Kognitiivisen oppipoikamallin menetelmien soveltaminen ohjelmoinnin opetuksessa

Pessi Moilanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 11. toukokuuta 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Pessi Moilanen

Työn nimi — Arbetets titel — Title

Kognitiivisen oppipoikamallin menetelmien soveltaminen ohjelmoinnin opetuksessa

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	11. toukokuuta 2014	19

Tiivistelmä — Referat — Abstract

Kognitiivista oppipoikamallia pystyy soveltamaan tietojenkäsittelytieteen opetukseen. Kerromme hieman tehostetusta oppipoikamallista, mikä on laajennettu kognitiivisesta oppipoikamallista.

Tutkielmassa kerromme kognitiivisen oppipoikamallin käytännön sovelluksista, menetelmistä ja oikeista tilanteista, joissa sitä on käytetty. Tilanteita joista kerromme ovat kognitiivisen oppipoikamallin käyttö teoreettisen tietojenkäsittelytieteen opettamisessa, lasten opetuksessa ja kognitiivisen oppipoikamallin laajennuksen tehostetun oppipoikamallin käyttöä ohjelmoinnin perusteet- ja jatkokursseilla.

Halusimme tutkia kognitiivista oppipoikamallia, koska sen avulla on mahdollista parantaa opetuksen laatua ja korvata perinteistä tapaa opettaa. Esitämme syitä miksi kognitiivinen oppipoikamalli on hyvä vaihtoehto perinteiselle opetukselle ja kuinka sitä voi ottaa käyttöön opetuksessa erilaisin keinoin. Näytämme myös muutamia valittuja tutkimustuloksia, jotka tukevat kognitiivisen oppipoikamallin toimivuutta käytännössä.

Avainsanat — Nyckelord — Keywords
kognitiivinen oppipoikamalli, tehostettu oppipoikamalli, tietojenkäsittelytieteen opetus

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Opettamiseen on olemassa monia erilaisia tekniikoita ja menetelmiä. Tässä kirjoituksessa käymme läpi perinteisen oppipoikamallin ja kognitiivisen oppipoikamallin perusideat ja kognitiivisen oppipoikamallin erilaiset opetusmenetelmät. Kerromme miten kognitiivistä oppipoikamallia on sovellettu tietojenkäsittelytieteen opetuksessa. Lopuksi kerromme muutamia tapoja kuinka kognitiivinen oppipoikamallia pystyy hyödyntämään.

Ihmiset oppivat asioita seuraamalla ja matkimalla toistensa toimintoja. Tiedon ja taidon eteenpäin siirtäminen jälkipolville on ollut avaimena ihmiskunnan kehittymiselle ja selviytymiselle. Siirrettävän tiedon määrä kasvaa vuosi vuodelta ja tiedon ylläpitoon ja omaksumiseen kuluu ihmisillä enemmän aikaa. Ihmisellä on rajallinen aika oppia asioita, joten oppiminen on tehtävä mahdollisimman tehokkaasti ajankäytön suhteen.

Perinteinen oppipoikamalli (apprenticeship) on peräisin vanhoilta ajoilta. Kyseisesessä mallissa mestari ja oppipoika tekevät töitä yhdessä. Perinteisessä oppipoikamallissa on suoritettava työ jotakin konkreettista, joka on yleensä helposti tarkkailtavissa. Työtehtävä voi olla esimerkiksi haarniskan takominen, kakun leipominen tai paidan kutominen. Mestari ohjeistaa oppipoikaa työn suorituksessa soveltaen erilaisia opetustekniikoita suoritettavasta työstä riippuen. [5]

