Jussi Rajaniemi

Staattinen analyysi ohjelmakoodin automaattisessa arvioinnissa

Tietotekniikan
pro gradu -tutkielma
19. huhtikuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Kokkolan yliopistokeskus Chydenius

Tekijä: Jussi Rajaniemi
Yhteystiedot: jussi.rajaniemi@gmail.com
Puhelinnumero: 046 811 8508
Ohjaaja: Mikko Myllymäki
Työn nimi: Staattinen analyysi ohjelmakoodin automaattisessa arvioinnissa
Title in English: Static analysis in automated assessment of programming code
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 47 + 9
Tiivistelmä: Tämän pro gradu -tutkielman tavoitteena oli kehittää toimiva konsepti MOOC-ohjelmointikurssien automaattiselle arvioinnille. Tutkielman ajurina toimi tutkimuskysymys: ”Millaisella konseptilla voidaan toteuttaa C++-lähdekoodin automaattinen arviointi?”. Tutkimusmenetelmänä toimi kehittämistutkimuksen ja ketterän Scrum-mallin yhdistelmä, jolla pyrittiin varmistamaan tutkimuksellinen ote ja ketterä sovelluskehitysmalli. Tutkimuksen ensimmäisessä vaiheessa kartoitettiin ongelma-analyysin avulla MOOC-kurssien ja automaattisen arvioinnin vahvuuksia sekä haasteita. Toisessa vaiheessa aloitettiin kehitystyö, joka päättyi käyttäjätestaukseen. Tulosten perusteella konsepti täytti sille asetetut vaatimukset, ja onnistuttiin ratkaisemaan testitapausten laadintaan ja on/off-tyyppisen arvioinnin haasteet menettämättä automaattisen arvioinnin vahvuuksia.
Avainsanat: MOOC, C++, automaattinen arviointi, staattinen analyysi, ohjelmointi
Abstract: The purpose of this master thesis was develop a workable concept for automated assessment for MOOC programming courses. Thesis research work is driven by the question: ”What kind of concept can be used to implement automated assessment in C++ source code?”. The research method was a combination of design research and Scrum development model, with the aim of ensuring a research approach and agile application development model. In the first phase of study, the strengths and challenges of MOOC courses and automatic assessment were survey using problem analysis. In the second phase, development started and it ended with user testing. The results showed that the concept met requirement specifications and we were able to solve challenges of making test cases and on/off-type assessment without losing the strength of automated assessment.
Keywords: MOOC, C++, automated assessment, static analysis, programming

Versio 0.9.72

Sanasto

API	Application Programming Interface
AST	Abstract Syntax Tree
CICD	Continuous Integration and Continuous Delivery
CLI	Command Line Interface
E2E	End To End
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MOOC	Massive Open Online Course
cMOOC	The Connectivist MOOC
xMOOC	Extended MOOC
qMOOC	quasi-MOOC
REST	Representational State Transfer
SCSS	Sassy CSS
SPA	Single Page Application
TDD	Test-Driven Development
UI	User Interface
VSS	Valuated State Space
WCAG	Web Content Accessibility Guidelines
XML	Extensible Markup Language

Sisältö

1 Johdanto

MOOC (Massive Open Online Courses) -kursseista on kehittynyt viime vuosina uusi ilmiö korkeakouluille. Termejä, joita MOOCiin liitetään, ovat avoin, globaali, ilmainen, videopohjainen sisältö, videot, verkko-opetus, vertaisoppiminen ja sähköinen oppimisalusta [4].

Laadukas ja suosittu MOOC-kurssi tuo kouluille näkyvyyttä ja mainetta, mutta vastapainona MOOC-kurssin suunnittelu ja toteutus vaatii huomattavasti enemmän kuin perinteinen suljettu verkkokurssi. MOOCin suuri opiskelijamäärä tuottaa ongelmia kurssin hallinnalle - se synnyttää paljon kysymyksiä? Miten oppimista ohjataan? Kuinka harjoitustöiden edistymistä seurataan ja arvioidaan, millaista palautetta annetaan ja miten harjoituksia palautetaan? Miten annetaan palautetta suorituksista? Kustakin kysymyksestä voisi tehdä oman pro gradu-tutkielman, mutta tässä tutkimuksessa pyritään löytämään konsepti ja tekninen toteutus C++-koodin automaattiseen arviointiin. Tutkimuskysymykseni on:

Millaisella konseptilla voidaan toteuttaa C++-lähdekoodin automaattinen arviointi?

Automaattista ohjelmakoodin arviointia on tutkittu yllättävän vähän suhteessa ohjelmistoalan opetuksen määrään. Alan tutkimus keskittyy tyypillisesti yhteen menetelmään tai teknologiaan, mutta koontitutkimuksia on hyvin vähän tarjolla. Yksi parhaimpia koontitutkimuksia oli Ala-Mutkan *A survey of automated assessment approaches for programming assignments* [2], jossa hän on tutkinut ohjelmointikurssien harjoitusten automaattista arviointia hyvin kattavasti. Artikkeliin on viitattu yli 520 kertaa, joten useimmat tämän kirjoitelman lähteet on poimittu näiden viittausten tai Ala-Mutkan lähdekirjallisuuden joukosta.

Luvussa 2 esittelen gradun tutkimusmenetelmän, joka on kehittämistutkimuksen ja ketterän Scrum-mallin yhdistelmä sekä siihen liittyviä oleellisimpia tuotekehityksen toimintamalleja. Luvussa 3 on katsaus aikaisempiin aiheesta tehtyihin tutkimuksiin ja se toimii ongelma-analyysina kehittämistutkimukselle. Luvussa 4 käsittelen sovelluksen toimintaa ja arkkitehtuuria. Luvussa 5 esittelen tutkimuksen tuloksia ja jatkokehitysideoita. Viimeisessä luvussa on lyhyt yhteenveto tutkimuksesta.

2 Tutkimusmenetelmä

Tämän gradun tavoitteena oli kehittää toimiva konsepti MOOC-ohjelmointikurssien arvioinnille. MOOC-ohjelmointikurssien suurimpia haasteita on ohjelmakoodien arviointi, joka johtuu suuresta opiskelijamäärästä. Perinteisessä luokka- tai etäopetuksessa osallistuu ohjelmoinnin kurssille tyypillisesti 10–30 opiskelijaa, jolloin koodien arviointi on vielä tehtävissä manuaalisesti pienellä vaivalla. Toisaalta MOOC-kursseille osallistuu helposti satoja tai tuhansia opiskelijoita, jolloin koodien manuaalinen arviointi on erittäin työlästä ja käytännössä mahdotonta. Tästä johtuen MOOC-kurssien osallistujamääriä joudutaan hyvin usein rajoittamaan.

Konseptin kehittämiseksi tarvitaan joustava menetelmä, jossa on tutkimuksellinen ote ja ketterä sovelluskehitysmalli. Tässä kehitystyössä on sovellettu kehittämistutkimusmallia ja ketterää Scrum-mallia toimivan ohjelmistokonseptin toteuttamiseen sekä käytetty jatkuvan integraation ja julkaisun menetelmää sovelluksen testauksessa ja julkaisussa. Seuraavissa luvuissa esitellään näiden menetelmien perusperiaatteet.

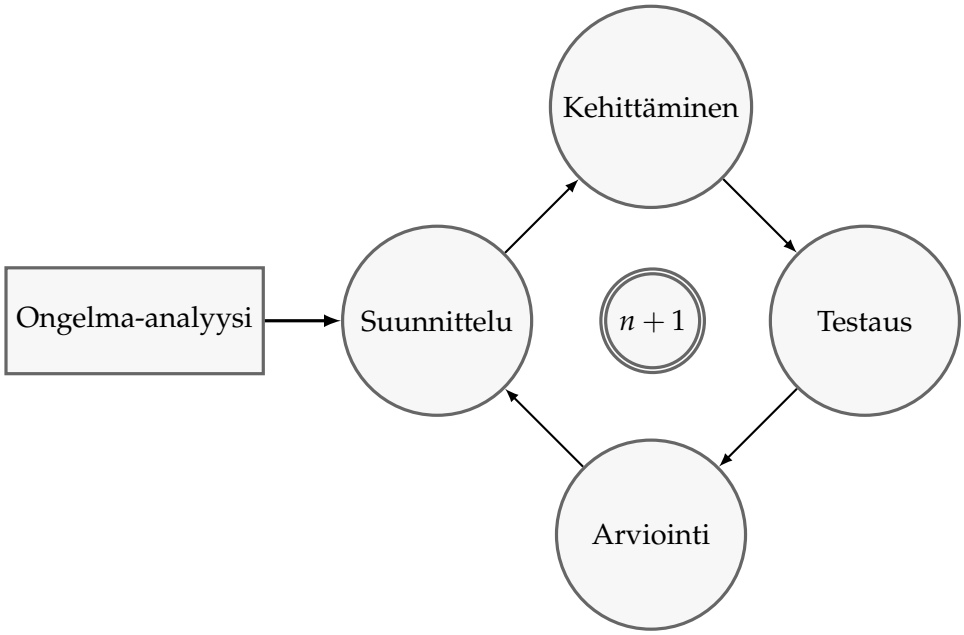
2.1 Kehittämistutkimus

Työssä on käytetty tutkimusmenetelmänä kehittämistutkimusta. Termin englanninkielinen vastine on *design research* tai *design-based research* [23, s. 1]. Kehittämistutkimuksen historia alkaa vuodesta 1992, mutta laajemmin sitä on alettu käyttämään opetusalalla 2000-luvun puolella [23, s. 2].

Kehittämistutkimuksen ideana on tarjota menetelmä, jossa opetusta kehitetään tutkimuksen menetelmin opetuksen tarpeiden pohjalta. Kehittämistutkimus vastaa kritiikkiin, jonka mukaan teoreettinen tutkimus ei ole tuottanut opettajan työtä tukevaa tietoa [23].

Kehittämistutkimuksen tunnusmerkkejä ovat: a) tutkimusvetoinen iteratiivinen kehittämisprosessi, b) tutkimus vastaa todelliseen tarpeeseen c) tutkimus tuottaa artefaktin, joka auttaa opettajia ja opiskelijoita toimimaan paremmin ja d) tuottaa uutta tietoa opettamisesta ja oppimisesta [9, s. 108–109, 116].

Kehittämistutkimus koostuu kehityssykleistä, joiden sisällä on useita kehitysi-



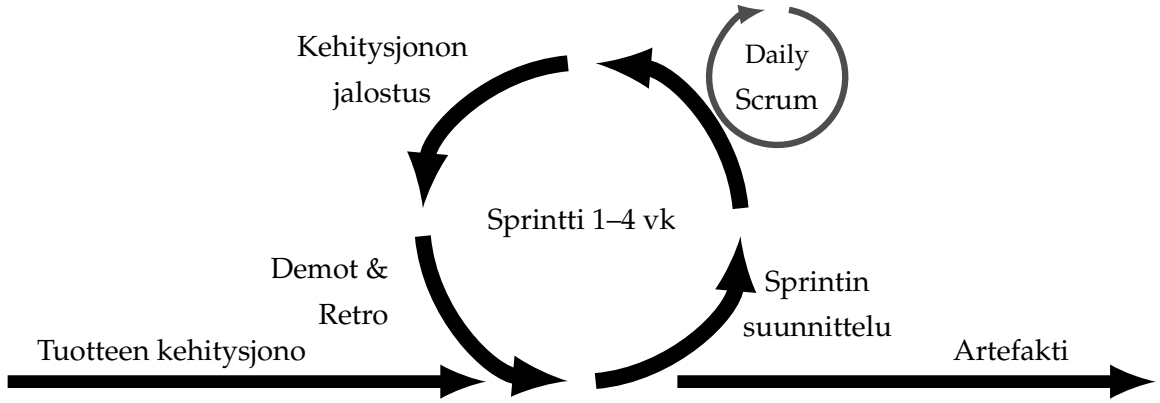
Kuva 2.1: Kehityssyklin kehitysiteraatio

teraatioita. Sykli alkaa aina ongelma-analyysista, jonka perusteella määritetään kehittämissuunnitelma. Kun suunnitelma on valmis, aloitetaan kehitystyö tekemällä ensimmäinen versio tuotteesta eli artefaktista. Kehitystyön jälkeen artefaktia testataan ja arvioidaan vastaako tuotos tarpeita ja tavoitteita. Arvioinnin pohjalta laaditaan uusi kehittämissuunnitelma, jonka jälkeen voidaan aloittaa uusi kehitysiteraatio (kuva 2.1). Iterointia jatketaan niin kauan, kunnes ongelma-analyysin haasteet, tarpeet ja tavoitteet on täytetty [23, s. 6–7].

2.2 Scrum

Ohjelmistokehityksessä on laajasti käytetty Scrum-mallia, jonka ensimmäiset toimintamallit esitteli Sutherland [33, s. 174–175] ja Schwaber [29, s. 117–134]. Scrum-ohjelmistokehitysmalli muistuttaa hyvin paljon kehittämistutkimuksen toimintamallia, mutta se on ketterämpi kehitysmalli kuin kehittämistutkimus. Kirjassa Agile software development with Scrum Schwaber et al. [30] esittelivät Scrum-mallia nykymuodossaan.

Scrum-malli alkaa aina (kuva 2.2) tuotteen kehitysjonosta (product backlog), jossa on kehitettävän artefaktin kehityssuunnitelmat osioitu pienempiin osakokonaisuuksiin prioriteettijärjestyksessä [30, s.32]. Sprintti (sprint) on vakiomittainen ke-



Kuva 2.2: Scrum-kehitysmalli

hityssykli, jonka lopputuloksena on aina toimiva artefakti. Sprintti suunnitellaan aina ennen sen käynnistämistä. Sprintin suunnittelupalaverissa valitaan seuraavat kehityskohteet kehitysjonosta, arvioidaan työmäärät ja asetetaan niille hyväksyntäkriteerit. Suunnittelun tavoitteena on saada suunniteltua sprintin kehitysjono, joka voidaan toteuttaa sen aikana [30, s. 49].

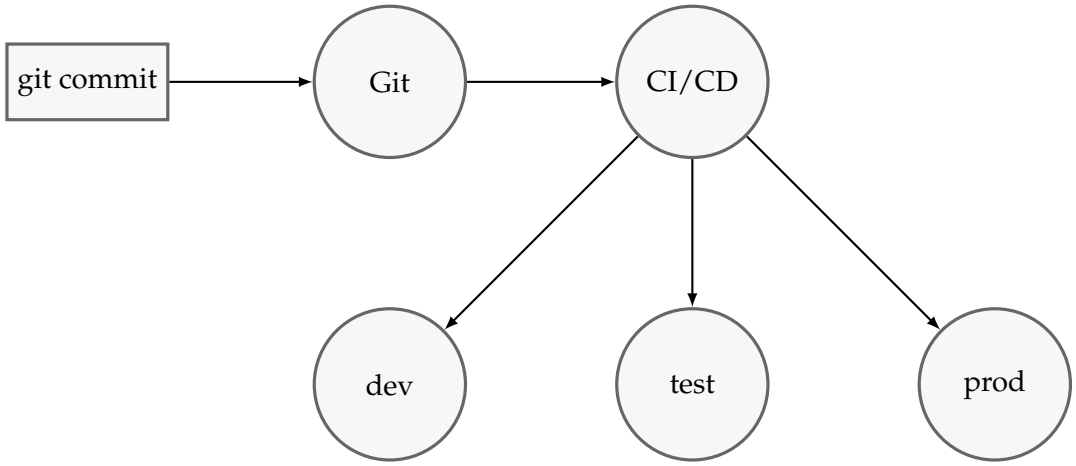
Kun sprintti on käynnistetty, aloitetaan kehitystyö ottamalla kehitysjonosta kehitystehtäviä työnalle. Tämän jälkeen tiimi pitää tyypillisesti 15 minuutin mittaisen päiväpalaverin (daily scrum), jossa tiimi voi synkronoida työtehtävät ja pohdiskella esiintyneitä haasteita. Päiväpalaverissa käydään läpi jokaisen jäsenen edelliset tehtävät ja tulevat tehtävät seuraavalle päivälle. Jos jollain tiimin jäsenellä on ongelmia, niin Scrum Master pyrkii ratkaisemaan niitä [30, s. 44–45]. Sprintin aikana siivotaan, järjestellään ja lisätään uusia tehtäviä tuotteen kehitysjonolle (backlog refining) [30, s. 53]. Näin myös kehitysjono uudistuu projektin aikana kehitystiimin todellisen tarpeen johdosta.

Sprintin päätteeksi pidetään katselmointipalaveri (review) ja retro (retrospective). Katselmoinnissa käydään sprintin tuotokset läpi esittelemällä tuotteen toimintaa ja katsotaan täyttyvätkö kehityskohteiden hyväksyntävaatimukset [30, s. 54]. Usein kehitetty tuote julkaistaan testiympäristöön testattavaksi. Viimeisenä pidetään retro, jossa katselmoidaan kehitysprosessia: missä onnistuttiin, missä epäonnistuttiin ja miten voitaisiin kehitysprosessia parantaa.

Kun sprintti on saatu valmiiksi, alkaa seuraava sprintti ja sen suunnittelu. Uusia sprinttejä käynnistetään niin kauan, kunnes tuotteen kehitysjono on tyhjä tai kehitettävä artefakti todetaan olevan riittävän hyvä tuotantoon vietäväksi. Tällöin kehitysjonossa olevat tehtävät muuttuvat jatkokehitystehtäviksi.

Scrum on kehitetty tiimin projektinhallintamalliksi, mutta se soveltuu myös yhden hengen tai parin kehitysmalliksi. Tällöin luonnollisesti osa Scrumin toimintamalleista jää merkityksettömiksi.

2.3 Sovelluksen jatkuva julkaisu



Kuva 2.3: Jatkuvan julkaisun toiminta

Kiinteä osa ketterää Scrum-mallin mukaista ohjelmistokehitystä on kehitettävän sovelluksen jatkuva integraatio ja jakelu (Continuous Integration and Continuous Delivery, CI/CD) eli jatkuva julkaisu [30, s. 29]. Arachchi et al. [3, s. 156–161] esittelevät julkaisussaan jatkuvan julkaisun menetelmän periaatteita kattavasti.

Jatkuvan julkaisun ideana on automatisoida koko ohjelmiston julkaisuprosessi: a) sovelluksen kääntö, b) koodin arviointi, c) sovelluksen yksikkötestaus, d) asennus joko kehitys-, testi- tai tuotantoympäristöön sekä e) sovelluksen suorituskyky- ja regressiotestaus [3, s. 159].

Jatkuva julkaisu käynnistyy aina kun versionhallinnan haaraan tulee koodarilta muutos (kuva 2.3). Esimerkiksi jos versiohallinnan development-haaraan tulee muutos, käynnistyy CI/CD-prosessi. Se kääntää sovelluksen, arvioi koodin laatua, tekee yksikkötestit, asentaa sovelluksen kehitysympäristöön ja suorittaa regressiotestauksen.

Jatkuvan julkaisun hyödyt ovat ilmeisiä. Jatkuva integraatio vähentää riskejä ja tekee sovelluksesta luotettavamman ja se sisältää vähemmän ohjelmointivirheitä. Jatkuva jakelu nopeuttaa sovelluksen vientiä eri ympäristöihin, lisää tuottavuutta,

parantaa julkaisun luotettavuutta sekä asiakkaan tyytyväisyyttä [3, s. 156].

