Turun yliopisto
University of Turku

Motivaatio ensimmäisellä ohjelmointikurssilla

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos
Pro gradu -tutkielma
12.05.2019
Justus Hedberg

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

JUSTUS HEDBERG: Motivaatio ensimmäisellä ohjelmointikurssilla

Tutkielma, 90 s., 1 liites.
Tietojenkäsittelytiede
Toukokuu 2019

Ensimmäisten ohjelmointikurssien keskeytysaste on yhä verrattain suuri, vaikka tilanteen parantamisen eteen on tehty jo pitkään töitä. Motivaation roolia opiskelijoiden menestyksessä ei ole etenkään juuri tällä saralla tutkittu. Tämän tutkielman tarkoituksena onkin kartoittaa ensimmäisen ohjelmointikurssin opiskelijoiden motivaatiota ja asenteita ohjelmointia kohtaan. Myös yliopiston ja avoimen yliopiston eroja motivaatiossa ja kurssisuoriutumisessa tarkastellaan. Lopuksi tutkitaan vielä, miten motivaatio ja kurssisuoriutuminen korreloivat keskenään.

Tutkielmassa on toteutettu määrällinen tutkimus, jonka tärkeimpänä aineistona oli motivaatiokyselydata, joka kerättiin ennen ja jälkeen ensimmäisen ohjelmointikurssin. Samaiselta kurssilta kerättiin myös opiskelijoiden suoriutumista kurssilla mittaavaa dataa. Kyselyn sekä datan keruun kohteena olivat Turun yliopiston ensimmäisen ohjelmointikurssin opiskelijat. Myös saman kurssin samaan aikaan suorittaneet avoimen yliopiston opiskelijat olivat tutkimuksen kohteena. Kurssi toteutettiin loppusyksystä 2017.

Tulokset osoittavat, että motivaatio ja asenteet ovat ensimmäisellä ohjelmointikurssilla korkeat. Ne eivät myöskään näyttäneet juurikaan muuttuvan kurssin aikana. Mitatun motivaation ja kurssisuoritusten väliltä ei löydetty tilastollisesti merkittävää korrelaatiota. Tuloksista voidaan siis päätellä, että opiskelijoiden oma arvio motivaatiostaan ei suoraan vaikuta heidän kurssisuorituksiinsa.

Tulevaisuudessa olisi mielenkiintoista suorittaa sama motivaatiokysely myös jollain muulla ensimmäisellä ohjelmointikurssilla, jossa opetusmetodit ja -järjestelmät ovat erilaiset. Myös itse motivaatiokyselyä voisi parannella kyselyn ollessa ensimmäinen laatuaan.

Asiasanat: motivaatio, motivaatiokysely, ensimmäinen ohjelmointikurssi, ohjelmointi

UNIVERSITY OF TURKU
Facility of Future Technologies

JUSTUS HEDBERG: Motivation on a first programming course

Thesis, 90 p., 1 appendix p.
Computer science
May 2019

The drop out rates in first programming courses are still high, regardless of the long-lasting efforts towards making them better. The effect that motivation has in students' performance hasn't been studied that much, especially in this area. The meaning of this study is to investigate the motivations and attitudes of students towards programming. The differences in motivations and the overall success in the first programming course between university students and open university's students is also examined. Finally, an investigation was performed on how motivation and course performance correlate with each other.

The study made in this thesis is a quantitative one, and the main material consisted of the data from a motivation questionnaire, which was collected before and after the first programming course. Data that measured students' performance on the same course was also collected. The target group of these data collections were students of the first programming course arranged by University of Turku. Students participating in a corresponding course at the open university were also part of the data collection. The programming course at hand was arranged in the Autumn of 2017.

The result show that motivation and attitudes are high on the first programming course. In addition, no significant change in these was observed during the course. No correlation was found between the measured motivation and the course performance. From these findings it can be concluded, that students' performance is not affected by how they perceive their own motivation.

In the future it would be interesting to use the same motivation questionnaire for courses that have different kinds of teaching methods and systems. In addition, the motivation questionnaire could be enhanced as it's first of its kind in this area.

Keywords: motivation, motivation questionnaire, first programming course, programming

Sisällys

1 JOHDANTO

Ohjelmointi on haastava ja laaja aihealue, jonka haasteina ovat muun muassa vaikeat abstraktit konseptit, sekä muutenkin laaja asioiden kirjo, joka tulee omaksua verrattain nopeasti (Lahtinen, Ala-Mutka ym. 2005, Milne, Rowe 2002, Robins, Rountree ym. 2003). Yksi ensimmäisten ohjelmointikurssien ongelmista onkin suuri keskeytysprosentti (Bennedsen, Caspersen 2007, Watson, Li 2014, Lahtinen, Ala-Mutka ym. 2005). Tilanne on parantunut huomattavasti vuosien varrella, mutta keskeytysprosentit ovat silti maailmanlaajuisesti suuria. Tähän voisi löytyä ratkaisu muualtakin kuin erilaisista opetusmetodeista, joilla tilannetta on koitettu pitkään parantaa: voisiko esimerkiksi opiskelijoiden asenne ja motivaatio olla avainasemassa? Jotta opiskelijoiden asenteisiin ja motivaatioihin voitaisiin puuttua, tulisi niistä kuitenkin tietää jotain. Ongelmana on, että varsinkaan ohjelmointikursseille ei juurikaan ole kohdennettuja kyselyjä, joilla näitä voisi mitata. Tässä tutkielmassa tehty motivaatiokysely oli ensimmäinen laatuaan sen kohdekurssilla, sekä kohdeyliopiston ohjelmointikursseilla yleisestikään. Millainen on siis ensimmäiselle ohjelmointikurssille osallistuvan opiskelijan motivaatio? Entä suhteutuuko tämä motivaatio jotenkin opiskelijan kurssisuorituksiin?

Tutkimuksen tavoitteena on muodostaa erityisesti ohjelmointikurssille sopiva motivaatiota ja asenteita mittaava kysely, ja tämän kyselyn avulla kartoittaa ensimmäisen ohjelmointikurssin opiskelijoiden motivaatiota ohjelmointia kohtaan. Samalla tarkastellaan yliopisto-opiskelijoiden ja avoimen yliopiston opiskelijoiden motivaatioiden eroja. Lisäksi tavoitteena on tutkia, löytyykö motivaatiokyselyn sekä oppilaiden kurssisuoriutumisen väliltä jonkinlaista korrelaatiota. Tässä yhteydessä tarkastelun rajoitteena voidaan pitää sitä, että kyseinen kysely toteutetaan ensimmäistä kertaa: näin ollen sen täyttä toimivuutta ei ole vielä todennettu. Käytetty kysely pohjautuu kuitenkin hyvin pitkälti Wong K. Y. ja Chen Q. (2012) kehittämään ja toimivaksi todettuun asennekyselyyn, joten tulokset ovat käyttökelpoisia.

Tutkimuksen kohteena ovat Turun yliopiston ensimmäisen ohjelmointikurssin opiskelijat, niin yliopiston puolella kuin avoimenkin yliopiston puolella opiskelevat. Kerättävä kyselydata on kvantitatiivista, samoin kuin on myös kurssilta yleisesti saatava

suoriutumisdata. Tutkimusta lähestytään avaamalla ensin, minkälaisia metodeja ja malleja nykyään on käytössä ohjelmoinnin opetuksessa. Näin pyritään saamaan yleinen käsitys siitä, mitä vaikeuksia ohjelmoinnin opetuksessa on, ja miten niitä on pyritty opetuksen avulla ratkaisemaan. Esimerkiksi käänteinen oppiminen ja pariohjelmointi ovat esimerkkejä menetelmistä, joita käytetään ohjelmoinnin opetuksessa.

Myös ohjelmoinnin oppimisen vaikeutta tarkastellaan yleisellä tasolla: miten oppiminen siis oikeasti tapahtuu, ja mitä tekijöitä siihen vaikuttaa? Nykyään ohjelmoinnin yhteydessä suosittu oppimisteoria on konstruktivistinen lähestymistapa, joka korostaa ihmisen omaa roolia tiedon rakentajana, eikä pelkästään sen vastaanottajana (Schunk 2012). Lopulta päädytään kuitenkin motivaation rooliin opetuksessa ja oppimisessa: jotta ihminen pystyisi rakentamaan omaa tietouttaan, tulisi tällä olla myös jonkinlainen halu siihen (Schunk 2012). Behavioristista, passiivista aivojen tiedolla tankkaamista, ei enää nykypäivänä katsota hyvällä - saati koeta tehokkaaksi ja mieluisaksi oppimistavaksi. Toki motivaatiotakin on monenlaista, esimerkiksi sisäistä ja ulkoista, kuten muun muassa Ryan ja Deci (2000) esittävät. Oli käsitys motivaatiosta millainen tahansa, sen roolin oppimisprosessissa voidaan katsoa olevan joka tapauksessa hyvin tärkeä.

Tulevat kappaleet jäsentyvät edellä esitetyn laisesti, ikään kuin ulkoa sisäänpäin oppimista tarkasteltaessa. Aluksi esitetään, millä tavoin ohjelmointia opetetaan, miten se siis tapahtuu ulkoisesti havainnoituna. Tämän jälkeen perehdytään, miten oppiminen, ja erityisesti ohjelmoinnin oppiminen, tapahtuu. Mitä siis tapahtuu pään sisällä opetuksen vaikutuksesta. Lopuksi pyritään miettimään, miksi oppimista tapahtuu; mikä on se jokin ihmiseen sisälle rakennettu systeemi, joka tekee oppimisesta tehokasta ja ennen kaikkea mielekästä. Tarkasteluosuuden jälkeen esitellään itse tutkimus sekä kohderyhmä, sekä saadut tutkimustulokset. Lopuksi tutkimustuloksista nostetaan esille muutamia mielenkiintoisimpia kohtia, ja tarkastellaan mahdollisia syitä saaduille tuloksille. Viimeisenä yhteenvedossa tiivistetään jälleen tutkielman sisältö ja sen esittämät pääkohdat.

2 OHJELMOINNIN OPPIMINEN

Ohjelmoinnin on todettu olevan yleisesti haastava aihealue niin oppimisen kuin opetuksenkin saralla, ja etenkin alustavilla ohjelmointikursseilla (introductory programming courses) ensi kertaa olevien opiskelijoiden keskeytysaste on suuri (Bennedsen, Caspersen 2007, Watson, Li 2014, Lahtinen, Ala-Mutka ym. 2005). Vaikeimpia asioita on todettu olevan muun muassa konseptit, jotka vaativat ymmärrystä isommista kokonaisuuksista, sekä etenkin abstraktit konseptit, kuten osoittimet ja muistin toimintaan liittyvät asiat (Lahtinen, Ala-Mutka ym. 2005, Milne, Rowe 2002, Robins, Rountree ym. 2003). Myös esimerkiksi luokkakokojen suuruus sekä niiden heterogeenisyys mainittiin hankaloittavana tekijänä, jolloin jonkin yhden yleisen opetustavan käyttö ei välttämättä tehoa kaikkiin, mikä hankaloittaa edelleen etenkin suurien luokkien tehokasta opettamista (Wulf 2005, Lahtinen, Ala-Mutka ym. 2005). Kanditason opiskelijoille yleisenä ohjenuorana ohjelmoinnin opetuksessa voidaan pitää ACM:n ja IEEE-CS:n ohjeita, jotka kertovat sekä yleisesti mitkä opetuksen tavoitteet ovat, että mitä opetuksen tulisi sisältää ja minkä verran (Task Group on Information Technology 2017). Tämä heijastaa hyvin myös niitä oppimistavoitteita ja -määriä, joita yleisestikin ensimmäisenä ohjelmoinnin kannalta olisi hyvä opettaa ja oppia. Tärkeänä päämääränä opinnoissa näiden ohjeiden mukaan pidetään hyviä valmiuksia työelämään, sekä avointa ja joustavaa asennetta nopeasti muuttuvaa alaa ja teknologioita kohtaan. Mitä itse oppimiseen tulee, artikkeli ei kata erilaisia oppimismenetelmiä tai tapoja, joilla asiat tulisi opettaa, vaan antaa lähinnä tavoitteet, joihin kunkin laitoksen tulisi itse valitsemillaan metodeilla päästä.

Kuten jo mainittiin, etenkin ensimmäiset ohjelmointikurssit ovat osoittautuneet erityisen hankaliksi ja työläiksi: olisiko siis ohjelmoinnin oppimisessa syytä ottaa huomioon kenties jotain erityistä? Kuten Lahtinen E. ym. (2005) sekä Butler M. ja Morgan M. (2007) ovat saaneet selville, yleisimpiä heikkoja kohtia oppilailla ensimmäisellä ohjelmointikurssilla ovat abstraktien asioiden ja kokonaisuuksien ymmärtäminen, sekä erityisesti tällaisten abstraktien rakenteiden käytännön soveltaminen. Nämä asiat ovat yleisestikin vaikeita, koska ne ovat, kuten sanottu, abstrakteja: ihmismielen on

vaikeampi saada ote jostain abstraktista ja ei-konkreettisesta asiasta. Itse oppimisprosessin vaikeutta ohjelmoinnin näkökulmasta on kuitenkin vaikea lähteä arvioimaan. Onneksi on tehty tutkimuksia, jotka näyttävät minkä tapaisia oppimis- ja opetusmenetelmiä ohjelmoinnin kanssa kannattaisi käyttää. Esimerkiksi Lahtinen E. ym. toteavat, että ohjelmoinnin opettelun tulisi sisältää enemmän käytännön harjoittelua: tämä näyttäisi viittaavan siihen, että heidän mielestään konstruktivistisen oppimisen tulisi olla enemmän käytössä ohjelmoinnin opetuksessa käytännönläheisyytensä takia. Heidän mukaansa myös tiedon käsittely -teoriaa olisi hyvä soveltaa: kyseinen teoria kuvaa oppimisen tapahtuvan assosioimalla opeteltava asia jo johonkin opittuun/tunnettuun asiaan. Näin pystyttäisiin kenties helpottamaan abstraktien asioiden ja kokonaisuuksien oppimista, kun uudet ja oudot asiat pystyttäisiin linkittämään tietoihin, jotka oppilaalla jo on. Tämän lisäksi Butler M. ja Morgan M. löysivät tarpeen suuremmalle määrälle palautetta ja opetusinteraktiota: Ohjelmointia on heidän mukaansa vaikea oppia, jos palautetta ei saa tarpeeksi ja oikeissa kohdissa. Tämä ajatusmalli viittaa puolestaan sosiaaliskognitiiviseen tarpeeseen oppimisessa, jossa ohjaajan rooli korostuu: oikeanlaisen ohjauksen avulla oppija pystyisi paremmin havaitsemaan ja oppimaan vaikeita syy-ja-seuraus-suhteita.

Seuraavaksi esitellään, millaisia oppimisteorioita yleisesti löytyy, ja avataan etenkin nykyään suosiossa olevia kognitivistisia teorioita hieman enemmän. Tämän jälkeen esitellään oppimista enemmän ohjelmointia silmällä pitäen, sekä avataan hieman teknologian osuutta oppimisessa, joka korostuu etenkin juuri ohjelmoinnin yhteydessä. Lopuksi käydään läpi myös muita tekijöitä, jotka vaikuttavat ohjelmoinnin oppimiseen, kuten motivaatio ja opiskelutavat.

2.1 Erilaiset oppimisteoriat yleisesti

Oppiminen voi tapahtua luonnollisesti monella eri tavalla ja yleensä ihmisillä on myös persoonalliset taipumuksensa oppimisen suhteen. Schunk D. (2012) esittää neljä oppimisteoriaa, joihin käsitys oppimisesta voidaan sanoa perustuvan: behaviorismi (behaviorism), sosiaaliskognitiivinen teoria (social cognitive theory), tiedon käsittely -teoria (information processing theory) ja konstruktivismi (constructivism). Esimerkiksi

näistä ennen vallassa ollut käsitys, behaviorismi, on nykyään jäänyt taka-alalle, ja sen paikan oppimisprosessin kuvaajana ovat korvanneet kognitivistiset teoriat. Tällaisia ovat loput kolme, joissa oppiminen perustuu kognitiivisiin tapahtumiin käytöksen muuttamisen sijaan. Kukin näistä kolmesta käsittelee oppimista omalla tavallaan. Esimerkiksi sosiaaliskognitiivinen teoria pohjautuu ajatukseen siitä, miten ihminen oppii sosiaalisten kokemusten kautta, esimerkiksi havainnoimalla, kun jotain tehdään, ja erilaisia syy-seuraus-suhteita ymmärtämällä ja niistä oppimalla. Toisaalta taas konstruktivismi korostaa ihmisen sisäisiä kognitiivisia prosesseja: Ihminen rakentaa itse oppimansa asian ulkoisten ärsykkeiden avulla. Tieto ei siis vain välity ja sitä ei opita pelkästään jotain opiskelemalla, vaan tietous syntyy henkilön itsensä rakentamana, etenkin yrityksen ja erehdyksen kautta. Teorioita on siis monia, ja ne ovat syystäkin erillisiä teorioita, mutta nykyaikaisen oppimiskäsityksen keskiössä on kuitenkin oppiminen kognitiivisten prosessien kautta.

Myös esimerkiksi Kay D. ja Kibble J. (2016), sekä Pylkkä O. (2018) ovat päätyneet samantapaiseen jakoon behavioristisen sekä kognitivistisen teorioiden välillä. Vaikka heidän esittelemänsä teoriat noudattelevatkin pitkälti Schunkin esittämä jakoa, myös pieniä eroavuuksia löytyy. Lähes samaa jakoa käyttävät artikkelissaan Kay D. ja Kibble J., jossa he jakavat teoriat viiteen eri osaan: behaviorismiin (behaviorism), sosiaaliskognitiiviseen teoriaan (social cognitive theory), kognitiiviseen oppimisteoriaan (cognitive learning theory), konstruktivismiin (constructivism) ja sosiaaliseen konstruktivismiin (social construktivism). Jako behaviorismiin sekä kognitivistisempiin teorioihin näkyy tässäkin, kuten näkyy myös jako kognitiivisiin teorioihin sekä konstruktiivisiin teorioihin. Myös Pylkkä O. on tehnyt eron behavioristisen sekä kognitivistisen oppimisteorian välille. Tämän lisäksi kognitivistinen oppimisteoria on jaettu vielä kolmeen osaan: kognitivistiseen, konstruktivistiseen sekä kokemukselliseen paradigmaan.

Kaikissa näissä kolmessa voidaankin siis nähdä tietty jako, joka on nykyaikaiselle oppimiskäsitykselle yleinen: hieman vanhentunut käsitys ihmisestä pelkästään käytöksen kautta muokattavana objektina on jäänyt erilleen (behaviorismi), ja vallan

ovat saaneet käsitykset ihmisestä tietoisena tiedon käsittelijänä (kognitivistiset teoriat). Kuten edellä kuitenkin kävi ilmi, tätä ihmisen tietoista informaation käsittelyä ja oppimista voidaan kuitenkin lähestyä eri näkökulmista. Seuraavaksi käydään läpi yllä mainittuja oppimisteorioita, jaoteltuna behaviorismiin, kognitivistisiin teorioihin, sekä konstruktivistisiin teorioihin.

2.1.1 Behavioristinen oppimisteoria

Behavioristiseen oppimisteoriaan kuuluu käsitys ihmisestä passiivisena tiedon vastaanottajana: oppiminen on siis erilaisten ärsyke-reaktio -kytkentöjen muodostumista ja vahvistumista (Pylkkä 2018). Oppimisen voidaan ajatella olevan enemmänkin vain positiivisten ja negatiivisten assosiaatioiden muodostamista ja muokkaamista, eikä opetuksella tai oppimisella ole itsessään väliä, vaan ne ovat ikään kuin vain välineitä informaation siirtämisessä. Opettajan rooli behavioristisessa opetuksessa on suuri, sillä tämä on vastuussa sellaisen ympäristön luomisesta ja kontrolloimisesta, jossa ärsykkeet ovat oikeanlaiset, ja että oppilaiden vasteita niihin pystytään havainnoimaan mahdollisimman hyvin. Opettajan tehtävänä on myös tarkoituksellisen rakenteen luominen opetusmateriaalille niin, että toivotunlaista suorittamista pystytään lisäämään ja ei-toivottua vähentämään (Kay, Kibble 2016). Kuitenkin se miten opetetaan, on pienessä roolissa, eikä opetusmetodeilla ole niinkään väliä, kunhan oppilaiden ja oppimisen ulkoinen säätely onnistuu.

Kuten myös Schunk (2012) kirjoittaa, behaviorismi selittää oppimisen ympäristöön liittyvillä tapahtumilla. Se ei kuitenkaan kiellä esimerkiksi mentaalisten prosessien olemassa oloa, tai kognitiivisia teorioita, vaan sanoo vain, etteivät ne ole välttämättömiä oppimiselle. Schunk nostaa vahvasti myös esille B. F. Skinnerin, jota pidetään välineellisen ehdollistumisen (operant conditioning, Schunk käyttää operant learning) isänä: Skinnerin idean pohjana on käytännössä juuri ehdollistuminen, erilaiset ärsyke-reaktio -kytkentöjen sarjat. Ihmisen ajatellaan oppivan asioita samalla tapaa kuin eläimetkin: kun ärsykkeeseen saadaan halutunlainen reaktio, sitä vahvistetaan palkinnolla, ja ei-halutunlaisesta reaktiosta rangaistaan. Tällä tavalla ajatellen oppimiseen vaikuttavat siis vain kehityksen tila sekä ärsykevahvisteiden historia; näin ollen myös henkilön reaktiot heijastavat vain henkilön oppimia ärsykevahvisteita.

Tällainen käsitys ihmisen oppimisesta alkaa kuitenkin olla jo vanhentunut, ja kuten Schunk mainitsee, sen haastoivat ja pikkuhiljaa korvasivat 1960-luvun vaihteessa kognitivistiset teoriat, jotka nykyään ovat vallassa.

2.1.2 Kognitivistiset teoriat

Kognitivistiset teoriat saivat alkunsa 1960-luvun alussa, syrjäyttäen behavioristisen oppimiskäsityksen (Pylkkä 2018). Kuten sen nimikin sanoo, kognitivistiset teoriat pohjaavat ajattelunsa ihmismielen sisäisiin ilmiöihin, kognitiivisiin prosesseihin. Behavioristisen oppimiskäsityksen vastaisesti oppimisen ei enää ajateltu tapahtuvan ulkoisten tekijöiden summana, vaan päinvastoin oppiminen ajateltiin olevan ihmisestä lähtöisin: oppiminen nähdään tiedon/informaation prosessointina ihmisen itsensä toimesta. Olennaista on juuri ajatus informaation prosessoinnista, jossa oppiminen ajatellaan tietynlaisena sarjana prosesseja, jotka johtavat tiedon varastointiin aivoihin. Yleinen kognitivistinen ajatus onkin oppimisprosessin jakautuminen pienempiin paloihin, jotka muodostavat tämän ”prosessiputken” informaation vastaanottamisesta sen tallentamiseen: sensoriset rekisterit, lyhytkestoinen muisti (usein käytetään myös termiä työmuisti), sekä pitkäkestoinen muisti (Kay, Kibble 2016, Schunk 2012). Sensoriset rekisterit vastaanottavat tietoa (esimerkiksi silmä tai korvat), ja ovat koko ajan altistuneina jonkinlaiselle informaatiolle. Sensoriset rekisterit ”koodaavat” tämän informaation ja pitävät sitä tallessa pienen hetken, jonka jälkeen informaatio häviää, mikäli sille ei anneta sen suurempaa huomiota. Vasta kun ihminen päättää keskittyä johonkin tiettyyn informaatioon, se siirtyy eteenpäin lyhytkestoiseen muistiin, jossa kyseinen informaatio oikein käsiteltynä päätyy pitkäkestoiseen muistiin. Tällaiset menetelmät, joilla informaatio käytännössä siirtyy lyhytkestoisesta muistista pitkäkestoiseen muistiin, ovat yksi debatin aihe, ja yhteisymmärrystä jostain tietystä mallista ei ole. Yksi vallalla oleva malli esittää, että informaation ollessa lyhytkestoisessa muistissa, sinne tuodaan jokin pitkäkestoisen muistin pala, johon uusi informaatio sitten yhdistetään, ja näin siirretään pitkäkestoiseen muistiin. Pitkäkestoisesta muistista myös haetaan tietoa, kun sitä tarvitaan, esimerkiksi kun koitetaan muistaa jotain tiettyä asiaa. Tällainen malli on juuri kognitivistiselle ajatusmallille oleellinen, sillä siinä uusi opittava

asia sidotaan johonkin jo omattuun informaatioon. Tähän ajatukseen perustuu myös esimerkiksi skeemateoria, joka on myös psykologiassa hyvin tunnettu.

Skeemateoria kuvaa ihmisen ajattelua skeemojen avulla: kun ihminen havaitsee jotain, hän muodostaa havainnosta hyvin nopeasti jonkin mallin päässään, jonka hän perustaa aiempaan tietoon kyseisestä havainnosta. Esimerkiksi jos ihminen havaitsee appelsiinin, hän nopeasti muodostaa mielessään kuvan siitä, millainen sen tulisi olla ja miten sen tulisi reagoida: appelsiinin kuuluisi tuntua hieman pehmeältä, ja sen sisällön tulisi olla tietynlaista. Jos kuitenkin esimerkiksi kuoren alta löytyy jonkinlainen muu sisusta kuin appelsiinilla kuuluisi olla, tämä havainto ei sovi henkilön skeemaan. Skeeman voidaan siis ajatella olevan malli jostain, yleensä reaalimaailman asiasta. Kuten Arbib M. (1992) sanoo, skeema on jotain, mitä on opittu maailmasta, yhdistämällä tieto sen soveltamiseen tarvittaviin prosesseihin. Skeemojen voidaan ajatella sijaitsevan pitkäkestoisessa muistissa, jossa niitä käytetään uuden oppimisessa, ja jossa niitä myös päivitetään. Skeemateorian mukaan oppiessa tapahtuu siis juuri tätä, uusien skeemojen muodostumista ja vanhojen päivittymistä.

Kognitivististen teorioiden yhteydessä on saanut suosiota myös teorioiden niin sanottu sosiaalinen puoli, sosiaaliskognitiivinen teoria. Se korostaa oppimisen tapahtumista sosiaalisessa ympäristössä: myös muita tarkkailemalla ihminen voi oppia asioita, kuten tietoa, taitoja, asenteita, tapoja jne. Ihmiset oppivat erityisesti malleista, joissa kohtaavat oikeanlainen käyttäytyminen ja sen aiheuttamat seuraukset. Mallit ovatkin oikeastaan sosiaaliskognitiivisen teorian perusta: teorian mukaan ihminen pystyy oppimaan myös muiden toteuttamista käytösmalleista. Esimerkiksi jos oppija tarkkailee jotain toista, joka toteuttaa tietyn käyttäytymismallin, ja tästä aiheutuva vaste on positiivinen, oppija oppii, että tätä mallia toteuttamalla vasteen pitäisi olla positiivinen. Oppijan ei siis tarvitse itse suorittaa kyseistä mallia, ja oppia kokeilun ja erehtymisen kautta, vaan oppiminen voi tapahtua suoraan pelkän tarkkailun avulla, ja kun oppija havainnoi toisen suorittaman mallin tapahtumaketjun palaset, hän pystyy myös itse toistamaan kyseisen mallin (Kay, Kibble 2016, Schunk 2012). Ulkoisesti tämä malli voi vaikuttaa aika behavioristiselta, sillä se perustuu siihen, saako jokin toiminta positiivisen vai

negatiivisen vasteen. Sosiaaliskognitiivinen teoria kuitenkin pohjaa oppimiskäsityksensä kognitiivisiin prosesseihin: se uskoo, että tällaiset tapahtumat toimivat ulkoisena ärsykkeenä sisäisen kognitiivisen oppimisprosessin laukaisijana. Etenkin skeemateoria on ainakin ideatasolla hyvin lähellä sosiaaliskognitiivista teoriaa.

2.1.3 Konstruktivistiset teoriat

Kuten Schunk (2012) toteaa, konstruktivismin ei voida varsinaisesti sanoa olevan teoria, sillä siitä ei ole jotain tiettyä ja yhtä käsitystä, jota pystyttäisiin käyttämään tutkimuksiin ja hypoteesien kehittämiseen: se on lähinnä tietynlainen katsantokanta oppimiseen. Siitä pystytään kuitenkin vetämään suurpiirteisiä linjoja, joita pystytään käyttämään teorian tapaan, joten ainakin tämän tutkielman ja fokuksen puitteissa lienee oikeutettua käsitellä konstruktivismia teoriana.

Vaikka konstruktivistiset teoriat pohjaavat kognitiivisiin prosesseihin, ne eroavat lähtökohdiltaan kognitivistisista teorioista. Kun kognitivistiset teoriat kuvaavat oppimisen tietynlaisena tiedon siirtymisenä, konstruktivistiset teoriat ajattelevat oppimisen enemmänkin aktiivisena tiedon konstruointiprosessina, jossa tietoa rakennetaan aktiivisesti aktiivisen vastaanottamisen sijaan. Konstruktivismin keskeisenä ajatuksena siis on, että tieto ei siirry, vaan oppija konstruoi eli rakentaa sen itse uudelleen. Oppijan aikaisemmat käsitykset, tiedot ja kokemukset määrittävät sen, miten oppija uuteen informaatioon suhtautuu ja miten hän asian tulkitsee. Olennaista on siis oppilaan oma toiminta sekä ajattelun aktivoituminen. Toisin sanoen, kuten Pylkkä O. (2018) asian ilmaisee, oppiminen on oppijan oman toiminnan tulosta. Kay D. (2016) puolestaan korostaa, miten konstruktivismissa käsitetään ihmisen mieli ja maailmankuva yleisesti eri tavalla. Aikaisemmissa teorioissa oletuksena oli, että ulkoista maailmaa tarkastellaan aina samalla tapaa, ja että ihmismieli on tyhjä paperi, johon ulkoisesta maailmasta kopioidaan tietoa. Konstruktivismi puolestaan esittää, miten jokainen ihminen tarkastelee ulkoista maailmaa oman ”linssinsä” läpi, miten tieto on subjektiivista, ja miten sitä rakennetaan aktiivisesti oppijan omien kokemusten kautta.

Myös konstruktivismista voidaan erotella vielä enemmän sosiaalisuutta painottava haara. Ajatusmalli on muuten sama kuin konstruktivismissa yleisestikin, mutta

sosiaalista interaktiota ympäröivien ihmisten ja kulttuurien kanssa korostetaan suuresti (Kay, Kibble 2016). Toisin sanoen oppija muovautuu ”väkisinkin” ympäristönsä kaltaiseen suuntaan. Tämä voidaan konstruktivismista periaatteessa suoraankin johtaa: jos ympäristö on muokannut oppijan ”linssin” tietynlaiseksi, tämä oppii tämän linssin korostamia asioita. Ympäristön vaikuttavia kulttuurisia ja sosiaalisia tekijöitä ovat esimerkiksi kieli, kirjoitus, kalenteri, taide, teknologia; loppujen lopuksi mikä vain väline, jota ihminen käyttää selvitäkseen ja menestyäkseen ympäristössään (Kay, Kibble 2016). Kulttuurin, yhteiskunnan sekä sosiaalisen kanssakäymisen rooli oppimisessa siis korostuu, ja niiden ajatellaan vaikuttavan suuresti juuri siihen, mitä ja miten ihminen oppii.