Mestari on henkilö, joka on alansa ammattilainen eli erittäin kokenut tietyssä asiassa. Tämä kokemus on seurausta pitkäaikaisesta ja kovasta harjoittelusta. Mestari pystyy antamaan työnvaiheissa erilaisia neuvoja, miten ja miksi asiat tapahtuvat. Syynä tähän on monissa tilanteissa tehtävien tärkeys ja vaikeus, mikä ei jätä tilaa virheille. Oppipoika on henkilö, joka on noviisi opetettavassa asiassa ja pyrkii oppimaan mestarin opastuksella. Oppipojalle yksi oppimistavoista on vierestä tarkkaileminen, kun mestari suorittaa jotakin työtehtävää. Mestarilla voi olla useampia oppipoikia samaan aikaan. Mestari pyrkii opettamaan ja antamaan oppipojille töitä suoritettavaksi, joidenka seurauksena oppipojille karttuisi ammattitaitoa harjoiteltavasta asiasta. Oppipoika saa suoritettavakseen mestarin avustuksella pienempiä tehtäviä, jotka ovat sopivia oppipojan sen hetkiselle taitotasolle. Oppipojalle on mielekkäämpää tehdä pienempiä tehtäviä, koska tällöin hän näkee oman työnsä jäljen ja pystyy nauttimaan onnistumisista useammin. Pienten tehtävien myötä oppipojalle kertyy kokemusta, jota hän pystyy hyödyntämään laajemmissa tehtävissä. Tarpeeksi opittuaan hän on valmis täysin itsenäiseen työskentelyyn. [5]

1.1 Perinteinen opetus

Perinteinen opetus käsittää perinteisiä luentoja ja laskuharjoituksia, jotka sisältävät minimaalista kommunikaatiota oppilaiden ja opettajan välillä. Ohjelmointi on taito, jota pystyy oppimaan tehokkaasti henkilöltä, joka on

ammattilainen ohjelmoinnissa [15]. Opetuksessa käytettävien keinojen valitseminen on oleellista, koska kaikki tavat eivät tue tehokasta oppimista [13]. Suurinosa yliopistotason ohjelmointikursseista ovat sekoitus nettimateriaaleja ja paikallisia luentoja [13].

Suurin osa yliopistoista käyttävät edelleen perinteistä tapaa opettaa ohjelmoinnin peruskurssia. Ohjelmoinnin peruskurssilla on ollut kaikkialla taipumuksena sisältää suuria kurssin keskeytysprosentteja. Helsingin yliopistolla keskeytysprosentin keskiarvo pidemmältä aikaväliltä on ollut 45%. Perinteinen tapa opettaa kattaa luentoja, kotona suoritettavia tehtäviä ja mahdollisia harjoitustilaisuuksia, joissa tehtävien malliratkaisut esitetään ja käydään läpi. [14]

Luennot tapaavat olla rakenteeltaan ohjelmointikielen ominaisuuksiin painottuneita sen sijaan, että niillä käsiteltäisiin yleisiä strategioita käsitellä ongelmia ja haasteita. Perinteistä tapaa käytetään laajalti, vaikka useat tutkimukset osoittavat, että oppimiseen liittyvät ongelmat eivät liity syntaksin oppimiseen tai kielikohtaisiin semanttisiin seikkoihin [14]. Näiden seikkojen sijaan pitäisi keskittyä siihen kuinka opetetaan keinoja, joita soveltamalla pystyy rakentamaan toimivia ohjelmia.

Itsenäisesti tehtävien kotitehtävien ongelmana on se, että osa oppilaista jää jumiin tehtäviin, jos ne ovat liian vaikeita. Oppilaat saattavat lopettaa kurssin kesken, jos he kokevat kurssin liian haasteelliseksi ja vastenmieliseksi. Toinen haitta kotona suoritettavissa kotitehtävissä on, että oppilaat oppivat huonoja työtapoja ja menetelmiä ratkaistaessaan tehtäviä itsenäisesti. Oppilas saa äärimmäisen pientä ohjausta tällaisessa tilanteessa. Opetuspsykologiassa tunnetusti kyseiset tilanteet eivät ole parhaimpia aloittelijoille, jotka yrittävät oppia kognitiivisesti haastavia asioita. [14]

Kun ohjelmointia opetetaan täysin kokemattomille, ei tavoitteena ole luoda oppilaita, jotka osaavat tehdä toimivia ohjelmia konemaisesti ilman syvempää ymmärrystä asiasta. Turhan usein opetuksessa keskitytään ohjelman valmistumiseen ja lopputulokseen. On pyrittävä luomaan oppilaita, jotka osaavat soveltaa ohjelmoinnin käsitteitä ja rakenteita ohjelmointiin liittyviin ongelmiin.