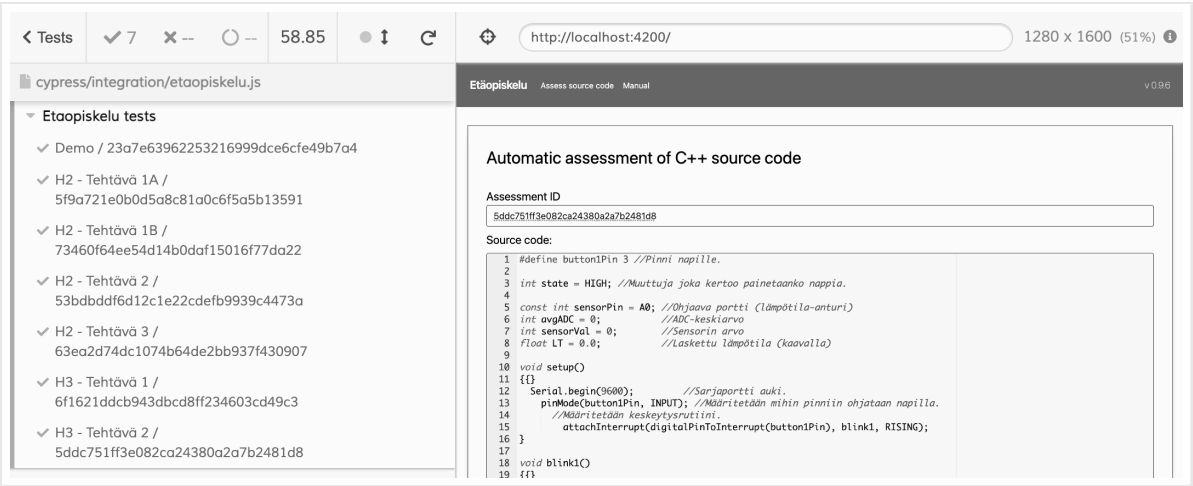
2.4 Sovelluksen testaus

Sovelluksen käyttöönoton yhteydessä törmättiin jatkuvasti samaan ongelmaan, jossa havaittiin semanttisessa analyysissä puutteita tai virheellistä toimintaan. Koska analyysin toteutus on varsin monimutkainen, niin sen muuttaminen aiheutti helposti uusia piileviä virheitä. Esimerkkinä muuttujan nimi: button1Pin. Ensimmäisessä versiossa semanttinen analyysi pilkkoi muuttujan kolmeen osaan:

```
["WORD", "NUMBER", "WORD"]
```

Kun siitä olisi pitänyt tulla pelkkä ”WORD”. Analyysiä muutettiin ja kaikki näytti toimivan oikein. Viikkoa myöhemmin havaittiin, että kaksi harjoitusta oli lakannut toimimasta muutoksesta johtuen. Vika korjattiin ja taas muutaman päivän päästä huomattiin, että yksi harjoitustehtävä lakkasi toimimasta.

Ongelmaa lähdettiin ratkaisemaan automaattisella regressiotestauksella, joka ajetaan aina paikallisesti kun lähdekoodiin tulee muutos, tai kun jatkuva julkaisu vie uutta versiota tuotantoon palvelimille. Testauksessa käytettiin Cypress-testaustyökalua, joka käynnistää testiaineiston ajon aina kun koodieditorissa tallennetaan muutos lähdekoodiin.

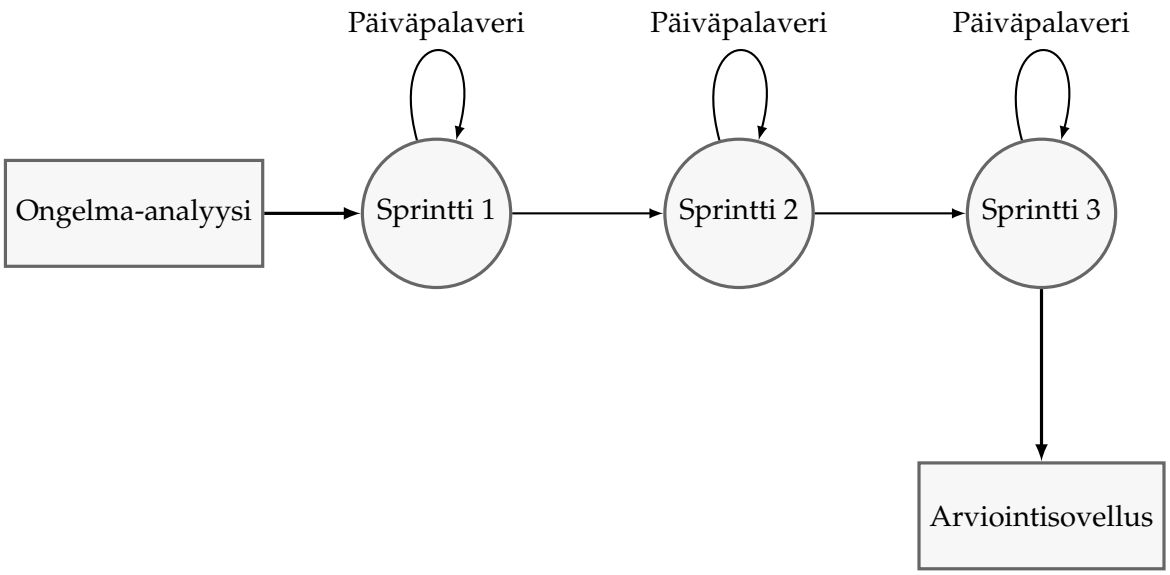


Kuva 2.4: Cypress-testityökalun käyttöliittymä

Cypress osoittautui äärimmäisen tehokkaaksi testaustyökaluksi paljastaen yllättäen neljä aikaisemmin havaitsematonta virhettä. Kuvassa 2.4 on seitsemän lähde-

koodin onnistunut testitapaus suoritettuna. Testattavat lähdekoodit vietiin Cypressille lähdekooditiedostoina ja testitapaukset, jossa vertailtiin arviointisovelluksen tuloksia oikeaan tulokseen, kirjoitettiin JavaScript-kielellä.

2.5 Kehittämistutkimus ja Scrum-malli tässä tutkimuksessa



Kuva 2.5: Kehittämistutkimus Scrum-mallilla

Tässä tutkimuksessa on käytetty kehittämistutkimuksen ja Scrum-ohjelmistokehitysmallin yhdistelmää, jossa ongelma-analyysissä on määritetty tutkimuksen haasteita ja tarpeita sekä asetettu tavoitteet. Ongelma-analyysin jälkeen siirryttiin Scrum-malliin, jossa ongelma-analyysin perusteella luotiin artefaktin kehitysjono (product backlog). Sprintin (sprint) pituudeksi asetettiin neljä viikkoa, jonka aikana tulisi saada aina toimiva artefakti. Käytännön syistä johtuen työn ohjaajan kanssa pidettiin 30 minuutin mittainen päiväpalaveri (daily scrum) vain perjantaisin. Päiväpalaverissa katsottiin mitä on saatu aikaan ja mitä tehdään seuraavaksi sekä lisättiin uusia ideoita, parannuksia ja korjaustarpeita kehitysjonoon. Tässä yhteydessä myös ratkaistiin esiin tulleita ongelmia.

Kehittämistutkimus alkoi toimeksiantajan tarpeesta ratkaista Arduino-kurssin arviointihaasteita. Lähtökohtana oli tietokoneavusteisen arviointimenetelmän kehittäminen. Kevään aikana ideoimme automaattiselle arvioinnille useita ratkaisumalleja ja samaan aikana aloin tutustumaan MOOC-kursseista ja ohjelmakoodin au-

tomaattisesta arvioinnista tehtyihin tutkimuksiin ja menetelmiin sekä havaittuihin haasteisiin. Tavoitteena oli löytää niissä ilmenneitä haasteita ja vahvuuksia, joiden perusteella tehtiin ongelma-analyysi ja asetettiin tavoitteet kehittämistyölle. Ongelma-analyysin jälkeen siirryttiin kehitysvaiheeseen ja kehitystyö saavutti asetetut tavoitteet kolmannen sprintin aikana.

3 Ongelma-analyysi

Ongelma-analyysin tavoitteena on tutkia mitä vahvuuksia ja haasteita MOOC-kurssien arvioinnissa on havaittu ja millaisilla automaattisilla arviointi menetelmillä haasteita on lähdetty opetusalalla ratkaisemaan. Ongelma-analyysissä on käyty ensin läpi yleisellä tasolla mitä vahvuuksia ja haasteita on ohjelmistoalan MOOC-kursseilla havaittu. Toisessa vaiheessa on luotu katsaus automaattisesta arvioinnista tehtyihin tutkimuksiin ja pyritty löytämään niiden vahvuuksia ja haasteita.

3.1 Katsaus MOOC kursseista tehtyihin tutkimuksiin

Baturay [4] antoi hyvän katsauksen korkeakoulujen MOOC-kurssien tilanteesta. Viime vuosina ilmiöksi on noussut Courseran, Udacity ja Udemyn kurssit, joiden perässä yliopistot, kuten MIT ja Harvard, julkaisivat omat kurssinsa EdX-alustalla. Euroopassa Futurelearn ja Iversity tarjoavat vastaavia MOOC-kursseja omalla alustoillaan.

Kaikille näille alustoille yhteistä on niiden avoimuus. Kuka tahansa voi osallistua MOOC-kurssille, kunhan on pääsy Internetiin. MOOC-kurssi osallistaa käyttäjää enemmän, koska hänen tulee luoda ja jakaa tuloksia. Baturay mukaan iso osa kursseista perustuu yhteistyöhön, jossa opiskelijat jakavat osaamista verkossa. Isona ongelmana Baturay [4, s. 431] pitää arviointia, jota tuottaa suuret opiskelijamäärät. Yleisesti on käytetty automaattista arviointia, kuten monivalintakysymyksiä sekä vertaisarviointia.

Pieterse [24] mukaan ohjelmoinnin opetuksessa MOOC-kontekstissa suurin haaste on arviointi. Suurinta lisäarvoa tuottaisi nopea ja tarkka automaattinen arviointi. Pieterse mukaan automaattista arviointia ehdotti jo Hollingsworth [13] paljon ennen henkilökohtaisten tietokoneiden tuloa. Pieterse mukaan hyvän MOOC-kurssin menestystekijöitä ovat: a) laadukas arviointi, b) selkeät ja täsmälliset tehtävän annot, c) hyvin valittu testiaineisto, d) hyvä formatiivinen palaute, e) rajoittamaton harjoitusten lähetys mahdollisuus, f) opiskelijan kokemus testivetoisesta sovelluskehityksestä ja g) henkilökohtaisen ohjauksen tarjoaminen.

Pieterse [24] korosti myös harjoitustehtävien laatimista, sillä pienikin virhe teh-

tävän annossa voi johtaa ongelmiin automaattisessa arvioinnissa. Automaattisen arvioinnin ei todettu vähentävän opettajan työmäärää, vaan työn luonne on muuttunut manuaalisesta arvioinnista arvioinnin suunnitteluun ja toteuttamiseen. Yhtenä haasteena Pieterse [24] piti luovuuden tukahtumista, johon automaattinen arviointi johtaa. Automaattisessa arvioinnissa joudutaan tehtävän antoa rajaamaan tiukasti testiaineistoa vasten, jolloin opiskelijan luovuus voi tukahtua.

Plagiointia Pieterse [24] piti ongelmana, jota hän perusteli Suleman [32] ja Chen [8] tutkimuksilla, joiden mukaan automaattinen arviointi näyttäisi lisäävän opiskelijoiden plagiointia. Toisaalta MOOC-kurssien konsepti taas voi vähentää plagiointia, kun kursseja ei käydä opintopisteiden toivossa, vaan halusta oppia uutta. Myös arviointisovelluksen käyttö voi olla haaste itsessään, joten myös käytettävyyteen tulee kiinnittää huomiota.

Pieterse [24] kiinnitti huomiota myös tietoturvaan, sillä Hollingsworth [13], Chen [8] Ala-Mutka [2] ja Cerioli et al. [6] olivat huomanneet tarpeen ajaa arviointisovellusta omassa rajatussa hiekkalaatikossa (sandbox). Viimeisenä hän kiinnitti huomiota myös testimenetelmien rajalliseen kyvykkyyteen. Yksikkötestauksella voidaan testata sovelluksen toiminnallisuutta, mutta ei sitä, miten se on toteutettu. Metriikoilla voidaan mitata sovelluksen kompleksisuutta, mutta ei sen toiminnallisuutta ja hyvyyttä. Ihannetapauksessa tulisi arviointisovellukseen yhdistää eri testausmenetelmiä, jolloin arvioinnin kattavuus paranisi.

Bey et al. [5] tutkivat kahta automaattisen arvioinnin menetelmää MOOC-kurssien yhteydessä. Ensimmäinen menetelmistä perustui staattiseen analyysiin ja toinen dynaamiseen analyysiin, joka oli toteutettu yksikkötestauksena. Bey et al. [5, s. 259] totesivat että suuri haaste on arviointi, joka ruuhkautuu suuresta harjoitustehtävien palautusmäärästä, sillä jokainen palautus tarvitsee palautteen ja arvion. Toinen haaste on antaa riittävän laadukasta palautetta [5, s. 268], jotta opiskelija pääsee etenemään harjoitustyössään. Tutkimuksen perusteella dynaamisen ja staattisen analyysin välillä ei ole lopputuloksen kannalta suurta eroa, vaikka molemmat mittaavat eri asioita.

Xiong et al. [39] tutkivat MOOCin mahdollisuuksia, haasteita ja tulevaisuuden suuntia yleisellä tasossa. He jakoivat MOOC ilmiön kolmeen erityyppiseen luokkaan xMOOC (extended MOOC), cMOOC (connectivist MOOC) ja qMOOC (quasi-MOOC), joissa kahdessa ensimmäisessä oli eri pedagoginen lähtökohta. cMOOC-kurssien lähtökohtana on yhteistoiminnallinen oppiminen, kun taas xMOOC-kurssien pedagoginen malli on toimijalähtöinen. xMOOC-kurssilla opiskelijat katsovat itse-

näisesti video-opetusta, tekevät harjoituksia ja palauttavat ne opettajalle, eli se on opettajavetoista oppimista. qMOOC-kurssilla tarkoitetaan ”kurssin” kaltaista luentoa, kuten Youtube-videoita tai Khan Academyn koulutuksia.

Erilaisilla pedagogisilla malleilla oli vaikutusta arviointimenetelmiin. cMOOC-kurssien haasteen on mitata yksilön oppimista, kun oppiminen tapahtuu yhdessä tietoa jakaen. xMOOC-kurssien osalta arviointi on helpompaan, kun oppiminen on toimijalähtöistä. qMOOC-kursseja ei tyypillisesti arvioida lainkaan, koska niissä ei palauteta tehtäviä, eikä olla kontaktissa opettajaan [39, s.244–245].

Xiong et al. [39, s.245] mukaan vahvuuksia oli MOOC-kurssien tuki elinikäiselle oppimiselle, mutta suurimpana haasteena pidettiin arviointia. Rajallinen määrä resursseja sekä suuri määrä eri lähtökohdista tulevia opiskelijoita tekivät palautteen annosta ja arvioinnista hyvin työvoimavaltaista. Xiong et al. [39, s.248] mukaan parhaiten MOOC-kurssien yksilöllisen ja formaalin arvioinnin tarpeet saataisiin ratkaistua joko koneellisella arvioinnilla tai vertaisarvioinnilla.

3.2 Katsaus automaattisesta arvioinnista tehtyihin tutkimuksiin

Tirronen et al. [35, s. 180–181] ovat tutkineet opiskelijan toimintaa ja käyttäytymistä automaattisen arvioinnin kanssa ja kehittäneet menetelmän luokitella eri käyttäytymistilanteita. Menetelmä perustuu tilastolliseen sisältöanalyysiin, jossa 9 423 tehtäväpalautusta jaettiin tehtäväkohtaisiin sessioihin. Yhteen sessioon sisältyy opiskelijan kaikki saman tehtävän palautukset. Sessioita kertyi kaikkiaan 2 097 kappaletta. Kun sessioista tehtiin sisältöanalyysi, niin tehtäväpalautuksista voitiin muodostaa neljä luokkaa: a) hylätyt (abandon), b) myllätä (churn), c) vaillinaiset (incomplete) ja d) kehittää (improve).

Opiskelijan hylkäämillä tehtävillä tarkoitettiin tehtäväpalautuksia, joissa opiskelija toi järjestelmään arvioitavaksi tehtävän, joka ei toiminut ja se on arvioitu hylätyksi. Tirrosen mukaan tämä osoittaa opiskelijan turhautumista ja tehtävän vaikeutta. Tyypillisesti käännösvirheet tuottivat paljon tehtävän hylkäyksiä ja huonoa palautetta. Samoin tehtävät, joissa opiskelijan tuli toteuttaa useita funktioita arvioinnin saamiseksi, saivat opiskelijan hylkäämän tehtävän. Hylkäämisiä aiheuttivat myös tehtävät, joissa täytyi tietää tietorakenteita, kuten muuttujien tyypit. Näissä tapauksissa opiskelijat todennäköisesti ymmärsivät, ettei heillä ole riittävää määrää tietoa tehtävän loppuun saattamiseksi. Ylipäätään opiskelijat hylkäsivät tehtävät, joissa vaadittiin paljon monimutkaista logiikkaa, ja niissä nähtiin paljon yrityksiä ja ereh-

dyksiä ennen kuin opiskelija luovutti tehtävän teon.

Toinen luokka on kokeilemalla suoritetut tehtävät (churn). Niissä opiskelijan sessio koostui yli kymmenestä yrityksestä ilman hyväksyttyä arviointia. Tämä luokituskriteeri suunniteltiin tunnistamaan ”yritä ja erehdy”-käyttäytymistä. Jos lyhyissä harjoituksissa ”yritä ja erehdy”-menetelmä ei johtanut oikeaan lopputulokseen, se kertoo ongelmista kurssimateriaalissa. Tämä kriteeri nousi esiin myös asteittain etenevässä työmallissa ja siten se ei ole hyvä indikaattori vaikeiden harjoitustöiden kohdalla.

Osaan harjoituksista oli sisällytetty optioina lisätehtäviä. Kolmas luokka oli vaillinaiset tehtävät. Tässä luokassa opiskelija oli palauttanut tehtävän minimivaatimuksilla, mutta ei ollut saanut tehtyä optiona olevia lisätehtäviä hyväksytysti. Yleisin ongelma näissä tapauksissa oli lisätehtävien vaatima ylimääräinen tietotarve. Lisäksi lisätehtävät vaativat usein erilaisen ratkaisustrategian kuin perustehtävät. Yleisesti ottaen vaikea ja pitkä harjoitus altistaa jättämään lisätehtävät kesken, vaikka ne eivät olisi vaikeita. Tirronen et al. [35] piti tätä tuntomerkkinä opiskelijan väsymiselle ja tunteelle että suoritus on riittävän valmis.

Viimeisenä luokkana oli kehittyvät tehtävät. Tässä luokassa opiskelija jatkoi harjoitustyönsä parantelua, vaikka se täyttäisi hyväksytyn kriteerit. Noin 60 prosentissa kehittämisluokkaan kuuluvista harjoitustöistä, opiskelija onnistui parantamaan työtään. Työn jatkokehittäminen oli huomattavasti yleisempää koneen tuottamissa palautteissa kuin ilman palautetta. Tästä päätellen automaattisella arvioinnilla saattaa olla positiivinen vaikutus opiskelijan käyttäytymiseen.

Tirrosen et al. [35] tutkimus on linjassa kirjallisuuden kanssa. Huonosti suunniteltu harjoitustyö tai puutteellinen tukimateriaali haittaa opiskelijan etenemistä tai jopa aiheuttaa opiskelun keskeyttämisen. Lisäksi menetelmällä voidaan tunnistaa kun opiskelija etenee harjoituksessa ”yritä ja erehdy”-menetelmällä. Samoin riittämätön palaute tai pelkkä hyväksytty/hylätty-arvio voi pahentaa opiskelijan ongelmia. Toisaalta formatiivisella palautteella on myös positiivista vaikutusta opiskelijan käyttäytymiseen.

Hameer et al. [12] ovat kirjoittaneet kokemuksista automaattisen arviointijärjestelmän käytöstä funktionaalisen ohjelmoinnin opetuksessa. Heillä on käytössä Learn-OCaml harjoitusalusta, joka on kehitetty MOOC-kurssille *Introduction to Functional Programming in OCaml*. Järjestelmässä opiskelija voi kirjoittaa koodia, tehdä tyyppitarkistuksia sekä ajaa koodia suoraan verkkoselaimessa. Järjestelmään on integroitu koodin arviointityökalu, jota opiskelija voi käyttää harjoitustöissään. Opis-

kelija voi käyttää arviointityökalua harjoituksen aikana ja parannella koodia niin kauan kunnes on tyytyväinen koodiinsa. Integroituun arviointityökaluun tehdään kullekin harjoitustyölle testejä TDD-menetelmällä (Test-Driven Development) Learn-OCaml:n omalla testikirjastolla. Hameer et al. [12] kokemusten perusteella laadukkaiden testitapausten kirjoittaminen on automaattiselle arviointijärjestelmälle aikaa vievää, opiskelijat käyttävät automaattitestejä koodin testaamiseen Learn-OCaml omien testaustyökalujen sijasta ja testauskirjaston rajoitteet tekevät vaativimpien testien tekemisestä haastavaa.

