Graafinen ohjelmointi luokanopettajille: teoria ja käytäntö

Pro gradu -tutkielma
Turun yliopisto
Tietotekniikan laitos
Tietojenkäsittelytiede
2021
Heidi Kaarto

TURUN YLIOPISTO
Tietotekniikan laitos

Heidi Kaarto: Graafinen ohjelmointi luokanopettajille: teoria ja käytäntö

Pro gradu -tutkielma, 69 s.
Tietojenkäsittelytiede
Toukokuu 2021

Ohjelmointi on tullut Suomessa vuonna 2014 osaksi perusopetusta: ohjelmointi lisättiin Opetushallituksen perusopetuksen opetussuunnitelmien perusteisiin osaksi muita oppiaineita, kuten matematiikkaa ja käsitöitä. Siispä vuodesta 2016 lähtien alakoulun luokanopettajien on vuosiluokilla 3–6 opetettava ohjelmointia graafisessa ohjelmointiympäristössä. Monilta luokanopettajilta voi kuitenkin puuttua ohjelmoinnin ja ohjelmoinnin opetuksen koulutus, koska se on pääasiassa perustunut vapaaehtoisuuteen.
Tässä tutkimuksessa on tarkoitus selvittää, mitä tietoa ohjelmointia opettava luokanopettaja tarvitsee. Tutkimus keskittyy graafisen ohjelmoinnin opetukseen alakoulun vuosiluokilla 3–6. Luokanopettajan tarvitsema tieto jäsennellään teknologispedagogisen sisältötiedon (Technological, Pedagogical, and Content Knowledge, TPACK) viitekehyksen avulla. Lisäksi tutkimuksessa luodaan täydennyskoulutusmalli, jonka tarkoitus on välittää graafisen, lohkopohjaisen ohjelmoinnin ja sen opettamisen tieto koulutukseen osallistuville opettajille. Keväällä 2021 pidetyn alakoulun luokanopettajille suunnatun täydennyskoulutuksen graafisen ohjelmoinnin ja sen opettamisen osuudet perustuvat tähän tutkimukseen. Koulutus oli onnistunut ja osallistujat kokivat sen hyödylliseksi. Monet opettajat kaipaavat kuitenkin valmiita materiaaleja, joita he voisivat hyödyntää opetuksessaan.

Asiasanat: ohjelmoinnin opetus, alakoulu, täydennyskoulutus, graafinen ohjelmointi, lohkopohjainen ohjelmointi

UNIVERSITY OF TURKU
Department of Computing

HEIDI KAARTO: Graphical Programming for Primary School Teachers: Theory and Practise

Master of Science Thesis, 69 p.
Computer Science
May 2021

In 2014, programming become part of basic education in Finland: programming was added to the Finnish core curriculum for basic education by the National Agency of Education. Programming has been added to other subjects such as mathematics and crafts. Therefore many primary school teachers have been required to teach programming in grades 3–6 since 2016. Many teachers' may still have too little knowledge on programming and teaching programming since training in them has primarly been voluntary.
The purpose of this research is to investigate what knowledge the teachers would need to teach programming in primary school. The research focuses on graphical programming in grades 3–6. The knowledge of the teacher is outlined with the Technological, Pedagogical, and Content Knowledge (TPACK) framework. In addition to that, a design for how to offer this knowledge to the teachers is created. The design have been implemented into in-service training for teachers in Spring 2021. The training was a success and the teachers felt that it was useful. However, many teachers are wishing for teaching materials that they could use in their teaching.

Keywords: teaching programming, primary education, in-service training, graphic programming, block-based programming

Sisällys

Kuvat

Taulukot

1 Johdanto

Digitaalisuus ja teknologia ovat koko ajan enemmän ja tiukemmin osa niin arkija työelämää kuin koulutustakin. Vuonna 2020 87 prosentilla suomalaisista kotitalouksista oli tietokone [1] ja vähintään 10 henkilöä työllistävien yritysten henkilöstöstä 81 prosenttia käytti työssään tietokonetta [2]. Jopa 86 % suomalaisista (16–89-vuotiaista) on käyttänyt Internetiä päivittäin tai lähes päivittäin [1]. Internetiä käytettiin kotitalouksissa pöytätietokoneilla, kannettavilla tietokoneilla, tablettitietokoneilla ja matkapuhelimilla [1].

Vielä 2000-luvun alkupuolella ajateltiin, että digitaalisena aikana lapsista kasvaa diginatiiveja, jotka oppivat luonnostaan käyttämään teknologioita ja että oppilaat ovat teknologisilta taidoiltaan parempia kuin opettajansa, on myöhemmin kuitenkin huomattu, ettei se pidä paikkansa [3]. Vuoden 2019 Lukiolaisbarometrin mukaan lukiolaiset käyttävät omasta mielestään sujuvasti (80 prosenttia vastaajista oli samaa tai täysin samaa mieltä) yleisimpiä toimisto-ohjelmia (Word, Excel, PowerPoint tai muut vastaavat), mutta 75 % vastaajista ei koe osaavansa tehdä tai päivittää Internet-sivuja, 53 % ei koe olevansa taitava luomaan ja editoimaan kuvia, videoita ja/tai musiikkia ja 88 % ei koe olevansa taitava koodaamaan [4]. Teknologioiden eli laitteiden ja niiden sovellusten käyttöliittymät on nykyään rakennettu usein niin helpoiksi ja selkeiksi käyttää, ettei teknologioiden toimintaperiaatteista tarvitse ymmärtää paljoakaan.

Taidottomuus digitaalisissa asioissa voi kuitenkin nykypäivänä rajoittaa opiskelu-

ja työmahdollisuuksia. Lisäksi digitaalisuuden vaikutusten ymmärtäminen on tarpeellista omien oikeuksiensa turvaamiseksi ja yhteiskunnan kehityksestä käytyyn keskusteluun osallistumiseksi [5]. Olisi siis hyödyllistä opettaa ymmärrystä teknologioiden toiminnasta ja ohjelmoinnillista ajattelua [6] eli tulevaisuuden taitoja (engl. *21 century skills*). Voidaan siis olla sitä mieltä, että tulevaisuuden taitoja tarvitaan ja niitä tulisi opettaa kaikille jo perusopetuksesta lähtien.

Tulevaisuuden taitojen tarve ei rajoitu Suomeen. Monissa muissa maissa on myös todettu tulevaisuuden taitojen tarve, joten niitä on sisällytetty koulutukseen. Australiassa on vuodesta 2014 ollut digitaalisten teknologioiden oppiaine (*Digital Technologies*) [7], jossa tarkoitus on kehittää ohjelmoinnillista ajattelua ja oppia datasta, digitaalisista järjestelmistä ja ratkaisujen toteuttamista ohjelmoimalla [8]. Englannissa on yhtä kauan ollut oppiaineena *computing* [9], joka sisältää tietojenkäsittelytiedettä (*computer science*), tietoteknologiaa (*information technology*) ja digitaalista luku- ja kirjoitustaitoa (*digital literacy*) [8]. Myös Uudessa-Seelannissa (vuodesta 2011) [8] ja Puolassa (1960-luvulta asti) [10] on omat oppiaineensa samantapaisille aiheille.

Espanjassa taas vaihtelee, opetetaanko ohjelmointia omana oppiaineenaan vai integroituna muihin oppiaineisiin kuten matematiikkaan tai luonnontieteisiin [11]. Ruotsissa ohjelmointia on opetettu osana matematiikkaa vuodesta 2018 lähtien [12]: vuosiluokilla 1–3 annetaan vaiheittaisia toimintaohjeita ja vuosiluokilla 4–6 ohjelmoidaan [8].

Suomessa on päädytty samantapaiseen ratkaisuun kuin Ruotsissa. Vuonna 2014 julkaistuihin perusopetuksen opetussuunnitelman perusteisiin lisättiin ohjelmoinnin opetusta [13]. Siispä vuodesta 2016 alkaen vuosiluokilla 1–2 harjoitellaan vaiheittaisten toimintaohjeiden seuraamista ja laatimista ja vuosiluokilla 3–6 toimintaohjeet muutetaan tietokoneohjelmiksi graafisissa ohjelmointiympäristöissä ja tutustutaan robotiikkaan ja automaatioon. Yläkoulun puolella vuosiluokilla 7–9 ohjelmoidaan

tekstipohjaisilla ohjelmointikielillä, hyödynnetään sekä itse kirjoitettuja että valmiita ohjelmia matematiikassa ja opitaan laitteen ohjaamista ohjelmoimalla käsitöissä.

Monet luokanopettajat eivät ole saaneet opettajankoulutuksessaan koulutusta ohjelmoinnin opettamiseen, minkä voi nähdä esimerkiksi vuonna 2015 ohjelmoinnin verkkokurssille ilmoittautuneiden opettajien suuressa määrässä [14]. Vuonna 2018 tilanne ei ollut parantunut tarpeeksi: Niemelän [15] mukaan suomalainen opettajankoulutus ei ollut täysin mukautunut uudistuksiin ja sekä opettajaksi opiskelevia että valmistuneita opettajia on koulutettava tietojenkäsittelytieteen konsepteista.

Vuonna 2018 suurin osa opettajista koki, että heillä on vähintään perustason tieto- ja viestintäteknologiset taidot (luokanopettajista 90 %), mutta jopa 59 % opettajista sanoi, ettei osaa graafista ohjelmointia [16]. Vuonna 2019 kaksi kolmasosaa opettajista jäi vaille pisteitä alkeisohjelmointitehtävissä [17]. Luokanopettajat saivat keskimäärin 0,6 pistettä kahdesta ja oppilaat 0,1 (tytöt) tai 0,2 (pojat) [17].

Vaikuttaa siis siltä, että opettajat tarvitsevat tukea graafiseen ohjelmointiin liittyvässä opetuksessa, jotta voidaan myös varmistaa oppilaiden ohjelmoinnin osaaminen tulevaisuudessa. Osa opettajista on myös itse sanonut kaipaavansa täydennyskoulutusta digiaiheista (61 % vuonna 2017 ja 59 % 2018) [16].

Tämän tutkimuksen tarkoitus on pyrkiä vastaamaan graafisen ohjelmoinnin ja sen opettamisen täydennyskoulutustarpeeseen matematiikan oppiaineen osalta. Tutkimus pyrkii selvittämään, millaista tietoa luokanopettaja tarvitsee voidakseen opettaa ohjelmointia graafisessa ohjelmointiympäristössä ja miten tämä tieto voitaisi opettajille tarjota. Tutkimuksessa kartoitetaan, millaista tietoa opettaja tarvitsee ja luodaan graafisen ohjelmoinnin opetuksen täydennyskoulutusmalli. Mallia kokeiltiin osana keväällä 2021 pidettyä alakoulun luokanopettajien ohjelmoinnin täydennyskoulutusta.

Tutkimuskysymykset ja tutkimuksessa käytetty menetelmä esitellään luvussa 2. Luokanopettajan tarvitsemaa tietoa jäsennellään tutkimuksessa teknologis-pedago-

gisen sisältötiedon viitekehyksen avulla: viitekehys esitellään luvussa 3 ja tieto luvussa 4. Tämän tiedon tarjoamiseksi opettajille on kehitetty täydennyskoulutusmalli, joka esitellään luvussa 5. Täydennyskoulutus on järjestetty keväällä 2021 ja sen tuloksia käydään läpi ja pohditaan luvussa 6.

2 Tutkimuksen kuvaus

Tutkimuksen tarkoitus on tukea alakoulun graafisen ohjelmoinnin opetukseen liittyvään täydennyskoulutustarpeeseen vastaamista. Tutkimuksessa kartoitetaan suunnittelun tutkimuksen menetelmin sitä tietoa, jota luokanopettaja tarvitsee voidakseen opettaa graafista ohjelmointia ja luodaan siihen perustuva täydennyskoulutusmalli. Mallia myös kokeillaan osana alakoulun luokanopettajille suunnattua täydennyskoulutusta.

2.1 Tutkimuksen tarkoitus ja tutkimuskysymykset

Perusopetuksen opetussuunnitelman perusteissa tieto- ja viestintäteknologialla on kaksi roolia: se toimii sekä oppimisen välineenä että kohteena [13]. Oppilaiden on tarkoitus oppia ymmärtämään tieto- ja viestintäteknologian käyttöperiaatteita, mutta myös toimintaperiaatteita. Vuosiluokkien 1–2 matematiikassa tutustutaan ohjelmoinnin alkeisiin laatimalla ja testaamalla vaiheittaisia toimintaohjeita. Vuosiluokkien 3–6 matematiikassa toimintaohjeet siirretään tietokoneohjelmiksi graafisessa ohjelmointiympäristössä. Käsityössä harjoitellaan laitteiden ohjaamista ohjelmoimalla (esim. robotiikka). Ohjelmoinnin opetuksen tarkoituksena on oppia ymmärtämään, miten teknologiat toimivat ja sen, että toiminta riippuu ihmisen tekemistä ratkaisuista.

Luokanopettajilla on siis velvoite ottaa ohjelmointi osaksi vuosiluokkien 3–6 opetusta. Monelta luokanopettajalta voi kuitenkin puuttua ohjelmointi- ja ohjelmoin-

nin opetustaidot, koska niiden opetus on perustunut pääasiassa vapaaehtoisuuteen. Opettajat ovat voineet osallistua täydennyskoulutuksiin, joissa ohjelmointia on opetettu, koska koulutuksia on järjestetty useita sekä ennen opetussuunnitelman voimaan tuloa että sen jälkeen. Koska opettajien ohjelmointitaito ei kuitenkaan ole vielä kovin hyvä (kuten voi nähdä Digiajan peruskoulu II -julkaisun esittelemistä tuloksista [17]), ovat täydennyskoulutukset silti tarpeellisia.

Tämän tutkimuksen tarkoitus on selvittää, mitä tietoa luokanopettajat tarvitsevat voidakseen opettaa ohjelmointia graafisessa ohjelmointiympäristössä vuosiluokilla 3–6. Lisäksi tutkimuksen tarkoitus on muodostaa täydennyskoulutusmalli, jonka avulla määritelty tieto voidaan tarjota luokanopettajille. Tutkimuksen varsinaiset tutkimuskysymykset ovat

1. Mitä luokanopettajan on tiedettävä voidakseen opettaa ohjelmointia graafisessa ohjelmointiympäristössä?

2. Miten ohjelmointi- ja ohjelmoinnin opetustieto voidaan välittää opettajille?

Tutkimus liittyy Opetushallituksen rahoittamaan Ohjelmoinnillisen ajattelun työkalut peruskoulussa -projektiin, jonka tarkoitus on tukea peruskoulun opettajien ohjelmoinnin ja sen opettamisen osaamista. Projektissa järjestetään kaksi koulutusta, jotka on suunnattu alakoulun (keväällä 2021) ja yläkoulun (syksy 2021) opettajille. Tämä tutkimus keskittyy alakoulun koulutuksen graafisen ohjelmoinnin osuuteen.

2.2 Tutkimuksen toteutus

Tutkimuksessa pyritään selvittämään, mitä luokanopettajien on tiedettävä, jotta he voivat opettaa ohjelmointia graafisessa ohjelmointiympäristössä ja miten ohjelmointi- ja ohjelmoinnin opetustieto voidaan luokanopettajille tarjota. Tutkimuksen

kysymykset ovat laajoja ja tarkoituksena on muodostaa sekä tiedon että sen välittämisen mallit. Tutkimuksen tarkoitus on tarjota teoreettista näkemystä tutkittuun aiheeseen (eli mallin siitä tiedosta, jonka luokanopettaja tarvitsee) ja käytännön ratkaisu (eli ratkaisu sille, miten tieto voidaan välittää opettajille). Näin ollen tutkittavana on sekä ohjelmointiin (tietojenkäsittelytiede) että opetukseen (pedagogiikka) liittyvät asiat.

Laajojen ja avointen kysymysten tutkimiseen soveltuu suunnittelun tutkimus (*design research*, myös kehittämistutkimus). Suunnittelun tutkimusta on käytetty esimerkiksi englantilaisessa ScratchMaths-projektissa, jonka konteksti ja aiheet ovat hyvin samanlaisia kuin tässä tutkimuksessa. Projektin tarkoitus oli luoda yksityiskohtaiset ohjelmoinnin opetussuunnitelma ja materiaalit vuosiluokille 5 ja 6 [18]. Lisäksi projektissa järjestettiin täydennyskoulutusta opettajille [18], mutta koulutus vaikuttaa keskittyneen lähinnä opetussuunnitelmaan ja sen aiheisiin.

Suunnittelun tutkimuksen tarkoitus on keskittyä teoriaan ja käytäntöön tarjoten sekä teoreettista tietoa tutkittuun aiheeseen liittyen että tosielämän ratkaisun havaittuun ongelmaan [19], [20]. Tutkimuksessa pyritään muodostamaan käytännön ratkaisu, mikä selvästi edistää käytännön tietoa, mutta myös epäsuorasti teoreettista ymmärrystä yhtenä esimerkkinä siitä, miten suunnitelmat voidaan konkretisoida.

Suunnittelun tutkimus voidaan kuvata kolmivaiheisena [19]. Ensimmäisenä kartoitetaan ongelma ja sen ratkaisulle asetetut vaatimukset. Toisessa vaiheessa ongelmalle suunnitellaan ratkaisu. Kolmannessa vaiheessa suunniteltua ratkaisua ja sen toimivuutta arvioidaan. Vaiheet muodostuvat sykleistä, joiden avulla hiotaan ja parannellaan vaiheen tulosta [20], [21].

2.2.1 Ongelman kartoitus

Ensimmäisessä vaiheessa muodostetaan selitys ongelmasta ja määritellään mahdollisten ratkaisujen vaatimukset [19], [20]. Ongelman, kontekstin ja muiden merkityk-

sellisten asioiden ymmärtämistä tuetaan sisäisen ammattitaidon ja kirjallisuuden avulla saadulla teorialla [19]. Kun ongelma ymmärretään paremmin ja ongelman konteksti selkeytyy, siirrytään hakemaan tietoa laajemmin samankaltaisista ongelmista ja ratkaisuista niihin [19]. Hevner [21] kuvaa ongelman määrittelyä asiaankuuluvuussyklillä (*relevance cycle*), jonka avulla tutkimus sidotaan kontekstiinsa. Kontekstin avulla saadaan määriteltyä sekä tutkimuksen vaatimukset (eli tutkimusongelma ja mahdollisuudet ratkaisuille) että kriteerit tutkimuksen tulosten lopulliselle arvioinnille.

Tämä tutkimus lähti liikkeelle siitä, että haluttiin kehittää alakoulun opettajille suunnattu täydennyskoulutus, jossa on ohjelmoinnin osaamisen lisäksi kiinnitetty myös huomiota siihen, miten ohjelmointia opetetaan. Kontekstia tarkennettiin perusopetuksen opetussuunnitelmien perusteilla: ohjelmointia opetetaan graafisessa ohjelmointiympäristössä pääasiassa osana matematiikkaa ja hieman käsitöissä (robotiikka ja automaatio). Koulutuksessa päätettiin keskittyä matematiikan osuuteen, mikä tarkensi kontekstia myös graafisen ohjelmoinnin osalta. Teknologis-pedagogisen sisältötiedon viitekehyksen perusteella koulutuksessa päätettiin käsitellä graafisen ohjelmoinnin opetuksen sisältöjen lisäksi myös pedagogista sisältötietoa eli sitä, miten ohjelmointia opetetaan. Kun ongelma ja konteksti olivat selkeytyneet, etsittiin niihin liittyvää aiempaa tutkimusta, jonka perusteella lähdettiin sitten kokoamaan ratkaisua määriteltyyn ongelmaan.

2.2.2 Ratkaisun kehitys

Toisessa vaiheessa suunnitellaan ja kehitetään ratkaisu määriteltyyn ongelmaan [19], [20]. Suunnittelussa potentiaalisia ratkaisuja luodaan, tutkitaan ja pohditaan [19] suunnittelusyklissä [21]. Sykli iteroi suunnitelmien, niiden arvioinnin ja palautteen välillä, jotta suunnitelmaa saadaan hiottua eteenpäin.

Tässä tutkimuksessa ratkaisun suunnittelu ja kehittäminen lähtivät liikkeelle

koulutuksen sisältöjen ideoimisella. Koulutuksen tarkoitus on tarjota opettajille tietoa, jota he tarvitsevat, joten nopeasti huomio kiinnittyi erilaisiin ohjelmointiin ja ohjelmoinnin opetukseen liittyviin tutkimuksiin. Koska kontekstina oli alakoulu ja graafinen ohjelmointi, rajattiin tämän tutkimuksen kannalta oleellisimmat tutkimukset niihin, joissa oli erityisesti huomioitu graafinen ohjelmointi.