Nykyään myönnetään, että perinteiset luennot eivät ole kovin tehokas tapa opettaa ohjelmointia ja harjoitustilaisuuksissakin oppiminen jää vähälle, koska ohjausta on yleensä liian vähän tarjolla [15].

Ohjelmointitehtäviä suunniteltaessa kannattaa tehtävä muodostaa pienistä palasista kokonaisuudeksi. Kun oppilaat alkavat opettela ohjelmointia, on tärkeää tuoda esiin kuinka ohjelmat muodostuvat pienistä paloissa. Ison kokonaisuuden ollessa pienissä palasissa oppilaat saavat niistä enemmän irti, koska he oppivat, miksi jokaisella tehtävän osalla tulee olemaan merkitys lopputuloksen kannalta ja miten isompi ohjelma muodostuu. Näin oppilaalle pyritään mallintamista hyödyntäen rakentamaan vaihe vaiheelta muodostuva kokonaisuus, jossa eri vaiheiden merkitys ymmärretään. [6][13]

Ohjelmoinnin oppiminen on vaikeaa [15]. Ohjelmoinnin johdatuskurssin

keskeytysluvut ovat suuret, joten on aika selvää, että perinteistä lähestymistapaa opettaa on parannettava [14]. Nykyään monet myöntävät, että luennointi ei ole paras tapa opettaa ohjelmoimaan, mutta myös harjoitustilaisuudetkin ovat huonohkoja [15]. Harjoitustilaisuuksissa saa minimaalista ohjausta tehtävien parissa. Aloittelijoille kyseinen lähestymistapa ei ole hyvä [15].

Oppilaat saattavat oppia huonoja ohjelmointikäytänteitä jos he joutuvat tekemään ohjelmointitehtävät täysin itsenäisesti [14]. Ilman palautetta on vaikeaa kehittää omia taitojaan ja korjata virheitään, koska niiden löytäminen itse on vaikeaa ja kuinka ne kannattaa korjata ei välttämättä ole oppilaalle tiedossa [18]. Tekstikirjat eivät paljasta prosessia kuinka ongelmia pitää lähteä ratkaisemaan ja miten niissä pitää ajatella [4]. Liian moni oppilas ei osaa tehdä järkeviä ohjelmia opiskeltuaan vuoden tai kaksi ohjelmointia [4].

Ohjelmointi ei ole pelkästään yksinkertaisten syntaksirakenteiden ja niiden sovellusten tekemistä, vaan käytännön ongelmanratkaisutaitojen harjoittamista tarkoituksellisessa kontekstissa [16]. Tehokkain tapa kerätä tietoa ja taitoa kognitiivisen tieteellisen tutkimuksen mukaan tulee aktiivisen ongelmanratkaisun aikana [1]. Korkeamman tason opetuksessa ohjelmointi nähdään taitona, joka hyötyy suorasta kontaktista, tuesta ja palautteesta henkilöiltä jotka ovat jo mestareita siinä [9].

2 Kognitiivinen oppipoikamalli

Kognitiivinen oppipoikametelmä on hyvä työkalu asioiden opettamiseen varhaisessa vaiheessa ja sen avulla pystyy syventymään asioihin [10]. Se keskittyy kehittämään taitoja ja pätevyyttä ympäristössä, jossa on apua tarjolla [10] .

Kognitiiviseen oppipoikamalliin on määritelty kuusi opetusmenetelmää, jotka ovat mallintaminen (modelling), valmentaminen (coaching), oppimisen oikea-aikainen tukeminen (scaffolding), artikulointi (articulation), peilaaminen (reflection) ja tutkiminen (exploration). Ne tukevat, hyödyntävät ja ovat vahvasti sidoksissa toisiinsa. [5]

Mallintaminen, valmentaminen ja oppimisen oikea-aikainen tukeminen ovat päätekniikat kognitiivisessa oppipoikamallissa [10]. Ne on suunniteltu niin, että oppilaat saavat rakennettua käsitteellisen mallin ongelman tilasta ja he pystyvät kehittämään kognitiivisia taitoja tarkkailun ja harjoittelun kautta [10].