Rubio et al. [27] tutki opiskelijoiden kokemuksia Mooshak-arviointijärjestelmän käytöstä. Mooshak on WWW-pohjainen ohjelmointikilpailuiden hallintaan kehitetty järjestelmä, jonka yhtenä ominaisuutena on ohjelman automaattinen arviointi. Kyselytutkimuksessa etsittiin vastauksia siihen korvaako opiskelijat debug- ja testaustyökalut Mooshakilla, millaisia kokemuksia opiskelijoilla on Mooshakista ja vaikuttaako sen käyttö opetuksen keskeyttämisiin? Tutkimuksen perusteella automaattisella arvioinnilla ei ollut vaikutusta kurssin keskeyttämisiin [27, s. 14]. Opiskelijat pitivät Mooshakia hyvänä ideana, mutta sen antama palaute tulisi olla kattavampaa. Opiskelijat käyttivät sitä testaukseen ja virheiden etsimiseen. Ongelmia tuotti muun muassa opiskelijan oma erilainen kääntäjä ja tai sen asetukset.

Chen et al. [7] tutkivat ohjelmointityylin ja koodin laadun automaattista arviointia. Lähdekoodin luettavuus on yksi mitattavista suureista, kun puhutaan ohjelmistojen ylläpidosta ja jatkokehityksestä [7, s. 1203]. Tutkimuksessa käytettiin Checkstyle-ohjelmaa Java-lähdekoodin laadun ja koodaustyylin arviointiin. Käytetty arviointimalli on jaettu neljään tasoon (taulukko 3.1). Ensimmäisellä tasolla lähdekoodi käännetään ja tarkistetaan läpäiseekö se käännöksen virheittä. Toisella tasolla tutkitaan, onko koodi kopioitu joltain toiselta opiskelijalta. Kolmannella tasolla testataan ohjelman funktionaalinen toiminta ja viimeisellä tasolla arvioidaan ohjelmakoodin laatua tutkimalla kuinka hyvin opiskelija noudattaa annettua koodin tyyliohjeistusta.

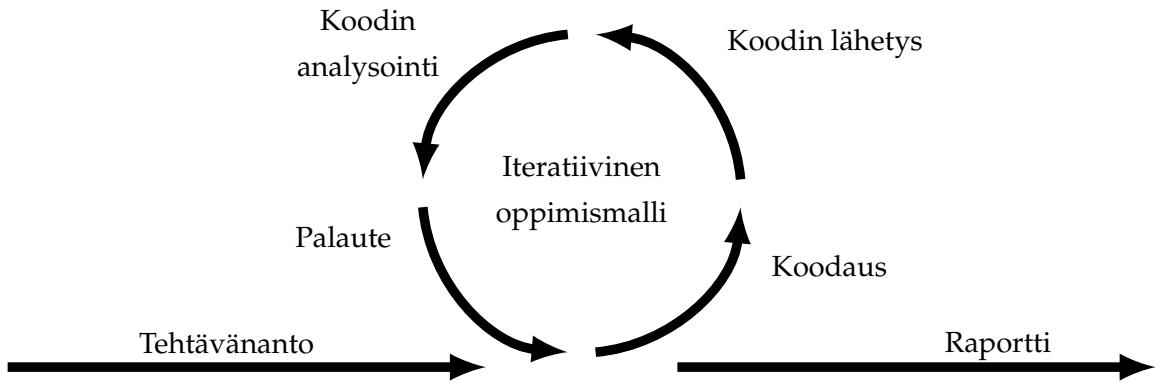
Taulukko 3.1: Arvioinnin luokittelu [7, s. 1210]

Taso 1	Onnistunut ohjelman käännös
Taso 2	Ei plagiointia
Taso 3	Läpäisee funktionaaliset testit
Taso 4	Läpäisee koodin laatuanalyysin

Chen et al. [7] kertoivat käyttävänsä iteratiivista oppimisprosessia, jossa opiske-

lija etenee harjoitustehtävässä vaihe vaiheelta eteenpäin (kuva 3.1). Hän voi lähettää koodin automaattiseen arviointiin niin usein kuin haluaa, mutta arvioinnin tulokset tallennetaan järjestelmään historiatietona. Yleisimpiä ongelmia koodin luettavuuden suhteen olivat:

1. lähdekoodin sisentämiseen liittyvät ongelmat (1787 kpl),
2. välilyöntiin liittyvät ongelmat (803 kpl),
3. muuttujien ja funktioiden nimeäminen (248 kpl),
4. sulkeiden käyttö (203 kpl),
5. metodien määrittelyt (155 kpl),
6. Javadoc-kommentoinnin ongelmat tai puutteet (54 kpl).



Kuva 3.1: Iteratiivinen oppimismalli

Chen et al. [7, s. 1209] totesivat, että koodin tyylioppailla ja sen varmistamisella harjoituksien yhteydessä testaamalla voisi olla merkittävä vaikutus ohjelmistoteollisuudelle, sillä ohjelmiston elinkaarikustannuksista 40 % tulee ylläpitovaiheesta. Toisaalta ongelmaa on lähdetty purkamaan myös toisesta suunnasta. Koodi-editoreihin on nykyisin varsin kattavasti integroitu koodin tarkastusohjelmia (linter), jotka tekevät saman asian. Esimerkiksi tässä tutkimuksessa käytetyn Checkstyle-ohjelman voi liittää Visual Studio Code -koodieditoriin.

Silva et al. [31] tutkivat puurakenteeseen perustuvaa automaattista arviointimenetelmää. He olivat havainneet, että useimmat automaattiset lähdekoodin arviointijärjestelmät käyttivät testivetoista arviointia, jossa arvioidaan mitä funktio

tai sovellus palauttaa lopputuloksena, ei *miten* lopputulos saatiin aikaiseksi. Järjestelmä koostui erilaisista analysaattoreista (syntaksi-, funktio-, rakenne- ja semanttinen analysaattori). Näiden analysaattorien tulokset on arvioitu VSS (Valuated State Space) -tekniikalla, joka tarkoittaa painotettua keskiarvoa. VSS-tekniikassa jokaiselle analysaattorille on annettu oma painoarvo asteikolla 0–5. Kun jokainen analysaattori arvioi lähdekoodia, saadaan tehtävän palautus arvioitua kattavammin. VSS-tekniikkaa ei artikkelissa kuvattu, mutta Porto et al. [25, s. 46] ovat esitelleet Valuated State Space -tekniikan toimintaperiaatteen omassa tutkimuksessaan.

Yhteenvetona Silva et al. [31] kokivat, että useasta analysaattorista koostuva menetelmän on hyödyllisempi kuin aikaisemmat arviointijärjestelmät. Sillä oli mahdollista tunnistaa erityyppisiä ei odotettuja toimintatapoja, kuten eri kieliversioiden käyttöä, odottamattomien tietorakenteiden käyttöä sekä tehtävänantoon liittyviä ongelmia.

Ala-Mutka et al. [1] tutkivat C++-kurssin harjoitusten koodin tyylin automaattista arviointia. Arvioinnin tavoitteena oli opettaa opiskelijoille parempia työskentelymenetelmiä koodin tyylioppaiden avulla ensimmäisten kurssien aikana, ja tätä pyrittiin arvioimaan automaattisella arviointijärjestelmällä. Koodauksen tyylillä on merkitystä koodin laatuun ja luettavuuteen. Ohjelmia tehdään yleensä tiimeissä, jolloin koodin luettavuudella on suuri merkitys koodarien tehokkaalle yhteistyölle.

Näitä ongelmia on lähdetty ratkaisemaan yhteisillä tyylioppailla ja niiden säännöillä sekä automaattisella arvioinnilla. Automaattisella tyylien arvioinnilla Ala-Mutka et al. [1, s. 245–246] mukaan oli vaikutusta opiskelijoiden tuottaman koodin laatuun ja se antoi opettajalle enemmän aikaa paremman palautteen ja kurssin merkittävimpien asioiden tekemiseen. Automaattinen arviointi mahdollistaa laajojen kurssikokonaisuuksien toteuttamisen ilman apuopettajia.

Gerdes et al. [10] tutkivat sähköisen ASK-ELLE tutorin käyttöä Haskell-ohjelmointikurssin aikana. ASK-ELLE toimi vaiheittain tarjoten vinkkejä, arvioi keskeneräisiä sovelluksia ja antoi palautetta niistä. Tehtävänannot on tallennettu ASK-ELLE-järjestelmään ja niiden mukana on muutamia malliratkaisuja sekä harjoitustyön vaatimukset. Arviointipalautteeseen voidaan liittää malliratkaisuja sekä vaatimukset. Tutkimuksella oli kolme tavoitetta: a) asteittainen oppiminen harjoitustyötä tehtäessä, b) automaattinen palautteenanto sekä c) opettajakohtaiset harjoitukset, ratkaisut ja vaatimukset.

ASK-ELLE tarjosi informatiivisen tavan opiskella ohjelmointia suoraan verkkoselaimella. Selainnäkymässä on tehtävänannon lisäksi koodieditori, selitys tehtäväs-

tä, vinkki sekä esimerkkitoteutus opiskeltavasta asiasta [10, s. 4]. Opettaja voi mukauttaa tehtävään useita erilaisia ratkaisuvaihtoehtoja ja tarjota niille vinkkejä ja ratkaisumalleja [10, s. 7]. ASK-ELLE tukee asteittaista ohjelmoinnin opiskelua, tarjoaa laadukasta dynaamista palautetta ja vinkkejä harjoituksen tekemiseen. Mielenkiintoinen näkökulma oppimisen kannalta on esittää opiskelijalle useita ratkaisuvaihtoehtoja. Jatkokehityskohteina ja haasteina pidettiin hyväksytty/hylätty-arviointia, syntaksivirheiden esitystapaa ja malliratkaisujen parempaa integrointia yksittäiseen harjoitustehtävään [10, s. 30–32].

Insa et al. [17] vertailivat kokeen automaattista arviointia manuaaliseen arviointiin Java-ohjelmointikurssin yhteydessä. Tutkimuksessa mielenkiintoisinta oli opettajien arviointien virhemäärä. Noin viisi prosenttia arvioista oli vääriä. Opettajat tekivät erilaisia virheitä, ja virheet olivat joskus opiskelijalle negatiivisia, joskus positiivisia [17, s. 62]. Tulos ei ole hyväksyttävä, koska sillä on vaikutusta opiskelijan koulumenestykseen.

Insa et al. [17] tutkimuksessa todettiin automaattisen arvioinnin eduiksi nopeus, oikeudenmukaisuus ja itsenäisyys. Automaattinen arviointi ja palaute olivat välitöntä ja opiskelija pystyi tekemään itsearvion kokeesta ja ottaa oppia tuloksesta. Manuaalisessa arvioinnissa tuloksia saatiin tyypillisesti parin viikon viiveellä, joten itsearvion tekeminen on hankalaa, kun asiat eivät ole enää muistissa. Automaattista arviointia pidettiin oikeudenmukaisena, koska siinä on kaikilla samat kriteerit. Itsenäisyydellä tarkoitettiin sitä, ettei aikaisempi koe tai koulumenestys vaikuttanut arviointiin [17, s. 62].

Ala-Mutka [2] kävi omassa artikkelissaan läpi automaattisen arvioinnin ominaisuuksia. Ensimmäinen haaste on sovelluksen dynaaminen testaus. Kay et al. [19] totesivat, ettei ole mahdollista arvioida sovelluksia järjestelmällisesti ja läpikohtaisesti ilman automatisoitua testausta. Tämä korostuu sovelluksien (C++) dynaamisissa ominaisuuksissa, jolloin pienessä sovelluksessa voi olla lukemattomia ratkaisupolkuja. Automaatio tarjoaa systemaattisen tavan käydä läpi suuren määrän ratkaisuvaihtoehtoja. Itsessään opiskelijan koodin suoritus opettajan koneella on suuri riski. Opiskelijan koodi voi yrittää tuhota, lukea tai muokata tiedostoja ajoympäristössä. Se voi kuluttaa kaiken muistin tai prosessoriajan, joten koodia pitää ajaa omassa turvatussa ajoympäristössä eli hiekkalaatikossa [2, s. 87].

Toinen tapa arvioida sovelluksia on tarkistaa palauttaako sovellus oikean arvon. Funktionaalista testausta tehdään yleensä useilla data-aineistoilla. Ohjelmalle viedään testiarvoja ja katsotaan millaisen arvon sovellus palauttaa. Osa testiaineistoista

tyytyy sovelluksen tilakoodin tarkistamiseen, osa vertailee tekstiarvoja testiaineistoon [2, s. 88].

Kolmas Ala-Mutkan [2, s. 88] esittelemä tapa arvioida sovellusta on sen suorituskyvyn mittaus. Hän viittasi Jackson et al. [18, s. 338] esimerkkiin, jossa he käyttivät kehittämäänsä ASSYST-sovellusta automaattiseen arviointiin. Sillä pystyi mittaamaan prosessoriaikaa yksinkertaisesti kutsumalla järjestelmän *timer*-rutiinia. Toinen tapa mitata suorituskykyä oli laskea, kuinka monta kellojaksoa sovellus käytti mitattavaan sovellukseen. Jackson et al. [18, s. 339] vaativat opiskelijoilta omaa dataaineistoa ja valmiiksi mietittyjä testejä omalle harjoitukselle. Ajatuksena oli mitata kuinka kattavasti testiaineisto testaa opiskelijan tuottamaa koodia. Ajatus oli varsin mullistava ja aikaansa edellä, sillä opiskelija joutuu miettimään sovellusta testaamisen näkökulmasta.

Neljäs arviointimenetelmä oli staattinen arviointi, jossa perinteisesti opettaja on silmäillyt koodin läpi ja antanut arvion sen perusteella [2]. Automaattinen arviointi voi käydä koodia formaalisella arvioinnilla, jossa tarkistetaan, onko koodi syntaksin ja semantiikan osalta toteutettu oikein. Tästä menetelmästä Ala-Mutkalla ei ollut esimerkkiä, mutta esimerkiksi Truong et al. [36] käytti omassa tutkimuksessaan staattista arviointia ohjelman rakenteen tutkimiseen.

Viidentenä arviointitapana oli koodin tyylin arviointi [2]. Tyylin osalta näkemykset olivat samat kuin Chen et al. [7, s. 1209] ja Ala-Mutkan et al. [1] aikaisemmassa tutkimuksessa.

Kuudentena arvioinnin kohteena Ala-Mutka esitteli toiminnalliset ohjelmointivirheet. Toiminnallisia virheitä tunnistettiin dynaamisesti testiaineistolla ja staattisella arvioinnilla, usein ne ovat sidoksissa ohjelmointityyliin. Artikkelissaan hän viittasi mielenkiintoiseen Xie et al. [38] tutkimukseen, jossa arvioitiin turhan koodin määrää. Esimerkkinä turhasta koodista ovat käyttämättömät muuttujat ja funktiot, idempotenttit operaatiot, mahdottomat tilat sekä moninkertaiset toistot tai laskutoimitukset [38, s. 51]. Idempotentteja operaatioita ovat operaatioita, joissa ei tehdä mitään. Esimerkiksi muuttuja sijoittaa itse itsensä: $(x = x)$, muuttuja toimii jakolaskun osoittajana ja nimittäjänä (x/x) tai palauttaa aina tosi-arvon $(x\&x)$ funktiosta [38, s. 52].

Seitsemäntenä menetelmänä esiteltiin ohjelmistometriikat, jotka yksinkertaisimmillaan voivat olla esimerkiksi koodirivien, funktioiden, metodien, muuttujien tai olioiden lukumääriä [2]. Numeerista metriikkatietoa on helppo kerätä automaattisen arvioinnin yhteydessä. Esimerkkinä Ala-Mutka käytti Jackson et al. [18, s. 338]

artikkelia sekä McCaben [21] sovelluksen syklomaattisen monimutkaisuuden teoriaa. Teoriassa monimutkaisuus koostuu siitä, kuinka monta itsenäistä suorituspolkua sovelluksesta löytyy. Syklomaattisen monimutkaisuuden teoria, jossa koodin tai graafin G syklomaattinen monimutkaisuus v voidaan esittää kaavalla [21, s. 308]:

$$v(G) = e - n + p.$$

Teoria koostui solmupisteistä n (node), viivoista e (edge) sekä komponenttien lukumäärästä p. Teoriassa solmupisteellä tarkoitettiin haarautuvaa koodia, kuten if-else-ehtolausetta. Viivat tarkoittavat ehtolauseen haarautumien lukumäärää ja komponenttien lukumäärä vastaa esimerkiksi funktioiden lukumäärää [21, s. 308].

Seuraavana arviointikohteena Ala-Mutka käsitteli, miten ohjelmiston suunnittelua voidaan arvioida. Esimerkkinä hän käytti Thorburn et al. [34] tutkimusta, jossa ohjelmallisesti verrattiin opettajan suunnittelemaa ohjelmiston rakennetta opiskelijan toteutukseen. Jos opiskelijan sovellus noudatti opettajan malliratkaisun rakennetta ja funktiot palauttivat oikeita tuloksia, katsottiin opiskelijan sovelluksen olevan hyvin suunniteltu. Toisena esimerkkinä hän käytti Truong et al. [36] tutkimusta, jossa staattisella analyysillä tutkittiin ohjelman rakennetta ja suunnittelua. ELP-järjestelmässä opettajan malliratkaisu ja opiskelijan palauttama lähdekoodi muutettiin ensin semanttisesti luettavampaan XML-muotoon. Näitä kahta vertailemalla voitiin koneellisesti arvioida, kuinka hyvin ohjelmakoodi on toteutettu [36, s. 320–321].

Viimeisinä arviointikohteina Ala-Mutka esitteli hakusanoihin perustuvan analyysimenetelmän ja plagioinnin tunnistuksen [2]. Esimerkkinä hakusanoihin perustuvasta analyysistä hän käytti Saikkosen et al. [28, s. 134–135] tutkimusta, jossa säännöllisen lausekkeen (regular expressions) avainsanoilla voidaan hakea koodin osia arviointia varten. Tässä työssä plagiointi on rajattu pois vaatimuksissa, joten sitä ei käsitellä tässä yhteydessä.

Hung et al. [15] tutkivat ohjelmistojen metriikoiden käyttöä sovelluksen automaattisessa arvioinnissa. Tutkimuksessa he esittelivät lyhyesti Halsteadin [11] ja McCaben [21] sovelluksen monimutkaisuuden laskentametriikat sekä Reesin [26] tyylimetriikat. Tutkimuksessa käytettiin seuraavia koodimetriikoita:

- koodilistauksen rivimäärä,
- koodilauseiden lukumäärä,
- koodirivien lukumäärä pois lukien kommentit,

- kommenttien lukumäärä,
- operaattoreiden ja operandien summa ($N = N_1, N_2$),
- uniikkien operaattoreiden ja operandien summa ($n = n_1, n_2$),
- ohjelman volyymi ($V = N\,log_2\,n$),
- varoitusten lukumäärä puuttuvista operaattoreista ja operandeista,
- ajonaikaiset virheet ja käännösvirheet.

Hung et al. [15] tutkimuksen perusteella näillä metriikoilla analysointiohjelma kykeni erottelemaan hyvät ja huonot ohjelmoijat toisistaan.

Useissa tutkimuksissa mainittiin myös lyhenne AST (Abstract Syntax Tree), joka on yksi lähdekoodin käännön vaiheista [20][10][12][17]. Kirjassaan Holub [14] esittelee lähdekoodin käännösprosessin, joka koostuu kuudesta vaiheesta: a) esikäsittely (preprocessor), b) saneistaminen (lexical analyzer, tokenizer), c) parsinta (syntax and parse trees), d) binäärikoodin generointi, e) koodin optimointi ja f) tallennus suoritettavaksi ohjelmaksi [14, s. 2–5]. Esimerkiksi Truong et al. [36] tutkimuksessa Java-lähdekoodin arviointia varten toteutettiin käännösprosessin vaiheet a–c, jolloin koodia oli helpompi verrata mallitoteutusta varten.

3.3 Yhteenveto

Taulukkoon 3.2 on koostettu automaattista arviointia koskevia vahvuuksia ja haasteita, joita havaittiin katsauksessa aikaisempiin aiheesta tehtyihin tutkimuksiin. Kirjallisuuden perusteella automaattisen arvioinnin suurimpia hyötyjä oli opettajan ja oppilaan tuottavuuden nousu, laadukas palaute sekä nopea ja parempi arviointi opiskelijalle. Jos järjestelmä tuki konstruktiivista pienin askelin etenevää oppimismallia, sen koettiin antavan parempaa tukea opiskelijalle parantaen koodin laatua ja luettavuutta.