2.2 Oppimisteoriat ohjelmoinnin oppimisessa

Miten oppimisteoriat sitten näkyvät käytännön oppimisessa ja opetuksessa? Kuten nimetkin kertovat, ne ovat teorioita, jotka yrittävät kuvata, miten oppiminen tapahtuu. Opetuksessa näitä teorioita ei pysty suoraan soveltamaan, vaan niille pitää löytää jotkin käytännön toteuttamistavat ja metodit. Esimerkiksi konstruktivistisen teorian mukaan oppiminen tapahtuu oppijan itsensä toimesta rakentamalla vastaanotettava tieto uudestaan, ja että olennaista on oppijan oma aktiivisuus ja ongelmanratkaisu opittavaa asiaa kohtaan. Käytännön toteutuksena ei siis vain riitä, että annetaan oppijalle materiaalia, josta tämän tulisi innostua ja oppia, vaan myös ympäristön ja ohjauksen tulee tukea oppimista. Mahdollisuuksia ja tapoja oppimisen tukemiseen on monia, ja yhden mallin esittelee esimerkiksi Jonassen D. (1999). Se koostuu kuudesta kohdasta, joita opettajan tulisi toteuttaa: tarjoa ongelma/kysymys/projekti, tarjoa esimerkkitapaus, tarjoa helposti lähestyttävää materiaalia, tarjoa kognitiivisia työkaluja (tarkoituksena siis kohdistaa kognitiivinen toiminta ydinasioihin, esimerkiksi helpottamalla tiedon hakua), tarjoa mahdollisuus ja työkalut keskustelulle ja yhteistyölle, ja tarjoa sosiaalista/kontekstuaalista tukea. Jonassen korostaa miten oppimisen ytimen ollessa oppijan oma tiedon rakentamisessa kokemuksien tulkinnan kautta, myös opetuksen pitäisi olla sellaista, että oppija pystyy reflektoimaan uutta informaatiota itsenäisen tulkinnan kautta.

Wulf T. (2005) puolestaan keskittyy siihen, miten konstruktivistinen opetusmalli toimii erityisesti ohjelmoinnin opetuksessa. Hän rinnastaa konstruktivismin perinteiseen behavioristiseen opetusmalliin, jossa opettaja vain luennoi ja oppilaat kuuntelevat, ja esittää, että konstruktivistinen malli on parempi etenkin ohjelmoinnin opetukseen: Tällöin oppilaskeskeisyys korostuu, ja näin ollen pystytään aktivoimaan laajempaa skaalaa oppilaita esimerkiksi lähtötasosta riippumatta. Ennen kaikkea on hyödyllistä saada oppilaat aktivoitua harjoittelemaan myös käytännön ohjelmointia, jonka puute koetaan yhdeksi suureksi tekijäksi ja syyksi miksei ohjelmointia opita (Lahtinen, Ala-Mutka ym. 2005, Robins, Rountree ym. 2003). Myös Erdei R. ym. (2017) tarkastelevat konstruktivistisen lähestymistavan vaikutusta ohjelmointikurssiin: heidän tutkimuksessaan opetus suoritettiin kognitiivisen tuen kautta (cognitive scaffolding), jossa kurssin alussa oppilaita tuetaan paljon, ja kun oppilaat alkavat hallitsemaan oppimismenetelmät, tukea vähennetään pikkuhiljaa kurssin loppua kohden. Tutkimuksessa vertailtiin erilaisia konstruktivistiseen teoriaan pohjautuvia menetelmiä, ja kuten Erdei R ym. mainitsevatkin, nämä menetelmät ovat jo onneksi löytäneet tiensä ohjelmoinnin opetukseen.

Ohjelmoinnin opetuksessa on aktiivisessa käytössä myös Bloomin taksonomia: sen avulla pystytään määrittämään eri osaamistasoja oppijalle, mikä puolestaan helpottaa oppilaan etenemisen tarkkailua, ja näin ollen siihen vaikuttamista. Kuten Buckley ja Exton (2003) esittävät, nämä tasot pyrkivät määrittämään oppijan kulloisenkin tiedon tason käsillä olevan aiheen suhteen. Tasolta toiselle siirrytään aina, kun edellisen tason määrittelemä tietous on saavutettu. Tasot ja niiden rajat eivät kuitenkaan ole niin selviä kuin miltä ne tässä ehkä kuulostavat, vaan ne pohjautuvat enemmän kognitiivisiin virstanpylväisiin kuin selkeisiin tasoihin. Vuonna 1956 taksonomian kehittänyt Benjamin Bloom jakoi oppimisen kolmeen eri osa-alueeseen, joista yksi oli kognitiivinen osa-alue, johon taas Bloomin taksonomian kuusi tasoa kuuluvat: nämä tasot määrittelevät siis ennen kaikkea kognitiivisen teorian mukaista oppimista. Bloomin taksonomian on todettu olevan hyödyllinen myös ohjelmoijien tietotasojen määrittelyssä, ja sitä onkin käytetty useaan otteeseen pohjana erilaisissa arvioinneissa (Buckley, Exton 2003, Scott 2003, Johnson, Fuller 2006). Taksonomiaa on myös kritisoitu

ja on koitettu löytää parempia malleja kuvaamaan näitä tasoja (Johnson, Fuller 2006, Krathwohl 2002, Thompson, Luxton-Reilly ym. 2008). Bloomin taksonomiasta onkin kehitetty myös uusi, päivitetty versio, joka kulkee nimellä päivitetty Bloomin taksonomia (revised Bloom's taxonomy), joka tulee erottaa alkuperäisestä taksonomiasta sen erilaisen tasojaon tähden (Krathwohl 2002). Bloomin taksonomia on hyvä esimerkki kognitivistista oppimisteoriaa toteuttavasta ajatusmallista, joka on helpompi tuoda käytännön opetukseen kuin itse teoria.

Kuten aikaisemmin todettiin, oppimisteoriat kuvaavat loppujen lopuksi sitä, miten oppimisen ajatellaan ihmisen mielessä tapahtuvan; näiden teorioiden käytännön soveltaminen on kuitenkin hieman vaikeampaa. Ohjelmoinnin oppimisen voidaan ajatella tapahtuvan oppimisteorioiden tasolla samalla tapaa kuin minkä tahansa muunkin oppimisen: esimerkiksi nykyään on suosiossa käsitys ihmisestä kognitiivisten prosessien kautta oppivana olentona, sekä edelleen käsitys konstruktivistisesta oppimisesta. Toteutustavat ovat kuitenkin se, missä ohjelmoinnin oppimisen voidaan ajatella erottuvan ”tavallisesta oppimisesta”, ja esimerkiksi miten konstruktivistinen opetus näkyy ohjelmoinnissa voi erota jonkin toisen aiheen opetuksesta. Thompson ja Luxton-Reilly ym. (2008) mukaan olisi tärkeää tarkkailla ohjelmoinnissa tarvittavia kognitiivisia prosesseja, jolloin ohjelmoinnin oppimisen vaikeuksia pystyttäisiin ehkä ymmärtämään paremmin. Oppimisteoriat voivatkin auttaa ymmärtämään, miksi ohjelmointia on niin vaikea oppia: esimerkiksi vaikeiksi oppimisen kohteiksi todetut abstraktit konseptit voivat olla vaikeita juuri niiden abstraktiuden takia. Kuitenkin kun tämä seikka on tunnistettu, siihen on mahdollista myös puuttua, ja pystytään esimerkiksi miettimään keinoja, joilla autettaisiin opiskelijoita samaistumaan näihin abstrakteihin käsitteisiin.

2.3 Teknologia ohjelmoinnin oppimisessa

Teknologialla on huomattava rooli etenkin ohjelmoinnin opetuksessa: on helpompi opetella ohjelmointia suoraan kääntäjän kautta, kuin erikseen esimerkiksi paperille kirjoittamalla. Teknologian positiiviset vaikutukset oppimistuloksiin ovat kuitenkin olleet pitkään debatin aiheena, ja esimerkiksi Ally M. (2004) ja Price L. (2014) toteavat,

ettei mitään kunnon todisteita teknologian suorasta hyödystä oppimiseen voida osoittaa. Ally M. argumentoi, että teknologiasta ei suoranaisesti olisi hyötyä itse oppimisprosessissa, vaan se vain mahdollistaa laajemman ja interaktiivisemman materiaalin jaon ja kanssakäymisen. Price L. mukaan teknologian käyttöönotolla ei ole saatu hyötyjä suureksi osaksi sen takia, että käyttöönoton pohjana on yleensä pidetty opettajien subjektiivisia käsityksiä mahdollisista hyödyistä. Hän mainitsee myös, miten useat tutkimukset eivät tuo julki suoraan mitään valideja tuloksia, vaan ovat usein ”lessons learned” -päättöisiä tutkimuksia: näin kunnon dataa teknologian ja oppimisen yhteydestä ei välttämättä ole syntynyt, esimerkiksi vaillinaisten lähteiden takia. Kuitenkin esimerkiksi visualisoinnin on todettu auttavan oppimisessa, etenkin ohjelmoinnin saralla (Kaila, Rajala ym. 2010). Vaikkei sitä puhtaan teknologiseksi työkaluksi voidakaan välttämättä kutsua, sen helppous juuri teknologian avulla on huomionarvoista. Myös esimerkiksi Martin-Blas T. ja Serrano-Fernández A. (2009) toteavat, että yhteistyö ja kommunikaatio lisääntyvät teknologian käytön myötä sekä oppilaiden kesken, että opettajan kanssa. Tätä voidaan pitää hyvänä asiana Lahtinen E. ym. (2005) sekä Wulf T. (2005) tutkimuksiin pohjustaen, joissa todettiin suurten ryhmäkokojen olevan yksi ohjelmoinnin oppimisen ongelmista. Myös ”puhtaita” hyviä tuloksia löytyy: Kaila E. ym. (2015) osoittivat, että oppimisteknologiaa käyttämällä onnistuttiin parantamaan oppilaiden tuloksia aktiivisten ja yhteistyöhakuisten oppimisprosessien kautta ohjelmointikurssilla.

Kuten aikaisemmin on todettu, ohjelmoinnin oppimisen suurimpia haasteita ovat esimerkiksi abstraktien konseptien oppiminen, suuret ja heterogeeniset ryhmäkoot, sekä kyky soveltaa teoriaa käytäntöön, eli siis käytännön ohjelmoinnin harjoittelun vähäisyys. Juuri todettiin myös, miten esimerkiksi visualisoinnin on todettu auttavan asioiden oppimisessa etenkin ohjelmoinnin yhteydessä. Tämä yhteys on kuitenkin lähes pääteltävissä, sillä visualisointi auttaa huomattavasti asioiden hahmottamisessa, joka puolestaan on yksi suuri tekijä ohjelmoinnin oppimisen vaikeudessa. Visualisoinnin voidaan siis todeta oleva ainakin ohjelmoinnin oppimisessa tärkeä väline. Visualisoinnin hyödyllisyyttä on myös pyritty mittaamaan ja ymmärtämään. Esimerkkinä Naps T. ym. (2002) kehittämä sitoutumistaksonomia (engagement

taxonomy), jonka tarkoituksena on mitata Bloomin taksonomiaan perustuen oppilaan aktiivista osallistumista visuaaliseen opetukseen ja materiaaliin; Naps T. ym. mukaan visualisointi ei itsessään ole tehokasta, vaan siihen tulee liittyä myös oppilaan aktiivista osallistumista.

Ohjelmoinnin oppimista helpottamaan on kehitetty myös muita teknologioita, kuten esimerkiksi ohjelmointikielet Logo ja Scratch. Näiden tarkoituksena ei ole opettaa mitään tiettyä ohjelmointikieltä, vaan ennemminkin ohjelmoinnissa tarvittavia tietorakenteita, konsepteja ja komentoketjuja. Näistä etenkin Scratch hyödyntää myös visualisointia, sen ollessa visuaalinen ohjelmointikieli. Siinä yhdistyy myös muita hyviä opetuksellisia elementtejä, kuten esimerkiksi runsas kommunikointi, palautteen anto ja esimerkkien kautta oppiminen, sekä oppilaan oman aktiivisuuden korostaminen (Maloney, Resnick ym. 2010). Scratchin ominaisuuksiin kuuluu lisäksi se, miten se poistaa mahdollisuudet esimerkiksi syntaksivirheille, jolloin sen kanssa ohjelmoitaessa huomio keskittyy enemmänkin esimerkiksi siihen, millaisia rakenteita ohjelmoimalla voi saada aikaan.

Kuten edellä annettu esimerkki Scratchistä kertoo, oppimisteknologian avulla pystytään parantamaan monia asioita. Oppimista edistää myös esimerkiksi lisääntynyt kommunikointi: ryhmissä opiskelijat pystyvät miettimään ratkaisuja ongelmiin yhdessä, joka taas puolestaan on yksi ydinasioista konstruktivistisessa opetuksessa. Kun tähän yhdistetään helposti saatava yhteys opettajaan, hyvä pohja oppimiselle on valmis: esimerkiksi juuri oppilaan lyhentynyt odotusaika avun saamisessa todettiin olevan hyödyllistä etenkin ohjelmoinnin oppimisessa (Lahtinen, Ala-Mutka ym. 2005, Wulf 2005). Oppimisteknologia mahdollistaa myös yksilöllisemmän opetuksen ja oppilaan tarkkailun, vaikka tämä ei olekaan vielä kovin yleistä. Esimerkiksi Turun yliopistossa käytettävä ViLLE-oppimisjärjestelmä kerää oppilaista suoritusdataa, jota voidaan käyttää tulosten tarkasteluun ja esimerkiksi erilaisten opetuskokeilujen toimivuuden kartoittamiseen (ViLLE Team 2015). Sama järjestelmä mahdollistaa myös automaattitarkisteiset tehtävät, jolloin opiskelija saa palautteen tehtävästään heti sen palautettuaan; vaikka tällainen palaute ei aivan vastaakaan opettajan antamaa

palautetta, on palaute yleisesti hyödyksi oppimiselle (Pashler, Cepeda ym. 2005). Opetusjärjestelmät helpottavat myös esimerkiksi yhteistyötä painottavien tehtävien ja harjoitusten toteuttamisessa, sekä lisäksi helpottavat opettajan työtä yleisesti, jolloin aikaa jää enemmän oppilaiden auttamiseen.

Opetusteknologiasta on suurta hyötyä myös käytännön ohjelmoinnin harjoittelussa: esimerkiksi edellä mainittu ViLLE-oppimisjärjestelmä mahdollistaa ohjelmointitehtävien automaattisen arvioinnin, jossa oppilas saa välittömän palautteen lisäksi myös visualisointeja ohjelmointiongelmista ja -ratkaisuista. Tällainen lähestyminen yhdessä esimerkiksi yhteisöllisen opetusmetodiikan kanssa antaa hyvät raamit opiskelijalle harjoitella käytännön ohjelmointia. Opetusteknologiasta voidaan siis ajatella olevan hyötyä myös käytännön ohjelmoinnin harjoittelussa niin ViLLEn kaltaisten alustojen avulla (Erkki Kaila, Teemu Rajala ym. 2015), kuin esimerkiksi Scratchin kaltaisten ohjelmointikieltenkin avulla (Rizvi, Humphries ym. 2011).

Teknologiasta vaikuttaisi siis olevan yleisesti hyötyä ohjelmoinnin opetuksessa. Hieman spekulatiivista kuitenkin on, auttaako opetusteknologia kirjaimellisesti oppimista, vai mahdollistaako se pikemminkin oppimismateriaalin interaktiivisuuden ja paremman jakelun. Esimerkiksi: teknologia mahdollistaa videoiden jaon ennen oppitunteja, jolloin kontaktiopetus on enemmän jo videoista opitun tiedon soveltamista, ja opettaja pystyy näin auttamaan oppilaita enemmän. Vaikka teknologia siis näin auttaa oppimisessa ja opetuksessa, voidaanko sen sanoa varsinaisesti opettavan mitään. Myös esimerkiksi visualisoinnin tapauksessa teknologiasta on suuri apu, koska sen kautta pystytään visualisoimaan paljon asioita suurelle määrälle ihmisiä, mutta voidaanko tällöinkään sanoa teknologian opettavan mitään, visualisointihan on kuitenkin mahdollista myös ilman teknologiaa. Joka tapauksessa voidaan todeta, että teknologiasta on suurta apua opetuksessa ja oppimisessa, niin yleisesti kuin ohjelmoinnin opetuksessakin. Teknologian merkitys etenkin juuri ohjelmoinnin oppimisessa korostuu, koska sen avulla pystytään helpottamaan huomattavasti kriittisiä oppimisen osa-alueita, kuten abstraktien konseptien ymmärtämistä. Teknologia voidaan ajatella myös enemmän yleisenä välineenä, jolloin siitä voidaan osoittaa olevan hyötyä sellaisenaan: kuten

Ivanovic M. ym. (2017) osoittaa, teknologisesti paranneltu oppiminen auttoi etenkin opetusmateriaalin organisoinnissa ja jakelussa, tehtäväaktiviteeteissa, sekä tiedotuksessa.

2.4 Muiden tekijöiden vaikutus ohjelmoinnin oppimiseen

Oppimiseen eivät vaikuta pelkästään oikein valittu oppimisteoria, sitä toteuttavat opetusmetodit, sekä opetusta ja oppimista helpottavat teknologiat; oppimiseen liittyy loppujen lopuksi paljon muutakin. Esimerkiksi teknisetkin asiat tuntuvat vaikuttavan: Koulouri T. ym. (2015) osoittavat, miten syntaktisesti helpommat ohjelmointikielet, kuten Python, ovat parempia opettamaan ohjelmointikonsepteja, kuin syntaksiltaan vaikeammat kielet, kuten Java. He osoittavat myös, että mikäli ennen ohjelmointia oppilaille opetetaan yleisiä ongelmanratkaisutaitoja, heidän suorituksensa paranevat huomattavasti.

Myös opiskelijoiden omien opiskelumetodien on todettu vaikuttavan näiden opintomenestykseen: Willman S. ym. (2015) toteavat, että oppilaiden kurssimenestystä pystyttiin ennustamaan esimerkiksi yleisistä tavoista palauttaa sähköisiä kotitehtäviä. Esimerkiksi parhaimmat tulokset saivat oppilaat, jotka aloittivat ja lopettivat tehtävien teon ajoissa, eivätkä työskennelleet viikonloppuisin eivätkä iltaisin. Tässä tapauksessa tehtävät tehtiin selaimen kautta toimivassa opetusjärjestelmässä, jolloin oppilailla oli mahdollisuus tehdä ja palauttaa tehtäviä koska tahansa.

Oppilaan aikaisemman tietouden ja kokemuksen on myös havaittu vaikuttavan opiskelumenestykseen ohjelmointikurssilla. Etenkin hyvät taidot matematiikassa tuntuvat indikoivan hyviä mahdollisuuksia menestyä myös ohjelmoinnissa (Gomes, Anabela, Carmo ym. 2006). Kuten Gomes A. ym. toteavat, syynä tälle vaikuttaisi olevan matematiikan kautta kehittynyt ja parantunut ongelmanratkaisutaito, joka tuntuu näyttelevän merkittävää osaa ohjelmoinnissa ja sen opettelussa. Myös aiemman ohjelmointikokemuksen on todettu vaikuttavan opiskelumenestykseen, mikä ei ehkä ole kovin yllättävää, mutta huomion arvoista kylläkin: Hagan D. ja Markham S. (2000) osoittivat, että ensimmäisellä ohjelmointikurssilla pärjäsivät huomattavasti paremmin ne oppilaat, joilla oli jo ennestään kokemusta ohjelmoinnista. Kuitenkin, kuten aiemmin

todettiin, kun oppilaiden ryhmät ovat koko ajan heterogeenisempiä, se luo haasteita: kaikille pitäisi pystyä tarjoamaan haastavia ja kehittäviä harjoitteita tasosta riippumatta saman kurssin sisällä.

Yhtenä suurena tekijänä oppimistuloksissa ja oppilaiden menestyksessä on luonnollisesti myös motivaatio; tämä lienee yleistä aiheesta riippumatta. Jos henkilö on motivoitunut oppimaan jotain, häntä ei tarvitse välttämättä edes opettaa erikseen, vaan hän ottaa itse asioista selvää ja harjoittelee itsenäisesti. Kuitenkin etenkin opinnoissa olisi hyvä pystyä sanomaan, onko joku motivoitunut vai ei, jonka jälkeen asialle pystyttäisi toivottavasti tekemään jotain. Motivaatiota on kuitenkin vaikea mitata, joten sen vaikutusta mihinkään on näin ollen myös vaikea mitata. Motivaatiota on olemassa erilaista, niin sisäistä kuin ulkoistakin, joten välillä henkilön itsensäkään ei ole helppo eritellä, mikä häntä loppujen lopuksi motivoi tekemään jotakin. Schunk D. (2012) määrittelee motivaation seuraavasti: “motivation is the process of instigating and sustaining goaldirected behaviour”, eli vapaasti suomennettuna ”motivaatio on prosessi, joka yllyttää ja ylläpitää päämäärähakuista käyttäytymistä”. Tällainen käsitys ihmisen motivaatiosta on kognitivistinen, eli kuten aikaisemminkin on kuvattu, se pohjautuu ajatukseen kognitiivisista prosesseista ja ennen kaikkea ihmisestä itseohjautuvana oppijana. Tietysti motivaationa voi olla esimerkiksi hengissä selviytyminen, jolloin henkilön motivaatio on enemmänkin pakon sanelemaa kuin itse haluttua, mutta tällöinkin ihminen yleensä haluaa pysyä hengissä; jos ei, niin silloin sen eteen ei tehdä mitään. Motivaation voidaan ajatella olevan siis hyvin keskeinen asia kaikenlaisessa tekemisessä. Kuitenkin etenkin oppimisessa sen rooli korostuu; jos ihmistä ei motivoi oppia jotakin, tämä tuskin tekee sen eteen töitä vapaaehtoisesti. Näin ollen myös ohjelmoinnin oppimisessa motivaation rooli korostuu: oppimisen ollessa haastavaa, siihen kuitenkin motivoitunut oppilas haluaa oppia aiheen haastavuudesta huolimatta, tai ehkä jopa sen takia. Oppilaiden motivaatioon on koitettu vaikuttaa jo pitkään, juuri parempien tulosten toivossa; mitään suoraa lääkettä ja metodia motivaation nostamiselle ei kuitenkaan ole. Motivaatiota on kuitenkin pyritty nostamaan usein eri keinoin, kuten erilaisin motivaatiota nostavin opetusmetodein:

esimerkiksi käyttämällä jotain uusia harjoituksia, jotka saisivat oppilaat innostumaan aiheesta, tai esittelemällä sama vanha asia jossain uudessa ja mielenkiintoisessa valossa.

3 OHJELMOINNIN OPETUS

Ohjelmoinnin oppimiseen liittyy olennaisesti, miten sitä opetetaan: oppimisen vaikeutta ei määritä suinkaan vain opeteltavan aiheen vaikeus, vaan myös opetuksen laatu ja siinä käytettävät keinot ovat tärkeässä roolissa. Etenkin ohjelmoinnin opetuksessa on paljon haasteita, eikä kaikkea ole suinkaan vielä ratkaistu, vaan haasteita koitetaan ratkoa jatkuvasti. Esimerkiksi olio-ohjelmoinnin oppimisen vaikeus on ollut tutkimuksen kohteena jo vuodesta 1981, ja jo tuolloin Mayer R. (1981) pohti, miten ideat kognitiivisesta ja opetuksellisesta psykologiasta pystyisivät vastaamaan tähän haasteeseen. Ohjelmointi ja ohjelmoinnin opetus ovat kehittyneet paljon noista päivistä, ja esimerkiksi nykyään on harvemmin ongelmana, ettei oppija ole koskaan aikaisemmin edes käyttänyt tietokonetta. Kuitenkin oppimisen lähestyminen ennemminkin psyykkisten tekijöiden kautta on edelleen oleellinen lähestymistapa: Esimerkiksi Wiedenbeck S. ym. (2004) tarkastelevat, miten minäpystyvyys (self-efficay) ja mentaaliset mallit (mental models) vaikuttavat oppimiseen. He toteavat näillä olevan etenkin yhdessä vaikutus oppilaiden menestykseen ja oppimistuloksiin, ja että opettajien tulisi keskittyä entistä enemmän niiden huomiointiin, sekä lisäämiseen kursseilla. Etenkin mentaalinen malli vaikuttaa suoraan oppilaan menestykseen, mikä on ollut opettajilla myös tiedossa; minäpystyvyys ei kuitenkaan ole näin selkeä eikä tunnettu. Wiedenbeck S. ym. korostavat, miten minäpystyvyyttä, eli oppilaiden omaa kokemusta omista taidoistaan, tulisi huomioida entistä enemmän opetuksessa, jotta oppilas ei tuntisi oloaan kykenemättömäksi ja menettäisi näin kiinnostustaan.

Ohjelmoinnin oppimista helpottamaan on kehitetty useita erilaisia keinoja, joilla esimerkiksi juuri oppilaan minäpystyvyyttä ja suoriutumista yleisestikin pystyttäisiin kohentamaan. Eräs tällainen keino on paljon huomiota saanut ja laajalti käytetty visualisointi, jonka on jo aikaisemminkin todettu olevan erittäin hyödyllinen työkalu etenkin ohjelmoinnin opetuksessa (Kaila, Rajala ym. 2010). Sen avulla pystytään esittämään esimerkiksi erilaisia tietorakenteita lähestyttävämmällä tavalla, tai esittämään koodin suoritusta havainnollistavammin. Ohjelmointia opeteltaessa on tietysti oleellista harjoitella myös itse käytännön ohjelmointia: tämä on tärkeää, jotta

tietoja osattaisi myös soveltaa, eikä ajattelu jäisi pelkästään konseptien tasolle. Butler R. ja Morgan M. (2007) esittävät, miten oppilaat pystyvät kyllä oppimaan konsepteja ja teorioita, mutta ongelmaksi muodostuu useimmille juuri niiden käytännön soveltaminen. Kuten muun muassa Lahtinen E. ym. (2005) toteavat, pelkkä ohjelmointikonseptien tunteminen ei siis itsessään vielä riitä, vaan oppimisen edistämisessä tärkeintä olisi itseasiassa käytännön harjoittelu, jossa opittuja konsepteja päästäisiin soveltamaan käytännön ongelmiin.

Erilaisia opetusmetodeja on useita, kuten parikoodaus (Nosek 1998), käänteinen opetus (Tucker 2012, Flipped Learning Network 2014) ja aktiivinen oppiminen (Bonwell, Eison 1991, Prince 2004). Ne kaikki lähestyvät opetusta hieman eri lähtökohdista ja hieman eri näkökulmalla; jokaisessa on hyvät puolensa, ja myös näiden yhdistelmiä on käytössä. Ennen kuin sukelletaan syvemmälle näiden opetusmetodien pariin, on kuitenkin syytä tarkastella itse ”ohjelmoinnin ydintä” hieman tarkemmin: nimittäin ohjelmointiparadigmoja. Ohjelmointiparadigmat kertovat, miten suoritettava ohjelma mallinnetaan, ja millaisista elementeistä se rakentuu: rakentuuko se esimerkiksi olioista vai funktioista, miten nämä kommunikoivat ja miten tieto elementtien välillä kulkee.

Ensimmäisenä tässä luvussa siis esitellään suosituimmat ohjelmointiparadigmat, ja miten ne nivoutuvat opetukseen: onko jotain vaikeampi opettaa kuin toista, ja mikä olisi etenkin ensimmäiselle ohjelmointikurssille kenties sopivin. Tämän jälkeen tutustutaan erilaisiin opetusmetodeihin, sekä tarkastellaan mitkä niistä ovat ajankohtaisimpia. Lopuksi pyritään luomaan katsaus asioihin, jotka ovat juuri ohjelmoinnin opetuksen kannalta vaikeimpia.

3.1 Suosituimmat ohjelmointiparadigmat ja niiden merkitys opetuksessa

Kun ajatellaan ohjelmointia ja sen opetusta, on ensimmäisenä syytä tarkastella ohjelmointiparadigmoja, joihin ohjelmointi ja ohjelmointikielet perustuvat. Näitä paradigmoja voidaan kutsua ohjelmointikielten ytimeksi: ne määrittelevät, miten niitä toteuttava ohjelmointikieli toimii, eli millä keinoin se edessä olevaa ongelmaa lähtee

ratkaisemaan. Paradigmat ovat siis abstrakti esitys ohjelmoinnin toiminnasta. Esimerkiksi funktionaalinen ohjelmointi on yksi ohjelmointiparadigma, ja se toimii nimensä mukaisesti funktioiden kautta: siinä siis kirjoitetaan erilaisia funktioita jonkin ongelman ratkaisemiseksi, käytännössä jollain tietyllä ohjelmointikielellä (Hudak 2000, Machado 2013). Helposti tulee ajatelleeksi, että koska esimerkiksi Java on olio-ohjelmointikieli, se on näin ollen olio-ohjelmointia. Asia ei kuitenkaan ole aivan näin yksinkertainen, vaan kuten M. Selvakumar (2017) esittää, olio-ohjelmointi on itsessään ohjelmointiparadigma, ja Java toteuttaa useita tällaisia paradigmoja, ei vain olio-ohjelmointia. Java perustuu kyllä olio-ohjelmointiin, mutta on vuosien saatossa ottanut vaikutteita myös muista paradigmoista. Sama pätee nykyään moneen muuhunkin ohjelmointikieleen. Hämmennystä saattaa aiheuttaa myös suomennos ”olio-ohjelmointi”, joka käytännössä tarkoittaa paradigmaa, vaikka siinä onkin ”ohjelmointi”-sana mukana. Englanniksi ero on selvempi: ”object-oriented”, joka vastaa itse paradigmaa, ja ”object-oriented programming”, joka vastaa paradigmaa toteuttavaa ohjelmointia. Ohjelmoinnin ja ohjelmointikielen alla ja perustana on siis aina vielä jokin ohjelmointiparadigma, jota ohjelmoidessa toteutetaan, vaikkei asiaa tulisi sen enempää ajatelleeksikaan.

Ohjelmointiparadigmat ovat tärkeitä opetuksessa juuri niiden abstraktiuden takia: jotta oppilas pystyisi oppimaan ohjelmointia, on hänen ensin hyvä oppia ohjelmoitavan kielen perusperiaatteet, eli millaisia ongelmia jollakin ohjelmointikielellä on paras lähteä ratkaisemaan. Useassa tutkimuksessa on todettu, että pelkästään jonkin ohjelmointikielen opettaminen ei ole tarpeeksi tai edes suotavaa, vaan tulisi ennemminkin opettaa opiskelijat ajattelemaan ohjelmointia abstraktimmalla tasolla (Beheshti, Gunawardena 2001, Samuel 2015, Van Roy, Armstrong ym. 2003). Yksi tärkeä taito on nimittäin myös oikeanlaisen ohjelmointikielen ja -paradigman valitseminen juuri käsillä olevaan ongelmaan (Ortin, Redondo ym. 2016).