Sekä artikulointi että peilaaminen pyrkivät saamaan oppilaat sisäistämään havainnot ja kokemukset, mutta myös samalla kehittävät uutta tietoa ja ongelmanratkaisutaitoja [10]. Tutkiminen edistää itsenäisyyttä ja rohkaisee itsenäiseen ongelman muodostamiseen ja ratkaisemiseen [10].

Kognitiivinen oppipoikamalli on tehokas tapa opettaa noviiseille ohjelmointia[2]. Se auttaa oppilaita rakentamaan parempaa ymmärrystä konsepteista ja sa-

malla heistä tulee itsenäisempiä kokemuksen myötä [2].

Todellinen haaste on kuitenkin se, kuinka oppilas osaa hyödyntää opittua syntaksia ongelmien ratkaisemiseen [2]. Tässä pisteessä hän tarvitsee oppiakseen jonkin asteen ohjausta tai neuvontaa [2]. Opettajan rooli on antaa neuvoja ja ohjeita tähän uuteen maailmaan [2].

Kognitiivinen oppipoikamalli (cognitive apprenticeship) on teoria prosessista, jossa mestari opettaa taitoa oppipojalle [3]. Siinä käsitellään opettamista tilanteissa, joissa usein työnvaiheet opetettavasta asiasta eivät ole konkreettisesti näkyvillä oppipojalle. Tälläisiä asioita voivat olla esimerkiksi yleinen ongelmanratkaisu, algoritmin valinta tai luetun ymmärtäminen. Kisällin on vaikea oppia mestarilta, jos ainoat näkyvät työnvaiheet ovat kaavojen pyörittelyt ja ratkaisun ilmestyminen paperille. Mestarin on saatava ajatuksensa näkyviin. Toisinaan ajattelu on kuitenkin abstrahoitunut ja yhdistynyt muihin korkeamman tason käsitteisiin, jotka eivät ole oppilaalle vielä tuttuja. Tällöin opettajan on selitettävä asia oppipojalle ymmärrettävällä tavalla. Ajatusten näkyviin tuontiin on erilaisia tekniikoita, joita voidaan soveltaa tilanteesta riippuen. Kognitiivisessä oppipoikamallissa on samat periaatteet kuin perinteisessä oppipoikamallissa, poikkeavuutena on opetettavien asioiden luonne ja kuinka oppi saadaan perille oppilaalle [14]. Kognitiivisessä oppipoikamallissa on suuri painotus itse työskentelyssä lopputuotteen valmistumisen sijaan [14]. Kognitiivinen oppipoikamalli vaikuttaa myönteisesti opiskelijan motivaatioon ja opiskelun mukavuuteen, mikä puolestaan vaikuttaa huomattavasti oppimiseen [15].

Useat tutkimukset osoittavat sekä motivaatiolla että mukavuustasolla olevan huomattava merkitys oppilaan oppimiseen. Kognitiivinen oppipoikamalli sisältää monia elementtejä, jotka kehittävät molempia, mutta ohjelmointiharjoitusten osuus on merkittävä. Liian vaikeat tehtävät tappavat motivaation. [14]

Se kuinka opitaan kirjoittamaan koodia on samantapaista oppimista kuin vieraan kielen oppiminen. Opiskelijan pitää oppia sanoja monille objekteille ja toiminnoille ja kieliopilliset säännöt lauseilla. [2]