Luokanopettajan tarvitsemaa tietoa jäsenneltiin teknologis-pedagogisen sisältötiedon viitekehyksen avulla, koska se kuvaa opettajan tarvitsemaa laajaa tietoa opetukseen ja oppimiseen vaikuttavista tekijöistä. Viitekehyksen osa-alueisiin keskityttiin yksi kerrallaan, jotta jokaiselle saatiin muodostettua malli siitä tiedosta, jota siinä tarvitaan.

Kun luokanopettajan tarvitsema tieto oli saatu koottua, alettiin niitä siirtää täydennyskoulutukseen opetettaviksi sisällöiksi. Prosessi sisälsi sisältöjen ideointia, kirjoitusta ja valmistelua, mutta myös muiden kouluttajien kanssa pohtimista ja materiaalien muokkausta. Täydennyskoulutuksessa sisältöjen laajuus jouduttiin sopeuttamaan käytettävissä olevaan aikaan. Lopputuloksena saatiin kuitenkin graafisen ohjelmoinnin täydennyskoulutuskokonaisuus, joka sisältää kaikki aihekokonaisuudet (pidempinä ja tarkempina tai vain lyhyesti), jotka määriteltiin luokanopettajan tarvitsemassa tiedon määritelmässä.

2.2.3 Ratkaisun arviointi

Kolmannessa vaiheessa ratkaisua havainnollistetaan ja arvioidaan [19], [20]. Lisäksi reflektoidaan sekä tutkimuksessa että kehityksessä tapahtunutta (sisältäen teoreettiset syötteet, empiiriset havainnot ja subjektiiviset reaktiot) [19]. Reflektoinnin tarkoitus on saavuttaa teoreettista ymmärrystä [19], kun saatu ratkaisu yhdistetään olemassa olevaan tietoon [20]. Hevner [21] kutsuu tätä tarkkuussykliksi (*rigour cycle*). Sen tarkoitus on reflektoida tutkimusta aiempaan tietoon, jotta voidaan varmistua sen uutuudesta.

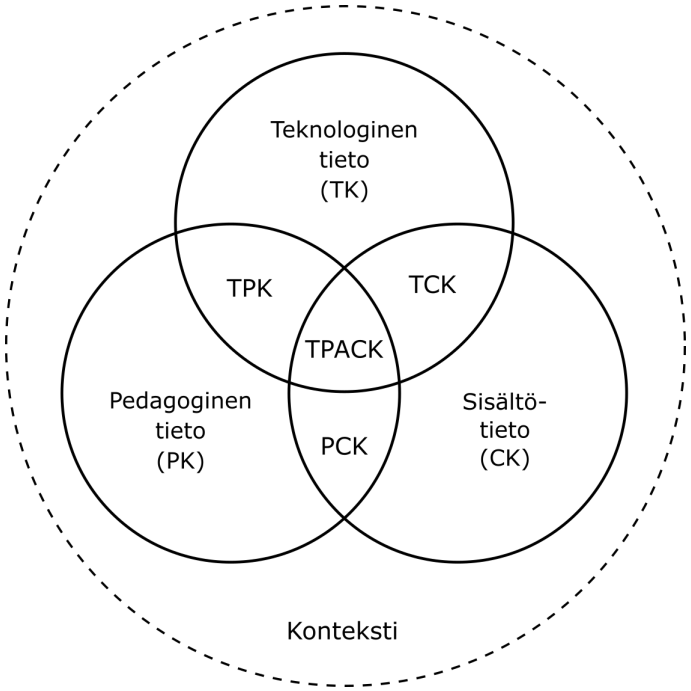
Tässä tutkimuksessa saatua ratkaisua eli luokanopettajan tarvitsemasta tiedosta koottua täydennyskoulutusmallia havainnollistettiin pitämällä koulutus, jossa mallia hyödynnettiin. Täydennyskoulutusmallia ja sen sisältöjä (eli luokanopettajan tarvitsemaa tietoa) arvioitiin sekä koulutukseen osallistuvien tekemän ohjelmoinnillisen ajattelun testin tulosten että osallistujien antaman palautteen perusteella. Ennen varsinaisen arvioinnin aloittamista testin tuloksista ja palautteista poimittiin ne osallistujat, jotka olivat kurssin aluksi antaneet luvan käyttää tuloksia ja palautteita tutkimuksessa anonyymisti. Arviointi tehtiin kokoamalla testin tulokset ja palautteet kaavioiksi, joita tulkittiin. Tulkinnan jälkeen koko koulutusta, testin tuloksia ja osallistujilta saatua palautetta reflektoitiin.

3 Teknologis-pedagoginen sisältötieto tietojenkäsittelytieteessä

Opettajien kyky opettaa jotakin aihetta vaatii monenlaista tietoa. Opetusta ohjaavat opetussuunnitelmat ja sitä rajoittavat ja mahdollistavat opetusresurssit. Opettajan tieto sisällöstä ja pedagogiikasta viimeistelevät opetuksen. Shulman [22] on luonut pohjan pedagogiselle sisältötiedolle, joka yhdistää opettajan tiedot opetettavasta sisällöstä pedagogiseen tietoon. Shulmanin mukaan opettaja tarvitsee tietoa opetettavista sisällöistä, yleisesti pedagogiikasta ja opetussuunnitelmasta, pedagogisesta sisältötiedosta, oppilaista ja heidän ominaisuuksistaan, opetuskontekstista ja opetuksen tavoitteista ja arvoista. Pedagoginen sisältötieto eroaa sisältötiedosta ja pedagogiikasta. Se tarkoittaa sisällön ja pedagogiikan yhdistämistä eli ymmärrystä sisältöihin soveltuvasta pedagogiikasta ja toisaalta pedagogiikan vaikutuksesta opetettaviin sisältöihin.

Kun teknologian käyttö on laajentunut myös opetukseen, on pedagogista sisältötietoa laajennettu. Teknologis-pedagogisen sisältötiedon viitekehys (*Technological, Pedagogical, and Content Knowledge Framework*, TPACK) (kuva 3.1) yhdistää kolme eri tietoaluetta yhdeksi kokonaisuudeksi, jonka tarkoitus on varmistaa, että pedagogiset ja teknologiset valinnat tukevat sisältöjen opetusta ja oppimista.

Viitekehys koostuu kolmesta pääosa-alueesta: teknologinen tieto, pedagoginen tieto ja sisältötieto, mutta sen ydin on niiden yhdistelmässä [23]. Viitekehyksen tar-



Kuva 3.1: Teknologisen, pedagogisen ja sisältötiedon viitekehys

koitus on korostaa sitä, että tehokkain opetus mahdollistuu, kun opettaja hallitsee näiden kolmen pääosa-alueen yhdistelmän eli teknologis-pedagogisen sisältötiedon. Minkään yksittäisen osa-alueen osaaminen ei riitä, vaan opettajan on osattava myös yhdistää osa-alueiden osaaminen yhdeksi kokonaisuudeksi ja ottaa huomioon opetuksen konteksti, jotta opetuksesta tulee mahdollisimman hyvää.

Viitekehystä ovat hyödyntäneet ja tutkineet niin opettajat kuin opetuksen tutkijat monissa eri aihealueissa, mutta tietojenkäsittelytieteessä (*computer science*) vähemmän [24]. Ohjelmoinnin tulkitaan tässä olevan osa tietojenkäsittelytiedettä. Tietojenkäsittelytieteen ja opetuksen yhdistäminen johtaa ainutlaatuiseen tilanteeseen, jossa teknologialla on rooli sekä oppimisen ja opetuksen apuvälineenä (esim. tekstinkäsittelyohjelmat) että opetettavana sisältönä (esim. ohjelmoinnissa ohjelmointiympäristö tai -kieli).

Seuraavaksi käsitellään TPACK tietojenkäsittelytieteen näkökulmasta. Käsittely on tässä jaettu viitekehyksen kolmen pääosa-alueen mukaan kolmeen osaan: sisäl-

tötieto (luku 3.1), pedagoginen tieto (luku 3.2) ja teknologinen tieto (luku 3.3). Pedagoginen sisältötieto, teknologinen sisältötieto ja pedagogis-teknologinen tieto käsitellään alaluvuissa, mutta luvussa 3.4 käsitellään teknologis-pedagogista sisältötietoa tietojenkäsittelytieteessä.

3.1 Sisältötieto

Sisältötieto kuvaa opettajan tietoa aiheesta, jota on tarkoitus oppia ja opettaa. Shulmanin kuvauksen mukaan sisältötieto sisältää tietoa konsepteista, teorioista, ideoista, viitekehyksistä, todistusaineistosta ja vakiintuneista käytännöistä niiden oppimiseen. On tärkeää, että opettaja ymmärtää opettamaansa aihetta syvällisemmin kuin opettaa. [23]

Tietojenkäsittelytieteen opetus on mahdollista integroida muihin oppiaineisiin eli sitä voidaan käyttää apuna tukemaan tarkoituksellista oppimista muissa oppiaineissa tai muiden oppiaineiden aiheita voidaan käyttää kontekstina tietojenkäsittelytieteen opetuksessa. Vivian ja Falkner [24] nimesivätkin kaksi alakategoriaa sisältötiedolle: digitaaliset teknologiat (*Digital Technologies* eli australialaisen opetussuunnitelman tietojenkäsittelytiede) ja muut aiheet. He kuitenkin huomauttavat, että koulutuksen tehtävät ovat todennäköisesti vaikuttaneet vahvasti siihen, että opettajat ovat yhdistäneet tietojenkäsittelytiedettä muihin aiheisiin. Tietojenkäsittelytieteen opetuksessa vaadittavaan sisältötietoon (digitaaliset teknologiat) voidaan siis yhdistää muidenkin oppiaineiden ja aiheiden sisältötietoa (muut aiheet).

3.2 Pedagoginen tieto

Pedagogisella tiedolla tarkoitetaan opettajan syvällisiä tietoja opettamisen ja oppimisen prosesseista, tavoista ja menetelmistä. Siihen sisältyvät koulutustavoitteet, arvot, oppimisen ymmärrys, yleiset luokanhallintataidot, oppijoiden arviointi, ope-

tusmenetelmät ja kohdeyleisön luonne. Lisäksi opettajan pedagogiseen tietoon liittyy ymmärrys oppilaiden tiedon rakentamisesta, taitojen saavuttamisesta, ajattelutapojen kehityksestä ja positiivisista asenteista oppimiseen. Pedagogiseen tietoon kuuluu myös oppilaiden ymmärryksen arvioinnin strategiat. [23] Vivian ja Falkner [24] jakoivat tutkimuksessaan opettajien pedagogisen tiedon kahteen strategiaan: opettamisen tietoon ja oppimisen tietoon. He raportoivat pääasiassa opetus- ja oppimismenetelmiä, jotka nousivat tutkimukseen osallistuneilta opettajilta.

Toinen jaottelu, joka Vivianin ja Falknerin [24] tutkimuksessa tuli ilmi, liittyy opetusmenetelmien opettaja- ja oppilaskeskeisyyteen. Opettajat valitsevat opetukseen todennäköisemmin opettajakeskeisempiä menetelmiä, jos heillä on hyvä itseluottamus opettamastaan aiheesta. Oppilaskeskeisiä opetusmenetelmiä valitsivat ne opettajat, joilla on heikompi itseluottamus opetettavasta aiheesta. Opettajat myös yhdistivät opettaja- ja oppilaskeskeisiä menetelmiä.

Vivianin ja Falknerin [24] tutkimuksessa opettajat käyttivät paljon sellaisia pedagogisia strategioita, joita voitaisi soveltaa missä tahansa aiheessa (kuten ryhmätyöskentely), mutta myös tietojenkäsittelytieteelle ominaisia strategioita (kuten pariohjelmointi ja vuokaavioiden luominen). Näin ollen oppiaine tai käsiteltävä aihe voi vaikuttaa käytettäviin pedagogisiin ratkaisuihin ja siten opettajan pedagogiseen tietoon. On kuitenkin huomattava, että tämä ei tarkoita pedagogisen ja sisältötiedon yhdistämistä vaan kyseessä on pikemminkin kontekstin vaikutus pedagogisiin ratkaisuihin.

Pedagoginen sisältötieto kuvaa pedagogista tietoa, joka soveltuu tietyn sisällön opettamiseen. Keskeistä on aihealueen sisällön muuttaminen opetettavaan muotoon. Shulmanin mukaan tämä muutos tapahtuu, kun opettaja tulkitsee sisältöä, löytää useita tapoja sen esittämiseen ja mukauttaa opetusmateriaalit sopimaan oppijoiden erilaisten käsitysten ja aiemman tiedon kanssa. Tehokkaan opetuksen kannalta on tärkeää tietää yleisistä solmukohdista ja osata yhdistää eri sisältölähtöisiä ideoita,

oppilaiden aikaisempaa tietoa ja vaihtoehtoisia opetusmenetelmiä. Lisäksi saman idean tai ongelman vaihtoehtoisien tarkastelutapojen tutkiminen ja etsiminen tuovat joustavuutta opetukseen. [23]

3.3 Teknologinen tieto

Teknologinen tieto ei ole niin selvärajainen kuin kaksi muuta mallin pääosa-aluetta, koska teknologia kehittyy niin nopealla tahdilla. Sen määrittely onkin siksi hankalaa: mikä tahansa määritelmä vanhentuisi hyvin nopeasti [23]. On kuitenkin olemassa yleisiä tapoja työskennellä teknologian kanssa [23], jotka eivät ole alttiita muutokselle ja jotka pätevät kaikkiin teknologisiin työkaluihin ja resursseihin.

TPACK-viitekehyksessä tietotekniikan ymmärrys määritellään laajaksi niin, että sitä voi hyödyntää järkevästi työssä ja arkielämässä. Tärkeää on ymmärtää, milloin tietotekniikasta on hyötyä tai haittaa tavoitteiden saavuttamisessa, ja miten voidaan sopeutua tietotekniikan muutoksiin. Tavoiteltu ymmärrys on siis laajempaa kuin perinteinen tietokoneiden ymmärrys, joka mahdollistaa erilaisten tietotekniikkaa vaativien tehtävien suorittamisen ja suoritustapojen kehittämisen. Teknologinen tieto onkin koko ajan kehittyvä, avoin vuorovaikutus teknologian kanssa. [23]

Tietotekniikan alalla teknologialla on suuri merkitys, koska se voi sekä tukea opittavan sisällön esittämistä että olla itse sisällön keskipiste [24]. Vivian ja Falkner [24] tunnistivat tutkimuksessaan kolme alakategoriaa teknologiselle tiedolle: tietojenkäsittelylle ominainen teknologia, yleinen teknologia ja yleiset materiaalit. Tietojenkäsittelyn opetuskäyttöön tarkoitetut laitteet ja ohjelmistot kuuluvat tietojenkäsittelylle ominaiseen teknologiaan. Opetuksessa voidaan käyttää myös sellaista teknologiaa, joka tukee opetusta, kuten videot, digitaaliset pelit, laitteet ja sovellukset. Opetusta tukevan teknologian Vivian ja Falkner ovat luokitelleet yleiseksi teknologiaksi ja se vastaa TPACK-viitekehyksessä määriteltyä teknologista tietoa. Vaikka teknologinen tieto viittaa alun perin digitaalisten teknologioiden käyttöön,

opettajat Vivianin ja Falknerin tutkimuksessa viittasivat myös muunlaisiin opetusmateriaaleihin ja työkaluihin, jotka eivät ole digitaalisia. Näitä ovat esimerkiksi käsinkosketeltavat opetuksen apuvälineet (kuten ohjelmointikortit).

Teknologisen tiedon yhdistäminen sisältötiedon kanssa tarkoittaa ymmärrystä teknologian vaikutuksesta opetettavaan ja opittavaan sisältöön [23]. Teknologisen sisältötiedon avulla opetustarkoitukseen sopivia teknologioita voidaan käyttää ja kehittää. Teknologioiden valikoima voi laajentaa ja rajoittaa opetettavissa olevia sisältöjä ja opetettavat sisällöt voivat rajoittaa hyödynnettäviä teknologioita. Teknologia voi myös tarjota uusia, erilaisia esitystapoja ja joustavuutta erilaisten esitystapojen välillä navigoimiseen esimerkiksi kuvien, videoiden ja ohjelmistojen avulla.

Teknologinen sisältötieto tarkoittaakin ymmärrystä tavoista, joilla teknologia ja sisältö vaikuttavat toisiinsa [23]. Opettajan on hallittava enemmän kuin opetettavat aiheet, jotta hän osaa hyödyntää teknologioiden mahdollisuuksia tuoda aiheita eri tavoin esille. Opettajan on ymmärrettävä, mitkä teknologiat sopivat parhaiten opetettavaan sisältöön ja miten sisältö ja teknologia vaikuttavat toisiinsa.

Teknologis-pedagoginen tieto kuvaa, miten opettamista ja oppimista voidaan muuttaa tavalla, jolla teknologiaa käytetään [23]. Siinä teknologiset mahdollisuudet yhdistetään pedagogisiin teorioihin ja strategioihin. Teknologis-pedagoginen tieto on ymmärrystä teknologian luomista mahdollisuuksista, rajoituksista ja käyttötarkoituksista opetuksessa. Teknologis-pedagogisen tiedon kehittäminen mahdollisuuksien, rajoituksien ja käyttökontekstin ymmärtämisen. On tärkeää ymmärtää, miten eri tavoin teknologiaa voi hyödyntää erilaisissa konteksteissa ja tarkoituksissa. Monia suosittuja ohjelmia ei ole kehitetty opetuskäyttöön vaan esimerkiksi työympäristöön, viihteeksi, viestintään tai sosiaaliseen kanssakäymiseen, joten pedagogis-teknologista tietoa tarvitaan niiden soveltamiseen opetuksessa. Opettajien on pystyttävä katsomaan teknologioiden tavallisimpien käyttötapojen taakse nähdäkseen niiden mahdollisuuksia opetuskäytössä. Siksi teknologinen ja pedagoginen tieto vaatiikin luovaa

ja ennakkoluulotonta suhtautumista teknologian hyödyntämiseen oppilaiden oppimisen ja ymmärryksen edistämiseksi.

3.4 Teknologis-pedagoginen sisältötieto

Teknologis-pedagoginen sisältötieto on enemmän kuin sen kolme pääosa-aluetta yksinään. TPACK tarkoittaa ymmärrystä, joka tulee esiin sisältö-, pedagogisen ja teknologisen tiedon välisestä vuorovaikutuksesta: TPACK on siis enemmän kuin osa-alueiden summa. Pohjimmiltaan TPACK on lähtökohta tehokkaalle opetukselle, jossa hyödynnetään teknologiaa. Se vaatii

- ymmärrystä siitä, miten konseptit esitetään teknologian avulla,
- pedagogisia menetelmiä, jotka käyttävät teknologioita rakentavalla tavalla sisällön opettamiseksi,
- tietoa siitä, mikä tekee konsepteista vaikeita tai helppoja oppia,
- tietoa teknologian hyödyntämisestä oppilaiden kohtaamien ongelmien korjaamisessa,
- tietoa oppilaiden aikaisemmasta tiedosta ja
- tietoa siitä, miten uusia tietokokonaisuuksia voidaan luoda tai vanhoja vahvistaa teknologiaa hyödyntämällä. [23]

TPACK-viitekehys esittääkin, että sisältö, pedagogiikka, teknologia ja opettamisen/oppimisen konteksteilla on kaikilla roolit sekä yksittäin että yhdessä.

Tietojenkäsittelytieteen kontekstissa TPACK-viitekehystä (kuva 3.1) voidaan tarkentaa Vivianin ja Falknerin tutkimuksen tuloksilla (taulukko 3.1). Vivian ja Falkner [24] ovat sitä mieltä, että TPACK tukee opettajien osallistuneisuutta, kun tietojenkäsittelytieteitä tuodaan perusopetukseen, koska opettajat voivat hyödyntää heillä jo olevaa pedagogista ja teknologista tietoa uusiin sisältöihin.

Taulukko 3.1: TPACK-viitekehys tietojenkäsittelytieteessä (TKT)

Teknologinen tieto	TKT:lle ominaiset teknologiat
	Yleiset teknologiat
	Yleiset materiaalit
Pedagoginen tieto	TKT:lle ominaiset pedagogiset strategiat
	Yleiset pedagogiset strategiat
Sisältötieto	Digitaaliset teknologiat
	Muut aiheet

4 Luokanopettajan graafisen ohjelmoinnin tieto

Tässä luvussa määritellään tieto, jonka luokanopettaja tarvitsee voidakseen opettaa graafista ohjelmointia alakoulussa eli vastataan tutkimuksen ensimmäiseen tutkimuskysymykseen. Graafista ohjelmointia opetetaan vuosiluokilla 3–6 sekä matematiikan että käsitöiden oppiaineissa. Tieto, jonka luokanopettaja tarvitsee graafisen ohjelmoinnin opetusta varten, esitellään tässä TPACK-viitekehyksen avulla.

