Kollaboratiivinen ongelmanratkaisu ja debuggaus neljäsluokkalaisten pariohjelmointitilanteissa

Sini-Anna Salmela

Kasvatustieteen pro gradu –tutkielma
Kevätlukukausi 2018
Kasvatustieteiden laitos
Jyväskylän yliopisto

TIIVISTELMÄ

Salmela, Sini-Anna. 2018. Kollaboratiivinen ongelmanratkaisu ja debuggaus neljäsluokkalaisten pariohjelmointitilanteissa. Kasvatustieteen pro gradu -tutkielma. Jyväskylän yliopisto. Opettajankoulutuslaitos. 71 sivua.

Tutkimuksessa tarkastellaan erään koulun 4. luokan oppilaiden ohjelmoinnillista ajattelua ohjelmoinnin harjoitteluun keskittyvässä opetuskokonaisuudessa. Erityisesti keskitytään yhteen ohjelmoinnillisen ajattelun osa-alueeseen: debuggaukseen eli virheenkorjaukseen. Koska oppilaat työskentelivät pareittain, myös heidän keskinäistä vuorovaikutustaan ja yhteistyötään tarkastellaan erityisesti kollaboratiivisen ongelmanratkaisun näkökulmasta.

Kyseessä on laadullinen tapaustutkimus. Aineisto kerättiin videoimalla oppilaiden luonnollista parityöskentelyä debuggaukseen keskittyneillä tunneilla, joilla tutkija myös osallistui oppilaiden työskentelyyn ohjaajan roolissa. Videoinnin lisäksi oppilaiden työskentelystä tallennettiin näyttökaappaukset, joilta käy yksityiskohtaisemmin ilmi, kuinka oppilaat yrittivät ratkaista ongelmia. Aineiston analyysiin käytettiin sekä temaattista että diskurssianalyysia.

Tulokset osoittavat oppilaiden käyttäneen ongelmanratkaisussaan pääasiassa kahta strategiaa: niin kutsuttuja tarkan korjaamisen sekä kokeilun ja erehdyksen strategioita. Oppilaiden debuggausprosessin kulku vaihteli valitun strategian mukaan. Tarkan korjaamisen strategiaa käytettäessä prosessissa oli havaittavissa suunnitelmallisuutta ja reflektointia, jolloin ongelmanratkaisu oli myös tehokasta. Kokeilun ja erehtymisen strategiassa ongelmaa taas lähestyttiin toiminnan kautta, reflektoiva vaihe jäi ongelmanratkaisuprosessista pois, eikä ongelmanratkaisu ollut yhtä tehokasta kuin tarkan korjaamisen strategialla. Yhden oppilasparin kohdalla vuorovaikutuksen ongelmat estivät ongelmanratkaisun kokonaan.

Tuloksista voidaan päätellä, että ohjelmoinnillisen ajattelun harjoittelemisessa on syytä kiinnittää huomiota tehokkaiden ongelmanratkaisun strategioi-

den harjoitteluun. Oppilaiden on tärkeää tutustua ohjelmoinnillisiin konsepteihin ja käytettävään ohjelmointikieleen. Lisäksi oppilaita tulee ohjata debuggauksessaan suunnitelmallisuuteen ja yhteistyöhön.

Asiasanat: ohjelmoinnillinen ajattelu, debuggaus, pariohjelmointi, kollaboratiivinen ongelmanratkaisu, 2000-luvun taidot, OPS 2014

SISÄLTÖ

SISÄLTÖ

1 JOHDANTO

Teknologia on kehittynyt muutamassa vuosikymmenessä valtavasti, ja tämä informaatio- ja viestintäteknologian muutos on muuttanut myös käsitystämme maailmasta ja tiedosta (Dede 2009). Digitalisoituneessa nyky-yhteiskunnassa rutiininomainen fyysinen ja kognitiivinen työ on siirtynyt robottien ja tietokoneiden tehtäväksi. Tämän myötä ihmisiltä edellytetään työntekijöinä ja kansalaisina yhä enemmän niin sanottuja 2000-luvun taitoja (mm. OECD 2013), kuten vuorovaikutustaitoja, kriittistä ajattelua, TVT-taitoja, innovatiivisuutta ja luovaa ongelmanratkaisukykyä.

Nykyisissä perusopetuksen opetussuunnitelman perusteissa ohjelmointi on osa matematiikan opetussisältöjä (OPH 2014, 129 & 235). Lisäksi se mainitaan tieto- ja viestintäteknologian laaja-alaisen osaamisalueen yhteydessä (OPH 2014, 157). Perusopetuksen opetussuunnitelman perusteiden mukaan (OPH 2014, 101) ohjelmointi on tapa tuottaa digitaalista sisältöä ja saada kokemuksia "digitaalisen median parissa työskentelystä". Lisäksi se on keino oppia ymmärtämään "miten teknologian toiminta riippuu ihmisen tekemistä ratkaisuista" (OPH 2014, 157). Ohjelmointi linkittyy tämän lisäksi oppiainerajat ylittävästi moniin 2000-luvun taitoihin, kuten ongelmanratkaisuun, itsensä ilmaisemiseen ja luovuuteen, monilukutaitoon ja jopa vuorovaikutustaitoihin.

Ohjelmoinnin opettamisella on myös yhteiskunnallinen merkitys, sillä monet yhteiskunnan rakenteet ovat jo nyt erilaisten teknologisten ohjelmien varassa. Digitalisaation myötä olemme yhä riippuvaisempia erilaisista sähköisistä järjestelmistä sekä kansalaisina että työntekijöinä. Tämän vuoksi ohjelmointitaidon uskotaan tulevaisuudessa olevan lukutaidon perusta (diSessa 2000; Prensky 2008; Vee 2013). Ottaen huomioon, kuinka paljon jokapäiväinen elämämme sisältää koodia huomaamattamme, on tärkeää, ettei ohjelmointi ole vain marginaalisen ryhmän taito. Perusopetuksen kautta yhä useampi saa mahdollisuuden oppia ohjelmointia ja innostua siitä. Tämä ei tarkoita, että kaikkien tulisi osata ohjelmoida, vaan tärkeämpää on, että ohjelmointia koskevat ennakkoluulot vähenevät ja yhä useampi ymmärtää, kuinka ohjelmointi toimii.

Ohjelmoinnin opettamiseen lapsille on kehitetty lukuisia työkaluja, joiden avulla voi harjoitella ohjelmoinnin alkeita ymmärrettävällä ja motivoivalla tavalla. Kiinnostuin aiheesta talvella 2015 tutustuttuani sattumalta Linda Liukkaan Hello Ruby –lastenkirjaan, jonka avulla lapsille voi opettaa ohjelmoinnillista ajattelua. Tein kandidaatin tutkielmani ohjelmoinnin harjoittelun yhteydestä lasten sisäiseen motivaatioon ja flow-tilaan. Tulokset viittasivat siihen, että ohjelmoinnin harjoittelun yhteys sisäiseen motivaatioon ja flow'hun oli vahva. Lasten autoteelinen kokemus oli erityisen vahva, he siis kokivat tekemisen itsessään mielyttäväksi. Mitä enemmän tutustuin ohjelmoinnin opetukseen, sitä kiinnostuneemmaksi tulin. Tämän takia päätin jatkaa aiheen parissa myös pro gradu –tutkielmassani.

Ohjelmointi on siis oppilaiden mielestä motivoivaa. Lisäksi perusopetuksen perusteissa ohjelmoinnin uskotaan kehittävän ennen kaikkea tieto- ja viestintäteknologisia taitoja (OPH 2014 101, 157) sekä ajattelun taitoja (OPH 2014 120, 235). Ohjelmoinnissa tarvittavia ajattelun taitoja kutsutaan ohjelmoinnilliseksi ajatteluksi (Wing 2011), ja tämä tutkimus keskittyy tarkastelemaan alakoulun ohjelmointia juuri ohjelmoinnillisen ajattelun näkökulmasta. Tutkimusta aiheesta on vielä hyvin vähän. Lye ja Koh (2014) huomauttavat kirjallisuuskatsauksessaan, että alakouluun sijoittuvista tutkimuksista suuri osa (85%) keskittyy ohjelmoinnillisten konseptien oppimiseen, kun taas ohjelmoinnilliset käytänteet, kuten debuuggaus ja ongelmanratkaisu jäävät tutkimuksissa vähemmälle huomiolle. Haluan tutkielmani kautta tarkastella, voiko ohjelmointi opettaa lapsille 2000-luvun taitoja, erityisesti ajattelun taitoja ja ongelmanratkaisukykyä.

2 OHJELMOINNILLINEN AJATTELU

2.1 Ohjelmoinnillinen ajattelu

Ohjelmoinnillisella ajattelulla tarkoitetaan niitä ajattelun taitoja ja kognitiivisia prosesseja, joita tarvitaan, kun ratkotaan todellisen elämän ongelmia tietokoneen avulla (Wing 2011; Buckley 2012). Jo lähes neljäkymmentä vuotta sitten Seymour Papert (1980) tutki, kuinka tietokoneen kanssa työskentely vaikutti lasten oppimiseen. Papert oli erityisesti kiinnostunut siitä, kuinka ohjelmoinnin kautta lapsille voitaisiin opettaa ajattelun taitoja. Hänen mukaansa lapset oppivat syvällisemmin, kun he saavat rakentaa omia projektejaan reflektoiden toimintaansa yhdessä toisten oppijoiden kanssa. Hän myös havaitsi lasten kielentävän omaa ajatteluaan LOGO-ohjelmointiympäristössä työskennellessään, ja uskoi, että ”tietokoneet johtavat todennäköisesti harkitumpaan itsetajuiseen ajatteluun” (Papert 1980, 38). Tällä hän tarkoitti, että tietokoneiden käyttö matematiikan oppimisessa tukisi metakognitiivisten taitojen kehittymistä ja opettaisi oppilaat ajattelemaan ajattelua. (Papert 1980.)

Papert’n työ johti siihen, että ohjelmointia alettiin tutkia yhä enemmän myös pedagogisesta näkökulmasta (esim. Carver 1988; Katz & Anderson 1987). Suurin osa kuitenkin piti ohjelmointia liian vaikeana ja jopa turhana asiana opettaa kaikille lapsille, niinpä ohjelmointi ei vielä saanut jalansijaa kouluissa. Innokkaimmat ohjelmoinnin opetuksen kannattajat kuitenkin jatkoivat tutkimusta ohjelmoinnin opetuksen hyödyistä ja kehittivät parempia työkaluja ohjelmoinnin opettamisen tueksi (esim. Cope & Simmons 1994; diSessa 2000).

Ohjelmoinnin rooli perusopetuksessa nousi puheenaiheeksi uudelleen, kun Wing (2006) teki ohjelmoinnillisen ajattelun termin tunnetuksi. Hän määritteli ohjelmoinnillisen ajattelun ”tietojärjestelmätieteilijän tavoin ajattelemiseksi” ja hänen mukaansa se sisältää ongelmanratkaisua, järjestelmien suunnittelua ja ihmisten käyttäytymisen ymmärtämistä tietojärjestelmätieteiden perusteella (Wing 2006). Myöhemmin Wing tarkensi määritelmäänsä: ohjelmoinnillisella

ajattelulla tarkoitetaan ongelmien ja niiden ratkaisujen muodostamista sellaiseen muotoon, että ne ovat tietokoneen suoritettavissa (Wing 2011).

2010-luvun alusta lähtien ohjelmoinnillista ajattelua on pyritty lisäämään kouluissa ympäri maailman, ja ohjelmoinnilliselle ajattelulle on kehitetty lukuisia määritelmiä juuri kasvatusalan näkökulmasta (mm. CSTA 2011; CAS 2015). Vuoden 2014 (Balanskat & Engelhardt 2014) tilastoista huomataan, että lähes kaikki Euroopan maat olivat jo silloin integroineet tai suunnittelivat integroivansa ohjelmoinnin opetussuunnitelmiinsa. Eri maiden opetussuunnitelmissa painotetaan eri asioita, esimerkiksi Suomessa ja Ruotsissa ohjelmointi on osana laaja-alaista osaamista ja matematiikan oppisisältöjä, kun taas Tanskan ja Norjan opetussuunnitelmissa ohjelmointi on osa tietojärjestelmätieteiden oppiainetta (Bocconi, Chioccariello, Earp 2018). Ohjelmoinnillisen ajattelun innostunut vastaanotto ja nopea yleistyminen opetussuunnitelmissa ympäri maailman on herättänyt myös kritiikkiä. Mielipiteet ohjelmoinnillisen ajattelun merkityksestä ovat jakautuneet kahteen linjaan: toisten mielestä ohjelmoinnillinen ajattelu on jokapäiväistä luovaa ongelmanratkaisua, toisten mielestä sitä tarvitaan ensisijaisesti tietokoneiden parissa työskennellessä (Bocconi ym. 2018).

Esimerkiksi Wingin (2006) mielestä ohjelmoinnillisen ajattelun tulisi olla osa jokaisen lapsen analyyttistä osaamista. Nykymaailmassa, jossa tietokoneet ratkovat nopeasti valtavia määriä algoritmisia ongelmia ihmisen puolesta, ihmismielen ratkaistavaksi ovat jääneet avoimet, moniulotteiset ja kriittistä ajattelua vaativat ongelmat (Buckley 2012). Ohjelmoinnillinen ajattelu tarjoaa välineet paitsi yksiviivaisten ongelmien, myös monimutkaisten tosielämän ongelmien ratkaisemiseen (Shute, Sun & Asbell-Clarke. 2017). Ohjelmoinnillista ajattelua tarvitaan Buckleyn (2012) mukaan ongelmanratkaisuun, jossa yhdistyvät korkean tason ajattelun taidot – kuten luovuus, kriittinen ajattelu ja päätöksenteko – sekä tietokoneen käyttö ongelmanratkaisun välineenä. Lisäksi ohjelmoinnillinen ajattelu pitää sisällään asenteita ja luonteen ominaisuuksia, joita monimutkaisten ongelmien ratkaiseminen vaatii, kuten kestävyyttä ja epävarmuuden sietämistä,

itsevarmuutta ja vuorovaikutustaitoja (Barr, Harrison & Conery 2011). Ohjelmoinnillinen ajattelu on siis paitsi ohjelmoinnissa, myös tosielämässä tarvittavia ajattelun ja ongelmanratkaisun taitoja.

Kriitikoiden mukaan ohjelmoinnillinen ajattelu on tärkeä taito tietokoneiden ja ohjelmoinnin kanssa työskennellessä, mutta ei ole sellaisenaan sovellettavissa kaikille aloille tai kaikkiin arkipäivän tilanteisiin, eikä siksi ole tarpeellinen taito opettaa laajasti. Denning (2017) myöntää, että ohjelmoinnillisen ajattelun opettaminen kaikille kuulostaa hienolta teoriassa, mutta käytännössä ohjelmoinnillisen ajattelun määritelmä on hänen mukaansa liian häilyvä, ja oppimista on siksi vaikeaa arvioida. Ohjelmoinnillisen ajattelun laaja-alaista hyödyllisyyttä ei myöskään ole empiirisesti todistettu, vaikka siitä on puhuttu jo vuosikymmeniä. (Denning 2017.)

Wingin (2006) mukaan ohjelmoinnillinen ajattelu pitää sisällään viisi kognitiivista prosessia: ongelman uudelleenmäärittely, rekursio, ongelman jakaminen osiin, abstraktio ja systemaattinen testaaminen. *Ongelman uudelleenmäärittelyllä* tarkoitetaan ongelman rajaamista helpommin ratkaistavaan muotoon. *Rekursio* on matemaattinen termi, mutta ohjelmoinnillisen ajattelun yhteydessä sillä tarkoitetaan tapaa ajatella kehämäisesti ja kasautuvasti niin, että uusi ajatus rakentuu aina edelliselle. *Ongelman jakaminen osiin* on tyypillinen ongelmanratkaisun keino, jossa yksi suuri ongelma jaetaan pienempiin, helpommin ratkaistaviin osiin. *Abstraktio* tarkoittaa kykyä muuntaa monimutkaisia ongelmia tai ideoita esitettävään muotoon tai malliksi. *Systemaattinen testaaminen* puolestaan on aktiivista ja järjestelmällistä pyrkimistä ratkaisuun. (Wing 2006; Shute ym. 2017.)

Resnick ja Brennan (2012) määrittelevät ohjelmoinnillisen ajattelun kokonaisuutena, jossa on kolme ulottuvuutta: *ohjelmoinnilliset konseptit, ohjelmoinniliset käytänteet* ja *ohjelmoinnilliset perspektiivit*. Grover ja Pea (2018) seuraavat samaa kolmijakoa, mutta painottavat ulottuvuuksien sisällä eri asioita kuin Resnick ja Brennan. Seuraavissa alaluvuissa tarkastellaan konsepteja, käytänteitä ja perspektiivejä lyhyesti näiden artikkelien pohjalta. Koska tässä tutkimuksessa keskitytään erityisesti oppilaiden debuggaustaitoon, debuggaukselle on ohjelmoinnillisten käytänteiden yhteydessä oma alalukunsa.

2.2 Ohjelmoinnilliset konseptit

Ohjelmoinnilliset konseptit ovat erilaisia toimintoja ja taitoja, joita tarvitaan ohjelman rakentamiseen. Tällaisia konsepteja ovat Resnickin ja Brennanin (2012) mukaan esimerkiksi algoritmit, sekvenssit, tapahtumat, rinnakkaisuus, loopit, operaattorit, data ja ehtolauseet. Nämä kaikki ovat ohjelmoinnin perustermistöä, joka tulee aloittelevalle ohjelmoijalle tutuksi melko nopeasti hänen alkaessa ohjelmoida. (Resnick & Brennan 2012.)

Siinä missä Resnick ja Brennan painottavat teknisiä ohjelmoinnin rakenteita, Grover ja Pea (2018) puolestaan ottavat ohjelmoinnillisiin konsepteihin mukaan myös ajattelun taitoja. He määrittelevät ohjelmoinnillisiksi konsepteiksi logiikan ja loogisen ajattelun, algoritmit ja algoritmisen ajattelun, kaavat ja kaavojen tunnistamisen, abstraktion ja yleistämisen, arvioinnin sekä automaation (Grover & Pea 2018).

Oppilaiden on havaittu oppivan käyttämään varsinkin yleisimpiä konsepteja tekemisen kautta ilman erityistä ohjausta, mutta harvinaisempien konseptien, kuten muuttujien käyttö vaatii todennäköisesti ohjausta (Maloney, Peppler, Kafai, Resnick & Rusk 2008). Oppilaiden on myös havaittu hyötyvän siitä, että ohjaaja tietoisesti sanallistaa ohjelmoinnillisia konsepteja heille (Grover ja Pea 2012).

2.3 Ohjelmoinnilliset käytänteet

Ohjelmoinnillisilla käytänteillä tarkoitetaan laajempaa osaamista: erilaisia tapoja työskennellä ohjelmoinnin parissa (Resnick ja Brennan 2012; Grover & Pea 2018). Näitä ovat ohjelman rakentaminen vähitellen pienissä erissä, suunnitelmien joustavuus ja kehämäinen työskentely (*being incremental ja iterative*), luovuus sekä ohjelman testaaminen ja debuggaus (Resnick & Brennan 2012). Näiden lisäksi Grover ja Pea (2018) lisäävät ohjelmoinnillisiin käytänteisiin ongelman jakamisen osiin (*problem decomposition*) sekä ohjelmoinnillisten rakennelmien kehittämisen (*creating computational artefacts*).

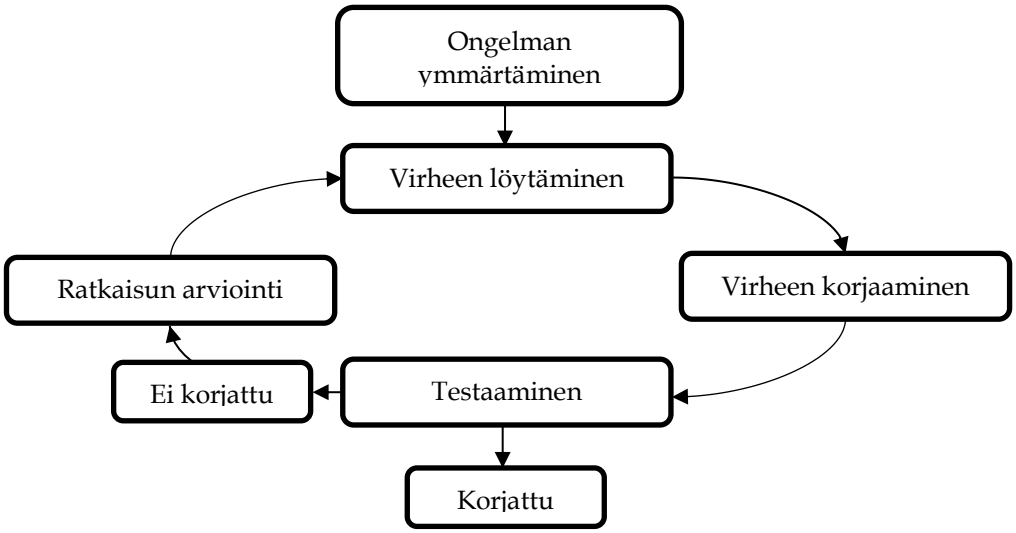
2.3.1 Debuggaus

Tämä tutkimus keskittyy ennen kaikkea debuggaukseen ongelmanratkaisun muotona. *Debuggaus* on olennainen taito ohjelmoinnissa ja sillä tarkoitetaan ohjelman virheiden (*bug*) löytämistä ja niiden korjaamista (McCauley, Fitzgerald, Lewandowski, Murphy, Simon, Thomas & Zander 2008). Perimmiltään virheenkorjaus on ongelmanratkaisua, joka vaatii ohjelmoinnillisten konseptien ja käytössä olevan ohjelmointikielen tuntemista. Debuggauksen vastaavana käsitteenä voitaisiin käyttää myös suomenkielistä termiä *virheenkorjaus*. Tässä tutkimuksessa käytän kuitenkin termiä debuggaus, sillä se on ohjelmoinnin yhteydessä yleisesti käytettävä termi ja viittaa mielestäni selkeämmin kokonaiseen ongelmanratkaisuprosessiin, jossa virheen korjaaminen on vain yksi vaihe.