Haasteiksi koettiin tehtävien antoon liittyvät haasteet, työläs testitapausten laadinta, on/off-tyyppinen arviointi ja huono palaute sekä iteroiva yrityksen ja erehdyksen kautta oppiminen. Vaikka huono tukimateriaali mainittiin vain kerran, oli useimmissa tutkimuksissa rivien välistä luettavissa, ettei oppimateriaali ollut ajan tasalla tai linjassa harjoitustehtävän tavoitteiden kanssa. Oppimateriaalin tulee tukea itsenäistä opiskelua.

Taulukko 3.2: Automaattisen arvioinnin (AA) vahvuudet ja haasteet

	Tirronen et al. [35]	Hameer et al. [12]	Rubio et al. [27]	Chen et al. [7]	Silva et al. [31]	Ala-Mutka et al. [1]	Gerdes et al. [10]	Insa et al. [17]	Ala-Mutka [2]	Ala-Mutka [34]	Truong [36]	Saikkosen et al. [28]	Yhteensä
Vahvuudet													
Laadukas palaute opiskelijalle	x			x		x	x						4
Nostaa työn tuottavuutta						x	x	x		x	x	x	6
Parempaa arviointia				x	x			x		x		x	5
Parempaa koodin laatua				x		x					x		3
Parempaa koodin luettavuutta				x		x			x				3
Konstruktiivinen oppiminen				x		x	x						3
Tukee opiskelijan suoritusta							x				x		2
Välitön itsearvio								x	x		x		3
Vähentää arviointivirheitä								x		x			2
Arvioinnin saatavuus									x		x	x	3
Haasteet													
Teht. antoon liittyvät ongelmat	x	x			x						x		4
Testitapauksen laadinta		x									x	x	3
Puutteellinen tukimateriaali	x												1
Iteroiva oppiminen	x			x									2
Hyväksytty ja hylätty arviointi	x				x		x						3
AA:n käyttö kääntäjänä		x	x										2
Huono palaute opiskelijalle	x		x				x						3
Opiskelijan versio- ja asetusongelmat			x									x	2
Plagiointi				x									1
Koodin analysointi					x		x						2
Opisk. odottamat. käyttäytyminen					x								1
Koodin turvallinen kääntö					x							x	2

Näiden lisäksi optionaalisia vaatimuksia aikataulun puitteissa olisi AST-pohjainen koodisyntaksin tarkistus, tuki useiden analyysimenetelmien käytölle arviointisovelluksessa [24], metriikoiden käyttö arvioinnissa sekä formatiivisen ja kannustavan palautejärjestelmän kehitys.

4 Sovelluksen kehitystyö

Tämän kehittämistutkimuksen tavoitteena oli löytää tietotekniikan MOOC-kurssien arviointihaasteelle tekninen ratkaisu. Suuresta opiskelijamäärästä johtuen harjoitustöiden arviointi manuaalisesti on erittäin haastavaa, tarkkuutta vaativaa ja työlästä. Perinteisessä luokka- tai etäopetuksessa osallistuu ohjelmoinnin kurssille tyypillisesti 10 – 30 opiskelijaa, jolloin koodien arviointi on vielä tehtävissä manuaalisesti. Vastaavasti MOOC-kursseille osallistuu helposti satoja tai tuhansia opiskelijoita, jolloin koodien manuaalinen arviointi on mahdotonta. Tästä johtuen MOOC-kurssien osallistujamääriä joudutaan hyvin usein rajoittamaan.

Kehitetty ratkaisu koostuu käyttöliittymästä, jolla opiskelija voi lähettää ohjelmointiharjoitustehtävän palvelimella toimivalle arviointisovellukselle. Arviointisovellus käyttää kolmea testausmenetelmää: merkkijonotestiä, staattista testausta sekä tarkistaa kääntyykö lähdekoodi oikein kääntäjässä. Jokaista ohjelmoinnin harjoitustehtävää varten on muodostettu oma testiaineisto, jota vasten opiskelijan harjoitustyötä testataan.

Avainsanatestissä etsitään lähdekoodista opettajan määrittämiä avainsanoja, kuten funktioiden tai muuttujien nimiä sekä niiden arvoja. Staattisessa analyysissä ohjelmakoodi puretaan semanttiseksi listaksi, jolloin siitä voidaan etsiä opettajan määrittämiä ohjelmarakenteita. Lähdekoodi myös käännetään C++-kääntäjällä, jolla varmistetaan että harjoitustyö kääntyy oikein. Jos opiskelijan lähettämä lähdekoodi kääntyy onnistuneesti ja siitä löytyy vaaditut avainsanat sekä ohjelmarakenteet, niin arviointisovellus palauttaa tulokset käyttöliittymälle tarkasteltavaksi. Arviointisovelluksella voidaan myös muodostaa opettajalle puolivalmiita testiaihioita. Seuraavissa luvuissa on kuvattu järjestelmän vaatimusmäärittelyä, arkkitehtuuria ja sovelluksen toimintaa tarkemmin.

4.1 Sovelluksen arkkitehtuuri ja vaatimukset

Ongelma-analyysin perusteella määriteltiin seuraavat vaatimukset automaattiselle arviointisovellukselle:

- automaattinen lähdekoodin staattinen analyysi [5][36],

- skaalautuva sovellus- ja teknologia-arkkitehtuuri [4][39],
- lähdekoodin turvallinen kääntäminen palvelimella [8] [13][2][6][16],
- selkeä REST API-arkkitehtuuri ja
- helppo tapa luoda automaattisia testejä [12][36].

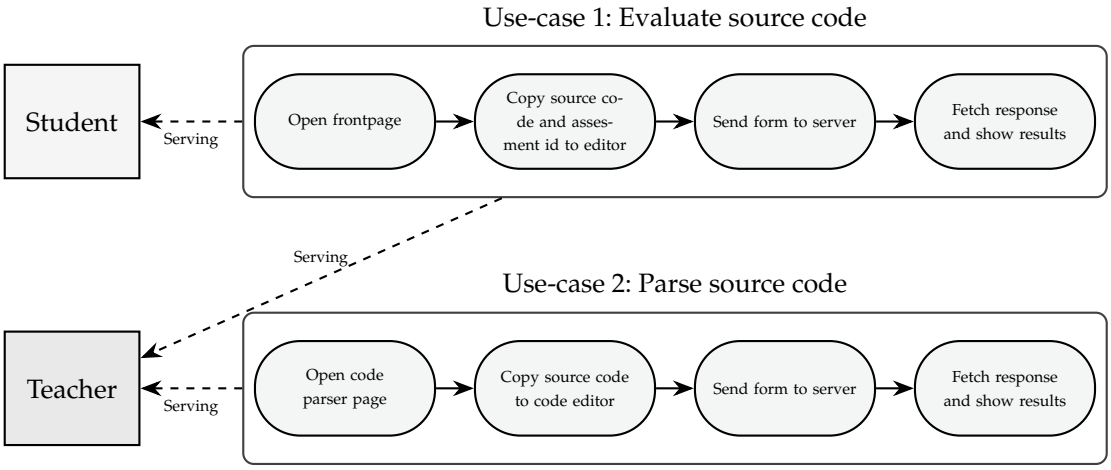
Vaatimukset on johdettu ongelma-analyysin katsauksesta aikaisempiin aiheesta tehtyihin tutkimuksiin. Ensimmäisenä vaatimuksena on automaattinen lähdekoodin staattinen analyysi, jossa koodille tehdään avainsanahaku ja semanttinen vertailu opettajan testin perusteella [5][36]. Toisena vaatimuksena sovellus- ja teknologia--arkkitehtuurin tulee skaalautua erilaisille käyttäjämäärille, sillä MOOC-kursseilla voi olla vaihteleva määrä opiskelijoita kerrallaan [4][39].

Vaatimus lähdekoodin käännöstä omassa suljetussa ajoympäristössä tuli Ala-Mutkan [2] ja Ihantola et al. [16] tutkimuksien pohjalta. API-pohjainen arkkitehtuuri valittiin, jotta järjestelmä voidaan liittää mahdollisimman moneen oppimisympäristöön. Kirjallisuuden perusteella opettajilla oli haasteena luoda käytettyyn järjestelmään soveltuvia testitapauksia Hameer et al. [12] ja Truong [36]. Semanttista analyysiä varten opettajalla tulee olla oma työkalu, joka muodostaa annetusta lähdekoodista puolivalmiin testitapauksen.

4.2 Käyttötapaukset

Arviointisovelluksessa on kaksi käyttötapausta (kuva 4.1). Ensimmäisessä opiskelija tai opettaja voi arvioida harjoitustyön lisäämällä etusivun lomakkeelle harjoitustyön tunnistekoodin sekä kopioimalla lähdekoodin sivun koodieditoriin. Painamalla ”Evaluate source code”-nappia lähettää sovellus koodin palvelimelle arvioitavaksi. Kun koodi on arvioitu, tulostetaan arvio välittömästi käyttäjän selaimelle.

Toisessa käyttötapauksessa opettaja voi viedä minkä tahansa C++-koodin järjestelmälle ja muodostaa koneelliset arviointikriteerit puoliautomaattisesti. Palvelin käy lähdekoodin läpi ja etsii merkkijonotestejä (string tests) varten kaikkien muuttujien ja funktioiden nimet. Samoin se muodostaa koodiriveistä ketjutestit (chain tests).



Kuva 4.1: Sovelluksen käyttötapaukset

4.3 Sovellusarkkitehtuuri

Kuvassa 4.2 on esitetty käyttötapaus 1 ja sovellusarkkitehtuuri samassa kuvassa. Järjestelmä koostuu selaimessa ajettavasta edustasovelluksesta sekä palvelimessa ajettavasta arviointisovelluksesta. Järjestelmän edustasovellus on hyvin yksinkertainen. Se koostuu kolmesta ohjelmistokomponentista: Frontpage-, Parser- ja Editor-komponentista sekä Router- ja Http-palveluista. Router-palvelun tehtävänä on ohjata loppukäyttäjä aina oikean komponentin pariin. Etusivulla se lataa Frontpage-komponentin ja Code Parser -sivulla ParserComponentin. Kumpikin sivu käyttää EditorComponentin koodieditoria. EditorComponent tekee REST-kyselyitä arviointisovellukselle HttpServicen avulla.

Varsinainen arviointisovellus toimii palvelimella. Arviointisovelluksessa on sovellusreititin, joka tarjoaa kaksi rajapintaa: analyysi- ja arviointirajapinnat. Analyysi rajapinnassa suoritetaan basicValidate-komponentista koodin esikäsittely sekä koodin semanttisen analyysi. Analyysin tuloksena saadaan JSON-koodi, jossa on automaattisesti luotu harjoitustehtävän ID, merkkijonotestit sekä yksinkertaisimmat ketjutestit (listaus 4.1). Tämä puolivalmis testiaihio palautetaan edustasovellukselle näytettäväksi. Kurssin opettaja voi itse täydentää aihiota omilla testeillä ja testata muutoksia arviointisovelluksen puolella.

Arviointirajapinnassa käynnistyy opiskelijan lähdekoodin arviointi. Lähdekoodi tallennetaan palvelimelle, käännetään C++-kääntäjällä, luetaan opettajan laatimat testit JSON-muodossa, ladataan basicValidate-komponentti, tehdään semanttinen analyysi ja lopuksi verrataan analyysin tuloksia opettajan testeihin. Arvioinnin

lopputulos palautetaan edustasovellukselle näytettäväksi.

Listaus 4.1: Semanttisen analyysin lopputulos

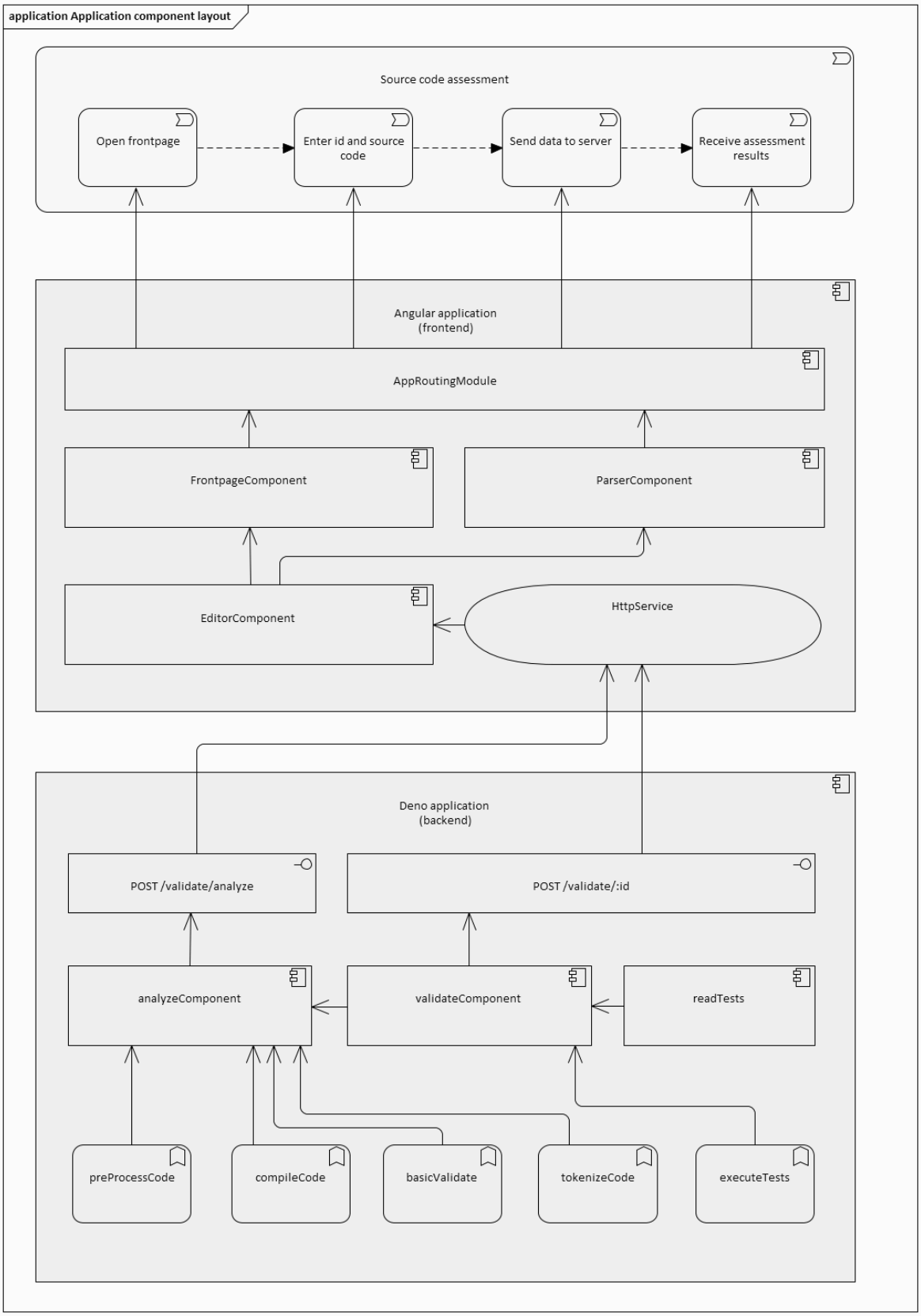
```
{
  "7f893c33efb3c79f17dee37a98364c3a": {
    "stringTests": [
      "ledPin", "inPin", "setup", "pinMode", "loop",
      "val", "digitalRead", "digitalWrite"
    ],
    "chainTests": [
      [ "TYPE", "WORD", "ASSIGN", "NUMBER", "SEMICOLON" ],
      [ "WORD", "LP", "WORD", "COMMA", "NUMBER", "WORD", "NUMBER", "RP"
        , "SEMICOLON" ]
    ]
  }
}
```

4.4 Teknologia-arkkitehtuuri

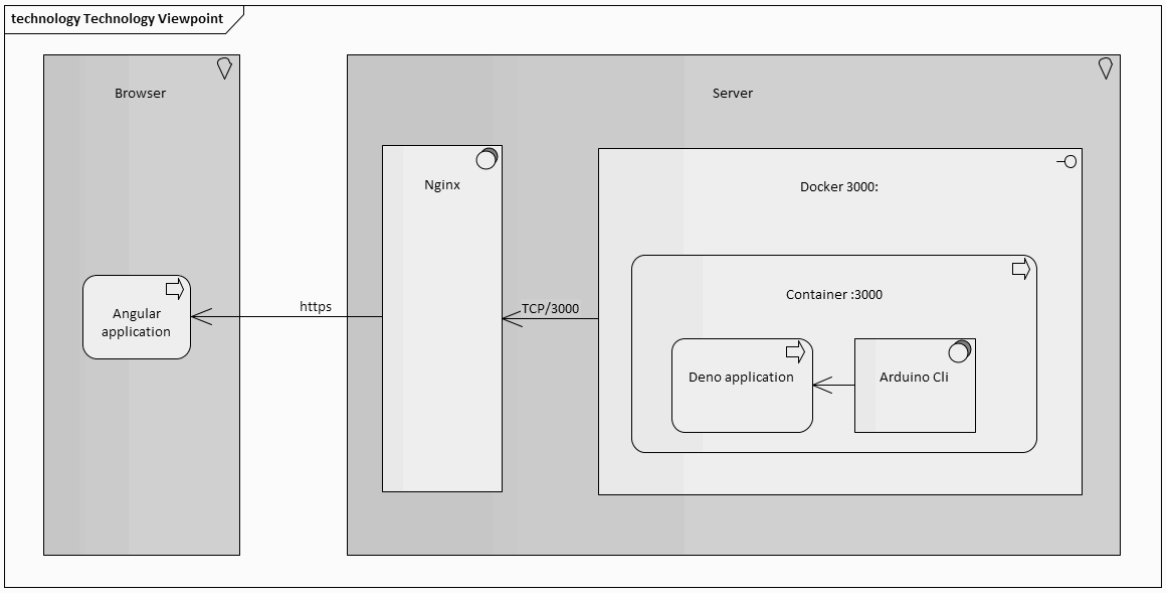
Kuvassa 4.3 on esitetty käytetty teknologia-arkkitehtuuri. Siinä on selaimessa suoritettava, toteutettu WWW-pohjainen edustasovellus sekä mikropalveluperiaatteita [22, s. 2] noudattava palvelimella ajettava arviointisovellus. WWW-pohjainen edustasovellus on toteutettu Angular 9 SPA -sovelluksena TypeScript, HTML ja SCSS-kielillä.

Arviointisovellus on toteutettu TypeScript-kielellä ja Deno-kirjastolla, joka on sijoitettu Docker-konttiin. Docker-kontti mahdollistaa käytännössä kaksi teknistä vaatimusta: sovelluksen skaalautuvuuden sekä C++-koodin käännön ja suorittamisen turvallisesti palvelimella omassa Docker-hiekkalaatikossa (sandbox). Docker-konttiin on asennettu Arduino CLI -sovellus, jota arviointisovellus käyttää opiskelijan lähdekoodin kääntämisessä.

Docker-ajoympäristön edustana toimii Nginx-webbipalvelin, joka toimii käänteisenä proxyna (Reverse Proxy), kuorman tasaajana sekä jakelee rajapinnat mikropalveluille. Docker-ajoympäristön ansiosta arviointijärjestelmä voidaan sijoittaa pilvipalveluihin tai omalle palvelimelle.



Kuva 4.2: Liiketoiminta- ja sovelluskerros



Kuva 4.3: Teknologiakerros (Archimate 3)

4.5 Arviointisovelluksen toiminta

Arviointisovellus käyttää kolmea eri testausmenetelmää: lähdekoodin kääntötesti, merkkijonotesti sekä staattinen testaus. Näin harjoitustyön arviointi on hyvin suoraviivainen toimenpide ja toimintaperiaate käy ilmi kuvasta 4.4. Kun arviointisovellus saa POST-sanoman mukana arviointitunnisteen ja lähdekoodin, se luo väliaikaisen lähdekooditiedoston omaan alihakemistoon. Tämän jälkeen sovellus käännetään Arduino CLI -sovelluksella. Käännöksen lopputulos tallennetaan muistiin.

Käännöksen jälkeen lähdekoodille tehdään esikäsittely, jossa Arduinon headertiedostojen vakiot viedään opiskelijan lähdekoodiin. Esikäsittelyn jälkeen koodi on valmis saneistamiselle (tokenizer), jossa C++-lähdekoodi muutetaan saneiksi (token). Tällä muutoksella lähdekoodin eri rakenteet saavat semanttisen merkityksen. Kun lähdekoodi on muutettu semanttiseksi listaksi, luetaan opettajan laatima testitiedosto järjestelmään ja aletaan etsimään vastaavuuksia semanttisesta listasta.