Taulukosta 1 huomataan, miten ohjelmointikielet ja -paradigmat suhteutuivat toisiinsa suosion mukaan vuonna 2016. M. Selvakumar (2017) esittää, että nykyään yleisimmät ja laajasti

Taulukko 1 Taulukossa on esitetty suosituimmat ohjelmointikielet vuonna 2016, sekä niiden käyttämät ohjelmointiparadigmat. (M. Selvakumar Samuel 2017)

Numero	Ohjelmointi-kieli	Pääohjelmointiparadigma(t)
1	Java	Olio-ohjelmointi, tapahtumapohjainen GUI:n kanssa, Rinnakkaisuus, Funktionaalinen, Geneerinen, Reflektio
2	C	Imperatiivinen (Proseduraalinen ja Strukturaalinen)
3	C++	Imperatiivinen, Olio-ohjelmointi
4	Python	Imperatiivinen, Olio-ohjelmointi, Funktionaalinen, Tapahtumapohjainen GUI:n kanssa, Rinnakkaisuus, Reflektio, Metaohjelmointi
5	C#	Olio-ohjelmointi, Imperatiivinen, Tapahtumapohjainen GUI:n kanssa, Funktionaalinen, Rinnakkaisuus, Geneerinen, Reflektio
6	R	Funktionaalinen, Olio-ohjelmointi, Tapahtumapohjainen GUI:n kanssa, Imperatiivinen, Reflektio, Array
7	PHP	Imperatiivinen, Olio-ohjelmointi, Tapahtumapohjainen GUI:n kanssa, Funktionaalinen, Reflektio
8	Java Script	Skriptaus
9	Ruby	Funktionaalinen, Olio-ohjelmointi, Imperatiivinen, Tapahtumapohjainen GUI:n kanssa, Reflektio
10	Go	Rinnakkaisuus, Looginen, Funktionaalinen, Olio-ohjelmointi

käytössä olevat ohjelmointiparadigmat ovat funktionaalinen (Functional), imperatiivinen (Imperative), olio-ohjelmointi (Object-Oriented), ja looginen (Logic).

Kyseisten neljän paradigman argumentoidaan olevan myös sekä teollisesti että akateemisesti ne suosituimmat paradigmat. Opetuskäytössä olevien pääparadigmojen M. Selvakumar toteaa kuitenkin olevan hieman erilaiset, vaihtaen loogisen paradigman tapahtumapohjaiseen ohjelmointiparadigmaan graafisen käyttöliittymän kanssa (event-driven with GUI, GUI = Graphical User Interface). Hän esittää myös, että näistä dominoivimpana voidaan pitää olio-ohjelmointia, ja nousevana tulokkaana funktionaalista paradigmaa. Seuraavaksi käydään läpi kyseiset opetuskäytössä suosituimmat paradigmat, esitellään niiden perusideat, sekä millaisiin ongelmiin ne ovat omiaan vastaamaan. Esitetään myös lyhyet koodiesimerkit kustakin paradigmasta, sekä sitä toteuttavasta ohjelmointikielestä.

Kuten edellä jo hieman vihjattiin, funktionaalinen ohjelmointiparadigma perustuu siis funktioihin; funktiot puolestaan ovat käytännössä laskulausekkeiden määritelmiä. Tätä paradigmaa toteuttava ohjelma koostuu siis laskulausekkeista, jotka suoritettava ohjelma sitten laskee kirjoitetussa järjestyksessä. Pääohjelma on itsessään vain yksi funktio, joka vain koostuu useammasta laskelmasta. Ohjelmoijan tarvitsee keskittyä vain tämän pääohjelman luomiseen, ohjelma jakaa itse funktion pienempiin osiin. (M. Selvakumar Samuel 2017) Funktionaalinen ohjelmointi perustuu alkujaan lambdakalkyyliin, jossa matemaattisia funktioita käytetään ohjelman rakennuspalikoina (Machado 2013, M. Selvakumar Samuel 2017). Koodiesimerkissä 1 esitetään, miten yksinkertainen ongelman ratkaistaan käyttäen funktionaalista ohjelmointiparadigmaa sekä sitä toteuttavaa ohjelmointikieltä C:tä.

Koodiesimerkki 1: Esimerkissä on esitetty Fibonaccin lukujonon muodostus käyttäen C-ohjelmointikieliä. C toteuttaa funktionaalista paradigmaa, ja se hoitaa ongelman kertomalla mitä ohjelman tulee käytännössä suorittaa.

```
int fib (int n) {
     if (n < 2)
         return n;
     else
          return fib (n-1) + fib (n-2);
}
```

Imperatiivinen ohjelmointiparadigma perustuu puolestaan siihen, että ohjelmoija määrittelee, mitä haluaa tehdä ja miten ohjelman tulisi tähän lopputulokseen päästä. Kuten sen nimikin jälleen antaa vihiä, imperatiivinen ohjelmointiparadigma perustuu siis erilaisiin käskyihin ja määräyksiin, jotka annetaan tietokoneelle suoritettavaksi. Imperatiivinen ohjelmointi tunnetaan myös nimellä algoritminen ohjelmointi, ja sen tunnettuja johdannaisia ovat proseduraalinen ohjelmointi sekä strukturoitu ohjelmointi. (M. Selvakumar Samuel 2017) Näistä etenkin proseduraalinen ohjelmointi on yleisesti käytössä, ja se sekoitetaankin välillä imperatiiviseen ohjelmointiin, vaikka ne eivät aivan sama asia olekaan: proseduraalinen ohjelmointi perustuu erilaisiin pieniin kokonaisuuksiin, proseduureihin, joiden käyttö imperatiivisessa ohjelmoinnissa ei ole kuitenkaan välttämätöntä. Koodiesimerkki 2 näyttää, miten yksinkertainen ongelma ratkaistaan käyttämällä imperatiivista ohjelmointiparadigmaa ja sitä toteuttavaa ohjelmointikieltä, Haskellia.

Koodiesimerkki 2: Käytetään samaa esimerkkitilannetta kuin ylempänä, eli Fibonaccin lukujonon muodostusta: Nyt kuitenkin käytetään Haskell-ohjelmointikieltä, joka toteuttaa puolestaan imperatiivista paradigmaa. Ohjelman tehtävänä on siis tällä kertaa kertoa miten ohjelman tulisi suorittaa operaatioita, jotta ongelma ratkeaisi.

```
fib n =
     if (n < 2) then n else fib (n-1) + fib (n-2)
```

Olio-ohjelmointi perustuu, jälleen nimensä mukaisesti, olioihin: nämä oliot ovat käytännössä erilaisia luokkia ja objekteja, jotka sisältävät dataa ja operaatioita. Näiden objektien keskinäinen kommunikointi muodostaa lopulta itse olio-ohjelmointia toteuttavan sovelluksen. Olio-ohjelmoinnin suurta suosiota voi selittää esimerkiksi juuri näiden objektien välinen kommunikointi, ja sen läheisyys reaalimaailmaan, ainakin verrattuna muihin paradigmoihin. Esimerkiksi olio-ohjelmoinnissa reaalimaailman objektien muodoista voidaan keskustella sovelluksen sisällä käyttäen ”oikeita” nimiä, kuten oikea tai vasen sivu, eikä pelkästään näiden numeerisia arvoja, kuten sivun pituuksia (Hofstedt 2011). Olio-ohjelmointiparadigmalla pystytään siis ratkaisemaan hyvin myös monimutkaisia ongelmia (M. Selvakumar Samuel 2017). On myös hyvä

muistaa, että vaikka jokin ohjelmointikieli perustuisikin olio-ohjelmointiin, etenkin nykyään se sisältää myös mahdollisuuksia toteuttaa muitakin paradigmoja. Koodiesimerkistä 3 näkyy, miten olio-ohjelmointia ja sitä käyttävää ohjelmointikieltä Javaa käytetään yksinkertaisen ongelman ratkaisemiseen. Ongelma on sama kuin aikaisemminkin, eli miten Fibonaccin lukujono rakennetaan rekursiota käyttäen.

Koodiesimerkki 3: Esimerkki näyttää, miten Fibonaccin lukujono rakentuu olio-ohjelmointia ja Javaa käyttäen. Esimerkistä huomaa hyvin, miten koodi perustuu luokkiin ja objekteihin: itse lukujonon laskeva osuus tehdään erillään, ja myöhemmin sitä vain pyydetään suorittamaan laskenta jollain arvolla. (SSS IT Pvt Ltd 2018)

```
class FibonacciExample2{
 static int n1=0,n2=1,n3=0;
 static void printFibonacci(int count){
   if(count>0){
       n3 = n1 + n2;
       n1 = n2;
       n2 = n3;
       System.out.print(" "+n3);
       printFibonacci(count-1);
    }
 }
 public static void main(String args[]){
  int count=10;
  System.out.print(n1+" "+n2);//printing 0 and 1
  printFibonacci(count-2);//n-2 because 2 numbers are already printed
 }
}
```

Tapahtumapohjaisessa ohjelmointiparadigmassa graafisen käyttöliittymän kanssa luonnollisesti ohjelmointiparadigma ja graafinen käyttöliittymä (GUI) ovat syvästi linkittyneet toisiinsa: siinä missä paradigma käyttää erilaisia graafisen käyttöliittymän tapahtumia osana tapahtumaketjuaan, myös graafiset käyttöliittymät perustuvat tapahtumapohjaiselle paradigmalle (M. Selvakumar Samuel 2017). Paradigma siis pohjautuu erilaisille tapahtumille ja niiden interaktioille keskenään, sekä interaktioihin

mahdollisesti jonkun ulkoisen tapahtuman kanssa. Tapahtumapohjaista paradigmaa graafisen käyttöliittymän kanssa käyttävät nykyään hyvin moni sovellus, ja näin ollen myös ohjelmointikieli. Kuten ylempää löytyvästä taulukosta 1 nähdään, kuusi kymmenestä suosituimmasta ohjelmointikielestä käyttää tätä paradigmaa ainakin jollain tasolla. Koodiesimerkissä 4 demonstroidaan, miten tapahtumapohjainen paradigma toimii. Esimerkki on esitetty pseudokoodilla sen yleisyyden takia useissa ohjelmointikielissä, jolloin paradigman tapahtumalähtöisyys hahmottuu parhaiten (Ferg 2006).

Koodiesimerkki 4: Esimerkissä näkyy hyvin, miten tapahtumapohjainen paradigma toimii tapahtumien (event) kautta, ja miten se odottaa interaktiota graafisen käyttöliittymän kanssa ennen kuin se aloittaa minkään tapahtuman suorittamisen. (Ferg 2006).

```
do forever:    # the event loop

   get an event from the input stream

   if   event type == EndOfEventStream :
        quit # break out of event loop

   if   event type == ... :
        call the appropriate handler subroutine,
        passing it event information as an argument

   elif event type == ... :
        call the appropriate handler subroutine,
        passing it event information as an argument

   else:   # handle an unrecognized type of event
        ignore the event, or raise an exception
```

Mikä on siis paradigmojen vaikutus ja rooli ohjelmoinnin opetuksessa? Paradigmathan ovat ikään kuin yleistys ohjelmoinnista, joten voisi olla loogista päätellä niiden olevan merkittävässä roolissa ohjelmoinnin opetuksessa. Useat tutkimukset toteavatkin, että ohjelmointiparadigmoja tulisi opettaa ja että erityisesti niihin tulisi myös fokusoida, sillä niiden opettelu on tärkeää ohjelmoinnin oppimista ajatellen (Beheshti, Gunawardena 2001, Samuel 2015, Van Roy, Armstrong ym. 2003). Muun muassa Selvakumarin (2017) tutkimuksessa, jossa ohjelmointia opetetaan ensimmäisellä ohjelmointikurssilla juuri paradigmoihin keskittyen, tällainen lähestymistapa otettiin positiivisesti vastaan opiskelijoiden kesken: lähes kaikki ymmärsivät kurssin jälkeen ohjelmointikieltä, arkkitehtuuria ja ohjelmoinnin rakennetta paremmin kuin aiemmin. Paradigmoihin keskittyvä lähestymistapa tarkoittaa tässä kohtaa siis sitä, että ongelmaan mietitään ensin ratkaisua paradigmojen kautta sen sijaan, että aloitettaisiin suoraan ohjelmoimaan. Yleinen kanta kuitenkin tuntuu olevan, ettei paradigmoja tarvitse erikseen opettaa, vaan että ne ovat luonnollisesti mukana ohjelmointikursseilla: paradigmojen ajatellaan sisältyvän käytännön ohjelmointiin sen verran vahvasti, ettei edelläkin mainitulle paradigmoihin keskittyvälle lähestymistavalle nähdä tarvetta. Tämä näkyy esimerkiksi erilaisista tutkimuksista ja artikkeleista, joissa paradigmojen ja ohjelmoinnin opetus kulkevat käsi kädessä ilman, että niiden opettelua on nähty tarpeelliseksi erottaa. Paradigmoista kyllä puhutaan ja niiden merkitys ymmärretään, mutta niiden erillistä opettelua ennen käytännön ohjelmointia ei nähdäkseni myöskään ole tarkasteltu (Goosen, Mentz ym. June 20, 2007, Pears, Seidman ym. 2007, Weir, Vilner ym. 2005, Oliveira Aureliano 2013).

Paradigmat ovat siis luonnollisesti tärkeä osa ohjelmoinnin opetusta ja oppimista, ja niiden tärkeys myös ymmärretään. On kuitenkin vielä hieman tulkinnanvaraista, tulisiko paradigmoja opettaa vielä enemmän, kuten esimerkiksi aluksi keskittyä vain paradigmojen opettamiseen, ennen kuin siirrytään käytännön ohjelmoinnin pariin. On kuitenkin todettu, että paradigmoja opitaan myös käytännön ohjelmoinnin kautta, joka voikin olla ohjelmointitaidon kannalta jopa parempi. Jos kuitenkin opiskelijan tarkoitus olisi tulevaisuudessa keskittyä uusien algoritmien ja etenkin abstraktien ratkaisujen

kehittämiseen, voisi olla perusteltua opettaa algoritmeja ainakin keskimääräistä enemmän.

Tällainen paradigmoihin keskittyvä lähestymistapa voi olla vaarassa, mikäli opetuksessa keskitytään vain yhteen ohjelmointikieleen ja -paradigmaan. Esimerkiksi Java on yleinen opetukseen valittu ohjelmointikieli (Pears, Seidman ym. 2007, M. Selvakumar Samuel 2017). Se on pohjimmiltaan olio-ohjelmointia toteuttava ohjelmointikieli, joten tätä myös opetellaan Javaa käytettäessä. Vaikka onkin argumentoitu, että olio-ohjelmointi olisi ensisijaisesti opetettava ohjelmointiparadigma esimerkiksi sen korkean abstraktiotason takia (Van Roy, Armstrong ym. 2003) ja koska se on niin yleisesti käytössä teollisuudessa (Pears, Seidman ym. 2007, M. Selvakumar Samuel 2017), muitakin paradigmoja tulisi opettaa. Java kyllä taipuu useampaankin paradigmaan, kuten esimerkiksi taulukosta 1 nähtiin, joten sillä pystyy muitakin paradigmoja opiskelemaan ja opettamaan. Opiskeltavien paradigmojen moninaisuudesta tulisi kuitenkin pitää huolta, sillä vaikka nykyään olio-ohjelmointi onkin suuressa roolissa, kaikkiin ongelmiin se ei ole paras mahdollinen. Opetuksessa tulisi myös tukea laaja-alaista oppimista, jolloin oppilas oppii myös kriittistä ajattelua ja esimerkiksi itse tunnistamaan oikeanlaiset työkalut erilaisten ongelmien ratkaisemiseen, sen sijaan että aina olisi jokin valmis ratkaisu tai vaihtoehto.

3.2 Erilaiset ohjelmointiopetuksessa käytetyt opetusmetodit

Opetuksen kohteen lisäksi on syytä miettiä myös millä tavoin opetus olisi parasta toteuttaa. Opetuksessa yleisesti hyväksi havaitut menetelmät toimivat varmasti suurimmaksi osaksi myös ohjelmoinnin opetuksessa, mutta joitain ominaispiirteitäkin löytyy. Esimerkiksi visualisointi on hyvä tapa helpottaa ja auttaa oppilaita ymmärtämään ja soveltamaan algoritmeja tai koodin pätkiä, jotka muuten voivat olla oppilaille liian haastavia, etenkin jos oppilaalla ei ole aikaisempaa ohjelmointikokemusta (Rajala, Laakso ym. 2008, Lahtinen, Ala-Mutka ym. 2005). Visualisoinnista on varmasti hyötyä myös muilla aihealueilla, mutta ohjelmoinnin opetuksessa ja oppimisessa sen merkitys korostuu. Kuten esimerkiksi Rajala T. ym. (2008) osoittavat, visualisoinnin lisääminen ohjelmoinnin opetuksessa on todettu

auttavan oppilaita ymmärtämään ohjelmia ja ohjelmointikonsepteja paremmin. Visualisointi on myös ollut käytössä jo jonkun aikaa: esimerkiksi Brown M. ja Sedgewick R. (1984) käsittelevät artikkelissaan ensimmäisiä animoituja algoritmeja, jotka käytännössä vastaavat visualisoinnin periaatteita, ja joita pystyttiin käyttämään esimerkiksi juuri opetuksessa. Visualisointi ei siis ole enää mikään kovin uusi asia ohjelmoinnin opetuksenkaan saralla. Sen hyödyllisyys ei kuitenkaan ole hävinnyt, vaan se onkin yksi ohjelmoinnin opetuksen keskeisistä työkaluista.

Visualisointia voidaan kuitenkin pitää ennemmin työkaluna kuin varsinaisena opetusmetodina, vaikka se näytteleekin suurta osaa ohjelmoinnin opetuksessa. Ohjelmoinnin opetuksessa käytetään yleisestikin tunnettuja opetusmenetelmiä, kuten käännetty opetus (flipped classroom) ja aktiivinen oppiminen (active learning), mutta myös enemmän ohjelmoinnille ominaisia menetelmiä, kuten pariohjelmointia (pair programming). Seuraavaksi esitellään muutama suurempi ja yleisesti suositumpi opetusmenetelmä, kuten juuri käännetty opetus sekä pariohjelmointi.

3.2.1 Käänteinen opetus

Ensimmäisenä käsittelyssä on käänteinen opetus (flipped classroom), joka on noussut suuren suosioon yleisestikin opetuksessa viimeisen noin kymmenen vuoden aikana (Toivola Marika a 2018). Käänteisen opetuksen rinnalla on myös toinen lähes samanlainen teoria/metodi, käänteinen oppiminen (flipped learning). Käänteisen opetuksen ja käänteisen oppimisen välillä on kuitenkin merkittävä ero, josta niiden nimetkin jo hieman antavat vihiä: Käänteinen opetus keskittyy spesifisti oppimisprosessin muuttamiseen, ja sen myötä oppimisen parantamiseen. Käänteinen oppiminen on puolestaan enemmän kokonaisvaltainen muutos opetuksessa, jossa opettajan rooli on totuttaa oppilaat omaehtoiseen ja oma-aloitteiseen oppimiseen ja tukea oppilaan valinnanvapautta myös pedagogisessa mielessä (Toivola Marika b 2018).

Ohjelmoinnin opetuksessa ei ole taidettu vielä päästä käänteisen oppimisen tasolle asti, spekuloiden esimerkiksi sen takia, ettei ohjelmointi ole kovin suuressa roolissa esimerkiksi perusopetuksessa. Käänteinen opetus lienee siis se metodi, jota ohjelmointiopetuksessa lähinnä käytetään. Käänteinen opetus perustuu suurelta osin

perinteisen ”opetussyklin” kääntämiseen: siinä varsinainen aiheen opiskelu tehdään itsenäisesti kotona, jolla valmistaudutaan seuraavaan lähiopetuskertaan, jossa sitten tehdään aiheeseen liittyvät ”kotitehtävät”. Näin lähiopetuskerralla pystytään käsittelemään ja pohtimaan käsillä olevaa aihetta soveltavasti, kun alustava opettelu on tehty jo kotona. Tällä tavoin myös opettajan rooli ohjaajana ja auttajana pääsee paremmin oikeuksiinsa: opettaja pystyy auttamaan ja havaitsemaan oppilaille haasteellisia asioita helpommin, kun hän pääsee osaksi oppilaan soveltamisprosessia (Tucker 2012). Käänteisessä opetuksessa korostuu lisäksi myös ryhmätyöskentely: lähiopetuskerroilla tehtäviä suoritetaan paljon ryhmissä, jolloin oppilaat saavat tukea toisiltaan ja pystyvät yhdessä miettimään ratkaisuja ongelmiin.

Käänteistä opetusta käytetään myös esimerkiksi Turun yliopistossa: Ohjelmointiharjoituksia suoritetaan ryhmissä, ja vapaa keskustelu tehtävistä on sallittua ja jopa kannustettavaa. Myös harjoituksissa tarvittavat materiaalit on saatu ennen ohjelmointiharjoitusten suorittamista, kuten harjoituskertaa edeltävällä luennolla. Opiskelijoilla on lisäksi vapaa pääsy netissä oleviin materiaaleihin ympäri vuorokauden, joissa käytännössä on luennoilla läpikäydyt asiat. Omassa yliopistossani tutkimuksia ei ole tehty niinkään käänteisen opetuksen käytöstä tai sen vaikutuksista, vaan tutkimukset keskittyvät spesifimpiin kysymyksiin, kuten millainen vaikutus visualisoinnilla tai luentoläsnäoloilla on oppilaiden suoriutumiseen. Kuitenkin esimerkiksi Amresh A. ym. (2013) ovat tutkineet käännetyn opetuksen vaikutuksia ensimmäiseen ohjelmointikurssiin. He huomasivat, että käänteisessä opetuksessa on potentiaalia ensimmäisiä ohjelmointikursseja opettaessa, ja sen käyttöönotto paransikin heidän tutkimuksissaan oppilaiden tuloksia. Wiedenbeck ym. (2004) esittävät, että myös oppilaiden minä-pystyvyys tuntui parantuvan käänteisen opetuksen käyttöönoton myötä: hyvän minä-pystyvyyden on taas puolestaan jo aikaisemmin todettu vaikuttavan positiivisesti oppilaan menestykseen. Amresh A. ym. esittävät myös 13 kohdan listan, jossa eritellään, mitä hyötyjä käänteisestä opetuksesta on. Käänteisen opetuksen tuomista hyödyistä huolimatta sen toteutuksessa ja käyttöönotossa on silti haasteita: Samaisessa artikkelissa tuotiin esiin myös, miten oppilaat voivat kokea ikävänä ja/tai uutena asiana opiskelun käänteisyyden, jolloin kotona pitäisikin tehdä

suurin opiskelutyö koulun sijaan. Haasteena mainittiin myös opettajien puolelta oheismateriaalin, esimerkiksi videoiden, oikeanlainen luominen siten, että niiden katselu kotona olisi oppilaista mukavaa ja motivoivaa, ja että ne linkittyisivät seuraavaan lähiopetustuntiin. Opetusmetodina käännetty opetus voi siis olla jopa raskaampi opettajalle kuin perinteinen, mutta tutkimusten valossa varmasti myös palkitsevampi, ja enemmän opettajaa itseäänkin osallistava.

3.2.2 Aktiivinen oppiminen

Aktiivinen oppiminen ei välttämättä nimeltään kuulosta niinkään opetusmetodilta; siinähän puhutaan oppimisesta, ei opetuksesta. Termin voisi periaatteessa kääntää myös aktiiviseksi opetukseksi, mutta nimi voisi jälleen olla hieman hämäävä. Aktiivisen oppimisen ytimessä nimittäin on oppilaiden erilainen aktiivinen osallistuminen opetukseen: tarkoituksena on siis korostaa oppilaan aktiivista ja osallistuvaa roolia, ennemmin kuin opettajan aktiivista roolia. Opettajan tehtävänä on luonnollisesti luoda tämän mahdollistava ympäristö, joten opettajan roolia ei sovi myöskään vähätellä. Käytännössä aktiivinen oppiminen käsittää siis sekä oppilaan aktiivisen roolin, että myös opettajan roolin luoda aktiivisen oppimisen mahdollistava ympäristö.

Aktiivinen oppiminen on ideana verrattain vanha, ja esimerkiksi sen puuttumista ja vähäistä käyttöönottointoa yliopistoissa on kritisoitu jo 1990-luvun alussa (Bonwell, Eison 1991). Kuten mainittu, aktiivisessa oppimisessa on ideana, että oppilas saadaan osallistumaan oppimistapahtumaan aktiivisesti, eikä oppiminen tapahdu vain kuuntelemalla, kuten tavallisessa opetuksessa on enemmän tapana. Oppilaan tulisi esimerkiksi lukea, kirjoittaa, keskustella tai ratkoa ongelmia opetettavasta aiheesta. Ennen kaikkea olisi tärkeää saada oppilas aktivoitua opetukseen niin sanotusti korkeammalla ajattelun tasolla, kuten opetettavaa aihetta analysoimalla, asioita yhdistämällä, ja arvioimalla. Korostetaan siis oppilasta aktiivisena tekijänä oppimisprosessissa pelkän passiivisen oppijan sijaan.

Bonwell C. ja Eison J. (1991) tiivistävät, miten aktiivinen oppiminen johtaa parempiin asenteisiin oppilailla, sekä parantuneisiin ajattelu- ja kirjoitustaitoihin. Prince M. (2004) argumentoi, ettei aktiivista oppimista tulisi kuitenkaan ajatella vain yhtenä objektina,

jonka toteuttamalla päästään parempiin oppimistuloksiin. Vaikka tämä ehkä onkin sanomattakin selvää, Prince M. erottelee aktiivisesta oppimisesta useampia kategorioita ja erillisiä toteuttamismetodeja, jotka ovat lähempänä itse aktiivisen oppimisen käytäntöä. Näitä kategorioita voi pitää myös erillisinä opetusmetodeina, mutta yhteistä niillä kaikilla on aktiivinen oppiminen, kun esimerkiksi yhteisöllisessä oppimisessa (collaborative learning) fokus on aktiivisen oppimisen lisäksi etenkin ryhmissä oppimisessa ja työskentelyssä. Muita Prince M. esittämiä metodeja ovat yhteistoiminnallinen oppiminen (coopertavie learning) ja ongelmalähtöinen oppiminen (problem-based learning). Vaikka hän käsitteleekin näitä kolmea erillään aktiivisesta oppimisesta, on kaikkien pohja kuitenkinkin sama: oppilaiden aktivointi oppiaineeseen opetustilanteessa.

Kuten ehkä pystyy jo ajattelemaankin, juuri korkeamman tason kokonaisuuksien hahmottaminen ja oppilaan aktiivinen rooli opetuksessa ovat hyödyksi erityisesti ohjelmointia opiskeltaessa. Myös tietojenkäsittelyn saralla onkin siis tehty aktiivisen oppimisen perusteella niin tutkimusta, käytännön sovelluksia kuin käyttöönottojakin. Etenkin tietojenkäsittelytieteen opiskelijoille aktiivista oppimista toteuttaessa oikeanlainen ympäristö on todettu tärkeäksi asiaksi: varsinkin tilan interaktiivisuus ja ryhmätyömahdollisuuksien tukeminen on noussut tärkeiksi huomion kohteiksi (Hakimzadeh, Adaikkalavan ym. 2011). Myös erilaisten oppimistyökalujen käyttö, kuten esimerkiksi erilaiset tietokilpailut, on todettu olevan tärkeitä tekijöitä (Shambhavi 2017).

Powers K. (2004) kuvaa, miten ensimmäisen ohjelmointikurssin opiskelijoille pystytään opettamaan tehokkaasti esimerkiksi keskusyksikön (CPU) toimintaa erityisesti varten räätälöidyllä aktiivisen oppimisen ratkaisulla. Hän korostaa etenkin tietojenkäsittelytieteiden alkuvaiheen opintojen tärkeyttä, ja sitä miten oppilaiden oppimistuloksia pystytään parantamaan esimerkiksi juuri aktiivisen oppimisen avulla. Myös esimerkiksi visualisointi auttaa huomattavasti aktiivisessa oppimisessa (Schweitzer, Brown 2007). Schweitzer D. ja Brown W. osoittavat, että etenkin aktiivisen oppimisen ja visualisoinnin yhdistäminen tuottaa hyviä oppimistuloksia, sekä myös

hyvää palautetta opiskelijoilta. Erilaisia algoritmien ja ohjelmien visualisointeja voidaankin periaatteessa pitää ohjelmointiopetukselle leimallisena aktiivisen oppimisen merkkinä; visualisoinnilla pystyy kyllä demonstroimaan asioita, mutta sen myötä saatavat edut korostuvat etenkin aktiivisen oppimisen yhteydessä. Kuten on jo myös aikaisemmin todettu, visualisointi itsessäänkin on jo tehokas oppimisen väline (Rajala, Laakso ym. 2008, Schweitzer, Brown 2007).

3.2.3 Pariohjelmointi

Pariohjelmointi on erityisesti ohjelmointiin vahvasti liittyvä opetusmetodi. Pariohjelmoinnin perusidea on yksinkertainen, ja myös aika ilmeinen: tarkoituksena on ohjelmoida tai harjoitella ohjelmointia pareina. Pariohjelmointi on paitsi opetusmenetelmä, myös teollisuudessa käytetty ohjelmointityyli. Pariohjelmointi myös valmistaa opiskelijoita työelämään, harjoittaen esimerkiksi yhteistyö- ja kommunikointitaitoja ohjelmointitaitojen ohella. Pariohjelmoinnissa saavutetut hyödyt ovat siis ilmeiset, ja ne on myös ymmärretty jo jokin aika sitten: jo ennen vuosituhannen vaihdetta niin sanottu ryhmäohjelmointi (team programming, collaborative programming), joka jo tuolloin tarkoitti käytännössä pareittain tapahtuvaa ohjelmointia, todettiin toimivaksi konseptiksi. (Nosek 1998)

Nosek J. (1998) toteaa, että vastoin silloisia oletuksia ohjelmoijapari suoriutui ohjelmointitehtävästä paremmin, nautti enemmän työstään, ja heidän luottamuksensa ratkaisuunsa oli suurempi, kuin jos saman tehtävän olisi suorittanut yksi ohjelmoija. Myös hieman myöhemmässä tutkimuksessa Williams L. ym. (2000) todistivat, että tällöin jo pariohjelmoinniksi kutsuttua ohjelmoijien yhteistyötä oli kannattavaa harrastaa yritysmaailmassa muun muassa juuri ohjelmoijien paremman tyytyväisyyden ja tuotteelle saatavan aikaisemman julkaisuajankohdan takia. Tutkimuksia on tehty myös pariohjelmoinnin käytöstä opetuksessa, ja tulokset ovatkin olleet positiivisia opetuskäyttöä ajatellen (Wood, Parsons ym. 2013, McDowell, Hanks ym. 2003). Löydettyjä hyviä puolia ovat esimerkiksi suuremmat läpäisyprosentit oppilailla, oppilaiden parantunut tyytyväisyys ja asenne kurssiin ja ohjelmointiin, sekä myös oppilaiden kirjoittamat laadukkaammat ohjelmat.