Yksi varhaisimpia ongelmanratkaisun malleja on Polyan (1945) kehittämä malli, jossa ongelmanratkaisuprosessin vaiheita ovat ongelman ymmärtäminen, suunnitelman tekeminen, suunnitelman toteuttaminen ja ratkaisun arviointi (ks. luku 3.2.2). Debuggausprosessi seuraa hyvin samanlaista kaavaa. Katz ja Anderson (1987) toteavat debuggauksen olevan verrannollinen tiettyyn ongelmanratkaisun alalajiin, *troubleshootingiin*, josta on löydettävissä neljä vaihetta: ymmärtäminen, testaaminen, ongelman löytäminen ja korjaaminen. Vessey (1985) puolestaan havaitsi debuggauksen välivaiheita olevan ongelman määrittäminen, ohjelman rakenteeseen tutustuminen, testaaminen, ohjelman arviointi ja hypoteesin muodostaminen sekä virheen korjaaminen.

Kuviossa 1 on esitetty näihin teorioihin perustuva näkemys debuggauksen kehästä. Kaikki mallit painottavat ongelman ratkaisun alkavan ongelman ymmärtämisen vaiheesta. Ennen varsinaisen virheen etsimistä ja korjaamista on tärkeää ymmärtää, mitä ohjelman halutaan tekevän, ja mitä se sen sijaan tekee (*discrepancy*) (Klahr ja Carver 1988). Vasta sitten, kun ongelma ymmärretään, voidaan alkaa etsiä ongelman aiheuttajaa eli virhettä. Kun virhe on löydetty ja korjattu, siirrytään testaamisvaiheeseen. Debuggauksen malleissa (Katz & Anderson 1987; Vessey 1985) ei ole mainittu ratkaisun arviointia, mutta Polyan (1945) mallissa se on tärkeässä asemassa. Ratkaisun arvioinnin vaiheessa reflektoidaan, mikä toimi

ja mikä ei, ja tällöin tapahtuu myös oppimista ja soveltamista (Polya 1945). Debuggauksen kehällä ratkaisun arvioinnin vaiheessa voidaan pohtia, onko virhe sittenkin jossain muualla tai pitäisikö se korjata eri tavalla. Kehä pyörii niin kauan, että testaamisen jälkeen ohjelma todetaan korjatuksi (Katz & Anderson 1987).



KUVIO 1. Debuggauksen kehä (vrt. Polya 1945; Katz & Anderson 1988; Vessey 1985.)

Aiemmassa tutkimuksessa on havaittu erilaisia debuggauksen strategioita. Wyeth (2008) on todennut, että aiemmin kohdattujen rakenteiden uudelleenkäyttö on myös tärkeä ohjelmoinnin taito, eli aiemmissa ongelmatilanteissa havaittuja ratkaisuja voidaan soveltaa uuden ongelman ratkaisemiseksi. Wyethin (2008) mukaan myös ohjelman jakaminen osiin ja osien testaaminen erikseen voi olla yksi debuggausstrategia. Aloittelevien ja edistyneiden debuggaajien strategioissa on myös havaittu eroja. Edistyneet debuggaajat pyrkivät ymmärtämään ohjelman ja kykenevät näin löytämään virheen nopeammin (Ahmadzadeh, Elliman & Higgins 2005).

Toisten strategioiden on todettu olevan tehokkaampia kuin toisten. Kokonaiskuvan ymmärtämisen on havaittu olevan vasta-alkajille vaikeaa. Aloittelijoiden on myös vaikeaa joustaa ratkaisuehdotuksestaan, kun taas ekspertit ovat valmiita luopumaan tekemistään hypoteeseista, jos huomaavat niiden olevan virheellisiä. (Vessey 1985.)

Hyvillä debuggaajilla on yleensä käytössään enemmän tietoa kuin heikoilla (Ahmadzadeh, ym. 2005). Kokeneet ohjelmoijat aloittavat virheen korjauksen kokonaiskuvan tarkastelusta, aloittelijat taas pyrkivät ainoastaan löytämään virheen ja korjaamaan sen, ymmärtämättä itse ohjelmaa ensin (Vessey 1985). Taitava debuggaaja myös tietää, millaisia virheitä ohjelmissa esiintyy usein ja kuinka ne korjataan. Myös yleinen ohjelmoinnin osaaminen sekä käytössä olevan ohjelmointikielen tuntemus on tietoa, joka auttaa debuggausprosessissa (Ahmadzadeh ym. 2005).

Kaiken tämän tiedon ansiosta taitava debuggaaja pystyy korjaamaan ohjelman nopeasti ja tarkasti, usein yhden puhtaan virheenkorjauskierroksen (ymmärtäminen – testaaminen – virheen löytäminen – virheen korjaaminen) aikana. Klahr ja Carver (1988) kutsuvat tällaista strategiaa ”keskittyneen haun” (*focused search*) strategiaksi. Keskittyneessä haussa debuggaaja tekee päätelmiä ohjelmasta tietojensa perusteella ja keskittyy etsimään virhettä sieltä, mistä uskoo todennäköisimmin sen löytyvän. Vähemmän kokeneet debuggaajat puolestaan turvautuvat ”raa’an voiman” (*brute-force*) strategiaan, jossa he käyvät koodia läpi rivi riviltä yrittäessään paikantaa virhettä. Molemmilla strategioilla ohjelma saatiin korjattua, mutta ensimmäinen oli nopeampi. (Klahr & Carver 1988.)

2.4 Ohjelmoinnilliset perspektiivit

Ohjelmoinnillisilla perspektiiveillä viitataan laajempaan ymmärrykseen itsestä suhteessa muihin ja suhteessa teknologiseen maailmaan. Ohjelmoinnillisen ajattelun perspektiivejä ovat ilmaisu, yhdistäminen ja kyseenalaistaminen. Ilmaisu pitää sisällään sisällön tuottamisen sen sijaan, että ainoastaan kuluttaisi sitä, mitä muut ovat ohjelmoineet. Ohjelmoinnillinen ajattelija näkee itsensä aktiivisena ja luovana tuottajana. Yhdistäminen viittaa ohjelmoinnillisen ajattelijan kykyyn nähdä itsensä osana yhteisöä, joka ohjelmoidessaan on jatkuvassa vuorovaikutuksessa muiden kanssa. Vuorovaikutteisuus on vahvassa roolissa ohjelmoijien kesken; kukaan ei ohjelmoi vain itseään varten, vaan koodaus tapahtuu sekä muiden kanssa että muita varten. Kyseenalaistaminen taas tarkoittaa sitä,

että ohjelmoija ymmärtää, kuinka hän voi ohjelmoinnin avulla vaikuttaa maailmaan. Hän oppii näkemään ohjelmoinnin merkityksen maailmassa sen sijaan, että ottaisi ohjelmoinnin vaikutukset itsestään selvyyksinä. (Resnick & Brennan 2012.)

2.5 Visuaaliset ohjelmointikielet, esimerkkinä Scratch

Lasten ohjelmoinnillisen ajattelun oppimisen tueksi on kehitetty monia työkaluja. Yksi vaihtoehto ovat visuaaliset ohjelmointikielet, joissa monimutkaisen ja hienovaraisen koodin kirjoittamisen sijaan ohjelmointi tapahtuu graafisia osia siirtelemällä ja yhdistelemällä. Tässä tutkimuksessa ohjelmoinnillisen ajattelun harjoitteluun käytettiin Scratchiä, joka on yksi suosituimmista ja eniten tutkituista (esim. Calder 2010; Lopez & Hernandez 2015; Maloney, Resnick, Rusk, Silverman & Eastmond 2010; Resnick ym. 2009) visuaalisista ohjelmointikielistä. Visuaalisten ohjelmointikielten on todettu olevan toimiva tapa opettaa oppilaille ohjelmointia ja ohjelmoinnillista ajattelua, koska niissä käskyt muistuttavat usein puhuttua kieltä ja ovat siksi helppoja ymmärtää (Lye & Koh 2014). Scratchissä käyttäjä voi luoda omaa mediaansa, kuten interaktiivisia tarinoita tai pelejä yhdistelemällä erilaisia visuaalisia lohkoja toisiinsa käskysarjojen luomiseksi. Lohkot ovat siinä muodossa, että ne liittyvät toisiinsa aina loogisesti (ks. esimerkiksi luku 6.2.1 kuva 1). Lapselle ohjelmointilohkojen yhdisteleminen on helpompaa kuin sanallisen koodin kirjoittaminen (Maloney ym. 2010).

Scratch on suunniteltu niin, että se tukee ohjelmoinnillisen ajattelun ja debuggaustaidon kehittymistä. Koska oppilas ei itse kirjoita koodia, vaan käyttää käskysarjojen rakentamiseen lohkoja, vältytään kirjoitusvirheen aiheuttamilta ”bugeilta” (Kelleher & Pausch 2005). Koska lohkot yhdistyvät loogisesti, ohjelma tekee aina jotakin, vaikkakaan ei välttämättä sitä, mitä ohjelmoija haluaa. Niinpä oppilas saa toiminnastaan aina konkreettisen palautteen eikä pelkkää virheilmoitusta. Tämä ohjaa hänen ajatteluaan, auttaa virheen paikantamisessa ja motivoi työskentelemään (Maloney ym. 2010).

Luovuuteen ja itseilmaisuun kannustavien visuaalisten ohjelmointiympäristöjen on todettu olevan tasapuolisesti molempia sukupuolia kiinnostava tapa opetella ohjelmoimaan. Craig ja Horton (2009) toteuttivat tutkimuksen, jossa 85 tyttöä osallistui tytöille suunnatulle Scratch-leirille. Leirin jälkeen osallistujia pyydettiin nimeämään tulevaisuuden toiveammattejaan, ja tietojärjestelmätieteet oli toisiksi suosituin vaihtoehto lääkärin uran jälkeen. Koska tietokoneet ja ohjelmointi ovat perinteisesti olleet ennemminkin poikien kuin tyttöjen kiinnostuksen kohde, tyttöjen havaittu kiinnostus opetella ohjelmointia graafisen ohjelmointiympäristöjen avulla on merkittävä huomio demokratian ja tasa-arvon kannalta. (Maloney, Peppler, Kafai, Resnick & Rusk 2008; ks. myös Grover & Pea 2012.)

Toisaalta visuaaliset ja interaktiiviset ohjelmointiympäristöt ovat saaneet osakseen myös kritiikkiä. Koska oppilas saa niistä jatkuvaa visuaalista palautetta, niiden on havaittu lisäävän kokeilun ja erehtymisen strategiaa sen sijaan, että oppilas suunnittelisi työtään etukäteen (Cope & Simmons 1994). Toisaalta arvailu sekä kokeilu ja erehtyminen ovat ongelmanratkaisun strategioita, joita voidaan hyödyntää erityisesti tietokoneella, kun monien arvausten testaaminen on nopeaa (Barr, Harrison & Conery 2011). Kuitenkin tällöin esimerkiksi matemaattisten käsitteiden, kuten astelukujen tai etäisyyden ymmärtäminen jää heikoksi, kun oppilas voi arvaamalla haarukoida oikean vastauksen (Hillel, Kieran & Gurtner 1989). Myöskään korkean tason ajattelu, kuten suunnitelmien tekeminen tai hypoteesien muodostaminen ja testaaminen saattavat jäädä visuaalisessa ohjelmointiympäristössä kehittymättä, jos niitä ei tietoisesti harjoitella, sillä ongelmat ovat ratkaistavissa matalan tason ajattelun keinoin (Simmons & Cope 1993).

3 NÄKEMYKSIÄ TULEVAISUUDEN PERUSTAIDOISTA

3.1 2000-luvun tietoyhteiskunnan perustaidot

Viimeisten vuosikymmenien aikana maailma on muuttunut hyvin nopeasti. Yhteiskunnan ja työmarkkinoiden muutokseen on vaikuttanut kolme suurta tekijää: teknologinen muutos, globalisaatio ja demografia (Wilson 2013). Digitalisaation myötä monet perinteiset, fyysistä työtä vaatineet ammatit ovat katoamassa, ja niiden tilalle on tullut uudenlaista tietotaitoa, kuten digitaalista monilukutaitoa vaativia ammatteja (Griffin, Care & McGaw 2011). Tietoyhteiskunnan työelämässä tarvittavia taitoja kutsutaan myös 2000-luvun taidoiksi (esim. P21, 2007; Griffin ym. 2011; Binkley ym. 2012). Myös perusopetuksen opetussuunnitelman perusteissa (OPH 2014) painotetaan näitä tulevaisuuden taitoja erityisen laaja-alaisen osaamisen muodossa:

> Laaja-alaisen osaamisen lisääntynyt tarve nousee ympäröivän maailman muutoksista. Ihmisenä kasvaminen, opiskelu, työnteko sekä kansalaisena toimiminen nyt ja tulevaisuudessa edellyttävät tiedon- ja taidonalat ylittävää ja yhdistävää osaamista. (OPH 2014, 20.)

Uusia olennaisia taitoja ovat esimerkiksi uuden oppiminen, yhteistyön tekeminen ja ongelmanratkaisu (Griffin ym. 2011). Wengin (2015) mukaan tulevaisuuden työssä tarvittavat kahdeksan perustaitoa ovat ohjelmoinnillinen ajattelu, uusi medialukutaito, ”sense-making”, kolmen älykkyyden (sosiaalinen, emotionaalinen sekä kulttuurinen älykkyys) kehittäminen, ”design-thinking”, sopeutuva ajattelu, kognitiivisen kuorman helpottaminen ja kulttuurien välinen osaaminen. Care ja Griffin (2014) ovat vertailleet viittä eri määritelmää 2000-luvun taidoille, ja ne on esitetty taulukossa 1.

TAULUKKO 1. Vertailu 2000-luvun taitojen määritelmistä (Care & Griffin 2014, 370).

ATC21S (Binkley ym. 2012)	UNESCO (Delors ym. 1996)	OECD (2013)	Partnership for 21st Century Skills (2007)	Euroopan komissio (Gordon ym. 2009)
Ajattelun tavat: luovuus ja innovaatio kriittinen ajattelu ongelmanratkaisu päätöksenteko oppimaan oppiminen metakognitio	**Tietämään oppiminen**		**Oppiminen ja innovaatio:** luovuus kriittinen ajattelu ongelmanratkaisu	**Oppimaan oppiminen**
Työskentelyn tavat: kommunikaatio kollaboraatio	**Tekemään oppiminen**	**Heterogeenisissä ryhmissä toimiminen:** muihin ihmisiin suhtautuminen positiivisesti yhteistyö konfliktien hallinta ja ratkaisu	kommunikaatio kollaboraatio	kommunikaatio äidinkielellä ja vierailla kielillä
Työskentelyn välineet: informaation lukutaito TVT-lukutaito	**Tekemään oppiminen**	**Välineiden käyttäminen vuorovaikutuksellisesti:** kielen, symbolien ja tekstien käyttö tiedon ja informaation käyttö teknologian käyttö	**Informaatiomedia ja teknologia** informaation lukutaito medialukutaito TVT-lukutaito	matemaattiset, tieteelliset ja teknologiaosaaminen digitaalinen osaaminen
Maailmassa eläminen: kansalaisuus – paikallinen ja globaali elämä ja ura henkilökohtainen ja sosiaalinen vastuu – kulttuurinen tietoisuus ja osaaminen	**Olemaan oppiminen** **Yhdessä elämään oppiminen**	**Autonominen toiminta:** elämän suunnitelmien tekeminen oikeuksien puolustaminen	**Elämä ja ura:** joustavuus ja sopeutuminen aloitteen tekeminen sosiaalisen ja kulttuurien väliset taidot tehokkuus johtajuus vastuu	sosiaaliset kyvyt aloitteen tekeminen ja yrittäjyys kulttuurinen tietoisuus ja ilmaisu

Taulukosta 1 havaitaan, kuinka kaikissa eri määritelmissä taidot jäsentyvät hieman eri tavalla, mutta kaikissa on kuitenkin huomioitu samoja suurempia ylä-

kategorioita, esimerkiksi yhteistyötaidot tai globaalissa maailmassa elämisen taidot. Näiden yläkategorioiden sisällä taas tapahtuu erilaisia painotuksia yksittäisten taitojen suhteen.

Yksi määritelmistä 2000-luvun taidoille Organization for Economic Cooperation and Development -organisaation (OECD 2013) määritelmä, jonka mukaan avaintaitoja ovat autonominen ja reflektiivinen toiminta, työkalujen interaktiivinen käyttö sekä sosiaalisesti heterogeenisiin ryhmiin liittyminen ja niissä toimiminen. Toinen, erityisesti kasvatusalalle suunnattu, määritelmä on Partnership for 21st Century Skills -organisaation (P21 2007) kehittämä, ja siinä tärkeimmiksi taitoalueiksi nimetään niin sanotut 4C-taidot: kriittinen ajattelu (*critical thinking*), luovuus (*creativity*), kommunikointi (*communication*) ja yhteistyötaidot (*collaboration*). Lisäksi siinä painottuvat TVT-osaaminen ja -lukutaito sekä erilaiset elämäntaidot kuten johtajuus, etiikka, tuottavuus ja vastuuntunto (P21 2007).

Koska yksi koulun tärkeimmistä tehtävistä on valmistaa ihmisiä yhteiskunnan toimiviksi kansalaisiksi ja työntekijöiksi, maailman muutoksen myötä myös koulun tulee muuttua (Wilson 2013; OPH 2014, 18) ja mahdollistaa oppilaille ajattelun, joustavan ongelmanratkaisun, kollaboraation ja viestinnän taitojen oppimisen (Griffin ym. 2011). Suomessa vuonna 2016 voimaan tullut valtakunnallinen opetussuunnitelma sisällyttää 2000-luvun taitoja niin sanottuihin laaja-alaisiin oppimiskokonaisuuksiin. Näitä ovat 1) ajattelu ja oppimaan oppiminen, 2) kulttuurinen osaaminen, vuorovaikutus ja ilmaisu, 3) itsestä huolehtiminen ja arjen taidot, 4) monilukutaito, 5) tieto- ja viestintäteknologinen osaaminen sekä 6) työelämätaidot ja yrittäjyys (OPH 2014).

Silva (2008) toteaa 2000-luvun taitojen opettamisesta ytimekkäästi, että tärkeää ei ole se, mitä tietoa oppilailla on vaan se, kuinka he osaavat sitä hyödyntää. Tällä Silva viittaa luovuuteen ja luovaan tapaan ratkoa ongelmia. Floridan (2002) näkemys luovuudesta on hyvin samanlainen kuin Silvan (2008) näkemys 2000-luvun taidoista: hän vertaa tietoa ja informaatiota materiaaleiksi ja työkaluiksi, joista vain luovuuden avulla voidaan kehittää talouskasvua edistäviä innovaatioita (Florida 2002).

3.2 2000-luvun taidot ja ohjelmointi

Suomen valtakunnallisessa peruskoulun opetussuunnitelmassa (OPH 2014), ohjelmointi on mainittu enimmäkseen matematiikan yhteydessä. Ohjelmointi ei rajoitu kuitenkaan vain matematiikkaan, vaan sillä on suuri merkitys 2000-luvun taitojen harjoittelussa. Ohjelmoinnillinen ajattelu kulkee käsi kädessä muun muassa ongelmanratkaisun (Wing 2006; Buckley 2012), luovuuden (Wing 2006, deSchryver & Yadav 2015) ja monilukutaidon (Vee 2013) kanssa, vaikka ohjelmointia ei opetussuunnitelmassa (OPH 2014) mainitakaan näiden monialaisten taitojen yhteydessä. Ohjelmointi on monipuolisesti integroitavissa eri oppiaineisiin ja se on myös keino syventää ja soveltaa osaamista eri aloilla (ks. esim. Wilensky, Brady & Horn 2014). Nämä mahdollisuudet voivat kuitenkin helposti jäädä käyttämättä, ellei opettajalla ole tietoa, taitoa tai uskallusta hyödyntää ohjelmointia osana laaja-alaista opetusta.