Testejä on kahta tyyppiä: merkkijonotestejä ja ketjutestejä. Ensin suoritettavassa merkkijonotestissä etsitään listasta haluttuja merkkijonoja. Esimerkiksi listauksessa 4.1 semanttisessa analyysissä on tunnistettu muuttujien, metodien ja funktioiden nimiä saneiksi ”WORD”. Niinpä merkkijonotestissä etsitään semanttisesta listasta hakusanaa ledPin. Jos merkkijono löytyy listasta, muutetaan merkkijonotestin status *pass*-tilaan, muutoin se asetetaan *fail*-tilaan.

Listaus 4.2: Muuttujan määrittely

```
1  // LED_BUILDIN on vakio, jonka arvo on 13
2  int ledPin = LED_BUILTIN;
3
4  /** Semanttinen esitystapa **
5  [ "TYPE", "WORD", "ASSIGN", "NUMBER", "SEMICOLON" ]
6  */
```

Ketjutestissä etsitään semanttisia ketjuja. Listauksessa 4.2 on int-tyyppiseen muuttujaan asetettu arvo 13. Kun se muutetaan semanttiseen muotoon, sitä on helppo ja nopea etsiä semanttisesta listasta. Mikäli muuttujan määrittelyn ketjutesti löytyy semanttisesta listasta esimerkiksi kolme kertaa, asetetaan ensimmäisen testin *passed* arvoksi 3. Ketjutestit eivät ota kantaa siihen, mikä on muuttujan nimi tai siihen sijoitettu arvo, vaan sillä tutkitaan, onko harjoitustehtävässä toteutettu vaadittuja rakenteita.

Listaus 4.3: Arvioinnin paluusanoma

```
1  {
2    "compiler": "Sketch uses 892 bytes (2%) of program storage space.
        Maximum is 32256 bytes.\nGlobal variables use 9 bytes (0%) of
        dynamic memory, leaving 2039 bytes for local variables. Maximum is 2
        048 bytes.\n",
3    "tests": {
4      "id": "f7f3275eb5704b1a00f665f767b97c2c",
5      "stringTestFails": 0,
6      "stringTests": [
7        { "string": "setup", "status": "pass" },
8        { "string": "loop", "status": "pass" },
9        { "string": "ledPin", "status": "pass" },
10       { "string": "pinMode", "status": "pass" },
11       { "string": "inPin", "status": "pass" }
12     ],
13     "chainTest": [
14       { "test": 0, "passed": 3 },
15       { "test": 1, "passed": 1 }
16     ]
17   }
18 }
```

Merkkijono- ja ketjutestien lisäksi arviointisovellukseen toteutettiin metriikoita varten oma metriikkatestin prototyyppi, jossa on esimerkkitoteutuksena laskurit muuttujien, funktioiden sekä koodirivien lukumäärälle. Metriikkatesti on jatkokehityskohde, ja se on rajattu tämän kehittämistutkimuksen ulkopuolelle.

Kun kaikki testit on suoritettu, muodostetaan edustasovellusta varten JSON muotoinen paluusanoma (listaus 4.3), joka koostuu nimi-arvo-pareista. Paluusanomassa on kaksi jäsentä: compiler ja tests. Compiler-jäsen pitää sisällään C++-kääntäjän paluuarvon. Jos käännös onnistui, palautuu tietoja käännöksestä. Epäonnistuneesta käännöksestä palautuu kääntäjän virheilmoitus.

Tests-jäsenessä on merkkijono- ja ketjutestien lopputulokset sekä testin tunniste-ID sekä epäonnistuneiden testien lukumäärä, joka ei ole tässä versiossa käytössä. Lopuksi paluusanoma palautetaan edustasovellukselle.

4.6 Analysointisovelluksen toiminta

Analysointisovellus toimii pitkälti samalla tavoin kuin arviointisovellus (kuva 4.4). Analysointi alkaa ensin koodin esikäsittelyllä, jossa Arduinon header-tiedostojen vakiot viedään lähdekoodiin ja koodista siivotaan kommentit pois.

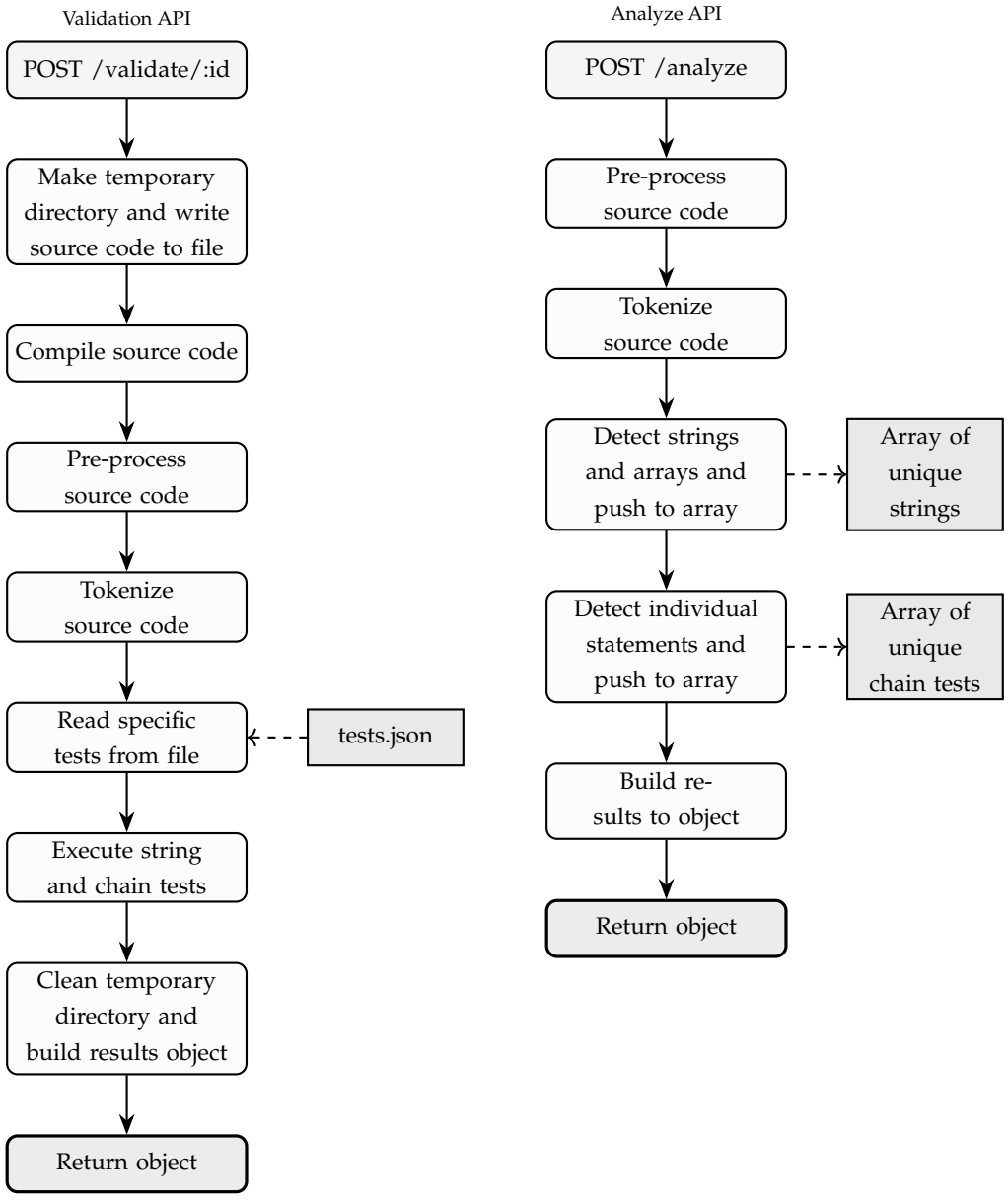
Esikäsittelyn jälkeen tehdään lähdekoodille saneistus. Saneistuksen jälkeen tehdään semanttista analyysiä, jossa koodista tunnistetaan muuttujien, metodien ja funktioiden rakenteita. Kun rakenne on tunnistettu, sen nimi tallennetaan string-testiin ja itse rakenne ketjutestiin. Listauksessa 4.4 on esimerkki analyysin lopputuloksesta.

4.7 Käyttöliittymä

Vaikka käyttöliittymän suunnittelu ei kuulunut tämän kehittämistutkimuksen ydinasioihin, sisällytettiin se konseptiin. Ei riitä, että tiedetään palvelinsovelluksen toimivan, vaan myös rajapintojen pitää toimia todistettavasti uusimmissa React-, Vue- ja Angular UI -sovelluskehitysympäristöissä. Käyttöliittymän on oltava myös käytettävä ja saavutettava [24].

Käyttöliittymässä on kaksi näkymää: etusivu, jossa opiskelija voi testata harjoitustehtävää sekä koodin analyysisivu, jossa opettaja voi tehdä omalle koodille analyysin ja saada puolivalmiin testipohjan tulevalle harjoitustyölle.

Käyttöliittymän suunnittelussa oli neljä vaatimusta: hyvä käytettävyys, saavu-

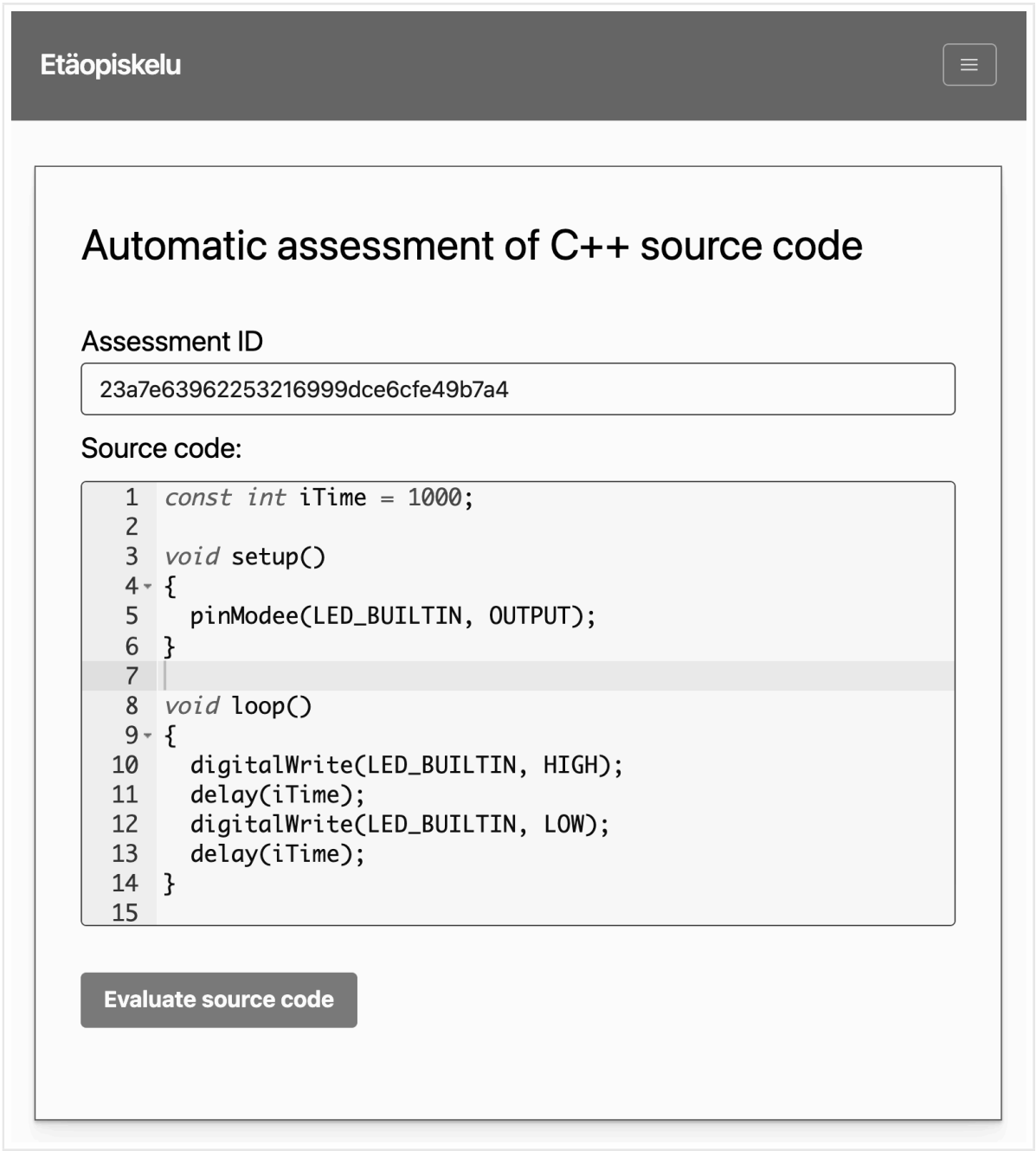


Kuva 4.4: Sovelluksen toiminta

Listaus 4.4: Analyysin paluusanoma

```
{
  "541b83c75b8885217e82ed28dfae21b8": {
    "stringTests": [
      "ledPin",
      "inPin",
      "setup",
      "pinMode",
      "loop",
      "digitalRead",
      "digitalWrite"
    ],
    "chainTests": [
      ["TYPE", "WORD", "ASSIGN", "NUMBER", "SEMICOLON"],
      [
        "WORD",
        "LP",
        "WORD",
        "COMMA",
        "NUMBER",
        "WORD",
        "NUMBER",
        "RP",
        "SEMICOLON"
      ],
      ["WORD", "ASSIGN", "WORD", "LP", "WORD", "RP", "SEMICOLON"],
      ["WORD", "LP", "WORD", "COMMA", "WORD", "RP", "SEMICOLON"]
    ]
  }
}
```

tettavuus, responsiivisuus sekä selkeä palaute opiskelijalle. Hyvä käytettävyys mielessä suunniteltiin käyttöliittymästä mahdollisimman selkeä ja minimalistinen. Harjoituksen arviointia varten opiskelijan tarvitsee syöttää harjoituksen tunniste-ID (Assessment ID) ja kopioida oma lähdekoodi selainikkunaan (Source code) sekä painaa *Evaluate source code* -nappia, joka käynnistää automaattisen arvioinnin (kuva 4.5).



Kuva 4.5: Arviointisovelluksen käyttöliittymä

Kun arviointi on suoritettu, käyttöliittymä palauttaa arvion lähdekoodista. Arvio on jaettu kolmeen osaan: merkkijono- ja ketjutesti sekä C++-käännöksen onnistuminen tai virheet (kuva 4.6). Mikäli merkkijono löytyy koodista, on se esitetty vihreällä. Jos jokin merkkijono puuttuu, se näytetään punaisella taustalla. Ketjutestissä esitetään kaikki testit ja kuinka monta kertaa rakenne on löytynyt opiskelijan lähdekoodista. Arvioinnin lopussa näkyy C++-kääntäjän ilmoitukset.

Esimerkissä opiskelijalla on ongelmia pinMode-funktion nimessä (kuvat 4.5 ja 4.6). Siihen on lipsahtanut yksi e-merkki liikaa, joten merkkijonotestissä *pinMode* on merkitty punaisella taustavärillä ja lopussa on siitä aiheutuva C++-kääntäjän virheilmoitus.

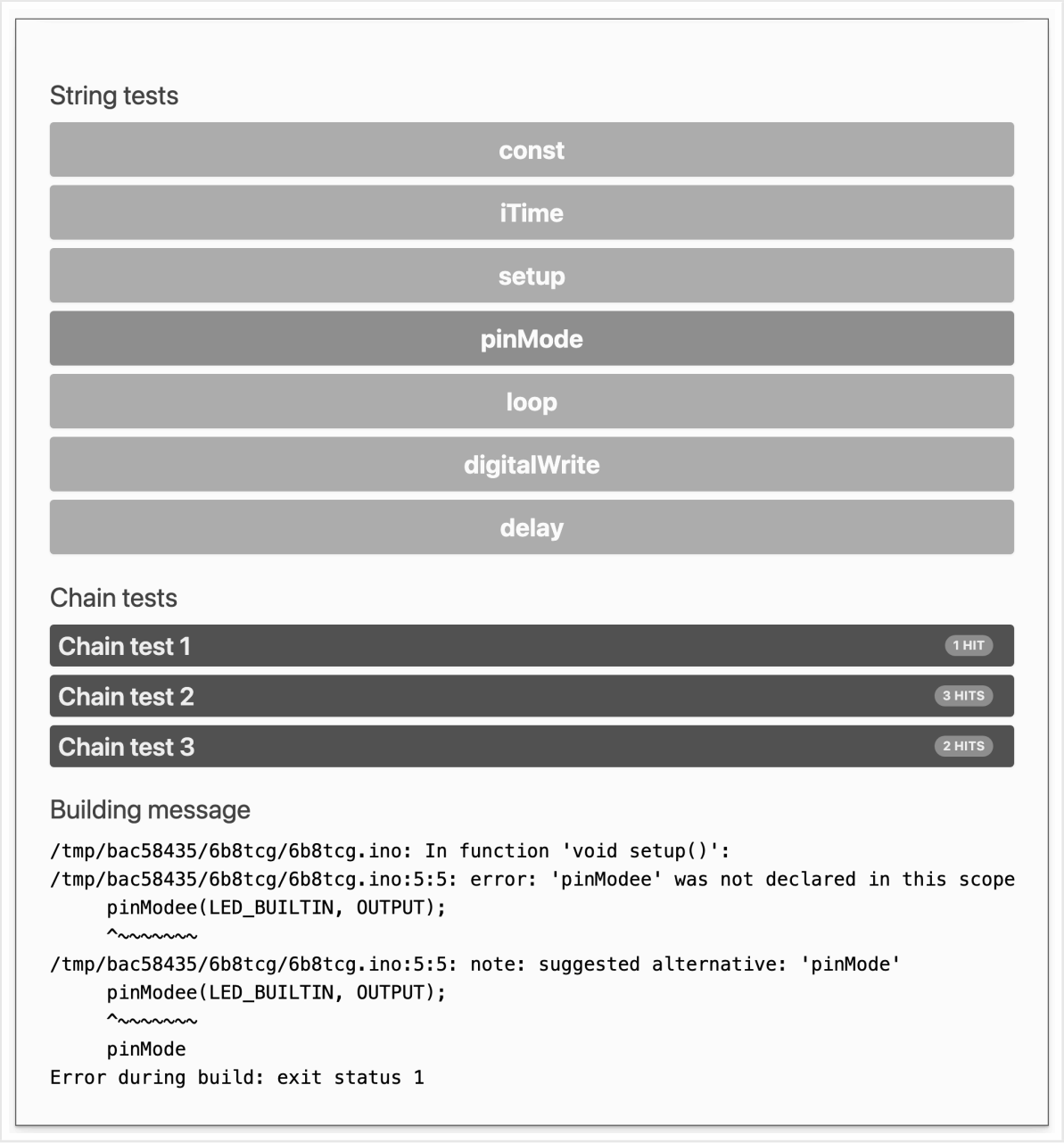
Virheilmoitus on sama, jonka Arduino IDE tuottaa virheiden kohdalla. Tässä esimerkissä se osoittaa, että rivillä 5 funktiossa setup() on funktiokutsu, jota ei ole esitelty. Lisäksi se ehdottaa ratkaisuksi pinModee()-funktion nimeämistä pinMode()-funktioksi. Opiskelijoiden omatoimista arviointia varten sovellukseen lisättiin yksinkertainen suomenkielinen käyttöohje (kuva 4.7), jossa kerrotaan, miten arviointia käytetään, millaisia tuloksia arviointisovellus palauttaa ja miten harjoitustyö palautetaan opettajalle.

Käyttöliittymä täyttää WCAG 2.1 AAA-saavutettavuustason vaatimukset kaikilta muilta osin lukuun ottamatta dynaamisesti luotavaa ACE-koodieditorin textarea-kenttää [37, W3C]. Sivun saavutettavuutta on testattu pa11y-työkalulla ja Chrome-selaimeen integroidulla Lighthouse-työkalulla, joka antoi saavutettavuudesta täydet pisteet. Käyttöliittymä on toteutettu responsiiviseksi ja se toimii oikein hyvin matkapuhelimella ja tabletilla myös jos joku haluaa käyttää arviointityökalua alemman resoluution näytöllä. Responsiivisuuden ansiosta sivua voi suurentaa WCAG-vaatimusten mukaisesti 200 % ja ylikin ilman, että käytettävyys heikkenee.