2.1 Kognitiivisen oppipoikamallin opetusmenetelmät

Mallintamisessa työn suorittamisen eri vaiheet pyritään saamaan näkyviin. Sen suorittaa yleensä opettaja tai joku muu oppilasta kokeneempi henkilö. Työ mallinnetaan tavalla, jossa vaiheet havainnollistetaan tasolla, jolla oppilas pystyy ne ymmärtämään. Oppilas pyrkii muodostamaan käsitteellissen mallin suoritetuista vaiheista ja hän näkee työn eri vaiheiden valmistumisesta koituvat seuraamukset. Oppilas saa käsityksen, miten suoritetut vaiheet johtavat onnistuneeseen lopputulokseen. Kognitiivisessä oppipoikamallissa ongelmana on ajatustyön esille saanti oppilaalle, joten on pyrittävä käyttämään keinoja, joilla oppilas pystyy hahmottamaan ne. Ongelmanratkaisuprosessia mallintaessa mestari kertoo kussakin työn vaiheessa ääneen mitä hän tekee

ja miksi, jolloin oppilas saa käsityksen ongelmanratkaisuprosessin vaiheista ja tarkoituksista. [5]

Valmentamisessa opettaja tarkkailee oppilaan työskentelyä ja antaa hänelle henkilökohtaisia neuvoja ja pyrkii avustamaan oppilasta kriittisillä hetkillä. Opettaja pyrkii samaistumaan oppilaan osaamiseen ja pyrkii ohjaamaan tätä oikeaan suuntaan. Opettaja suunnittelee ja rakentaa oppilaalle tehtäviä, jotka ovat oppilaan osaamistasolle sopivia. [5]

Oppimisen oikea-aikainen tukeminen käsittää erilaisia tekniikoita ja strategioita [5]. Opettajalle on tärkeää, että hän pystyy arvioimaan oppilaan taitotasoa, kykyä oppia ja opetettamiseen varattua aikaa. Opettajan on tarkkailtava oppilaan työskentelyä varmistaakseen, että oppilas ei jää jumiin työssään, josta seuraa oppilaan turhautuminen ja motivaation menettäminen [1]. Opettajan on annettava oppilaalle oikea-aikaisia neuvoja tehtävissä etenemiseen. Opettaja antaa oppilaalle apua tehtävän vaiheissa, joita oppilas ei vielä itsenäisesti osaa suorittaa. Opettaja voi antaa sekä suullista ja kirjoitettua palautetta oppilaalle, jonka perusteella oppilas pystyy parantamaan työskentelyään [1]. Opettajan täytyy suhteuttaa käytettävissä oleva aika opetettavien asioiden laajuuteen ja sisältöön. Pieniä yksityiskohtia ei pysty hiomaan loputtomiin, jos aikaa on rajallisesti. Annettua apua ja neuvoja ei saa olla liikaa, jotta tehtävän tuoma oppimismahdollisuus ei mene pilalle [16]. Oppilaalle voidaan antaa tehtäviä, jotka sisältävät tekniikoita, joita oppilas ei entuudestaan osaa [5].

Opettaja vähentää oppimisen oikea-aikaista tukemista (fading) sitä mukaan, kun oppilaalle kertyy kokemusta työstettävästä asiasta, jättäen oppilaalle enemmän vastuuta. Nämä keinot kehittävät oppilaan itsetietoisuutta ja oppilas oppii korjaamaan itse virheitään, jonka seurauksena hän ei tarvitse enää niin paljoa ulkopuolista apua [5]. Henkilökohtaista apua ei tarvitse olla paljon, kunhan oppilasta silloin tällöin ohjataan oikeaan suuntaan [15].

Artikuloinnissa oppilaiden on muutettava ajatuksensa, tietonsa ja ajatusprosessinsa sanoiksi. Artikuloinnin seurauksena oppilaat voivat vertailla paremmin omaa ajatteluaan opettajan ajatteluun. Opettaja pääsee käsiksi oppilaan ongelmanratkaisuprosessiin, kun hän saa selville miten oppilas ajattelee erilaisissa tilanteissa. Hän voi pyytää oppilasta puhumaan ääneen näiden suorittaessa jotakin tehtävää, näin oppilas joutuu muotoilemaan ajatuksensa sanoiksi. [5]

Peilaamisessa oppilas vertailee työskentelyään opettajaan, kokeneempaan henkilöön tai toiseen oppilaaseen. Oppilas saa tätä kautta tietoa siitä, miten hänen työskentelynsä poikkeaa tämän toisen henkilön työskentelystä. Näin oppilas näkee mihin asioihin hänen on panostettava tullakseen paremmaksi. Oppilas voi muistella aiempaa työskentelyään ja miettiä, mitä hän on oppinut. Peilaamisen seurauksena oppilaan itsetietoisuus ja itsekriittisyys kehittyvät. [5]