Tutkimuksessa keskitytään ainoastaan graafiseen ohjelmointiin ja sen opettamiseen, joten luokanopettajan tarvitseman tiedon määrittely aloitetaan graafisen ohjelmoinnin sisältötiedosta luvussa 4.1. Sisältöjen opetusta varten luokanopettaja tarvitsee oman (yleisen) pedagogisen tietonsa lisäksi myös pedagogista sisältötietoa, joka kuvataan tässä luvussa 4.2. Opetusta tukemaan opettaja tarvitsee myös tietoa graafiseen ohjelmointiin liittyvästä teknologiasta, joka käsitellään luvussa 4.3. Graafisen ohjelmoinnin teknologis-pedagoginen sisältötieto kuvataan luvussa 4.4.

4.1 Sisältötieto

Luokanopettajan tarvitsemaa sisältötietoa määrittelevät Opetushallituksen opetussuunnitelman perusteet. Perusteissa on määritelty, että ohjelmointia harjoitellaan matematiikan ja käsityön oppiaineissa, joista tässä tutkimuksessa käsitellään vain matematiikan osuus. Matematiikassa opetuksen tavoitteena on ”innostaa oppilasta

laatimaan toimintaohjeita tietokoneohjelmina graafisessa ohjelmointiympäristössä" [13]. Ohjelmia sekä suunnitellaan että toteutetaan. Alakoulun loputtua hyvä (tai arvosanan 8) osaaminen tarkoittaa, että "oppilas osaa ohjelmoida toimivan ohjelman graafisessa ohjelmointiympäristössä" [13].

Brennan ja Resnick [25] ovat määritelleet graafiselle ohjelmoinnille mallin, joka jäsentää ohjelmoinnissa tarvittavia tietoja ja taitoja. Malli on kehitetty lähinnä oppilaita seuraten, joten se sisältää alakouluikäisille oppilaille suunnattuja aiheita. Sen vuoksi malli soveltuu hyvin juuri luokanopettajien tiedon määrittelyyn. Mallia on kuitenkin hieman täydennettävä, koska opettajien on hyvä tietää opettamastaan aiheesta enemmän kuin mitä on opetettava.

Graafisen ohjelmoinnin malli on kehitetty lohkopohjaiselle ohjelmointikielelle graafisessa ohjelmointiympäristössä Scratchissä. Malli soveltuu kuitenkin myös muille ohjelmointikielille – niin lohkopohjaisille kuin tekstipohjaisillekin [26]. Ohjelmointikielet ovat perinteisesti tekstipohjaisia eli niillä ohjelmoidaan kirjoittaen. Lohkopohjaisissa ohjelmointikielissä ohjelmointi tapahtuu raahaamalla ohjelmointiohjeita eli lohkoja yhteen. Ohjelmointiympäristön graafisuus tarkoittaa erilaisia visuaalisia elementtejä. Näitä käsitellään tarkemmin luvussa 4.3.1.

Malli koostuu ohjelmoinnin konsepteista, ohjelmointikäytännöistä ja ohjelmoinnillisista näkökulmista. Ohjelmoinnin konseptit ovat ohjelmoinnin tärkeimpiä käsitteitä. Ohjelmointikäytännöt kuvaavat ohjelmointiprosessia ja ohjelmoinnilliset näkökulmat sellaisia näkökulmia, joita ohjelmointi avaa. Konseptien, käytäntöjen ja näkökulmien avulla voidaan jäsentää sitä sisältötietoa, jota luokanopettaja tarvitsee.

4.1.1 Ohjelmoinnin konseptit

Graafisen ohjelmoinnin mallissa ohjelmoinnin konseptit ovat ohjelmoinnin tärkeimpiä käsitteitä ja rakenteita [25]. Ne ovat peruselementtejä, jotka löytyvät monista

(teksti- ja lohkopohjaisista) ohjelmointikielistä. Konseptit ovat

- peräkkäisyys (*sequences*),
- samanaikaisuus (*parallelism*),
- tapahtumat (*events*),
- ehdot eli ehtolauseet (*conditionals*),
- silmukat eli toistolauseet (*loops*),
- operaattorit (*operators*) ja
- data (*data*).

Peräkkäisyydellä tarkoitetaan sitä, että ohjelmointiohjeet eli lauseet suoritetaan yksi lause kerrallaan siinä järjestyksessä kuin ne on kirjoitettu ohjelmakoodiin. Samanaikaisuus tarkoittaa sitä, että kahta (tai useampaa) peräkkäisrakennetta suoritetaan yhtä aikaa. Samanaikaisuuden vaikutelma voidaan saavuttaa myös rinnakkaisuudella (*concurrency*), jossa kahta (tai useampaa) peräkkäisrakennetta suoritetaan vuorotellen. Tällöin käyttäjästä vaikuttaa siltä, että peräkkäisrakenteita suoritettaisi samanaikaisesti, koska tietokone pystyy suorittamaan peräkkäisrakenteet niin nopeasti.

Ohjelmakoodin suoritus alkaa, kun ohjelma käynnistetään, mutta jotkin koodin osat voivat kuitenkin vaatia lisäksi jonkin tapahtuman, jotta niiden suoritus aloitetaan. Tapahtumat voivat myös lopettaa tai keskeyttää suorituksen tai muuttaa sitä jollakin tavalla. Tapahtumat mahdollistavat ohjelman interaktiivisuuden.

Ehtolauseiden avulla ohjelmoinnin suoritus voidaan haarauttaa jonkin ehdon perusteella. Ehtolauseissa tarkistetaan, onko jokin ehto totta vai ei ja sen perusteella joko suoritetaan tai ei suoriteta jokin tietty peräkkäisrakenne. Toisaalta jos ehto ei ole totta, voidaan suorittaa myös vaihtoehtoinen peräkkäisrakenne. Ehtolauseita voi

myös ketjuttaa eli jos ensimmäinen ehto ei ole totta, voidaan siirtyä tarkistamaan jokin toinen ehto.

Silmukoiden eli toistolauseiden avulla samaa peräkkäisrakennetta voidaan suorittaa monta kertaa peräkkäin ilman, että sitä täytyy kirjoittaa ohjelmakoodiin monta kertaa. Toistojen määräksi voidaan asettaa jokin tietty luku tai silmukkaa voidaan toistaa niin kauan, kunnes jokin ehto muuttuu todeksi (tai epätodeksi).

Operaattoreiden avulla voidaan muodostaa matemaattisia (esim. yhteen- ja vähennyslaskut), loogisia (esim. suurempi kuin ja pienempi kuin -vertailut) ja merkkijonolausekkeita (esim. merkkijonojen yhdistäminen). Operaattorit (esim. plusmerkki) siis määrittelevät, mitä operandeille (esim. kaksi lukua) tehdään. Tuloksena on lausekkeet, joita voidaan käyttää ohjelmointiohjeiden eli lauseiden osana. Esimerkiksi ehtolauseen ehto voi olla lauseke, jossa vertaillaan kahden luvun suuruutta.

Datan konsepti sisältää tiedon säilyttämisen, hakemisen ja päivittämisen. Ohjelmoinnissa tieto on aina jonkin tyyppistä (esim. kokonaisluku- tai merkkijonotyyppistä). Ohjelmointikielestä riippuen tyypit voivat näkyä tai olla näkymättä ohjelmakoodissa. Operaatioiden toimimiseksi on ohjelmoijan kuitenkin aina tiedettävä, minkä tyyppisillä arvoilla operaatiota voidaan suorittaa. Joissain tapauksissa väärän tyyppinen arvo lausekkeessa voi keskeyttää ohjelman suorituksen tai estää suorituksen aloittamisen kokonaan.

Dataa säilytetään ohjelmoidessa muuttujissa eli data-arvolle annetaan ohjelmakoodissa nimi tai tunniste, jonka avulla dataa käytetään ohjelmakoodissa. Arvo tallennetaan tietokoneen muistiin ja sitä voidaan myöhemmin käyttää ohjelmassa muuttujan nimellä. Muuttujiin voidaan asettaa vain yksi arvo kerrallaan, mutta arvo voi muuttua ohjelman aikana.

Dataa voidaan säilyttää myös tietorakenteissa, joihin voi tallentaa useampia arvoja. Tietorakenteelle annetaan yksi nimi tai tunniste ja sen sisältämiä arvoja voidaan käyttää tietorakenteesta riippuen eri tavoin. Tietorakenteita ovat esimerkiksi

taulukot, listat ja puurakenteet. Yksi tärkeimpiä tietorakenteita on lista, jonka alkioita eli yksittäisiä arvoja voi lisätä, poistaa ja muuttaa. Alkioihin viitataan indekseillä eli järjestysnumeroilla, jotka ohjelmoinnissa alkavat usein nollasta.

Konsepteihin voi katsoa kuuluvan myös syöte ja tuloste (*input and output*) [27]. Ne kuvaavat sitä, että ohjelmoimalla tietystä syötteestä seuraa tietty tuloste. Käyttäjä voi ohjailla ohjelman toimintaa syötteillä ja saa lopputuloksena tulosteita syötteiden perusteella. Ohjelmoinnissa on tärkeää huolehtia syötteiden testauksesta, jotta tulosteet ovat oikein.

Zhang ja Nouri [27] korostavat eroa syötteen ja tapahtuman välillä. Tapahtumat ilmentävät syy–seuraus-suhdetta, mutta syöte on itsenäinen ohjelmoinnillisen prosessin elementti. Syötteet ovat kuitenkin läheisessä suhteessa tapahtumiin: syöte, kuten hiiren klikkaus, laukaisee tapahtuman eli ilman syötettä tapahtuma ei ole mahdollinen.

Alakoulun oppilaille nämä käsitteet voivat riittää, mutta kuten aiemmin on todettu, on opettajan hyvä ymmärtää opettamaansa aihetta syvällisemmin kuin opettaa. Siksi luokanopettajan tietoon konsepteista voidaan lisätä vielä aliohjelmat ja kirjastot.

Aliohjelmien avulla samaa peräkkäisrakennetta voidaan suorittaa monta kertaa kirjoittamatta sitä aina uudelleen toistolauseiden tapaan. Aliohjelmat ovat kuitenkin pääohjelmasta erillisiä, joten niitä voidaan kutsua eli käyttää yhdellä lauseella monessa kohtaa ohjelmakoodia. Aliohjelmat siis eroavat toistolauseista, joita voidaan käyttää vain siinä kohtaa peräkkäisrakennetta, johon se on kirjoitettu. Aliohjelmien suoritusta voidaan ohjata parametreilla eli syötteillä ja lisäksi ne voivat palauttaa arvoja siihen kohtaan, jossa niitä on kutsuttu. Aliohjelmat siis muistuttavat matematiikan funktioita.

Kirjastot ovat ohjelmaan lisättäviä paketteja, jotka laajentavat ohjelmointikielen perusrakenteita valmiiksi toteutetuilla ominaisuuksilla. Kirjastot voivat olla osa

ohjelmointikieltä tai toisten ohjelmoijien luomia. Kun kirjastot on avattu, niiden ominaisuuksia voi hyödyntää ohjelmassa ilman, että niitä pitää toteuttaa uudelleen.

Opetussuunnitelman perusteet eivät määrittele, mitä konsepteja on opetettava. Kaikkien näiden konseptien osaaminen on kuitenkin hyödyllistä luokanopettajalle, vaikka niitä ei suoraan opettaisikaan. Edistyneempiä konsepteja voi hyödyntää esimerkiksi tehtävien tekemisessä ja oppilaiden kysymyksiin vastatessa (jos konseptit ovat mahdollisia käytetyssä ohjelmointiympäristössä, on mahdollista, että oppilaat löytävät ne ja kyselevät niistä).

4.1.2 Ohjelmoinnin käytännöt

Ohjelmoiminen ei ole täysin lineaarinen prosessi, jossa ohjelma kirjoitetaan kokonaisuudessaan, minkä jälkeen se olisi täysin toimiva, valmis ohjelma. Ohjelmia joudutaan usein muokkaamaan, jotta ne johtaisivat haluttuun lopputulokseen. Lisäksi ohjelmia voidaan täydentää uusilla ominaisuuksilla tai niissä olevia virheitä voidaan korjata.

Graafisen ohjelmoinnin mallissa ohjelmoinnin käytännöt kuvaavat ohjelmointiprosessia [25]. Brennan ja Resnick sanovat, että käytännöt ovat se, *miten* opitaan (toisin kuin konseptit, jotka ovat sitä, *mitä* opitaan). Ohjelmointiprosessin hallitseminen eli ohjelmointitaidon kehittyminen vaatii kuitenkin usein harjoittelua. Siksi onkin ehdotettu, että ohjelmointikielen (tai ohjelmoinnin konseptien) opettamisen sijaan tai sen lisäksi opetettaisi ohjelmointiprosessia [28].

Ohjelmoinnin käytännöt ovat

- inkrementaalisuus ja iteratiivisuus (*being incremental and iterative*),
- testaaminen ja debuggaaminen (*testing and debugging*),
- uudelleen käyttäminen ja yhdisteleminen (*reusing and remixing*),
- abstrahointi ja modularisointi (*abstracting and modularizing*) [25],

- ennustava ajattelu (*predictive thinking*),
- koodin lukeminen, tulkkaaminen ja kommunikoiminen (*reading, interpreting and communicating code*) ja
- multimedian käyttö (*multimodal design*) [27].

Inkrementaalisuus ja iteratiivisuus kuvaavat sitä, että ohjelmia täydennetään, muokataan ja korjataan yhä uudelleen ja uudelleen. Ohjelmia on testattava, jotta voidaan varmistua niiden toiminnasta. Jos ohjelma ei toimi, kuten sen pitäisi täytyy ongelma etsiä ja korjata (siis debugata). Ohjelmoidessa hyödynnetään usein muiden kirjoittamaa koodia eli käytetään aiemmin luotuja koodeja uudelleen yhdistellen niitä toisiinsa täydennellen uudella koodilla. Ohjelmointi vaatii myös ongelman tarkastelua eri tasoilta eli abstrahointia ja ongelman pilkkomista osiin eli modularisointia.

Ennustavalla ajattelulla viitataan siihen, että lopputuloksia ennustetaan ennen ohjelman suoritusta, minkä jälkeen ennustusta voidaan verrata todelliseen lopputulokseen. Koodin lukeminen ja tulkkaaminen ovat tärkeitä taitoja etenkin virheiden debuggaamisessa. Niiden lisäksi ideoiden ilmaiseminen ohjelmoinnillisilla käsitteillä on tärkeää pari- ja ryhmätyöskentelyssä, joita ammattiohjelmoijatkin hyödyntävät.

Multimedian käytöllä viitataan erilaisten medioiden (kuten äänien, kuvien ja toimintojen) yhdistämiseen ohjelmassa. Tämä tulee esille etenkin Scratchissä (johon graafisen ohjelmoinnin malli perustuu), koska siinä taustakuvilla ja hahmoilla on suuri merkitys. Taustakuvia ja hahmoja käytetään ohjelmassa monella tapaa, ja niitä on lisäksi mahdollista muokata ja jopa luoda itse.

Käytännöillä siis kuvataan ohjelmointiprosessia. Luokanopettajan on hyvä ymmärtää, millainen ohjelmointiprosessi on, jotta hän voi tukea oppilaita prosessin aikana. Lisäksi ohjelmointiprosessin ymmärtäminen auttaa opettajan oman ohjelmoinnin osaamisessa ja sitä kautta esimerkiksi sopivien tehtävien valinnassa ja tehtävien laatimisessa.

4.1.3 Ohjelmoinnilliset näkökulmat

Graafisen ohjelmoinnin malliin kuuluvat myös ohjelmoinnilliset näkökulmat, joilla viitataan ohjelmoinnin seurauksena oppilaiden ajattelussa ja käytöksessä tapahtuneisiin muutoksiin [25]. Näkökulmat ovat

- ilmaiseminen (*expressing*),
- yhteenkuuluvuus (*connecting*),
- kiinnostuminen (*questioning*) [25] ja
- ihmisen ja tietokoneen vuorovaikutus (*user interaction*) [27].

Muutos ilmaisussa koskee teknologian roolia: ohjelmoinnin avulla teknologiasta tulee käyttövälineen lisäksi myös itseilmaisun väline. Yhteenkuuluvuus korostaa oppimisen sosiaalisuutta ja luovuutta: luominen toisille ja toisten kanssa on arvokasta ja oppilaat kokevat, että yhdessä pystytään asioihin, joita on hankala saavuttaa yksin. Kiinnostumisen näkökulma liittyy oppilaiden suhteeseen teknologian kanssa: he kiinnostuvat teknologiasta ja alkavat kysellä siitä. Ihmisen ja tietokoneen vuorovaikutus on hyvin tärkeä osa ohjelmointia. Kun ihmisen ja tietokoneen vuorovaikutus otetaan huomioon ohjelman suunnittelussa, ohjelmista voidaan saada intuitiivisia ja käyttäjäystävällisiä ja niiden saavutettavuutta voidaan parantaa.

Yksittäisten ohjelmien kohdalla voidaan puhua myös ohjelman ymmärryksestä (*program comprehension*) eli ohjelmasta muodostetusta mentaalisesta mallista [29]. Ohjelman ymmärryksellä tarkoitetaan sitä, miten hyvin ohjelma, sen toiminta ja tarkoitukset ymmärretään. Ymmärrys on hyvin yksilöllistä, mutta sen muodostumista tukemalla voidaan tukea myös ohjelmoinnillisen näkökulmien kehittymistä.

Ohjelman ymmärryksen malli (*The Block Model*, tässä suomennettu ohjelman ymmärryksen malliksi, jotta nimi ei mene sekaisin lohkopohjaisen ohjelmoinnin kanssa) kuvaa yksittäisen ohjelman ymmärryksen alueita [30]. Ohjelman ymmärrys muo-

Taulukko 4.1: Ohjelman ymmärryksen malli

Makrotaso	Koko ohjelman koodi	Koko ohjelman toiminta suorituksessa	Ohjelman tarkoitus ja tavoitteet sen kontekstissa
Suhteet	Viittaukset koodissa eli esim. datan haku ja aliohjelmakutsut)	Viittausten sarjat suorituksessa eli esim. suorituksen siirtyminen aliohjelmasta toiseen	Alitavoitteiden liittyminen tavoitteisiin ja toiminnan saavuttaminen alitoiminnoilla
Lausekokonaisuudet	Kiinnostuksen kohteena olevien lausekokonaisuuksien koodi	Lausekokonaisuuden toiminta suorituksessa	Lausekokonaisuuden tarkoitus ja (ali)tavoite koko ohjelman kontekstissa
Atomit	Ohjelmointikielen elementit eli lausekkeet ja lauseet	Atomien toiminta suorituksessa	Atomien tarkoitus
	Koodi	**Suoritus**	**Tarkoitus**
Kaksinaisuus	***Rakenne***		***Tarkoitus***

dostuu koko ohjelman ja sen kaikkien pienempien osien ymmärryksestä [31] (katso kuva 4.1). Osia ovat atomit, lausekokonaisuudet, suhteet ja makrotaso.

Atomeilla tarkoitetaan yksittäisiä ohjelmointikielen elementtejä eli lauseita. Yksi lause kuvaa yhden toiminnon, jonka tietokone ohjelman suorituksessa suorittaa. Lausekokonaisuuksilla tarkoitetaan syntaktisesti tai semanttisesti yhtenäisiä yksiköitä. Syntaktisia lausekokonaisuuksia ovat esimerkiksi ehto- ja toistolauseiden muodostamat kokonaisuudet ja semanttisia esimerkiksi lauseet kahden muuttujan arvon vaihtamiseksi. Suhteilla tarkoitetaan viittauksia eli esimerkiksi datan hakemista muistista tai aliohjelmakutsuja. Makrotasolla tarkastelun kohteena on kokonainen ohjelma.

Osan ymmärrys voidaan jakaa sen rakenteen ja tarkoituksen ymmärtämiseen (kaksinaisuus). Rakenteeseen kuuluvat sekä osan ohjelmakoodi staattisena kokonaisuutena että sen toiminta suorituksessa. Jokaisella osalla on ohjelmassa myös oma tarkoituksensa ja tavoitteensa. Tarkoituksella viitataan siihen, minkälaisen toiminnon osa toteuttaa. Tavoitteet ovat ymmärrettävissä vasta koko ohjelman kontekstissa: ne kuvaavat syitä sille, miksi osa on ohjelmassa. Makrotasolla tavoitteet kuvaavat syitä sille, miksi koko ohjelma on toteutettu.