Ohjelmoinnin opetusta on tutkittu paljon yliopisto- ja lukiotason koulutuksessa, mutta peruskoulussa ohjelmointi on vielä melko uutta. Barr ja Stephenson (2011) kuvailevat ohjelmoinnillista ajattelua peruskoulussa CSTA (Computer Science Teachers' Association) – ja ISTE (International Society for Technology in Education) –organisaatioiden edustajien pohdintojen perusteella. Keskustelujen pohjalta ohjelmoinnillinen ajattelu nähtiin peruskoulussa ennen kaikkea aktiivisena ongelmanratkaisuna. Tähän kuuluu tutkiva, utelias asenne ongelmaa kohtaan sekä rohkeus kokeilla erilaisia ratkaisuja virheistä huolimatta. Keskusteluissa todettiin myös, että aktiivisen tekemisen kautta oppilaiden tulisi ymmärtää ongelmanratkaisun luovaa ja joustavaa luonnetta ja tottua siihen, että ongelmia voi ratkoa hyvin monella eri tavalla. Ohjelmoinnillisen ajattelun kehittymisessä pidettiin tärkeänä myös vuorovaikutuksellista tapaa työskennellä. Lisäksi nostettiin esille, että työskentelyn lomassa oppilaat tottuvat käyttämään puheessaan luontevasti ohjelmointiin liittyvää termistöä. (Barr & Stephenson 2011.)

3.3 Ohjelmointi lukutaitona

Ohjelmoinnillisen ajattelun rinnalla on tärkeää käsitellä myös ohjelmoinnillisen lukutaidon käsitettä, sillä ne kulkevat rinnakkain ja osittain päällekkäinkin. Tietokoneiden yleistyessä 1980-luvulla keksittiin käsite *tietokonelukutaito* (*computer literacy*) joka määriteltiin mm. kyvyksi kertoa tietokoneelle, mitä sen halutaan tekevän (McMillanin 1996 mukaan Luehrman 1982). Kemeny (Veen 2013 mukaan Kemeny 1981) uskoi, että tiedon digitalisoituessa tietokonelukutaidosta tulisi edellytys työllistymiselle, ellei jopa selviytymiselle. Kuitenkin vielä 1980-luvulla tietokonelukutaidolla viitattiin lähinnä tekniseen tietokoneen käyttötaitoon, kuten hiiren tai tallennuslevyjen käyttöön (esim. Lombardi 1983). Tietokoneiden kehittyessä ja yleistyessä nopeasti myös tietokonelukutaidon määritelmä on ollut jatkuvassa muutoksessa, ja käsitteellä saatetaan tarkoittaa eri konteksteissa eri asioita. Besserin (Childersin 2003 mukaan Besser 1993) mielestä tietokonelukutaidon tulisi olla ennemminkin kykyä ajatella lineaarisesti ja loogisesti, ja tätä kautta kommunikoida tietokoneen kanssa. Tällainen tietokonelukutaito muistuttaa jo paljon enemmän ohjelmoinnillista ajattelua.

Maailman digitalisoituessa lukutaidon ja tekstien määritelmät ovat myös muuttuneet. Perinteisen lukutaidon rinnalle on tullut monilukutaito, joka Luukan (2013) mukaan on ”kykyä hankkia, muokata, tuottaa, esittää ja arvioida tietoja eri muodoissa ja erilaisten välineiden avulla.” Ohjelmoinnin kohdalla käytetään nykyään käsitettä *ohjelmoinnillinen lukutaito* (*computational literacy*), jonka voidaan ajatella olevan monilukutaidon yksi osa-alue. Ohjelmoinnillisella lukutaidolla tarkoitetaan taitoa käyttää tietokonetta ajattelun jatkeena, välineenä jäsentää maailmaa ja ilmaista ajatuksia (diSessa 2000). Vee (2013) huomauttaa, että myös perinteinen luku- ja kirjoitustaito oli aikoinaan vain harvojen etuoikeus, mutta nykyään länsimaisessa maailmassa luku- ja kirjoitustaitoa pidetään jo itsestäänselvyytenä. Samalla tavalla vain harvat ovat tällä hetkellä ohjelmoinnillisesti lukutaitoisia, mutta ohjelmoinnillisen lukutaidon uskotaan kuitenkin lopulta leviävän kaikille kansalaisille (Vee 2013). Ohjelmoinnillisen lukutaidon us-

kotaan tulevaisuudessa olevan jopa yhtä tärkeää, kuin perinteinen luku- ja kirjoitustaito on tänä päivänä (diSessa 2000; Vee 2013). Prenskyn (2008) näkemyksen mukaan ohjelmointitaito tulee lopulta olemaan niin oleellinen lukutaidon osa-alue, että se erottaa lukutaitoiset ja lukutaidottomat toisistaan. Toisaalta on myös ajateltu, että ohjelmointi tullaan tekemään niin helpoksi, että varsinaisen koodin kirjoittamista ei tulevaisuudessa tarvitsisi opetella lainkaan (ks. Vee 2013). Toistaiseksi tällaisestä kehityksestä ei kuitenkaan ole merkkejä, ja siksi ohjelmointi tulee nähdä yhtenä kirjoittamisen muotona (Vee 2013), jota kaikkien tulisi oppia ainakin ymmärtämään.

Näiden ennusteiden valossa ohjelmoinnin opetus koulussa on tärkeää. Kun ohjelmointi otetaan osaksi opetussuunnitelmaa, sallitaan kaikille yhtäläinen mahdollisuus oppia ymmärtämään maailmaa, josta suuri osa toimii erilaisten ohjelmistojen ja koodien varassa. Tavoitteena ei ole, että kaikista tulisi ohjelmoijia, vaan että kaikilla olisi oikeus oppia ohjelmoinnin perusteet ja innostua ohjelmoinnista.

Maailman digitalisoituessa myös tekstit ovat muuttuneet multimodaalisiksi, ja koulun tulee valmistaa oppilaita sekä tulkitsemaan että tuottamaan tällaisia tekstejä (OPH 2014, 22–23). Digitalisoituneessa nykymaailmassa tietoa on tarjolla valtavia määriä. Sisällön tuottaminen ja kuluttaminen esimerkiksi sosiaalisessa mediassa on helppoa, nopeaa ja jatkuvaa. Niin kutsutut diginatiivit ovatkin kehittyneet taitaviksi käsittelemään nopeaa informaatiovirtaa ja jakamaan huomionsa monien toisistaan erillään olevien asioiden kesken (Prensky 2001). Tällainen nopea prosessointi on oleellinen taito uudenlaisten tekstien käsittelyssä, jotta tärkeä tieto saadaan suodatettua vähemmän tärkeästä. Nopean prosessoinnin rinnalla on kuitenkin tärkeää ottaa huomioon, että myös hitaalle prosessoinnille on paikkansa, ja sitä tulee harjoitella myös koulussa. Kallionpää (2014) huomauttaa, että nopean tulkitsemisen ja tuottamisen rinnalla ohjelmointi voi olla keino tuottaa sisältöä hitaasti ja luovasti.

3.4 Kollaboratiivinen ongelmanratkaisu

Tämän tutkimuksen kannalta erityisen keskeinen 2000-luvun taito on kollaboratiivinen ongelmanratkaisu. Tulevaisuuden työelämän tärkeimpiä taitoja ovat ongelmanratkaisu, tiimityöskentely ja vuorovaikutustaidot (Luckin, Baines, Cukurova, Holmes & Mann 2017). Kaikki nämä taidot yhdistyvät kollaboratiivisessa ongelmanratkaisussa, jota voidaan pitää kollaboratiivisen oppimisen yhtenä alamuotona (Luckin ym. 2017). Kollaboratiivinen oppiminen on hyvin laaja käsite ja voi merkitä lähes mitä tahansa yhteisestä ongelmanratkaisusta elinikäiseen vuorovaikutukseen ympäristön kanssa (Dillenbourg 1999).

Kollaboratiivisen oppimisen periaate perustuu Vygotskin sosiokulttuuriselle käsitykseen oppimisesta. Vygotskin teorian mukaan oppiminen ei tapahdu ainoastaan yksilön mielen sisällä vaan ensisijaisesti suhteessa ympäröivään kulttuuriin. Yksilö käyttää yhteisössä kehittyneitä kulttuurisia artefakteja – kuten kieltä, teknologiaa ja tieteellisiä konsepteja – oman oppimisensa välineinä (Crain 2005). Tällä tavalla oppiminen on sosiaalinen tapahtuma, ja yksilön kognitiiviset taidot kehittyvät vuorovaikutuksessa ympäristön kanssa. Vasta tämän vuorovaikutuksen jälkeen oppiminen siirtyy mielen sisäiseksi tapahtumaksi (Lee & Smagorinski 2000).

Kollaboratiiviselle ongelmanratkaisulle on paljon määritelmiä, mutta yksinkertaisimmillaan se voidaan määritellä dyadin tai pienen ryhmän pyrkimykseksi nykytilasta kohti haluttua päämäärää (Hesse, Care, Buder, Sassenberg & Griffin 2015, 39; ks. myös Littleton & Häkkinen 1999). PISA:n viitekehyksen mukaan kollaboratiivisessa ongelmanratkaisussa kaksi tai useampi toimija pyrkii ratkaisemaan ongelman jakaen keskenään sekä ymmärrystä että vaivannäköä (OECD 2015, 6). Damon ja Phelps (1989) painottavat lisäksi dyadin tai ryhmän sisäistä tasa-arvoa kollaboratiivisessa ongelmanratkaisussa. Kollaboratiivinen ongelmanratkaisu yhdistää kriittisen ajattelun, ongelmanratkaisun, päätöksen teon ja yhteistyön taitoja (Care & Griffin 2014), ja vaatii osallistujilta kykyä tunnistaa omia ja toistensa vahvuuksia, ja yhdistää niitä ongelman ratkaisemiseksi (Hesse ym. 2015). Kyse ei kuitenkaan ole vain työtaakan jakamisesta usealle henkilölle, vaan osallistujat rakentavat yhteistä tietoa ja ymmärrystä dialogin kautta

(Littleton & Häkkinen 1999; Roschelle & Teasley 1995). Oppimista tapahtuu osallistujan joutuessa selittämään ajatteluaan ja toimintaansa, jolloin vastaanottaja voi huomata puutteellisuuksia tai esittää vaihtoehtoisia ratkaisuja (Ploetzner, Dillenbourg, Preier & Traum 1999). Osallistujien välinen vuorovaikutus myös mahdollistaa esimerkiksi tiedon sisäistämisen ja kognitiivisen kuorman helpottumisen paremmin kuin yksin työskentely (Dillenbourg 1999). Koska jatkuva vuorovaikutus tapahtuu verbaalisin ja non-verbaalisin signaalein, ongelmanratkaisuprosessi on myös paremmin havaittavissa verrattuna yksilölliseen ongelmanratkaisuun (Hesse ym. 2015).

Hesse ym. (2015) jakavat kollaboratiivisessa ongelmanratkaisussa tarvittavat taidot sosiaalisiin ja kognitiivisiin taitoihin.

3.4.1 Sosiaaliset taidot: kollaboraatio

Kollaboratiivisen ongelmanratkaisun toinen puoli, kollaboraatio, vaatii ryhmän jäseniltä sosiaalisia taitoja. Vuorovaikutus- ja yhteistyötaidot ovat tärkeässä roolissa kollaboraatiossa; osallistujien on tärkeää osata ilmaista omat ajatuksensa toiselle, mutta myös nähdä asiat toisen näkökulmasta, ja arvioida kaikkien osallistujien kognitiivista kykyä osallistua tehtävän ratkaisemiseen (Care & Griffin 2014). Dillenbourg (1999) kuvailee kollaboraatiota sitoutumiseksi kohti yhteistä päämäärää, joka sisältää tiedon rakentamista vuorovaikutuksessa toisten kanssa.

Hessen ynnä muiden (2015) mukaan kollaboratiivisen ongelmanratkaisun ”kollaboraatio” koostuu sosiaalisista taidoista, joita ovat osallistuminen, perspektiivin ottaminen sekä sosiaalinen säätely. Osallistumisella tarkoitetaan sitä, että ryhmän jäsen on valmis jakamaan omaa ymmärrystään ja ajatuksiaan muille ja osallistuu työskentelyyn. Perspektiivin otto puolestaan on tärkeää ryhmän sisäisen vuorovaikutuksen kannalta, koska ilman sitä osallistujat ainoastaan puhuvat omia ajatuksiaan, jotka eivät välttämättä rakenna mitään yhteistä. Kun yksilö kykenee asettumaan muiden asemaan, hän voi ilmaista ajatuksensa muille ymmärrettävällä tavalla. Konfliktitilanteissa ryhmän jäsen osaa myös nähdä tilanteen toisen näkökulmasta ja pyrkii näin saavuttamaan kompromissin. Sosiaalisen

säätelyn taito puolestaan auttaa ryhmän jäseniä tulkitsemaan tilanteita ja ohjaamaan työskentelyä oikeaan suuntaan. (Hesse ym. 2015.)

Kollaboraatio on yksi merkittäviä 2000-luvun taitojen osa-alueita. Kilpailullisuuden sijaan nykykoulussa halutaan korostaa yhdessä tekemistä. Sekä OECD-maiden 2000-luvun taitojen määritelmässä (Care ja Griffin 2014; ks. luku 3.1) että perusopetuksen opetussuunnitelman perusteissa (OPH 2014) nostetaan esiin ryhmässä työskentelyn ja vuorovaikutustaitojen tärkeys.

Vaikka kollaboraatiolla on tutkimusten mukaan positiivinen vaikutus oppilaiden oppimiseen (ks. Luckin ym. 2017) ja motivaatioon (Johnson, Johnson, Roseth & Shin, 2014), kouluissa sitä toteutetaan vielä melko vähän. Tutkimukset osoittavat, että opettajat kokevat kollaboratiivisen opetuksen haasteelliseksi muun muassa ajanpuutteen ja kontrollin menettämisen pelon vuoksi. Toisaalta myös oppilaat voivat kokea kollaboratiivisen oppimisen haastavaksi ja stressaavaksi tilanteeksi, jos heillä ei ole tarvittavia yhteistyötaitoja. (Luckin ym. 2017.)

3.4.2 Kognitiiviset taidot: ongelmanratkaisu

Jos kollaboraatio vaatii osallistujilta sosiaalisia taitoja, ongelmanratkaisu puolestaan vaatii kognitiivisia taitoja. Nämä kognitiiviset taidot liittyvät *tehtävän hallintaan* sekä *oppimisen ja tiedon rakentamiseen*. Tehtävän hallinta vaatii alataitoja, kuten ongelman määrittäminen, suunnitelmallisuus ja välitavoitteiden asettaminen, tiedon analysointi, resurssien hallinta, tiedonhankinta, joustavuus ja systemaattisuus (ks. myös OECD 2010). Nämä ovat taitoja, joita tarvitaan missä tahansa ongelmanratkaisutilanteessa. Oppimisen ja tiedon rakentamisen taidot puolestaan liittyvät ennen kaikkea kollaboratiiviseen ongelmanratkaisuun. Tiedon rakentamisella tarkoitetaan kykyä jakaa ja yhdistellä tietoa ryhmän jäsenten välillä. (Hesse ym. 2015.)

Ongelmanratkaisusta puhuttaessa on lähdettävä liikkeelle ongelman määritelmästä. Haapasalo (2011) määrittelee ongelman seuraavasti:

> Ongelma on tilanne, johon liittyy yksilön kannalta ristiriita- ja epätasapainotila, loogis-kognitiivinen konflikti. Tämä aikaansaa päämäärähakuista ajattelutoimintaa tähdäten tämän ristiriidan poistamiseen, ratkaisun löytämiseen.

Ongelmaa seuraa siis yleensä ongelmanratkaisu. Ensimmäisiä ongelmanratkaisun määritelmiä on Polyan (1945) kehittämä malli ongelmanratkaisuprosessista. Hänen mukaansa ongelmanratkaisuun kuuluu neljä vaihetta: ongelman ymmärtäminen, suunnitelman tekeminen, suunnitelman toteuttaminen ja ratkaisun arvioiminen. Tätä ajatusta ovat edelleen kehittäneet Mason, Burton ja Stacey (1982), jotka jakavat ongelmanratkaisuprosessin kolmeen vaiheeseen: saapuminen (*entry*), hyökkäys (*attack*) ja arviointi (*review*). Saapumisvaiheessa ongelmaan tutustutaan tarkasti ja se pyritään sisäistämään. Hyökkäysvaiheessa puolestaan ongelma pyritään ratkaisemaan ja lopulta arviointivaiheessa tarkastetaan ratkaisun oikeellisuus, reflektoidaan oppimista ja laajennetaan ymmärrystä suurempaan kontekstiin. Polyan (1945) malliin verrattuna Mason, Burton ja Stacey (1982) painottavat ongelmanratkaisuprosessin kehämäisyyttä. Esimerkiksi hyökkäysvaiheessa ihminen saattaa usein jäädä jumiin, jolloin on tarpeen aloittaa prosessi alusta ja palata saapumisvaiheeseen.

3.4.3 Pariohjelmointi kollaboratiivisen ongelmanratkaisun muotona

Luvussa 2 käsitelty ohjelmoinnillinen ajattelu ja etenkin siihen sisältyvä debuggaus ovat pohjimmiltaan ongelmanratkaisua, jonka välineenä käytetään tietokonetta. Kun taas kaksi ohjelmoijaa ohjelmoivat yhdessä, eli ratkaisevat ongelmia tietokoneella kollaboraatiossa toistensa kanssa, puhutaan pariohjelmoinnista (Preston 2005). Pariohjelmoinnissa kaksi ohjelmoijaa työskentelevät yhteisen ohjelman parissa rinnakkain (deClue 2003). Heidän roolinsa vaihtelevat ”ajajan” (*driver*) ja ”navigoijan” tai ”observoijan” (*navigator, observer*) tehtävien välillä. Ajaja vastaa koodin kirjoittamisesta, kun taas navigoijan tehtävä on havainnoida ja arvioida koodia jatkuvasti (Williams & Upchurch 2001). Työparin välillä käydään siis jatkuvaa keskustelua, jonka kautta ajaja ja navigoija vaihtavat tietoa ja osaamista, ja reflektoivat samalla omaa toimintaansa (Cockburn & Williams 2000).

Pariohjelmoinnilla on havaittu olevan monia hyviä puolia. Pariohjelmointi muun muassa nopeuttaa ohjelmointia (Kuppuswami & Vivekanandan 2004), lisää ymmärrystä ja viihtyvyyttä (Liedenberg, Mentz & Breed 2012) sekä vähentää

virheitä ja parantaa ohjelmien laatua (Cockburn & Williams 2000;). Kuppuswami ja Vivekanandan (2004) havaitsivat myös ohjelmoinnin oppimisen olevan tehokkaampaa niillä oppilailla, jotka pariohjelmoivat kuin niillä, jotka ohjelmoivat yksin. Eräässä tutkimuksessa (deClue 2003) opiskelijat vertailivat kokemuksiaan itsenäisen ohjelmoinnin ja pariohjelmoinnin välillä ja kokivat parin auttavan heidän ohjelmointiaan. Heidän mielestään oli hyödyllistä, että omia ideoitaan pystyi jakamaan jonkun toisen kanssa ja saamaan niistä myös palautetta. Pariohjelmointi myös lisäsi opiskelijoiden luottamusta ohjelman toimivuuteen ja laatuun.

Toisaalta on havaittu, että eritasoisten ohjelmoijien välinen pariohjelmointi voi aiheuttaa turhautumista ja konflikteja, mikä puolestaan hidastaa työskentelyä ja huonontaa ohjelmointikokemusta (Lewis 2011; Kuppuswami & Vivekanandan 2004; deClue 2003). Myös Denner, Werner, Campe ja Ortiz (2014) havaitsivat tutkimuksessaan, että pariohjelmointitilanteessa, jossa toinen oppilaista oli haluton yhteistyöhön, myös toisen oppilaan oppiminen häiriintyi.

Tietokone voi myös toimia vuorovaikutuksen välineenä ja sitä kautta vaikuttaa kollaboratiiviseen ongelmanratkaisuun. Roschelle ja Teasley (1995) havaitsivat kollaboratiiviseen oppimiseen liittyvässä tutkimuksessaan tietokoneen tukevan oppilaiden vuorovaikutusta ja kollaboraatiota. Oppilaat, jotka eivät hallinneet tehtävään liittyvää termistöä tai eivät muuten osanneet sanallistaa ajatuksiaan pystyivät käyttämään tietokoneen näyttöä hyväksi yrittäessään jakaa parilleen ajatuksiaan. Tutkimuksessa havaittiin myös, että esimerkiksi hiiren liikkeet ja klikkaukset tai näppäimistön käyttö voidaan tulkita sanattomiksi vastauksiksi tai hyväksynnän osoituksiksi parin ehdotuksille.

4 TUTKIMUSKYSYMYKSET

Ohjelmoinnillisen ajattelun kehittymistä on alakoulun puolella tutkittu vielä hyvin vähän. Tutkimuksessani minua kiinnostaa ennen kaikkea *ohjelmoinnilliset käytänteet*, joka on ohjelmoinnillisia konsepteja laajemmin sovellettavissa oleva ohjelmoinnillisen ajattelun osa-alue. Ohjelmoinnillisista käytänteistä huomio kiinnittyy erityisesti debuggausaitoon eli virheiden korjaamiseen pariohjelmoinnin näkökulmasta. Debuggaus on kompleksinen taito, joka vaatii perusymmärrystä ohjelmoinnin logiikasta ja käytetystä ohjelmointikielestä. Ongelmanratkaisua ja debuggausta on tutkittu aiemminkin, ja ongelmanratkaisuprosessista sekä strategioista on useita malleja (esim. Polya 1945, Katz & Anderson 1987; Klahr & Carver 1988; Vessey 1985; Wyeth 2008). Ohjelmointi on kuitenkin alakoulun kontekstissa hyvin uusi aihe ja sitä varten on kehitetty monia uusia oppimisen välineitä ja materiaaleja. Tästä näkökulmasta katsottuna debuggausta on tutkittu hyvin vähän ja uutta tutkimusta kaivataan. Lye ja Koh (2014) toteavat, että juuri ohjelmoinnillisten käytänteiden ja perspektiivien oppiminen sekä oppilaiden ohjelmointiprosessit vaativat lisää tutkimusta. He mainitsevat myös, että ohjelmointiprosessia voi lähestyä Polyan (1945) ongelmanratkaisumallin näkökulmasta, kuten tässä tutkimuksessa on tehty.