Wood ja Parsons ym. (2013) puhuvat pariohjelmoinnin käytön puolesta etenkin ensimmäisellä ohjelmointikurssilla ja heidän mukaansa pariohjelmoinnin käyttö etenkin ensimmäisen ohjelmointikurssin ensimmäisillä kuudella viikolla tuottaa merkittäviä positiivisia vaikutuksia. Heidän mukaansa opiskelijat pääsevät näin paremmin sisään ohjelmointiin, ja heidän itsevarmuutensa kasvaa. Myös opettajilta kysyttiin mielipiteitä pariohjelmoinnin onnistumisesta, ja he osasivatkin nimetä useita hyötyjä, joita ei opiskelijadatasta välttämättä pysty erottamaan: Esimerkiksi oppilaiden odotusaika opettajan apuun pienentyi, johtuen vähentyneestä neuvottavien määrästä. Osasyyksi tähän arvellaan myös sitä, että pareittain opiskelijat pystyvät ratkomaan joitain ongelmia, joihin he muuten tarvitsisivat opettajan apua. Opettajat huomasivat myös oppilaiden kokeilevan haastavampia harjoituksia kuin aikaisemmin ennen pariohjelmointia: tämän arveltiin johtuvan esimerkiksi nousseesta itsevarmuudesta. Myös pariohjelmoinnista aiheutuvasta sosiaalisesta kanssakäymisestä muiden kurssilaisten kanssa nähtiin olevan hyötyä, suorasti esimerkiksi muiden auttamisen yleistymisenä sekä epäsuorasti itse pariohjelmoinnin helpottumisena, kun ”sosiaalinen muuri” kommunikoimiseen laski.

Pariohjelmoinnin hyödyt eivät siis rajoitu ainoastaan oppilaihin. Myös Williams L. (2007) havaitsi pariohjelmoinnin käyttöönoton hyödyttävän oppilaiden lisäksi opettajia. Opettajien näkökulmasta hyötyjä olivat muun muassa arvosteltavien suoritusten pienentynyt määrä, huijaamisen väheneminen parirakenteesta aiheutuvan luontaisen tuen saannin takia, sekä myös pienentynyt tarve auttaa pienimmissäkin ongelmatilanteissa. Opiskelijat myös sanoivat nauttivansa pariohjelmoinnissa korostuvasta yhteistyöstä, ja tästä johtuen suhtautuivat myös kurssiin positiivisemmin. Mainittavan arvoista on myöskin niin sanottujen ”ongelmaoppilaiden” vähentyminen, jonka Williams L. ajatteli johtuvan pariohjelmoinnin ryhmäpaineesta, jolloin opiskelija suoriutuu paremmin/tekee enemmän jo pelkästään sen takia, että pelkää vaikuttavansa negatiivisesti parinsa työhön ja arviointiin.

Jotta pariohjelmointi onnistuisi parhaiten ja olisi tehokkainta, tulee parien muodostuksen tuottaa hyviä pareja: pareja, jotka saavat aikaan hyviä tuloksia, eivätkä

siipeile toisen osaamisella. Tähän on kokeiltu erilaisia menetelmiä: esimerkiksi Williams L. (2007) käytti parien löytämiseen useampia erilaisia metodeja, pääasiassa perustuen erilaisiin pistemittareihin. Näillä mittareilla pyrittiin kartoittamaan esimerkiksi opiskelijan keskimääräisiä pisteitä tietojenkäsittelytieteiden opinnoissa tai mittamaan opiskelijan arviota itsestään. Tämän jälkeen yhdistettiin parit siten, että oppilaat, joilla on suunnilleen samat pistemäärät, päätyvät pariksi. Vaikka suurella osalla Williams L. käyttämillä mittareilla ei päästykään toivottuihin tuloksiin, esimerkiksi lukuvuoden puolivälissä mitatuilla pisteillä saatiin parit muodostettua kuitenkin hyvin tuloksin. Toimivaksi konseptiksi todettiin loppujen lopuksi se, että opiskelijat jaettiin mahdollisimman tasavertaisiin pareihin.

Myös Wood ja Parsons ym. (2013) käyttivät tasavertaisuutta parien muodostamisen perusteena. He kuitenkin käyttivät opiskelijoiden arviointiin hieman eri menetelmiä: ottaen huomioon ohjelmointiopetuksen alun tärkeyden, he koittivat tehdä hyvän arvion oppilaista jo kurssin alkupuolella, heti kahden viikon opiskelun jälkeen. Tämä toteutettiin käytännössä käyttäen opettajien valistuneita arvioita, ja Wood ja Parsons ym. kutsuvat näitä arviointiperusteita ”ohjelmointi-itsevarmuudeksi” (programming confidence): ne perustuvat kokeneiden opettajien arvioihin ja havaintoihin siitä, miten opiskelija suhtautuu ohjelmointitehtäviin. Ohjelmointi-itsevarmuus ei siis suoraan ole yhteydessä opiskelijan omaan itsevarmuuteen, vaan heijastaa pikemminkin oppilaan asennetta: jos opiskelija tarttuu tehtävään innolla ja kokeilee paljon erilaisia ratkaisuvaihtoehtoja oma-aloitteisesti, tämä saa korkeat pisteet, kun taas pieniin ongelmiin juuttuva eikä kovin ratkaisukeskeinen opiskelija saa pienemmät pisteet. Tällaisen pisteytyksen perusteella muodostetut tasapisteiset parit tuntuivat suoriutuvan hyvin. Jälleen siis tasavertaiset parit huomattiin toimivaksi yhdistelmäksi, ja jopa opettajien oma arviointi opiskelijoista ilman mitään varsinaista mittausjärjestelmää osoittautui toimivaksi.

Omanlaisensa lähestymistavan parinmuodostukseen kehittivät myös Radermacher A. ja Walia G. (2011), joka ei kuitenkaan osoittautunut niin tehokkaaksi kuin kaksi edellistä. He käyttivät parijaon perustana oppilaiden pääaineita: parit jaettiin siten, että parin

toinen puolisko oli tietojenkäsittelytieteitä opiskeleva, ja toinen oli jostain muusta pääaineesta. He pohjasivat kokeilun ajattelulle, että tällainen olisi tehokkainta oppimisen kannalta sekä miellyttävää oppilaille. Kuten sanottu, kyseinen lähestymistapa kuitenkin osoittautui vähemmän suotuisaksi kuin muut menetelmät, kuten esimerkiksi erilaisiin suorituspistemääriin perustuva parien jako. Tutkimuksessaan Radermacher A. ja Walia G. myös huomauttavat, ettei heidän tutkimustuloksistaan tulisi vetää johtopäätöstä, että opiskelijoiden pääaine olisi päätekijänä parien suoriutumisessa, vaan aihe vaatisi vielä lisätutkimusta. Voidaan kuitenkin todeta, että parinmuodostus tulisi suorittaa ennemmin kahdessa ensimmäisessä esimerkissä esitetyin tavoin, kuin viimeisen esimerkin pääainepohjaisen valinnan kautta.

3.2.4 Erilaisten opetusmetodien käyttö yleisesti

Yhteistyötä korostavat opetusmetodit tuntuvat olevan nosteessa tällä hetkellä, mutta ne eivät ole sitä aivan turhaan: Vihavainen A. ym. (2014) osoittavat, että tällaisilla metodeilla saadaan parhaimpia tuloksia verrattuna muihin laajasti käytössä oleviin opetusmetodeihin. Edellä esitellyissä opetusmetodeissa, käännetyssä oppimisessa, aktiivisessa oppimisessa sekä pariohjelmoinnissa, korostuu ainakin jollain tasolla yhteistyö. On kuitenkin olemassa myös muita opetusmetodeja, joilla on todettu olevan tuloksia parantava vaikutus verrattuna alkuperäisiin luentopohjaisiin kursseihin. Vihavainen A. ym. osoittavat viisi opetuksellista lähestymistapaa, joilla ohjelmointikurssien läpipääsyprosenttia on onnistuttu parantamaan. Näistä parhaimpia tuloksia antoi ”yhteistyö ja vertaistuki”-kategoria, joka sisältää muun muassa edellä esitetyn pariohjelmoinnin sekä samoja ajatuksia kuin aktiivisessa oppimisessa.

Metodeja on toki monia, ja muita Vihavainen A. ym. (2014) esittämiä opetusmetodeja olivat esimerkiksi alustavan kurssin (CS0-kurssi) toteuttaminen ennen varsinaista ohjelmointikurssia, sekä visuaalisten ohjelmointikielten käyttö. Vihavainen A. ym. kuitenkin mainitsevat, että erot eri opetusmetodien välillä olivat lähinnä marginaalisia. Opetusmetodeilla voi siis olla huomattaviakin vaikutuksia oppimistuloksiin, riippumatta kuitenkaan niinkään paljon siitä, millaisia metodeja käytetään. Kuten

Vihanainen A. ym. korostaa, tietoisesta muutoksesta seuraa lähes aina parantuneet läpäisyarvot verrattuna aikaisempaan, huolimatta siitä minkä keinon valitsee.

3.3 Nykyään käytössä olevat opetusmetodit ja uudet tulokkaat

Jo aikaisemmin mainitut Vihavainen A. ym. (2014) tutkivat laajasti, mitä ohjelmoinnin opetusmetodeja on käytössä, ja miten nämä vaikuttavat opetustuloksiin, joten tutkimusta voidaan pitää hyvänä katsauksena nykyään käytössä oleviin opetusmetodeihin. He pystyivät osoittamaan viisi eri kategoriaa, joihin erilaiset suositut ohjelmoinnin opetusmetodit voidaan jakaa: yhteistyö ja vertaistuki (collaboration and peer support), johdantokurssit ja -kielet (bootstrapping), samaistuttava sisältö ja kontekstualisointi (relatable content and contextualization), kurssijärjestelyt, arviointi, resurssointi, (course setup, assessment, resourcing) ja hybridiratkaisut (hybrid approaches). Kukin kuvaa siis nimensä mukaisesti jotain opetuksellista lähestymistapaa, jonka kautta oppimista ensimmäisellä ohjelmointikurssilla pystyttäisiin parantamaan.

Tehokkaimmaksi näistä osoittautui yhteistyö ja vertaistuki -kategoria: se sisältää yleisesti yhteistyöhön ja vertaistukeen liittyviä metodeja, kuten juuri pariohjelmointia sekä ryhmälähtöistä oppimista. Esimerkiksi ryhmälähtöisen oppimisen (eng. Team-based learning, TBL) perusperiaatteena on muuttaa kontaktiopetuksen merkitys verrattuna tavalliseen luennoivaan opetukseen: luennoinnin sijaan kontaktikerralla sovelletaan opeteltavia asioita ryhmissä. Oppilaiden on määrä tehdä valmistelevia harjoituksia, jotta kontaktitunnille tultaessa nämä olisivat valmiita myös soveltamaan tietojaan. Fokus tällaisessa oppimisessa on siis muiden opiskelijoiden kanssa yhdessä suoritetut harjoitukset, opettajan avun ollessa koko ajan käden ulottuvilla. Lasserre P. ym. (2011) toteuttamalla kurssilla korostettiin erityisesti myös vertaisarvioinnin roolia: Kurssin ryhmätehtävät toteuttivat tiettyä kaavaa, joka takasi, että jokaisella oppilaalla oli samanlainen harjoitus ryhmästä riippumatta. Lisäksi harjoitusten päätteeksi jokainen ryhmä raportoi omat tuloksensa koko luokalle. Vaikka ryhmälähtöinen oppiminen lasketaan aktiivisen oppimisen piiriin (Lasserre, Szostak 2011), siinä on myös vahvoja vaikutteita käänteisestä oppimisesta (Amresh, Carberry ym. October 2013). Hyvin samanlaisia tavoitteita on myös pariohjelmoinnilla sekä yhteisöllisellä oppimisella:

oppilaat järjestetään ryhmiin, ja näiden annetaan tehdä yhdessä soveltavia tehtäviä (Williams, Laurie 2007, Walker 1997).

Johdantokurssit ja -kielet -kategoriaan kuuluvat puolestaan metodit, joissa ohjelmointikurssin aloitusta on jotenkin koitettu helpottaa. Tämä voidaan toteuttaa esimerkiksi järjestämällä ennakkokurssi ennen varsinaista ohjelmointikurssia, tai käyttämällä aluksi visuaalista ohjelmointikieltä, kuten Scratchia tai Alicea. Yleisesti ennen ensimmäistä ohjelmointikurssia (CS1-kurssi) käytävää ”preppikurssia” kutsutaan CS0-kurssiksi (Vihavainen A. 2014). Kuten esimerkiksi Sloan R. ym. (2008) kuvaavat, CS0-kurssin tarkoituksena on tarjota helposti lähestyttävää ja kiinnostavaa materiaalia ja tekemistä, samalla kun oppilaiden käsitys ohjelmoinnista ja sen rakenteista yleisesti parantuu. On siis jopa hieman kyseenalaista, onko CS0-kurssissa kyse ohjelmointikurssista: ajatuksena on pikemminkin nostaa ymmärrystä ja kiinnostusta yleisesti tietojenkäsittelyä kohtaan. Tarkoituksena on myös kohdentaa tällainen kurssi oppilaisiin, joilla ei ole entuudestaan juurikaan kokemusta ohjelmoinnista; vastaavasti kokeneempien oppilaiden on helppo jättää CS0-kurssi väliin ja siirtyä suoraan CS1-kurssiin. Samalla tapaa, kun kurssilla käytetään visuaalista ohjelmointikieltä, opetuksen päämäärä on myös sama: ei välttämättä opettaa heti ohjelmointia, vaan pikemminkin parantaa käsitystä ohjelmoinnista, algoritmeista ja näiden rakenteista yleisesti, sekä lisätä oppilaiden innostusta tietojenkäsittelyyn (Rizvi, Humphries ym. 2011).

Samaistuttava sisältö ja kontekstualisointi -kategoriaan kuuluu nimensä mukaisesti metodit, jotka tuovat oppilaille helpommin lähestyttävää ja samaistuttavaa materiaalia. Tavoitteena tällöin on siis tehdä ohjelmoinnista jollain tapaa paremmin ymmärrettävää. Käytännön keinoja tähän ovat esimerkiksi kurssit, jotka pyrkivät tuomaan ymmärrystä pelien kautta, käytännönläheisten projektien myötä, tai rakentamalla kurssi yleisesti niin, että se tähtää johonkin konkreettiseen lopputulokseen. Näin oppilas pääsee näkemään ohjelmoinnin käytännön vaikutuksia, joka esimerkiksi Tew A. ym. (2005) mukaan vaikuttaa positiivisesti oppilaiden asenteisiin ohjelmointia ja yleisesti tietojenkäsittelytieteitä kohtaan. Tew A. ym. esittelemällä kurssilla käytettiin juuri

tällaisia keinoja: ohjelmointia ja sen konsepteja lähestyttiin käytännön esimerkkien kautta, esimerkiksi luomalla filttereitä kuville ja äänille, sekä generoimalla erilaisia animaatioita. Tämä oletettavasti helpottaa oppilaita muodostamaan mielleyhtymiä ohjelmointikonsepteihin, jotka muuten ovat abstraktiutensa takia haastavia.

Kurssijärjestelyt, arviointi, resurssointi -kategoria tuo jälleen nimensä mukaisesti opiskelijoille paranneltuja toimintoja, kuten uutta sisältöä tai uusi ja parempi ohjelmointiympäristö. Myös esimerkiksi luokkakoon muuttamisella huomattiin olevan positiivinen vaikutus. Kategorian metodit keskittyvät kuitenkin enemmän kurssin sisällön muutoksiin, eivätkä niinkään kurssin opetuksellisiin muutoksiin. Esimerkkinä tästä on Radosevic D. ym. (2009) esittelemä helpomman ohjelmointiympäristön käyttö kurssilla: Tämän avulla koitettiin saada oppilaat oppimaan esimerkiksi oikeanlaisen syntaksin tärkeys. Ohjelmointiympäristöön oli integroitu myös analysointi- sekä palautejärjestelmä, jonka tarkoituksena oli edesauttaa oppilaan ymmärrystä omasta koodistaan sekä siinä mahdollisesti tehdyistä virheistä. Tässäkin muutos keskittyy kuitenkin pääasiassa kurssin toteutustavan muuttamiseen, ei välttämättä niinkään opetusmetodien muuttamiseen. Vaikka esimerkiksi juuri välittömän palautteen onkin todettu olevan hyödyksi oppimiselle (Kaila, Rajala ym. 2010), lähestymistapa on kuitenkin hieman eri. Kuitenkin, kuten Radosevic D. ym. mainitsevat, tällaisillakin lähestymistavoilla onnistutaan parantamaan oppimistuloksia.

Viimeisin viidestä kategoriasta, hybridiratkaisut-kategoria, kuvastaa metodeja, joita ei pystytty yhdistämään mihinkään aiemmista kategorioista, tai jotka ovat edellä olleiden kategorioiden yhdistelmiä. Porter L. ym. (2013) esittelevät lähestymistavan, jossa on yhdistetty kolmea hyväksi todettua opetusmetodia: pariohjelmointia, vertaisohjausta sekä kontekstualisoitua laskentaa ja multimediaohjelmointia (media computing). Vaikka kolmen erillisen opetusmetodin implementointi samalle kurssille voikin kuulostaa hieman täyteen ahdetulta, Porter L. ym. osoittaa kuitenkin tämän lähestymistavan toimineen ainakin heidän tapauksessaan hyvin. Metodien käytännön implementointi kurssille ei tule kovin selvästi ilmi heidän artikkelistaan, joten tällaisen hybridiratkaisun tarkempi analysointi on hieman vaikeaa. On esimerkiksi vaikea sanoa,

miten tällaisen hybridiratkaisun käyttö kurssilla vaikuttaa oppimiseen verrattuna vain yhden metodin käyttöön. Hybridiratkaisujen käyttö on kuitenkin todettu hyödylliseksi, vaikkei sen erot yksittäisiin metodeihin olekaan täysin selviä.

Loppujen lopuksi näiden viiden kategorian sekä metodien välillä olevat erot tehokkuudessa ovat lähinnä marginaalisia. Kuitenkin näillä marginaalisilla eroilla ajateltuna on hieman yllättävää, että esimerkiksi pariohjelmointi oli toimivuudessaan listan häntäpäässä. Vihavainen A. ym. (2014) argumentoivat tämän voivan johtua esimerkiksi siitä, että kurssit, joille pariohjelmointi otettiin käyttöön, olivat kenties jo aikaisemmin onnistuneet parantamaan oppilaiden läpäisyprosenttia, jolloin tutkimuksen mittaama absoluuttinen parannus jää verrattain pieneksi. Vaikka spekuloitavaa hieman jääkin, on erot metodien välillä kuitenkin, kuten sanottu, pieniä. Vihavainen ym. mainitsevat lisäksi, että esimerkiksi juuri pariohjelmoinnin käyttö pelkästään ei välttämättä tuota parhaita mahdollisia tuloksia. Tämä lienee asian laita myös yleisesti: esimerkiksi aktiivinen oppiminen ei ole yhtään niin hyödyllistä yksistään, kuin jos sen yhdistää esimerkiksi mielenkiintoiseen ja paranneltuun oppimateriaaliin sekä ryhmätyöhön.

Yksi syy miksi juuri nämä metodit ovat laajalti käytössä on varmasti niiden todettu toimivuus. Tämä taas puolestaan johtuu osin siitä, että niitä on tutkittu paljon: mieluummin sitä ottaa käyttöön jonkin uuden opetusmetodin, kun tietää että sen pitäisi myös toimia. Uusia opetusmetodeja syntyy kuitenkin koko ajan, ja ne eivät suinkaan keksi pyörää uudestaan, vaan nojautuvat jo hyviksi todettuihin metodeihin ja teorioihin. Esimerkkinä tästä on verrattain uusi opetusmetodi, ilmiöpohjainen oppiminen, jonka Leppiniemi H. (2016) esittelee pro gradu -tutkielmassaan. Kyseinen metodi ei tunnu olevan vielä laajalti levinnyt, ainakaan ilmiöpohjaisen oppimisen (phenomenon based learning) nimellä. Metodi pohjautuu pitkälti vallalla oleviin teorioihin ja hyviksi todettuihin opetusmenetelmiin; siinä oppimisen ajatellaan olevan konstruktivistista, ja esimerkiksi yhteistyö sekä asioiden konkreettisuus ovat vahvasti esillä. Ilmiöpohjaisessa oppimisessa korostuu etenkin oppijan omat kokemukset sekä tämän oma kiinnostus käsillä olevaa asiaa kohtaan, ja uudet asiat pyritäänkin sitomaan oppijan omiin

kokemuksiin. Ymmärrys pyritään rakentamaan jonkin konkreettisen tosimaailman ilmiön pohjalle, joka mieluiten on oppiainerajat ylittävä, jolloin opittava asia on helpompi liittää osaksi suurempaa kokonaisuutta. Ilmiöpohjainen oppiminen voi kuitenkin olla hieman hankala soveltaa ohjelmoinnin opetukseen, ja etenkin abstrakteista konsepteista on vaikea löytää mitään tosimaailman rinnastettavia esimerkkejä; toisaalta jos tässä onnistuttaisiin, sen luulisi helpottavan huomattavasti ohjelmointikonseptien oppimista. Ilmiöpohjainen oppiminen on hyvä esimerkki siitä, miten hyvistä metodeista ja oppimisteorioista pystytään kehittämään uusia, lähestyen oppimista jälleen hieman uudella tavalla.

3.4 Ohjelmoinnin opetuksen vaikeudet

Kuten on jo aikaisemmin todettu, ohjelmoinnin oppiminen ja opetus on todettu haastaviksi etenkin ensimmäisillä ohjelmointikursseilla (Bennedsen, Caspersen 2007, Watson, Li 2014, Lahtinen, Ala-Mutka ym. 2005). Nämä eivät kuitenkaan ole varsinaisesti erillään toisistaan, vaan esimerkiksi opetuksen vaikeudet heijastavat luonnollisesti myös oppimisen vaikeuksia. Näin ollen voidaan päätellä, että kun oppilaat pitävät esimerkiksi olio-ohjelmoinnin abstrakteja rakenteita vaikeina, niin myös näiden opettaminen on opettajille haastavaa. Etenkin juuri olio-ohjelmoinnin opetuksen vaikeuteen ottavat kantaa Weir G., Vilner T., Mendes A. ja Nordström M. Heistä Weir G. tiivistää opetuksen vaikeuden ohjelmoinnin luonteeseen: Se vaatii sekä tietoa että käytännön toteutustaitoa. Pelkkä tiedon opettaminen ja oppiminen ei siis riitä, vaan tulisi keskittyä myös käytännön taitojen harjoittamiseen. Nordström M. lähestyy ongelmaa hieman toiselta kantilta, ja mainitsee ongelmien olevan enemmän itse paradigman vaikeudessa: valaisevia ja sopivia esimerkkejä on vaikea löytää, kun opetettava aihe on niinkin abstrakti kuin olio-ohjelmointi. (Weir, Vilner ym. 2005)

Ohjelmoinnin opetuksen vaikeuksia on kuitenkin syytä tarkastella myös olio-ohjelmoinnin ulkopuolella, sen suuresta suosiosta huolimatta. Esimerkiksi Dale N. (2006) kysyi ensimmäistä ohjelmointikurssia opettavalta henkilökunnalta, mikä heidän mielestään on vaikein aihe opettaa ensimmäisellä ohjelmointikurssilla. Dale N. onnistui kokoamaan vastauksista neljä kategoriaa, joihin vaikeasti opetettavat aiheet jakautuivat:

ongelmanratkaisu ja suunnittelu (problem solving and design), yleiset ohjelmointiaiheet (general programming topics), olio-ohjelmoinnin käsitteet (object-oriented constructs) ja oppilaiden kypsyys (student maturity). Näistä kolme ensimmäistä lienevät selviä nimiensä perusteella; viimeinen kategoria ”oppilaiden kypsyys” korostaa enemmänkin oppilaiden huonoja asenteita ja vaillinaisia opiskelutaitoja, jolloin opettamisen ei voida osoittaa olevan vaikeaa jonkin tietyn asian takia, vaan johtuvan yleisesti opiskelijasta itsestään. Myös Hegazi M. ja Alhawarat M. (2015) tekemässä tutkimuksessa ensimmäisen ohjelmointikurssin opettajille tehdyssä haastattelussa löytyi näitä kategorioita tukevia ongelmia: etenkin oppilaiden opiskelutaitojen ja motivaation vähäisyys, sekä puutteet ongelmanratkaisutaidoissa nousivat vahvasti esille. Samassa tutkimuksessa tehdyssä kyselyssä oppilaat eivät kuitenkaan itse kokeneet tai nähneet motivaatioon negatiivisesti vaikuttavia tekijöitä, ja pitivät ainakin osia suorittamastaan kurssista hyvinä ja opettavaisina. Ei siis ole aivan selvää mikä loppujen lopuksi on syy ja mikä seuraus: onko opeteltavat asiat vain niin vaikeita, ettei työn jälkeenkään tahdo näkyä tuloksia ja näin motivaatio ei pysy yllä, vai puuttuuko motivaatio jo alkujaan. Etenkin juuri Hegazi M. ja Alhawarat M. esittämässä tutkimuksessa korostuu, miten opettajien ja oppilaiden erilaiset roolit vaikuttavat myös kunkin subjektiiviseen näkemykseen asiasta. Ongelmakohdat hahmottuvat kuitenkin hyvin, oli syy mikä tahansa, ja niihin ollaankin onneksi jo osattu puuttua ja vastata, esimerkiksi edellisissä kappaleissa esitellyin tavoin ja metodein.

Guzdial M. ja Guo P. (2014) pohtivat, miksi ohjelmoinnin oppiminen on niin vaikeaa, ja millä tekijöillä tämä voisi selittyä. Keskeisessä roolissa heidän pohdinnoissaan on termi opittavuus (learnability): he argumentoivat, että ohjelmoinnin oppimisen haastavuus johtuu ohjelmointikielten huonosta opittavuudesta, tarkemmin niiden odotusarvon ja sosiaalisen hintansa suhteen. Odotusarvolla viitataan tässä kohtaa opittavan asian niin kutsuttuun kognitiiviseen kuormaan: se on henkilölle aiheutuva kuorma, käytännössä työmäärä, jonka tämä joutuu tekemään oppiakseen jonkin uuden asian. Jos oppilas kuitenkin arvioi tämän työmäärän olevan suurempi kuin mitä opitusta asiasta on käytännössä hyötyä, toisin sanoen mikä sen odotusarvo on, tämä johtaa alhaiseen sisäiseen motivaatioon. Tällaista ajatusta motivaatiosta kutsutaan odotusarvoteoriaksi

(expectancy-value theory), jossa siis ydinajatuksena on, miten ihminen arvioi oman kykynsä suorittaa jokin asia, ja miten arvokkaaksi hän tämän asian suorittamisen kokee (Wigfield, Eccles 2000). Guzdial M. ja Guo P. argumentoivat myös ennen kaikkea havaitun hyödyn puolesta: vaikka jokin taito voisi olla parempi verrattuna toiseen, esimerkiksi olio-ohjelmoinnin voidaan ajatella oleva parempi tietynlaisten ohjelmien rakentamisessa, mutta jos henkilö ei koe tällaista taitoa hyödylliseksi sen paremmuudesta huolimatta, havaittu hyöty ei ylitä havaittua hintaa. Tästä päästäänkin siis suoraan ohjelmoinnin opetukseen: jos siis oppilas ei yksinkertaisesti näe ohjelmoinnin opettelusta olevan merkittävää hyötyä, miksi pistää voimavaroja vaativaan ponnistukseen, jota ohjelmoinnin opettelun on todettu olevan. Ensi askeleet ohjelmoinnin opetuksessa siis tulisi olla vakuuttaa oppilaat ohjelmoinnin hyödyistä muutenkin kuin vain kurssin läpäisyä silmällä pitäen.

Guzidal M. ja Guo P. (2014) puhuvat lisäksi myös oppimisen sosiaalisesta hinnasta. Idea tässä on jokseenkin samanlainen kuin juuri esitetty odotusarvoteorian mukainen käsitys: Henkilö arvioi jonkin asian opettelun hyödyllisyyttä omasta näkökulmastaan, ja punnitsee, ovatko hyödyt suuremmat kuin opettelusta aiheutuva työmäärä. Tällä kertaa asioita kuitenkin punnitaan sosiaalisesta näkökulmasta. Artikkelissa on annettu hyvä esimerkki: Fyysikot suosivat verrattain alkeellisia ohjelmointikieliä laskelmissaan, kuten joitain Fortran-tyyppisiä kieliä. Voisi olla teoriassa järkevämpää siirtyä johonkin tehokkaampaan ja modernimpaan kieleen, mutta jälleen törmätään uuden kielen opettelukynnykseen: miksi jonkun yksittäisen fyysikon tekisi mieli opetella uusi ohjelmointikieli, kun kuitenkin muut fyysikot käyttäisivät vielä vanhoja. Muutoksen täytyisi olla kokonaisvaltainen, jotta kenenkään olisi järkeä opetella uutta ohjelmointikieltä. Tällainen kokonaisvaltainen muutos olisi kuitenkin todella suuri, ja vaikka jokin fyysikkoryhmä päättäisikin opetella jonkin uuden kielen, tulee esiin selviten asian sosiaalinen puoli: He jäisivät lähes täysin ulkopuolelle muusta fyysikkoyhteisöstä. Esimerkiksi koodin jakaminen muiden fyysikkojen kanssa olisi tälle eristäytyneelle ryhmälle lähes mahdotonta, saati muiden kirjastojen ja havaintojen käyttäminen. Havaittu sosiaalinen hinta kuvaa siis havaittua hyötyä ja haittaa esimerkiksi jonkin sosiaalisen ryhmän sisällä. Se on omalla tapaansa vielä entistä

subjektiivisempi kuin aiempi odotusarvo, johon motivaatiota voi etsiä esimerkiksi hyvistä työmarkkinoista; sosiaalinen hinta on enemmänkin ihmisen luonnollista yhteisöllisyyttä ja halua kuulua johonkin ryhmään.

Ohjelmoinnin opettaminen ei siis ole suoraviivaista, vaan siinä pitää ottaa huomioon paljon erilaisia asioita, kuten oppilaat sekä näiden taidot ja asenteet, itse käsillä oleva aihe, sekä erilaiset haasteet, joita erilaiset aiheet tuovat niin opettajan kuin oppilaankin eteen. Useiden tässäkin luvussa mainittujen lähteiden mielestä oppimisen ytimessä on kuitenkin loppujen lopuksi motivaatio, ja esimerkiksi juuri edellä esitetyt odotusarvoteoria sekä sosiaalinen hinta ovat hyviä esimerkkejä siitä, miten se voi näyttäytyä käytännössä. Loppujen lopuksi on kuitenkin kaikkien etu, jos oppilas innostuu toden teolla opiskelemastaan aineesta: etenkin ohjelmointia opiskelevien kohdalla tämä olisi tärkeää, sillä ohjelmointi on kuitenkin yleensä vain osa esimerkiksi tietojenkäsittelytieteitä opiskelevan opintoja, eikä ohjelmoinnin opettelu välttämättä ole opiskelumotivaation ytimessä alun alkujaankaan. Tulee myös pitää mielessä, että ensimmäinen ohjelmointikurssi voi olla useille ensimmäinen kosketus ohjelmoinnin lisäksi myös esimerkiksi algoritmiseen ajatteluun ja ongelman paloihin jakamiseen ohjelmoinnin vaatimalla tavalla: näin ollen myös siinä käytetyt abstraktit käsitteet ja rakenteet voivat olla outoja, ja motivoivan opetuksen rooli korostuu entisestään.