Todellinen ohjelman ymmärrys saavutetaan vasta, kun kaikki ohjelman ymmärryksen mallin alueet eli taulukon 4.1 solut on ymmärretty. Toisin sanoen mitä useamman alueen ymmärtää, sitä lähempänä on koko ohjelman ymmärrystä.

4.1.4 Yhteenveto

Luokanopettajan tarvitsema sisältötieto koostuu sekä ohjelmoinnin konsepteista, ohjelmointikäytännöistä että ohjelmoinnillisista näkökulmista ja ohjelman ymmärryksestä. Graafisen ohjelmoinnin malli on kokonaisuudessaan kuvattu taulukossa 4.2. Mallin lisäksi ohjelman ymmärryksen tunteminen auttaa opettajaa opetuksessa ja arvioinnissa.

Taulukko 4.2: Graafisen ohjelmoinnin malli eli ohjelmoinnin tiedot ja taidot

Konseptit	Peräkkäisyys
	Silmukat
	Samanaikaisuus
	Tapahtumat
	Ehdot
	Operaattorit
	Data
	Syöte/tuloste
Käytännöt	Inkrementaalisuus ja iteratiivisuus
	Testaus ja debuggaus
	Uudelleen käyttäminen ja yhdisteleminen
	Abstrahointi ja modularisointi
	Ennustava ajattelu
	Koodin lukeminen, tulkkaaminen ja kommunikoiminen
	Multimedian käyttö
Näkökulmat	Ilmaiseminen
	Yhteenkuuluvuus
	Kyseleminen
	Ihmisen ja tietokoneen vuorovaikutus

Ohjelmointiin liittyy siis muutakin kuin koodin kirjoittaminen ohjelmointirakenteita käyttäen. Ohjelmointia opettavan on tiedettävä ohjelmointiprosessin epälineaarisuudesta ja prosessiin liittyvistä haasteista. Opettajan on opetettava ohjelmoinnin rakenteiden lisäksi myös ohjelmointia toimintana, mikä saattaa olla haastavaa. Lisäksi tieto näkökulmista ja niissä tapahtuvista muutoksista auttaa opettajaa huomaamaan, milloin oppilas on oppinut ajattelemaan teknologiaa uusista näkökulmista. Näin ollen esimerkiksi oppilaiden taitojen arviointi ja sopivien tehtävien valitseminen helpottuvat.

Koska ohjelmoinnin opetus on osa matematiikan oppiainetta, on luokanopettajan hallittava myös matematiikan sisältötieto [26]. Graafista ohjelmointia on hyödynnetty matematiikan opetuksessa mm. ScratchMaths-projektissa [32]. Projektiin liittyvissä tutkimuksissa on huomattu, että matematiikan opettaminen ohjelmoinnin avulla tukee ohjelmoinnin oppimista heikentämättä matematiikan oppimista. Opettajat ovat kokeneet graafisen ohjelmoinnin hyödyllisenä ja tehokkaana työkaluna matematiikan opetuksessa [26]. Vaarana kuitenkin on, että oppilaan kognitiivinen kuorma kasvaa liian suureksi, jos samaan aikaan on tarkoitus oppia sekä ohjelmointia että matematiikan käsitteitä.

4.2 Pedagoginen tieto

Parhaimpien oppimistulosten varmistamiseksi opettajan on tiedettävä oppimiseen ja opettamiseen liittyvistä prosesseista. Tietojenkäsittelytieteisiin liittyvässä opetuksessa on riskinä, että opetus suuntautuu tietyn teknologian opetukseen, eikä teknologian käyttämien periaatteiden opetukseen [33]. Tätä Papert [34] kutsui jo 1980-luvulla englanninkielisellä termillä *technocentrism*. Papert oli sitä mieltä, että opetuksessa keskityttiin ja luotettiin liikaa teknologiaan: siihen, että opetetaan ainoastaan tietyn teknologian käyttöä ja että teknologian käyttö opettaisi samalla sen toiminnan periaatteita. Papertin mielestä opetuksessa pitäisi keskittyä enemmän

siihen, mitä ihminen voi teknologian avulla tehdä.

Brennanin [35] mukaan vielä 2010-luvullakin opetus on hyvin teknologiakeskeistä ja hän argumentoikin konstruktionistisemman (*constructionism*) opetuksen puolesta. Konstruktionismissa on kyse tehokkaan oppimiskokemuksen saavuttamisesta aktiivisella luomisella. Luomisen kohteeksi valitaan asioita, jotka ovat henkilökohtaisesti tai sosiaalisesti mielekkäitä. Niitä kehitetään muiden kanssa vuorovaikutuksessa niin yleisönä, kollegana tai ohjaajana. Oppiminen nähdään siis tapahtumana, jossa oppija on aktiivisessa roolissa ja oppijan omaa ajattelua tuetaan.

Brennan [35] luettelee neljä konstruktionistiselle oppimisympäristölle keskeistä aktiviteettia:

- suunnittelu (*designing*),
- personalisointi (*personalizing*),
- jakaminen (*sharing*) ja
- reflektointi (*reflecting*).

Suunnittelulla viitataan sellaisiin oppimistapahtumiin, jotka tukevat iteratiivista ajattelua, ongelmanratkaisua ja kriittistä luovuutta ja niiden kehittämistä. Suunnittelu edellyttää kykyä tunnistaa rajoituksia ja neuvotella niistä ja selventää ja hallita monitulkintaisuutta. Ohjelmointi on jo itsessään iteratiivinen prosessi, joka vaatii ongelmanratkaisua ohjelmoinnin keinoin, mutta sitä voidaan tukea tehtävillä, joissa vaaditaan iteratiivisuutta. Esimerkiksi tehtävät, joissa oppilaat jatkokehittävät aiemmin ohjelmoimaansa ohjelmaa, muokkaavat sitä ja lisäävät siihen uusia ominaisuuksia.

Personalisointi konstruktionistisena tavoitteena tarkoittaa oppimistapahtuman monitasoista yksilöimistä, jotta yksilön kiinnostus säilytetään. Personalisointia voidaan tukea laajemmilla tehtävänannoilla, joissa oppilaat voivat käyttää omaa luovuuttaan esimerkiksi ohjelman aiheen valitsemisessa.

Oppiminen on kuitenkin myös sosiaalinen tapahtuma, joten mahdollisuus vuorovaikutukseen tukee oppimista. Vuorovaikutusta tukevat esimerkiksi keskustelut ideoista, ohjelmista, ongelmista ja ratkaisuista. Lisäksi ohjelmoinnissa voidaan hyödyntää yhteistyötä esimerkiksi pariohjelmoinnilla, jota ohjelmoinnin ammattilaisetkin hyödyntävät. Opettajan on kuitenkin tällöin huomioitava oppilaiden taitoerot, jotta voidaan varmistua siitä, että kaikki pääsevät ohjelmoimaan ja oppimaan ohjelmointia.

Reflektiolla tarkoitetaan sellaisia tehtäviä, jotka kannustavat oppilaita pohtimaan omaa oppimistaan, ajatteluaan ja tietämistään. Tällaisilla metakognitiivisilla taidoilla on tärkeä osa oppimisessa ja kognitiivisissa prosesseissa. Siispä sen lisäksi että ohjelmoidaan, voi esimerkiksi valmiista ohjelmasta ja sen ohjelmointiprosessista tehdä kysymystehtäviä, jotka tukevat oppilaiden reflektiivistä ajattelua.

Lisäksi ohjelmointiprosessin opettamisessa tärkeää on näyttää sen epälineaarisuus. Valmiiksi optimoitujen, siistien ja yksinkertaisten ratkaisujen sijaan on tärkeää näyttää välivaiheita, ”epäsiistejä” ratkaisuja ja mieluiten koko ohjelmointiprosessi. Tällöin oppilaat eivät saa käsitystä, että heidän pitäisi pystyä ohjelmoimaan samanlaisia siistejä ratkaisuja suoraan [28].

Luokanopettajan on hyödyllistä tietää, millaisia ohjelmointitehtäviä voi olla, jotta tehtäviin saadaan vaihtelua ja voidaan tukea oppimista eri vaiheissa. Tätä käsitellään seuraavassa luvussa (4.2.1). Lisäksi ohjelmoinnin opetuksessa voidaan käyttää apuna viiden E:n viitekehystä (luku 4.2.2) ja TIPP&SEE-menetelmää (luku 4.2.3).

4.2.1 Ohjelmointitehtävistä

Perinteisten ohjelmakoodin kirjoitustehtävien lisäksi ohjelmoinnin oppimista voidaan tukea tehtävillä, jotka tukevat oppilaan ohjelman ymmärryksen kehittymistä. Ohjelman ymmärrystä kuvaava malli (taulukko 4.1 luvusta 4.1.3) jakaa ohjelman ymmärryksen eri alueisiin sen perusteella, kuinka suurta osaa ohjelmasta tarkastel-

laan ja tarkastellaanko sen koodia, suoritusta vai tarkoitusta. Ohjelmointitehtävien on käsitettävä kaikki mallin kaksitoista aluetta, jotta ohjelman ymmärrys voidaan saavuttaa tehtävien tuella. Ohjelman ymmärryksen tehtävä (*program comprehension task*) on tehtävä, joka vaatii oppilasta olemaan vuorovaikutuksessa ohjelmaa edustavan osan kanssa [29]. Vuorovaikutus stimuloi oppijaa rakentamaan, tarkentamaan ja parantelemaan omaa mentaalista malliaan ohjelmasta eli ohjelman ymmärrystä.

Ohjelman ymmärryksen tehtävien on katettava kaikki kaksitoista ohjelman ymmärryksen mallin aluetta, mutta tehtävät voidaan kuitenkin jakaa kolmeen päätehtävätyyppiin:

- ohjelman toiminnan jäljittäminen (*tracing tasks*),
- ohjelman toiminnan selittäminen omin sanoin (*"explain in your own words" tasks*) ja
- Parsonin ongelmat (*Parson problems*) [29].

Ohjelman toiminnan jäljittämistehtävissä vaaditaan ohjelmakoodin lukemista rivi riviltä. Tehtävissä jäljitetään ohjelmakoodin toimintaa ja pidetään kirjaa ohjelman tilasta eli esimerkiksi muuttujien arvoista jokaisen rivin jälkeen. Ohjelmaa ei siis suoriteta, vaan ohjelmakoodin toiminta täytyy päätellä ohjelmakoodista. Jäljitystehtävät koskevat ohjelman ymmärryksen mallin rakenneosuutta, koska ne vaativat ohjelmakoodin lukemista ja sen suorituksen päättelemistä.

Ohjelman toiminnan selittämistehtävissä selitetään ohjelman suoritus tai tarkoitus luonnollisella kielellä. Selitystehtävät voivat koskea ohjelmakoodin toimintaa, kun ohjelma suoritetaan tai koodin tarkoitusta ja tavoitteita. Kun osaa selittää, miten ohjelma toimii, voi varmistua siitä, että ymmärtää itse ohjelman toiminnan.

Parsonin ongelmissa on tarkoitus järjestellä koodin osat oikeaan järjestykseen. Osat voivat olla yksittäisiä lauseita tai jopa lausekokonaisuuksia riippuen ohjelmoinnin ymmärryksen mallin tasosta. Parsonin ongelmat poistavat koodin kirjoittamisen

tehtävästä, joten niiden avulla on hyvä esitellä uusia ohjelmointirakenteita.

Lisäksi ohjelmointiin kuuluvat ohjelmakoodin kirjoitustehtävät. Jotta kirjoitustehtävät eivät ole liian laajoja ja huomio voidaan kiinnittää esimerkiksi opetettavaan ohjelmoinnin konseptiin, voidaan hyödyntää mikromaailmoja. Mikromaailmat ovat pieniä, yksityiskohtaisesti kokonaisia maailmoja, joita voidaan toteuttaa sekä teksti- että lohkopohjaisilla ohjelmointikielillä [36]. Mikromaailmoissa pienellä, rajatulla määrällä komentoja voidaan tutkia esimerkiksi jotain yksittäistä ohjelmoinnin konseptia. Vaikka graafinen ohjelmointiympäristö ei ole mikromaailma, voi ympäristöön kuitenkin luoda oppilaille annettavia ohjelmia, jotka muistuttavat mikromaailmoja.

4.2.2 Viiden E:n viitekehys

Viiden E:n viitekehys on kehitetty tukemaan ohjelmoinnin opetusta ScratchMaths-projektissa, jonka tarkoitus on maksimoida ohjelmoinnin tuomat hyödyt oppilaiden matemaattisille taidoille [32]. Viitekehys soveltuu aihealueiden tai uusien konseptien opettamiseen ja oppimiseen eli se ei ole tarkoitettu yhden tehtävän rakenteeksi vaan se tukee oppimista kokonaisvaltaisemmin. Viitekehyksen viisi E-kirjainta (taulukko 4.3) ovat

- *Explore* eli tutkia,
- *Explain* eli selittää,
- *Envisage* eli kuvitella,
- *Exchange* eli tehdä yhteistyötä ja
- *bridgE* eli luoda yhteyksiä.

Tutkimisella viitataan konstruktionistisen ajattelun mukaisesti sellaisiin aktiviteetteihin, jotka tukevat oppilaiden omaa ajattelua. Oppilaille annetaan mahdolli-

Taulukko 4.3: 5E:n viitekehys

***E**xplore*	Tutkia	Ohjelman tutkiminen ja kokeileminen
***E**xplain*	Selittää	Ohjelman toiminnan selittäminen
***E**nvisage*	Kuvitella	Ohjelman toiminnan kuvitteleminen ennen sen kirjoittamista ja/tai suorittamista
***E**xchange*	Tehdä yhteistyötä	Yhteistyön tekeminen ja jakaminen muiden oppilaiden kanssa
*bridg**E***	Luoda yhteyksiä	Ohjelmoinnin yhdistäminen muihin oppiaineisiin

suuksia tutkia ideoita, kokeilla itse ja debugata käsitteellisiä tai teknisiä virheitä, kun se on tarpeellista.

Hyvä merkki jonkin asian ymmärtämiselle on se, että pystyy selittämään, mitä on oppinut, kommunikoimaan siitä erilaisin tavoin ja perustelemaan, miksi on tehty sellaisia ratkaisuja kuin on. Näin voidaan selventää ideoita ilmaisemalla niitä eksplisiittisesti ja vastaamalla vertaisten kysymyksiin. Selittämisellä korostetaan reflektiivisten kysymysten sisällyttämisen, vertaisten ja koko luokan kanssa keskustelemisen mahdollistamisen tärkeyttä.

Ohjelmoidessa on tärkeää olla mielessä tavoite jo ennen ohjelman kirjoittamisen aloittamista. Kuvittelun tarkoitus on kyetä ennustamaan, mihin lopputulokseen päädytään jo ennen kuin ohjelman suorittaa. Ohjelmointiympäristöt pystyvät usein hoitamaan itse syntaktisten virheiden käsittelyn, joten oppilaiden tulee debugata ohjelmaansa ainoastaan silloin, kun heillä on tiedossa ohjelman selkeä tavoite [32]. Tämä pitää paikkansa erityisesti graafisessa ohjelmoinnissa, joissa ohjelmakoodia ei tarvitse kirjoittaa itse ja ympäristö estää syntaktisesti virheellisen koodin muodostamisen kokonaan.

Tehokkaaseen oppimiseen kuuluu yhteistyö ja jakaminen, kuten aiemmin on jo todettu. Muilta saamansa palautteen ja keskustelun avulla oppilaat voivat nähdä oman ohjelmansa ja ongelmansa uudesta näkökulmasta. Näin he voivat kehittää omaa ajatteluaan. Oppilaat voivat saada tärkeitä ennakointi- ja selitystaitoja etenkin niitä tilanteita varten, kun oma idea ei ole täysin valmis [32]. Opettajan on kuitenkin syytä ottaa huomioon, että alakouluikäiset oppilaat kehittävät vielä omia yhteistyötaitojaan eli oppilaita on autettava yhteistyön sujuvuudessa. Tarkoituksenahan on lopulta luoda mielekkäitä mahdollisuuksia yhteistyöhön ja jakamiseen.

Yhteyksien luomisella mallissa tarkoitetaan ohjelmoinnin yhdistämistä muihin oppiaineisiin. ScratchMaths-projektissa ohjelmointia yhdistetään matematiikkaan [32], mutta Suomessa ohjelmointia on opetussuunnitelman perusteiden mukaan yhdistettävä myös käsitöiden oppiaineeseen. Ohjelmointia voi kuitenkin hyödyntää myös muissa oppiaineissa, mikä luokanopettajan on hyvä muistaa.

Viiden E:n viitekehystä voidaan soveltaa esimerkiksi uusien konseptien opetukseen niin, että huolehditaan, että jokaista E-kirjaimen edustamaa näkökulmaa käytetään tehtävissä. Yhden tehtävän ei siis tarvitse käyttää kaikkia näkökulmia. Tärkeää on, että yhtä asiaa tai konseptia opetetaan useamman tehtävän avulla kaikista näkökulmista oppimisen tukemiseksi.

4.2.3 TIPP&SEE-menetelmä

Ohjelmointitehtävien rakennetta kuvaavan TIPP&SEE-menetelmän [37] taustalla on pedagoginen lähestymistapa **Käytä**–**Muuta**–**Luo** (***Use***–***Modify***–***Create*** *pedagogical approach*). Käyttäminen viittaa erilaisiin esimerkkeihin, joita esimerkiksi uusista konsepteista annetaan. Lähestymistapa tukee oppimista ohjelmakoodin muokkaamistehtävillä ennen avoimempia luomistehtäviä, joissa kirjoitetaan ohjelmakoodia itse. Tämän tarkoitus on ohjata oppilaan huomio opittavana olevaan asiaan siten, ettei tehtävässä tarvitse heti kirjoittaa opittavan asian lisäksi kaikkea

Taulukko 4.4: TIPP&SEE-menetelmä

TIPP: Saa VIHJEitä projektisivulta		
T*itle*	Otsikko	Mikä on projektin otsikko? Mitä otsikko kertoo sinulle projektista?
I*nstructions*	Ohjeet	Mitä ohjeet sanovat, että sinun pitäisi tehdä?
P*urpose*	Tavoite	Mikä on tehtävän tarkoitus? Mitä tämä koodi opettaa sinulle?
P*lay*	Kokeile	Suorita ohjelma ja katso, mitä se tekee! Mitkä hahmot tekevät mitäkin?
SEE: KATSO sisälle		
S*prites*	Hahmot	Klikkaa hahmoa, jonka toiminnasta halua oppia tai jonka koodia haluat muuttaa.
E*vents*	Tapahtumat	Tarkastele tapahtumalohkoja, jotka aloittavat koodipätkät. Mitkä koodit ovat hyödyllisimpiä?
E*xplore*	Muuta	Kokeile tehdä erilaisia muutoksia koodeihin ja katso, mitä tapahtuu!

muuta tarvittavaa koodia. Tarjoamalla valmista koodia vähennetään kognitiivista rasitetta, minkä ansiosta on helpompi keskittyä opittavana olevaan asiaan.

TIPP&SEE-menetelmä tukee käytä ja muuta-osien tehtävien suunnittelua tarjoamalla siihen soveltuvan mallin. Menetelmä on kehitetty Scratch-ohjelmointiympäristölle ja sen on todettu parantavan oppilaiden suorituksia melkein kaikissa keskivaikeissa ja vaikeissa tehtävissä tarjoamalla apua ohjelman ymmärtämiseen [37]. Kuten taulukosta 4.4 voidaan nähdä, TIPP&SEE-menetelmän ensimmäinen osa ohjaa oppilaan tarkastelemaan ohjelmaa ilman koodin katsomista ja toinen osa tarjoaa vaiheet ohjelman koodiin tutustumiseen.

Menetelmän ensimmäinen osa kehottaa oppilaita pysähtymään uuden ohjelman äärelle ja tarkastelemaan sitä hitaasti vaihe kerrallaan. Ensin pohditaan, mitä ohjelmasta voi päätellä sen otsikon perusteella, minkä jälkeen siirrytään tarkastelemaan ohjeita. Näiden perusteella oppilas pohtii ohjelman tarkoitusta ja vasta aivan ensimmäisen osan lopuksi ohjelma suoritetaan ja kokeillaan käytännössä, mitä ohjelma tekee. Näin oppilas ei ryntää suorittamaan ohjelmaa ensimmäisenä pohtimatta, mihin ohjelma liittyy ja mikä sen tarkoitus on. Uuden ohjelman käsitteleminen hitaammin tukee oppilaan omaa ajattelua.

