja Mäntylä, M. V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. *2012 7th International Workshop on Automation of Software Test (AST)*. 2012, s. 36–42. DOI: 10.1109/IWAST.2012.6228988.

[14] *TUNI Course-gitlab*. URL: https://course-gitlab.tuni.fi (viitattu 14. 10. 2020).

[15] Glass, R., Ramesh, V. ja Vessey, I. An analysis of research in computing disciplines. eng. *Communications of the ACM* 47.6 (2004), s. 89–94. ISSN: 0001-0782.

[16] Fowler, Martin ja Beck, Kent. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999. URL: books.google.com.

[17] Fowler, M. *Refactoring : improving the design of existing code*. eng. The Addison-Wesley object technology series. Reading, MA: Addison-Wesley, 1999. ISBN: 9780134757681.

[18] Fowler, M. *Parameterize Function*. URL: https://www.refactoring.com/catalog/parameterizeFunction.html (viitattu 16. 11. 2020).

[19] Neha, S., Sukhdip, S. ja Suman. Code Clones: Detection and Management. *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, s. 718–727. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.080. URL: http://www.sciencedirect.com/science/article/pii/S1877050918308123.

[20] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. ja Merlo, E. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering* 33.9 (2007), s. 577–591. DOI: 10.1109/TSE.2007.70725.

[21] Ducasse, S., Rieger, M. ja Demeyer, S. A language independent approach for detecting duplicated code. eng. *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*. IEEE, 1999, s. 109–118. ISBN: 0769500161.

[22] Kamiya, T., Kusumoto, S. ja Inoue, K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. English. *IEEE Transactions on Software Engineering* 28.7 (heinäkuu 2002), s. 654–670. URL: https://libproxy.tuni.fi/login?url=https://www.proquest.com/docview/195579324?accountid=14242.

[23] Mayrand, Leblanc ja Merlo. Experiment on the automatic detection of function clones in a software system using metrics (1996), s. 244–253. DOI: 10.1109/ICSM.1996.565012.

[24] Patil, R. V., Joshi, S. D., Shinde, S. V., Ajagekar, D. A. ja Bankar, S. D. Code clone detection using decentralized architecture and code reduction (2015), s. 1–6. DOI: 10.1109/PERVASIVE.2015.7087126.

[25] Ferrante, J., Ottenstein, K. J. ja Warren, J. D. The Program Dependence Graph and Its Use in Optimization. *ACM Trans. Program. Lang. Syst.* 9.3 (heinäkuu 1987), s. 319–349. ISSN: 0164-0925. DOI: 10.1145/24039.24041. URL: https://doi.org/10.1145/24039.24041.

[26] *A+ LMS, the extendable learning management system*. URL: https://apluslms.github.io/architecture/ (viitattu 20. 11. 2020).

[27] *Tuni+ kurssipohja*. URL: https://course-gitlab.tuni.fi/ITC/CS/tie-99002/plussa-syksy2019/-/tree/1432c1568b4925f41e90000cbd41d3cc17abb4e0 (viitattu 16. 11. 2020).

[28] *A+ rst-tools*. URL: https://github.com/Aalto-LeTech/a-plus-rst-tools (viitattu 17. 04. 2020).

[29] *What is a Container? App Containerization*. URL: https://www.docker.com/resources/what-container (viitattu 18. 11. 2020).

[30] *Docker frequently asked questions*. URL: https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc (viitattu 18. 11. 2020).

[31] Simic, S. *Docker Image VS Container: What is the difference?* URL: https://phoenixnap.com/kb/docker-image-vs-container (viitattu 16. 11. 2020).

[32] *Dockerfile documentation*. URL: https://docs.docker.com/engine/reference/builder/ (viitattu 19. 03. 2020).

[33] *Docker: About storage drivers*. URL: https://docs.docker.com/storage/storagedriver/ (viitattu 18. 11. 2020).

[34] *reStructuredText markup syntax*. URL: https://docutils.sourceforge.io/rst.html (viitattu 09. 10. 2020).

[35] *Sphinx Documentation generator*. URL: https://www.sphinx-doc.org/en/master/ (viitattu 09. 10. 2020).

[36] *Toctree and the Hierarchical Structure of a Manual*. URL: https://docs.typo3.org/m/typo3/docs-how-to-document/master/en-us/WritingReST/MenuHierarchy.html (viitattu 14. 11. 2020).

[37] *Toctree Sphinx directive*. URL: https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree (viitattu 14. 11. 2020).

[38] *Python tkinter documentation*. URL: https://docs.python.org/3/library/tkinter.html (viitattu 11. 12. 2019).

[39] *Tcl/Tk graphical toolkit*. URL: http://www.tcl.tk/ (viitattu 11. 12. 2019).

[40] Martelli, A. *Python in a nutshell*. eng. Third edition. In a nutshell (O'Reilly & Associates). Sebastopol, CA: O'Reilly Media, Inc., 2017. ISBN: 1-4919-1383-5.

[41] *Grading-base container for A+ LMS*. URL: https://github.com/apluslms/grading-base (viitattu 14. 04. 2020).

[42] *A+ grading-base grader utilities*. URL: https://github.com/apluslms/grading-base/tree/master/bin (viitattu 17. 04. 2020).

[43] *Qt base everywhere*. URL: http://master.qt.io/archive/qt/5.14/5.14.2/submodules/qtbase-everywhere-src-5.14.2.tar.xz (viitattu 21. 10. 2020).

[44] *Ohjelmointi 3 -kurssin Qt5.9.6 Dockerfile*. URL: https://course-gitlab.tut.fi/pervasive_computing/plussa/dockerfiles/grader-qt596-ohj3 (viitattu 21. 10. 2020).

[45] *Dockerfiles for Qt with ubuntu as base image*. URL: https://course-gitlab.tut.fi/pervasive_computing/plussa/dockerfiles/ubuntu_qt (viitattu 21. 10. 2020).

[46] *GNU Wget*. URL: https://www.gnu.org/software/wget/manual/wget.html (viitattu 13. 10. 2020).

[47] *Software engineering — Product quality — Part 1: Quality model*. Standard. International Organization for Standardization, kesäkuu 2001. URL: https://www.iso.org/standard/22749.html.

[48] Jackson, Mike, Crouch, Steve ja Baxter, Rob. Software Evaluation: Criteria-based Assessment. eng (marraskuu 2011). URL: https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf.

[49] *CourseCreator repository*. URL: https://gitlab.tut.fi/honkan23/diplomityo (viitattu 22. 11. 2020).