4 MOTIVAATIO OHJELMOINNIN OPISKELUSSA

Opetusmetodien ja -teorioiden ohella yhtenä tärkeimpänä tekijänä oppimisessa voidaan pitää motivaatiota: sehän loppujen lopuksi määrittelee, onko henkilöllä oikeasti halua oppia jotain (Schunk 2012). Motivaation on myös tutkitusti todettu vaikuttavan oppimiseen (Gomes, A., Mendes October 2014, Nikula, Gotel ym. 2011). Mutta mitä motivaatio oikeastaan on? Pardee R. (1990) esittelee neljä klassista määritelmää ja teoriaa motivaatiolle: Maslowin tarvehierarkian, Herzbergin kaksifaktoriteorian, McGregorin XY-teorian sekä McClellandin suoritusmotivaatioteorian. Näistä kolme, Maslow, Herzberg sekä McClelland esittävät motivaation tarvelähtöisesti, joihin muiden motivaatioteorioiden voidaan ajatella nojaavan. McGregorin XY-teoria puolestaan käsittelee enemmänkin sisäisen ja ulkoisen motivaation ja motivoinnin eroja. Hän ei

varsinaisesti puhu sisäisestä ja ulkoisesta motivaatiosta, mutta käsittelee motivaatiota suhteessa sisäisiin ja ulkoisiin ärsykkeisiin.

Nämä teoriat ovat klassisia esimerkkejä siitä, miten motivaatiota ajatellaan ja miten sitä on koitettu määritellä. Modernimman näkökulman motivaatioteoriaan tarjoaa Salmela-Aro K. ja Nurmi J-E. (2005) kirjassaan ”Mikä meitä liikuttaa – Modernin motivaatiopsykologian perusteet.” Heidän teoriansa perustuu kolmeen käsitykseen motivaatiosta: Brian Littlen teoriaan henkilökohtaisista projekteista, Robert Emmonsin teoriaan henkilökohtaisista pyrkimyksistä, sekä kirjoittajien yhteiseen teoriaan, kontekstuaaliseen tavoiteteoriaan. Näiden kolmen muodostama moderni motivaatioteoria tutkii motivaatiota ihmisten kertomien tavoitteiden ja pyrkimysten avulla, sekä henkilökohtaisten projektien kautta. Siinä siis korostuu ihmisen itsensä tiedostamat ja muodostamat motivaatiot ja päämäärät, sekä huomattavan paljon myös ympäristön vaikutus päämääriin pääsyyn sekä niihin johtaviin suunnitelmiin. Moderni motivaatioteoria pitää ihmistä siis enemmänkin itseohjautuvana ja suunnittelevana toimijana, kuin pelkästään tiedostamattomien motivaatioiden ohjaamana.

Kuten jo aikaisemmin esitettiin, motivaation on todettu vaikuttavan merkittävästi oppimiseen, ja yksinkertaisesti lisäämällä oppilaiden motivaatiota on mahdollista parantaa näiden oppimista. Vaikka motivaation lisääminen itsessään ei olekaan enää niin yksinkertaista, on hyvä tiedostaa kuinka suuri merkitys sillä voi olla. Erityisesti ohjelmoinnin oppimisessa motivaatiolla on todettu olevan paljonkin vaikutusta: esimerkiksi Nikula U. ym. (2011) toteaa ensimmäisen ohjelmointikurssin keskeyttämisprosentin laskeneen huomattavasti, kun kurssin opiskelijoiden motivaatiota onnistuttiin lisäämään. Myös Gomes A. ym. (2014) toteaa huonon (sisäisen) motivaation olevan avainasemassa, kun ajatellaan ohjelmointia opiskelevien suoriutumista ensimmäisellä ohjelmointikurssilla. Etenkin ensimmäisellä ohjelmointikurssilla olisi siis hyvä keskittyä motivoimaan oppilaita, jolloin näiden asenteet ohjelmointia kohtaan paranisivat ja motivaatio jatkaa ohjelmointia kasvaisi.

Motivaation voidaan siis todeta olevan tärkeä tekijä oppimisessa, ja etenkin ohjelmoinnin oppimisessa. Yksi syys miksi motivaatioon puuttuminen on vaikeaa, on

se, että sitä on vaikea mitata: näin ollen on vaikea todentaa, mitkä keinot toimivat ja kuinka hyvin. Kuten Salmela-Aro K. ym. (2005) kirjoittavat, motivaation tutkimusmenetelmät ovat melko yksinkertaisia verrattuna motivaatiokäsitteen monimutkaisuuteen ja rikkauteen. He myös korostavat, miten motivaation tutkimus kohdistuu pääsääntöisesti tietoisen motivaation tutkimiseen, ja että tämä on kuitenkin vain murto-osa ihmisen koko motivaatiosta, josta suuri osa muodostuu juuri tiedostamattomista tekijöistä. Motivaatiota on kuitenkin mahdollista mitata jossain määrin, muutamilla eri tavoilla: Muun muassa Salmela-Aro ja Nurmi esittelevät joitain motivaatiokyselyitä, jotka heijastavat usein kyselyn luoneen tutkijan omaa teoriaa motivaatiosta. Motivaatiota pystytään tarkastella myös asenteiden kautta: Wong K. Y. ja Chen Q. (2012) ovat jalostaneet toimivan asennekyselyn, jonka avulla pystyään mittaamaan oppilaiden asenteita opiskelemaansa aihetta kohtaan, ja näin ollen analysoimaan näiden motivaatioita yleisesti oppiainettaan kohtaan. Kyseistä kyselyä on käytetty myös tämän tutkielman kyselyn pohjana.

Seuraavaksi esitellään muutamia motivaatioteorioita hieman tarkemmin, niin klassisia teorioita kuin modernimpiakin, ja pyritään avaamaan yleisesti mitä motivaatio on, ja mikä sen vaikutus on opiskelussa yleisesti. Tämän jälkeen pureudutaan motivaation vaikutukseen etenkin ohjelmoinnin opiskelussa.

4.1 Motivaatio ja sen merkitys yleisesti opiskelussa

Kuten jo aikaisemmin todettiin, motivaatiolle voidaan esittää muutama klassinen määritelmä. Pardee R. (1990) löytää näitä yhteensä neljä: Maslowin tarvehierarkian, Herzbergin kaksifaktoriteorian, McClellandin suoritusmotivaatioteorian sekä McGregorin XY-teorian. Näistä ensimmäinen ja tunnetuin on Maslowin tarvehierarkia, jonka mukaan ihmisten motivaatio muuttuu näiden tarpeiden mukaan: mikäli jokin perustarpeista ei ole kunnossa, esimerkiksi ruokaa ei ole tarpeeksi, ei motivaatio ylempiä tarpeita kohtaan ole kovin suurta, esimerkiksi motivaatio ihmissuhteiden ylläpitämiseen. Maslow määrittelee viisi eri tarvetasoa, joista alemmat tulee olla kunnossa ennen kuin motivaatio ylempiä kohtaan lisääntyy: fysiologiset tarpeet,

turvallisuuden tarpeet, liittymisen tarpeet, arvostuksen tarpeet, sekä itsensä toteuttamisen tarpeet (matalimmasta korkeimpaan).

Myös Herzberg perustaa teoriansa ihmisen perimmäisiin tarpeisiin. Hän jakaa kuitenkin motivaatioon vaikuttavat tekijät hieman eri tavalla, kahteen osaan, ja keskittyen enemmän työmaailmaan: hänen mukaansa on olemassa kahta erilaista motivaatiotyyppiä, työtyytyväisyyttä (motivoivat tekijät) ja työtyytymättömyyttä (hygieniatekijät). Herzberg korostaa, etteivät nämä ole toistensa vastakohtia, vaan liikkuvat ikään kuin omalla akselillaan. Esimerkiksi osana hygieniatekijöitä on työympäristö: mikäli työympäristö on huono, se aiheuttaa motivaation laskua työntekijässä. Kuitenkin jos työympäristö on poikkeuksellisen hyvä, se ei suoranaisesti nosta työntekijän motivaatiota, vaan pikemminkin vain neutralisoi työtyytymättömyyden. Työntekijä ei siis tee erityisen innostuneesti ja motivoituneesti töitä, koska tällä on hyvä työympäristö; erityisen motivoituneeseen työskentelyyn tarvitaan jonkin motivoivan tekijän parantamista, kuten esimerkiksi työn mielekkyyden lisäämistä. Herzberg ehdottaa myös, että motivoivat tekijät edustaisivat sisäistä motivaatiota, ja hygieniatekijät ulkoista motivaatiota.

McClelland puolestaan ehdottaa motivaation toimivan jälleen hieman eri tavalla: kun ihmisellä on korkea tarve johonkin, se motivoi ihmistä käyttäytymään siten, että kyseinen tarve pystytään tyydyttämään. Pääteemana McClellandilla onkin juuri nämä tarpeet, ja miten ihminen niitä oppii selvitäkseen ympäristössään. Ajatus on siis myös hieman behavioristinen, sillä tästä aiheutuu, että palkitun käyttäytymisen voidaan ajatella toistuvan useammin. McClellandin mukaan siis motivaatio syntyy ihmisen sen hetkisistä tarpeista, ja motivaatio on ikään kuin oikeanlaiseen toimintaan kannustava tekijä, jotta tarve pystyttäisiin tyydyttämään.

McGregorin XY-teoria poikkeaa aikaisemmista teorioista siinä, että se keskittyy pikemminkin yhteen motivaation osa-alueeseen: se pohjaa ideansa Maslowin tarvehierarkian korkeimpaan tarpeeseen, itsensä toteuttamisen tarpeeseen. Myös McGregorin teoriat suuntautuvat vahvasti työelämään. McGregorin Y-teorian mukaan ihminen on töissä itseohjautuva ja luova, kun tämä on oikealla tavalla motivoitunut.

Toinen puoli McGregorin teoriasta, X-teoria, on ikään kuin Y-teorian vastakohta: siinä ajatuksena on, että ihmistä pystyy ohjaamaan ja motivoimaan ainoastaan ulkoisten ärsykkeiden ja motivaattoreiden avulla. (McGregor 1960)

Vaikka teorioita onkin monia, eivät ne välttämättä kumoa toisiaan, vaan ennemminkin täydentävät toisiaan luomalla samalle ilmiölle useita mahdollisia selityksiä. Yleisimmät ja ensimmäiset motivaatioteoriat olivat siis Maslowin, Herzbergin ja McClellandin esittämät teoriat. Motivaatiosta on myös tuoreempia teorioita, kuten aiemmin esitellyssä Salmela-Aron ja Nurmen (2005) kirjoittamassa kirjassa. Myös esimerkiksi Schunk (2012) on ottanut kantaa motivaation määritelmään, ja motivaation määritelmä onkin näissä kahdessa aika samanlainen. Motivaatio sisältää käsitteenä kuitenkin niin paljon erilaisia asioita, että yhtä oikeaa ja kaiken kattavaa teoriaa tai tulkintaa ei varmaankaan voida osoittaa. Tämän tutkimuksen valossa lähestymme motivaatiota pitkälti sisäisen ja ulkoisen motivaation kautta, sekä pyrkimällä ottamaan mukaan myös modernin motivaatiopsykologian teemoja; näitä esitellään seuraavaksi siis hieman tarkemmin.

Ryan R. ja Deci E. (2000) käsittelevät yleisesti motivaation lisäksi sitä, miten motivaatiota on olemassa eri määriä ja eri muotoja: jollain voi olla suuri tai pieni motivaatio, mutta syy tälle voi olla esimerkiksi joko oma innostuneisuus tai halu miellyttää vanhempia. He pitävät yksinkertaisimpana jakona motivaatioiden syille juurikin tällaista jakoa sisäisiin ja ulkoisiin motivaatioihin, jossa motivaation ajatellaan kumpuavan henkilöstä itsestään, tai jostain ulkoisesta lähteestä, kuten esimerkiksi vanhempien tai koulun painostuksesta tai halusta miellyttää. Etenkin juuri kouluympäristöön tämä jako on toimiva, jolloin erottuvat hyvin opiskelijan oma kiinnostus ja motivaatio, sekä puolestaan koulun asettamat suoriutumisvaatimukset. Ryan ja Deci toteavat itsekin tällaisen motivaation tarkastelumallin olevan hyödyllinen koulu- ja oppimisympäristössä.

Klassisesti ajatellaan, että sisäinen motivaatio tuo parempia tuloksia oppimisessa, kuin ulkoinen motivaatio: on siis oppimisen suhteen parempi, jos oppija on itse motivoitunut opiskeltavasta aiheesta, kuin että esimerkiksi koulu koittaisi siihen motivoida. Ryan ja Deci (2000) kuitenkin argumentoivat, että pelkästään sisäinen motivaatio ei ole merkki korkealaatuisesta oppimisesta, vaan myös ulkoinen motivaatio voi saada sitä aikaan;

ulkoisen motivaation onkin todettu olevan monimutkaisempi käsite kuin miten klassinen käsitys ulkoisesta motivaatiosta sen kuvaa. Itseohjautuvuusteoria (self-determination theory) ehdottaa, että on olemassa erilaisia ulkoisen motivaation muotoja. Tämä tarkoittaa esimerkiksi sitä, että oppilas voi suorittaa jonkin tehtävän auliisti ja halukkaasti, jos tehtävä heijastaa oppilaan sisäistä hyväksyntää tehtävän arvoja ja hyödyllisyyttä kohtaan, vaikka tehtävä olisikin alkujaan ulkoisten motivaatiotekijöiden sanelema. Ryan ja Deci nostavatkin aivan aiheesta esiin etenkin tämän ulkoisen motivaation eri tyyppien merkityksen: kaikki opiskeltavat kurssit ja aiheet esimerkiksi tutkinto-ohjelmassa eivät voi aina olla jokaiselle kiinnostavia, vaan on löydettävä myös ulkoisesti motivoivia tekijöitä, joilla saataisi kuitenkin mahdollisimman hyviä oppimistuloksia. Tämän tutkielman tuloksia käsitellään pitkälti klassisen sisäisen ja ulkoisen motivaatioteorian kautta, koska sen on todettu olevan hyvä tapa lähestyä etenkin opiskelumotivaatiota, mutta motivaation luonnetta tätä laajempana käsitteenä ei tule kuitenkaan unohtaa.

Moderni motivaatioteoria määrittelee motivaatiota hieman eri lähtökohdista kuin klassiset motivaatioteoriat: kuten jo aikaisemmin esiteltiin, Salmela-Aro ja Nurmi (2005) kokoavat modernin motivaatioteorian alle Brian Littlen teorian henkilökohtaisista projekteista, Robert Emmonsin teorian henkilökohtaisista pyrkimyksistä, sekä oman yhdistetyn kontekstuaalisen tavoiteteoriansa. Littlen teoria pyrkii kuvaamaan ihmisen toimintaa omassa ympäristössään: ihminen käyttää henkilökohtaisia projekteja, eli toimintojen sarjoja, suhteessa ympäristöönsä. Little siis korostaa erityisesti ihmistä osana ympäristöään. Emmonsin teoriassa puolestaan kuvataan motivaatiota henkilökohtaisten pyrkimysten käsitteellä; henkilökohtaisen pyrkimyksen käsitteellä Emmons tarkoittaa siis sitä, mitä ihminen tyypillisesti tai luonteenomaisesti pyrkii elämässään tekemään. Hän korostaa myös motivaation ja tavoitteiden hierarkkisuutta, sekä näiden suhdetta ja vaikutusta toisiinsa; miten siis pyrkimys johonkin asiaan A vaikeuttaa pyrkimystä johonkin toiseen asiaan B. Kirjan kirjoittajat Salmela-Aro ja Nurmi ovat koonneet yhteen myös oman kuvauksensa motivaation toiminnasta, jota he kutsuvat kontekstuaaliseksi tavoiteteoriaksi: teorian mukaan ihminen ohjaa elämäänsä

asettamalla tavoitteita, kehittämällä suunnitelmia ja strategioita niihin pääsemiseksi, sekä arvioimalla toiminnan tuloksellisuutta.

Näissä kaikissa kolmessa teoriassa pohja on kuitenkin samantapainen, ja kirjassa esitetäänkin yhteenveto yhteisestä modernista motivaatioteoriasta. Siinä motivaatiota on tutkittu ihmisten kertomien tavoitteiden avulla, kuten minkälaisia tavoitteita, pyrkimyksiä tai hankkeita ihmisellä on, ja millaisiksi he arvioivat mahdollisuutensa toteuttaa niitä ja vaikuttaa niihin. Myös asioiden tärkeyden arviointi korostuu, sekä millaisia tunteita näihin tavoitteisiin, pyrkimyksiin tai hankkeisiin liittyy. Modernin motivaatioteorian pohjalta voitaisiin siis ajatella, että motivaatio on myös suurelta osin ihmisen tietoista ajattelua sekä kognitiivista toimintaa, ei siis pelkästään tiedostamatonta halua tehdä jotain, vaan myös tavoitteellista ja harkittua toimintaa. Myös esimerkiksi Schunk (2012) kuvaa motivaation ennen kaikkea kognitiivisena toimintana, ja korostaa sen kognitiivista roolia erityisesti osana opiskelua. Hän painottaa ihmisen kykyä tehdä suunnitelmia ja käyttää kognitiivisia prosesseja päästäkseen päämääriinsä, jotka ovatkin opiskelussa tärkeitä taitoja. Etenkin siis opiskelun yhteydessä ajatukset ympäristön vaikutuksesta sekä ihmisen kyvystä luoda tavoitteita ja suunnitelmia niihin pääsemiseksi ovat oleellisia.

Mutta miten motivaatio sitten käytännössä vaikuttaa oppimiseen? Jo yleisesti voisi ajatella motivaation olevan tärkeässä roolissa oppimisessa ja opiskelussa, ilman mitään erityisiä tutkimustuloksia. Jokaisella on tästä yleensä nimittäin ainakin jonkinlaisia omakohtaisia kokemuksia: jotakin itseä kiinnostavaa on huomattavasti mukavampi opiskella ja siitä on mukavampi ottaa selvää, kuin sellaisesta aiheesta joka ei itsessä herätä kovin suurta mielenkiintoa. Myös esimerkiksi Schunk (2012) puhuu vahvasti motivaation tärkeästä roolista oppimisessa. Hän puhuu myös toisesta verrattain intuitiivisesta, jo mainitusta, motivaation piirteestä: sisäisestä ja ulkoisesta motivaatiosta. On siis eri juttu, onko ihmisellä sisäinen motivaatio opiskella jotain asiaa, koska tämä on kiinnostunut aiheesta, vai onko jokin kurssi tai aihe pakollinen, jolloin sen sisältö ei välttämättä niinkään kiinnosta, mutta aihetta on pakko kuitenkin opiskella. Kuten aikaisemmin jo esitettiin, myös Ryan ja Deci (2000) puhuvat ulkoisesta ja sisäisestä

motivaatiosta, sekä siitä, miten nämä vaikuttavat henkilön oppimiseen. He painottavat erityisesti itsemääräämisoikeuden vaikutusta, joka heijastuu henkilön kokemaan niin sisäiseen kuin ulkoiseenkin motivaatioon. He myös ehdottavat, että oppimista voitaisi parantaa lisäämällä ihmisen perustarpeita opetustilanteessa: yhteyden tunnetta, tehokkuuden tunnetta, sekä avoimuutta uusia ideoita kohtaan ja uusien taitojen kokeilua.

Motivaation merkityksestä oppimisessa tunnutaan puhuttavan paljon, mutta sen vaikutuksesta suoraan oppimiseen tunnutaan sen sijaan puhuttavan vähän. Ennemmin puhutaan esimerkiksi motivaation vaikutuksesta tai suhteesta johonkin muuhun asiaan, jonka puolestaan ajatellaan kertovan opintomenestyksestä ja varsinaisesta oppimisesta. Esimerkiksi Pintrich P. (1999) kirjoittaa itseohjautuvasta oppimisesta, ja siitä, miten hyvä motivaatio näkyy muun muassa lisääntyneenä itseohjautuvana opiskeluna, joka taas puolestaan indikoi opiskelijan menestystä kurssilla. Myös esimerkiksi Boekaerts M. (2016) toteaa, että oppilaan sitoutumisen opiskeluun on todettu tukevan oppimista, ja tämä sitoutuminen puolestaan voidaan taas nähdä motivaation ilmentymänä. Schunk (2012) puolestaan nostaa motivaation tarveteorian pintaan eritoten juuri opiskelun yhteydessä: kyseinen teoria korostaa motivaation ilmentymistä erilaisten tavoitteiden kautta. Hän esittää tällä teorialla olevan paljon opetuksellisia sovellutuksia, joissa pyritään kasvattamaan oppilaiden halua oppia ja suoriutua hyvin. Näissä kaikissa on yhteistä se, että motivaation vaikutus näkyy jonkin toisen asian kautta, mutta motivaation suorasta vaikutuksesta oppimiseen ei osata sanoa erikseen mitään.

Kun motivaatiota tarkkaillaan tarpeeksi, pystytään sen rooli ja siihen vaikuttavat tekijät hahmottamaan paremmin. Näin ollen motivaatiolle on myös mahdollista tehdä jotain. Esimerkiksi Hung C-M. ym. (2012) osoittavat tarinankerronnallisen työkalun parantavan niin oppilaiden oppimistuloksia kuin motivaatiotakin. Myös Kopf S. ym. (2005) esittävät oppimisen sekä motivaation parantuneen, kun käytettiin osallistavaa simulointia oppilaiden aktivoimiseksi, joka puolestaan on siis konstruktionismiin perustuva opetusmetodi. Verrokkikohteina näillä tutkimuksilla on ollut perinteiset luentopohjaiset kurssit, joilla oppilailla ajatellaan yleisesti olevan aika huono motivaatio,

juurikin käytetyn opetusmetodin takia, joka ei juurikaan aktivoi eikä kannusta oppilaita osallistumaan tai tekemään mitään omatoimista. Motivaation nostaminen tällaiseen metodiin verrattuna voidaan siis ajatella olevan aika helppoa. Näistä motivaatiota nostavista keinoista on kuitenkin mahdollista hahmottaa yhteisiä piirteitä, kuten juuri osallistaminen ja oppijan oman tekemisen kautta oppiminen; näitä periaatteita käyttävät myös aikaisemmin esitellyt opetusmetodit.

4.2 Motivaation rooli ohjelmoinnin opiskelussa

Kuten aiemmassa luvussa esiteltiin, motivaatioteorioita on monenlaisia, ja motivaatiota voidaan lähestyä monella eri tavalla. Etenkin oppimisen näkökulmasta olennaisina teorioina voidaan pitää Ryan ja Deci (2000) esittämiä teorioita sisäisestä ja ulkoisesta motivaatiosta, sekä Salmela-Aron ja Nurmen (2005) esittämiä modernin motivaatiopsykologian näkökulmia. Motivaation rooli korostuu, kun puhutaan erityisesti ohjelmoinnin opiskelusta: ohjelmointia on vaikea oppia ilman jonkinlaista motivaatiota, sillä se vaatii paljon töitä, ja mikäli motivaatiota töiden tekemiseen ei ole, on myös oppiminen vaikeaa.

Motivaation vaikutuksesta ohjelmoinnin oppimiseen on myös tutkimustuloksia: Gomes A. ym. (2014) toteavat, miten huono (sisäinen) motivaatio on avainasemassa ensimmäisellä ohjelmointikurssilla, ja Nikula U. ym. (2011) toteavat ensimmäisen ohjelmointikurssin keskeytysprosentin laskeneen huomattavasti, kun kurssilaisten motivaatiota saatiin lisättyä. Myös muun muassa Behnke K. ym. (2016) toteavat parantuneen motivaation vaikuttaneen positiivisesti oppimistuloksiin, sekä tämän lisäksi myös muun muassa parantaneen oppilaiden asenteita tietojenkäsittelyä kohtaan, sekä rohkaisseen oppilaita jatkamaan tietojenkäsittelyn parissa. Esiin on nostettava hyvänä esimerkkinä erityisesti juuri Behnke K. ym. käyttämä Computer Science Principles -opetusmenetelmä (CS Principles), joka on erityisesti tietojenkäsittelyn opetukseen kehitetty viitekehys. CS Principles -viitekehyksen motivoivan aspektin perustana on itseohjautuvuusteoria (self-determination theory), joka erottaa erilaisia motivaatioita perustuen syihin tai päämääriin, jotka aiheuttavat toimintaa. Itse viitekehyksen päämääränä on esitellä tietojenkäsittelyn perusteita, iskostaa

laskennallisen ajattelun periaatteita oppilaisiin, kohottaa koodinlukutaitoa ja luovuutta, sekä saada oppilaat kiinnostumaan tietojenkäsittelyn mahdollisuuksista ja vaikutuksista. CS Principles on siis hyvä esimerkki juuri ohjelmoinnin opetuksen avuksi tehdystä opetusmenetelmästä.

Kuten jo mainittiinkin, motivaation rooli ohjelmoinnin opiskelussa on tärkeä, sillä asiat ja konseptit ohjelmoinnissa ovat vaikeita, ja vaativat paljon harjoittelua. Siksi on hyvä ymmärtää yleisesti motivaation rooli osana oppimisprosessia. Esimerkiksi juuri mainittu CS Principles on hyvä esimerkki, miten motivaatioteorioita pystytään hyödyntämään myös erityisesti ohjelmoinnin opiskelussa. Ryan ja Decin (2000) esittelemät teoriat ulkoisesta ja sisäisestä motivaatiosta ovat erittäin käypiä opetusmaailmaan, eli myös ohjelmoinnin opetukseen. Ohjelmointi-kontekstissa tämä tarkoittaa, että vaikka sisäinen motivaatio on pääsääntöisesti suurempi tekijä menestyksen osalta, myös ulkoisen motivaation tulee olla kohdallaan. Kuten Ryan ja Deci esittävät, mikäli jokin ulkoinen motivaattori saadaan esitettyä niin, että opiskelija ottaa sen omakseen, ja samaistuu siihen, on tämäkin voimakas motivaattori. Esimerkiksi kurssiarvosanan voidaan katsoa olevan ulkoinen motivaattori, mutta jos opiskelija mieltää kurssiarvosanan ja -menestyksen tärkeäksi osaksi omia työllistymismahdollisuuksiaan, hän toimii motivoituneesti saadakseen hyvän kurssiarvosanan. Motivaatioteorioiden hyödyntäminen oppimisprosessissa ei siis tarkoita välttämättä jonkin viitekehyksen tai -järjestelmän käyttöä, vaan niiden avulla pystytään myös miettimään opetusprosessia kokonaisuutena, ja sitä kautta vaikuttamaan oppimiseen positiivisesti.

Myös Salmela-Aron ja Nurmen (2005) katsaus moderniin motivaatiopsykologiaan antaa hyviä eväitä ohjelmoinnin opiskelun tarkastelemiseen. He korostavat erityisesti sitä, miten pelkkä halu oppia ei vielä tarkoita, että oppimista tapahtuisi ja että sitä tehtäisiin, vaan toiminnan pitää myös olla tämän halun mukaista. Käytännössä tämä tarkoittaa sitä, että jos opiskelijalla on halu oppia ohjelmoimaan, se ei vielä itsessään riitä oppimiseen, vaan opiskelijan tulee myös tavoitteellisesti toimia tämän saavuttaakseen. Tässä korostuu siis juuri Salmela-Aron ja Nurmen korostama kognitiivinen puoli

motivaatiosta: mikäli koetaan halua tehdä jotain, tulee siihen tietoisesti myös kohdentaa voimavaroja. Ohjelmoinnin opiskelussa tämä on erityisen tärkeää suuren työmäärän takia, joka ohjelmoinnin oppimiseen vaaditaan. Modernissa motivaatioteoriassa esitetään myös, miten henkilön omat suunnitelmat vaikuttavat tämän motivaatioon: ohjelmoinnin oppimisen tulisi siis olla jotenkin hyödyllistä suhteessa henkilön itselleen asettamiin tavoitteisiin, tai esimerkiksi tulevaisuuden työpaikkaan.

Jotta motivaatioon pystyttäisiin käytännössä vaikuttamaan, pitäisi sitä kuitenkin pystyä myös käsittelemään jotenkin konkreettisesti, eikä vain teoriatasolla. Motivaation tutkiminen ja mittaaminen ovat kuitenkin verrattain vaikeita tehtäviä (Salmela-Aro, Nurmi Jari-Erik 2005). Vaikka esimerkiksi Nikula ym. (2011) onnistuivat parantamaan kurssisuoriutumista motivaatioon keskittyvillä menetelmillä, tätä muutosta mitattiin pitkälti kurssisuoriutumisen parantumisena, ei niinkään suoraan motivaation muutoksena. Siitä huolimatta, että motivaatiota on itsessään vaikea mitata, kurssisuoriutumisen ja oppimisen parantamiseksi on kuitenkin olemassa nykyisin useita erilaisia keinoja, kuten erilaiset uudet opetusmetodit, esimerkiksi pariohjelmointi, tai kokonaan uuden oppimisteorian soveltaminen, kuten konstruktivismin. Myös juuri äsken esitelty CS Principles -viitekehys tarjoaa tähän eväitä ja työkaluja. Ajatus motivaatiosta ja sen lisäämisestä voidaan kuitenkin ajatella olevan näiden kaikkien pohjalla ainakin jollain tasolla, oli motivaatio sitten ulkoista tai sisäistä, omaa halua oppia tai tutkinto-ohjelman sanelemaa. Olisi siis hyödyllistä niin opiskelijoille kuin opettajillekin, jos oppilaiden motivaatiota pystyttäisiin tarkkailemaan: esimerkiksi jos jo kurssin alussa huomataan oppilaan motivaation olevan heikkoa, pystyttäisiin tähän kenties puuttumaan ajoissa. Myös erilaisten uusien opetusmetodien toimivuutta pystyttäisiin arvioimaan ja mittaamaan paremmin, jos niistä olisi saatavilla kvantitatiivista dataa. Tässä tutkielmassa on pyritty luomaan tämän kaltainen mittari, joka antaisi mahdollisuuden mitata ja tarkkailla oppilaiden motivaatiota niin ensimmäisellä ohjelmointikurssilla, kuin myös myöhemmillä kursseilla. Koska motivaation rooli ohjelmoinnin oppimisessa ja opiskelussa on suuri, sen konkreettisempi hahmottaminen auttaisi luonnollisesti paljon.