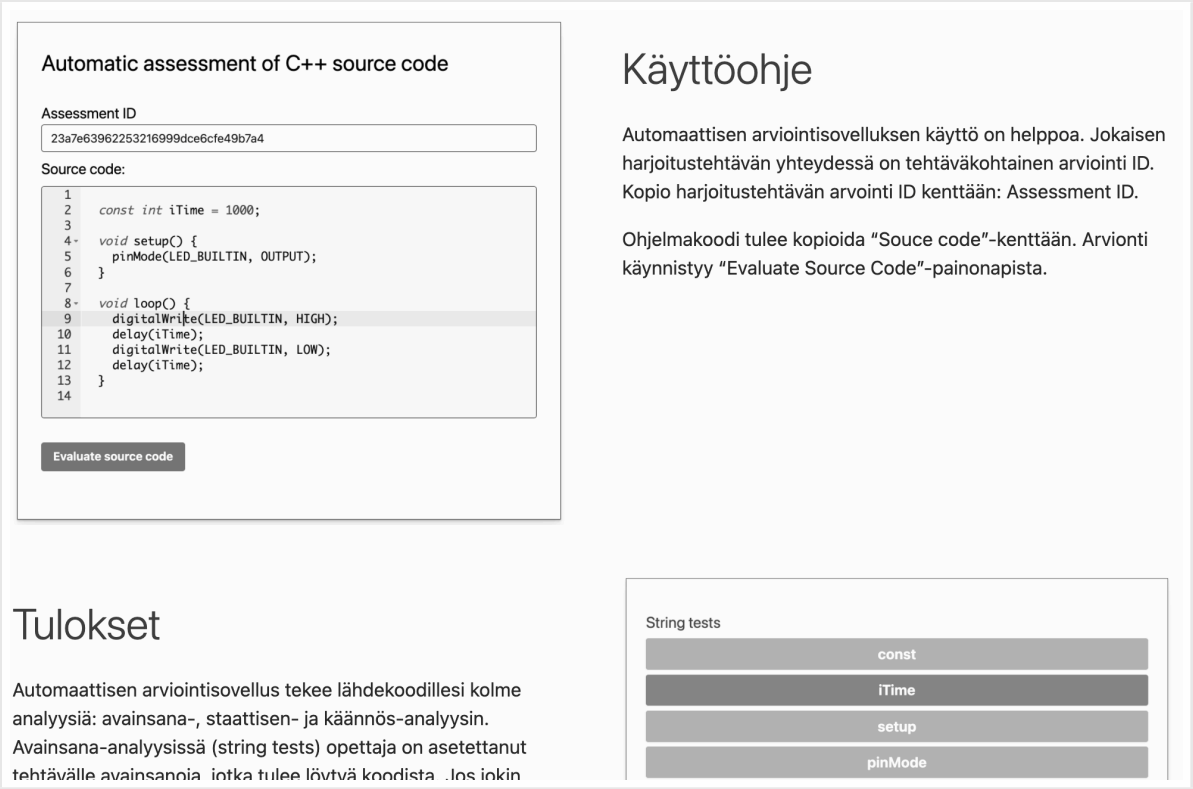
4.8 Arviointisovelluksen jatkuva julkaisu

Tässä kehitystyössä on käytetty jatkuvan sovelluksen julkaisun menetelmää. Kehitystyötä varten on rakennettu kaksi ympäristöä: paikallinen kehitysympäristö sekä testiympäristö Internetiin konseptin testausta varten.

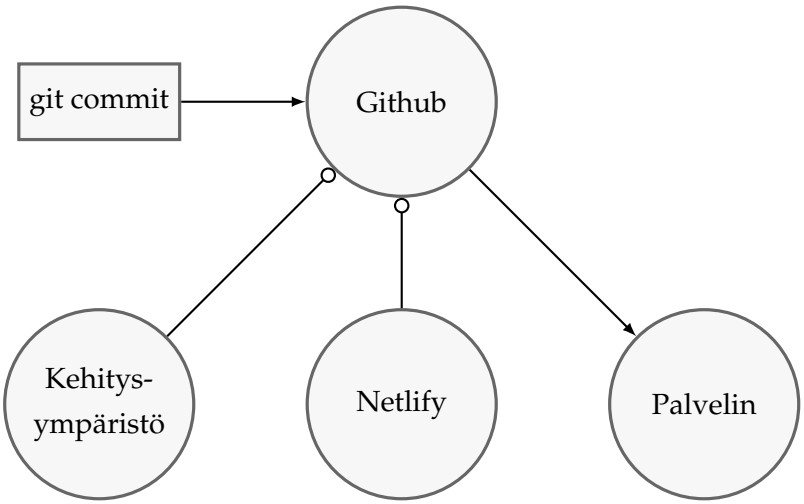
Kuvassa 4.8 on esitetty järjestelmän toimintaperiaate. Kehittäjän tehdessä muutoksen versiohallinnan development-haaraan käynnistyy kolme julkaisuprosessia. Paikallisessa lähiverkossa Jenkins havaitsee muutoksen ja kääntää UI-sovelluksen sekä uuden Docker-imagen ja julkaisee ne paikallisella palvelimella.



Kuva 4.6: Arvioinnin tulos käyttöliittymällä



Kuva 4.7: Arviointisovelluksen käyttöohje



Kuva 4.8: Arviointisovelluksen jatkuva julkaisu

Julkisen verkon puolella Netlify tarjoaa ilmaista palvelua käyttöliittymäsovelluksen automaattiseen julkaisuun. Versiohallinnan muutos käynnistää Netlifyn julkaisuprosessin, jonka tuotos julkaistaan verkkosivulla
`https://etaopiskelu.netlify.app`.

Kolmannessa julkaisuprosessissa käytetään Github:n Actions-julkaisuprosessia. Siinä palvelimella toimiva arviointisovellus käännetään Docker-konttiin, laitetaan kontti ajoon palvelimella ja tyhjennetään Nginx-sovelluksen välimuisti.

5 Tutkimuksen tulokset

Tutkimusaiheeksi valitsin automaattisen arvioinnin MOOC-ohjelmointikurssin yhteydessä. Tavoitteena oli tutkimuksellisella otteella kehittää toimiva automaattinen arviointisovelluksen konsepti ja testata sitä pienellä käyttäjäryhmällä. Luvussa 5.1 esitellään tutkimustuloksia ja luvussa 5.2 käydään läpi kehitystyön ja testauksen aikana löydettyjä jatkokehitysideoita.

Ongelma-analyysin perusteella arveltiin automaattisen arvioinnin parantavan opiskelijan tuottavuutta ja opettajan työnkuvan muutosta. Samalla odotettiin opettajan työmäärän vähenevän arvioinnin laadun kärsimättä, jolloin samalle kurssille voidaan ottaa enemmän opiskelijoita. Opiskelijoiden odotettiin muuttavan opiskelutyyliä enemmän kohti konstruktiivista oppimista, mutta samalla pelättiin iteroivan oppimisen lisääntyvän. Konstruktiivista oppimista tukee järjestelmän antama välitön palaute ja sitä tukee oppilaan itsearvio.

5.1 Tulokset

Kehitettyä arviointijärjestelmää testattiin 15.3. - 31.3.2021 vapaaehtoisilla opiskelijoilla. Käyttäjätesti suoritettiin ”TIEA3000 Johdatus sulautettuihin järjestelmiin”-kurssin kolmella ensimmäisellä tehtävällä. Tehtäviin sisältyi yhteensä neljä yksinkertaista harjoitustehtävää. Testi toteutettiin ennen varsinaisen kurssin alkua. Testausta varten kurssin opettaja hienosääti tehtäväannot automaattista arviointia varten. Tehtävänannon perusteella opettaja suunnitteli sille soveltuvan testiaineiston ja laati ohjeistuksen tehtävien palautusta varten.

Testiin ilmoittautui 7 opiskelijaa, joilla kaikilla oli aikaisempaa kokemusta ohjelmoinnista joko aikaisemmista opinnoista tai työelämästä. Seitsemästä ilmoittautuneesta neljä henkilöä teki kaikki harjoitustehtävät ja antoi palautteen 1.4. mennessä. Käyttäjätestauksessa opiskelijat käyttävät arviointityökalua harjoituksiinsa. He ottavat harjoitustehtävän valmistuttua kuvakaappauksen automaattisen arvion tuloksista ja palauttavat ne lähdekoodin kanssa opettajalle.

5.1.1 Käyttäjätestauksen palaute

Käyttäjätestauksen lopuksi kerättiin palautetta Google Forms -työkalulla (liite A). Palautteen perusteella kaikki opiskelijat pitivät automaattisen arvioinnin palautetta hyödyllisenä, mutta sen ei koettu nopeuttavan eikä hidastavan opiskelua (liite B). Palautteessa pyydettiin 14 väittämästä valitsemaan maksimissaan viisi tärkeintä ja kuvaavinta väittämää automaattisesta arviointijärjestelmästä. Kaikki neljä opiskelijaa kokivat että järjestelmä tuki harjoitustehtävän suoritusta sekä antoi siitä nopean arvion ja palautteen (taulukko 5.1).

Taulukko 5.1: Viisi tärkeintä ja kuvaavinta väitettä järjestelmästä

Antaa nopean arvion ja palautteen	4
Tukee harjoitustehtävän suoritusta	3
Antoi riittävää palautetta harjoitustyön loppuunsaattamiseksi	2
Antaa arvion ajasta ja paikasta riippumatta	1
Kannusti yritä ja erehdy menetelmällä opiskeluun	1
Nopeuttaa harjoitustehtävän suoritusta	1

Kyselyssä oli kolme avointa kysymystä, joista ensimmäisessä pyydettiin vertaamaan järjestelmän antamaa palautetta opettajan palautteeseen. Toisessa pyydettiin kertomaan järjestelmän hyödyistä ja haasteista sekä viimeisessä kysymyksessä pyydettiin jatkokehitysideoita opiskelijoilta.

Yksi käyttäjä piti opettajan antamaa palautetta tarkempana, mutta ei pitänyt merkittävänä sitä, kummalta palaute tulee, kunhan se vain on selkeää ja opiskelua tukevaa. Ongelmana muissa automaattisissa arviontijärjestelmissä hän piti sitä, että koodin tulee olla tarkkaan tietyn muotin mukainen, jotta ratkaisu menee läpi. Toinen palautteen antaja piti tärkeänä yleisellä tasolla, että arviointityökalun tulokset on avattu ja niiden käyttötarkoitus selitetty, jotta palautteesta olisi enemmän hyötyä. Yksi huomautti, ettei järjestelmä kiinnitä huomiota koodin ulkoasuun, josta opettaja saattaisi puolestaan antaa palautetta. Lisäksi harjoitustehtävien palauttamisen yhteydessä saatujen kommenttian perusteella yhdellä testaajalla oli ongelmia Google Chrome -selaimen kanssa, joka vaatii selvittelyä (liite D).

Järjestelmän hyötyinä pidettiin välitöntä palautetta sekä järjestelmän kykyä ilmoittaa puutteista tehtäväpalautuksessa sekä sen kykyä tukea itsenäistä tekemistä ja oppimista. Haasteena pidettiin yksityiskohtaista tehtävän antoa, joka rajoittaa useiden ratkaisuvaihtoehtojen lukumäärää. Lisäksi sovelluksen käytettävyyteen, toi-

mintavarmuuteen sekä validiliteettiin liittyviin ongelmiin tulisi kiinnittää huomiota.

Jatkokehitysideoita tuli muutamia. Ketjutestejä voisi kehittää edelleen siten, että ne antaisivat tarkempaa palautetta ja tehtävänannossa olisi avattu lisää ketjutestien vaatimuksia. Puuttuvien ketjutestien toivottiin käyttävän saamaa värikoodausta kuin avainsanatestissä. Avaimien toivottiin olevan selkokielisiä ja/tai sen voisi valita suoraan alasvetovalikosta. Tehtävä ID:n olisi hyvä näkyä myös tuloksissa, jolloin se näkyisi suoraan kuvakaappauksessa. Toivottiin myös vastausten lähettämistä opettajalle suoraan järjestelmästä.

5.1.2 Opettajan palaute

Myös opettaja antoi kokemuksia järjestelmän käytöstä (liite E). Saman toiminnallisuuden harjoitustehtävän ohjelmakoodissa voi toteuttaa useilla eri tavoin, mikä viittaa Pietersen [24] huomioon luovuuden tukahtumiseen. Niinpä tehtävänantoon tulee kiinnittää huomiota ja niiden tulee olla riittävän yksityiskohtaisia. Sulautetuissa järjestelmissä kytkennät voivat mennä väärin, joten siihen järjestelmästä ei ole apua. Opettaja koki, että olisi helpompaa tehdä harjoitustyöt suoraan puhtaalta pöydältä, kuin muokata olemassa olevaa harjoitustehtävää.

Opettajan työ kurssin suunnitteluvaiheessa koostuu lähinnä kahdesta työtehtävästä: harjoitustehtävän määrittämisestä arviointisovellukselle sekä tehtävänannon ja ohjeistuksen laatimisesta opiskelijoille. Näistä ensimmäinen osoittautui pienitöiseksi, kun jälkimmäiseen kului huomattavasti enemmän aikaa. Toisaalta iso työmäärä paransi myös tehtävänannon laatua. Kurssin harjoitustehtävät ovat pieniä, mutta miten järjestelmä toimii isoissa harjoitustöissä? Entä jos oppilaita on 100 tai 1000? Tarvitaan selvästi automaatiota, joka palauttaisi vain onnistuneet suoritukset opettajalle ilman kuvakaappauksia.

Miten opettaja koki tehtävien tarkistusvaiheen? Tarkistaminen oli nopeampaa, kun tarvitsi vain katsoa kuvia tuloksista. Yhden opiskelijan kohdalla jouduttiin tehtävää selvittämään myös ohjelmakoodista, kun oli epäselvyyttä automaattisessa arvioinnissa. Oletettavasti tämä ongelma ratkeaa, kun opettaja oppii uuden tavan tehdä tehtävänantoja.

Opettajalta tuli myös hyviä jatkokehitysideoita. Tuloksiin olisi hyvä saada kellon aika, jolloin samaa kuvaa on vaikea käyttää useita kertoja. Tulokset olisi hyvä saada menemään suoraan tietokantaan, tosin tämä voi heikentää järjestelmän yleiskäyttöisyyttä. Ketjutestiin olisi hyvä saada myös arvovaatimuksia, jolloin voisi vertailla

käyttääkö opiskelija esim. pinMode- tai digitalWrite-komentoa. Nythän järjestelmä tutkii, että kyseessä on sana, mutta ei sitä, mikä sana.

Käyttäjätestauksen ja palautteen perusteella järjestelmän vahvuudet viittaavat kirjallisuudessa tehtyihin havaintoihin, mikä parantaa opettajan ja opiskelijan tuottavuutta sekä tukee oppimista antaen välitöntä ja riittävän hyvää palautetta. Yleisimpiä haasteita kirjallisuuden perusteella olivat tehtävän antoon liittyvät ongelmat, huono palaute, testitapausten laadinta sekä liian jyrkkä on-off-tyyppinen arviointi. Käyttäjätestin palautteen perusteella järjestelmä ratkaisi osittain ainakin kaksi viimeisintä haastetta. Kurssiarvioinnissa voidaan huomioida myös epätäydelliset palautukset.

Testitapausten laadinta oli nopeaa, mikä tehostaa opettajan tuottavuutta, mutta vastaavasti tehtävänannon laadinta on hitaampaa. Opiskelijan saama arviointi oli riittävää palautteen perusteella, mutta toki siinä on myös paljon parannettavaa.

5.2 Kehitystyön ja testauksen aikana tulleita jatkokehitysideoita

Käyttäjätestin perusteella ketjutesti ei ole riittävän informatiivinen. Tarvitaan tietoa siitä, kuinka monta ketjutestin osumaa tulee olla, jotta tehtävä on täysin hyväksytty. Ketjutestissä voisi käyttää samaa värikoodausta kuin avainsanatestissä ja sitä voisi parantaa lisäämällä niihin arvovaatimuksia. Tällöin opettaja voisi muotoilla seuraavan harjoitustehtävän: Tee ohjelma, jossa on int-tyyppinen muuttuja ja alusta se luvulla 13. Tällöin ketjutesti voisi olla tämän näköinen:

```
["TYPE", "WORD", "ASSIGN", { "NUMBER", "13" }, "SEMICOLON"],
```

Opiskelijat ja opettaja toivoivat, että järjestelmän kautta voisi suoraan palauttaa harjoitustehtävän opettajalle tietokantaan ja palautuksen olisi hyvä saada aikaleima. Voisiko lähetyslomakkeessa olla rasti: "Mielestäni tehtävä on oikein, mutta kone arvioi sen ?". Tällöin opettaja voisi ottaa harjoitustyön manuaaliseen tarkasteluun.

Yhtenä jatkokehitysideana viitaten Tirrosen et al. [35, s. 180 – 181] tutkimukseen voisi kehittää arviointijärjestelmää formatiivisen palautteen suuntaan, jotta sillä olisi positiivinen vaikutus opiskelijan käyttäytymiseen. Esimerkiksi tällä hetkellä arviointijärjestelmä kertoo vain mitkä merkkijonotestit se läpäisi ja mitkä ei, kun se voisi antaa kannustavaa palautetta: *Neljä testiä viidestä on hyväksytty. Pinnistä vielä vähän, niin saat viimeisen testin valmiiksi!*

Arkkitehtuurin puolesta järjestelmään voisi liittää koodin tyyliä arvioivan analy-

saattorin sekä laajentaa semanttista analyysiä TDD (Test Driven Development) tyylisellä funktionaalisella testauksella. Tällöin arvioinnissa voisi käyttää VSS (Valuated State Space) -tekniikkaa (Porto et al. [25, s. 46]). Taulukossa 5.2 on esimerkki harjoitustehtävän arvioinnista monianalysaattoriympäristössä.

Taulukko 5.2: Tehtävän arviointi Valuated State Space -tekniikalla

Analysaattori	Painoarvo	Arvoalue ja arvosana
Semanttinen analyysi	4	Arvoalue 0 – 10, arvosana 7
Tyylianalyysi	2	Arvoalue 0 – 5, arvosana 2
TDD-analyysi	3	Arvoalue 0 – 5, arvosana 5
Arvio tehtävästä:	$(\frac{4}{9})(\frac{7}{10}) + (\frac{2}{9})(\frac{2}{5}) + (\frac{3}{9})(\frac{5}{5}) = 0,73 = 73$ % oikein	

Näiden lisäksi koodimetriikoita ei voi väheksyä. Hung et al. [15] tutkimuksessa esitetyt Halsteadin [11, s. 2] ja McCaben [21, s. 308] laskentametriikat olisivat suhteellisen pienellä työmäärällä liitettävissä arviointisovellukseen. Halsteadin metriikoista ohjelman pituus [11, s. 9], ohjelman volyymi [11, s. 19] ja haastavuus [11, s. 2] voitaisiin liittää nykyiseen toteutukseen noin viikon työpanoksella ja McCaben monimutkaisuusteorian laskentametriikat 1 – 2 viikon työpanoksella. Metriikoita voitaisiin käyttää arvioinnissa arvosanan annossa, keskeneräisen harjoitustehtävän kattavuuden arvioinnissa sekä tehtävän vaikeusasteen määrittelyssä.

Järjestelmällä olisi helppo kerätä tietoa oppimisanalytiikan avulla. Esimerkiksi käyttäjätestissä oli neljä testaajaa ja jokainen teki neljä tehtävää. Lokitiedon perusteella tehtäviä oli lähetetty arvioitavaksi kaikkiaan 83 kertaa, joista testaaja numero kaksi oli lähettänyt tehtävän 3 arvioitavaksi kaikkiaan 34 kertaa (liite C). Tirronen et al. [35] mukaan tällainen käyttäytyminen on tulkittavissa yritä- ja erehdy käyttäytymiseksi, mikä voi johtua oppimateriaalin tai tehtävänannon ongelmista.

Voisiko tätä staattista analyysiä käyttää muuhunkin kuin lähdekoodin arviointiin? Menetelmän ideana on purkaa formaali koodi semanttiseen muotoon ja tunnistaa siitä rakenteita, jotka ovat yhteisiä kaikille ratkaisuvaihtoehdoille. Näkisin, että samalla menetelmällä voisi ratkaista mitä tahansa formaaleita esitysmuotoja, joissa ollaan kiinnostuneista siitä, miten ratkaisuun päästiin, eikä vain lopputuloksesta.