Myös menetelmän toinen osa pyrkii hidastamaan oppilaan työskentelyä ja antamaan tilaa oppilaan omalle ajattelulle. Toisessa osassa siirrytään tarkastelemaan ohjelman koodia eli Scratchissä katsotaan ohjelman sisälle. Ensimmäisenä oppilaan tarkoitus on valita hahmo, jonka hän kokee hyödyllisimmäksi. Tämä tukee Brennanin mainitsemaa personalisointia antamatta oppilaalle kuitenkaan liian avointa tehtävää, joka voisi tuntua liian vaativalta. Hahmon valittuaan oppilas siirtyy tarkastelemaan hahmon koodia ja etenkin sen tapahtumia, jotka aloittavat erilaisia peräkkäisrakenteita. Valittuaan tapahtuman ja peräkkäisrakenteen oppilas pääsee muokkaamaan koodia. Tarkoitus on, että muokkaamalla oppilas oppii uusia ohjelmointirakenteita ja kehittää ohjelmoinnillista ajatteluaan.

4.2.4 Yhteenveto

Ohjelmoinnin opetuksessa on tärkeää kiinnittää huomiota siihen, että etenkin aluksi ohjelmointi opetetaan sen konsepteilla. Etenkin graafista ohjelmointia opettavan luokanopettajan on tärkeää korostaa opetuksessaan ohjelmoinnin konsepteja. Tällöin oppilaat voivat hyödyntää oppimaansa tekstipohjaisessa ohjelmoinnissa yläkoulun puolella, eikä oppimista tarvitse aloittaa täysin alusta. Tarkoitus siis olisi, että yläkoulun ohjelmoinnin opetukseen siirryttäessä oppilaiden ohjelmoinnillinen ajattelu olisi hyvä.

Tätä tukevat sekä viiden E:n viitekehys että TIPP&SEE-menetelmä, jotka tukevat ohjelmoinnin konseptien oppimisprosessia. Niiden avulla oppilaat kiinnittävät enemmän huomiota opittavana olevaan ohjelmoinnin konseptiin. Tämän lisäksi viitekehys ja menetelmä korostavat sitä, että oppilaita ei vaadita heti uuden konseptin esittelyn jälkeen käyttämään sitä. Oppilaat voivat ensin rauhassa tutkia konseptin toimintaa koodissa ja suorituksessa ja sitten muokata koodia ja selvittää, miten muokkaukset vaikuttavat toimintaan. Tärkeää on siis antaa oppilaille jotain, mistä lähteä liikkeelle, eikä vaatia heti konseptin käyttöä osana kaikkea muuta ohjelmoitavaa koodia.

4.3 Teknologinen tieto

Graafisen ohjelmoinnin opetusta varten tarvittava teknologinen tieto koskee pääasiassa graafista ohjelmointia ja siihen soveltuvia ympäristöjä.

4.3.1 Graafinen ohjelmointi

Opetussuunnitelman perusteissa puhutaan ohjelmoinnista graafisessa/visuaalisessa ohjelmointiympäristössä (*graphical/visual programming environment*). Graafisella ohjelmointiympäristöllä tarkoitetaan sitä, että ympäristössä hyödynnetään visuaalisia elementtejä. Vaikka ympäristö on graafinen, voi käytetty ohjelmointikieli silti olla tekstipohjainen (*text-based*) tai lohkopohjainen (*block-based*). Tekstipohjainen ohjelmointi tarkoittaa, että ohjelmakoodi kirjoitetaan merkki kerrallaan itse. Lohkopohjainen ohjelmointi taas tarkoittaa, että ohjelmakoodi rakennetaan raahaamalla valmiita lauseita eli lohkoja paikoilleen. Monet lohkopohjaiset ohjelmointikielet ovat myös graafisia. Graafinen, lohkopohjainen ohjelmointi eroaa siis tekstipohjaisista ja muista visuaalisista ohjelmointinäkökulmista [38].

Opetussuunnitelman perusteissa ei määritellä, millaista ohjelmointikieltä on käy-

tettävä. Tässä tutkimuksessa on kuitenkin päädytty käyttämään lohkopohjaista kieltä, koska sen on osoitettu olevan hyödyllinen ohjelmoinnin oppimisessa perusopetuksessa [39], [40]. Muualla tässä tutkielmassa käytetään termiä graafinen ohjelmointi tarkoittamaan graafista, lohkopohjaista ohjelmointia.

Graafisessa ohjelmoinnissa ohjelmat siis rakennetaan raahaamalla ohjelmointiohjeita yhteen. Ohjeissa on usein visuaalisia vihjeitä siitä, miten niitä voi yhdistellä ja missä [38]. Jos siis ohjeet eivät siis syntaktisesti muodosta sallittua lausetta, ympäristö estää niiden liittämisen yhteen kokonaan. Näin ollen graafinen ohjelmointiympäristö estää syntaksivirheiden muodostumisen säilyttäen kuitenkin ohjelmointiprosessin, jossa lauseita muodostetaan yksi kerrallaan [38]. Graafisen ohjelmoinnin raahausmenetelmä poistaa myös kirjoittamishaasteet ja erikoisten välimerkkien löytämisen näppäimistöltä, mikä tekee ohjelmoinnista saavutettavampaa niille, joilla on haasteita koneella kirjoittamisessa [38].

Raahausmenetelmä tarkoittaa samalla myös sitä, että käytettävissä olevat ohjeet ovat esillä ja ohjelmoijan selattavissa. Ohjelmoija voi siis selata käytettävissä olevia ohjeita ja nähdä, mitkä sopivat ohjelmaan, eikä hänen tarvitse tietää etukäteen, mikä ohjelmointikielessä on mahdollista [38]. Tämä tukee kokeilemista ja sitä kautta oivaltamista siitä, mitä eri ohjeet tekevät. Graafinen esitystapa mahdollistaa myös luonnollisen kielen käyttämisen ohjeiden kuvauksissa [38]. Ohjelmoijan ei siis tarvitse tutustua ja sitten yrittää muistaa erilaisia avainsanoja, joita tekstipohjaisissa ohjelmointikielissä käytetään.

Graafisten ympäristöjen on tutkittu olevan tekstipohjaisia ympäristöjä helpompia oppilaiden (lapsien) [41] ja jopa aikuisten [42] mielestä yllä mainituista syistä. Oppilaat ovat kuitenkin ilmaisseet myös huolta graafisten ympäristöjen rajallisuudesta eli siitä, että ympäristöllä ei voi tehdä kaikkea, mitä esimerkiksi Javalla voisi tehdä. Vaikkei se ole täysin totta, opettajien on kuitenkin huomioitava, että oppilaat ajattelevat näin. [41]

Ympäristöjen tarjoamien mahdollisuuksien rajallisuus ja se, että ohjelma rakennetaan raahaamalla värikkäitä palikoita leikkisässä ympäristössä johtavat kysymykseen siitä, nähdäänkö graafinen ohjelmointi ”oikeana ohjelmointina” [38]. Vaikka ohjelmointia osaava näkisikin graafisessa ympäristössä samoja konsepteja ja rakenteita kuin tekstipohjaisessa ohjelmointikielessä, ei aloitteleva ohjelmoija välttämättä tunnista niitä. Tämän takia opetuksessa on oltava erityisen tarkkoja siitä, mikä on oppimisen päämäärä ja mihin opetuksessa kiinnitetään huomiota etenkin silloin, kun käytetään graafisia ympäristöjä.

On osoitettu, että oppilaat suoriutuvat paremmin graafisissa ohjelmointiympäristöissä kuin tekstipohjaisissa ympäristöissä [39], [40]. Kuitenkin kun siirrytään graafisesta ohjelmoinnista tekstipohjaiseen, edistyneempään ohjelmointiin, ei graafisen ohjelmoinnin hyödyistä ole enää apua [38]. Graafisen ohjelmoinnin ei toisaalta todettu myöskään heikentävän oppimista jatkossakaan. Koska graafinen ohjelmointi on motivoivaa aloitteleville oppilaille, on se hyvä vaihtoehto ohjelmoinnin alkeiden oppimiseen. Kun opetuksessa keskitytään ohjelmoinnin konsepteihin ja ohjelmointiprosessiin, on graafisellakin ohjelmoinnilla mahdollista päästä hyvään alkuun ohjelmointitaidon kehittämisessä.

4.3.2 Graafiset ohjelmointiympäristöt

Graafiset ohjelmointiympäristöt ovat ohjelmoinnin opetuksen kontekstissa tietojenkäsittelytieteelle ominaisia teknologioita. Opetussuunnitelman perusteet eivät ole määritelleet, mitä graafista ohjelmointiympäristöä opetuksessa olisi käytettävä. Luokanopettajat tarvitsevat siis myös (teknologista) tietoa erilaisista ympäristöistä ja niiden ominaisuuksista, jotta he voivat valita sopivimman ympäristön opetusta varten. Taulukkoon 4.5 on listattu muutamia graafisia ohjelmointiympäristöjä ja niiden ominaisuuksia.

Ohjelmointiympäristön tulee soveltua opetuskontekstiin ja opettajan opetustyy-

Taulukko 4.5: Graafisia ohjelmointiympäristöjä

Nimi	**Kielet**	**Kuvaus**
Alice (`alice.org`)	Oma lohkopohjainen ohjelmointikieli ja englanti	Ilmainen 3D-animaatioiden ja -pelien kehitysympäristö tietokoneelle
Hopscotch (`gethopscotch.com`)	Oma lohkopohjainen ohjelmointikieli ja englanti	Pelien kehitysympäristö iPadille
Kodu (`kodugamelab.com`)	Oma lohkopohjainen ohjelmointikieli ja englanti	Ilmainen 3D-pelien kehitysohjelma tietokoneelle
Pencil Code (`pencilcode.net`)	Coffeescript, Javascript, HTML, CSS ja englanti	Ilmainen selainohjelma, jolla voi mm. piirtää ja tehdä musiikkia
Scratch (`scratch.mit.edu`)	Oma lohkopohjainen ohjelmointikieli ja suomi	Ilmainen animaatioiden ja pelien kehitysympäristö selaimeen
Snap! (`snap.berkeley.edu`)	Scratchiin pohjautuva oma lohkopohjainen ohjelmointikieli ja englanti	Ilmainen Scratchistä laajennettu animaatioiden ja pelien kehitysympäristö selaimeen
Tynker (`tynker.com`)	Scratchiin perustuva oma ohjelmointikieli ja englanti	Maksullinen selainpohjainen Tynker on ohjelmoinnin oppimisen ympäristö, jonka monenlaiset kurssit harjoittavat ohjelmoinnillista ajattelua ja koodaustaitoja

liin. Lisäksi ympäristön on tuettava oppilaiden oppimista eli ympäristö ei saa olla liian monimutkainen tai vaikea oppia. Ympäristön valintaan vaikuttavat oleellisesti ympäristön kieli ja se, millä laitteella ympäristöä käytetään ja onko ympäristö ladattava laitteelle vai ei. On tärkeää, että ympäristö soveltuu opetustilanteeseen, eikä aiheuta liikaa vaivaa tai vie liikaa aikaa. Ohjelmointiympäristöä tai ohjelmointikieltä ei siis kannata välttämättä valita pelkkien ohjelmointiominaisuuksiensa perusteella vaan tärkeämpää on sen soveltuvuus opetukseen ja opetuskontekstiin. Esimerkiksi aloitteleville ohjelmoijille rajallinen, helppokäyttöinen ympäristö on parempi kuin paljon erilaisia ominaisuuksia sisältävä ympäristö.

Graafista ohjelmointia voi hyödyntää myös erilaisten laitteiden ohjaamisessa, kuten käsitöiden oppiaineessa on tarkoitus. Taulukossa 4.6 on listattu muutama laite. Laitteita ohjaamalla opetukseen saadaan konkretiaa: jotain, mitä pidellä käsissä. Esimerkiksi BBC micro:bitin on osoitettu motivoivan oppilaita sillä, että se on fyysinen laite ja oppilaat voivat nähdä ohjelmakoodinsa tekevän fyysisesti jotain [43]. Tämä saattaa auttaa joitain oppilaita ymmärtämään ohjelmoinnin tarkoituksen: tietokoneen ohjaaminen. Lisäksi laitteita voi hyödyntää ohjelman syötteenä (esimerkiksi BBC micro:bit toimii Scratchissä peliohjaimen tavoin), jolloin syötteen ja tapahtuman eroa voi korostaa.

4.3.3 Yhteenveto

Opettajan on hyvä olla tietoinen graafisen ohjelmoinnin hyödyistä ja siihen liittyvistä haasteista. Esimerkiksi kysymys siitä, onko graafinen ohjelmointi ”oikeaa ohjelmointia” voi herätä oppilaiden keskuudessa [38]. Tällöin opettajan on hyvä pystyä vastaamaan, miksi graafinen ohjelmointi on rajallisuudestaan huolimatta hyvä aloitteleville ohjelmoijille. Lisäksi tieto erilaisista ohjelmointiympäristöistä tukee opettajaa tulevaisuudessa, kun olosuhteista johtuen voi olla tarpeellista valita toinen ympäristö tai kieli. Tällöin opettaja, jolla on hyvä teknologinen tieto, osaa valita

Taulukko 4.6: Laitteita, joita voi ohjata ohjelmoimalla graafisesti

Nimi	**Kielet**	**Kuvaus**
BBC micro:bit (`microbit.org`)	Scratch, Microsoft MakeCode, Python	Piirilevyn näköinen pienoistietokone, jota voi ohjelmoida monella eri laitteella ja selaimessa
Lego Mindstorms (`lego.com/themes/mindstorms/about`)	lohkopohjainen ohjelmointikieli ja englanti	Robotti, jonka toimintaa ohjataan ohjelmoimalla monella eri laitteella
Sphero (`sphero.com`)	lohkopohjainen ohjelmointikieli ja englanti	Pallomainen robotti, jonka toimintaa ohjataan ohjelmoimalla monella eri laitteella
Vex IQ (`vexrobotics.fi`)	C-kieleen perustuva lohkopohjainen ohjelmointikieli, C-kieleen perustuva luonnollinen kieli ja C-kieli	Robotti, jota ohjelmoidaan omassa ohjelmointiympäristössään

uuden hyvän kielen ja ympäristön opetusta varten.

4.4 Teknologis-pedagoginen sisältötieto

Luokanopettajan graafisen ohjelmoinnin opetusta varten tarvittava tieto on laaja kokonaisuus ohjelmoinnin sisällöistä, pedagogiikasta ja teknologiasta. Ohjelmoinnin sisältöihin kuuluvat konseptit, käytännöt ja näkökulmat. Niiden tunteminen antaa opettajalle varmuutta siitä, mitä hän on opettamassa ja mihin sisältöihin ja aiheisiin oppilaiden huomio opetuksessa kannattaa kiinnittää. Pedagoginen sisältötieto auttaa näiden sisältöjen ja aiheiden opetuksessa. Pedagogiseen sisältötietoon kuuluvat ohjelmoinnin opetukseen soveltuvat menetelmät, tiedot ja taidot. Hyvän pedagogisen sisältötiedon avulla opettajat pystyvät luomaan ohjelmoinnin opetukseen sopivia tehtäviä ja oppimistilanteita, joissa oppilaiden ohjelmoinnin oppimista tuetaan pedagogisella osaamisella.

Digitalisoituvassa maailmassa teknologialla on yhä suurempi merkitys opetuksen ja oppimisen prosesseissa. Ohjelmoinnin opetus eroaa kuitenkin hieman muista perusopetukseen sisällytetyistä aiheista, koska ohjelmoinnissa teknologialla on suurempi rooli. Muissa oppiaineissa ja aiheissa teknologian avulla tuetaan oppimista eli teknologia ei ole välttämätön aiheen oppimisessa, vaikka sillä voikin olla positiivinen vaikutus oppimiseen. Ohjelmoinnissa teknologia on kuitenkin välttämätön eli luokanopettajan on otettava teknologia mukaan opetukseen. Jotta teknologiaa voidaan sisällyttää opetukseen tehokkaasti ja oppimista edistävästi, on luokanopettajalla oltava hyvä teknologinen sisältötieto.

Sisältöjen opetusta ja oppimista voidaan siis tukea sekä pedagogisilla että teknologisilla valinnoilla, jotka tukevat juuri niiden sisältöjen opetusta. Lisäksi on kuitenkin erittäin tärkeää, että pedagogiset ja teknologiset valinnat sopivat yhteen, jotta oppimisprosesseissa ei ole ristiriitoja tai toisiaan hankaloittavia osuuksia. Pedagogis-teknologinen tieto voi kuitenkin myös osaltaan vaikuttaa sisältöihin. Esimerkiksi si-

sältöjen käsittelyjärjestystä voidaan muuttaa pedagogisten ja teknologisten valintojen perusteella. Vaikka ohjelmoinnissakin on aiheita, jotka on opetettava ennen toisia, on niissä kuitenkin jonkin verran liikkumavaraa. Ohjelmoinnissa usein opetetaan esimerkiksi silmukat vasta mm. muuttujien jälkeen. Graafisessa ohjelmoinnissa kuitenkin hyödynnetään usein erilaisia hahmoja, joiden liikuttaminen ei vaadi datan käsittelyä. Tällöin silmukat voi olla hyödyllisempää opettaa ensin, jotta hahmot jatkaisivat esimerkiksi liikettään tarpeeksi kauan ilman, että samaa lohkoa tarvitsee käyttää useaan kertaan.

Kaiken tämän ytimessä on siis teknologis-pedagoginen sisältötieto eli opettajan ymmärrys siitä, miten ohjelmoinnin oppimista voidaan tukea pedagogis-teknologisilla valinnoilla. Vaikka luokanopettajan tarvitsema tieto jaetaan tässä sisältötietoon, pedagogiseen tietoon ja teknologiseen tietoon, on parhaimman mahdollisen opetuksen ja oppimisen kannalta oleellista, että ne kolme osa-aluetta yhdistetään. Ohjelmoinnin opetuksessa on tärkeää valita oikeat pedagogiset menetelmät ja teknologiat tiettyjen konseptien ja käytäntöjen opetukseen.

Koska ohjelmointi on osa matematiikan oppiainetta, on opettajan otettava huomioon myös matematiikan teknologis-pedagoginen sisältötieto. Israelin ja Lashin [44] tutkimuksen mukaan opettajat ovat yhdistäneet tietojenkäsittelyä ja ohjelmoinnillista ajattelua matematiikan oppitunneilleen niin, että aiheittain siirrytään helpommista tehtävistä vaikeimpiin luokka-asteiden yli. Lisäksi yksittäisten oppituntien sisällä tietojenkäsittelyä ja ohjelmoinnillista ajattelua on joko yhdistetty matematiikkaan täysin, osittain tai ei ollenkaan. Opettaja siis tarvitsee hyvin laajaa teknologis-pedagogista tietoa, jotta hän voi valita sopivan integrointiasteen taatakseen sekä matematiikan että ohjelmoinnin oppimisen.

5 Graafisen ohjelmoinnin täydennyskoulutus

Aiemmin on perusteltu, miksi opettajat tarvitsevat lisäkoulutusta ja mikä on se tieto, joka luokanopettajille pitäisi välittää, jotta he voisivat opettaa graafista ohjelmointia peruskoulun vuosiluokilla 3–6 mahdollisimman hyvin. Tässä luvussa keskitytään siihen, miten luokanopettajien tarvitsema tieto voidaan heille tarjota eli vastataan tutkimuksen toiseen tutkimuskysymykseen. Tavoitteena on luoda sellainen täydennyskoulutusmalli, joka käsittelee mahdollisimman hyvin sen tiedon, joka aiemmin on määritelty.

Koska tutkimus liittyy Opetushallituksen rahoittamaan projektiin **Ohjelmoinnillisen ajattelun työkalut perusopetuksessa** ja täydennyskoulutusmalli on luotu suoraan projektissa järjestettävään täydennyskoulutukseen alakoulun opettajille, malli esitellään tässä projektissa järjestetyn koulutuksen avulla ja sen kontekstissa. Projektin tarkoitus on tarjota koko perusopetuksen opettajille sitä tietoa, jota he tarvitsevat ohjelmoinnin opetukseen. Koska alakoulussa opetetaan graafista ohjelmointia ja yläkoulussa tekstipohjaista ohjelmointia, järjestetään projektissa kaksi koulutusta: alakoulun opettajille keväällä 2021 ja yläkoulun opettajille syksyllä 2021. Koska tutkimus käsittelee graafista ohjelmointia, käsitellään tässä ainoastaan alakoulun opettajille suunnattua koulutusta.