Tutkimuksessa tarkastellaan ohjelmointiin vasta tutustuneiden oppilasparien ongelmanratkaisuprosesseja sekä heidän keskinäistä vuorovaikutustaan kouluopetuksen kontekstissa. Oppilaiden työskentelystä etsitään myös erilaisia debuggausstrategioita. Lopuksi tarkastellaan, kuinka oppilaiden käyttämät ongelmanratkaisustrategiat vaikuttivat koko ongelmanratkaisuprosessiin.

1. Miten debuggaus ilmenee oppilaiden keskinäisessä vuorovaikutuksessa?
2. Millaisia strategioita oppilaat käyttivät korjatessaan virheitä?
3. Kuinka erilaisten strategioiden käyttö näkyi oppilaiden ongelmanratkaisuprosesseissa?

5 TUTKIMUKSEN TOTEUTTAMINEN

5.1 Tutkimuskohde ja lähestymistapa

Tutkimustavaltaan tutkimus on laadullinen tapaustutkimus. Laadullinen tutkimus pyrkii kuvaamaan todellista elämää (Hirsjärvi, Remes & Sajavaara 2000). Laadullinen tutkimusote on määrällistä sopivampi, kun halutaan kuvata ilmiöitä ja tapahtumia luonnollisessa tilanteessa (Metsämuuronen 2006).

Tapaustutkimuksen tavoitteena on tavallisesti jonkin uuden ilmiön yksityiskohtainen ja syvällinen tarkasteleminen yksittäisen tapauksen kautta (Saarela-Kinnunen & Eskola 2010). Epistemologisesta näkökulmasta katsoen kyse ei ole tutkimuksen tulosten yleistettävyydestä, vaan ilmiön syvällisestä ymmärtämisestä (Metsämuuronen 2006, 91). Tapauksen määritelmä on hyvin laaja; tapaus voi olla esimerkiksi yksilö, ryhmä, kylä, organisaatio tai koulu (Metsämuuronen 2006, 90; Saarela-Kinnunen & Eskola 2010). Tapaustutkimus pyrkii toisaalta kuvailemaan, mutta myös selittämään tutkittavaa ilmiötä (Syrjälä 1994, 11).

Tapaustutkimuksessa on tärkeää erottaa toisistaan itse tapaus ja tutkimuskohde (Laine, Bamberg & Jokinen 2007, 11). Tämän tutkimuksen tapauksena on jyväskyläläisen koulun kolme 4.-luokkaa, joissa tutkijat kävivät noin kymmenen viikon ajan opettamassa ohjelmointia ja keräämässä aineistoa tutkimusta varten. Tutkimuskohteena puolestaan on ohjelmoinnillisen ajattelun opettaminen ja oppiminen alakoulussa. Tutkimusta voi luonnehtia tapaukseksi, koska tutkittavissa luokissa ei oltu aikaisemmin opetettu ohjelmointia, eikä opettajilla ollut ohjelmoinnin opettamisesta aikaisempaa kokemusta.

5.2 Osallistujat

Tutkimukseen osallistui 14 4.-luokan oppilasta, jotka olivat samassa koulussa kolmella eri luokalla. Tutkimuksen ajan nämä oppilaat työskentelivät enimmäkseen pareittain tunneilla, joiden aiheena oli ohjelman virheiden korjaaminen

eli debuggaus. Kukaan osallistujista ei ollut aiemmin ohjelmoinut. Oppilaat saivat eteensä valmiita ohjelmia, joihin oli piilotettu virheitä. Ohjelman ohjeissa kerrottiin, kuinka ohjelman olisi tarkoitus toimia ja kuinka se sen sijaan toimi. Oppilaiden tehtävä oli löytää ohjelmien käskysarjoihin piilotetut virheet ja korjata ne niin, että ohjelma toimisi halutulla tavalla.

Aineisto kerättiin kahden eri session – ”Debuggaus 1” ja ”Debuggaus 2” – aikana. Joiltain oppilaspareista aineistoa kerättiin molemmilla sessioilla, jotkut pareista taas olivat mukana vain toisella sessiolla. Tarkoituksena ei kuitenkaan ole tehdä vertailevaa interventiotutkimusta, sillä sessioiden tehtävät ja tutkimusasetelmat eivät ole täysin verrattavissa. Sessioiden välissä oppilaiden kanssa käsiteltiin kuitenkin erilaisia ohjelmoinnin rakenteita kolmen viikon ajan. Siksi onkin hyvä ottaa huomioon, että oppilaiden ohjelmoinnilliset käytänteet ovat mahdollisesti kehittyneet debuggaus-sessioiden välissä, mikä saattaa vaikuttaa myös heidän ongelmanratkaisuprosesseihinsa.

Tutkittavien ja tutkijoiden välillä tapahtui paljon vuorovaikutusta johtuen tutkijoiden roolista opetuksen pääasiallisina järjestäjinä. Vaikka tämä aiheutti joitain haasteita tutkimuksen teolle, vuorovaikutus on olennainen piirre tapaustutkimukselle (Syrjälä 1994, 14) ja lisäsi myös luottamusta tutkijoiden ja tutkittavien välillä. Varmasti osittain tästä johtuen oppilaat olivat hyvin yhteistyöhaluisia, eivätkä ensimmäisten tuntien jälkeen ujostelleet tutkijoita tai tutkimusvälineitä.

5.3 Aineiston keruu

Aineisto kerättiin tammi-huhtikuussa 2017. Oppitunnit kuva- ja äänitallennettiin kokonaisuudessaan. Oppilaiden valmiit tehtävät tallentuivat Scratch-ympäristön studioihin myöhempää tarkastelua varten. Resnick ja Brennan (2011) toteavat, että valmis Scratch-projekti kertoo oppilaan ohjelmoinnillisesta ajattelusta jo paljon, ja toimii sen arvioinnin välineenä. Pelkän projektin tarkastelu ei kuitenkaan kerro siitä prosessista, jolla oppilas on päätynyt lopputulokseen. Niinpä Resnickin ja Brennanin (2011) mukaan arvioinnissa on tärkeää kuulla myös op-

pilasta, ja näin saada ymmärrystä hänen ajatusprosessistaan. Tämän vuoksi oppilaiden työskentelystä tallennettiin myös näyttökaappaukset, joilta näkyy yksityiskohtaisemmin, kuinka oppilaat ovat ratkaisseet tehtäviä. Lye ja Koh (2014) ehdottavat, että ohjelmoinnin prosessin tutkimisessa olisi hyödyllistä käyttää think-aloud –menetelmää, jossa tutkittavia kannustetaan puhumaan ajatuksiaan ja toimintaansa ääneen. Totesin menetelmän kuitenkin olevan haastava näin nuorille oppilaille, ja epäilin tutkijan jatkuvan läsnäolon ja kehotusten saattavan vaikuttaa oppilaiden ajatteluprosessiin. Oppilaat kuitenkin työskentelivät pareittain, jolloin oppilaiden keskinäinen vuorovaikutus, ääneen ajattelu sekä esimerkiksi osoitukset paljastavat jo paljon heidän ajattelustaan.

Opetuskokonaisuus oli yhteensä yhdentoista viikon mittainen. Tästä kokonaisuudesta oma tutkimukseni keskittyy kahteen tuntiin (Debuggaus 1 ja Debuggaus 2), ja tiettyihin oppilaspareihin näiden tuntien sisällä. Debuggauksen lisäksi opetuskokonaisuuden aikana harjoiteltiin erilaisia ohjelmoinnillisia rakenteita ja perustaitoja. Opetuskokeilun kulku ja tätä tutkimusta varten tehty aineistonkeruu on havainnollistettu taulukossa 1.

TAULUKKO 1.

Tunti	Oppitunnin aihe	Osallistujat	Kerätty aineisto
1	Scratch-ohjelmointiympäristöön tutustuminen		
2	Skriptin luominen		
3	Ohjelmien suunnittelua omassa luokassa		
4	Ohjemointilohkoihin tutustuminen suunnitelmien pohjalta		
5	Debuggaus 1	Laura ja Julia Reetta ja Iida Anna ja Suvi Tuomas ja Roope	Näyttökaappaukset GoPro-nauhoitukset ja ääninauhat Valmiit tehtävät Itsearviointilomakkeet jokaiselta oppilaalta.
6	Viestin lähettäminen ja vastaanottaminen Rinnakkaisuus		
7	Ehtorakenne		
8	Debuggaus 2	Saku ja Jesse Maija ja Neea Silja ja Salli	Näyttökaappaukset GoPro-nauhoitukset ja ääninauhat Valmiit tehtävät Itsearviointilomakkeet jokaiselta oppilaalta.
9	Loppuprojektin suunnittelua		
10	Loppuprojektin toteuttaminen		
11	Loppuprojektin toteuttaminen		

Kahdella debuggaukseen keskittyneellä tunnilla oppilaiden tehtävänä oli pareittain ratkoa valmiita projekteja, joihin oli piilotettu virheitä eli ”bugeja”. Koska ohjelmat eivät olleet oppilaiden itsensä ohjelmoimia, oli tärkeää tehtävän aluksi kertoa oppilaille, minkälaisen ongelman virhe ohjelmassa aiheutti. Siksi

jokaisen projektin ohjeissa oli kerrottu, mitä ohjelmalta haluttiin ja mitä ohjelmassa sen sijaan tapahtui (Klahr & Carver 1988). Tällä tavalla debuggausprosessin ensimmäinen vaihe (virheen löytäminen, ks. alaluku 2.2.1) oli jo tehty lasten puolesta, mikä auttoi heitä pääsemään prosessin alkuun. Ongelman määrittäminen lapsille valmiiksi myös auttoi heitä ymmärtämään ohjelman kokonaiskuvaa, minkä on todettu olevan aloitteleville debuggaajille vaikeaa (Vessey 1985).

Havainnoitavien oppilasparien työskentelyä ja puhetta nauhoitettiin sekä ääninauhureilla että GoPro-kameroilla. Kuten taulukosta näkee, osa oppilaspareista oli samoja sekä Debuggaus 1 – että Debuggaus 2 –tunneilla, kun taas osaa pareista havainnoitiin vain toisella tunnilla. Debuggaus 1 –tunnilla havainnoitiin neljää ja Debuggaus 2 –tunnilla kolme oppilasparia. Pareilta nauhoitettiin enimmäkseen heidän keskinäistä luonnollista puhettaan, mutta pyrin myös kysymään heiltä tarkentavia kysymyksiä ongelmanratkaisuprosessiin liittyen. Havainnoitavien oppilasparien tietokoneilta kerättiin myös näyttökaappaukset, joissa näkyy kaikki tunnin aikana näytöllä tapahtunut toiminta. Videomateriaali kerättiin kahdella kameralla, jotka kuvasivat jokaisella tunnilla kahden eri oppilasparin työskentelyä. Oppilaiden puheen, heidän eleittensä ja osoitustensa sekä näytöllä tapahtuvan toiminnan perusteella pyrittiin luomaan käsitystä siitä, millaisia ongelmanratkaisustrategioita he käyttivät ratkoessaan ongelmia.

5.4 Aineiston analyysi

Aineisto analysoitiin temaattisen analyysin sekä diskurssianalyysin keinoin. Kahteen ensimmäiseen tutkimyskysymykseen vastatakseni analysoin aineistoa temaattisesti aikaisemman tutkimuksen valossa ongelmanratkaisu- ja debuggaustaitojen löytämiseksi. Koska oppilaat työskentelivät pareittain, myös kollaboratiivisella ongelmanratkaisulla oli suuri merkitys itse debuggaukseen. Kolmas tutkimuskysymys keskittyykin juuri tähän vuorovaikutukselliseen puoleen, jonka tarkastelu diskurssianalyysin keinoin on temaattista analyysia tarkoituksenmukaisempaa.

Analyysi aloitettiin teorialähtöisesti taustakirjallisuudesta saatujen teemojen mukaisesti. Tilaa jätettiin kuitenkin myös aineistolähtöisyydelle sitä mukaan, kun aineistosta paljastui tutkimuskysymysten kannalta mielenkiintoisia teemoja. Braunin ja Clarken mukaan teemaa ei määritä se, kuinka usein se ilmenee aineistossa, vaan ennen kaikkea se, kertooko se jotain tutkimuskysymyksen kannalta olennaista (Braun & Clarke 2006).

Oppilaiden keskinäinen puhe, sekä vuorovaikutus opettajien ja tutkijoiden kanssa litteroitiin yksinkertaisesti tutkimuksen tavoitteiden mukaisesti. Vaikka analyysissa käytettiin myös diskurssianalyysia, tutkimuksen keskiössä ei ole kielenkäyttö, eikä tarkoituksena ole tehdä tarkkaa keskustelunanalyysia, joten litteroinnin ei ollut tarpeen olla kovin yksityiskohtaista. Oppilaat myös osoittivat usein sormellaan näyttöä, sekä kuljettivat hiirtä näytöllä ajattelunsa tukena tai siirtelivät lohkoja ja kokeilivat erilaisia ratkaisuja. Kaikki tällainen merkittävältä vaikuttava toiminta merkittiin myös litteraatteihin, samoin kuin näyttökaappauksista havaitut yksityiskohdat.

Sekä temaattisessa että diskurssianalyysissa, kuten lähes kaikessa laadullisessa tutkimuksessa analyysi alkaa siitä, että tutkija syventyy aineistoonsa lukien sitä uudestaan ja uudestaan, mieluiten mahdollisten alkuperäisten video- ja ääninauhoitusten kanssa. Tämän jälkeen aineisto koodataan ja tällä tavalla organisoidaan uudelleen, jotta aineistosta voidaan löytää merkityksiä ja teemoja. (Tuomi & Sarajärvi 2018; Braun & Clarke 2006.)

5.4.1 Temaattinen analyysi

Temaattinen analyysi on metodi, joka mahdollistaa teemojen ja kaavojen tunnistamisen, analysoimisen ja raportoinnin aineistosta (Braun & Clarke 2006). Se on joustava analyysimetodi, joka kuitenkin tuottaa hyvin yksityiskohtaista tietoa (Braun & Clarke 2006). Tämän tutkimuksen temaattinen analyysi perustui aikaisemmassa tutkimuksessa havaituille teemoille. Lye ja Koh (2014) kehottavat lasten ohjelmoinnillisen ajattelun, ja etenkin debuggauksen tutkimuksessa luomaan valmiita kategorioita aikaisemman kirjallisuuden pohjalta. Heidän mukaansa erityisesti Polyan (1945) ongelmanratkaisun malli voi olla hyödyllinen

näkökulma debuggauksen tutkimisessa. Niinpä Polyan (1945) ongelmanratkaisun teoria, mutta myös Katzin ja Andersonin (1987) näkemys ongelmanratkaisun kehästä ohjasivat tämän tutkimuksen analyysia. Taustakirjallisuuden perusteella aineisto koodattiin alustavasti ATLAS.ti-ohjelmalla seuraavin koodein: ohjelmaan tutustuminen, suunnittelu, virheen löytäminen, virheen korjaaminen, testaaminen ja ratkaisun arviointi. Myös parien käyttämät strategiat koodattiin aineistosta perustuen aiemmassa tutkimuksessa (Klahr & Carver 1988; Vessey 1985; Wyeth 2008) havaittuihin debuggausstrategioihin.

Temaattinen analyysi eteni Tuomen ja Sarajärven (2018, 104–107) kuvaaman laadullisen tutkimuksen analyysin rungon mukaan. Ensimmäiseksi päätin taustakirjallisuuden perusteella, mikä minua aineistossa kiinnostaa. Kuten edellä kuvattu, loin koodeja teorialähtöisesti ja näitä koodeja käyttäen kävin aineiston läpi. Litteroin videoaineiston ja merkitsin litteroituun aineistoon oppilaiden yksittäisiä lausumia ja signaaleja sekä pidempiä episodeja, joista kävi selvästi ilmi ongelmanratkaisun eri vaiheita. Vertailin oppilaspareja keskenään ja havaitessani oppilasparien ongelmanratkaisun eroavan toisistaan teemoittelin aineiston sen mukaan, millaisia ongelmanratkaisun strategioita kukin oppilaspari käytti. Teemoja olivat *tarkka korjaaminen*, *kokeilu ja erehdys* sekä *välttely*.

5.4.2 Diskurssianalyysi

Oppilasparien sisäinen vuorovaikutus, paridynamiikka ja neuvottelu otettiin huomioon kollaboratiivisen ongelmanratkaisun näkökulmasta, ja sen tarkastelussa käytettiin diskurssianalyysin keinoja. Gee (2011, 128) kuvailee diskurssianalyysia kehäksi, jossa liikutaan edestakaisin kielen rakenteiden ja niiden sisältämien merkitysten välillä. Kieltä käytetään hyvin erilaisiin tarkoituksiin (Gee 2011), ja tässä tutkimuksessa olennaista oli tarkastella, kuinka kielellä rakennetaan käytänteitä – tässä tapauksessa parin yhteistä ongelmanratkaisua – ja toisaalta, kuinka oppilaat käyttävät kieltä rakentaakseen keskinäistä suhdettaan.

Parit olivat vuorovaikutuksessa sekä verbaalisin että non-verbaalisin viestein. Tämän vuoksi pelkkä sanallisten diskurssien tarkastelu olisi antanut parin

vuorovaikutuksesta hyvin vajaan kuvan. Tärkeä osa aineistoa olivatkin maininnat oppilaiden sanattomasta viestinnästä, kuten osoituksista tai muista eleistä. Vaikka diskurssianalyysissä mielenkiinnon kohteena on kieli, myös sanattomat viestit otettaan analyysissä huomioon sanallisen viestinnän täydentäjänä. Analyysissä lähdettiin siitä, että kaikella viestinnällä oppilaat pyrkivät aina johonkin. Aineistosta etsittiin merkittäviä kohtia, joissa oppilaat pyrkivät joko edistämään ongelmanratkaisua tai vaikuttamaan keskinäiseen suhteeseensa. Tällaisia kohtia merkittiin koodeilla, kuten ”neuvottelu” tai ”konflikti”.

Parien dynamiikkaa määritteli parien työnjako ja diskurssianalyysillä pyrittiin myös selvittämään, millaisia rooleja pareilla oli. Pariohjelmoinnissa on yleensä selkeät roolit: ajaja ja navigoija (ks. luku 3.2.3), joten kohdat joissa oli havaittavissa ajaja–navigoija-jakoa merkittiin koodilla ”ajaja–navigoija”. Usein tällaisissa kohdissa navigoija antoi ajajalle käskyjä joko suullisesti tai esimerkiksi osoittamalla näyttöä, ja ajaja puolestaan käytti hiirtä ja näppäimistöä ohjeiden mukaisesti (ks. esimerkki 1)

> Esimerkki 1.
> Jesse: Vastaanota seis. Laitetaan se kaikille. Laita se tohon (osoittaa näyttöä), ”pysäytä”.
> Saku: Näinkö?
> (Saku kiinnittää ”kun vastaanotan viestin” –lohkon ”pysähdy”-lohkoon)

Vertailin oppilaiden vuorovaikutusta ja ongelmanratkaisua ja pyrin löytämään yhteneväisyyksiä.

5.5 Eettiset ratkaisut

Lapsia tutkittaessa on tutkimuksen eettisiin ratkaisuihin kiinnitettävä erityistä huomiota. Aarnos (2001) nostaa esille erityisesti tutkimuksen lapsiystävällisyydestä huolehtimisen. Tutkimukseen osallistumisen tulee olla lapselle miellyttävä kokemus, ja tutkijan tulee huolehtia tästä antamalla lapsen rauhassa tutustua tutkijaan. On myös tärkeää tehdä yhteistyötä luokanopettajan kanssa, jotta varmis-

tetaan, että tutkimukseen osallistuu sellaisia lapsia, joille tutkimustilanne olisi aidosti miellyttävä ja turvallinen. Esimerkiksi kovin arkaa lasta ei kannata valita haastateltavaksi. (Aarnos, 2001, 144–145.)

Omassa tutkimuksessani aineistonkeruu tapahtui melko pitkällä aikavälillä ja useamman tutkijan läsnä ollessa. Olin itse opetusjaksolla mukana alusta asti, ja ohjasin oppilaiden työskentelyä, jolloin oppilaat tottuivat läsnäolooni luokassa. Tutkijat myös esittelivät itsensä, tutkimusvälineistön sekä tutkimustensa tarkoituksen oppilaille ennen varsinaisten oppituntien alkamista.

Kameran ja ääninauhurin äärellä työskentely saattaa olla joillekin oppilaille vaikeaa, jos heitä jännittää ajatus heidän oman ajatusprosessinsa ja työnsä tallentumisesta jonkun aikuisen tarkasteltavaksi. Kuten Aarnos (2001, 145) toteaa, aineistonkeruu on sovitettava ”lapsen ajattelun kehitysvaiheeseen ja hänen itseilmaisunsa tyyliin ja määrään”. Lapsilta kysyttiin aina erikseen, saako heidän työpisteensä viereen asentaa GoPro-kameran tai jättää nauhurin. Kun oppilaille selitettiin kameran kuvaavan vain näyttöä ja heidän käsiään, he pystyivät olemaan täysin luonnollisesti myös kameran ollessa päällä. Itse asiassa kameran läsnäolo kiinnosti enemmän vieressä istujia, kuin itse kuvattavia.