Yksi jatkokehityskohde menetelmälle voisi olla järjestelmän kehitys matematiikan ja fysiikan tehtävien automaattiseen arviointiin. Esimerkiksi matematiikan tehtävä voisi olla: Ratkaise yhtälö $x + 1 = 5$. Esitä vastauksessa ratkaisun kaikki vaiheet.

Tällöin oppilaan kirjoittama vastaus tietokoneen näyttöpäätteellä voisi olla:

```
x + 1 = 5;
x + 1 − 1 = 5 − 1;
x = 4;
```

Jo ihan tällaisenaan arviointisovelluksen analysaattori tekee aika hyvän testirungon opettajalle ilman muutoksia sovelluksen lähdekoodiin:

```
"chainTest": [
  [ "WORD", "ADDITION", "NUMBER", "ASSIGN", "NUMBER", "SEMICOLON" ],
  [ "WORD", "ADDITION", "NUMBER", "SUBTRACTION", "NUMBER", "ASSIGN",
    "NUMBER", "SUBTRACTION", "NUMBER", "SEMICOLON" ],
  [ "WORD", "ASSIGN", "NUMBER", "SEMICOLON" ]
]
```

Käytännössä nykyistä toteutusta ei tarvitsisi muuttaa kuin laittamalla puolipisteen tilalle enter-näppäimen painallus, jolloin opiskelija voisi syöttää yksinkertaisen harjoitustehtävän suoraan HTML-lomakkeeseen ratkaisuvaiheineen.

6 Yhteenveto

Tämän tutkimuksen tavoitteena oli löytää ja toteuttaa konsepti C++-koodin automaattiseen arviointiin. Kirjallisuuden perusteella arviointi on yksi suurimmista haasteista MOOC-kursseilla, jossa yhdelle kurssille voi osallistua suuri määrä opiskelijoita eri lähtökohdista. Miten voidaan tarjota tasalaatuista, yksilöllistä ja formaalia arviointia ohjelmointikurssille? Tutkimuksen ajurina toimi kysymys: ”Millaisella konseptilla voidaan toteuttaa C++-lähdekoodin automaattinen arviointi?”

Koska tarkoituksena oli myös toteuttaa konsepti, niin tarvittiin joustava tutkimusmenetelmä, jossa on tutkimuksellinen ote ja ketterä sovelluskehitysmalli. Tutkimusmenetelmäksi valikoitui kehittämistutkimuksen ja ketterän Scrum-mallin yhdistelmä, jossa ongelma-analyysissä määritettiin tutkimuksellisia haasteita ja tarpeita. Scrum-sovelluskehitysmallia käytettiin arviointisovelluksen kehitystyössä ja malliin liitettiin myös parhaimpia ohjelmistokehityksen toimintamalleja, kuten sovelluksen jatkuva julkaisu sekä automaattinen E2E-testaus.

Tutkimuksen tuloksena saatiin semanttiseen- ja avainsana-analyysiin perustuva verkkopohjainen automaattinen arviointityökalu, joka täytti kaikki sille asetetut vaatimukset: a) automaattinen lähdekoodiin staattinen analysointi, b) skaalautuva sovellus- ja teknologia-arkkitehtuuri c) turvallinen lähdekoodin kääntäminen palvelimella sekä d) helppo tapa tuottaa automaattisia testiaihioita.

Tutkimukseen kuului myös käyttäjätestaus, jossa automaattista arviointisovellusta testattiin pienellä 4 henkilön opiskelijaryhmällä ja kerättiin palautetta Google Forms -lomakkeella. Testauksen perusteella konsepti antoi samanlaisia tuloksia kuin aikaisemmat tutkimukset, mutta onnistuimme ratkaisemaan muutamia arvioinnin laatuun ja tuottavuuteen liittyviä ongelmia.

Tutkimuksen pohjalta kehitettyä konseptia voi sellaisenaan käyttää Arduino-pohjaisten sulautettujen järjestelmien ohjelmointikurssien automaattisena arviointityökaluna ja kohtuullisella lisätyöllä se soveltuu minkä tahansa ohjelmointikurssin työkaluksi. Tämä kehittämistutkimuksen tulokset toimivat hyvänä pohjana jatkotutkimuksille ja antaa lisää tietoa staattista analyysiä soveltavalle tutkimuksille.

Lähteet

[1] ALA-MUTKA, K., UIMONEN, T., JA JARVINEN, H.-M. Supporting students in c++ programming courses with automatic program style assessment. *Journal of v: Research 3*, 1 (2004), 245–262. JUFO1.

[2] ALA-MUTKA, K. M. A survey of automated assessment approaches for programming assignments. *Computer science education 15*, 2 (2005), 83–102. JUFO2.

[3] ARACHCHI, S., JA PERERA, I. Continuous integration and continuous delivery pipeline automation for agile software project management. Julkaisusarjassa *2018 Moratuwa Engineering Research Conference (MERCon)* (2018), IEEE, 156–161.

[4] BATURAY, M. H. An overview of the world of moocs. *Procedia-Social and Behavioral Sciences 174* (2015), 427–433. JUFO1.

[5] BEY, A., JERMANN, P., JA DILLENBOURG, P. A comparison between two automatic assessment approaches for programming: An empirical study on moocs. *Journal of Educational Technology & Society 21*, 2 (2018), 259–272. JUFO1.

[6] CERIOLI, M., JA CINELLI, P. Grasp: Grading and rating assistant professor. *ACM-IFIP* (2008).

[7] CHEN, H.-M., CHEN, W.-H., JA LEE, C.-C. An automated assessment system for analysis of coding convention violations in java programming assignments. *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34*, 5 (2018), 1203–1221. JUFO1.

[8] CHEN, X., FRANCIA, B., LI, M., MCKINNON, B., JA SEKER, A. Shared information and program plagiarism detection. *IEEE Transactions on Information Theory 50*, 7 (2004), 1545–1551. JUFO3.

[9] EDELSON, D. C. Design research: What we learn when we engage in design. *The Journal of the Learning sciences 11*, 1 (2002), 105–121. JUFO3.

[10] GERDES, A., HEEREN, B., JEURING, J., JA VAN BINSBERGEN, L. T. Ask-elle: an adaptable programming tutor for haskell giving automated feedback. *International Journal of Artificial Intelligence in Education 27*, 1 (2017), 65–100. JUFO1.

[11] HALSTEAD, M. H. *Elements of software science*. Elsevier Scientific Publishing Company, 1977. ISBN: 0-444-00205-7, JUFO2.

[12] HAMEER, A., JA PIENTKA, B. Teaching the art of functional programming using automated grading (experience report). *Proceedings of the ACM on Programming Languages 3*, ICFP (2019), 1–15. JUFO1.

[13] HOLLINGSWORTH, J. Automatic graders for programming classes. *Communications of the ACM 3*, 10 (1960), 528–529. JUFO3.

[14] HOLUB, A. I. *Compiler design in C*. Prentice Hall Englewood Cliffs, NJ, 1990. ISBN: 0-13-155045-4.

[15] HUNG, S.-L., KWOK, I.-F., JA CHAN, R. Automatic programming assessment. *Computers & Education 20*, 2 (1993), 183–190. JUFO3.

[16] IHANTOLA, P., AHONIEMI, T., KARAVIRTA, V., JA SEPPÄLÄ, O. Review of recent systems for automatic assessment of programming assignments. Julkaisusarjassa *Proceedings of the 10th Koli calling international conference on computing education research* (2010), 86–93. JUFO1.

[17] INSA, D., JA SILVA, J. Automatic assessment of java code. *Computer Languages, Systems & Structures 53* (2018), 59–72. JUFO1.

[18] JACKSON, D., JA USHER, M. Grading student programs using assyst. Julkaisusarjassa *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education* (1997), 335–339. JUFO2.

[19] KAY, D. G., SCOTT, T., ISAACSON, P., JA REEK, K. A. Automated grading assistance for student programs. *ACM SIGCSE Bulletin 26*, 1 (1994), 381–382. JUFO2.

[20] MCBROOM, J., KOPRINSKA, I., JA YACEF, K. A survey of automated programming hint generation–the hints framework. *arXiv preprint arXiv:1908.11566* (2019). JUFO0.

[21] McCabe, T. J. A complexity measure. *IEEE Transactions on software Engineering*, 4 (1976), 308–320. JUFO3.

[22] Newman, S. *Building microservices: designing fine-grained systems*. "O'Reilly Media, Inc.", 2015.

[23] Pernaa, J. Kehittämistutkimus tutkimusmenetelmänä. *Teoksessa Kehittämistutkimus* (2013).

[24] Pieterse, V. Automated assessment of programming assignments. *Computer Science Education Research Conference 13* (2013), 4–5.

[25] Porto, V., ja Fogel, L. *Evolving Command, Control, Communications, Computers, and Intelligence, Surveillance and Reconnaissance (C4ISR) Decisions*. Tekninen raportti, NATURAL SELECTION INC LA JOLLA CA, 2002.

[26] Rees, M. J. Automatic assessment aids for pascal programs. *ACM Sigplan Notices 17*, 10 (1982), 33–42. JUFO0.

[27] Rubio-Sánchez, M., Kinnunen, P., Pareja-Flores, C., ja Velázquez-Iturbide, Á. Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior 31* (2014), 453–460. JUFO2.

[28] Saikkonen, R., Malmi, L., ja Korhonen, A. Fully automatic assessment of programming exercises. Julkaisusarjassa *Proceedings of the 6th annual conference on Innovation and technology in computer science education* (2001), 133–136. JUFO2.

[29] Schwaber, K. Scrum development process. Kirjassa *Business object design and implementation*. Springer, 1997, ss. 117–134. JUFO2.

[30] Schwaber, K., ja Beedle, M. *Agile software development with Scrum*, vol. 1. Prentice Hall Upper Saddle River, 2002. JUFO1.

[31] Silva, M. T., de Barros Costa, E., de Sousa Miranda, M., ja Silva, E. T. A tree inclusion analyzer for examining introductory programming codes. Julkaisusarjassa *2015 IEEE Frontiers in Education Conference (FIE)* (2015), IEEE, 1–7. JUFO1.

[32] Suleman, H. Automatic marking with sakai. Julkaisusarjassa *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists*

and Information Technologists on IT research in developing countries: riding the wave of technology (2008), 229–236.

[33] SUTHERLAND, J. Business object design and implementation workshop. Julkaisusarjassa *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum)* (1995), Springer, 170–175. JUFO2.

[34] THORBURN, G., JA ROWE, G. Pass: An automated system for program assessment. *Computers & Education 29*, 4 (1997), 195–206. JUFO3.

[35] TIRRONEN, V., JA TIRRONEN, M. A framework for evaluating student interaction with automatically assessed exercises. Julkaisusarjassa *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (2016), 180–181. JUFO1.

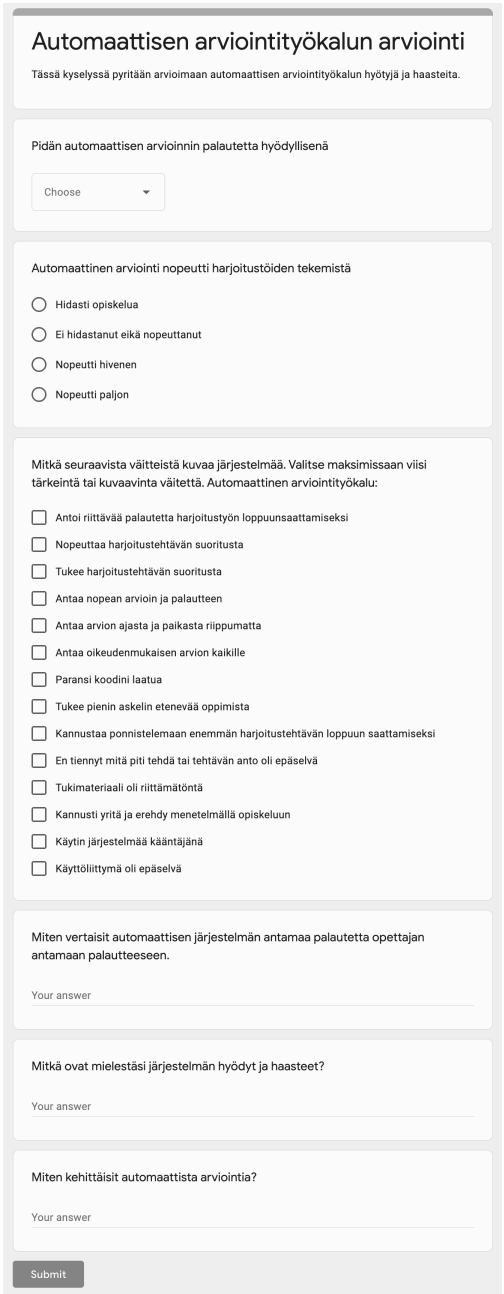
[36] TRUONG, N., ROE, P., JA BANCROFT, P. Static analysis of students' java programs. Julkaisusarjassa *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30* (2004), Citeseer, 317–325. JUFO1.

[37] WORLD WIDE WEB CONSORTIUM, ET AL. Web content accessibility guidelines (WCAG) 2.1, June 2018. URL: https://www.w3.org/TR/WCAG21/.

[38] XIE, Y., JA ENGLER, D. Using redundancies to find errors. Julkaisusarjassa *Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering* (2002), 51–60. JUFO2.

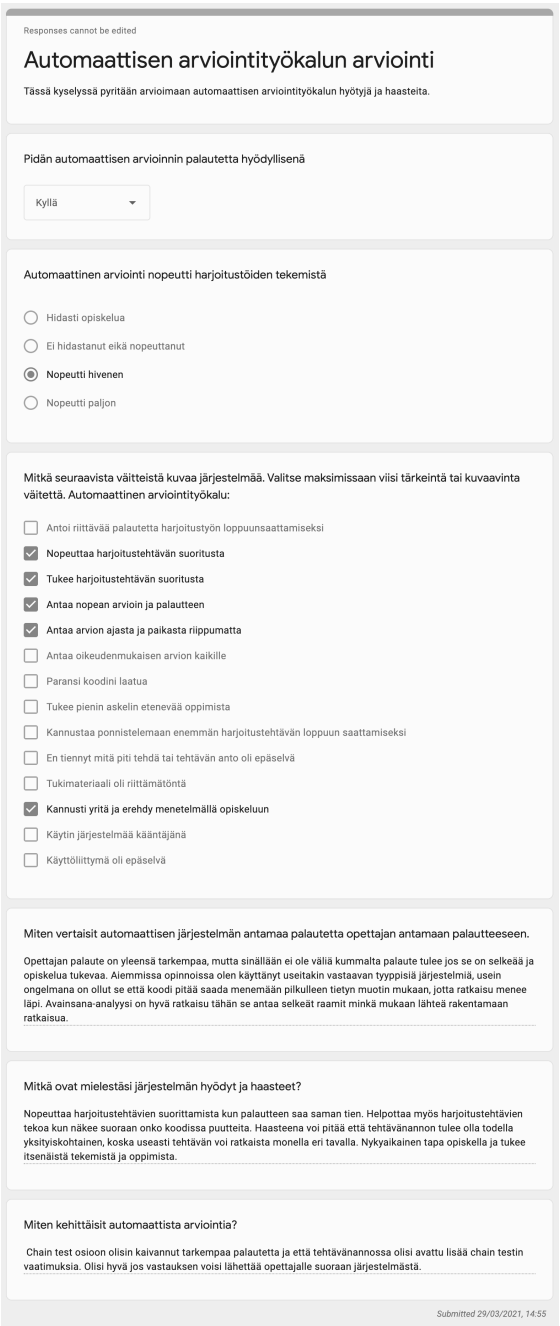
[39] XIONG, Y., JA SUEN, H. K. Assessment approaches in massive open online courses: Possibilities, challenges and future directions. *International Review of Education 64*, 2 (2018), 241–263. JUFO1.

A Kyselytutkimuksen sähköinen kyselylomake



Kuva A.1: Sähköinen kyselylomake

B Kyselytutkimuksen vastaukset



Responses cannot be edited

Automaattisen arviointityökalun arviointi

Tässä kyselyssä pyritään arvioimaan automaattisen arviointityökalun hyötyjä ja haasteita.

Pidän automaattisen arvioinnin palautetta hyödyllisenä

Kyllä

Automaattinen arviointi nopeutti harjoitustöiden tekemistä

- ○ Hidasti opiskelua
- ○ Ei hidastanut eikä nopeuttanut
- ◉ Nopeutti hivenen
- ○ Nopeutti paljon

Mitkä seuraavista väitteistä kuvaa järjestelmää. Valitse maksimissaan viisi tärkeintä tai kuvaavinta väitettä. Automaattinen arviointityökalu:

- ☐ Antoi riittävää palautetta harjoitustyön loppuunsaattamiseksi
- ☑ Nopeuttaa harjoitustehtävän suoritusta
- ☑ Tukee harjoitustehtävän suoritusta
- ☑ Antaa nopean arvioin ja palautteen
- ☑ Antaa arvion ajasta ja paikasta riippumatta
- ☐ Antaa oikeudenmukaisen arvion kaikille
- ☐ Paransi koodini laatua
- ☐ Tukee pienin askelin etenevää oppimista
- ☐ Kannustaa ponnistelemaan enemmän harjoitustehtävän loppuun saattamiseksi
- ☐ En tiennyt mitä piti tehdä tai tehtävän anto oli epäselvä
- ☐ Tukimateriaali oli riittämätöntä
- ☑ Kannusti yritä ja erehdy menetelmällä opiskeluun
- ☐ Käytin järjestelmää kääntäjänä
- ☐ Käyttöliittymä oli epäselvä

Miten vertaisit automaattisen järjestelmän antamaa palautetta opettajan antamaan palautteeseen.

Opettajan palaute on yleensä tarkempaa, mutta sinällään ei ole väliä kummalta palaute tulee jos se on selkeää ja opiskelua tukevaa. Aiemmissa opinnoissa olen käyttänyt useitakin vastaavan tyyppisiä järjestelmiä, usein ongelmana on ollut se että koodi pitää saada menemään pilkulleen tietyn muotin mukaan, jotta ratkaisu menee läpi. Avainsana-analyysi on hyvä ratkaisu tähän se antaa selkeät raamit minkä mukaan lähteä rakentamaan ratkaisua.

Mitkä ovat mielestäsi järjestelmän hyödyt ja haasteet?

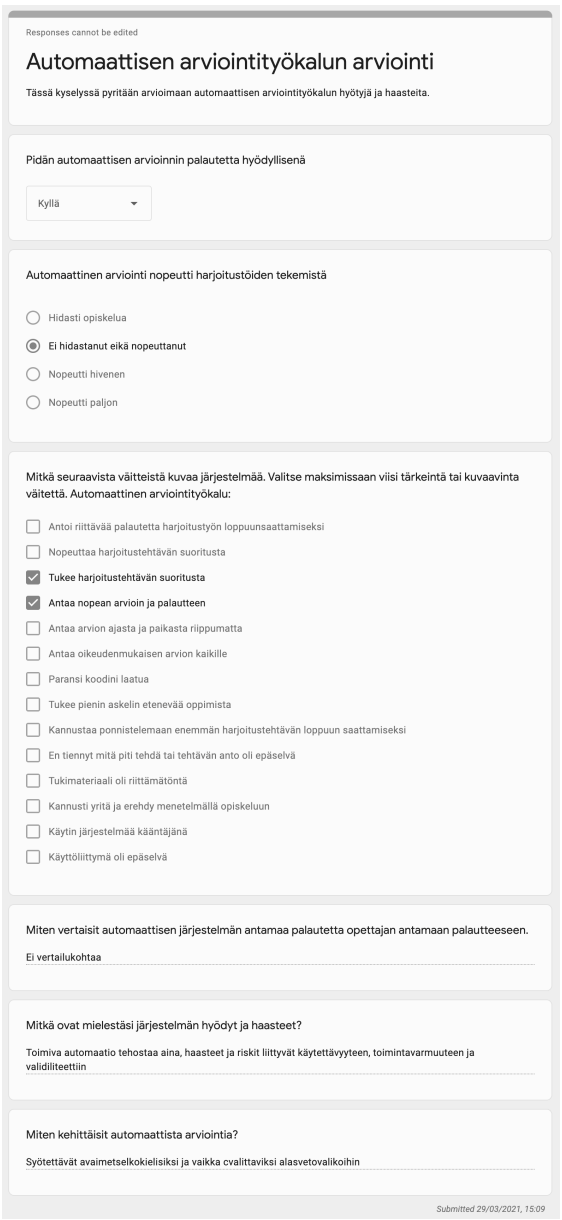
Nopeuttaa harjoitustehtävien suorittamista kun palautteen saa saman tien. Helpottaa myös harjoitustehtävien tekoa kun näkee suoraan onko koodissa puutteita. Haasteena voi pitää että tehtävänannon tulee olla todella yksityiskohtainen, koska useasti tehtävän voi ratkaista monella eri tavalla. Nykyaikainen tapa opiskella ja tukee itsenäistä tekemistä ja oppimista.