5 TUTKIMUSJÄRJESTELYT

Tämän tutkielman tutkimus toteutettiin Turun yliopistossa, tulevaisuuden teknologioiden laitoksella. Kohderyhmänä oli ”Algoritmien ja ohjelmoinnin peruskurssille” (jatkossa käytetään lyhennettä AOP) osallistuneet opiskelijat. Kurssi luetaan ensimmäiseksi ohjelmointikurssiksi (CS1-kurssi), ja se on Turun yliopiston tietojenkäsittelytieteiden opiskelijoille pakollinen kurssi. Kyseinen kurssi on suunniteltu toteutettavaksi ensimmäisen vuoden syksyllä. Kurssilla saattaa olla myös vanhemman vuosikurssin tietojenkäsittelytieteiden opiskelijoita, jotka eivät ole saaneet kurssia suoritettua ensimmäisellä kerralla. Kurssin voivat suorittaa myös muiden tiedekuntien opiskelijat. Tämä kurssi on siis ensimmäinen varsinainen ohjelmointikurssi, jonka vasta-aloittaneet opiskelijat käyvät. Kurssilla käytetään ViLLE-oppimisjärjestelmää, jonka avulla tehdään suurin osa kurssiin liittyvistä suorituksista: luentotehtävät, tutoriaalit, tentti sekä motivaatiokysely. Perinteisemmin paperilla suoritettiin vain kurssin ”kotitehtävät” eli demonstraatiot. Opiskelijoiden vastaamishalukkuutta motivaatiokyselyyn nostettiin antamalla kyselyyn vastanneille pisteitä, jotka edesauttavat kurssin suorittamisessa. Pisteitä annettiin kuitenkin sen verran vähän, että sen käytännön vaikutus kurssin suorittamisessa oli hyvin pieni. Näin otokseen saatiin kuitenkin hyvin suuri osa kurssiin osallistuneista opiskelijoista. Seuraavaksi esitellään tutkimuksen kohteena ollutta ensimmäistä ohjelmointikurssia AOP:ia, sekä sen toteuttamisessa pitkälti käytettyä ViLLE-oppimisjärjestelmää tämän tutkielman puitteissa. Lopuksi esitellään tutkimuksessa käytettyjä metodeja, sekä kuvataan tutkimuksen kohderyhmää hieman tarkemmin.

5.1 Kurssin esittely

Kuten sanottua, tutkimuksen kohteena oli AOP:lle osallistuvat opiskelijat. AOP on Turun yliopistossa ensimmäinen ohjelmointikurssi (CS1), jonka osallistujat ovat pitkälti tietojenkäsittelytieteen opiskelijoita. Kurssin pystyvät suorittamaan kuitenkin myös muut kuin tietojenkäsittelytieteiden pääaineopiskelijat. AOP:lla tavoitteena on tutustua olio-ohjelmointikielen peruskäsitteisiin ja -rakenteisiin, opetella ohjelmoinnissa tarvittavaa algoritmista ajattelua, sekä hankkia editorin ja kääntäjän kanssa

työskentelyyn riittävä ohjelmointitaito. Keskeisenä tavoitteena on oppia laatimaan pieniä, toimivia, 1-3 aliohjelman laajuisia sovelluksia, jotka perustuvat peräkkäisyyteen, valintaan ja toistoon. Kurssilla käytetään oppimisen apuvälineenä ja esimerkkiohjelmointikielenä Javaa. Käsiteltäviä aiheita ovat ohjelmien kirjoittaminen editorilla, kääntäjän käyttäminen, hyvät ohjelmointitavat, muuttujat, viittaukset, taulukot, peruskontrollirakenteet, syöttö ja tulostus, algoritminen ongelmanratkaisu, modulaarisuus, metodit, alku- ja loppuehdot sekä rekursio. Osaa näistä ollaan käyty teoriatasolla läpi jo yhdellä aikaisemmalla kurssilla, mutta vain pientä osaa ja hyvin perustavalla tasolla. (Tulevaisuuden teknologioiden laitos 2017) AOP:ssa tutustutaan myös olioihin ja niiden käyttöön, mutta omien olio-ohjelmointia edustavien luokkien muodostaminen ei kuulu vielä tämän kurssin aihepiiriin, vaan niitä esitellään vasta AOP:n jatkokurssilla. AOP:n aikana opiskelijalta edellytetään teorian opiskelun lisäksi kurssin alusta asti jatkuvaa, itsenäistä käytännön harjoittelua kääntäjällä.

Kurssin toteutukseen sisältyy luentoja, harjoitustehtäviä, tutoriaaleja, demonstraatioita, tentti, sekä itsenäistä työskentelyä (Tulevaisuuden teknologian laitos 2018). Tarkempi sisällön ja työmäärän jakautuminen esitetään taulukossa 2. Kurssilla käytettävä ViLLE-opetusjärjestelmä mahdollistaa useiden erityyppisten harjoitusten käytön, niiden

Taulukko 2 AOP:in osuuksien tuntijakauma

Luennot	**14h**
Tutoriaalit	**14h**
Demonstraatiot	**8h**
ViLLE-tehtävät, tentti & itsenäinen opiskelu	**107h**

automaattisen tarkastuksen, datan keruun kaikista opiskelijoiden suorituksista, ja lisäksi se pystyy toimimaan yleisesti kurssin ”kotisivuna”, josta löytyy kurssimateriaalit sekä ajankohtaiset tiedotteet. ViLLE:n vahvuuksia ovat juurikin lukuisat erilaiset sekä täysin

opettajan muokattavissa olevat harjoitukset, jotka hyödyntävät paljon visualisointia, ja joita opiskelija pystyy suorittamaan koska vain selaimensa kautta. Lisäksi automaattinen tarkastus helpottaa opettajien työtä ja sen on todettu myös parantavan oppimistuloksia (Laakso 2010). ViLLE kerää myös talteen opiskelijoiden suoritukset, joita pystytään käyttämään esimerkiksi tutkimustarkoituksiin tai oppilaiden suoriutumista tarkasteltaessa ja analysoitaessa. ViLLE-järjestelmän käyttöön kannustetaan myös omatoimisessa opiskelussa, sillä se tarjoaa hyvän ympäristön opetteluun: sen avulla on helpompi suorittaa yksinkertaisia Java-ohjelmia kuin tavallisilla ohjelmointiympäristöillä (Erkki Kaila ym. 2015), ja siinä hyödynnetään runsaasti visualisointia, jonka on puolestaan todettua jo aikaisemmin auttavan oppimisessa. ViLLE-järjestelmä on siis avainasemassa AOP:lla aivan syystä.

Kurssin kotona suoritettavat harjoitustehtävät tehdään kontaktiopetuksen ulkopuolella ViLLE-järjestelmää käyttäen. Kurssilla suoritettavat tutoriaalit tarkoittavat AOP:n tapauksessa ryhmäopetussessioita, joissa koko kurssi kokoontuu yhteen ja suorittaa pienissä ryhmissä harjoituksia opettajien valvomina. Luennot ja tutoriaalit muodostavat kokonaisuuden pareittain siten, että viikossa on aina ensin yksi luento jostain tietystä aiheesta, jonka osaamista viikon toisella tapaamiskerralla, eli käytännössä tutoriaaleissa, syvennetään ja sovelletaan. Ryhmät koostuvat yleensä kahdesta henkilöstä, ja tutoriaalit suoritetaan suoraan ViLLE-järjestelmässä. Nämä ViLLE-tutoriaalit ovat yleensä yhdistelmä automaattisesti arvioitavia tehtäviä, sekä erilaisia oppimateriaaleja, kuten kuvia, videoita, tai luentomonisteita. Tutoriaalien aikana itse harjoitusta koneella kirjoittava henkilö vaihtuu välillä, mutta ryhmä yhdessä miettii ratkaisuja tehtäviin: käytännössä siis pyritään toteuttamaan pariohjelmointia oppilaiden määrän puitteissa. Tarkoituksena on, että opiskelijat pystyvät ryhmissä ja mahdollisesti ryhmiä yhdistämällä suorittamaan tehtävät alusta loppuun, ja opettajan rooli on vain valvoa ja loppu viimeksi auttaa niitä, jotka eivät ryhmäytymisestä huolimatta saa tehtäviä tehdyiksi. Tutoriaalit ovat pakollinen osa kurssia, ja tutoriaalitilaisuuksissa on myös läsnäolopakko. Hyvin suoritetut tutoriaalit vaikuttavat positiivisesti loppuarvosanaan kurssilla kerättävän yhteispistemäärän muodossa. (ViLLE Team 2015) Kaila ym. (2015) ovat todenneet näiden aktiivista oppimista hyödyntävien tutoriaalien vaikuttavan

positiivisesti kurssin läpäisyprosenttiin, ja myös oppilaat ovat kokeneet ne positiiviseksi muutokseksi verrattuna pelkkiin luentoihin.

Kurssilla suoritettavat demonstraatiot ovat käytännössä pakollisia kotitehtäviä: ne tehdään paperille, ja esitellään yhteisissä palautussessioissa viikoittain. Demonstraatiot suoritetaan AOP:lla suurien ryhmäkokojen vuoksi hieman eri tavalla kuin normaalisti: Jokaisessa palautussessiossa paikallaolijat jaetaan pienempiin ryhmiin, jonka jälkeen jokainen pienryhmä tarkistaa ensin keskenään omat ratkaisunsa. Tämän jälkeen yksi tällainen pienryhmä esittelee yhden tehtävän ratkaisun koko paikalla olevalle demoryhmälle. AOP:lla demonstraatiot ovat käytännössä paperilla tehtäviä ohjelmointiharjoituksia. Demonstraatioiden on tarkoitus tukea oppimista ja testata, miten hyvin opiskelija on käsitellyn aiheen sisäistänyt ja miten hän osaa sitä soveltaa. Vaikka demonstraatiot AOP:n tapauksessa tehdäänkin paperilla, niiden suoritusmerkinnät ladataan ViLLE:en, josta niitä pystyvät seuraamaan niin opettajat kuin oppilaatkin. Demonstraatiopisteitä saa myös osallistumisesta sekä demonstraatiotehtävien esittämisestä, eikä siis pelkästään suorittamalla tehtävät ajallaan. Tehtävistä saadut pisteet muodostavat silti suurimman osan demonstraatioista saatavista yhteispisteistä. Opettaja pystyy myös itse vaikuttamaan siihen, kuinka paljon pisteitä mistäkin aktiviteetista saa.

Myös kurssin tentti suoritetaan ViLLE-järjestelmää käyttäen. Tämä tuo hyödyn muun muassa automaattisen tarkistuksen kautta, jonka lisäksi myös manuaalisesti tarkistettavat tehtävät on helpompi tarkastaa esimerkiksi kirjoituksen ollessa selkeää. Huomattavan edun verrattuna paperiin ja kynään tuo myös se, että ViLLE:n avulla ohjelmointitehtävissä oppilaat pystyvät kokeilemaan reaaliaikaisena erilaisia ratkaisuja ja testaamaan vastauksiensa toimivuutta, niin kuin se olisi mahdollista oikeassakin elämässä (Erkki Kaila ym. 2015).

5.2 Käytetyt metodit

Tämän tutkielman fokuksena oleva motivaatiokysely toteutettiin ViLLE-järjestelmää käyttäen. Tämä helpotti kyselyyn vastaamisen lisäksi myös itse datan keruuta ja edelleen sen analysointia. Suoritettu motivaatiokysely toteutettiin Turun yliopiston

ensimmäisellä ohjelmointikurssilla AOP:lla aivan kurssin alussa, ja se tehtiin nyt ensimmäistä kertaa. Sama motivaatiokysely päätettiin tehdä myös kurssin loputtua, jolloin kyselyn tuloksia pystyttäisiin vertailemaan. AOP muodostaa yhdessä Tietojenkäsittelyn perusteet -kurssin sekä Olio-ohjelmoinnin perusteet -kurssin kanssa yhtenäisen kurssikokonaisuuden: tämä mahdollisti sen, että kurssin lopuksi suoritettava motivaatiokysely pystyttiin toteuttamaan AOP:a seuraavan Olio-ohjelmoinnin perusteet -kurssin (jatkossa käytetään lyhennettä OOP) alussa. Osallistujakunta näillä kursseilla on myös pitkälti sama juurikin luonnollisen kurssijatkuvuuden takia. Kyselyn suorittaminen AOP:a seuraavalla OOP-kurssilla mahdollisti myös sen, että opiskelijat olivat saaneet AOP:n loppuarvosanansa loppukyselyä tehdessään, jolla on todennäköisesti ollut vaikutus koettuun omaan motivaatioon. Loppukyselynä käytettiin siis lähes samaa motivaatiokyselyä, jota käytettiin kurssin alussakin: erona oli yksi loppukyselyyn lisätty kysymys, joka kysyi kokonaisuudessaan oppilaan kokemaa asennemuutosta AOP:n jälkeen. Molemmissa kyselyissä opiskelijoiden tiedot anonymisoitiin niin, ettei tutkimuksesta pysty nimeämään yksittäisiä henkilöitä.

Suoritetun kyselyn tarkoituksena oli mitata oppilaiden asenteita ja motivaatio yleisesti ohjelmointia kohtaan. Kyselyn perustana käytettiin Wong K. Y. ja Chen Q. (2012) muodostamaa asennekyselyä matematiikan opiskelijoille, ja tämä kysely on edelleen käännetty ja muokattu vastaamaan ohjelmointia. Alkuperäinen kysely pohjautuu 57 kohtaiseen ALM-kyselyyn (Attitudes toward Learning Mathematics, Asenteet matematiikan opiskelua kohtaan), jonka Wong K. Y. ja Chen Q. olivat onnistuneet lyhentämään 24-kohtaiseksi. Lyhentämisestä huolimatta he olivat onnistuneet säilyttämään alkuperäisen kyselyn psykometriset tekijät: tämän saavuttamiseksi he olivat käyttäneet EFA-menetelmää (Exploratory Factor Analysis, eksploratiivinen faktorianalyysi). Lopuksi he validoivat tekemänsä lyhentämisen sekä löydetyt psykometriset tekijät CFA-menetelmällä (Confirmatory Factor Analysis, konfirmatorinen faktorianalyysi). Nämä menetelmät ovat laajalti käytössä, ja tukevat hyvin toisiaan, kuten myös Suhr D. (2006) toteaa: EFA-analyysi tutkii ensin, millaisia yhteisiä tekijöitä datasta löytyy, ja erottelee sitten datan näiden tekijöiden mukaan. Juuri Wong K. Y. ja Chen Q. käyttämän kyselyn puitteissa tämä tarkoitti sitä, että

alkuperäiselle kyselylle tehtiin EFA-analyysi, jolla pystyttiin selvittämään sen pohjalla olevat tekijät. Tämän jälkeen käytetty CFA-analyysi puolestaan on tilastollinen menetelmä, jonka tarkoituksena on vahvistaa jokin datasetin tekijärakenne. Tämä jälleen tarkoittaa Wong K. Y. ja Chen Q. kyselyn puitteissa sitä, että EFA-analyysillä löydetyt tekijät vielä vahvistettiin CFA-analyysilla. Näin jäljelle jäi kysely, joka pitää sisällään samat oleelliset tekijät kuin alkuperäinenkin kysely, mutta sen pituus onnistuttiin lyhentämään puoleen. Wong K. Y. ja Chen Q. testasivat uutta kyselyään myös käytännössä, ja totesivat uuden ja lyhyemmän kyselyn olevan verrannollinen vanhaan, pidempään kyselyyn.

Tarkastellaan kyselystä löydettyjä psykometrisiä tekijöitä vielä hieman tarkemmin; samat elementit ovat myös tässä tutkielmassa käytetyn kyselyn pohjana. Wong K. Y. ja Chen Q. (2012) löysivät siis EFA-menetelmän avulla alkuperäisestä kyselystä kuusi psykometrista tekijää, jotka he sitten nimesivät ja tiivistivät omiksi kategorioikseen:

1. Kategoria 1: Vastausten tarkistaminen (Checking solutions): tämän kategorian kysymykset kysyvät, kuinka hyvin ja millä tavoin oppilas suhtautuu omien tehtyjen tehtäviensä tarkastamiseen.

2. Kategoria 2: Itsevarmuus (Confidence): kartoittaa oppilaan omaa käsitystä siitä, miten hyvin tämä pystyy ratkaisemaan matemaattisia tehtäviä.

3. Kategoria 3: Nauttiminen (Enjoyment): tarkoituksena on saada kuva siitä, kuin paljon oppilas nauttii matematiikan opiskelusta.

4. Kategoria 4: Tietotekniikan käyttö (Use of IT): mittaa miten paljon oppilaat kokevat tietotekniikan voivan auttaa matematiikan opiskelussa.

5. Kategoria 5: Useampi vastausvaihtoehto (Multiple solutions): kartoittaa kuinka taipuvaisia oppilaat ovat etsimään useampia vastauksia käsillä olevaan ongelmaan.

6. Kategoria 6: Matematiikan hyödyllisyys (Usefulness of mathematics): tarkoituksena mitata, kuinka hyödyllisenä ja yleispätevänä oppilaat pitävät matematiikkaa heidän jokapäiväisessä elämässään.

Wong K. Y. ja Chen Q. (2012) muodostama kysely perustuu siis näihin kuuteen kategoriaan. Näiden pohjalta he muodostivat neljä kysymystä kustakin kategoriasta, jolloin kyselyn pituus saatiin sopivammaksi. Kuten voidaan huomata, kategoriat perustuvat vahvasti alkuperäiseen ympäristöönsä, eli matematiikkaan; tämän tutkimuksen yksi tehtävistä olikin luoda kyseisestä kyselystä ohjelmoinnin opiskelua mittaava kysely. Suurimmaksi osaksi tämä on tarkoittanut matematiikka-käsitteiden vaihtamista ohjelmointi-käsitteiksi. Kategorian pohjalla vaikuttava psykometrinen tekijä ei tässä siis muutu, vaan ainoastaan sen kohde. Esimerkiksi ”matematiikan hyödyllisyys”-kategoria on vaihtunut ”ohjelmoinnin hyödyllisyys”-kategoriaan, jolloin ainoastaan kohde, matematiikka ja ohjelmointi, muuttuu, mutta ei sen syvempi merkitys, eli kokeeko oppilas sen hyödyllisenä. Suurimman muutoksen alkuperäiseen verrattuna on kokenut ”Tietotekniikan käyttö”-kategoria, jonka kysymykset pohjautuvat nyt paljon käytetyn ViLLE-järjestelmän hyödyllisyyteen. Kategorioita on myös lisätty kahdella: mukaan ovat tulleet seitsemäs kategoria ”Motivaatio” sekä kahdeksas kategoria ”Matematiikka”. Itse alkuperäiset kysymykset/väittämät on käännetty mahdollisimman suoraan englannista suomeksi, kuitenkin ottaen huomioon, että tuleva kysely on tarkoitettu ohjelmointikeskeiseksi; esimerkiksi englanninkielinen väittämä ”I find mathematics easy” on käännetty suomenkieliseksi sekä ohjelmointikeskeiseksi väittämäksi ”Ohjelmointi on helppoa.”

Uuden ”Motivaatio”-kategorian ideana on tuoda mukaan muutama kysymys suoraan motivaatiota ajatellen: pohja näille kysymyksille on otettu jo aikaisemmin mainitusta Salmela-Aron ja Nurmen (2005) kirjasta ”Motivaatiopsykologian perusteet – Mikä meitä liikuttaa?” Kirjassa dosentti Petteri Niitamo kertoo tunne- ja tietoperäisestä motivaatiosta, jonka on todettu olevan myös pätevä jako motivaatiolle. Se pohjaa ideansa McClellandin ym. (Pardee R.L. 1990) vuonna 1989 esittämään jakoon implisiittisiin ja eksplisiittisiin motiiveihin: tarkoituksena on jakaa motiivit piileviin,

tietoisuudelle ja ulkoiselle tarkastelulle etäisempiin, kenties opittuihin, ominaisuuksiin, sekä henkilön itsensä raportoimiin ja löydettävissä olevin ominaisuuksiin. Niitamo kuitenkin argumentoi, että jaon tulisi perustua ennemminkin psyykkisten prosessien, tunteiden ja kognition eroihin. Hän siis kutsuukin näitä kahta juurikin tunne- sekä tietoperäiseksi motivaatioksi.

Niitamon (Salmela-Aro, Nurmi Jari-Erik 2005) esittämä jako korostaa erityisesti sitä, miten henkilöllä voi olla tunnemotivaatiota, mutta ettei se itsessään vielä johda mihinkään toimintaan, vaan kuvaa enemmänkin mitä henkilön ”mielessä liikkuu.” Kun halutaan ymmärtää ihmisen varinaista toimintaa, tulisi keskittyä siihen, miten tunneperäinen motivaatio kanavoituu tiedollisille ja toiminnallisille uomille. Tunteiden ja kognitiivisen toiminnan ero ja yhteys on siis oleellinen. Hän mainitsee myös esimerkin, jossa tunneperäisesti suoriutumismotivoituneet opiskelijat eivät kaikki toimineet suoritushakuisesti, vaan ainoastaan ne, jotka olivat tiedostaneet kurssimenestyksen olevan osa menestystä tulevalla työuralla.

Tästä lähestymistavasta juontaa juurensa siis tässä kyselyssä käytetyt ”Motivaatio”-kategorian kysymykset: kysymyksen 7, ja osin ”Hyödyllisyys”-kategorian kysymyksen 22, tarkoituksena on kartoittaa opiskelijan tunneperäistä motivaatiota, ja kysymysten 15 ja 23 tietoperäistä motivaatiota. Kysymykset 15 ja 23 kysyvät tietoperäistä motivaatiota hieman eri kulmista: oppilas voi esimerkiksi tiedostaa ohjelmointitaidon merkityksen työelämässä, tai hän voi olla kiinnostunut ohjelmointitaitojensa kehittämisestä enemmän henkilökohtaisena taitona. Kuitenkin molemmat näistä ovat tiedostettuja asioita, joihin oppilas pystyy tunneperäisen motivaationsa kanavoimaan. ”Motivaatio”-kategorian taustalla on siis hypoteesi, että mikäli oppilaalla on sekä korkea tunneperäinen motivaatio (kysymykset 7 ja 22) että tietoperäinen motivaatio (kysymykset 15 ja 23), tämä suorituskyky kurssilla on hyvä. Kysymyksiä voidaan ajatella myös Ryan ja Decin (2000) esittämän sisäisen ja ulkoisen motivaation kautta, joka heijastelee samoja ominaisuuksia kuin äsken mainittu tunne- ja tietoperäinen motivaatio. Etenkin ulkoisen motivaation erilaisten ilmentymien rooli korostuu, kun niitä ei ajatellakaan pelkästään henkilön ulkopuolelta tulevina kannusteina, vaan että

henkilö voi suhtautua tällaisiinkin asioihin motivoituneesti, jos hän kokee nämä asiat itselleen tärkeiksi ja hyväksyy niiden arvot.

Myös matematiikasta haluttiin ottaa mukaan oma kategoriansa, sillä matemaattisilla taidoilla on todettu olevan vahva korrelaatio ohjelmoinnissa pärjäämiselle: esimerkiksi Gomes A. ym. (2006) ovat tehneet tämän saralta tutkimusta. He toteavat, että yleisesti ottaen heikkoudet matematiikassa heijastavat heikkoihin ongelmanratkaisutaitoihin, jotka puolestaan vaikeuttavat ohjelmoinnin oppimista. Tämän matematiikkakategorian tarkoituksena on siis kartoittaa opiskelijan omaa kokemusta matematiikasta ohjelmointia helpottavana tekijänä, sekä yhden kysymyksen verran hahmottaa tämän matemaattisia valmiuksia. Esimerkiksi, kuten Gomes A. ym. mainitsevat, etenkin sanallisten ratkaisujen muuntaminen matemaattiseen muotoon on yksi ongelma, johon opiskelijat jatkuvasti törmäävät: tätä on pyritty selvittämään erityisesti kysymyksellä 24.

Kategorioiden mukaan tehtyjen kysymysten määrää muokattiin myös hieman: aiemmin kullakin kategorialla oli neljä kysymystä, mutta nyt kun kategorioita ja kysymyksiä tuli lisää, päätettiin alkuperäisiä kysymyksiä hieman vähentää. Nyt uudessa kyselyssä kutakin kategoriaa kohden on pääsääntöisesti kolme kysymystä, poikkeuksena kategoriat ”Vastausten tarkistaminen” sekä ”Nauttiminen”, joissa kysymyksiä on neljä. Näiden kysymysten lisäksi loppuun on lisätty myös kolme niin sanottua jokerikysymystä, jotka eivät istuneet mihinkään tiettyyn kategoriaan, mutta jotka tuntuivat järkeviltä kysymyksiltä pitää kuitenkin mukana. Nämä kysymykset eivät kuitenkaan ole sen suuremmin tutkimuksen tarkastelun kohteena, vaan ovat mukana lähinnä yleisen mielenkiinnon takia. Kuten alkuperäisessäkin kyselyssä, kysymykset esiintyvät järjestyksessä peräkkäin yksi kerrallaan ja kategorioittain: ensimmäinen kysymys on siis kategoriasta yksi, toinen kysymys kategoriasta kaksi, ja niin edelleen, kunnes kierros käynnistyy kahdeksannen kysymyksen jälkeen uudestaan. Kohdat 25 ja 26 on omistettu ”Vastausten tarkistaminen”- sekä ”Nauttiminen”-kategorioiden kysymyksille, ja loput kohdat 27, 28 ja 29 ylimääräisille kysymyksille. Tässä tutkimuksessa käytetyssä loppukyselyssä on viimeisenä, 30. väitteenä, väite ”Asenteeni ohjelmointia kohtaan on muuttunut AOP-kurssin alusta.” Tämän tarkoituksena oli

luonnollisesti kysyä, miten opiskelijan asenteet ovat muuttuneet kurssin jälkeen. Tämä 30. kysymys on myös niin sanottu jokerikysymys, eikä kuulu mihinkään erityiseen ryhmään. Kysely löytyy tutkielman liitteistä (Liite 1).

Kuten jo aikaisemmin mainittiin, suurin osa nyt luodun ohjelmointikyselyn kysymyksistä pystyttiin muuntamaan suoraan alkuperäisestä kyselystä: kaiken kaikkiaan 19 koko kyselyn 24:stä kysymyksestä pystyttiin kääntämään. Näistä kaksi, alkuperäisen kyselyn numerot 12 ja 20, uuden kyselyn 6 ja 18, eivät ole täysin suoria käännöksiä niin kuin muut, mutta vastaavat idealtaan pitkälti samaa asiaa. Kyselystä jätettiin siis pois kysymykset 2, 4, 10, 18 ja 23 (alkuperäisen kyselyn numeroina). Nämä tippuivat pois käytännössä sen takia, että niiden kääntäminen suoraan ei olisi ollut mielekästä. Myös alkuperäisten kysymysten määrän karsiminen neljästä per kategoria kolmeen per kategoria mahdollisti joidenkin kysymysten pois jättämisen, ja nämä vaikeasti käännettävissä olevat kysymykset olivat näin ollen ensimmäisiä, jotka pudotettiin pois. Kyselyssä käytettiin myös käänteisiä/negatiivisa kysymyksiä, kuten ”Ohjelmointi on tylsää” tai ”En pidä matematiikkaa tärkeänä osana ohjelmointia”. Näiden kysymysten tarkoitus on erottaa joukosta sellaiset tapaukset, jotka ovat vastanneet kyselyyn pelkästään sattumanvaraisesti. Molemmissa kyselyissä, niin Wong K. Y. ja Chen Q. (2012) alkuperäisessä kuin tässä ohjelmointikyselyssä, on käytetty Likert-tyyppistä viiden kohdan arviointiasteikkoa, jossa arvoasteikot olivat ”Täysin eri mieltä”, ”Osittain eri mieltä”, ”Ei eri eikä samaa mieltä”, ”Osittain samaa mieltä” ja ”Täysin samaa mieltä”.

Tässä tutkielmassa esitettyä ja osittain luotua kyselyä ei ole validoitu Wong K. Y. ja Chen Q. (2012) tapaan EFA- ja CFA-analyyseilla, koska kysely pohjautuu niin suurelta osin alkuperäiseen kyselyyn, jolle nämä analyysit oli jo tehty. Kysely myös suoritettiin nyt ensimmäistä kertaa, joten kyseisiä analyysejä ei olisi voitu aikaisemmin edes suorittaa. Kyselylle tehtiin kuitenkin jälkeenpäin EFA-analyysi, käytännössä yliopiston sekä avoimen yliopiston alkukyselylle, sekä yliopiston OOP-kurssilla tehdylle loppukyselylle. Avoimen yliopiston loppukysely jäi EFA-analyysin ulkopuolelle Kayser-Mayer-Olkin-testin (KMO-testi) perusteella, jossa sen pistemäärä jäi alle

vaaditun 0,6:en; yliopiston alkukyselyn KMO-tulos oli 0,854, avoimen yliopiston alkukyselyn KMO-tulos oli 0,780, ja yliopiston loppukyselyn KMO-tulos oli 0,848. Tehdyistä faktorianalyyseistä kuitenkin paljastui, että faktoreita olisi vain 5-7, tässä tutkimuksessa käytetyn kahdeksan sijaan. Tässä tutkimuksessa esitettyä kyselyä olisi siis syytä vielä muokata ja parannella, esimerkiksi kysymyksiä muuttamalla. Tähän otetaan kantaa vielä tutkielman lopussa.

Tutkimuksessa käytettyjen kahdeksan kategorian reliabiliteettia testattiin Cronbachin alfan avulla. Vanhaa kyselyä oli ilmeisesti muokattu hieman liikaa, ja kahdeksasta kategoriasta 4-6 sai hyväksyttävät pisteet, riippuen hieman, oliko kyseessä esimerkiksi yliopisto-opiskelijoille tehty kysely, vai avoimen yliopiston opiskelijoille tehty kysely. Positiivista kuitenkin oli, että riippumatta siitä, oliko kysely tehty esimerkiksi yliopisto-opiskelijoille vai avoimen yliopiston opiskelijoille, samat neljä kategoriaa näyttivät aina saavan tarpeeksi pisteitä läpäistäkseen reliabiliteettitarkastelun (lukuun ottamatta yhtä kohtaa, jonka pistemäärä oli 0,593, hyväksyttävän rajan ollessa >0,6). Nämä neljä kategoriaa olivat: 1 ”Tehtävien tarkistaminen”, 2 ”Itseluottamus”, 3 ”Nauttiminen”, sekä 7 ”Motivaatio”. Myös tämän perusteella tässä tutkimuksessa käytetyn kyselyn kategorioita pitäisi hieman miettiä uudestaan ja parannella. Myös tähän otetaan kantaa vielä tutkielman lopussa.