Tutkimukseen osallistumiseen pyydettiin lupa oppilaiden huoltajilta, ja tutkimukseen osallistuivat vain ne oppilaat, joiden huoltajat antoivat siihen suostumuksensa. Aineisto oli kahden tutkimusta toteuttaneen tutkijan käytössä. Tätä tutkimusta varten minulla oli kopiot alkuperäisistä video- ja äänitiedostoista, sekä näyttökaappaukset kahdelta eri oppitunnilta (ks. taulukko 1). Säilytin nämä kopiot aineistosta itselläni tutkimuksen valmistumiseen saakka, jonka jälkeen huolehdin niiden hävittämisestä.

Pseudonyymien käyttö on yleisin kvalitatiivisen tutkimusaineiston anonymisoinnin keino (Kuula 2006, 215), ja myös tässä tutkimuksessa osallistujille annettiin peitenimet heidän henkilöllisyytensä suojaamiseksi. Tutkimuksessa jossa osallistujia on useita ja heihin viitataan toistuvasti, peitenimien käyttö toisaalta myös ”säilyttää aineiston sisäisen koherenssin” (Kuula 2006, 215).

6 TULOKSET

6.1 Oppilaiden keskinäinen vuorovaikutus

Suurimmalla osaa oppilaspareista kollaboratiivisen ongelmanratkaisun määritelmä täyttyi; parit sitoutuivat yhteisen päämäärän tavoittelemiseen ja rakensivat yhteistä ymmärrystä (ks. Dillenbourg 1999) sekä verbaalein että non-verbaalein viestein. Toisilla pareilla yhteistä tietoa rakennettiin aktiivisemmin kuin toisissa, ja erään parin kohdalla kollaboraatio estyi parin sisäisen konfliktin vuoksi kokonaan.

Oppilaille ei nimetty tiettyjä rooleja, vaan oppilaat saivat itse päättää työnjaostaan. Pariohjelmoinnissa tavallinen työnjako on ajaja–navigoija. Myös suurin osa oppilaista jakoi tehtävät näin, koska oli helpompaa, että vain toinen käyttää näppäimistöä ja hiirtä. Ajaja–navigoija –työnjakoon kuitenkin kuuluu se, että myös navigoija osallistuu aktiivisesti työskentelyyn kommentoimalla ja arvioimalla ohjelmaa (Cockburn & Williams 2000; Williams & Upchurch 2001), mikä ei kaikkien parien kohdalla toteutunut.

6.1.1 Oppilaspari 1: Saku ja Jesse

Tunnin alussa Saku käytti näppäimistöä ja Jesse hiirtä, mutta toisen tehtävän kohdalla Saku otti itselleen myös hiiren ja samalla ajajan roolin. Jesse puolestaan oli navigoija ja teki ehdotuksia ja kommentoi. Hän kommunikoi Sakun kanssa myös non-verbaalisti osoittelemalla asioita näytöltä. Pariohjelmoinnin näkökulmasta oli mielenkiintoista, että Sakun ja Jessen ajaja–navigoija –työnjako tapahtui niin luonnollisesti ja onnistuneesti, vaikka oppilaita ei erikseen ohjattu mihinkään tiettyyn työnjakoon.

Esimerkki 2.

Jesse: ”Vastaanota seis”. Laitetaan se kaikille. Laita se tohon (osoittaa näyttöä), ”pysäytä”.

Saku: Näinkö?

(Saku kiinnittää ”kun vastaanotan viestin” –lohkon ”pysähdy”-lohkoon. Oppilaat testaavat ohjelmaa.)

Saku: Seis. Seeeis!
Jesse: Venaa.
(Ohjelma pyörii hetken, kunnes yksi hahmoista pysähtyy.)
Jesse: Noni nyt eka pysähty. Nyt sun pitää laittaa niihin kaikkiin muihin se.

Saku ja Jesse eivät juuri käyneet neuvotteluja tai tehneet suunnitelmia, vaan työskentely oli hyvin nopeaa ja keskittynyttä, ja suuri osa oppilaiden välisestä viestinnästä oli sanatonta. Esimerkiksi Jesse teki navigoijana paljon tärkeitä ehdotuksia, ja Saku hyväksyi ehdotukset yksinkertaisesti toimimalla niiden mukaisesti. Oppilaat eivät selittäneet toisilleen, miksi heidän mielestään pitäisi toimia tietyllä tavalla, vaan jommankumman keksiä jonkin ratkaisuehdotuksen sitä testattiin heti.

6.1.2 Oppilaspari 2: Silja ja Salli

Myös Silja ja Salli noudattivat ajaja–navigoija –työnjakoa, mutta toisin kuin Saku ja Jesse, he vaihtoivat rooleja aina tehtävän vaihtuessa. Tällä tavalla molemmat pääsivät käyttämään hiirtä ja näppäimistöä yhtä paljon. Oppilaat olivat todennäköisesti sopineet roolien vuorottelusta aikaisemmilla tunneilla, sillä työnjako vaikutti olevan molemmille hyvin selkeä.

Silja ja Salli puhuivat ajatuksiaan ääneen ja kävivät paljon yhteistä neuvottelua siitä, kuinka tehtävän kanssa tulisi edetä. Verrattuna Sakuun ja Jesseen he tekivät päätöksensä useammin yhdessä ja työskennellessään he samalla kommentoivat ja arvioivat ohjelmaa sekä selittivät omaa ajatteluaan toisilleen.

Esimerkki 3.
Salli: Kokeillaan nyt.
(Silja käynnistää ohjelman ja painaa välilyöntiä, mutta hahmot liikkuvat edelleen ruudulla.)
Salli: Nyt se menee samasta paikasta samaan paikkaan.
Silja: Niin menee. (yksi hahmoista pysähtyy näytöllä) Eiku toi pysähty!
Salli: Noin.
Silja: Nyt kaikki pitää pysähtyä.
(Silja alkaa muuttaa muiden hahmojen skriptjeä samanlaisiksi, kuin pysähtyneellä hahmolla.)

Silja: Mä pysäytän ne siis vaan näin.
(Oppilaat pääsevät viimeisen hahmon kohdalle, jolla onkin rinnakkainen skripti ”kun vastaanotan viestin” lohkon alla.)
Silja: (henkäisee) Se [”kun vastaanotan viestin”-lohko] pitää laittaa tänne näin! (klikkaa takaisin edelliseen hahmoon)

6.1.3 Oppilaspari 3: Anna ja Suvi

Anna ja Suvi olivat hyvin syventyneitä työskentelyynsä koko tunnin ajan. Heidän työnjakonsa oli selvä: Anna käytti näppäimistöä ja Suvi hiirtä. Ongelmanratkaisussa Anna tuntui ottavan aktiivisemman roolin. Hän pohti ratkaisuja ääneen ja osoitti Suville näytöltä, mistä halusi tämän klikkaavan. Toisaalta myös Suvi toi omia ajatuksiaan aktiivisesti esiin. Yhdessä vaiheessa Suvi jopa kurkotti käyttämään näppäimistöä korjatakseen Annaa. Kokonaisuudessaan tyttöjen työskentely oli melko hyvässä tasapainossa; molemmat tekivät ehdotuksia ratkaisujen suhteen ja arvioivat ohjelmaa ääneen rakentaen tällä tavalla yhteistä ymmärrystä.

Esimerkki 4.
Anna: Okei, sille ei tarvii mitään. Sitte, paina tohon (Anna osoittaa kettu-hahmoa). Sitte, paina – ei.
(Suvi raahaa ketun hiirellä takaisin alkuperäiselle paikalleen)
Anna: Noni, paina tosta. (Anna osoittaa vihreää lippua) Eei, ku tosta. Ja sitte. (Anna liu’uttaa sormeaan näytöllä mietteliäänä) Oota hetki. Hä? Pitääks meidän saada tää niinku miten?
Tutkija: No sillee et se sitte kun sitä painaa sitä lippua uudestaan, et sit se toimis samalla tavalla. Kokeilitteks te painaa vielä uudestaan sitä lippua?
Anna: Joo.
Suvi: Sillee, mistä me tiedetään, että missä se oli? (liikuttaa hiirtä editorin ylänurkassa, jossa näkyy ketun koordinaatit) Haa, nyt mä tajusin.
Anna: Onks se tää ”toista”?
Tutkija (Suville): Mitäs sulle tuli nyt mieleen?
Suvi: Että, niinku sitte kun se sanoo ”Oikealla puolella on myös kivaa” nii siihen loppuun, nii sitte se, että-
Anna: Vaikka että ”piilota” tai jotain.

Suvi: Ei, vaan silleen että ”liu’u takaisin” (Suvi liikuttaa hiirtä editorin ylänurkassa, missä ketun koordinaatit näkyvät)

6.1.4 Oppilaspari 4: Maija ja Neea

Maija otti heti alussa ajajan roolin, ja Neea navigaattorin. Maijan ja Neean yhteistyössä mielenkiintoista oli se, että vaikka yleensä ajaja ottaa helposti myös johtajan roolin, tässä tapauksessa Neea oli navigaattorina selvästi kahdesta oppilaasta aktiivisempi. Maija yleensä odotti Neean ohjetta ennen kuin teki hiirellä tai näppäimistöllä mitään, ja haki ongelmatilanteissa Neean tukea katseilla ja eleillä.

Esimerkki 5.
Neea: Laita se ruskea kissa.
(Maija avaa ruskean kissan skriptin)
Neea: Täytyy vaa laittaa siihen…
(Maija siirtyy valkoisen kissan skriptiin)
Neea: ”Vastaanota viesti”. Siihen alapuolelle.
(Maija lisää kaikkien kissojen skriptiin ”vastaanota viesti”-lohkon)

Maijan ja Neean litteraatissa koodit ”konflikti” ja ”hyvä suhde” kulkivat usein rinnakkain, mikä kertoo ehkä siitä, että vaikka ulkopuolisen silmin tilanne saattoi vaikuttaa konfliktin alulta, oppilaille kyseessä oli lähinnä ystävällinen kinastelu. Huumorin käyttö tehtävien aikana oli oppilasparille tärkeää ja lähensi heitä toisiinsa ja oli selvästi keino luoda hyvää yhteishenkeä ja selvitä hankaluuksista.

6.1.5 Oppilaspari 5: Laura ja Julia

Lauran ja Julian työskentely oli melko passiivista, ja he tarvitsivat paljon aikuisen tukea edetäkseen tehtävissä eteenpäin. He pitivät pitkiä taukoja, eikä kumpikaan ei ottanut selkeää johtajan roolia, vaan molemmat tuntuivat odottavan aloitetta toiselta. Yleensä Laura teki lopulta jonkin ehdotuksen. Kommenteista kävi ilmi, että ohjelmoinnilliset rakenteet olivat oppilaille melko vieraita. He viittasivat ohjelmoinnillisiin rakenteisiin usein sanoilla ”tämmönen” tai ”juttu”. Tämä saattaa

kertoa siitä, että he eivät olleet vielä ehtineet tutustua Scratchiin kunnolla, mikä ehkä nosti kynnystä ryhtyä ratkaisemaan tehtäviä.

Esimerkki 6.
Laura: Tohon pitää laittaa se... toista kaks kertaa.
(0:19)
Julia: Mut sehän voi olla kymmenen kertaa. Kun ne katotaan.
Laura: Mut ei siin, ei se toimi niin.
Julia: Aa… Mut eihän tää auta mitään. Ömm… Katsopa. Nyt jos katotaan tää.
(0:08)
Julia: Tää, tänhän ei pitäis olla tässä
Laura: Miten nii?
Julia: Ei täs tartte ... Hei eiks tän vikatyypinki pitäis?
(0:40)
Julia: Mitä me tehään? … Sä saat sanoo mulle.
Laura: Varmaan siihen pitää laittaa…
Julia: Mihin kohtaan?
Laura: Siihe samaan, mut sit siihen pitää laittaa joku toinen juttu.
Julia: Mikä numero? Pitääks sit niinku…
Laura: Niinku.
Julia: Joo.

6.1.6 Oppilaspari 6: Reetta ja Iida

Reetta ja Iida eivät olleet aivan yhtä keskittyneitä työhönsä, kuin esimerkiksi oppilasparit 1 ja 2. Heillä oli hieman vaikeuksia ongelmanratkaisussa, ja varsinkin Reetta ilmaisi useasti turhautumisensa huonosti toimivaan hiireen. Turhautumisensa seurauksena oppilaat menettivät mielenkiintonsa ja suuntasivat keskittymisensä pois tehtävästä esimerkiksi keskinäiseen kinasteluunsa ja hassutteluunsa.

Ensimmäisen tehtävän ratkaiseminen onnistui oppilailta melko hyvin. Heillä ei ollut selkeää työnjakoa, vaan molemmat käyttivät hiirtä ja näppäimistöä sen mukaan, kummalla sattui olemaan jokin idea ongelman ratkaisemiseksi. Loput tehtävät olivat kuitenkin Reetalle ja Iidalle selvästi vaikeita, minkä voi pää-

tellä esimerkin 7 viimeisestä lauseesta: ”Me ei osata!”. Oppilaat tarvitsivat ongelmanratkaisussaan paljon aikuisen tukea pysyäkseen keskittyneinä tehtävään, mikä käy myös selväksi esimerkistä 7.

Esimerkki 7.
Tutkija: Mä annan sellasen vinkin että… Älä, älä jooko koske siihen kameraan.
Reetta (kameralle): Huhuu!
Tutkija: Hei. Ethän koske siihen kameraan Reetta.
Reetta: En mä koske siihen (Reetta kurkistelee kameraan)
Tutkija: Reetta?
Reetta: No?
Tutkija: Älä koske kameraan. Mä annan sellasen vinkin, että ne numerot jotka täällä ekaks oli, nii niillä numeroilla se menee ihan kokonaisen ympyrän. Muistatteko vielä mitä siellä oli?
Reetta: Yheksänkyt, yheksänkyt, yheksänkyt.
Tutkija: Joo.
Iida: Nyt otetaan toi pois.
Reetta: Eiku otetaan… noi pois. Pistetään kaks näitä ja koitetaan sitten. Melkein. Ei se mee. Siinä pitää olla varmaan yks vielä.
(Reetta painaa välilyönnin pohjaan, apina heiluu ruudulla. Oppilaiden kiljumista.)
Iida: Hei kattokaa, Saku ja Jesse!
(Oppilaiden naurua)
Tutkija: Saisko sen silleen, ettei sitä tarvii painaa koko ajan sitä välilyöntiä?
(Reetta painelee välilyöntiä)
T: Okei, no nyt se –
Iida: Me ei osata!

6.1.7 Oppilaspari 7: Tuomas ja Roope

Tuomaksen ja Roopen parityöskentelyssä Roope otti ajajan roolin ja Tuomas navigoijan. Ensimmäisen tehtävän ajan heidän yhteistyönsä sujui aikuisen tukemana melko hyvin; Tuomas teki aktiivisesti ehdotuksia ja osoitteli asioita näytöltä Roopen käyttäessä näppäimistöä ja hiirtä. Roope vaikutti kuitenkin alusta asti turhautuneelta pariinsa ja joko ohitti tämän kommentit tai reagoi niihin vähättelevästi. Luokanopettaja kertoi myös, että Roopen yhteistyötaidot olivat yleisesti ottaen heikot. Parilla oli ollut hankaluuksia jo edellisillä tunneilla, ja heidän

välisensä konflikti jatkui edelleen. Roope pyrki tahallaan provosoimaan Tuomasta, joka hermostui nopeasti ja vetäytyi työskentelystä kokonaan. Oppilaat erosivat selvästi taitotasoltaan toisistaan, mikä saattoi turhauttaa Roopea, joka oli parin taitavampi osapuoli. Tuomas puolestaan koki olevansa tarpeeton. Tämä kävi selvästi ilmi siitä, että hän ei koskaan halunnut nimeään korjatun ohjelman tietoihin, koska ei mielestään ollut osallistunut ohjelman korjaamiseen.

Esimerkki 8.
Roope: Alright, what.
(Tuomas tarttuu hiireen ja klikkaa astelukua yhdessä lohkoista.)
Tuomas: Vaikka näihin kahteen [epäselvää] ja tohon [epäselvää].
Roope: No hyvä on sitten, mutta sitten sä –
Tuomas: Noihin kolmeen. (Tuomas osoittaa lohkoja)
Roope: Joo, mutta sitten sä saat nimes tähän.
Tuomas: Ei!
Roope: Toi oli sun ehotus.
Tuomas: Ei ollu. Mun nimee ei tuu siihen.
(Roope muuttaa numeroita lohkoissa. Tuomas lähtee taas paikaltaan.)

6.2 Oppilaiden debuggausstrategiat

Oppilaiden virheenkorjausta tarkasteltiin ensisijaisesti teorialähtöisesti. Virheenkorjausprosessista löytyi jonkin verran aikaisempaa tutkimustietoa (Katz & Anderson 1987; Klahr & Carver 1988; Vessey 1985), jonka pohjalta oppilaiden virheenkorjausta tarkasteltiin. Lisäksi aineiston analyysissa sovellettiin Polyan (1945) kehittämää ongelmanratkaisun kehän mallia. Näiden pohjalta virheenkorjausprosessin vaiheiksi määritettiin ohjelman ymmärtäminen, testaaminen, virheen löytäminen ja virheen korjaaminen.

Oppilasparit kategorisoitiin tyyppeihin sen mukaan, kuinka he etenivät virheenkorjauksen kehän näkökulmasta ja millaisia strategioita he useimmiten käyttivät virheen korjauksessa. On kuitenkin tärkeää huomata, että oppilasparien eteneminen virheenkorjauksen kehällä sekä heidän käyttämänsä strategiat

vaihtelivat, eikä kukaan siten edustanut puhtaasti vain yhtä virheenkorjausstrategiaa.

6.2.1 Tarkan korjaamisen strategia

Erityisesti kolme eri oppilasparia käyttivät tehtävissään tarkan korjaamisen strategiaa. Heidän strategiansa oli pysähtyä ensin tutkimaan ohjelmaa ja pohtimaan virheen sijaintia ja mahdollisia ratkaisuvaihtoehtoja. Tavallisesti he tutustuivat ohjelmaan ensin lukemalla ohjeen, sitten testaamalla ohjelmaa ja lopuksi avaamalla editorin ja tutkimalla ohjelman skriptejä. Mietittyään hetken oppilaat yleensä onnistuivat ratkaisemaan tehtävän melko nopeasti. Tarkan korjauksen strategiassa tyypillistä oli rauhallisuus, omien ajatusten kommunikointi parille ja keskittyminen työskentelyyn.

KUVA 1. Debuggaustehtävä ”Piko ja Nano”

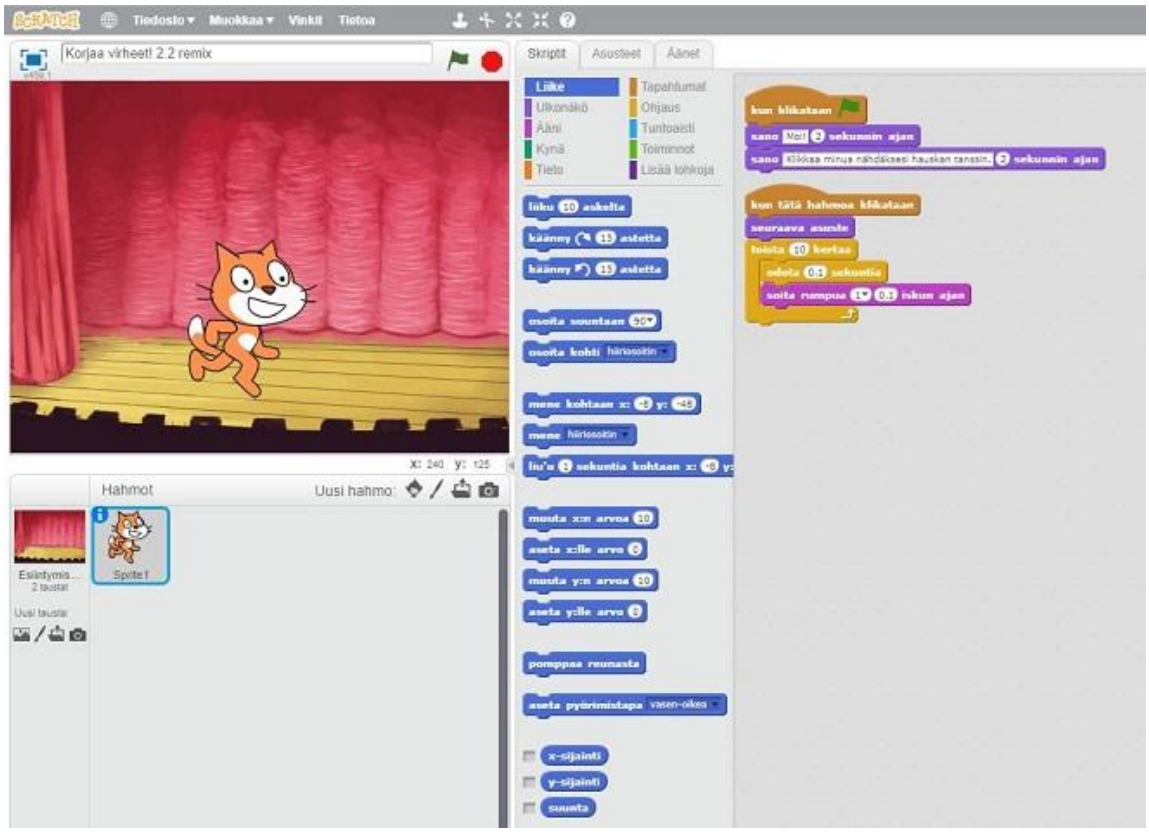
Esimerkissä 9 oppilasparin tehtävänä oli saada toinen ohjelman hahmoista (Piko) liikkumaan toisen hahmon (Nano) luo ja pysähtymään siihen. Ongelmana

oli, että Piko oli ohjelmoitu liikkumaan, kunnes se koskettaa ruudun reunaa. Ratkaistakseen tehtävän oppilaiden tuli muuttaa ehtolauseketta niin, että Piko liikkuu, kunnes koskettaa Nanoa.