Tutkimisessa oppilas joutuu keksimään uusia tapoja, strategioita ja erilaisia keinoja lähestyä ongelmia. Oppilas joutuu kehittämään itse kysymyksiä

ja ongelmia, joita hän pyrkii ratkaisemaan. Hän joutuu miettimään asioita eri näkökulmasta, kuin tilanteessa jossa hän saa tehtävän mestarilta. Oppilaan muodostaessa omia ongelmia ratkaistavaksi, kehittyy oppilaan taito määritellä ja mallintaa tehtävän vaatimat askeleet. Hän pystyy käymään läpi ongelman vaatimat ominaisuudet ja mitä oikein pitää osata kyseisen ongelman ratkaisuun. Samalla oppilaan taito itsenäiseen työskentelyyn kehittyy. [5]

3 Kognitiivinen oppipoikamalli tietojenkäsittelytieteen opetuksessa

Kognitiivisen oppipoikamallin opetusmenetelmiä voidaan soveltaa käytännössä erilaisissa ympäristöissä. Käymme läpi muutamia tilanteita ja tapoja, kuinka menetelmiä on sovellettu käytännössä ja miten se on sujunut.

3.1 Kognitiivinen oppipoikamalli teoreettisen tietojenkäsittelytieteen opettamisessa

Opetuksen kehittäminen perinteisestä opetuksesta kognitiivisen oppipoikamallin suuntaan on tuottanut positiivisia tuloksia muun muassa Potsdamin yliopistolla Saksassa, missä erään teoreettisen kurssin läpäisyprosentti on kasvanut kognitiivisten oppipoikamallin menetelmien käyttöön ottamisen jälkeen. Kurssin opetussuunnitelmaan kuuluuvat säännölliset kielet, kontekstivapaat kielet, erityyppisiä kielioppeja, automaatteja ja Turingin kone. Kurssilla on ollut vuosittain 150-300 osallistujaa. Kurssille osallistuvat opiskelijat ottavat kurssin normaalisti kolmannella tai neljännellä lukukaudellaan. [8]

Kurssin opetusmalli sisälsi alunperin 135 minuuttia viikottaisia luentoja, joita laitoksen henkilökunta luennoi. Luentojen perusteella oppilaat saivat viikottain kotitehtäviä, jotka oli ratkaistava henkilökohtaisesti ja palautettava tarkastettavaksi. Kurssi sisälsi 90 minuuttia laskuharjoitustilaisuuksia kahden viikon välein. Niissä oli paikalla 25-30 opiskelijaa ja niissä käytiin aiemman viikon tehtävät läpi, mikä mahdollisti oppilaiden tekemien ratkaisujen tarkastelun. Kurssin lopussa on kirjallinen tentti, joka ratkaisi kurssin läpäisyn. [8]

Tehostaakseen opetusta ottivat kurssivastaavat käyttöön laskuharjoitustilaisuuksissa ratkaistavat tehtävät. Oppilailla oli tarkoitus suorittaa nämä ylimääräiset harjoitukset laskuharjoitustilaisuuden aikana. Tuutorit pyrkivät rohkaisemaan paikalla olevia opiskelijoita keskustelemaan erilaisista aiheista ja he myös kyselivät kysymyksiä. Oppilaat saivat viikottain noin viisi oikeinväärin tietovisa kysymystä liittyen edellisviikon asioihin. Näiden kysymysten avulla oppilaiden oli tarkoitus tarkistaa ymmärryksensä käytyyn asiaan ja ne toimivat samalla lämmittelynä oikeiden harjoitusten parissa työskentelyyn.