Graafisen ohjelmoinnin lisäksi koulutukseen haluttiin TPACK-viitekehyksen mu-

kaisesti sisällyttää ohjelmoinnillista ajattelua ja graafisen ohjelmoinnin pedagogiikkaa, kuten Kong ja muut [45] ovat tutkimuksessaan tehneet. Kongista ja muista poiketen koulutuksessa ohjelmoinnillinen ajattelu käsiteltiin kuitenkin ohjelmoinnin konsepteista erillisenä, koska käytetty ohjelmoinnillisen ajattelun tulkinta on hieman eri. Kong ja muut käyttävät Brennanin ja Resnickin mallia ohjelmoinnillisen ajattelun perustana, mutta tässä tutkimuksessa malli on esitelty graafisen ohjelmoinnin mallina (katso luku 4), koska etenkin mallin konseptit ovat hyvin ohjelmistotekniset. Ohjelmoinnillisen ajattelun käsitteleminen ohjelmoinnista erillisenä tukee myös vuosiluokkien 1–2 opetukseen suunnattua osuutta koulutuksesta. Vuosiluokilla 1–2 harjoitellaan nimenomaan ohjelmoinnillista ajattelua toiminnallisten harjoitusten avulla.

Koulutuksen kohderyhmä, tavoitteet ja rakenne määritellään luvussa 5.1. Koulutuksen graafisen ohjelmoinnin sisältöihin paneudutaan luvussa 5.2 ja ohjelmoinnin opetuksen sisältöihin luvussa 5.3.

5.1 Koulutuksen kohderyhmä, tavoitteet ja rakenne

Täydennyskoulutus on avoin kaikille, mutta kohderyhmään kuuluvat kaikki ne työssäkäyvät luokanopettajat, joilla ei ole ohjelmoinnin opetuksen koulutusta, vaikka he sitä tarvitsisivat. Koulutuksen tavoitteena on antaa opettajille työkaluja, joiden avulla he voivat ymmärtää ja osata hyödyntää eri opetusmateriaaleja. Syy tälle on se, että oppilaille ja opetukseen suunnattuja materiaaleja löytyy kyllä paljon oppikirjoista ja Internetistä, mutta opettajan voi olla vaikea hyödyntää niitä ja tukea oppilaiden oppimista, jollei hän itse tiedä aiheesta tarpeeksi. Koulutus antaa opettajalle tietoa, jonka avulla hän pystyy tekemään valintoja siitä, mitä materiaaleja opetuksessa olisi hyvä käyttää ja miten niitä voisi käyttää parhaimman oppimistuloksen varmistamiseksi.

Koulutus koostuu verkkosisällöistä ja työpajoista. Verkkosisällöillä tarkoitetaan

tässä itsenäisesti opiskeltavaa materiaalia, joka on osallistujien saatavilla koulutuksessa käytetyssä verkko-oppisympäristössä ViLLEssä [46]. Verkkosisältöihin kuuluu myös tehtäviä, jotka tukevat osallistujien oppimista, ja joiden avulla varmistetaan, että osallistujat ovat tutustuneet sisältöihin. Osallistujilla on pääsy verkkosisältöihin myös koulutuksen jälkeen.

Verkkosisältöjen aiheiden käsittelyä jatketaan työpajoissa. Koulutuksen työpajat oli tarkoitus järjestää lähiopetuksena, mutta Covid-19-viruksen aiheuttaman pandemiatilanteen vuoksi ne järjestetään videoyhteyksin. Työpajoja ei kuitenkaan nauhoiteta eli osallistujilta vaaditaan osallistuminen työpajoihin silloin, kun ne pidetään. Tällä tavalla pyritään varmistamaan työpajoihin haluttu vuorovaikutus osallistujien ja kouluttajien kesken. Vuorovaikutus edistää koulutuksessa esitellyn tiedon yhdistämistä opettajilla jo olevaan tietoon, mikä tukee opettajien kokonaisvaltaisen teknologis-pedagogisen sisältötiedon muodostumista. Työpajojen rakenne on seuraava:

1. Työpajan aikataulun ja aiheiden esittely, yhteenveto edeltävästä verkkoluennosta ja edellisten kotitehtävien ratkaisujen läpikäynti (30 minuuttia)
2. Kolme työskentelysessiota työpajan aiheista (3 x 60 minuuttia)
3. Keskustelua työpajan aiheista ja seuraavan kotitehtävän esittely (30 minuuttia)

Koulutuksen lopuksi osallistujat tekevät projektin, joka sitoo yhteen koulutuksen aiheita. Projektissa heitä pyydetään luomaan ohjelmoinnin oppituntisuunnitelma, jossa otetaan huomioon koulutuksessa käytyjä aiheita. Projektin tarkoitus on saada osallistujat pohtimaan ohjelmoinnin opettamista ja oppimista. Osallistujat saavat projekteistaan palautetta kouluttajilta. Koulutuksen läpäistäkseen osallistujien on myös tehtävä ohjelmoinnillista ajattelua mittaava testi.

Taulukko 5.1: Työmäärän jakautuminen koulutuksessa

	Tunnit
Verkkosisällöt	18
Työpajat	16
Kotitehtävät	10
Projekti	9
Testi	1
Yhteensä	54

Koulutuksen kokonaistyömäärä on kaksi opintopistettä eli noin 54 tuntia. Työmäärä jakautuu koulutuksessa verkkosisältöjen, työpajojen ja niissä annettavien kotitehtävien, projektin ja ohjelmoinnillisen ajattelun testin välillä taulukon 5.1 mukaisesti.

Koulutuksessa käsiteltävä sisältö voidaan jakaa kolmeen pääsisältöalueeseen:

1. Matematiikka, ohjelmointi ja ohjelmoinnillinen ajattelu
2. Tietokoneiden mahdollisuudet ja graafinen ohjelmointi
3. Ohjelmoinnin pedagogiikka

Koulutuksen neljä verkkosisältöä ja neljä työpajaa käsittelevät sisältöalueet taulukon 5.2 mukaisesti.

Graafisen ohjelmoinnin sisältötieto ja teknologinen sisältötieto käsitellään varsinaisesti sisältöalueessa 2 eli kahdella verkkoluennolla ja kahdessa työpajassa (ja niiden kotitehtävissä). Ensimmäisen sisältöalueen aiheet kuitenkin tukevat graafisen ohjelmoinnin oppimista, koska siellä käsitellään ohjelmoinnillista ajattelua. Graafisen ohjelmoinnin pedagogista sisältötietoa käsitellään sisältöalueessa 3 eli yhdellä verkkoluennolla ja yhdessä työpajassa.

Taulukko 5.2: Koulutuksen rakenne

Sisältöalue	**Verkkosisällöt**	**Työpajat**
1	Matematiikka, ohjelmointi ja ohjelmoinnillinen ajattelu	Ohjelmoinnillisen ajattelun harjoituksia alkuopetukseen
2	Tietokoneen mahdollisuuksista ohjelmointiin	Tutustutaan graafiseen ohjelmointiin
2	Scratch-ohjelmointi ja sen soveltaminen matematiikassa	Scratch-ohjelmointi
3	Ohjelmoinnin opetuksen menetelmiä	Ohjelmoinnin opetus

Kuten taulukosta 5.2 voi huomata, koulutuksessa käytetään ohjelmointiympäristönä Scratchiä. Scratch on kaikkein menestyksekkäin graafinen ohjelmointiympäristö [38], jossa ohjelmoidaan lohkopohjaisella ohjelmointikielellä. Scratch on kehitetty opetustarkoitukseen ja sitä on käytetty monissa tutkimuksissa ympäri maailman (esim. [12], [25], [26], [32], [37], [40]). Scratch tukee useita kieliä, kuten myös suomea ja ruotsia, minkä takia se soveltuu myös Suomen alakouluihin. Scratchin käyttö on ilmaista ja sitä voi käyttää sekä laitteelle ladattavalla sovelluksella että selaimessa, mikä tekee siitä joustavan ympäristön erilaisiin opetuskonteksteihin. Scratch myös tukee opettajia erityisellä opettajatilillä, joka avaa uusia mahdollisuuksia opetukseen Scratchillä.

5.2 Graafinen ohjelmointi

Ennen graafisen ohjelmoinnin osuutta, on koulutuksessa jo käsitelty ohjelmoinnillista ajattelua, josta siirrytään sujuvasti graafiseen ohjelmointiin ja ohjelmoinnillisen ajattelun hyödyntämiseen ohjelmoinnissa. Koulutuksen graafisen ohjelmoinnin

osuus aloitetaan teknologisesta sisältötiedosta. Toisen verkkosisällön aluksi esitellään erilaisia graafisia ohjelmointiympäristöjä, joissa ohjelmoidaan lohkopohjaisilla ohjelmointikielillä. Esiteltyihin ohjelmointiympäristöihin kuuluu myös muutama robotti tai muu laite, jota ohjataan lohkopohjaisella ohjelmoinnilla graafisessa ympäristössä. Teksti- ja lohkopohjaisen ohjelmoinnin eroja ei käsitellä, mutta osallistujille annetaan vinkkejä siitä, millaisia asioita ympäristön valinnassa tulisi ottaa huomioon.

Ohjelmistoympäristöjen esittelyn jälkeen siirrytään graafisen ohjelmoinnin sisältötietoon. Sisältötiedosta suurin painopiste annetaan koulutuksessa ohjelmoinnin konsepteille. Toisessa verkkosisällössä esitellään suurin osa ohjelmoinnin konsepteista (aliohjelmat ja kirjastot esitellään kolmannessa verkkosisällössä). Toisessa työpajassa konsepteja aletaan soveltaa ohjelmoimalla Scratchillä. Kolmannessa verkkosisällössä esitellään aliohjelmat ja kirjastot. Kolmas verkkosisältö käy läpi myös Scratch-esimerkit kaikille konsepteille, jotta opettajat voivat hyödyntää verkkosisältöjä myöhemmin ja ohjelmoinnin soveltamista matematiikassa.

Koulutuksessa käsitellyt konseptit on listattu käsittelyjärjestykseen taulukkoon 5.3. Konseptien käsittelyjärjestyksen perusteluna on osittain konseptien kompleksisuus ja vaikeusaste ja toisaalta se, missä järjestyksessä ne tulevat vastaan ohjelmoidessa Scratchissä. Esimerkiksi samanaikaisuus (tai edes rinnakkaisuus) on ohjelmoinnissa usein vaativampia aiheita, mutta Scratchissä samanaikaisuus toteutuu hyvin helposti. Toisaalta taas ehtolauseet on luonnollista käsitellä ennen silmukoita, koska silmukoissakin käytetään ehtoja. Aliohjelmat ja kirjastot taas on jätetty viimeisiksi, koska ne ovat edistyneempiä konsepteja, eivätkä ne eivät suoraan kuuluu alakoulussa opetettaviin konsepteihin. Kuten aiemmin on todettu, niistä kuitenkin voi olla hyötyä opettajille, joten ne on sisällytetty koulutukseen. Syöte ja tuloste on jätetty pois niistä konsepteista, jotka koulutuksessa on esitelty ohjelmoinnin konsepteina, mutta syötteen ja tulosteen merkitys on tullut esille koulutuksen muissa

Taulukko 5.3: Ohjelmoinnin konseptien käsittelyjärjestys täydennyskoulutuksessa

Konsepti	**Verkkosisältö**	**Työpaja**
Operaattorit (operandit, lausekkeet, lauseet eli Scratch: lohkot)	2	2
Peräkkäisyys, samanaikaisuus	2	2
Tapahtumat	2	2
Data (tyypit, muuttujat, tietorakenteet: lista)	2	2
Ehdot (ehtolauseet)	2	3
Silmukat (toistolauseet)	2	3
Aliohjelmat (Scratch: omat lohkot)	3	3
Kirjastot (Scratch: laajennokset)	3	3

osissa.

Konseptit käsitellään sekä verkkosisällöissä että työpajoissa, koska tällä tavoin verkkosisällöissä voidaan käsitellä konsepteja teoreettisemmin yhdistämättä niitä suoraan Scratchiin, mikä korostaa konseptien ymmärtämisen tärkeyttä. Verkkosisällöissä selitetään tarkasti konseptien periaatteet, toiminta ja merkitys ja työpajoissa käydään läpi ainoastaan konseptien pääpiirteet, koska työpajojen tarkoitus on siirtää konseptit käytäntöön eli ohjelmiksi Scratchiin.

Koulutukseen sisällytettiin myös pieni osuus BBC micro:bitistä. Micro:bit valittiin koulutukseen edullisen hintansa ja käyttökelpoisuutensa vuoksi. Lisäksi micro:bittiä voi hyödyntää Scratchissä, kun ottaa micro:bit-laajennoksen käyttöön. Koulutuksessa käytetään kuitenkin micro:bitin ohjelmoimiseen Microsoft MakeCode -ohjelmointiympäristöä, koska siitä löytyy Scratchin laajennosta enemmän toimintoja ja micro:bit-levyn simulaattori, jonka avulla ohjelmia voi testata myös ilman laitetta. MakeCode on graafinen, lohkopohjainen ohjelmointiympäristö, jota voi käyttää

ilmaiseksi selaimessa.

Micro:bittiä käsitellään koulutuksessa vain yhden työpajan yhden työskentelysession ajan, mikä ei ole tarpeeksi pitkä aika kaikkien micro:bitin ominaisuuksien opettamiseen. Tarkoitus onkin pääasiassa vain esitellä osallistujille, ettei uusien teknologioiden käyttöön ottaminen välttämättä vaadi kokonaan uuden oppimista, vaan samat ohjelmoinnin konseptit ovat sovellettavissa sielläkin. Micro:bitin esittelyllä toivotaan osallistujien myös saavan ideoita omaan opetukseensa.

Koulutuksessa työpajoissa ja kotitehtävissä käytetyt ohjelmointitehtävät on pyritty luomaan niin, että niissä selvästi harjoitellaan jotain tiettyä konseptia. Osassa tehtävistä on vain kirjoitettuna tehtävänanto, ja osassa osallistujat saavat jonkin verran valmista koodia. Verkkosisältöjen tehtävät on pyritty laatimaan niin, että niissä pohditaan ohjelmoinnin konsepteja yleisemmin, eikä mihinkään tiettyyn ohjelmointikieleen sidottuina. Tällä tavoin saadaan korostettua konseptien merkitystä ja sitä, että ne tosiaan ovat ohjelmointikielistä riippumattomia.

Luokanopettajan tarvitsemaan tietoon kuuluu myös tieto ohjelmoinnin käytännöistä eli ohjelmointiprosessista. Käytäntöjä ei kuitenkaan suoraan sellaisenaan käsitellä koulutuksessa, koska alakoulussa ohjelmoinnin opetuksen painopiste on ohjelmoinnin perusteissa ja lisäksi koulutuksessa aikaa on rajallisesti. Siitä huolimatta käytäntöihin ja ohjelmointiprosessiin liittyvät periaatteet näkyvät koulutuksessa mm. tehtävissä, joissa uusia konsepteja sovelletaan aiempien konseptien käsittelyn aikana tehtyjen ohjelmien kehitykseen. Ohjelmoinnilliset näkökulmat ja ohjelmointiprosessi sopisivat kuitenkin hyvin jatkokoulutuksen aiheiksi, jotta luokanopettajien tietoa niissä voitaisi syventää.

Koska koulutus keskittyi ohjelmointiin matematiikan opetuksen yhteydessä, on koulutuksessa pyritty pitämään matematiikka läsnä koko koulutuksen ajan. Siksi tehtävissäkin hyödynnetään paljon alakoulun matematiikan käsitteitä. Osa tehtävistä on valittu ScratchMaths-projektista (katso esim. [18], [32] ja `https://www.`

ucl.ac.uk/ioe/research/projects/ucl-scratchmaths/ucl-scratchmaths-curriculum), jonka opetussuunnitelman tehtävät on suunniteltu opettamaan sekä ohjelmointia että matematiikkaa Scratchillä alakouluikäisille. Tällä on pyritty konkretisoimaan opettajille sitä, miten ohjelmointia voi yhdistää matematiikkaan.

5.3 Ohjelmoinnin opetus

Luokanopettajan graafisen ohjelmoinnin opetusta varten tarvitsemaan tietoon kuuluu myös pedagoginen sisältötieto eli tieto siitä, miten graafista ohjelmointia voi ja kannattaa opettaa. Internetistä ja oppikirjoista löytyy paljon erilaisia ohjelmointitehtäviä alakoulun graafisen ohjelmoinnin opetusta varten, mutta sopivimpien tehtävien valitsemiseen tarvitaan pedagogista sisältötietoa. Täydennyskoulutukseen sisällytetty ohjelmoinnin opetuksen osuus pyrkii tukemaan osallistujien pedagogista sisältötietoa tarjoamalla tietoa erilaisista ohjelmointitehtävistä ja menetelmistä. Osuus myös pyrkii osaltaan korostamaan ohjelmoinnin konseptien oppimisen tärkeyttä.

Ohjelmoinnin opetuksen verkkosisällön osuus aloitetaan toiminnallisten harjoitusten hyödyntämisestä ohjelmoinnin konseptien opetuksessa. Toiminalliset harjoitukset on esitelty osallistujille ensimmäisessä sisältöalueessa koulutuksen alussa, kun keskityttiin pääasiassa vuosiluokkien 1–2 opetukseen. Graafisen ohjelmoinnin opetuksessa toiminallisia harjoituksia voi kuitenkin hyödyntää konseptien esittelyssä oppilaille, minkä takia niitä käsitellään koulutuksessa myös graafiseen ohjelmointiin liittyen. Toiminnallisissa harjoituksissa on erittäin tärkeää painottaa konsepteja, jotta niistä on hyötyä graafiseen ohjelmointiin siirryttäessä.

Toiminnallisten harjoitusten hyödyntämisen jälkeen viimeisessä verkkosisällössä käsitellään TIPP&SEE-menetelmä. Aluksi selitetään **Käytä**–**Muuta**–**Luo**-lähestymistapa, jota menetelmä tarkentaa, minkä jälkeen TIPP&SEE-menetelmä selitetään yksityiskohtaisesti. Menetelmän jälkeen esitellään viiden E:n viitekehys, joka tukee

ohjelmoinnin yhdistämistä matematiikan aiheisiin.

Lopuksi viimeisessä verkkosisällössä esitellään ohjelman ymmärrys ja sen malli. Jotta opettajat saisivat käsityksen siitä, miten oppilaiden ohjelman ymmärrystä voidaan konkreettisesti tukea ohjelmointitehtävillä, käydään läpi erilaisia tehtäväideoita jokaiselle ohjelman ymmärryksen alueelle. Tehtäväideat on koottu Izun ja muiden [29] artikkelin pohjalta. Ne on koulutukseen muokattu graafiseen ohjelmointiin sopiviksi, koska alun perin ne on tekstipohjaisille ohjelmointikielille suunniteltu, joten kaikki eivät suoraan toimi lohkopohjaisessa ohjelmoinnissa.

Ohjelmoinnin opetuksen verkkosisältö keskittyy pääasiassa ideoiden, vinkkien ja työkalujen antamiseen koulutukseen osallistuville, mutta työpajoissa pohditaan yhdessä ohjelmoinnin opetuksen haasteita ja opetuksessa huomioon otettavia asioita. Lisäksi työpajassa osallistujat luovat yhdessä ideoita ohjelmointitehtäville, mikä valmistaa heitä sekä koulutuksen lopuksi tehtävää projektia että ohjelmoinnin opettamista varten. Tällä pohdinnalla tuetaan opettajien osallisuutta ja sillä pyritään osoittamaan, että opettajilla on jo paljon pedagogista ja teknologista tietoa, jota he voivat hyödyntää myös ohjelmoinnin opetuksessa. Työpajassa suurin osa ajasta kuluu yhdessä pohtimiseen ja ideointiin pienryhmissä, joten ryhmien pohtimat asiat ja ideoimat tehtävät laitetaan kaikkien osallistujien nähtäville verkko-oppimisympäristöön, josta opettajat saavat ne luettua ja hyödynnettyä myöhemminkin.

6 Tulokset

Tässä tutkimuksessa on määritelty se tieto, jonka luokanopettaja tarvitsee graafisen ohjelmoinnin opetusta varten. Tiedosta on muodostettu täydennyskoulutusmalli, joka on esitelty edellisessä luvussa. Täydennyskoulutusmalli sisällytettiin keväällä 2021 järjestettyyn Ohjelmoinnillisen ajattelun työkalut alakoulussa -täydennyskoulutukseen ja sen tuloksista kerrotaan tässä luvussa.