Esimerkki 9.
Jesse: Nonii, se on varmaan väärä toi, mihin kohtaan se [Piko] menee.
0:03
Jesse: Eiku siir… siirretään toi toinen.
(Saku raahaa Nanon hahmon lähemmäs reunaa. Oppilaat tutkivat hetken Nanon skriptiä.)
Jesse: Sitte meidän pitää laittaa toi.
(Saku käynnistää ohjelman. Ohjelman käynnistyessä Nano siirtyy kuitenkin takaisin omalle paikalleen. Oppilaat katsovat ohjelman kulkua.)
Saku: Hippa!
Jesse: Se [Nano] ei mee sinne.
Saku: Mitä täs piti tehä?
Jesse: Sitte se niinku…
(Oppilaat palaavat lukemaan ohjelman ohjeita, Jesse lukee ohjetta ääneen.)
Jesse: Mää tiän! Katso sisälle.
Saku: Siihen pitää laittaa sit se, et se [Piko], liikkuu tohon.
(Jesse osoittaa kohtaa ”skriptit”.)
Jesse: Skriptit. Öö, toi (osoittaa Pikon hahmoa). Sitten tuolta (osoittaa lohkoa Pikon skriptissä). ”Koskettaa reunaa”, se!
(Saku klikkaa osoitettua lohkoa, jolloin valikko avautuu. Saku valitsee ”reunan” sijaan ”Nanon”.)
Jesse: Kokeile nyt.
(Saku käynnistää ohjelman, ja Piko pysähtyy Nanon kohdalle.)

Oppilasparin ensimmäinen ratkaisu oli siirtää Nano ruudun reunalle eli samaan kohtaan, mihin Piko oli ohjelmoitu pysähtymään. Testatessaan ohjelmaa he kuitenkin huomasivat, että Nanolle oli määrätty tietty sijainti, ja siksi sitä ei voinut siirtää. Huomatessaan ratkaisunsa huonoksi he palasivat virheenkorjauksen kehällä takaisin alkuun ja yrittivät ymmärtää ohjelmaa paremmin, esimerkiksi kysyen ”Mitä täs piti tehä?”. Tämä tuotti nopeasti uuden ratkaisuvaihtoehdon, joka paljastui toimivaksi.

Myös oppilaspari 2 keskittyi työskentelyssään ohjelman ymmärtämiseen, mikä johti tarkkaan virheenkorjaukseen. Seuraavassa sitaatissa oppilaspari 2 ratkaisee tehtävää, jossa heidän tulisi saada kissa tanssimaan siirtämällä ”seuraava asuste” –niminen lohko toistorakenteen sisään.



KUVA 2. Debuggaustehtävä ”Kissan tanssi”

Esimerkki 10.

Tutkija: Mikä noista lohkoista on se, että… Mikä saa sen kissan tekemään sen yhen askeleen?

Silja (Oppilas 1): Onks se toi, mikä se on? Toi… Onks se ”seuraava asuste” vai ”odota”? Eiku ei se ”odota” oo.

Tutkija: Se on ”seuraava asuste”.

Silja: Nii.

Salli: Voiks siihen laittaa sit siihen kymmenen kertaa?

Silja: Miten se pitää laittaa sitte?

Tutkija: No siinäpä vasta kysymys. (0:04) Mitä sä ajattelit? Mikä sul oli se idea?

Silja: Et siihen ois laittanu sen kymmenen, toista kymmenen kertaa, tai siis siihen niinku (osoittaa näytöltä toistorakennetta) vähemmän sitä… niinku vähemmän, et se ois toistanu sen saman liikkeen. (0:01) Ehkä.

Tutkija: Nii että jos laittas vaikka, että toista seitsemän kertaa ni –
Salli: Nii –
Tutkija: – sittenkö?
Salli: – se toistais sen liikkeen.
Tutkija: Nii. Mut mitenkäs se saatais toistamaan se *liike* seitsemän kertaa, tai niin monta kertaa ku tossa nyt on? Kun nyt se vaan soittaa sitä rumpua kymmenen kertaa.
Silja: *Niii.* Ai pitäiskö toi ”seuraava asuste” sit laittaa, et kymmenen kertaa –
Salli: Nii.
Silja: – et toista kymmenen kertaa? Nii, siihen.
(Molemmat oppilaat huitovat sormellaan toistorakennetta kohti.)
Salli: Joo.
Silja: Et siihen, tohon väliin (osoittaa toistorakenteen sisään).

Tutkittuaan hetken skriptejä oppilasparit tavallisesti avasivat keskustelun; jompikumpi pareista teki aloitteen ehdottamalla ratkaisua tai mainitsemalla jonkin havainnon, tässä tapauksessa esimerkiksi ”Voiks siihen laittaa sit siihen kymmenen kertaa?”. Toimiva kollaboraatio oli tässä avainasemassa, oppilaat puhuivat ajatuksiaan ääneen ja pystyivät rakentamaan yhteistä ymmärrystä toistensa kommenttien perusteella. Erityisen mielenkiintoinen huomio on se, että oppilaspari 2 ei koskenut hiireen tai näppäimistöön, ennen kuin he olivat käyneet ajatuksensa suullisesti läpi ja heillä oli suunnitelma siitä, kuinka tulisi toimia. Keskustelun ja suunnittelun jälkeen parit pystyivät usein paikantamaan ja korjaamaan virheen hyvin nopeasti.

Yksi tarkan korjaamisen strategian tunnuspiirteistä on ongelmanratkaisun joustavuus. Oppilaspareilla ei ollut vaikeuksia siirtyä yhdestä ratkaisuehdotuksesta täysin uuteen vaihtoehtoon, jos he huomasivat ensimmäisen toimimattomaksi. Kuten esimerkistä 9 huomataan, oppilaspari hylkäsi ensimmäisen, huonoksi havaitun ratkaisuvaihtoehtonsa: ”Se [Nano] ei mee sinne.”. He palasivat hetkeksi miettimään: ”Mitä täs piti tehä?” ja lähestyivät ongelmaa tämän jälkeen täysin uudesta näkökulmasta: ”Siihen pitää laittaa sit se, et se [Piko], liikkuu tohon.”.

6.2.2 Kokeilun ja erehdyksen strategia

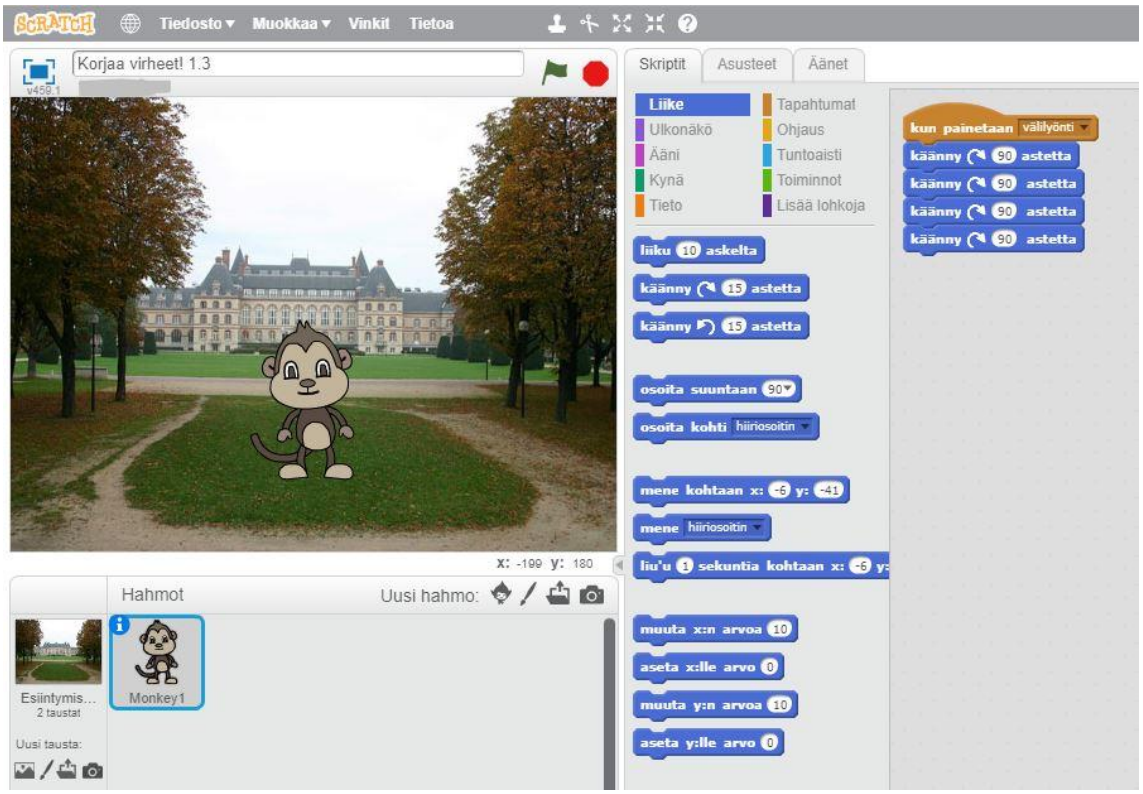
Oppilasparit, jotka ratkaisivat tehtäviä useimmiten kokeilemisen ja erehtymisen strategialla, lähestyivät tehtävää suunnittelemisen sijaan tekemisen kautta ja pyrkivät löytämään oikean ratkaisun kokeilemalla. Esimerkissä 11 Maija ja Neea yrittävät ratkaista ohjelmaa, jossa Pikon hahmon tulisi liikkua Nanon hahmon luo. Heidän ratkaisuehdotuksensa on muuttaa Pikon askelmäärää niin, että se pysähtyisi halutulle kohdalle.

Esimerkki 11.
(Maija muuttaa ”liiku”-lohkon lukemaksi 100.)
Tutkija: Sata askelta on aika –
(Maija testaa ohjelmaa, ja hahmo katoaa ruudun reunan yli.)
Tutkija: – paljon.
Neea: Noku ei se liiku muuten.
Maija: No pannaan sit kuuskytseittemän
(Maija muuttaa lukemaa taas ja testaa ohjelmaa. Hahmo liikkuu edelleen liian pitkälle. Maija muuttaa lukemaa pienemmäksi, jolloin hahmo liikkuu liian vähän.)
Maija: Kakskyt.
(Maija muuttaa lukemaksi 20 ja testaa ohjelmaa, jolloin hahmo siirtyy jo melko lähelle toista hahmoa.)
Tutkija: Sen saa myös silleen, et se… Sen saa myös silleen, et se vaan liikkuu eteenpäin, kunnes se koskettaa tota sen kaveria –
(Maija muuttaa lukemaa, jolloin hahmo siirtyy haluttuun kohtaan näytöllä.)
Tutkija: – nii teidän ei tarvii arvailla.
Neea: Laita vielä kaks askelta. Laita vielä kaks askelta siihen!
Maija: Ookoo. Kakskytseittemän.
(Maija muuttaa lukemaksi 27 ja testaa ohjelmaa, jolloin hahmo liikkuu haluttuun pisteeseen.)

Muokatessaan ”liiku”-lohkon lukemaa oppilaat kohtasivat haasteen yrittäessään saada lukeman juuri oikeaksi. He ratkaisivat ongelman kokeilemalla ja haarukoimalla sopivaa lukua, kunnes saivat lukeman oikeaksi. Esimerkissä 11 oppilasparin ratkaisu ei ollut tehokkain mahdollinen, mutta he kuitenkin yrittivät niin kauan, että virhe tuli korjatuksi. Ratkaisuun saattoi vaikuttaa se, että

”liiku”-lohko oli oppilaille tuttu aikaisemmista ohjelmista, kun taas ehtorakenne oli heille täysin uusi ohjelmoinnillinen konsepti. Aikaisemmin kohdattujen ratkaisujen käyttäminen on yleinen virheenkorjausstrategia (Wyeth 2008). Niinpä oppilasparin oli helpompi tarttua tuttuun ratkaisuun, vaikka tehokkaampikin olisi ollut tarjolla.

Myös oppilasparit 5 ja 6 turvautuivat kokeilemisen ja erehtymisen strategiaan. Esimerkissä 12 Laura ja Julia yrittävät saada apinan heittämään volttia. Apinan skriptissä oli peräkkäin neljä ”käänny 90 astetta” –lohkoa, ja lohkojen väleihin olisi tarvittu ”odota”-lohkot. Sekä oppilaspari 5 että 6 kuitenkin pyrkivät muuttamaan ”käänny”-lohkojen astelukuja.



KUVA 3. Debuggaustehtävä ”Apinan voltti”

Esimerkki 12.

Laura: Pitäis laittaa isommat numerot sinne.

Julia: Laitetaan jokaisiin sata.

Laura: Tai sit laitetaan vähä enemmän, koska se…

Julia: Kaks sataa.

Laura: Kaks sataa.

(0:04)

Julia: Noni koitetaa tätä. Öö.
Julia: Hei tähän voi laittaa sen kolmesataa. No ei. Tästä tää toimii.
(Oppilaat testaavat ohjelmaa, mutta se ei toimi)
Julia: Juu tota, laitetaan se neljäsataa.
Laura: Laitetaan.
Julia: Okei.
(0:13)
Julia: Mitä ihmettä toi tyyppi duunaa?
Laura: Laita tuhat.

Myös Reetta ja Iida pyrkivät ratkaisemaan ”Apinan voltti” –tehtävää kokeilun ja erehtymisen strategialla.

Esimerkki 13.
Tutkija: Mitä te haluaisitte sen tekevän?
Iida: Pyörii tolleen ja sit se menis takasin.
Tutkija: Nii, et se menis oikein päin. Ja sit te ootte muuttanu noita numeroita tuolla ”käänny”-jutussa.
Iida: Mm.
Tutkija: Keksitteks te omasta päästä noi numerot? Et, kokeillaan tommosta?
Iida: Nii.

Suurena erona tarkkoihin korjaajiin on kokeilijoiden ja erehtyjien joustamattomuus ratkaisuehdotuksissaan. Esimerkissä 11 huomataan, kuinka Maija ja Neea pitäytyivät ratkaisussaan silloinkin, kun tutkija yritti ohjata heitä tehokkaamman ratkaisun suuntaan. Tehtävän tekemiseen kului enemmän aikaa, sillä tehottomia strategioita ei pystytty vaihtamaan tehokkaampiin.

6.2.3 Välttelijät

Välttelijät pysäyttivät työskentelynsä tehtävien välissä tai turhautuessaan ja siirsivät keskittymisensä johonkin tehtävän ulkopuoliseen toimintaan. Oppilaspari 7 ei kyennyt yhteistyöhön ollenkaan, minkä vuoksi heidän ongelmanratkaisunsa jäi vähäiseksi. He saivat aikuisen tukemana tehdyksi kaksi tehtävää, mutta lopun ajasta he käyttivät riitelyyn.

Myös oppilaspari 6 ryhtyi tekemään jotain muuta aina tehtävien välissä sekä turhautuessaan niiden aikana. Oppilaspari 6:n kohdalla syy ei ollut yhteistyön haasteissa vaan pikemminkin liian vaikeissa tehtävissä ja siitä seuranneessa turhautumisessa. Erään tehtävän kohdalla oppilaat olivat jo tyytyväisiä ratkaisuunsa, kun heitä pyydettiin vielä hienosäätämään ohjelmaa:

> Esimerkki 14.
> Iida: Nonii, remixistä. Sini, tää on nytte hyvä. Meidän mielest ainakin.
> Tutkija: No, toimiiko? Okei! No nyt se heittää kyllä voltin mutta se heittää sen vähän hitaasti. Miten sen sais nopeutettuu?

Tehtävä oli jo muutenkin ollut oppilaille haastava, ja he olivat käyttäneet sen ratkaisemiseen kokeilemisen ja välttelemisen strategiaa. Kun heitä vielä pyydettiin jatkamaan, he vaihtoivat välttelyn strategiaan: vain joitain sekunteja tutkijan kommentin jälkeen oppilaat kiinnittivät huomionsa pöydällä olevaan ääninauhuriin ja jättivät tehtävänsä kokonaan.

6.2.4 Strategioiden vertailua

Oppilasparit eivät välttämättä yksiselitteisesti edustaneet vain jompaakumpaa ongelmanratkaisun strategiaa, vaan saattoivat vaihtaa strategiaa kohdatessaan erilaisia ongelmia. Taulukossa 2 on esitetty, kuinka monta oppilasparia käytti mitäkin strategiaa eritasoisten tehtävien kohdalla (Debuggaustehtävä 1 oli helpoin ja Debuggaustehtävä 4 vaikein). Taulukossa on otettu huomioon myös virheenkorjaukseen käytetty aika.

TAULUKKO 2 Oppilaiden debuggausstrategioiden vaihtelu

	Hidas debuggaus (n)	Nopea debuggaus (n)
Debuggaustehtävä 1	Välttely (1)	Tarkka korjaaminen (6)
Debuggaustehtävä 2	Välttely (1) Kokeilu ja erehdys (2)	Tarkka korjaaminen (4)
Debuggaustehtävä 3	Kokeilu ja erehdys (1)	Tarkka korjaaminen (3)
Debuggaustehtävä 4		Tarkka korjaaminen (2)

Vaikutti siltä, että tehtävien vaikeutuessa kokeilun ja erehdyksen strategia yleistyi. Tehtäviä oli neljä, ja ne vaikeutuivat järjestyksessä. Ensimmäinen tehtävä oli helpoin, ja lähes kaikki oppilasparit pystyivät ratkaisemaan sen tarkan korjaamisen strategialla, sillä heillä oli tarpeeksi tietoa ohjelman rakenteesta ja virheen luonteesta. Kun tehtävän vaikeustaso ylitti oppilasparin taitotason, tarkka korjaaminen ei kuitenkaan enää ollut mahdollista, sillä oppilaat eivät välttämättä enää ymmärtäneet ohjelmaa. Tällöin oppilaspari vaihtoi kokeilun ja erehdyksen strategiaan. Strategian vaihto tapahtui aina tarkasta korjaamisesta kokeiluun ja erehdykseen, ei päinvastoin. Tästä voitaisiin päätellä, että tarpeeksi vaikean tehtävän edessä myös ne parit, jotka saivat kaikki tehtävänsä ratkaistua tarkan korjaamisen strategialla, olisivat tehtävien vaikeutuessa lopulta myös siirtyneet kokeilun ja erehdyksen strategiaan.

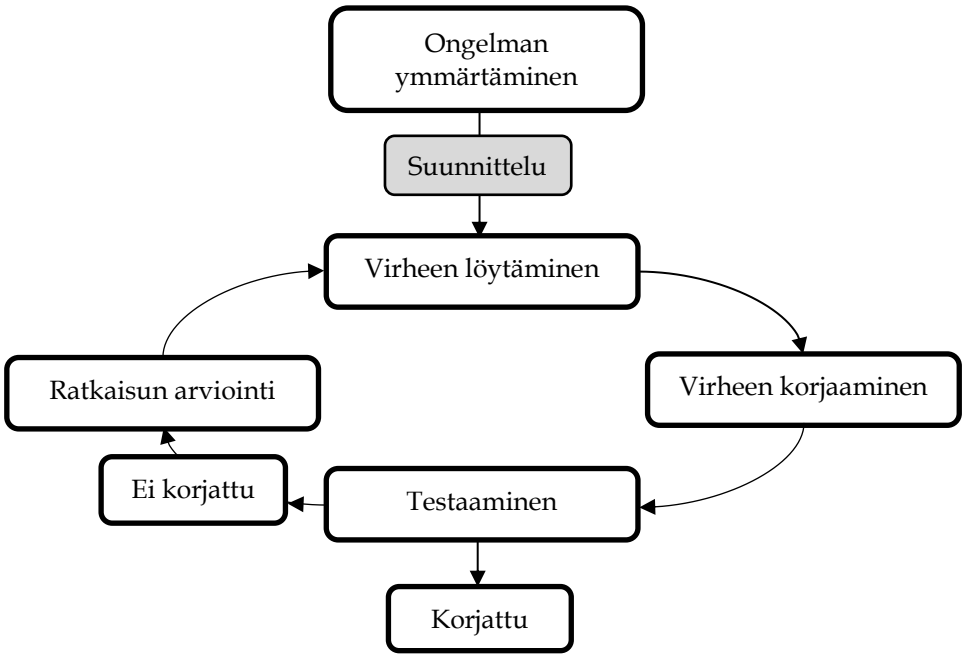
Kaikki oppilasparit saivat vähintään kaksi tehtävää ratkaistua. Tehtävien 1 ja 2 välillä kaksi oppilasparia vaihtoi strategiansa tarkan korjaamisen strategiasta kokeiluun ja erehdykseen. Neljä oppilasparia pääsi tehtävään 3 asti, ja vain kaksi oppilasparia seitsemästä sai kaikki neljä tehtävää tehtyä. Molemmat näistä oppilaspareista olivat motivoituneita, keskittyneitä ja parien sisäinen yhteistyö toimi hyvin. He käyttivät jokaisessa tehtävässä tarkan korjaamisen strategiaa, mikä selittää heidän nopeutensa.