Miten kehittäisit automaattista arviointia?

Chain test osioon olisin kaivannut tarkempaa palautetta ja että tehtävänannossa olisi avattu lisää chain testin vaatimuksia. Olisi hyvä jos vastauksen voisi lähettää opettajalle suoraan järjestelmästä.

Submitted 29/03/2021, 14:55

Kuva B.1: Vastaus 1



Responses cannot be edited

Automaattisen arviointityökalun arviointi

Tässä kyselyssä pyritään arvioimaan automaattisen arviointityökalun hyötyjä ja haasteita.

Pidän automaattisen arvioinnin palautetta hyödyllisenä

Kyllä

Automaattinen arviointi nopeutti harjoitustöiden tekemistä

- ○ Hidasti opiskelua
- ◉ Ei hidastanut eikä nopeuttanut
- ○ Nopeutti hivenen
- ○ Nopeutti paljon

Mitkä seuraavista väitteistä kuvaa järjestelmää. Valitse maksimissaan viisi tärkeintä tai kuvaavinta väitettä. Automaattinen arviointityökalu:

- ☐ Antoi riittävää palautetta harjoitustyön loppuunsaattamiseksi
- ☐ Nopeuttaa harjoitustehtävän suoritusta
- ☑ Tukee harjoitustehtävän suoritusta
- ☑ Antaa nopean arvioin ja palautteen
- ☐ Antaa arvion ajasta ja paikasta riippumatta
- ☐ Antaa oikeudenmukaisen arvion kaikille
- ☐ Paransi koodini laatua
- ☐ Tukee pienin askelin etenevää oppimista
- ☐ Kannustaa ponnistelemaan enemmän harjoitustehtävän loppuun saattamiseksi
- ☐ En tiennyt mitä piti tehdä tai tehtävän anto oli epäselvä
- ☐ Tukimateriaali oli riittämätöntä
- ☐ Kannusti yritä ja erehdy menetelmällä opiskeluun
- ☐ Käytin järjestelmää kääntäjänä
- ☐ Käyttöliittymä oli epäselvä

Miten vertaisit automaattisen järjestelmän antamaa palautetta opettajan antamaan palautteeseen.

Ei vertailukohtaa

Mitkä ovat mielestäsi järjestelmän hyödyt ja haasteet?

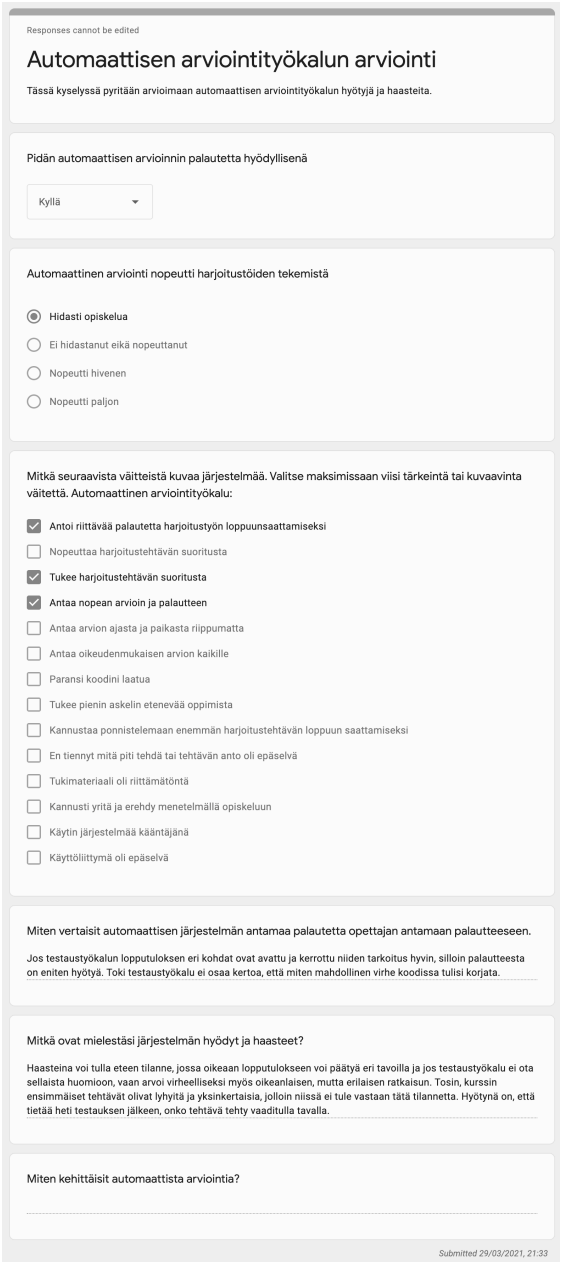
Toimiva automaatio tehostaa aina, haasteet ja riskit liittyvät käytettävyyteen, toimintavarmuuteen ja validiliteettiin

Miten kehittäisit automaattista arviointia?

Syötettävät avaimetselkokielisiksi ja vaikka cvalittaviksi alasvetovalikoihin

Submitted 29/03/2021, 15:09

Kuva B.2: Vastaus 2



Responses cannot be edited

Automaattisen arviointityökalun arviointi

Tässä kyselyssä pyritään arvioimaan automaattisen arviointityökalun hyötyjä ja haasteita.

Pidän automaattisen arvioinnin palautetta hyödyllisenä

Kyllä

Automaattinen arviointi nopeutti harjoitustöiden tekemistä

- (•) Hidasti opiskelua
- ( ) Ei hidastanut eikä nopeuttanut
- ( ) Nopeutti hivenen
- ( ) Nopeutti paljon

Mitkä seuraavista väitteistä kuvaa järjestelmää. Valitse maksimissaan viisi tärkeintä tai kuvaavinta väitettä. Automaattinen arviointityökalu:

- [x] Antoi riittävää palautetta harjoitustyön loppuunsaattamiseksi
- [ ] Nopeuttaa harjoitustehtävän suoritusta
- [x] Tukee harjoitustehtävän suoritusta
- [x] Antaa nopean arvioin ja palautteen
- [ ] Antaa arvion ajasta ja paikasta riippumatta
- [ ] Antaa oikeudenmukaisen arvion kaikille
- [ ] Paransi koodini laatua
- [ ] Tukee pienin askelin etenevää oppimista
- [ ] Kannustaa ponnistelemaan enemmän harjoitustehtävän loppuun saattamiseksi
- [ ] En tiennyt mitä piti tehdä tai tehtävän anto oli epäselvä
- [ ] Tukimateriaali oli riittämätöntä
- [ ] Kannusti yritä ja erehdy menetelmällä opiskeluun
- [ ] Käytin järjestelmää kääntäjänä
- [ ] Käyttöliittymä oli epäselvä

Miten vertaisit automaattisen järjestelmän antamaa palautetta opettajan antamaan palautteeseen.

Jos testaustyökalun lopputuloksen eri kohdat ovat avattu ja kerrottu niiden tarkoitus hyvin, silloin palautteesta on eniten hyötyä. Toki testaustyökalu ei osaa kertoa, että miten mahdollinen virhe koodissa tulisi korjata.

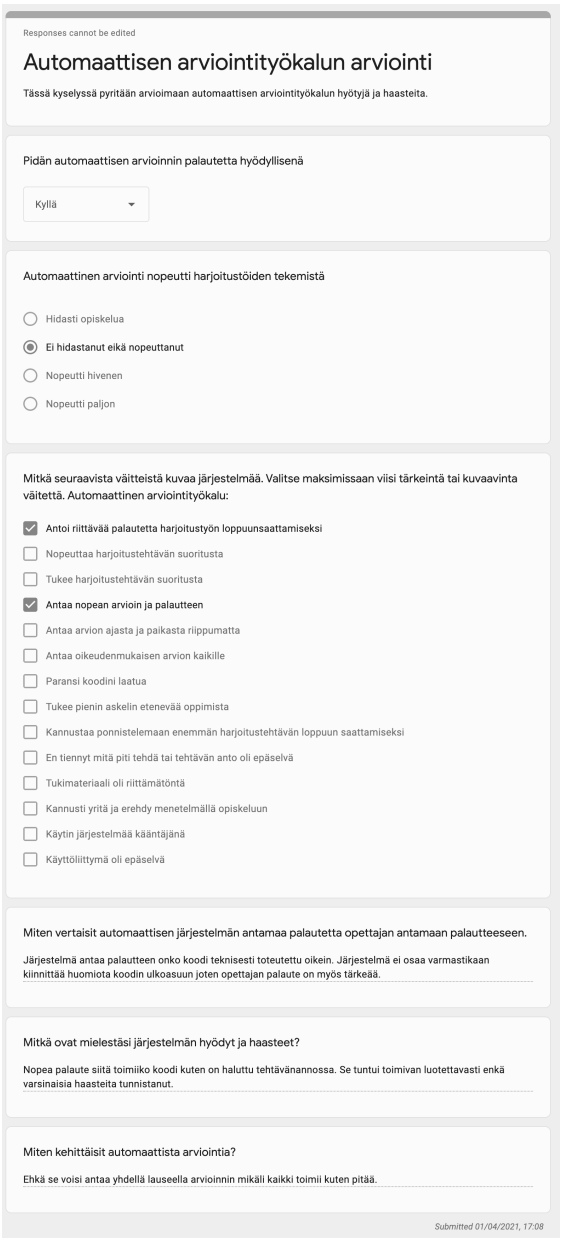
Mitkä ovat mielestäsi järjestelmän hyödyt ja haasteet?

Haasteina voi tulla eteen tilanne, jossa oikeaan lopputulokseen voi päätyä eri tavoilla ja jos testaustyökalu ei ota sellaista huomioon, vaan arvoi virheelliseksi myös oikeanlaisen, mutta erilaisen ratkaisun. Tosin, kurssin ensimmäiset tehtävät olivat lyhyitä ja yksinkertaisia, jolloin niissä ei tule vastaan tätä tilannetta. Hyötynä on, että tietää heti testauksen jälkeen, onko tehtävä tehty vaaditulla tavalla.

Miten kehittäisit automaattista arviointia?

Submitted 29/03/2021, 21:33

Kuva B.3: Vastaus 3



Responses cannot be edited

Automaattisen arviointityökalun arviointi

Tässä kyselyssä pyritään arvioimaan automaattisen arviointityökalun hyötyjä ja haasteita.

Pidän automaattisen arvioinnin palautetta hyödyllisenä

Kyllä

Automaattinen arviointi nopeutti harjoitustöiden tekemistä

- ○ Hidasti opiskelua
- ◉ Ei hidastanut eikä nopeuttanut
- ○ Nopeutti hivenen
- ○ Nopeutti paljon

Mitkä seuraavista väitteistä kuvaa järjestelmää. Valitse maksimissaan viisi tärkeintä tai kuvaavinta väitettä. Automaattinen arviointityökalu:

- ☑ Antoi riittävää palautetta harjoitustyön loppuunsaattamiseksi
- ☐ Nopeuttaa harjoitustehtävän suoritusta
- ☐ Tukee harjoitustehtävän suoritusta
- ☑ Antaa nopean arvioin ja palautteen
- ☐ Antaa arvion ajasta ja paikasta riippumatta
- ☐ Antaa oikeudenmukaisen arvion kaikille
- ☐ Paransi koodini laatua
- ☐ Tukee pienin askelin etenevää oppimista
- ☐ Kannustaa ponnistelemaan enemmän harjoitustehtävän loppuun saattamiseksi
- ☐ En tiennyt mitä piti tehdä tai tehtävän anto oli epäselvä
- ☐ Tukimateriaali oli riittämätöntä
- ☐ Kannusti yritä ja erehdy menetelmällä opiskeluun
- ☐ Käytin järjestelmää kääntäjänä
- ☐ Käyttöliittymä oli epäselvä

Miten vertaisit automaattisen järjestelmän antamaa palautetta opettajan antamaan palautteeseen.

Järjestelmä antaa palautteen onko koodi teknisesti toteutettu oikein. Järjestelmä ei osaa varmastikaan kiinnittää huomiota koodin ulkoasuun joten opettajan palaute on myös tärkeää.

Mitkä ovat mielestäsi järjestelmän hyödyt ja haasteet?

Nopea palaute siitä toimiiko koodi kuten on haluttu tehtävänannossa. Se tuntui toimivan luotettavasti enkä varsinaisia haasteita tunnistanut.

Miten kehittäisit automaattista arviointia?

Ehkä se voisi antaa yhdellä lauseella arvioinnin mikäli kaikki toimii kuten pitää.

Submitted 01/04/2021, 17:08

Kuva B.4: Vastaus 4

C Anonymisoitu lokitieto palvelimelta

```
111.111.111.111:[23/Mar/2021:09:45:26+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
111.111.111.111:[23/Mar/2021:09:45:34+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
111.111.111.111:[23/Mar/2021:09:46:08+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
111.111.111.111:[23/Mar/2021:09:46:49+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
111.111.111.111:[23/Mar/2021:10:15:23+0100]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
111.111.111.111:[23/Mar/2021:10:16:58+0100]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
111.111.111.111:[23/Mar/2021:11:53:59+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
111.111.111.111:[23/Mar/2021:11:55:00+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
111.111.111.111:[23/Mar/2021:11:55:42+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
111.111.111.111:[23/Mar/2021:12:40:25+0100]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
222.222.222.222:[19/Mar/2021:11:12:45+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
222.222.222.222:[19/Mar/2021:11:24:56+0100]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
222.222.222.222:[19/Mar/2021:11:30:19+0100]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
222.222.222.222:[19/Mar/2021:11:39:18+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
222.222.222.222:[19/Mar/2021:14:27:32+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:14:31:36+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:14:31:36+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:14:31:38+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:14:31:42+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:15:25:53+0100]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
... 32 kpl
222.222.222.222:[19/Mar/2021:17:17:27+0100]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
222.222.222.222:[19/Mar/2021:17:43:54+0100]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
222.222.222.222:[19/Mar/2021:17:51:37+0100]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
222.222.222.222:[19/Mar/2021:17:54:39+0100]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
222.222.222.222:[19/Mar/2021:17:58:26+0100]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
333.333.333.333:[29/Mar/2021:17:47:29+0200]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
333.333.333.333:[29/Mar/2021:18:00:14+0200]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
444.444.444.444:[29/Mar/2021:18:54:32+0200]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
... 5 kpl
444.444.444.444:[29/Mar/2021:18:55:20+0200]:/etaopiskelu/5f9a721e0b0d5a8c81a0c6f5a5b13591
444.444.444.444:[29/Mar/2021:19:02:31+0200]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
... 4 kpl
444.444.444.444:[29/Mar/2021:19:04:09+0200]:/etaopiskelu/73460f64ee54d14b0daf15016f77da22
333.333.333.333:[29/Mar/2021:19:04:40+0200]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
444.444.444.444:[29/Mar/2021:19:12:23+0200]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
333.333.333.333:[29/Mar/2021:19:27:20+0200]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
444.444.444.444:[29/Mar/2021:19:41:07+0200]:/etaopiskelu/63ea2d74dc1074b64de2bb937f430907
444.444.444.444:[29/Mar/2021:19:51:45+0200]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
... 4 kpl
444.444.444.444:[29/Mar/2021:20:31:23+0200]:/etaopiskelu/53bdbddf6d12c1e22cdefb9939c4473a
```

D Harjoitustehtävien palautusten yhteydessä saadut kommentit oppilailta

1. Harjoitus 3 Chain test 2 ei näkynyt tarkistustyökalussa, eli odotettua rakennetta ei näkynyt koodissa. Olisi hyödyllistä jos puuttuva Chain test näkyisi punaisena kuten puuttuvat avainsana-analyysit. Jonkinlaista palautetta tarkistustyökalusta voisi tällaisesta puuttuvasta osuudesta myös tulla, se helpottaisi koodin korjausta.

2. Tarkistustyökalu toimi mielestäni hyvin, eikä minulla ollut mitään ongelmia sen kanssa. Screenshotin ottaminen oli hieman vaikeaa kannettavan tietokoneen ruudulta. Olisi ehkä hyvä että assesment id näkyisi myös alla jossa tarkistukset ilmoitetaan.

3. Tarkistustyökalu ei toiminut minulla Chrome selaimella ollenkaan jostain syystä. Firefoxilla toimi oikein. Käyttöjärjestelmänä Windows 7 professional.

E Opettajan kommentit

Kokemuksia tekemisvaiheesta:

1. Opiskelija voi tehdä työn monella tapaa niin, että se täyttää tehtävänannon. Esimerkiksi LED-pinnit voi määritellä alussa muuttujina tai sitten pinnin numeron voi suoraan kirjoittaa koodiin. Pinnin numeron voi alustaa int-tyyppisenä tai const byte:nä tai #define määreellä jne.

2. Näitä pitää siis ohjeistaa tarkemmin, esim. "Alusta muuttujat alussa"tai "käytä do-while silmukkaa",

3. Vaatii aika yksityiskohtaiset tehtävän ohjeet. Jos ohjeet tekisi yleisemmällä tasolla, niin riski vääriin ratkaisuihin (sellaisiin, jotka toimisivat kääntäjässä, mutta eivät täyttäisi tarkastustyökalun muita tarkastuskohteita) kasvaisi. Toisaalta harkkatyöstä voi jättää jotain tarkistamatta, ja silloin siltä osin voi jättää myös ohjeistamatta. Eli kaikkea mitä pystyy, ei tarvitse tarkistaa kaikista töistä.

4. Ei kerro kytkennöistä mitään. Eli ei kerro esim. onko LED oikein päin tmv.

5. Olisi ehkä helpompi, jos harkkatyön tekisi puhtaalta pöydältä ja voisi lähtökohtaisesti suunnitella tehtävät sellaisiksi, että ne voi tällä tarkistaa.

6. Opettajalla kahdenlaista tekemistä. Teknistä tarkistustyökalulla tehtävää (tarkistuskohteiden määrittämistä työkaluun yms.) ja sitten toisaalta tehtävän ohjeistuksen muokkaamista.

7. Ensimmäinen on melko pieni homma loppujen lopuksi. Tosin vaatii ainakin ensimmäisillä kerroilla paljon tarkastamista ja huolellisuutta

8. Jälkimmäinen isompi homma. Toisaalta tämä kyllä tuntuisi parantavan tehtävänantoa ainakin opettajan näkökulmasta, koska pakottaa tarkempaan ohjeistukseen.

9. Koodit tällä kurssilla aika yksinkertaisia. Miten toimisi sitten, jos olisi selkeästi monimutkaisempia tehtäviä. Pitäisikö palastella pienempiin palasiin vai onnistuisiko ohjeistamaan ja määrittelemään tarkastukset pidempään koodiin

10. Entä jos kurssilla olisi muutaman kymmenen opiskelijan sijasta 100 tai 1000 opiskelijaa. Kuvan palauttaminen ei ainakaan olisi enää relevantti tapa, koska niiden läpikäynti kestäisi liian kauan. Pitäisi vain tulla tieto, ketkä tehneet onnistuneesti.

Kokemuksia tarkastusvaiheesta:

1. Tarkastaminen on nopeampaa, kun tarvitsee katsoa vain kuva.
2. Nyt vielä joidenkin opiskelijoiden (1/4 opiskelijan) kohdalla joutui katsomaan koodista, kun oli joku epäselvyys tarkastuksessa. Luulisin, että tämä paranee, kun tehtäviä oppii tekemään siten, että tehtävänantoon ei jää ”aukkoja”, että tehtävät voisi tehdä usealla tavalla.

Kehitysajatuksia:

1. Kellonaika saattaisi ehkäistä sitä, että käyttää toisen kuvaa tai vanhaa kuvaa
2. Tietokantaan suoraan tieto (opiskelijan pitäisi antaa nimensä). Tämä toki heikentäisi yleiskäyttöisyyttä ja tekisi toteutuksesta kurssikohtaisen.
3. Voisiko ketjutarkastuksissa olla ketjun keskellä jokin tarkka sana tai arvo. Tällä voisi esim erottaa pinMode komennon tarkastamisen DigitalWriten tarkastamisesta.