[8]

Kurssia järjestettäessä on huomattu, että on todella tärkeää laittaa kotitehtävien palauttaminen pakolliseksi, koska muutoin oppilaat eivät työskentele tarpeeksi säännöllisesti pärjätäkseen lopputentissä. Säännöllisen työskentelyn kannustamiseksi on opiskelijoita kannustettu tiimityöhön. Opiskelijat saavat tehdä kotitehtävät 2-4 hengen ryhmissä ja palauttaa ne kirjoitetussa muodossa. Jokainen palautettu tehtävä käydään läpi ja opiskelijat saavat pisteitä suoritetuista tehtävistä. Heidän on saatava 50% kokonaispisteistä osallistuakseen lopputenttiin. Tämän tyyppinen malli kannustaa oppilaita työskentelemään säännöllisesti, paneutumaan aihealueeseen ja tekemään tehtäviä. Harjoitellakseen koetilannetta, on kurssin puolivälissä eräänlainen välikoe, joka käsitellään kuitenkin laskuharjoitusten tapaan, se ei vaikuta kurssinläpäisyyn. Välikoe sisältää kuitenkin tehtäviä, jotka ovat vaikeudeltaan samantasoisia kuin kurssikokeessa olevat ja niitä on yhtä paljon. Opiskelijat näkevät kyseisen kokeen perusteella, miten heidän ymmärryksensä ja työskentelynsä suhtautuu kurssinvaatimustasoon ja heillä on aikaa skarpata kurssin loppua kohden. [8]

Kurssin harjoituksista pyrittiin tekemään näkyvämpiä oppilaille. Mallintamista tuotiin opetuksiin opastustilaisuuksissa, joissa oppilaita tuettiin oikea-aikaisesti pitämällä vahva yhteys harjoitusten, kotitehtävien ja loppukokeen välillä. Oppilaita myös pyrittiin valmentamaan harjoitustilaisuuksissa ja antamalla heille palautetta heidän palauttamista kotitehtävistä. Samaan aikaan loppukokeiden taso pidettiin samana kuin ennenkin. Uusien opetuskäytäntöjen käyttöön ottamisen jälkeen epäonnistumisprosentit pienenivät noin 10 prosenttiyksiköllä. Kurssin kehittämisen seurauksena sen epäonnistumisprosentti pysyi kokoajan alle 10% kaksi vuotta peräkkäin. Tulokset osoittivat, että on mahdollista pienentää epäonnistumisprosentteja teoreettisen tietojenkäsittelytieteen kursseissa muuttamalla opettamistapoja ja samalla pitämällä vaatimustason normaalilla tasolla. Kurssilla ei ole kuitenkaan käytössä kaikkia kognitiivisen oppipoikamallin keinoja, joten tuloksissa on parantamisen varaa. [8]

Teoreettisessa tietojenkäsittelytieteessä suuret epäonnistumismäärät ovat yleinen ongelma ainakin Euroopan ja Pohjois-Amerikan yliopistoissa. Oppilailla on vaikeuksia sisältää teoreettisten kurssien asiaa niitten abstraktin ja teoreettisen luonteen vuoksi. [8]

3.2 Tehostettu oppipoikamalli

Tehostettu oppipoikamalli (Extreme Apprenticeship) on kognitiivisen oppipoikamallin laajennus. Siinä ovat vahvassa asemassa tekemisen painottaminen, luentojen kyseenalaistaminen ja opettaja-oppilas yhteistyö [9][14]. Parhaita tehostetun oppipoikamallin ominaisuuksia on, että sitä pystyy soveltamaan opiskelijamääriltään isoillakin kursseilla. Sitä on sovellettu ainakin 67, 44, 192 ja 147 oppilaan tietojenkäsittely kursseilla Helsingin yliopiston

tietojenkäsittelytieteen laitoksella ja se on toiminut onnistuneesti [15].