6.3 Oppilaiden debuggausprosessit

Oppilaiden käyttämät debuggausstrategiat heijastuivat suoraan siihen, kuinka heidän debuggausprosessinsa eteni. Oikeastaan jako strategioiden ja prosessien välillä paljastui melko keinotekoiseksi, sillä strategia ohjasi voimakkaasti ongelmanratkaisuprosessin etenemistä. Tämän vuoksi kehä näytti hieman erilaiselta riippuen siitä, mitä strategiaa oppilaspari käytti. Tutkimuksessa havaittiin kaksi erilaista debuggauksen kehää, joista toista ohjasi tarkan korjaamisen strategia ja toista kokeilun ja erehdyksen strategia. Kuviot 2 ja 3 kuvaavat näitä kahta eri prosessia, jotka perustuvat luvussa 2.2.1 esitettyyn debuggauksen kehään (kuvio 1).

Eri oppilasparien debuggausprosessit erosivat selkeimmin siinä, että tarkan korjaamisen strategiaa käyttävät oppilasparit kävivät debuggauskehän läpi vain muutamia kertoja, kun taas kokeilun ja erehdyksen strategiaa käyttävät jäivät kiertämään kehää uudelleen ja uudelleen. Tällöin ongelman ratkaisemiseen kului paljon aikaa ja tehtävä saattoi ajan puutteen vuoksi jäädä kokonaan ratkaisematta. Samasta syystä tarkan korjaamisen strategiaa käyttäneet olivat debuggauksessaan nopeampia ja saivat enemmän tehtäviä tehtyä.

Tarkan korjaamisen strategiassa havaittu rauhallisuus, suunnitelmallisuus ja reflektio näkyivät myös oppilasparien debuggausprosessissa. Kuviossa 2 on esitetty debuggausprosessin eteneminen tarkan korjaamisen strategialla (vrt. Kuvio 1 luvussa 2.2.1).



KUVIO 2. Debuggauksen kehä tarkan korjaamisen strategialla

Suunnitteluvaiheessa oppilaat eivät välttämättä tehneet varsinaista suunnitelmaa, vaan suunnittelulla tarkoitetaan tässä pikemminkin pysähtymistä, ohjelmaan tutustumista, virheen sijainnin päättelyä ja ääneen ajattelua parin kanssa. Pysähtyminen auttoi oppilaita muodostamaan kokonaiskuvan ohjelman toiminnasta, mikä helpotti myöhemmin virheen paikantamista.

Esimerkki 15.

Salli osoittaa vihreää lippua ohjelman ruudulla merkiksi siitä, että ensin kokeillaan ohjelmaa.

Silja: Mut ei, älä vielä (osoittaa näytöltä ohjeita ja lukee ne ääneen).

Salli: Lippu, lippu! (Salli klikkaa lippua, jolloin Piko lähtee liikkeelle näytöllä ja liukuu ruudun reunaan) Hippa!

Silja: Miten tää nyt toimii? (avaa editorin) Piko, sen pitää, niinku neljä askelta eli sen pitäis, tuohon (osoittaa oikeaa kohtaa näytöllä, johon hahmon tulisi pysähtyä). Joten kolme.

Salli: Kolme.

Silja huomaa ”kosketa”-lohkon ja osoittaa sitä näytöllä

Silja: Eikun ”kosketa.. reunaa”

Salli: Tossa

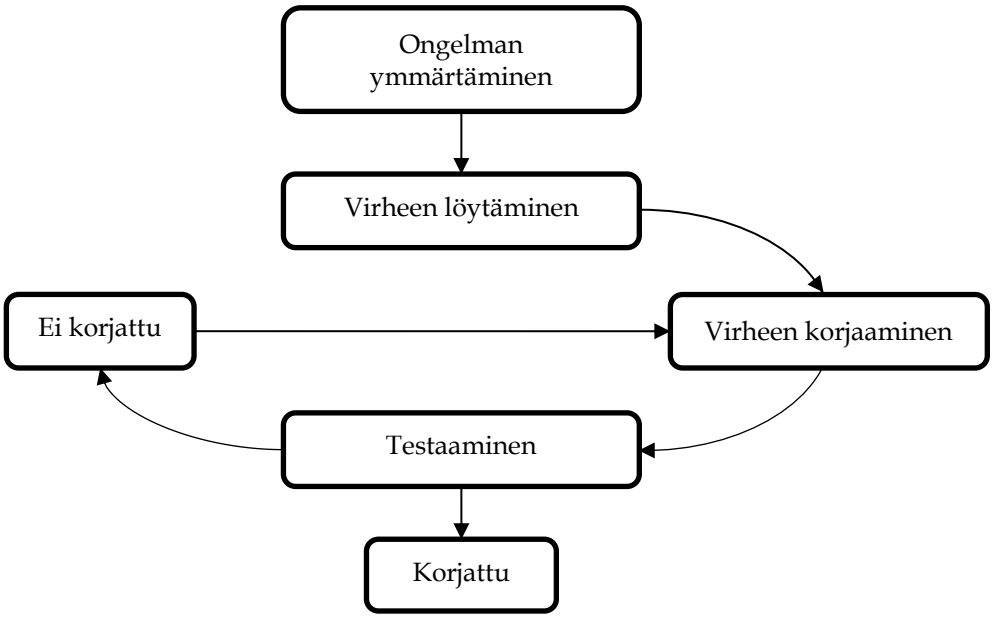
Salli avaa ”kosketa”-lohkon valikon, jossa eri vaihtoehtoja kosketus-ehdolle

Silja: (osoittaa valikkoa) ”Nano”!

Silja kurkottaa käyttämään hiirtä yhdessä Sallin kanssa, vaihtavat kosketus-lohkon valikosta ”reuna”-vaihtoehdon tilalle ”Nanon”.
Salli: Nyt.
Oppilaat testaavat ohjelmaa, ja se toimii.
Molemmat: Jee!

Erityisesti oppilasparit 1 ja 2 käyttivät paljon tarkan korjaamisen strategiaa. Esimerkissä 15 on havaittavissa tarkan korjaamisen debuggausprosessi. Ongelman ymmärtämisen vaiheessa oppilaat tutkivat ensin ohjeita ja sitten skriptejä ennen kuin tekevät muutoksia. Tämän jälkeen heidän oli helppo löytää virhe skriptistä, ja he pystyivät myös nopeasti hylkäämään ensimmäisen ratkaisuehdotuksensa. He kävivät usein debuggauksen kehän läpi vain yksi tai kaksi kertaa ennen kuin saivat ongelman ratkaistua. Jos ongelma ei ratkennut ensimmäisellä kerralla, oppilaat siirtyivät ”ratkaisun arviointi” –vaiheeseen, jossa he arvioivat ensimmäistä ratkaisuehdotustaan kriittisesti ja yrittivät keksiä paremman vaihtoehdon.

Kokeilun ja erehdyksen strategiassa tyypillistä oli saman ratkaisuvaihtoehdon toistaminen uudestaan ja uudestaan. Vaikutti siltä, että tällöin ratkaisu arviointi jäi vähäiseksi tai sitä ei ollut ollenkaan. Kuvio 3 havainnollistaa debuggauksen kehän kokeilun ja erehdyksen strategialla.



KUVIO 3. Debuggauksen kehä kokeilun ja erehdyksen strategialla

Kokeilun ja erehdyksen strategialla debuggauskehästä jätettiin suunnitteluvaihe kokonaan pois. Suunnittelemisen sijaan oppilasparit lähestyivät ongelmaa toiminnan kautta ja laittoivat usein ensimmäisen mieleensä tulevan idean käytäntöön. Kokeilun ja erehdyksen strategialla debuggauksen kehästä puuttui myös kokonaan ratkaisun arvioinnin vaihe, jonka aikana olisi tarkasteltu ongelmaa ja ratkaisua uudelleen, arvioitu ratkaisuvaihtoehdon sopivuutta tilanteeseen ja mahdollisesti tehty muutoksia. Tämän reflektoivan vaiheen puuttuessa ongelmanratkaisu perustui usein pelkälle arvailulle. Esimerkissä 16 Maija ja Neea yrittävät korjata Piko ja Nano tehtävää kokeilun ja erehdyksen strategialla., Vaikka ongelmaa ei saada ratkaistua lukemia muuttamalla, oppilaat eivät harkitse virheen etsimistä jostain muualta, vaan yrittävät edelleen arvailla oikeaa lukemaa. Joskus ongelma saatiin ratkaistua tälläkin menetelmällä, mutta ongelmanratkaisusta puuttui tällöin kokonaan oman toiminnan arviointi.

Esimerkki 16.

(Neea muuttaa Pikon ”liiku”-lohkon lukemaa. Oppilaat testaavat ohjelmaa.)

Neea: Tää ei mee tohon kohtaan. Tää ei mee tohon kohtaan mitä tää sanoo.

Tutkija: Onks se, onks se oikein se kohta varmasti?

Neea: Öö, ei. Katsotaan. (Neea muuttaa taas ”liiku”-lohkon lukemaa.) Miinus seitsemän…

Maija: Mitä sä?

Neea: No on se suurinpiirtein.

Maija: On se.

(Oppilaat testaavat ohjelmaa, ja Piko hyppää ruudun reunan yli)

Maija: Öö.

(Neea huokaisee ja muuttaa taas ”liiku”-lohkon askelmäärää)

7 POHDINTA

7.1 Tulosten tarkastelu

Ohjelmointi peruskoulun oppisisältönä on edelleen hyvin uusi ajatus, ja Suomi on saanut olla edelläkävijän asemassa ottaessaan ohjelmoinnin osaksi perusopetuksen opetussuunnitelmaa ensimmäisten maiden joukossa (OPH 2014). Kuten niin usein muutoksen tullessa, myös ohjelmointi on herättänyt mielipiteitä sekä puolesta että vastaan. Tutkimuksellani pyrin ennen kaikkea vahvistamaan käsitystä siitä, että ohjelmoinnilla todella on paikkansa opetussuunnitelmassa. Ohjelmointi tarjoaa kouluun paljon mahdollisuuksia opettaa 2000-luvun taitoja laaja-alaisesti. Opetussuunnitelman laaja-alaisista alueista ohjelmoinnin opetuksessa toteutuvat erityisesti ajattelu ja oppimaan oppiminen, monilukutaito sekä tieto- ja viestintäteknologian käyttö.

Tässä tutkimuksessa keskityttiin erityisesti ajattelun taitojen harjoitteluun debuggauksen kautta, ja koska oppilaat työskentelivät pareittain, tutkimus kohdentui nimenomaan kollaboratiiviseen ongelmanratkaisuun. Tutkimuksen tarkoituksena oli selvittää, millaista 4.-luokkalaisten oppilasparien vuorovaikutus on pariohjelmointitilanteessa ja millaista heidän ongelmanratkaisunsa on heidän debugatessaan ohjelmia. Tulokset olivat samansuuntaisia aiemman tutkimuksen kanssa ja täydensivät aiempaa tutkimusta. Aiemmassa tutkimuksessa (Polya 1945; Katz & Anderson 1987; Vessey 1985) määritellyt ongelmanratkaisun ja debuggauksen vaiheet olivat havaittavissa myös tutkimukseen osallistuneiden oppilaiden ongelmanratkaisussa. Lisäksi debuggauksen kehältä onnistuttiin lisäksi erittelemään vaiheita, jotka vaikuttivat olevan merkittäviä tehokkaan debuggauksen kannalta: suunnittelu ja ratkaisun arviointi. Tältä osin tutkimus täydentää aiempaa tutkimusta debuggauksen vaiheista.

Tutkimuksessa havaittiin, että oppilaspareissa tapahtui paljon kollaboraatiota ja yhdessä oppimista (ks. myös Cockburn & Williams 2000; Williams & Upchurch 2001). Kaikkien parien kohdalla oli havaittavissa kiinnostusta tehtävää kohtaan, keskittymistä ja ongelmanratkaisua. Oppilasparien sisäinen työnjako

oli selkeä, ja usein he toteuttivat pariohjelmoinnin työnjakoa ajaja–navigoija (ks. myös Williams & Upchurch 2001). Aiemman tutkimuksen (Roschelle & Teasley 1995) mukaisesti myös tässä tutkimuksessa tietokoneen havaittiin toimivan kollaboraation ja vuorovaikutuksen tukena. Yhdellä oppilasparilla vuorovaikutuksen ongelmat aiheuttivat oppilaissa turhautumista ja estivät siksi pariohjelmoinnin kokonaan (ks. myös Lewis 2011; Kuppuswami & Vivekanandan 2004; deClue 2003).

Oppilasparien havaittiin käyttävän ongelmanratkaisussaan kolmea eri ongelmanratkaisun strategiaa, jotka nimettiin: *tarkka korjaaminen, kokeilu ja erehdys* sekä *välttely*. Tarkan korjaamisen sekä kokeilun ja erehdyksen strategiat perustuvat Klahrin ja Carverin (1988) Logo-ohjelmointiympäristössä havaitsemille *focused search*- ja *brute-force* –debuggausstrategioille, mutta soveltuvat paremmin nykypäivän ohjelmoinnin opetukseen, ja erityisesti Scratch-ympäristössä tapahtuvan debuggauksen tarkasteluun. Lisäksi huomattiin, että oppilaat valitsivat eri strategioita sen mukaan, kuinka hyvin heidän taitotasonsa vastasi tehtävän vaikeustasoa. Helpon tehtävän kohdalla lähes kaikki oppilasparit käyttivät tarkan korjaamisen strategiaa, mutta tehtävien vaikeutuessa yhä useammat oppilasparit vaihtoivat kokeilun ja erehdyksen strategiaan. Välttelyn strategiaa taas käytettiin yleensä turhautumisen yhteydessä. Strategioista tehokkain oli tarkan korjaamisen strategia, jolla oppilaat yleensä saivat tehtävänsä ratkaistua hyvin nopeasti (ks. myös Klahr & Carver 1988). Kokeilun ja erehtymisen strategialla tehtävän ratkaisuun meni usein paljon aikaa, mutta useammassa tapauksessa ohjelma saatiin kuitenkin korjattua. Välttelyn strategia luonnollisesti esti koko debuggausprosessin tapahtumisen.

Aiemmassa tutkimuksessa (esim. Klahr & Carver 1988; Ahmadzadeh ym. 2005) on todettu, että mitä enemmän ohjelmoijalla on tietoa ohjelmasta ja mitä kehittyneempi debuggaaja hän on, sitä harvempia debuggauskierroksia hän tarvitsee. Tämä tutkimus tukee näitä tutkimustuloksia. Niin kauan, kun tehtävän taso vastasi oppilasparin taitotasoa, he kykenivät hyödyntämään tarkan korjauksen strategiaa. Tällöin he myös kävivät debuggauskierroksen läpi vain muutamia kertoja ja heidän debuggauksensa oli siksi nopeaa ja tehokasta. Samasta syystä

joidenkin parien debuggaus hidastui tehtävien vaikeutuessa ja heidän vaihtaessaan kokeilun ja erehdyksen strategiaan.

7.2 Implikaatioita

Aikaisemman tutkimustiedon perusteella debuggauksen opetteluun on tärkeää käyttää tarkoituksenmukaisesti aikaa (Klahr & Carver 1988; Chmiel & Loui 2004). Koulussa debuggaustaitoa harjoiteltaessa opettajan on hyvä olla tietoinen erilaisista strategioista, ja ohjata oppilaita tehokkaiden strategioiden käyttöön. Tarkan korjaamisen strategia vaatii muun muassa ohjelmoinnin perusteiden tuntemista, suunnitelmallisuutta ja reflektiokykyä, sekä pariohjelmointitilanteessa myös vuorovaikutustaitoja.

Tämän tutkimuksen perusteella tehokkaampien debuggausstrategioiden käyttöä voidaan tukea esimerkiksi seuraavasti:

1) Oppilas tutustuu käytettyyn ohjelmointikieleen (esim. Scratch) ja ohjelmoinnillisiin peruskonsepteihin (kuten toistorakenne tai ehtolause) ohjatusti ja tavoitteellisesti.

2) Oppilasta ohjataan suunnittelemaan ongelmanratkaisuaan etukäteen ja arvioimaan ratkaisuehdotustaan jälkikäteen, esimerkiksi kysymällä kysymyksiä ja pyytämällä häntä jakamaan ajatuksiaan ääneen.

3) Luokassa harjoitellaan vuorovaikutus- ja yhteistyötaitoja, ja opettaja kiinnittää huomiota parinvalintaan.

Visuaalisten ohjelmointiympäristöjen on todettu kannustavan kokeilun ja erehdyksen strategiaan tavallista ohjelmointikieltä enemmän (Cope ja Simmons 1994). Skriptiin tehdyt muutokset vaikuttavat välittömästi ohjelmaan, jolloin oppilaan on helppoa kokeilla eri ratkaisuvaihtoehtoja satunnaisesti niin kauan, että jokin vaihtoehdoista lopulta saa ohjelman toimimaan. Kokeiluun ja erehtymiseen on helppo turvautua varsinkin silloin, kun ohjelmaa on vaikea ymmärtää esimerkiksi siksi, että siinä käytetyt käskyt ovat oppilaalle vieraita. Scratchin opetuskäytössä olisikin tärkeää kiinnittää huomiota siihen, että oppilaat ymmärtävät ohjelmoinnissa tarvitsemansa ohjelmoinnilliset konseptit kuten ehtolause tai

toistorakenne, ja matemaattiset konseptit kuten esimerkiksi suunta, asteluku tai koordinaatisto. Ohjelmointi on myös hyvin konkreettinen tapa harjoitella matemaattisten käsitteiden ymmärtämistä ja soveltamista.

Kuten tutkimuksessa havaittiin, suunnitelmallisuus on tärkeä osa ohjelmoinnillista ajattelua ja tehokasta debuggausta. Tehokas debuggaus (vrt. tarkan korjaamisen strategia) mahdollistuu, kun ohjelmoijalla on tarpeeksi tietoa ohjelmasta (Ahmadzadeh ym. 2005; Klahr & Carver 1988). Tällaista tietoa on esimerkiksi aikaisempi tieto erilaisista virheistä ja ohjelmoinnillisista rakenteista tai tieto kyseessä olevasta ohjelmasta ja siitä, kuinka sen kuuluisi toimia. Vaikka tässä tutkimuksessa kaikilla oppilailla oli saman verran kokemusta ohjelmoinnista, vaikuttaa siltä, että ne jotka pysähtyivät miettimään ja suunnittelemaan ennen varsinaista virheen korjaamista (tarkat korjaajat), saivat enemmän tietoa ohjelmasta. Tämän tiedon perusteella he tekivät päätelmiä virheen sijainnista, ja siksi virheen paikantaminen oli helpompaa ja nopeampaa. Kokeilijat ja erehtyjät sen sijaan eivät osanneet tehdä päätelmiä ohjelmasta, vaan rajasivat etsintänsä sillä perusteella, mitkä lohkot olivat heille entuudestaan tuttuja.

Tarkasteltaessa debuggauksen kehän muuttumista eri strategioilla (luku 6.3) voidaan huomata, että juuri suunnitelman tekeminen ja suunnitelman uudelleenarviointi olivat vaiheita, jotka esiintyivät ainoastaan tarkan korjaamisen strategialla. Jo Polya (1945) painotti ongelmanratkaisun mallissaan suunnitelman tekemisen tärkeyttä. Jotta ohjelmoinnin opettelu ei jää pelkäksi "puuhasteluksi", opettajan on tärkeää painottaa tarkan suunnitelman tekemistä ennen toimintaan ryhtymistä. Tällöin kehitetään ajattelun taitoja.

Ratkaisun arviointi sekä onnistuneen että epäonnistuneen ongelmanratkaisun kohdalla on myös tärkeää. Jos ratkaisuehdotus ei ratkaisekaan ongelmaa halutulla tavalla, oppilaan olisi tärkeää päästä "ratkaisun arvioinnin" vaiheeseen, jossa hän arvioi kriittisesti omaa ratkaisuehdotustaan ja ajatteluaan. Aloittelevien debuggaajien on aiemmassa tutkimuksessa havaittu olevan joustamattomia virheenkorjausprosesseissaan, jolloin ensimmäisestä ratkaisuehdotuksesta luopuminen voi olla vaikeaa, vaikka se ei toimisikaan (Vessey 1985). Siksi opetuksessa

on tärkeää, että oppilaita ohjataan arvioimaan toimintaansa kriittisesti ja kysymään kysymyksiä kuten ”Miksi ratkaisuni ei toimi?” ja ”Mitä voisin tehdä toisin?”. Myös onnistuneen ongelmanratkaisun jälkeen on tärkeää jättää aikaa ratkaisun arvioinnille, jotta varmistetaan oppilaan ymmärtäneen, mitä hänen ongelmanratkaisuprosessissaan tapahtui. Tällöin hänen debuggaustaitonsa kehittyy, ja hän osaa jatkossa soveltaa ratkaisua toisiin samanlaisiin ongelmiin. Onnistuneen debuggauksen jälkeen oppilaan pitäisi kysyä itseltään kysymyksiä, kuten ”Miksi ohjelma ei toiminut?”, ”Kuinka tiesin, mistä virhe löytyy?” tai ”Olenko aikaisemmin kohdannut samanlaisia ongelmia?”.