5.3 Kohderyhmä

Tutkielman kysely kohdistettiin Turun yliopiston ensimmäiselle ohjelmointikurssille loppusyksyllä 2017. Yliopistossa kurssi kuuluu osaksi tietojenkäsittelytieteiden koulutusohjelmaa ja perusopintoja, ja avoimessa yliopistossa se on osa kaikkien suorittamaa avoimen yliopiston opintopakettia. Kurssi on siis ainakin yliopistossa ensimmäinen ohjelmointikurssi, jolle opiskelija osallistuu. Tutkimuksen kohderyhmänä olivat siis ”Algoritmien ja ohjelmoinnin peruskurssin” osallistujat, niin yliopisto-opiskelijat kuin avoimen yliopistonkin opiskelijat. Yliopiston kurssille oli rekisteröitynyt yhteensä 286 opiskelijaa, joista 214 (N=214) vastasi alkukyselyyn, jolloin vastausprosentti oli 74,8%. Avoimen yliopiston kurssilla oli yhteensä 74 opiskelijaa, joista 50 (N=50) vastasi kyselyyn, jolloin vastausprosentti oli 67,6%. Loppukysely

suoritettiin AOP-kurssia seuraavan kurssin, Olio-ohjelmoinnin perusteet -kurssin alussa, koska suurin osa AOP-opiskelijoista suorittaa heti perään tämän OOP-kurssin, ja motivaatio tehdä kysely uudestaan oli oletetusti korkeampi uudella kurssilla, kuin se olisi ollut AOP-kurssin lopussa. OOP-kurssilla tehtyyn loppukyselyyn osallistui 176 (N=176) yliopisto-opiskelijaa, ja avoimen yliopiston puolelta 37 (N=37) opiskelijaa. Yliopistossa kurssille rekisteröityneitä oli 285, joten loppukyselyn osallistujaprosentti oli 61,8%, ja avoimen yliopiston OOP-kurssille rekisteröityneitä oli 74, joten avoimen loppukyselyn osallistujaprosentti oli 50%. Tämän loppukyselyn alhaisempi vastausprosentti todennäköisesti johtuu hieman vaihtuneesta osallistujajoukosta, jossa osa oli ehtinyt suorittaa AOP-kurssin jo joskus aikaisemmin, joten motivaatio täyttää ylimääräinen kyselyä ei oletettavasti ollut suuri.

Osallistujajoukko oli todennäköisesti hyvin heterogeeninen, ja opiskelijoiden taustat voivatkin vaihdella suuresti; osalle kurssi voi olla ensi kosketus kunnon ohjelmointiin, ja joillekin valmiiksi ohjelmointia osaaville kurssi on vain yksi pakollinen kurssi. Kurssin osallistujille ei ollut pakollista osallistua kyselyyn, mutta kuten jo aikaisemmin on todettu, kyselyyn vastaamisesta palkittiin pienellä määrällä pisteitä, jotka puolestaan vaikuttavan kurssin arvosanaan sekä läpipääsyyn.

6 TULOKSET

Kyselyssä käytetyt negatiiviset kysymykset toimivat kontrollikysymyksinä, joiden avulla koitettiin karsia pois sellaiset vastaajat, jotka eivät selvästikään vastanneet säännönmukaisesti, ja jotka todennäköisesti eivät siis esimerkiksi lukeneet kysymyksiä lainkaan. Näin saatiin karsittua otoksesta suurimmat häiriöt pois. Kontrollitarkastelun jälkeen tarkastelujoukoksi jäi alkukyselyn osalta yliopistolle 202 ja avoimelle yliopistolle 46, sekä loppukyselyn osalta yliopistolle 162 ja avoimelle yliopistolle 34. Kaikista alkukyselyyn vastanneista yliopisto-opiskelijoista 135 vastasi vielä loppukyselyyn, ja kaikista alkukyselyyn vastanneista avoimen yliopiston opiskelijoista 27 vastasi vielä loppukyselyyn.

Kyselyn tekeminen ohjelmointikurssilla oli itsessään uusi tapahtuma, ja vaikka kyselyn kohteena ollut kurssi onkin suoritettu pitkälti sähköistä opetusjärjestelmää ViLLEä käyttäen, ei kurssilaisten motivaatiosta ole aikaisemmin saatu tietoa. Kysely oli siis ensimmäinen laatuaan kyseisellä kurssilla, sekä myös yleisesti Turun yliopiston järjestämillä ohjelmointikursseilla. Taulukossa 3 on esitelty, miten motivaatiokyselyn kategorioiden keskihajonnat ja keskiarvot ovat jakautuneet. Tehdyn kyselyn perusteella voidaan todeta, että sekä yliopisto-opiskelijoiden, että avoimen yliopiston opiskelijoiden yhteinen motivaatio on verrattain korkea ensimmäisen ohjelmointikurssin alussa, sekä myös tämän jälkeen (kts. taulukko 3). Myöskään juurikaan minkään mittauskategorian (1-8) pisteet eivät heilahdelleet merkittävästi alku- ja loppukyselyn välillä. Pientä laskua on huomattavissa kurssin jälkeen, mutta muutos on hyvin pientä. Muutoksen pienuus itsessään on huomionarvoista ja positiivinen asia, sillä yleisesti ensimmäistä ohjelmointikurssia pidetään haastavana kurssina, ja suurelle osalle opiskelijoita asiat ovat uusia ja vaikeita.

Taulukko 2 Motivaatiokyselyn keskiarvot ja keskihajonnat kategorioittain, sekä koko kyselystä. Alku- ja loppukyselyihin otettu mukaan sekä avoimen yliopiston että yliopiston opiskelijoiden vastaukset.

	Alkukyselyn Ka	**Alkukyselyn Kh**	**Loppukyselyn Ka**	**Loppukyselyn Kh**
1. Vastausten tarkistaminen	3,83	0,98	3,80	0,91
2. Itsevarmuus	3,58	0,99	3,51	0,90
3. Nauttiminen	4,26	0,90	4,20	0,80
4. Tietotekniikan käyttö	4,38	0,86	4,36	0,80
5. Useampi vastausvaihtoehto	3,48	1,05	3,61	0,96
6. Ohjelmoinnin hyödyllisyys	4,33	0,87	4,25	0,76
7. Motivaatio	4,56	0,78	4,46	0,65
8. Matematiikka	3,98	0,97	3,85	0,95
Keskiarvo kaikista	4,05	0,92	4,01	0,84

Alkukyselyn suurimman keskiarvon 4,56 sai kategoria 7 ”Motivaatio”. Puolestaan alkukyselyn pienimmän keskiarvon 3,48 sai kategoria 5 ”Useampi vastausvaihtoehto”. Näiden kahden kategorian keskiarvojen välinen ero on siis aika suuri. Loppukyselyn osalta suurin keskiarvon oli jälleen kategorialla 7 ”Motivaatio”, keskiarvolla 4,46. Pienimmän keskiarvon sai puolestaan kategoria 2 ”Itsevarmuus”, keskiarvolla 3,51. Huomion arvoista on siis jälleen näiden kahden kategorian välinen ero keskiarvoissa, sekä myös miten pienimmän keskiarvon saanut kategoria vaihtui. Kategorian 2 ”Itsevarmuus” keskiarvon lasku ei ollut mitenkään järin dramaattinen. Vastaavasti kategorian 5 ”Useampi vastausvaihtoehto” keskiarvo nousi, jolla oli alkukyselyn pienin

keskiarvo. Kyseinen kategoria 5 oli myös ainoa kategoria, jonka keskiarvo nousi alkukyselystä loppukyselyyn.

Motivaatiokyselyssä ei löydetty juurikaan eroja yliopiston ja avoimen yliopiston opiskelijoiden motivaation ja asenteiden välillä (kts taulukko 4). Yleinen suunta kuitenkin on, että yliopisto-opiskelijoilla on hieman parempi motivaatio kuin

Taulukko 3 Motivaatiokyselyn keskiarvot kategorioittain, sekä koko kysely yhteensä. Eriteltynä avoimen yliopiston opiskelijoiden sekä yliopisto-opiskelijoiden vastaukset kyselyyn.

	Alkukysely yliopisto	**Alkukysely avoin**	**Loppukysely yliopisto**	**Loppukysely avoin**
1. Vastausten tarkistaminen	3,82	3,87	3,78	3,90
2. Itsevarmuus	3,59	3,52	3,56	3,27
3. Nauttiminen	4,27	4,21	4,20	4,18
4. Tietotekniikan käyttö	4,39	4,31	4,36	4,39
5. Useampi vastausvaihtoehto	3,53	3,25	3,63	3,51
6. Ohjelmoinnin hyödyllisyys	4,35	4,21	4,27	4,18
7. Motivaatio	4,59	4,41	4,49	4,33
8. Matematiikka	4,00	3,86	3,88	3,67
Keskiarvot yhteensä	4,07	3,95	4,02	3,93

avoimen yliopiston opiskelijoilla. Erot ovat kuitenkin pieniä, ja lähes kaikkien kategorioiden osalta muutos pysyi verrattain pienenä. Suurin ero yliopiston ja avoimen yliopiston välillä alkukyselyn osalta oli kategorian 5 ”Useampi vaihtoehto” keskiarvojen kohdalla. Tämä ero tasoittuu loppukyselyssä. Loppukyselyssä suurin ero yliopisto-opiskelijoiden ja avoimen yliopiston opiskelijoiden välillä on kategorian 2

”Itsevarmuus” keskiarvoissa. Tätä voidaan spekuloida huomion arvoiseksi, sillä itsevarmuus on kuitenkin erittäin oleellinen osa henkilön motivaatiota. Lisäksi muutos tässä kategoriassa on myös suurin avoimen yliopiston opiskelijoiden kesken. Kyseinen kategoria 2 on myös molemmilla, yliopistolla sekä avoimella yliopistolla, alhaisimman keskiarvon saanut kategoria loppukyselyssä. Suurimman keskiarvon sai kategoria 7 ”Motivaatio” lukuun ottamatta avoimen yliopiston loppukyselyä, jossa se jäi jälkeen kategoriasta 4 ”Tietotekniikan käyttö”. Kategoria 5 ”Useampi vastausvaihtoehto” oli jälleen ainoa kategoria, jonka keskiarvo nousi alkukyselystä loppukyselyyn mentäessä yliopiston puolella; avoimen yliopiston puolella keskiarvo nousi myös kategorioiden 1 ”Vastausten tarkistaminen” sekä 4 ”Tietotekniikan käyttö” kohdalla, muutoksen ollessa kuitenkin suurin kategorian 5 ”Useampi vastausvaihtoehto” kohdalla.

Kyselydatalle suoritettiin myös T-testi, jossa tehtiin oletus, että osajoukot ovat toisistaan riippumattomia. T-testin tarkoituksena oli kartoittaa, onko alku- ja loppukyselyjen välillä tapahtunut tilastollisesti merkittävää muutosta, niin yliopiston kuin avoimen yliopistonkin puolella. Testi osoitti, merkitsevyysrajan ollessa 0,05, että tilastollisesti merkittävää eroa alku- ja lopputestin avulla ei havaittu yliopiston tai avoimen yliopiston vastauksissa. Yliopisto-opiskelijoiden osalta p-arvoksi tuli p=0,194 ja avoimen yliopiston osalta p-arvoksi tuli p=0,923. Vaikka kurssin järjestäjien kannalta olisi ollut tietysti suotavaa, että loppukyselyn arvot olisivat olleet korkeammat kuin alkukyselyn, voidaan muutoksen puuttuminen tulkita myös positiivisena asiana: kun motivaatio ja asenteet tuntuivat olevan korkealla jo alkukyselyssä, lienee hyvä asia, etteivät tulokset ainakaan laskeneet. T-testi suoritettiin myös yliopiston ja avoimen yliopiston välillä niin alkutestin kuin lopputestinkin osalta. Merkitsevyysrajan ollessa yleinen 0,05 ei kummassakaan otoksessa havaittu tilastollisesti merkittävää eroa yliopiston ja avoimen yliopiston välillä. Alkukyselyvertailussa p-arvoksi saatiin p=0,304 ja loppukyselyvertailussa p=0,427.

Kyselydatan lisäksi käytössä oli myös kurssin suoritusdatat, joita pystytään käyttämään opiskelijoiden kurssiaktiivisuuden mittaamiseen. Parhaiksi opiskelijoiden kurssiaktiivisuutta mittaaviksi parametreiksi ajateltiin sopivan yliopisto-opiskelijoiden

osalta tutoriaalit, ViLLE-tehtävät sekä demonstraatiotehtävät, ja avoimen yliopiston opiskelijoiden osalta tutoriaalit sekä ViLLE-tehtävät. Tenttituloksia ei ole käytetty, sillä kurssi on mahdollista läpäistä myös pelkällä riittävällä aktiivisuudella. Datasta on karsittu pois esimerkiksi sellaiset henkilöt, jotka eivät ole suorittaneet kurssin aikana demonstraatioita (sillä demonstraatioiden osalta riittää, että ne on suoritettu jo edellisen kurssiyrityksen aikana) tai jotka ovat selvästi jättäneet kurssin kesken todella alhaisen pistemäärän takia. Tällöin yliopiston puolella suoritusdatan tarkastelujoukoksi tuli N=232, ja avoimen yliopiston tarkastelujoukoksi N=51. Avoimen yliopiston puolella demonstraatiot eivät ole osa kurssia, ja lisäksi muutamien harjoitteiden ja tutoriaalien maksimipisteet eroavat hieman yliopiston ja avoimen yliopiston välillä: tästä johtuen yliopiston ja avoimen yliopiston suoriutumisen välinen vertailu on siis suoritettu suhteellisia lukuja käyttäen. Yliopiston ja avoimen yliopiston suoriutumisen suhteelliset vertailut löytyvät taulukosta 5. Taulukossa nähtävä ”Suhteellinen keskimääräinen pistemäärä” on laskettu jakamalla keskimääräiset pisteet maksimipisteillä. Tällä

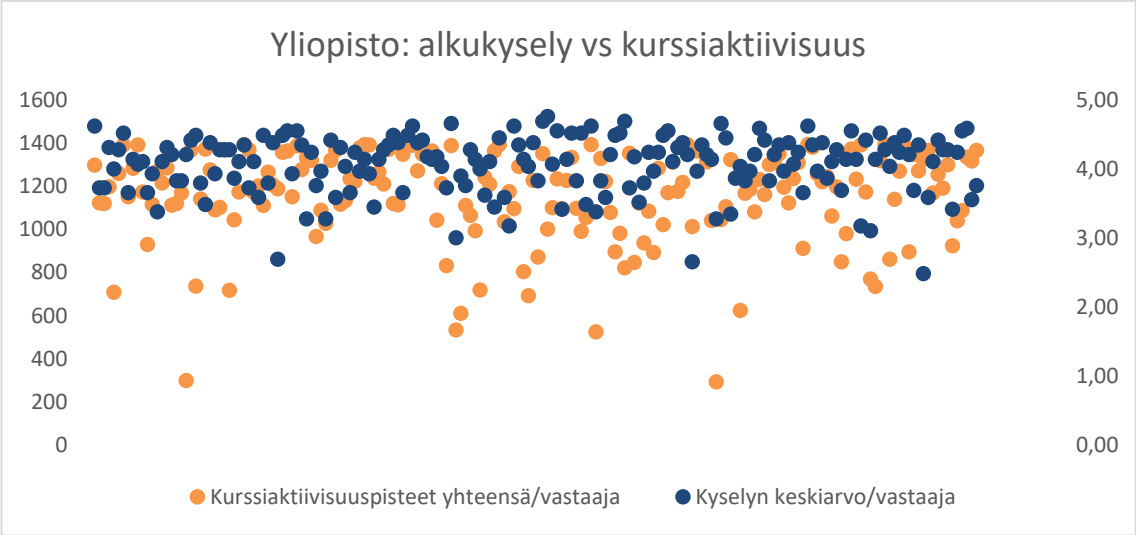
Taulukko 4 Avoimen yliopiston opiskelijoiden sekä yliopisto-opiskelijoiden kurssiaktiivisuuspisteiden statistiikkaa

	Yliopisto	**Avoin**
Keskimääräinen pistemäärä	1119,8 (max. 1392)	795,9 (max. 974)
Suhteellinen keskimäärinen pistemäärä	0,804 (max. 1)	0,817 (max. 1)
%-osuus, joka sai > 50% täysistä pisteistä	94%	92%
%-osuus, joka sai > 75% täysistä pisteistä	72%	75%
%-osuus, joka sai > 90% täysistä pisteistä	34%	51%
%-osuus, joka sai > 95% täysistä pisteistä	21%	41%

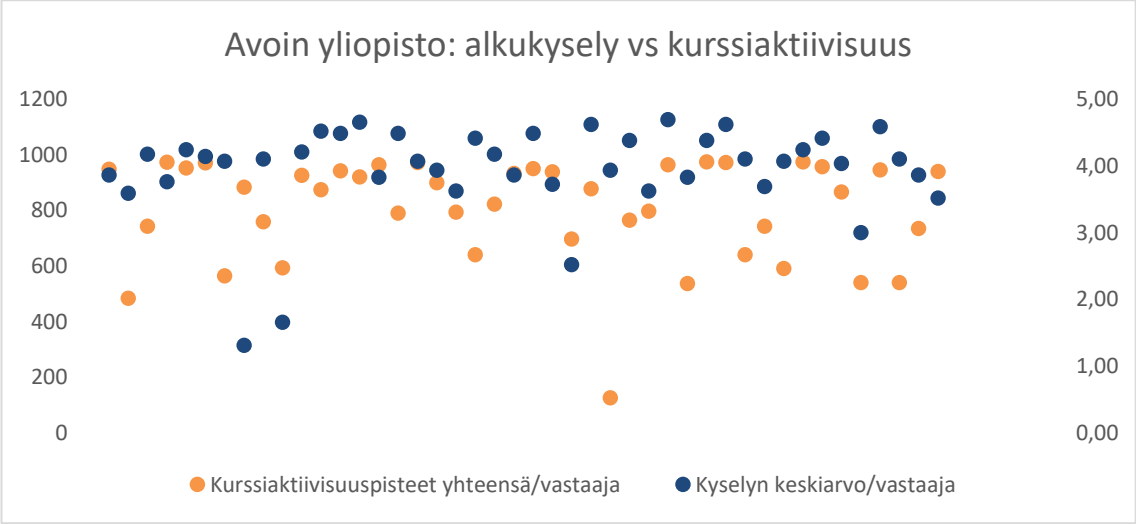
menetelmällä tehdyllä vertailulla huomataan, että yliopisto-opiskelijoiden ja avoimen yliopiston opiskelijoiden keskimääräinen kurssiaktiivisuus on ollut hyvin saman luontoista. Vaikuttaisi kuitenkin siltä, että suurempi osa avoimen yliopiston opiskelijoista suorittaa lähes täydet pistemäärät kurssilta kuin yliopisto-opiskelijat.

Myös suoritusdatan suhdetta kyselydataan voidaan tarkastella, eli löytyisikö jotain yhteyttä opiskelijan kurssisuoriutumisen ja motivaatiokyselyn tulosten välillä. Ennen tällaista tarkastelua yhdistetyille datoille tehtiin vielä yhteiskarsinta: yhdistettiin datasetit siten, että mukaan otettiin ainoastaan opiskelijat, jotka olivat suorittaneet kurssin loppuun, sekä vastanneet motivaatiokyselyyn mielekkäästi. Tämän karsinnan jälkeen otoskoot olivat alkukyselyn osalta yliopistolla 184 (N=184) ja avoimella yliopistolla 44 (N=44), ja loppukyselyn osalta yliopistolla 143 (N=143) ja avoimella yliopistolla 33 (N=33). Kuviot 1 ja 2 hahmottavat, miten kurssisuoriutuminen ja motivaatiokyselyn tulokset suhteutuvat. Kuvaajista voi päätellä muun muassa, että keskimäärin opiskelijat arvioivat motivaationsa korkeammaksi, kuin mitä varsinainen kurssiaktiivisuus käytännössä oli. Kuvaajista voidaan vetää joitain johtopäätöksiä ja spekulaatioita, esimerkiksi useista kurssiaktiivisuuspisteiden eroamista voidaan päätellä, etteivät oppilaiden kyselytulokset ja aktiivisuuspisteet ole kovin yhteneviä keskenään, ja korrelaatiokerroin tulee näin tuskin olemaan kovin korkea.

Koska datojen oletetaan olevan lineaarisessa suhteessa toisiinsa, tehtiin niille Pearsonin lineaarinen korrelaatioanalyysi. Tämän analyysin mukaan datojen väliltä ei löytynyt tilastollisesti merkittävää korrelaatiota (kts. taulukko 6). Huomion arvoista kuitenkin on, etteivät nämä tulokset kerro kausaliteetista; näyttäisi kuitenkin siltä, että oppilaiden motivaatiot ja asenteet eivät vaikuttaisi näiden kurssiaktiivisuuteen. Myöskään mikään tietty kategoria tai kysymys ei näyttänyt korreloivan merkittävästi oppilaiden aktiivisuuden kanssa. Korrelaatiokertoimet laskettiin myös aktiivisuuspisteiden ja loppukyselyiden väliltä, mutta aktiivisuuspisteiden korrelaatio loppukyselyiden kanssa osoittautui vielä pienemmäksi mitä niiden korrelaatio oli alkukyselyiden kanssa. Etenkin yliopiston loppukyselyn tapauksessa korrelaatiokertoimet pääasiassa laskivat:



Kuvio 1 Yliopisto-opiskelijoiden motivaatiokyselyn sekä kurssiaktiivisuuspisteiden suhde



Kuvio 2 Avoimen yliopiston opiskelijoiden motivaatiokyselyn sekä kurssiaktiivisuuspisteiden suhde

3/8 kategoriassa huomataan pientä korrelaatiokertoimen lisääntymistä, mutta muuten kertoimet laskivat. Avoimen puolella puolestaan 5/8 kategorian korrelaatio aktiivisuuden kanssa nousi hieman, ja kolmen puolestaan laski. Erityisesti avoimen kohdalla ensimmäisen kategorian korrelaation nousu ja nousun määrä on huomattavaa verrattuna muihin; siltikään korrelaatiokertoimet eivät ole avoimenkaan yliopiston kohdalla tilastollisesti merkittäviä. Kategorioiden korrelaatiot aktiivisuuden kanssa vaihtelivat välillä suuresti yliopiston ja avoimen yliopiston välillä, esimerkiksi

loppukyselyn kategorian 1 korrelaatio yliopisto-opiskelijoiden kurssiaktiivisuuden kanssa oli 0,109, kun taas samaisen kyselykategorian korrelaatio avoimen yliopiston opiskelijoiden kurssiaktiivisuuden kanssa oli 0,431. Näistäkään eroista on silti vaikea päätellä loppujen lopuksi mitään korrelaatioiden pysyessä pääsääntöisesti pieninä.

Taulukossa 6, jossa esitellään motivaatiokyselyn sekä kurssiaktiivisuuspisteiden keskinäiset korrelaatiokertoimet, on käytetty lyhenteitä muutamista kategorioista, joihin kyselyn kysymykset on jaettu kappaleessa 5.2 esitetyllä tavalla. Kategoriat ovat järjestyksessä siten, että ylin on kategoria 1, toinen on kategoria 2 ja niin edelleen. Lyhenteet ovat: tarkistus = ”Vastausten tarkistaminen”, itsevarm. = ”Itsevarmuus”, nautt. = ”Nauttiminen”, tt käyttö = ”Tietotekniikan käyttö”, vast.vaiht = ”Useampi vastausvaihtoehto”, hyödyll. = ”Ohjelmoinnin hyödyllisyys”. Kaksi viimeistä kategoriaa kuvastavat luonnollisesti itse kategorioiden nimiä (kategoria 7 ”Motivaatio” ja kategoria 8 ”Matematiikka”).

Taulukko 5 Motivaatiokyselyn sekä kurssiaktiivisuuspisteiden keskinäiset korrelaatiokertoimet: jaoteltuina sekä yliopisto-opiskelijoiden että avoimen yliopiston opiskelijoiden mukaan, sekä lisäksi myös kyselyssä käytettyjen kahdeksan kategorian mukaan

	Kurssiaktiivisuus:			**Kurssiaktiivisuus:**	
	Yliopisto	**Avoin**		**Yliopisto**	**Avoin**
Alkukysely – tarkistus	0,133	0,364	**Loppukysely – tarkistus**	0,109	0,431
Alkukysely – itsevarm.	0,246	0,251	**Loppukysely – itsevarm.**	0,303	0,209
Alkukysely – nautt.	0,124	0,149	**Loppukysely – nautt.**	0,087	0,076
Alkukysely – tt käyttö	0,011	0,077	**Loppukysely – tt käyttö**	0,035	0,125
Alkukysely – vast.vaiht	0,223	0,330	**Loppukysely – vast.vaiht**	0,182	0,356
Alkukysely – hyödyll.	0,108	0,079	**Loppukysely – hyödyll.**	0,025	-0,122
Alkukysely – motivaatio	0,088	0,128	**Loppukysely – motivaatio**	0,056	-0,072
Alkukysely – matematiikka	-0,026	0,260	**Loppukysely – matematiikka**	0,079	0,157
Alkukysely – yhteensä	0,194	0,260	**Loppukysely - yhteensä**	0,109	0,249

7 JOHTOPÄÄTÖKSET

Tutkimuksen kohdekurssin kurssijärjestelyt tarjosivat mahdollisuuden tarkastella yliopisto-opiskelijoiden lisäksi myös avoimen yliopiston opiskelijoita. Näin ollen myös näiden ryhmien välisiä eroja pystyttiin tarkastelemaan, joten vaikka nämä erot eivät pääfokuksena olleetkaan, tavoitteena oli myös tarkastella näiden kahden kohderyhmän välisiä mahdollisia eroja. Mikäli jakoa yliopistoon ja avoimeen yliopistoon ei erikseen mainita, oletetaan, että puhutaan näiden yhdistetystä otoksesta.

Ensimmäisen ohjelmointikurssin motivaatio- ja asennetekijöitä ei ole tutkittu paljoa, vaan tutkimukset yleensä vain toteavat motivaation olevan vaikuttava tekijä kurssisuoriutumisessa ja -menestyksessä (Kopf, Scheele ym. 2005, Behnke, Kos ym. 2016). Kuitenkin esimerkiksi Nikula ym. (2011) ja Jenkins T. (2001) ovat käsitelleet ensimmäisen ohjelmointikurssin opiskelijoiden motivaatiota hieman syvemmin, ja toteavat muun muassa sisäisen ja ulkoisen motivaation vaikuttavan opiskelijoiden motivaatioon ja asenteeseen ohjelmointia kohtaan. Myös hygieniatekijöiden vähentämisen todettiin vaikuttavan positiivisesti. Juurikaan tämän syvemmälle ei näissäkään tutkimuksissa ole menty.

Tämän tutkielman tutkimustulokset näyttävät, että yliopisto-opiskelijoiden sekä avoimen yliopiston opiskelijoiden motivaatio ja asenteet olivat verrattain korkeita, keskimäärin tasolla 4, asteikon ollessa 1-5. Opiskelijoiden motivaatio oli siis verrattain korkea kurssin alussa, joka on luonnollisesti hyvä asia. Oli positiivista huomata myös miten ensimmäisen ohjelmointikurssin jälkeen tehdyssä loppukyselyssä motivaation ja asenteiden ei todettu muuttuneen juuri lainkaan. Vaikka motivaation olisi tietysti suonut nousevan, voidaan motivaation pysyminen samana kokea hyvänä asiana, huomioiden erityisesti yleisesti korkean keskeytysprosentin sekä muutenkin ensimmäisen ohjelmointikurssin haastavuuden.

Kurssin alussa suoritetun motivaatiokyselyn kategorioiden pistemäärissä oli joitakin huomattavia eroja, kuten esimerkiksi kategorian 7 ”Motivaatio” ero kategoriaan 5 ”Useampi vastausvaihtoehto”. Näiden välinen ero oli reilun pisteen verran (4,56 vs 3,48),

joka on viisiasteisella mittaristolla merkittävä. Tämän eron syytä voidaan spekuloida esimerkiksi kategorioiden kysymysten kautta: esimerkiksi ”Motivaatio”-kategoria pyrkii mittaamaan pitkälti suoraan motivaatiota, kun taas ”Useampi vastausvaihtoehto”-kategoria kysyy enemmänkin opiskelijoiden toiminta-/opiskelutapoja ja -asenteita. Opiskelijalla voi siis esimerkiksi olla korkea motivaatio, mutta tämä voi olla esimerkiksi huono tarkistamaan vastauksiaan uudestaan, joka puolestaan heijastuu hieman keskiarvoa pienempänä pistemääränä. Alhaisen pistemäärän sai myös kategoria 2 ”Itsevarmuus”, joka jäi loppukyselyssä alhaisimmille pisteille (3,51). Tämä antaa viitteitä, kuinka opiskelijoiden itsevarmuus ohjelmointia kohtaan ei välttämättä ollut erityisen korkea. Kuitenkin opiskelijoiden motivaatio näytti olevan korkea; ”Motivaatio”-kategoria säilyi kurssin päätteeksi tehdyn motivaatiokyselyn korkeimman pistemäärän omaavana kategoriana pisteillä 4,46. Löydetty itsevarmuuden pienuus heijastelee kuitenkin todennäköisesti enemmän opiskelijoiden osaamisen kanssa, joka aivan oletettavasti on heikkoa vielä ensimmäisellä ohjelmointikurssilla. Itsevarmuus-kategorian pisteiden nousu olisi siis ollut toivottavaa, vaikkei niiden pieni lasku 3,58:sta 3,51:een kovin hälyttävää olekaan. Tämä kuitenkin antaa viitteitä esimerkiksi siitä, mitä asioita kurssilla voisi seuraavaksi parantaa.

Tutkimuksissa huomattiin myös, ettei motivaatioissa ja asenteissa ollut juurikaan eroja myöskään yliopisto-opiskelijoiden sekä avoimen yliopiston opiskelijoiden välillä, niin alku- kuin loppukyselyssäkään. Suurimmat erot yliopiston ja avoimen yliopiston väliltä löytyivät loppukyselyssä kategorian 2 ”Itsevarmuus” kohdalta, jossa ero oli yliopiston hyväksi 0,29. Tämä näyttäisi siltä, että yliopisto-opiskelijat olisivat ainakin keskimäärin hieman itsevarmempia ohjelmoinnissa kuin avoimen yliopiston opiskelijat, erityisesti ensimmäisen ohjelmointikurssin jälkeen. Ero on jälleen aika pieni, mutta kertoo hieman siitä, missä kurssilaiset voisivat kaivata kannustusta ja apua. Etenkin avoimen yliopiston puolella itsevarmuus ohjelmoinnista tuntuu hieman laskevan kurssin aikana: loppukyselyn ”Itsevarmuus”-kategorian pisteet laskivat 0,25 pistettä, verrattuna taas yliopisto-opiskelijoiden saman kategorian 0,03 pisteen laskuun. Esimerkiksi kokonaan avoimen yliopiston kurssilta puuttuvat demonstraatiotehtävät voivat osin selittää sitä, miksi avoimen yliopiston puolella itsevarmuus on hieman heikompaa; yliopisto-

opiskelijat voivat siis tämän perusteella saada jotain käytännön etua demonstraatioiden suorittamisesta.

Pelkästä kurssiaktiivisuusdatasta saatiin selville, että yliopiston ja avoimen yliopiston opiskelijoiden kurssiaktiivisuus oli keskimäärin hyvin samanlaista keskenään. Eroa löytyi kuitenkin tarkasteltaessa korkeimmat pistemäärät haalineita opiskelijoita: avoimen yliopiston puolella opiskelijat keräsivät keskimäärin korkeampia pistemääriä kuin yliopisto-opiskelijat, kun tarkastellaan opiskelijoita, jotka suorittivat 90% tai 95% maksimipisteistä. Tämä kuitenkin selittyy pitkälti sillä, että yliopisto-opiskelijoiden kurssisuorituksiin kuului myös haastavat demonstraatiot: kun demonstraatioiden osuutta ei otettu huomioon kurssisuorituksissa, tilanne kääntyi päälaelleen ja yliopisto-opiskelijoista suurempi osa suoritti yli 90% tai 95% vaadituista tehtävistä kuin avoimen yliopiston opiskelijoista. Yhtä kaikki on positiivista huomata, että niin yliopisto-opiskelijat kuin avoimen yliopistonkin opiskelijat suorittivat tehtäviä keskimäärin todella hyvin.