Täydennyskoulutukseen ilmoittautuneita oli 36, joista 30 kirjautui koulutuksessa käytettyyn verkko-oppimisympäristöön. Työpajoihin osallistui keskimäärin kaksikymmentä henkilöä. Koulutuksen läpäisi hyväksytysti 16 osallistujaa. Läpäisyyn vaadittiin 75 % verkko-oppimisympäristön mahdollisista pisteistä, projektin palauttaminen ja ohjelmoinnillista ajattelua mittaavaan testiin vastaaminen. Verkko-oppimisympäristön pisteitä oli mahdollista kerätä verkkosisältöjen tehtävistä, työpajoissa annetuista kotitehtävistä ja kurssin alussa ja lopussa olleista kyselyistä.

Tässä luvussa käsitellään täydennyskoulutuksen tuloksia ja pohditaan koulutuksen onnistuneisuutta. Osallistujien odotuksia koulutuksen aluksi kerrotaan luvussa 6.1. Ohjelmoinnillista ajattelua arvioivan testin tuloksia käsitellään luvussa 6.2. Osallistujien ajatuksia koulutuksen päätyttyä kerrotaan luvussa 6.3. Lopuksi koulutuksen onnistuneisuutta pohditaan kokonaisuutena luvussa 6.4.

6.1 Odotukset koulutuksen aluksi

Koulutuksen alussa osallistujia pyydettiin vastaamaan kyselyyn, jonka tarkoitus oli kartoittaa osallistujien aikaisempaa ohjelmointikokemusta ja odotuksia koulutukselle. Kyselyyn vastasi 22 osallistujaa. Suluissa oleva luku kertoo, kuinka monta kertaa asia mainittiin kyselyn vastauksissa.

Opettajat olivat hyvin kiinnostuneita koulutuksen aiheesta (keskiarvo 4,45 asteikolla 1–5). He halusivat osallistua koulutukseen oppiakseen pääasiassa ohjelmoinnin opettamista (12), mutta myös ohjelmointia (9). Myös halu oppia uutta, kehittää taitoja työssä ja varmistaa oma osaaminen mainittiin. Koulutuksen nimi herätti osallistujissa monenlaisia tunteita: pääasiassa positiivisia, mutta pari osallistujaa ilmaisi myös epätietoisuutta ja huolta koulutuksen vaativuudesta. Moni ajatteli koulutuksen olevan käytännönläheinen tai tarjoavan käytännön työkaluja (9). Koulutuksen ajateltiin myös olevan helposti lähestyttävä, ajankohtainen ja tarpeellinen.

Osallistujat toivoivat koulutukselta käytännön työkaluja, vinkkejä ja ideoita (18) ohjelmoinnin opetusta varten (6). Osa osallistujista koki kaipaavansa apua kaikessa koulutuksen aiheeseen liittyvässä (7), mutta osa tarkensi kaipaavansa apua perusasioissa (2), ohjelmoinnin opetuksessa (2) tai siinä liikkeelle lähtemisessä (3) ja omassa ohjelmointitaidossa (4) tai ohjelmissa (3). Suurin osa osallistujista opettaa pääasiassa luokkia 3–6 (16 osallistujaa), minkä vuoksi osa toivoikin painotusta vuosiluokkien 3–6 opetukseen ja toiminnallisista harjoituksista ohjelmointiin siirtymiseen.

Kaksikymmentä osallistujaa on ilmoittanut ohjelmoineensa ennen, mutta ohjelmointitaito arvioitiin silti suhteellisen alhaiseksi (keskiarvo 2,64 asteikolla 1-5, jossa 1 = en osaa ohjelmoida lainkaan ja 5 = osaan ohjelmoida ainakin peruskoulussa opetettavan ohjelmoinnin). Osallistujat olivat ohjelmoineet mm. Scratchillä (9), Javalla (1 maininta), Pythonilla (2), Pascalilla (3) ja HTML:llä (3). Lisäksi osa osallistujista oli käyttänyt micro:bittiä (4) tai muita laitteita (mm. Bee-Bot, Blue-Bot, mBot

ja Sphero).

Monella osallistujalla oli siis jonkinlaista kokemusta ohjelmoinnista, mutta suurin osa koki silti kaipaavansa apua ohjelmoinnissa ja etenkin sen opettamisessa. Opettajat ovat aiemmin käyttäneet monia ohjelmointikieliä, ympäristöjä ja robotteja, mikä indikoi, että opettajilla on hyvin erilaiset taustat ohjelmoinnissa. Sitä kautta myös opettajien oppilaille tarjoama opetus saattaa erota hyvin paljon toisistaan, vaikka perusopetuksen tarkoitus onkin tarjota oppilaille yhtenäistä opetusta.

6.2 Ohjelmoinnillisen ajattelun testin tulokset

Koulutuksen lopuksi osallistujat tekivät ohjelmoinnillisen ajattelun testin, joka mittaa ohjelmoinnillista ajattelua eli ohjelmoinnin peruskonseptien ymmärrystä [47]. Testi on suunnattu 12–14-vuotiaille (vuosiluokille 7 ja 8), mikä tukee testin sopivuutta alakoulun opettajien ohjelmoinnin osaamisen testaamiseen, koska opettajien on osattava vähintään saman verran ja mielellään vähän enemmän kuin oppilaiden. Testi on tarkoitettu tehtäväksi 45 minuutissa, mutta etätyöskentelyn takia osallistujien aikaa ei pystytty seuraamaan tai rajoittamaan. Testin alussa kuitenkin pyydettiin osallistujia tekemään testi 45 minuutissa. Testi on englanninkielinen ja sen teki 17 osallistujaa.

Testin kysymykset käsittelevät ohjelmoinnin peruskonsepteja, jotka ovat samoja, joita luokanopettaja tarvitsee ja joita koulutuksessa käytiin läpi. Testin konseptit ovat:

- peräkkäisyys,
- silmukat: *toista ... kertaa*,
- silmukat: *toista kunnes*,
- ehtolauseet: *jos, niin*,

- ehtolauseet: *jos, niin, muuten*,
- while-ehdot ja
- yksinkertaiset aliohjelmat.

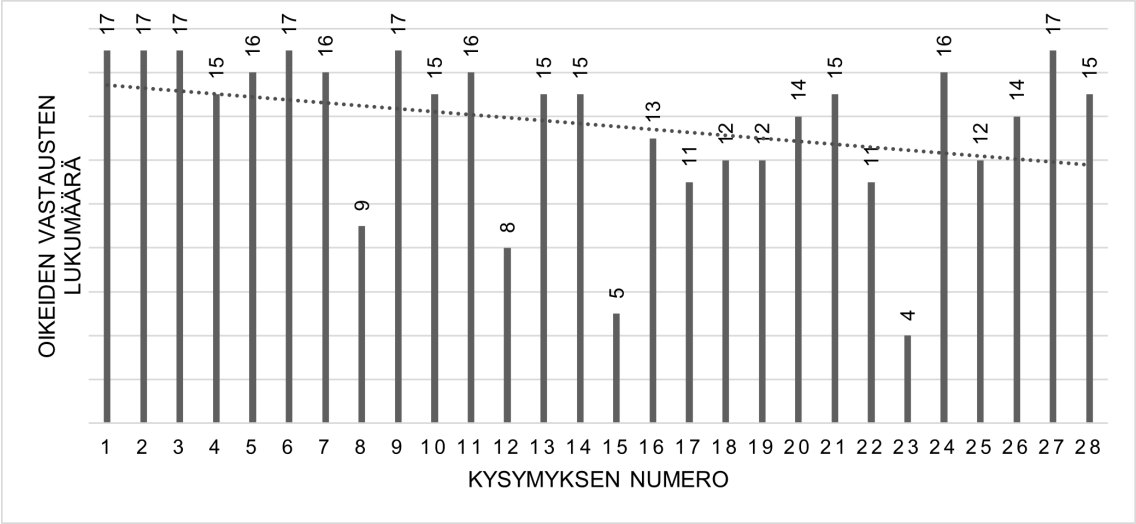
Jokaista konseptia kohti testissä on neljä monivalintakysymystä (neljä vaihtoehtoa, joista yksi on oikein).

Osallistujat osasivat ohjelmoinnillisen ajattelun periaatteita testin tulosten perusteella hyvin. 14 vastaajista sai 22 pistettä tai enemmän. 15–21 pistettä sai 2 vastaajaa ja yksi vastaaja sai 11 pistettä.

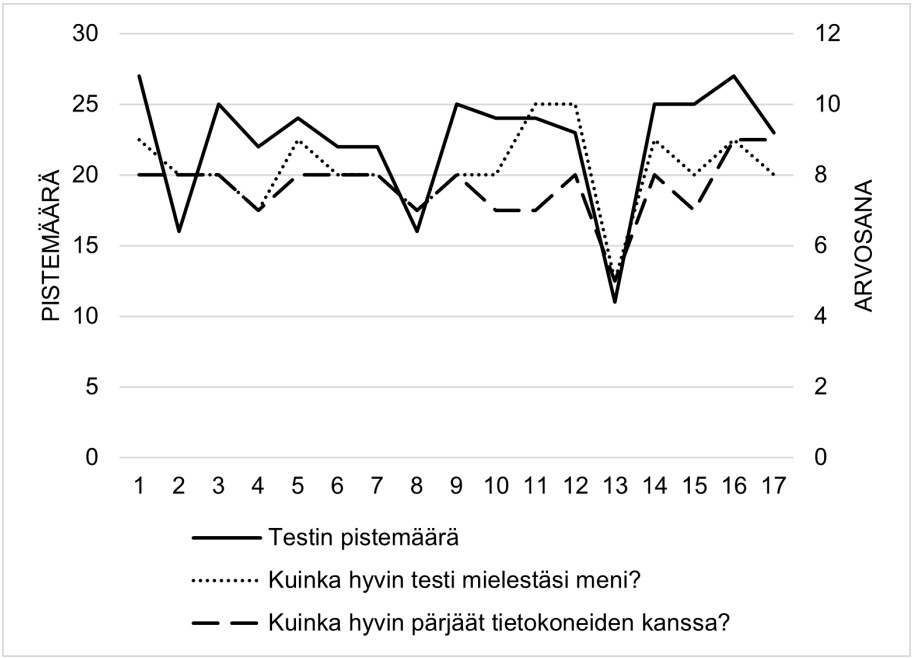
Toisaalta jos testin tuloksia tarkastellaan kysymyksittäin (kuva 6.1), on huomattavissa, että vastaajilla tuli virheitä eniten kysymyksissä 8, 12,15 ja 23, jotka edustavat eri aihealueita. Lisäksi kuvassa 6.1 näkyvä lievä pisteiden lasku loppua kohden antaa ymmärtää, etteivät opettajat hallitse vaikeampia asioita yhtä hyvin kuin helpompia.

Ohjelmointitestin lopuksi vastaajat saivat itse arvioida, kuinka hyvin he omasta mielestään osasivat ohjelmoinnillista ajattelua. Lisäksi heiltä kysyttiin, kuinka hyvin he pärjäävät yleisesti tietokoneiden kanssa. Kuvaan 6.2 on merkitty jokaisen vastaajan omat arviot ohjelmointitaidoistaan ja tietokoneiden kanssa pärjäämisestä ja testin pistemäärä. Kuvan perusteella näyttää siltä, että vastaajat arvioivat osaamisensa testissä pääosin oikein. Lisäksi myös vastaajien kokemus tietokoneiden kanssa pärjäämisestä näyttää noudattavan samanlaista kaavaa kuitenkin pienin poikkeuksin. Vaikuttaa siis siltä, että testin jälkeen vastaajat osaavat arvioida hyvin itse, kuinka hyvin he osaavat ohjelmointia. Ohjelmointitaito näyttää heijastavan myös vastaajien kokemuksiin tietokoneiden kanssa pärjäämisestä, joskin osa kokee pärjäävänsä tietokoneiden kanssa paremmin kuin osaa ohjelmoinnillista ajattelua tai ohjelmointia.

Testin perusteella voidaan sanoa, että testin tehneillä opettajilla on hyvät ohjel-



Kuva 6.1: Ohjelmointitestin oikeiden vastausten lukumäärä kysymyksittäin



Kuva 6.2: Ohjelmointitestin itsearviointi ja pistemäärä vastaajittain

moinnillisen ajattelun taidot, jotka he myös itse osaavat arvioida oikein. Sitä, miten suuri osa siitä on koulutuksen ansiota, ei tässä pystytä sanomaan, koska koulutuksen aluksi opettajilta kysyttiin ainoastaan heidän omaa kokemustaan ohjelmoinnin osaamisesta, eikä heidän ohjelmoinnillista ajatteluaan testattu ollenkaan.

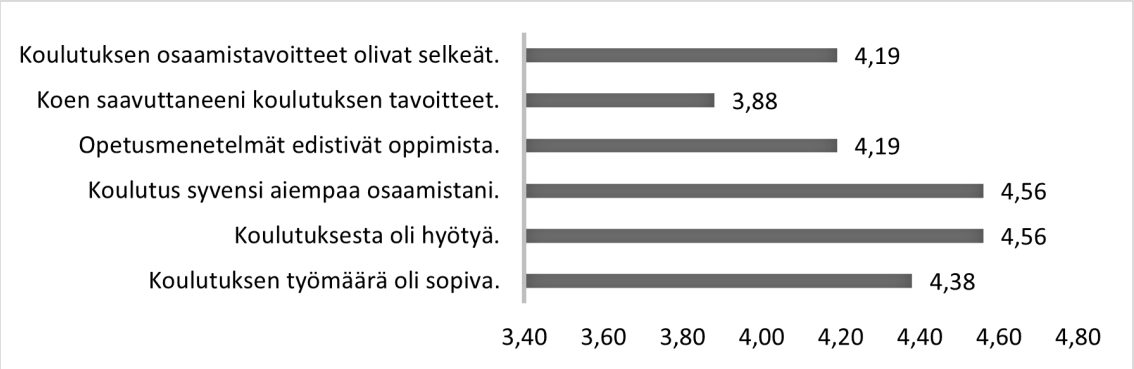
6.3 Osallistujien palaute

Koulutuksen lopuksi osallistujilta pyydettiin palautetta kirjallisen kyselyn muodossa. Kyselyssä osallistujilta kysyttiin ajatuksia yleisesti koulutukseen liittyen, mutta myös tarkemmin jokaisesta sisältöalueesta. Lisäksi kyselyn lopuksi osallistujilta kysyttiin, miten he ajattelevat, että koulutusta voisi kehittää. Palautekyselyyn vastasi 16 osallistujaa.

Koulutuksen loputtua osallistujat kokivat koulutuksen olleen hyvä kokonaisuus (5), joka oli rajattu hyvin (2) koskemaan alakoulua (1 maininta). Koulutus oli kiinnostava (2), hyödyllinen ja antoisa (3). Muutama osallistuja mainitsi oppineensa paljon, mutta pari oli sitä mieltä, että koulutus oli hankala. Pari osallistujaa mainitsi, että koulutuksesta sai käytännönläheisiä ideoita ja koulutuksen sanottiin olevan ”ihanan inhimillinen ohjelmointikurssi”, joka ”lunasti lupauksensa”.

Koulutuksen toteutustapaa sekä kehuttiin että kritisoitiin: monet pystyivät osallistumaan koulutukseen vain, koska se toteutettiin etänä, mutta muutamat olivat sitä mieltä, että lähiopetuksessa olisi oppinut paremmin. Verkkoluentojen yhdistäminen työpajojen kanssa oli kuitenkin hyvä, vaikka yhden osallistujan mielestä työpajat olivatkin liian pitkiä ja raskaiden verkkoluentojen yhteys käytäntöön jäi epäselväksi.

Yleisen palautteen asteikollisten kysymysten keskiarvot on kerätty kuvaan 6.3. Osallistujat eivät kokeneet saavuttaneensa koulutuksen tavoitteita täysin. Koska osallistujat ovat kuitenkin arvioineet koulutuksen olleen hyvä ja hyödyllinen ja työmäärän sopiva, on oppimistavoitteiden saavuttamatta jääminen saattanut johtua



Kuva 6.3: Osallistujien palautetta koulutuksesta

ennemmin henkilökohtaisista syistä. Eräs osallistujista kommentoi: ”Toivoin saavani Scratchin paremmin ’haltuun’, mutta se varmaan oli mahdotonta sillä ajalla, joka minulla oli siihen käyttää.” Kaikkien koulutuksessa käytyjen aiheiden koettiin kuitenkin vievän osaamista eteenpäin.

Ohjelmoinnin sisältöalueesta saatu palaute oli hyvää (kaikki keskiarvot yli 4 asteikolla 1–5). Osallistujat arvioivat työpajojen (4,60) olleen hieman hyödyllisempiä kuin verkkoluentojen (4,27) ja syventäneen osallistujien osaamista enemmän (4,67 työpajoille ja 4,20 verkkoluennoille). Eräs osallistuja sanoi, että ohjelmointitehtäviin tarjotut mallit ja muokattavat koodit olivat hyviä, jottei tarvitse aloittaa tyhjältä pöydältä. Myös kotitehtävien ratkaisujen käyminen läpi yhdessä seuraavassa työpajassa koettiin hyödylliseksi. Kritiikkiä sai aliohjelmiin käytetty aika, jonka yksi osallistujista mainitsi olevan liian lyhyt. Eräs osallistujista myös sanoi, että tehtävät olivat vaikeita.

Ohjelmoinnin opetuksen sisältöalueesta saatu palaute oli niin ikään hyvää (kaikki keskiarvot yli 4 asteikolla 1–5). Jälleen osallistujat arvioivat työpajat hyödyllisemmiksi (4,53 työpajoille ja 4,13 verkkosisällöille) ja niiden syventäneen osaamista enemmän (4,40 työpajoille ja 4,13 verkkosisällöille) kuin verkkosisällöt.

Aikaa toivottiin olevan enemmän sekä yleisesti että tehtävien tekoon työpajois-

sa. Työpajojen nauhoituksia ja vapaaehtoisia mahdollisuuksia keskustella tehtävistä kaivattiin, mutta myös koulutuksen muuttamista lähiopetukseksi. Yksi osallistujista toivoi, että ohjelmoinnissa olisi aloitettu ”vielä enemmän alkeista” ja toinen, että micro:bitin lisäksi olisi hyödynnetty muitakin laitteita.

Kun opettajilta kysyttiin, mitä mieltä he ovat jatkokoulutuksesta, osa oli sitä mieltä, että samantyyppinen koulutus olisi hyvä (2), osa taas sanoi, että nimenomaan ohjelmointikoulutus (2) suuremmalla käytännön painotuksella (1 maininta) olisi hyvä. Pelkkää Scratch-koulutusta (2) ja micro:bit-koulutusta (1 maininta) toivottiin, mutta myös erilaisista sovelluksista ja välineistä (1 maininta).

Opettajat kokivat, että ilman lisäresursseja (tuntiresursseja) ohjelmoinnin huomioiminen on vaikeaa (3). Opettajan oma mielenkiinto ja aktiivisuus vaikuttaa oppilaiden saamaan ohjelmoinnin opetukseen (1 maininta), joten lisäkoulusta kaikille opettajille tarvitaan niin matematiikan (3) kuin käsitöidenkin osalta (3). Lisäksi oppilaiden laitteissa on eroja, mikä vaikuttaa ohjelmoinnin opetukseen (1 maininta). Opettajat myös kaipasivat konkreettisia tapoja ottaa ohjelmointi osaksi matematiikkaa, käytännönläheisiä esimerkkejä (2) ja jopa kokonaisia tehtäväkokonaisuuksia joka luokka-asteelle.

Koulutuksen koettiin siis olevan hyödyllinen ja vievän osallistujien osaamista eteenpäin. Kuitenkin jatkokoulutuksistakin oltiin kiinnostuneita. Lisäksi opettajat olivat palautteen mukaan hieman huolissaan tuntiresurssien riittävyydestä ohjelmoinnin opetusta varten ja opettajien eritasoisesta ohjelmoinnin osaamisesta ja kiinnostuksesta, mikä vaikuttaa oppilaiden osaamiseenkin.

6.4 Pohdinta

Kokonaisuudessaan koulutus onnistui hyvin. Koulutuksen tarkoitus oli opettaa ohjelmointia ja ohjelmoinnin opetusta opettajille, eikä tarjota suoraan ohjelmoinnin opetusmateriaaleja opettajien käyttöön, mikä oli ristiriidassa muutamien osallistu-

jien ajatusten kanssa koulutuksesta. Osallistujat kuitenkin oppivat ohjelmoinnillista ajattelua, graafista ohjelmointia ja ohjelmoinnin opettamista ja olivat tyytyväisiä kurssin sisältöihin. Vaikka osa osallistujista osa ilmaisi olevansa tyytyväisiä ja osa jäi vielä kaipaamaan jotain, voidaan kuitenkin sanoa, että koulutus oli hyödyllinen. Palautteen numeeristen arvioiden korkeat keskiarvot osoittavat, että koulutuksen vaikutukset opettajien osaamiseen olivat positiivisia.