Aiemman tutkimuksen perusteella pareittain ohjelmoitaessa oppilaiden taitotason olisi tärkeää olla suunnilleen sama (Chaparro ym. 2005; deClue 2005) ja oppilaat myös toivovat voivansa työskennellä jonkun samantasoisen tai itseään taitavamman kanssa (Kuppuswami & Vivekanandan 2004). Taitavampi ohjelmoija saattaa pariohjelmointitilanteessa turhautua hitaampaan pariin (Lewis 2011). Myös vuorovaikutuksen ja yhteistyön toimimattomuus voi aiheuttaa turhautumista ja estää toisen tai molempien oppilaiden oppimisen (Denner, Werner, Campe ja Ortiz 2014). Myös tässä tutkimuksessa havaittiin, että taitotason epätasapaino ja oppilaiden heikot yhteistyötaidot aiheuttivat oppilaissa turhautumista ja saattoivat estää työskentelyn kokonaan.

Kuten luvussa 3.2 todettiin, myös kollaboratiivisen opetuksen toteuttaminen koulussa on tärkeää, ja opettajat tarvitsevat tähän myös tukea. Tässä tutkimuksessa oppilasparien kollaboraation ja vuorovaikutuksen huomattiin olevan enimmäkseen hyvin onnistunutta: yleensä oppilasparin molemmat oppilaat osallistuivat työskentelyyn aktiivisesti ja pyrkivät kohti yhteistä päämäärää. Pariohjelmointi voi olla motivoiva ja helppo tapa harjoitella kollaboraatiota, sillä pariohjelmoinnissa työnjako on selkeä ja tietokone toimii kommunikoinnin tukena.

7.3 Tutkimuksen luotettavuus

Tutkimus on tapaustutkimus, jonka tavoitteena oli tarkastella oppilaiden debuggausprosesseja yhdessä koulussa. Tutkimuksen siirrettävyys on vahva, sillä ohjelmointijakso toteutettiin osana koulun opetusta sellaisena kuin se tavallisestikin toteutettaisiin. Oppilaat olivat aineistonkeruuseen mennessä hyvin tottuneita tutkijoihin ja tutkimusvälineisiin, enkä usko näiden vaikuttaneen heidän käyttäytymiseensä tai kollaboratiiviseen ongelmanratkaisuunsa kovin paljon. Vaikka tutkimus toteutettiin vain yhden koulun 4.-luokilla, ja tutkittavia oppilaspareja oli vain seitsemän, saadut tulokset olivat yhteneväiset aiemman tutkimuksen kanssa (vrt. Klahr & Carver 1988; Katz & Anderson 1987).

Aineistonkeruuseen osallistuneet tutkijat olivat myös toteuttamassa opetuskokonaisuuden ja ohjaamassa oppilaita tuntien aikana. Koska kokonaisuus kesti useita viikkoja, tutkijalla ehti syntyä suhde tutkittaviin oppilaisiin. Tämä vaikutti tutkimuksen luotettavuuteen monella tapaa. Toisaalta uskon, että tutkijaan tutustuminen ja tottuminen tekivät tutkimustilanteesta oppilaille luontevamman ja turvallisemman, jolloin heidän oli helpompi toimia kuten he olisivat toimineet oman luokanopettajansa kanssa. Tämä vahvistaa tutkimuksen siirrettävyyttä. Toisaalta aineiston tulkitseminen objektiivisesti oli minulle tutkijana vaikeampaa, sillä tunsin oppilaat ja minulla oli jokaiseen heistä erilainen suhde. Aineiston analyysin ja tulosten raportoinnin aikana pyrin jatkuvasti olemaan tietoinen suhteestani eri oppilaisiin ja tarkastelemaan aineistoa objektiivisesti.

Oppilaiden debuggausprosessit saattoivat olla hyvin nopeita, ja kaikki oppilaat eivät välttämättä ilmaisseet ajatuksiaan ääneen. Niinpä prosessien etenemisen ja vaiheiden havaitseminen aineistosta oli paikoitellen vaikeaa. Oppilaiden luonnollinen puhe ja sanaton viestintä saattoivat olla tulkittavissa monilla eri tavoilla. Tutkijana pyrin kuitenkin mahdollisimman tarkkaan tulkintaan perehtymällä aineistoon huolellisesti useiden luku- ja katselukertojen kautta ja tekemällä paljon merkintöjä myös oppilaiden sanattomasta viestinnästä. Pyrin myös tulkinnan läpinäkyvyyteen lukijalle valitsemalla runsaasti aineistoesimerkkejä.

7.4 Jatkotutkimushaasteet

Debuggaus on hyvin merkittävä ohjelmoinnillinen taito, jonka avulla voidaan harjoitella ongelmanratkaisua. Debuggaukseen on aiemmassa tutkimuksessa kiinnitetty vain vähän huomiota, ja esimerkiksi tämän tutkimuksen tuloksia olisi tärkeää laajentaa ja vahvistaa suuremmalla aineistolla. Myös think-aloud –menetelmän käyttäminen lasten debuggausprosessien tutkimisessa olisi hyödyllistä, jotta lasten ajatuksen kulku saataisiin paremmin näkyväksi (Lye ja Koh 2014).

Ohjelmoinnillisen ajattelun uskotaan olevan tärkeä tulevaisuuden taito, jota tarvitaan jokapäiväisessä elämässä. Toiset taas epäilevät ohjelmoinnillisen ajattelun olevan hyödyllistä vain tietokoneiden parissa työskenteleville (Denning 2017). Mielenkiintoista ja tärkeää olisi kiinnittää huomiota esimerkiksi ongelmanratkaisutaidon siirtymiseen; pystyykö oppilas jatkossa hyödyntämään virheenkorjauksen kehää muissa konteksteissa kuin korjatessaan virheitä ohjelmointiympäristössä?

Monissa maissa ohjelmointi ja ohjelmoinnillinen ajattelu on otettu innolla opetussuunnitelmiin ja opetuksen tueksi on jo kehitetty paljon materiaaleja. Tieteellistä tutkimusta on kuitenkin vielä varsin vähän. Ohjelmoinnin opetusta kohtaan on edelleen paljon ennakkoluuloja, jotka saattavat jopa estää monia opettajia tutustumasta aiheeseen ja hyödyntämästä sen mahdollisuuksia omassa luokassaan. Jos opettajat kokevat ohjelmoinnin opetuksen vaikeaksi ja pelottavaksi, on olemassa riski, että ohjelmoinnin opetus jää myös hyvin pinnalliseksi. Jatkotutkimuksessa olisikin tärkeää keskittyä tukemaan opettajia kaikin tavoin uuden oppisisällön haltuun ottamisessa ja madaltamaan kynnystä ohjelmoinnin ja ohjelmoinnillisen ajattelun opettamiseen. Oppimisen arviointi, opetuksen eriyttäminen ja ohjelmoinnin opettamisen harjoitteet ovat aiheita, joihin jatkotutkimuksessa tulee kiinnittää huomiota.

LÄHTEET

Aarnos, E. 2001. Kouluun lapsia tutkimaan: havainnointi, haastattelu ja dokumentit. Teoksessa Aaltola, J. & Valli, R. (toim.) Ikkunoita tutkimusmetodeihin 1. Jyväskylä: PS-kustannus.

Ahmadzadeh, M., Elliman, D. & Higgins, C. 2005. An analysis of patterns of debugging among novice computer science students. ACM SIGCSE Bulletin 37 (3), 84-88.

Balanskat, A. & Engelhardt, K. 2014. Computing our Future: Computer programming and coding - Priorities, school curricula, and initiatives across Europe. European Schoolnet.

Barr, D., Harrison, J., Conery, L., 2011. Computational thinking: A digital age skill for everyone. Learning & Leading with Technology 38 (6), 20–23

Barr, V. & Stephenson, C. 2011. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? ACM Inroads 2 (1), 48–54.

Braun, V. & Clarke, V. 2006. Using thematic analysis in psychology. Qualitative research in psychology 3 (2), 77-101.

Buckley, S. 2012. The Role of Computational Thinking and Critical Thinking in Problem Solving in a Learning Environment. Proceedings of the 11th European Conference on e-Learning: ECEL, 63–70.

Calder, N. 2010. Using Scratch: An Integrated Problem-Solving Approach to Mathematical Thinking. Australian Primary Mathematics Classroom 15 (4), 9–14.

Care, E. & Griffin, P. 2014. An approach to assessment of collaborative problem solving. Research and practice in technology enhanced learning. 9 (3), 367–388.

Carver, S. 1988. Learning and transfer of debugging skills: Applying task analysis to curriculum design and assessment. Teoksessa R. E. Mayer (toim.) 1988. Teaching and learning computer programming. Hillsdale, NJ: Erlbaum

CAS 2015. Computational thinking – A guide for teachers. https://community.computingatschool.org.uk/resources/2324/single Luettu 6.6.2018.

Chaparro, E. A., Yuksel, A., Romero, P. & Bryant, S. 2005. Factors affecting the perceived effectiveness of pair programming in higher education. Proceedings of the 17th workshop of the Psychology of Programming Interest Group, 5–18.

Childers, S. 2003. Computer literacy: Necessity or buzzword?. Information Technology and Libraries, 22 (3), 100–104.

Chmiel, R. & Loui, M. C. 2004. Debugging: from novice to expert. ACM SIGCSE Bulletin 36 (1), 17–21.

Cockburn, A. & Williams, L. 2000. The costs and benefits of pair programming. Teoksessa Extreme programming examined, 223-247.

Cope, P. & Simmons, M. 1994. Some effects of limited feedback on performance and problem-solving strategy in a Logo microworld. Journal of Educational Psychology 86 (3), 368–379.

Craig, M. & Horton, D. 2009. Gr8 designs for Gr8 girls: a middle-school program and its evaluation. ACM SIGCSE Bulletin 41 (1), 221-225.

Crain, W. 2011. Theories of development: Concepts and application (6. painos). Lontoo: Pearson.

CSTA 2011. Operational definition of computational thinking for K-12 education.

https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf Luettu 6.6.2018

Damon, W. & Phelps, E. 1989. Critical distinctions among three approaches to peer education. International Journal of Educational Research. 13 (1), 9-19. doi: http://dx.doi.org/10.1016/0883-0355(89)90013-X Luettu 8.6.2018.

deClue, T. H. 2003. Pair programming and pair trading: Effects on learning and motivation in a CS2 course. Journal of Computing in Small Colleges 18, 49–56.

Dede, C. 2009. Comparing frameworks for ”21st century skills”. Harvard Graduate School of Education July, 2009.

Denner, J., Werner, L., Campe, S. & Ortiz, E. 2014. Pair programming: under what conditions is it advantageous for middle school students? Journal of research on technology in education 46 (3), 277–296.

Denning, P. 2017. Remaining trouble spots with computational thinking. Communications of the ACM 60 (6), 33–39.

deSchryver, M. D. & Yadav, A. 2015. Creative and computational thinking in the context of new literacies: working with teachers to scaffold complex

technology-mediated approaches to teaching and learning. Journal of Technology and Teacher Education 23 (3), 411-431.

Dillenbourg, P. 1999. Introduction: What do you mean by "collaborative learning"?. Teoksessa Dillenbourg, P. (toim.) 1999. Collaborative learning: Cognitive and computational approaches. Amsterdam: Pergamon, Elsevier Science.

diSessa, A. 2000. Changing minds: computers, learning, and literacy. Lontoo: The MIT press.

Erikson, K. A. & Simon, H. A. 1980. Verbal reports as data. Psychological Review 87 (3), 215–251.

Florida, R. 2002. The rise of the creative class. New York: Basic Books.

Gee, J. P. 2011. An introduction to discourse analysis: theory and method. London: Routledge.

Griffin, P., Care, E. & McGaw, B. 2011. The changing role of education and schools. Teoksessa Griffin, P., McGaw, B. & Care, E. (toim.) 2012. Assessment and teaching of 21st century skills. New York: Springer.

Grover, S. & Pea, R. 2013. Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. Teplsessa Proceeding of the 44th ACM technical symposium on Computer science education, 723-728.

Grover. S. & Pea. R. 2018. Computational thinking: A competency whose time has come. Teoksessa Sentence, S., Barendsen, E & Schulte, C. (toim.) 2018. Computer Science Education: Perspectives on teaching and learning. Lontoo: Bloomsbury Academic.

Haapasalo, L. 2011. Oppiminen, tieto ja ongelmanratkaisu. Joensuu: Medusa-Software

Hesse, F., Care, E., Buder, J., Sassenberg, K. & Griffin, P. 2015. A framework for teachable collaborative problem solving skills. Teoksessa Griffin, P., McGaw, B. & Care, E. (toim.) 2015. Assessment and teaching of 21st century skills. New York: Springer.

Hillel, J., Kieran, C. & Gurtner, J.-L. 1989. Solving structured geometric tasks on the computer: the role of feedback in generating strategies. Educational studies in mathematics 20, 1–39.

Hirsjärvi, S. & Sinivuori, E. 2000. Tutki ja kirjoita. 6. uud. laitos. Helsinki: Tammi.

Johnson, D. W., Johnson, R. T., Roseth, C. J. & Seob Shin, T. 2014. The relationship between motivation and achievement in interdependent situations. Journal of Applied Social Psychology. 44 (9), 622–633.

Kallionpää, O. 2014. Mitä on uusi kirjoittaminen? Uusien mediakirjoitustaitojen merkitys. Media & viestintä 37 (4), 60–78.

Katz, I. R. & Anderson, J. R. 1987. Debugging: an analysis of bug-location strategies. Human-Computer Interaction. 1987–1988 (3), 351–399.

Kelleher, C. & Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys 37 (2), 83–137.

Klahr, D. & Carver, S. 1988. Cognitive objectives in a LOGO debugging curriculum: instruction, learning and transfer. Cognitive Psychology 20 (3), 362–404.

Kuppuswami, S. & Vivekanandan, K. 2004. The effects of pair programming on learning efficiency in short programming assignments. Informatics in Education 3, 251–266.

Kuula, A. 2006. Tutkimusetiikka: Aineistojen hankinta, käyttö ja säilytys. Tampere: Vastapaino.

Laine, M., Bamberg, J. & Jokinen, P. 2007. Tapaustutkimuksen käytäntö ja teoria. Teoksessa Laine, M., Bamberg, J. & Jokinen, P. (toim.) Tapaustutkimuksen taito. Helsinki: Yliopistopaino.

Lee, C. D. & Smagorinsky, P. (toim.) 2000. Vygotskian perspectives on literacy research: Constructing meaning through collaborative inquiry. Cambridge University Press.

Lewis, C. M: 2011. Is pair programming more effective than other forms of collaboration for young students? Computer science education 21 (2), 105–134.

Liedenberg, J., Mentz E. & Breed, B. 2012. Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology. Computer science education 22 (3), 219–236.

Littleton, K. & Häkkinen, P. 1999. Learning together: Computer-Based Collaborative Learning. Teoksessa Dillenbourg, P. (toim.) 1999. Collaborative learning: Cognitive and computational approaches. Amsterdam: Pergamon, Elsevier Science.

Lombardi, J. V. 1983. Computer literacy: the basic concepts and language. Bloomington IN: Indiana University Press.

Lopez, V. & Hernandez, M. I. 2015. Scratch as a computational modeling tool for teaching physics. Physics Education 50 (3), 310–316.

Luckin, R., Baines, E., Cukurova, M., Holmes, W. & Mann, M. 2017. Solved! Making the case for collaborative problem-solving. A report for Nesta. Lontoo: Nesta.

Luukka, M.-R. 2013. Opetussuunnitelmat uudistuvat: tekstien lukijasta ja kirjoittajasta monilukutaituriksi. Kieli, koulutus ja yhteiskunta, joulukuu 2013.

Lye, S. Y. & Koh, J. H. L 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior 41 (2014), 51–61.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. 2008. Programming by choice: urban youth learning programming with Scratch. ACM SIGCSE Bulletin 40 (1), 367-371. doi: https://doi.org/10.1145/1352322.1352260 Luettu 8.6.2018.

Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. 2010. The scratch programming language and environment. ACM Transactions on. Computing. Education 10 (4), artikkeli no. 16. doi: http://doi.acm.org/10.1145/1868358.1868363. Luettu 30.10.2017.

Mason, J., Burton, L. & Stacey, K. 1982. Thinking mathematically. Lontoo: Addison-Wesley Pub. Co.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L. & Zander, C. 2008. Debugging: a review of the literature from an educational perspective. Computer Science Education 18 (2), 67–92.

McMillan, S. 1996. Literacy and computer literacy: definitions and comparisons. Computers & Education 27 (3-4), 161–170.

Metsämuuronen, J. 2006. Laadullisen tutkimuksen perusteet. Teoksessa Metsämuuronen, J. (toim.) Laadullisen tutkimuksen käsikirja. Jyväskylä: Gummerus Kirjapaino Oy.

OECD 2010. PISA 2012 Field trial problem solving framework. http://www.oecd.org/dataoecd/8/42/46962005.pdf Luettu 6.6.2018.

OECD. 2013. PISA 2015 Draft collaborative problem solving framework.

OECD 2015. PISA 2015 Collaborative problem-solving framework. https://www.oecd.org/pisa/pisaproducts/Draft%20PISA%202015%20Collaborative%20Problem%20Solving%20Framework%20.pdf Luettu 8.6.2018.

OPH 2014. Perusopetuksen opetussuunnitelman perusteet. 2014. Helsinki: Opetushallitus.

Papert, S. 1980. Mindstorms – Children, computers and powerful ideas. New York: Basic Books.

Ploetzner, R., Dillenbourg, P., Preier M. & Traum, D. 1999. Learning by explaining to oneself and to others. Teoksessa Dillenbourg, P. (toim.) 1999. Collaborative learning: Cognitive and computational approaches. Amsterdam: Pergamon, Elsevier Science.

Polya, G. 1945. How to solve it. Princeton, NJ: Princeton University Press.

Prensky, M. 2001. Digital Natives, Digital Immigrants. On the Horizon 9 (5), 1–6.

Prensky, M. 2008. Programming is the new literacy. Edutopia. https://www.edutopia.org/literacy-computer-programming Luettu 7.6.2018.

Preston, D. 2005. Pair programming as a model of collaborative learning: a review of the research. Journal of Computing Sciences in colleges, 20 (4), 39–45.

P21 2007. Framework for 21st century learning. http://www.p21.org/our-work/p21-framework Luettu 8.6.2018.

Resnick, M. & Brennan, K. 2012. New frameworks for studying and assessing the development of computational thinking. American Educational Research Association. Vancouver, Canada.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. 2009. Scratch: Programming for all. Communications of the ACM 52 (11), 60–67.

Roschelle, J. & Teasley, S. D. 1995. The construction of shared knowledge in collaborative problem solving. Teoksessa O'Malley, C. (toim.) 1995 Computer supported collaborative learning, 69-97. Springer, Berlin, Heidelberg.

Saarela-Kinnunen, M. & Eskola, J. 2001. Tapaus ja tutkimus = tapaustutkimus? Teoksessa Aaltola, J. & Valli, R. (toim.) Ikkunoita tutkimusmetodeihin 1. Jyväskylä: PS-kustannus.

Shute, V. J., Sun, C. & Asbell-Clarke, J. 2017. Demystifying computational thinking. Educational Research Review 22 (Syyskuu), 1–17.

Silva, 2008. Measuring skills for the 21st century. Education sector reports 11 (2008).

Simmons, M. & Cope, P. 1993. Angle and Rotation: Effects of Different Types of Feedback on the Quality of Response. Educational Studies in Mathematics, 24 (2), 163-176.

Syrjälä, L. 1994 Tapaustutkimus opettajan ja tutkijan työvälineenä. Teoksessa Syrjälä, L., Ahonen, S., Syrjäläinen, E. & Saari, S. (toim.) Laadullisen tutkimuksen työtapoja. Helsinki: Kirjayhtymä.

Vee, A. 2013. Understanding computer programming as a literacy. Literacy in composition studies 1 (2), 42–64.

Vessey, I., 1985. Expertise in debugging computer programs: A process analysis. International Journal of Man-Machine Studies 23, 459-494.

Weng, W. 2015. Eight skills in future work. Education 135 (4), 419–422.

Wilensky, U., Brady, C. E. & Horn, M. S. 2014. Fostering computational literacy in science classrooms. Communications of the ACM, 57 (8), 24-28.

Williams, L. & Upchurch, R. L. 2001. In support of student pair-programming. ACM SIGCSE Bulletin 33 (1), 327–331.

Wilson, R. 2013. Skills anticipation – the future of work and education. International journal of educational research 61 (2013), 101–110.

Wing, J. 2006. Computational Thinking. Communications of the ACM 49 (3), 33–35.

Wing, J. 2011. Research notebook: Computational thinking—What and why? The Link Magazine, Spring. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why Luettu 7.6.2018.

Wyeth, P. 2008. How young children learn to program with sensor, action, and logic blocks. The Journal of the learning sciences 17 (4), 517–550.