Kun motivaatiokyselydatat yhdistettiin kurssilta saatuihin suoritusdatoihin, pystyttiin tutkimaan näiden välistä korrelaatiota: onko siis opiskelijoiden motivaatiolla ja kurssiaktiivisuudella yhteyttä? Suoritetun Pearsonin lineaarisen korrelaatioanalyysin jälkeen pystyttiin toteamaan, että motivaatio- ja asennekyselyn sekä opiskelijoiden aktiivisuuden välillä ei ollut tilastollisesti merkittävää korrelaatiota. Myöskään kyselyssä käytettyjen erillisten kategorioiden välillä ei esiintynyt kovinkaan merkittäviä eroja korrelaatioissa. Syytä tälle on hieman vaikea lähteä spekuloimaan, mutta kategorioiden eroja on kuitenkin mielenkiintoista tarkastella. Esimerkiksi kategorian 1 ”Vastausten tarkistaminen” korrelaatiokerroin oli yliopisto-opiskelijoilla 0,109, kun se taas avoimen yliopiston opiskelijoiden kohdalla oli 0,431. Vaikka kumpikaan korrelaatio ei ole tilastollisesti merkittävä, voidaan todeta avoimen opiskelijoiden vastaamisasenteen kertovan enemmän näiden kurssiaktiivisuudesta kuin yliopisto-opiskelijoiden. Avoimen opiskelijoilla oli myös yliopistoa korkeammat keskimääräiset aktiivisuuspisteet, joten esimerkiksi vastaamisasenteen voitaisi päätellä vaikuttavan positiivisesti aktiivisuuteen. Kuitenkin kuten aikaisemminkin mainittiin, korrelaatiot

eivät ole tilastollisesti merkittäviä, joten johtopäätökset ovat pitkälti vain spekulointia. Muutenkin tulee muistaa, että pelkästä korrelaatiosta ei voida vetää johtopäätöksiä varsinaisesta kausaliteetista. Tulokset ovat silti mielenkointoisia, ja antavat kuvaa siitä, miten opiskelijat ajattelevat.

Tutkimuksen ja motivaatiokyselyn tuloksia tulee luonnollisesti tarkastella myös kriittisesti. Suurena tekijänä tuloksissa on tietysti käytetty kysely. Alkuperäinen Wong K. Y. ja Chen Q. (2012) luoma kysely oli testattu EFA- ja CFA-menetelmillä. Alkuperäinen kysely oli myös tehty matematiikkaympäristöön, joten sitä jouduttiin muokkaamaan ohjelmointiin sopivaksi. Tätä ohjelmointiympäristöön sopivaksi muunnettua uutta kyselyä käytettiin nyt vasta ensimmäistä kertaa, joten EFA- ja CFA-testejä ei pystytty tekemään ennen sen käyttöä. Vaikka käytetty kysely pohjautuukin hyvin pitkälti alkuperäiseen kyselyyn, on silti syytä ottaa huomioon, että muokkausta tehtiin, ja että se on mahdollisesti voinut vaikuttaa kyselyn validiteettiin. Myös joidenkin kysymysten esitysmuotoa voidaan hieman kritisoida: esimerkiksi ”Ohjelmointi on helppoa” voi olla hieman yksinkertainen kysymys viisiasteisella Likert-asteikolla yliopisto-opiskelijoille. Toisaalta se voi olla juuri yksinkertaisuutensa takia myös hyvä kysymys. Kuitenkin Likert-asteikon vaihtaminen viisiasteisesta esimerkiksi yhdeksänasteiseksi voisi olla perusteltua.

Vaikka kyselyyn voidaankin kohdistaa kritiikkiä, voidaan sitä kuitenkin pitää luotettavana mittarina. Se antaa hyvää osviittaa siitä, miten opiskelijat asennoituvat ohjelmointikurssiin, ja minkälainen motivaatio näillä on suorittaa kurssi. Myös itse motivaation, ja esimerkiksi sisäisen ja ulkoisen motivaation, eroja ja roolia voidaan spekuloida; kyselyn tarkoituksena oli pitkälti mitata opiskelijoiden sisäistä motivaatiota eri muodoissa. Kuitenkin ulkoisen motivaation roolia ei tule unohtaa, ja kuten Ryan ja Deci (2000) sanovat, alkujaan ulkoisena motivaationa havainnoitu motivaatio ei välttämättä olekaan kokonaan ulkoista, vaan voi heijastaa ja vaikuttaa hyvinkin vahvasti ihmisen sisäiseen motivaatioon. Hyvänä esimerkkinä tästä voidaan pitää sitä, miten tärkeänä opiskelija kokee ohjelmointitaidon työpaikan saamisessa: motivaatio on

pitkälti ulkoisten tekijöiden sanelema, mutta jos opiskelija kokee sen todeksi ja asiaksi, joka on tavoittelemisen arvoista, voi tästä muodostua korkeakin sisäinen motivaatio.

Jatkotutkimus etenkin kyselyn kehittämisen kannalta olisi suotavaa ja mielenkiintoista, sillä kyselylle on kuitenkin olemassa jo pohja, se pitäisi vain tuoda perusteellisemmin testaamalla ohjelmointiympäristöön. Myös ohjelmointikyselyssä käytettäviä kategorioita voisi miettiä vielä tarkemmin, ja käyttää analysointimenetelmiä enemmän ja laajemmin niiden määrittämiseksi. Kyselyn jatkokehitys voisi edelleen auttaa esimerkiksi riskiryhmään kuuluvien opiskelijoiden, eli keskimääräistä helpommin keskeyttävien, tunnistamista, joka puolestaan olisi hyödyksi sekä opiskelijoille että opettajille. Motivaatiokysely olisi mielenkiintoista suorittaa myös jollain muulla ohjelmointikurssilla, jossa käytetyt opetusmetodit ja -järjestelmät ovat erilaiset: näin tutkimuksia voisi vertailla ja vertailla esimerkiksi erilaisia opetusmetodeja.

8 YHTEENVETO

Yleisenä ongelmana ensimmäisillä ohjelmointikursseilla on ollut korkeat keskeytysprosentit, sekä yleisesti täysin uudenlaisten konseptien opettelu ja omaksuminen. Ongelmana on myös, miten vähän näiden ohjelmointia opiskelevien motivaatiosta tiedetään; motivaatio on tärkeä osa oppimista, ja se voi olla myös tärkeä linkki korkeiden keskeytysprosenttien selittämisessä. Myös omassa yliopistossani, Turun yliopistossa, ohjelmointikurssille ei ole tietääkseni aiemmin tehty motivaatiokyselyä. Tämän tutkielman tavoitteena oli siis suorittaa opiskelijoiden motivaatiota ja asenteita kartoittava motivaatiokysely ensimmäisen ohjelmointikurssin opiskelijoille, kohderyhmänä Turun yliopiston opiskelijat, sekä tutkia, onko opiskelijoiden motivaatiolla ja näiden kurssisuoriutumisella yhteyttä. Kurssijärjestelyt tarjosivat mahdollisuuden tarkastella myös yliopisto-opiskelijoiden sekä avoimen yliopiston opiskelijoiden välisiä eroja, joten vaikka nämä erot eivät pääfokuksena olleetkaan, tavoitteena oli tarkastella samalla näiden kahden kohderyhmän välisiä mahdollisia eroja.

Tutkimustulokset näyttävät, että yliopisto-opiskelijoiden sekä avoimen yliopiston opiskelijoiden motivaatio ja asenteet olivat verrattain korkeita kurssin alussa, keskimäärin tasolla 4, asteikon ollessa 1-5. Ensimmäisen ohjelmointikurssin jälkeen tehdyssä loppukyselyssä motivaation ja asenteiden ei todettu muuttuneen juuri lainkaan kurssin aikana. Tämä on positiivista erityisesti yleisesti korkean keskeytysprosentin sekä muutenkin ensimmäisen ohjelmointikurssin haastavuuden huomioiden. Tutkimuksissa huomattiin myös, ettei motivaatioissa ja asenteissa ollut juurikaan eroja myöskään yliopisto-opiskelijoiden sekä avoimen yliopiston opiskelijoiden välillä, niin alku- kuin loppukyselyssäkään. Lisäksi pystyttiin toteamaan myös, että motivaatio- ja asennekyselyn sekä opiskelijoiden aktiivisuuden välillä ei ollut tilastollisesti merkittävää korrelaatiota. Vaikkei korrelaation puuttumisesta voidakaan kausaalista suhdetta suoraan päätellä, antaa tulokset silti viitteitä siitä, että opiskelijoiden motivaatio ja asenteet eivät vaikuttaisi näiden kurssiaktiivisuuteen.

Tutkimuksen tuloksia tulee tarkastella myös kriittisesti. Etenkin motivaatiokysely, johon tutkimustulokset pitkälti perustuvat, on kriittisessä osassa koko tutkimusta. Sen osaksi voidaan antaa kritiikkiä muun muassa kysymysten osalta, joita käytettiin nyt ensimmäistä kertaa suomeksi, sekä muutamia kysymyksiä kokonaan ensimmäistä kertaa. Kysymysten toimivuutta ei ole siis voitu testata ennen niiden käyttöä, ja kysymyspalettiin voisikin kohdistaa pieniä uudistuksia. Myös viisiasteisen Likert-asteikon vaihtaminen yhdeksänasteiseen voisi olla perusteltua. Tuloksia voidaan kuitenkin pitää luotettavina, koska käytetty kysely perustuu hyvin vahvasti alkuperäiseen Wong K. Y. ja Chen Q. (2012) luomaan kyselyyn.

Jatkotutkimus etenkin kyselyn kehittämisen puitteissa olisi mielenkiintoista, ja alkuperäisen kyselyn vahvistettu siirtäminen ohjelmointiympäristöön olisi hyvä asia jatkotutkimuksia ajatellen. Motivaatiokysely olisi mielenkiintoista suorittaa myös jollain muulla ohjelmointikurssilla, jossa käytetyt opetusmetodit ja -järjestelmät olisivat erilaiset: näin tutkimustuloksia voisi vertailla ja spekuloida esimerkiksi erilaisten opetusmetodien toimivuutta opiskelijoiden motivaatioon ja asenteisiin peilaten.

LÄHTEET

ALLY, M., 2004. Foundations of educational theory for online learning. In: ANDERSON TERRY, ed, *Theory and practice of online learning.* Athabasca University Press, 2008, pp. 15-44.

AMRESH, A., CARBERRY, A.R. and FEMIANI, J., October 2013Evaluating the effectiveness of flipped classrooms for teaching CS1, *Frontiers in Education Conference*, 23-26 Oct. 2013 October 2013, IEEE, pp. 733-735.

ARBIB, M.A., 1992. Schema theory. *The Encyclopedia of Artificial Intelligence,* **2**, pp. 1427-1443.

BEHESHTI, M. and GUNAWARDENA, A.D., 2001Teaching Programming Paradigms Using a Laboratory Approach, 2001, Consortium for Computing Sciences in Colleges, pp. 144–148.

BEHNKE, K.A., KOS, B.A. and BENNETT, J.K., 2016Computer Science Principles: Impacting Student Motivation & Learning Within and Beyond the Classroom, 2016, ACM, pp. 171–180.

BENNEDSEN, J. and CASPERSEN, M.E., 2007. Failure Rates in Introductory Programming. *SIGCSE Bull.,* **39**(2), pp. 32–36.

BOEKAERTS, M., 2016. Engagement as an inherent aspect of the learning process. *Learning and Instruction,* **43**, pp. 76-83.

BONWELL, C.C. and EISON, J.A., 1991. *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.* ERIC Clearinghouse on Higher Education, The George Washington University, One Dupont Circle, Suite 630, Washington, DC 20036-1183 ($17.

BROWN, M.H. and SEDGEWICK, R., 1984A System for Algorithm Animation, 1984, ACM, pp. 177–186.

BUCKLEY, J. and EXTON, C., 2003Bloom's taxonomy: a framework for assessing programmers' knowledge of software systems, 2003, IEEE, pp. 165-174.

BUTLER, M. and MORGAN, M., 2007. Learning challenges faced by novice programming students studying high level and low feedback concepts. *Proceedings ascilite Singapore,* (99-107),.

DALE, N.B., 2006. Most Difficult Topics in CS1: Results of an Online Survey of Educators. *SIGCSE Bull.,* **38**(2), pp. 49–53.

ERDEI, R., SPRINGER, J.A. and WHITTINGHILL, D.M., 2017An impact comparison of two instructional scaffolding strategies employed in our programming laboratories: Employment of a supplemental teaching assistant versus employment of the pair programming methodology, 2017, IEEE, pp. 1-6.

ERKKI KAILA, TEEMU RAJALA, MIKKO-JUSSI LAAKSO, ROLF LINDÉN, EINARI KURVINEN, VILLE KARAVIRTA and TAPIO SALAKOSKI, 2015. Comparing student performance between traditional

and technologically enhanced programming course. *Proceedings of the Seventeenth Australasian Computing Education Conference ACE2016,* , pp. 147-154.

FERG, S., 2006. Event-Driven Programming:
Introduction, Tutorial, History , pp. 1-58.

FLIPPED LEARNING NETWORK, (., 2014. *The Four Pillars of F-L-I-P.*

GOMES, A. and MENDES, A., October 2014A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations, *Frontiers in Education Conference (FIE)*, 22-25 Oct. 2014 October 2014, IEEE, pp. 1-8.

GOMES, A., CARMO, L., BIGOTTE, E. and MENDES, A., 2006Mathematics and programming problem solving, *3rd E-Learning Conference–Computer Science Education* 2006, Citeseer, pp. 1-5.

GOOSEN, L., MENTZ, E. and NIEUWOUDT, H., June 20, 2007Choosing the "Best" Programming Language? June 20, 2007, ResearchGate, pp. 269-282.

GUZDIAL, M. and GUO, P., 2014. The Difficulty of Teaching Programming Languages, and the Benefits of Hands-on Learning. *Commun. ACM,* **57**(7), pp. 10–11.

HAGAN, D. and MARKHAM, S., 2000Does It Help to Have Some Programming Experience Before Beginning a Computing Degree Program? 2000, ACM, pp. 25–28.

HAKIMZADEH, H., ADAIKKALAVAN, R. and BATZINGER, R., 2011Successful Implementation of an Active Learning Laboratory in Computer Science, *ACM SIGUCCS*, November 12-17, 2011 2011, ACM, pp. 83–86.

HEGAZI, M.O. and ALHAWARAT, M., 2015The Challenges and the Opportunities of Teaching the Introductory Computer Programming Course: Case Study, *2015 Fifth International Conference of e-Learning*, 18-20 Oct. 2015 2015, IEEE, pp. 324-330.

HOFSTEDT, P., 2011. Programming Languages and Paradigms. *Multiparadigm Constraint Programming Languages.* Springer-Verlag Berlin Heidelberg, pp. 17-34.

HUDAK, P., 2000. *Haskell school of expression. Learning functional programming through multimedia.* Cambridge: Cambridge University Press.

HUNG, C., HWANG, G. and HUANG, I., 2012. A Project-based Digital Storytelling Approach for Improving Students' Learning Motivation, Problem-Solving Competence and Learning Achievement. *Journal of Educational Technology & Society,* **15**(4), pp. 368-379.

IVANOVIĆ, M., XINOGALOS, S., PITNER, T. and SAVIĆ, M., 2017. Technology enhanced learning in programming courses – international perspective. *Education and Information Technologies,* **22**(6), pp. 2981-3003.

JENKINS, T., 2001The Motivation of Students of Programming, 2001, ACM, pp. 53–56.

JOHNSON, C.G. and FULLER, U., 2006Is Bloom's Taxonomy Appropriate for Computer Science? 2006, ACM, pp. 120–123.

JONASSEN, D.H., 1999. Designing constructivist learning environments. *Instructional design theories and models: A new paradigm of instructional theory,* **2**, pp. 215-239.

KAILA, E., RAJALA, T., LAAKSO, M. and SALAKOSKI, T., 2010Effects of Course-long Use of a Program Visualization Tool, , January, 2010 2010, Australian Computer Society, Inc., pp. 97–106.

KAY, D. and KIBBLE, J., 2016. Learning theories 101: Application to everyday teaching and scholarship. *Advances in Physiology Education,* **40**(1), pp. 17-25.

KOPF, S., SCHEELE, N., WINSCHEL, L. and EFFELSBERG, W., 2005. Improving activity and motivation of students with innovative teaching and learning technologies.

KOULOURI, T., LAURIA, S. and MACREDIE, R.D., 2015. Teaching introductory programming: a quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE),* **14**(4), pp. 26.

KRATHWOHL, D.R., 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice,* **41**(4), pp. 212-218.

LAAKSO, M., 2010. *Promoting programming learning. engagement, automatic assessment with immediate feedback in visualizations*, TUCS Dissertations no 131.

LAHTINEN, E., ALA-MUTKA, K. and JÄRVINEN, H., 2005A Study of the Difficulties of Novice Programmers, 2005, ACM, pp. 14–18.

LASSERRE, P. and SZOSTAK, C., 2011Effects of Team-based Learning on a CS1 Course, 2011, ACM, pp. 133–137.

LEPPINIEMI, H., 2016. *Ilmiö nimeltä ilmiöpohjainen oppiminen*, Tampereen yliopisto.

M. SELVAKUMAR SAMUEL, 2017. An Insight into Programming Paradigms and Their Programming Languages. *Journal of Applied Technology and Innovation,* **1**(1), pp. 37-57.

MACHADO, R., 2013An Introduction to Lambda Calculus and Functional Programming, , 15-17 Oct. 2013 2013, IEEE, pp. 26-33.

MALONEY, J., RESNICK, M., RUSK, N., SILVERMAN, B. and EASTMOND, E., 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.,* **10**(4), pp. 16:1–16:15.

MARTÍN-BLAS, T. and SERRANO-FERNÁNDEZ, A., 2009. The role of new technologies in the learning process: Moodle as a teaching tool in Physics. *Computers & Education,* **52**(1), pp. 35-44.

MAYER, R.E., 1981. The Psychology of How Novices Learn Computer Programming. *ACM Comput. Surv.,* **13**(1), pp. 121–141.

MCDOWELL, C., HANKS, B. and WERNER, L., 2003Experimenting with Pair Programming in the Classroom, *ITiCSE '03*, June 30 - July 02, 2003 2003, ACM, pp. 60–64.

MCGREGOR, D., 1960. *The human side of enterprise.* New York.

NAPS, T.L., RÖSSLING, G., ALMSTRUM, V., DANN, W., FLEISCHER, R., HUNDHAUSEN, C., KORHONEN, A., MALMI, L., MCNALLY, M., RODGER, S. and VELÁZQUEZ-ITURBIDE, J.Á, 2002Exploring the Role of Visualization and Engagement in Computer Science Education, 2002, ACM, pp. 131–152.

NIKULA, U., GOTEL, O. and KASURINEN, J., 2011. A Motivation Guided Holistic Rehabilitation of the First Programming Course. *Trans. Comput. Educ.,* **11**(4), pp. 24:1–24:38.

NOSEK, J.T., 1998. The Case for Collaborative Programming. *Commun. ACM,* **41**(3), pp. 105–108.

OLIVEIRA AURELIANO, V.C., 2013A Methodology for Teaching Programming for Beginners, 2013, ACM, pp. 169–170.

ORTIN, F., REDONDO, J.M. and QUIROGA, J., 2016. Design of a programming paradigms course using one single programming language. *Advances in Intelligent Systems and Computing,* **445**, pp. 179-188.

PARDEE, R.L., 1990. *Motivation Theories of Maslow, Herzberg, McGregor & McClelland. A Literature Review of Selected Theories Dealing with Job Satisfaction and Motivation.* ERIC.

PASHLER, H., CEPEDA, N.J., WIXTED, J.T. and ROHRER, D., 2005. When does feedback facilitate learning of words? *Journal of Experimental Psychology: Learning, Memory, and Cognition,* **31**(1), pp. 3.

PEARS, A., SEIDMAN, S., MALMI, L., MANNILA, L., ADAMS, E., BENNEDSEN, J., DEVLIN, M. and PATERSON, J., 2007A Survey of Literature on the Teaching of Introductory Programming, 2007, ACM, pp. 204–223.

PINTRICH, P.R., 1999. The role of motivation in promoting and sustaining self-regulated learning. *International Journal of Educational Research,* **31**(6), pp. 459-470.

PORTER, L. and SIMON, B., 2013Retaining Nearly One-third More Majors with a Trio of Instructional Best Practices in CS1, 2013, ACM, pp. 165–170.

POWERS, K.D., 2004Teaching Computer Architecture in Introductory Computing: Why? And How? *Australasian Conference on Computing Education* 2004, Australian Computer Society, Inc., pp. 255–260.

PRICE, L. and KIRKWOOD, A., 2014. Using technology for teaching and learning in higher education: a critical review of the role of evidence in informing practice. *Higher Education Research & Development,* **33**(3), pp. 549-564.

PRINCE, M., 2004. Does Active Learning Work? A Review of the Research. *Journal of Engineering Education,* **93**(3), pp. 223-231.

PYLKKÄ OUTI, , Oppimiskäsitykset [Homepage of JAMK], [Online]. Available: http://oppimateriaalit.jamk.fi/oppimiskasitykset/ [Feb 23, , viittauspäivämäärä: 23.2., 2018].

RADERMACHER, A.D. and WALIA, G.S., 2011Investigating the Effective Implementation of Pair Programming: An Empirical Investigation, 2011, ACM, pp. 655–660.

RADOŠEVIĆ, D., OREHOVAČKI, T. and LOVRENČIĆ, A., 2009. *New Approaches and Tools in Teaching Programming.* Rochester, NY: SSRN.

RAJALA, T., LAAKSO, M.-., KAILA, E. and SALAKOSKI, T., 2008. Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education: Innovations in Practice, IIP,* **7**, pp. 15–32.

RIZVI, M., HUMPHRIES, T., MAJOR, D., JONES, M. and LAUZUN, H., 2011. A CS0 Course Using Scratch. *J. Comput. Sci. Coll.,* **26**(3), pp. 19–27.

ROBINS, A., ROUNTREE, J. and ROUNTREE, N., 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education,* **13**(2), pp. 137-172.

RYAN, R.M. and DECI, E.L., 2000. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology,* **25**(1), pp. 54-67.

SALMELA-ARO, K. and NURMI JARI-ERIK, 2005. *Mikä meitä liikuttaa - Modernin motivaatiopsykologian perusteet.* 2. painos edn. Keuruu: PS-kustannus.

SAMUEL, S., 2015. Teaching Programming Subjects with Emphasis on Programming Paradigms. In: L. HALIM, ed, *Proceedings of the 2014 International Conference on Advances in Education Technology.* Paris: Atlantis Press, pp. 94-97.

SCHUNK, D.H., 2012. *Learning theories an educational perspective sixth edition.* 6th Edition edn. Pearson.

SCHWEITZER, D. and BROWN, W., 2007Interactive Visualization for the Active Learning Classroom, *SIGCSE technical symposium on Computer science education* 2007, ACM, pp. 208–212.

SCOTT, T., 2003. Bloom's Taxonomy Applied to Testing in Computer Science Classes. *J. Comput. Sci. Coll.,* **19**(1), pp. 267–274.

SHAMBHAVI, B.R., 2017Effective collaborative activities and active learning in engineering education: A case study, *IEEE International Conference on MOOCs* 2017, IEEE, pp. 137-139.

SLOAN, R.H. and TROY, P., 2008CS 0.5: A Better Approach to Introductory Computer Science for Majors, 2008, ACM, pp. 271–275.

SSS IT PVT LTD, , Fibonacci series in Java. Available: https://www.javatpoint.com/fibonacci-series-in-java [18.1., 2018].

SUHR, D.D., 2006. *Exploratory or confirmatory factor analysis?* University of Northern Colorado: SAS Institute Cary.

TASK GROUP ON INFORMATION TECHNOLOGY, 2017. *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology.* New York, NY, USA: ACM.

TEW, A.E., FOWLER, C. and GUZDIAL, M., 2005Tracking an Innovation in Introductory CS Education from a Research University to a Two-year College, 2005, ACM, pp. 416–420.

THOMPSON, E., LUXTON-REILLY, A., WHALLEY, J.L., HU, M. and ROBBINS, P., 2008Bloom's Taxonomy for CS Assessment, 2008, Australian Computer Society, Inc., pp. 155–161.

TOIVOLA MARIKA, a-last update, Flipped learning. Available: https://www.utu.fi/fi/sivustot/koulutus-ja-kehittamispalvelut/oikeasti-oppimaan/paikalliset-toimijat/tieto-ja-viestintateknologian-hyodyntaminen/flipped-learning/Sivut/home.aspx [25.1., 2018].

TOIVOLA MARIKA, b-last update, Käänteinen oppiminen. Available: http://www.flippedlearning.fi/p/kaanteinen-oppiminen_12.html [25.1., 2018].

TUCKER, B., 2012. The Flipped Classroom. *Education Next; Cambridge,* **12**(1),.

TULEVAISUUDEN TEKNOLOGIOIDEN LAITOS, , Tietojenkäsittelytieteiden tutkinto-ohjelma (LuK). Available: https://nettiopsu.utu.fi/opas/koulutusohjelma.htm?opsId=260&uiLang=fi&lang=fi&lvv=2017&koulohj=TIKK2 [5.12., 2017].

VAN ROY, P., ARMSTRONG, J., FLATT, M. and MAGNUSSON, B., 2003The Role of Language Paradigms in Teaching Programming, 2003, ACM, pp. 269–270.

VIHAVAINEN, A., AIRAKSINEN, J. and WATSON, C., Jul 28, 2014A systematic review of approaches for teaching introductory programming and their influence on success, Jul 28, 2014, ACM, pp. 19-26.

VILLE TEAM, -11-10T10:29:24+02:00, 2015-last update, ViLLE - Ominaisuudet. Available: https://oppimisanalytiikka.fi/fi/ville [Dec 15, 2017].

WALKER, H.M., 1997Collaborative Learning: A Case Study for CS1 at Grinnell College and Austin, 1997, ACM, pp. 209–213.

WATSON, C. and LI, F.W.B., 2014Failure Rates in Introductory Programming Revisited, 2014, ACM, pp. 39–44.

WEIR, G.R.S., VILNER, T., MENDES, A.J. and NORDSTRÖM, M., 2005Difficulties Teaching Java in CS1 and How We Aim to Solve Them, 2005, ACM, pp. 344–345.

WIEDENBECK, S., LABELLE, D. and N R KAIN, V., May 1, 2004PPIG 2004 Factors Affecting Course Outcomes in Introductory Programming, May 1, 2004, Psychology of Programming Interest Group, pp. 97-110.

WIGFIELD, A. and ECCLES, J.S., 2000. Expectancy–Value Theory of Achievement Motivation. *Contemporary Educational Psychology,* **25**(1), pp. 68-81.

WILLIAMS, L., KESSLER, R.R., CUNNINGHAM, W. and JEFFRIES, R., 2000. Strengthening the case for pair programming. *IEEE Software,* **17**(4), pp. 19-25.

WILLIAMS, L., 2007. Lessons Learned from Seven Years of Pair Programming at North Carolina State University. *SIGCSE Bull.,* **39**(4), pp. 79–83.

WILLMAN, S., LINDÉN, R., KAILA, E., RAJALA, T., LAAKSO, M. and SALAKOSKI, T., 2015. On study habits on an introductory course on programming. *Computer Science Education,* **25**(3), pp. 276-291.

WONG, K.Y. and CHEN, Q., 2012Nature of an attitudes toward learning mathematics questionnaire, *Mathematics Education Research Group of Australasia* 2012, NIE, National Institute of Education.

WOOD, K., PARSONS, D., GASSON, J. and HADEN, P., 2013It's Never Too Early: Pair Programming in CS1, *ACE '13*, January 29 - February 01, 2013 2013, Australian Computer Society, Inc., pp. 13–21.

WULF, T., 2005Constructivist Approaches for Teaching Computer Programming, 2005, ACM, pp. 245–248.

Liite 1: Motivaatiokysely

Motivaatiokysely

1. Kun huomaan tehneeni virheen tehtävässä, pyrin selvittämään virheen syyn.
2. Osaan ratkaista loogista päättelyä vaativia ongelmia.
3. Ohjelmointi on tylsää.
4. Mielestäni ViLLE-tehtävät ja -tutoriaalit ovat hyödyllisiä.
5. En halua miettiä vaihtoehtoisia tapoja tehtävän ratkaisemiseksi.
6. Koen ohjelmoinnin hyödylliseksi taidoksi.
7. Ohjelmointikurssit ovat mielestäni tärkeitä.
8. Koen että matematiikasta on apua ohjelmoinnin oppimisessa.
9. Kun olen ratkaissut ongelman, palaan ratkaisuuni ja tarkistan, olenko tehnyt virheitä.
10. Ohjelmointi on helppoa.
11. Ohjelmointitehtävien tekeminen tuntuu minusta hyvältä.
12. ViLLE-tehtävät ja -tutoriaalit ovat auttaneet minua ohjelmoinnin oppimisessa.
13. Keksin usein monta vaihtoehtoista ratkaisua ohjelmointiongelmaan.
14. Ohjelmointi auttaa minua ymmärtämään miten eri nettisivut ja sovellukset toimivat.
15. Pidän ohjelmointia tärkeänä osana työelämässä tarvittavia taitoja.
16. En pidä matematiikkaa tärkeänä osana ohjelmointia.
17. Kun olen saanut ViLLE-tehtävään vastauksen, en tarkista vastaustani.
18. Koodin tarkoituksen hahmottaminen on minulle haastavaa.
19. Ohjelmointi tuntuu minusta yleisesti ottaen mukavalta.
20. Netistä löytyvät harjoitussivustot ja -oppaat auttavat minua oppimaan ohjelmointia.
21. Yritän ymmärtää muiden opiskelijoiden erilaisia tapoja ratkaista tehtäviä.
22. Ohjelmointitaito on tärkeää
23. Koen ohjelmointikurssien olevan hyödyllisiä ohjelmointitaitojeni kehittämisessä.
24. Minusta on helppoa muodostaa vastaus sanallisesta matematiikan tehtävästä.
25. Kun olen ratkaissut ongelman, kysyn itseltäni, onko vastauksessa järkeä.
26. Ohjelmointitehtävien ratkaiseminen on minusta hauskaa.
27. Opin paremmin, kun joku auttaa minua ohjelmointitehtävien tekemisessä.
28. Käytän riittävästi aikaa ViLLE-tehtäviin ja -tutoriaaleihin.
29. Minua kiinnostaa mitä ohjelmoinnin avulla voi saada aikaan.
30. Asenteeni ohjelmointia kohtaan on muuttunut AOP-kurssin alusta. (Vain loppukyselyssä)