Koko koulutuksessa otettiin huomioon alakoulun ja matematiikan konteksti. Alakoulun matematiikan aiheita esiteltiin niin ohjelmointiesimerkeissä ja -tehtävissä kuin ohjelmoinnin opetuksen osuudessakin. Osallistujat huomasivat alakoulun kontekstin, mutta ohjelmoinnin ja matematiikan yhdistämiseen kaivattiin vielä tukea. Osallistujat olivat myös huolissaan resurssien riittämisestä ohjelmoinnin opetukseen osana matematiikkaa. Koska ohjelmointi kuitenkin on osa matematiikan opetusta, olisi tärkeä pyrkiä vastaamaan opettajien pyyntöön, jotta voidaan varmistua siitä, ettei ohjelmointi jää liian vähälle.

Ohjelmoinnin ja matematiikan yhdistämisen tukemisen lisäksi opettajat kaipasivat myös valmiita ohjelmoinnin opetuksen materiaaleja, jotka olisi räätälöity suomalaiseen opetussuunnitelmaan. Opettajat mainitsivat esimerkiksi valmiit tehtäväpaketit jokaiselle luokka-asteelle. Muualla maailmassa on kehitetty vastaavanlaisia ohjelmoinnin opetussuunnitelmia ja oppituntimateriaaleja sekä omana oppiaineenaan (esim. micro:bit Lessons: `https://microbit.org/lessons/`) että yhdistettynä matematiikkaan (esim. ScratchMaths: `https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths/ucl-scratchmaths-curriculum`). Lisäksi Internetistä löytyy paljon erilaisia materiaaleja ohjelmoinnin opetukseen. Suuri osa materiaaleista on kuitenkin englanniksi, joten niiden käyttäminen vaatii opettajalta ensin kääntämistä. Lisäksi materiaalit eivät välttämättä ole aiheeltaan alakouluun sopivia: esimerkiksi matematiikan ohjelmointitehtävissä saatetaan käsitellä alakoulua edistyneempiä aiheita.

Koulutus siis sujui hyvin ja osallistujat saivat hyödyllistä tietoa, jota he voivat hyödyntää työssään. Koulutuksessa kuitenkin heräsi uusia kysymyksiä ja ideoita, jotka vaativat jatkotutkimusta. Lisäksi tarvitaan myös useita muita täydennyskoulutuksia, koska tähän osallistui vain kolmisenkymmentä opettajaa.

7 Yhteenveto

Ohjelmoinnin lisääminen perusopetuksen opetussuunnitelmien perusteisiin vuonna 2014 tarkoitti sitä, että vuoden 2016 syksystä lähtien alakoulun vuosiluokilla 3–6 opetetaan ohjelmointia graafisessa ohjelmointiympäristössä. Monen luokanopettajan koulutukseen ei ole kuulunut ohjelmointia, joten luokanopettajien oli täydennettävä osaamistaan. Opettajien ohjelmointitaito ei kuitenkaan ole vielä tarpeeksi korkea. Luokanopettajat tarvitsevat siis edelleen tukea ohjelmoinnin osaamisessa, jotta voidaan taata oppilaiden ohjelmoinnin osaaminen tulevaisuudessa.

Tämän tutkimuksen tarkoitus oli vastata osaltaan luokanopettajien tarvitsemaan täydennyskoulutukseen. Tutkimuksessa hyödynnettiin suunnittelun tutkimusta, koska tavoitteena oli muodostaa työelämässä olevien luokanopettajien tarvitsema tieto ja tiedon välittämiseksi koulutusmalli. Tutkimuksessa lähdettiin liikkeelle luokanopettajan tarvitseman tiedon määrittelystä, jotta täydennyskoulutukseen saadaan sisällytettyä tärkeimmät graafisen ohjelmoinnin opetuksen tiedot. Luokanopettajan graafisen ohjelmoinnin tietoa jäsenneltiin tutkimuksessa teknologis-pedagogisen sisältötiedon viitekehyksen avulla. Tutkimuksessa selvitetyn tiedon pohjalta luotiin täydennyskoulutusmalli, joka muodosti vuoden 2021 keväällä pidetyn luokanopettajille suunnatun täydennyskoulutuksen suurimman osan.

Luokanopettajan tarvitseman tiedon tärkein osuus on graafisen ohjelmoinnin sisältötieto. Sisältötieto koostuu ohjelmoinnin konsepteista, ohjelmointikäytännöistä ja ohjelmoinnillisista näkökulmista ja ohjelman ymmärryksen mallista. Sisältöjen

opetusta varten luokanopettajien yleisen pedagogisen tiedon tueksi tarvitaan pedagogista sisältötietoa. Siihen kuuluvat tieto ohjelman ymmärrystä tukevista tehtävämalleista, ohjelmoinnin opetusta tukevasta viiden E:n viitekehyksestä ja ohjelmien muokkaustehtäville suunnitellusta TIPP&SEE-menetelmästä. Lisäksi opettaja tarvitsee teknologista sisältötietoa eli tietoa lohkopohjaisten ja tekstipohjaisten ohjelmointikielten eroista ja ohjelmointiympäristöjen graafisuudesta.

Luokanopettajan tarvitsema graafisen ohjelmoinnin tiedosta on koottu täydennyskoulutusmalli, joka koostuu itsenäisesti opiskeltavista verkkosisällöistä ja yhteisistä työpajoista. Mallissa tiedon läpi käynti aloitetaan teknologisesta sisältötiedosta eli graafisista ohjelmointiympäristöistä ja perusteluista ympäristön valinnalle opetukseen. Suurimman osan mallista vie graafisen ohjelmoinnin sisältötieto, josta tarkimmin käsitellään ohjelmoinnin konseptit, koska ne ovat ohjelmoinnin perusteiden opettamisen kannalta oleellisin osa. Kun luokanopettajille on opetettu sisällöt, siirrytään koulutusmallissa sisältöjen opetukseen eli pedagogiseen sisältötietoon.

Täydennyskoulutusmallia hyödynnettiin keväällä 2021 järjestetyssä koulutuksessa, jonka aiheita olivat ohjelmoinnillinen ajattelu, graafinen ohjelmointi ja ohjelmoinnin opettaminen. Koulutus oli avoin kaikille, mutta suunnattu työelämässä oleville luokanopettajille. Koulutukseen osallistui reilu 30 henkilöä, joista hyväksytysti koulutuksen läpäisi 16. Täydennyskoulutuksen aluksi kartoitettiin osallistujien ohjelmointikokemusta ja odotuksia koulutukseen. Kartoituksen perusteella opettajilla oli kokemusta monista ohjelmointikielistä ja -ympäristöistä, mutta kaikki kaipasivat silti tuke ohjelmointi- ja ohjelmoinnin opetusosaamiseensa.

Täydennyskoulutuksen lopuksi osallistujat tekivät ohjelmoinnillista ajattelua mittaavan testin, jonka perusteella osallistujat osaavat ohjelmoinnin tärkeimmät konseptit hyvin. Koska koulutuksen alussa moni osallistuja määritteli ohjelmointitaitonsa heikoksi, on kehitystä tapahtunut jonkin verran. Tarkemmin kehityksen suuruutta ei tässä voida arvioida, koska osallistujien ohjelmointitaitoa ei koulutuksen alussa mi-

tattu. Koulutuksen lopussa osallistujilta pyydetyn palautteen perusteella osallistujat olivat tyytyväisiä koulutuksen sisältöihin. Täydennyskoulutusten pohjimmainen tarkoitus on parantaa oppilaiden oppimista. Tutkimukseen liittyvän täydennyskoulutuksen todellista vaikutusta ei pystytä tässä mittaamaan, koska sen voisi todentaa vasta koulutuksen jälkeen opettajia ja oppilaita seuraamalla [48].

Koulutukseen osallistuneiden palautteen perusteella koulutus vaikutti onnistuvan siinä, missä oli tarkoituskin eli opettajien graafisen ohjelmoinnin osaamisen ja opettamisen osaamisen tukemisessa. Jatkotutkimusta ja -koulutusta luokanopettajille tarvitaan kuitenkin vielä ohjelmointikäytäntöjen, ohjelmointiprosessin ja ohjelmoinnillisten näkökulmien osalta. On ehdotettu, että ohjelmointia ei opetettaisi keskittyen sen konsepteihin vaan prosessiin, mitä ei alakoulun ja graafisen ohjelmoinnin kontekstissa ole vielä tarpeeksi tutkittu. Lisäksi koulutukseen osallistuneet opettajat toivoivat suomenkielisiä ja Suomen opetussuunnitelmaan räätälöityjä ohjelmoinnin opetuksen materiaaleja, mikä edellyttäisi myös jatkotutkimusta ja kehittämistä.

Näiden lisäksi tutkimuksessa ja täydennyskoulutuksessa heräsi sekä tutkijan, kouluttajien että koulutukseen osallistujien osalta huoli resurssien riittävyydestä ohjelmoinnin opetukseen. Ohjelmointi on osa matematiikan oppiainetta, mutta ohjelmointi lisättiin opetussuunnitelmaan jättämättä kuitenkaan mitään matematiikan aiheita pois tai lisäämättä tuntiresursseja. Olisi hyödyllistä tutkia, miten luokanopettajat ovat integroineet ohjelmoinnin opetukseensa, miten integrointia voisi parantaa tai helpottaa ja miten opettajia voisi tukea integroinnissa.

Lähdeluettelo

[1] Suomen virallinen tilasto (SVT), *Väestön tieto- ja viestintätekniikan käyttö*. Helsinki: Tilastokeskus, 2020. url: http://www.stat.fi/til/sutivi/2020/sutivi_2020_2020-11-10_tie_001_fi.html.

[2] ——, *Tietotekniikan käyttö yrityksissä*. Helsinki: Tilastokeskus, 2020. url: http://www.stat.fi/til/icte/2020/icte_2020_2020-12-03_tie_001_fi.html.

[3] A. Vallinkoski, "Mikä ihmeen diginatiivi?", *Yliopisto-lehti*, nro 02, maaliskuu 2017.

[4] "D. Oppimisen digitalisaatio", teoksessa *Lukiolaisbarometri 2019*, 2019. url: https://www.otus.fi/julkaisu/lukiolaisbarometri-2019/.

[5] R. Hänninen, J. Karhinen, V. Korpela, L. Pajula, O. Pihlajamaa, M. Merisalo, O. Kuusisto, S. Taipale, J. Kääriäinen ja T.-A. Wilska, *Digiosallisuuden käsite ja keskeiset osa-alueet, Digiosallisuus Suomessa -hankkeen väliraportti*, sarja Valtioneuvoston selvitys- ja tutkimustoiminnan julkaisusarja 25. 2021.

[6] C. Duncan, T. Bell ja S. Tanimoto, "Should Your 8-Year-Old Learn Coding?", teoksessa *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, New York, NY, USA: Association for Computing Machinery, 2014, s. 60–69.

[7] K. Falkner, R. Vivian ja N. Falkner, "The Australian Digital Technologies Curriculum: Challenge and Opportunity", teoksessa *Proceedings of the Sixteenth Australasian Computing Education Conference*, sarja ACE '14, vol. 148, Auckland, New Zealand: Australian Computer Society, Inc., 2014, s. 3–12.

[8] F. Heintz, L. Mannila ja T. Färnqvist, "A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education", teoksessa *2016 IEEE Frontiers in Education Conference (FIE)*, 2016.

[9] N. C. C. Brown, S. Sentance, T. Crick ja S. Humphreys, "Restart: The Resurgence of Computer Science in UK Schools", *ACM Trans. Comput. Educ.*, vol. 14, nro 2, kesäkuu 2014.

[10] M. M. Sysło ja A. B. Kwiatkowska, "Introducing a New Computer Science Curriculum for All School Levels in Poland", teoksessa *Informatics in Schools. Curricula, Competences, and Competitions*, A. Brodnik ja J. Vahrenhold, toim., Cham: Springer International Publishing, 2015, s. 141–154.

[11] R. Hijón-Neira, L. Santacruz-Valencia, D. Pérez-Marín ja M. Gómez-Gómez, "An Analysis of the Current Situation of Teaching Programming in Primary Education", teoksessa *2017 International Symposium on Computers in Education (SIIE)*, 2017.

[12] C. Sjöberg, J. Nouri, R. Sjöberg, E. Norén ja L. Zhang, "Teaching and learning mathematics in primary school trough Scratch", teoksessa *International Conference on Education and New Learning Technologies, EDULEARN18 Proceedings*, 2018, s. 5625–5632.

[13] Opetushallitus, *Perusopetuksen opetussuunnitelman perusteet 2014*, 4. painos. 2016. url: `https://www.oph.fi/fi/koulutus-ja-tutkinnot/perusopetuksen-opetussuunnitelman-perusteet`.

[14] T. Toikkanen ja T. Leinonen, ”The Code ABC MOOC: Experiences from a Coding and Computational Thinking MOOC for Finnish Primary School Teachers”, teoksessa. huhtikuu 2017, s. 239–248.

[15] P. Niemelä, ”From Legos and Logos to Lambda: A Hypothetical Learning Trajectory for Computational Thinking”, tohtorinväitöskirja, Tampereen teknillinen yliopisto, syyskuu 2018.

[16] E. Tanhua-Piiroinen, S.-S. Kaarakainen, M.-T. Kaarakainen, J. Viteli, A. Syvänen ja A. Kivinen, *Digiajan peruskoulu*, sarja Valtioneuvoston selvitys ja tutkimustoiminnan julkaisusarja 6. Valtioneuvoston kanslia, helmikuu 2019.

[17] E. Tanhua-Piiroinen, S.-S. Kaarakainen, M.-T. Kaarakainen ja J. Viteli, *Digiajan peruskoulu II*, sarja Opetus- ja kulttuuriministeriön julkaisuja 17. Opetus- ja kulttuuriministeriö, toukokuu 2020.

[18] L. Benton, C. Hoyles, I. Kalas ja R. Noss, ”Bridging Primary Programming and Mathematics: Some Findings of Design Research in England”, *Digital Experiences in Mathematics Education*, vol. 3, nro 2, s. 115–138, 2017.

[19] S. McKenney ja T. C. Reeves, *Conducting Educational Design Research*. Routledge, 2019, vol. 2.

[20] M. Apiola ja E. Sutinen, ”Design Science Research for Learning Software Engineering And Computational Thinking: Four Cases”, *Computer Applications in Engineering Education*, vol. 29, s. 83–101, 7 kesäkuu 2020.

[21] A. R. Hevner, ”A Three Cycle View of Design Science Research”, *Scandinavian Journal of Information Systems*, vol. 19, nro 4, 2 2007.

[22] L. Shulman, ”Knowledge and Teaching: Foundations of the New Reform”, *Harvard Educational Review*, vol. 57, nro 1, s. 1–23, tammikuu 2011.

[23] M. Koehler ja P. Mishra, ”What is Technological Pedagogical Content Knowledge (TPACK)?”, *Contemporary Issues in Technology and Teacher Education*, vol. 9, nro 1, s. 60–70, maaliskuu 2009.

[24] R. Vivian ja K. Falkner, ”Identifying Teachers' Technological Pedagogical Content Knowledge for Computer Science in the Primary Years”, teoksessa *Proceedings of the 2019 ACM Conference on International Computing Education Research*, sarja ICER '19, Toronto ON, Canada: Association for Computing Machinery, 2019, s. 147–155.

[25] K. Brennan ja M. Resnick, ”New Frameworks for Studying And Assessing the Development of Computational Thinking”, teoksessa *Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada, 2012.

[26] C. Gleasman ja C. Kim, ”Pre-Service Teacher's Use of Block-Based Programming and Computational Thinking to Teach Elementary Mathematics”, *Digital Experiences in Mathematics Education*, vol. 6, s. 52–90, tammikuu 2020.

[27] L. Zhang ja J. Nouri, ”A Systematic Review of Learning Computational Thinking Through Scratch in K-9”, *Computers & Education*, vol. 141, s. 103 607, 2019.

[28] J. Bennedsen ja M. E. Caspersen, ”Revealing the Programming Process”, teoksessa *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, sarja SIGCSE '05, St. Louis, Missouri, USA: Association for Computing Machinery, 2005, s. 186–190.

[29] C. Izu, C. Schulte, A. Aggarwal, Q. Cutts, R. Duran, M. Gutica, B. Heinemann, E. Kraemer, V. Lonati, C. Mirolo ja R. Weeda, ”Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories”, teoksessa *Proceedings of the Working Group Reports on Innova-*

tion and Technology in Computer Science Education, sarja ITiCSE-WGR '19, Aberdeen, Scotland UK: Association for Computing Machinery, 2019, s. 27–52.

[30] C. Schulte, "Block Model: An Educational Model of Program Comprehension as a Tool for a Scholarly Approach to Teaching", teoksessa *Proceedings of the Fourth International Workshop on Computing Education Research*, sarja ICER '08, Sydney, Australia: Association for Computing Machinery, 2008, s. 149–160.

[31] S. Sentance, "The I in PRIMM", *Hello World*, nro 14, s. 50–53, syyskuu 2020.

[32] L. Benton, C. Hoyles, I. Kalas ja R. Noss, "Building Mathematical Knowledge with Programming: Insights from the ScratchMaths Project", teoksessa *Constructionism in Action 2016: Conference Proceedings*, Thung Khru, Thailand: Suksapattana Foundation, 2016, s. 26–33.

[33] J. Hromkovič ja R. Lacher, "The Computer Science Way of Thinking in Human History and Consequences for the Design of Computer Science Curricula", teoksessa *Informatics in Schools: Focus on Learning Programming*, V. Dagienė ja A. Hellas, toim., Cham: Springer International Publishing, 2017, s. 3–11.

[34] S. Papert, "Computer Criticism vs. Technocentric Thinking", *Information Technology and Education*, vol. 16, s. 22–30, 1 tammikuu 1987.

[35] K. Brennan, "Beyond Technocentrism: Supporting Constructionism in the Classroom", *Constructivist Foundations*, vol. 10, nro 3, s. 289–296, 2015.

[36] R. Pelánek ja T. Effenberger, "Design and analysis of microworlds and puzzles for block-based programming", *Computer Science Education*, s. 1–39, 2020.

[37] J. Salac, C. Thomas, C. Butler, A. Sanchez ja D. Franklin, "TIPP&SEE: A Learning Strategy to Guide Students Through Use - Modify Scratch Activities", teoksessa *Proceedings of the 51st ACM Technical Symposium on Com-*

puter Science Education, sarja SIGCSE '20, Portland, OR, USA: Association for Computing Machinery, 2020, s. 79–85.

[38] D. Weintrop, "Block-Based Programming in Computer Science Education", *Communications of the ACM*, vol. 62, nro 8, s. 22–25, heinäkuu 2019.

[39] D. Weintrop ja U. Wilensky, "Comparing Block-based and Text-based Programming in High School Computer Science Classrooms", *ACM Transactions on Computing Education*, vol. 18, nro 1, elokuu 2017.

[40] N. Simpkins, "I Scratch and Sense But Can I Program?: An Investigation of Learning with a Block Based Programming Language", *International Journal of Information and Communication Technology Education*, vol. 10, s. 87–116, heinäkuu 2014.

[41] D. Weintrop ja U. Wilensky, "To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming", kesäkuu 2015.

[42] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. Shepherd ja D. Franklin, "Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices", huhtikuu 2018.

[43] S. Sentance, J. Waite, S. Hodges, E. MacLeod ja L. Yeomans, ""Creating Cool Stuff": Pupils' Experience of the BBC Micro:Bit", teoksessa *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, New York, NY, USA: Association for Computing Machinery, 2017, s. 531–536.

[44] M. Israel ja T. Lash, "From Classroom Lessons to Exploratory Learning Progressions: Mathematics + Computational Thinking", *Interactive Learning Environments*, vol. 28, s. 1–21, lokakuu 2019.

[45] S.-C. Kong, M. Lai ja D. Sun, ”Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy”, *Computers & Education*, vol. 151, nro 103872, 2020.

[46] M.-J. Laakso, E. Kaila ja T. Rajala, ”ViLLE – collaborative education tool: Designing and utilizing an exercise-based learning environment”, *Education and Information Technologies*, vol. 23, heinäkuu 2018.

[47] M. Román-González, J.-C. Pérez-González ja C. Jiménez-Fernández, ”Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test”, *Computers in Human Behavior*, vol. 72, s. 678–691, 2017.

[48] M. Kennedy, ”How Does Professional Development Improve Teaching?”, *Review of Educational Research*, vol. 86, helmikuu 2016.