Tehostetussa oppipoikamallissa on monia tärkeitä arvoja, joita painotetaan kaikissa kurssin toimintojen yhteydessä. Oppilaitten on saatava jatkuvaa palautetta opettajilta ja palautteen on toimittava molempiin suuntiin. Opettaja antaa palautetta, joka kannustaa oppilaita, osoittaa heidän etenemisensä tehtävässä ja edistää heidän oppimistaan. Oppilas voi puolestaan antaa palautetta opettajalle, jotta opettaja pystyy kehittymään tiedon ja taitojen eteenpäin siirtämisessä. Opettaja pystyy seuraamaan palautteen avulla kuinka oppilaat etenevät ja, minkä tyyppisten asioiden omaksumisessa oppilailla on ongelmia. Opettaja pystyy muuttamaan opetustapaansa suuntaan, joka auttaa oppilaita enemmän. Jokaisen oppilaan on harjoiteltava taitoja, niin kauan kuin on tarve, että haluttu taito omaksutaan ja poikkeuksia ei tehdä yksilöiden kohdalla. Lopullinen tavoite kaikella neuvonnalla ja opetuksella on se, että oppilaasta lopulta tulee mestari. Tehostetun oppipoikamallin käyttö voi helposti kärsiä, jos perusarvoja ei noudateta.[14]

Kaksisuuntainen jatkuva palaute tekee oppimisprosessista merkityksellistä ja tehokasta. On huomattavasti tehokkaampaa, jos oppilas saa palautetta, että hänen työskentelynsä etenee oikeaan suuntaan. Jotta oppilas saa palautetta täytyy mestarin olla tietoinen oppilaan tekemisistä eli oppilaan onnistumisista ja kohtaamista haasteista. [14]

Sovellettua oppipoikamallia voi soveltaa myös online-ympäristön kautta esimerkiksi MOOC:issa (Massive Open Online Course). MOOC:ien menestymisen takana ovat opetus ja oppimisen tehokkuus, jonka takana on suoritusten arvostelu. Perinteiset tavat arvostella suorituksia eivät ole skaalattavia ja ne eivät anna oikein ajoitettua tai interaktiivista palautetta oppilaille. [12]

Helsingin yliopistolla on käytössä ollut seuraava toimintamalli tietojenkäsittelytieteen kursseilla, joilla tehostettu oppipoikamalli on ollut käytössä. Opettajan on otettava huomioon muutamia tärkeitä seikkoja opettaesaan asioita. Hänen on vältettävä liiallista yksittäisiin asioihin takertumista luennoilla, koska luentojen pitää kattaa vain sen verran tietoa, että tehtävissä pääsee alkuun. Opettajan on pyrittävä näyttämään esimerkkejä, jotka liittyvät tehtäviin, jotka oppilaat saa tehtäväkseen. Tällöin oppilaat pystyvät yhdistämään luennoilla oppimiaan malleja ja käytänteitä omiin tehtäviinsä. Tehtävien tekeminen on alettava mahdollisimman aikaisin ja tehtäviä on oltava paljon, jotta heille syntyy vahva rutiini koodin kirjoittamisesta ja he saavat motivaatiota. Rutiinin saamisen kannalta tehtävien määrän pitää olla iso ja niissä on oltava jonkin verran toistoa, jotta asiat jäävät kunnolla mieleen. Niissä on oltava myös selvät ohjeistukset kuinka aloittaa ratkaisemaan tehtävää. Tehtävät tehdään pajassa, jossa on paikalla ohjaajia, jotka pystyvät edistämään oppilaan oppimisprosessia oppimisen oikea-aikaisella ohjaamisella. Tehtävät on jaettu pieniksi osiksi, mikä asettaa opetuksessa välitavoitteita. Pienet askeleet takaavat sen, että oppilaat tuntevat oppivansa ja edistyvänsä jatkuvasti. Tehtävien on oltava välttämättömyys läpipääsylle, koska kurssilla oppiminen pohjautuu tehtävien tekoon. Oppilaita kannattaa rohkaista etsi-

mään tietoa itsenäisesti, koska kaikkea mahdollista ei aina käsitellä luennoilla tai materiaalissa, joten on tärkeää osata etsiä tietoa muualtakin. [15]

Mallintaminen tulee näkyviin oppimateriaalien ja luentojen kautta. Näillä kahdella tavalla voidaan oppilaille näyttää kuinka asiat rakentuvat ja muodostuvat. Ohjelmointi on taito, joka vaatii paljon harjoittelua. Luennot ja materiaalit eivät painota kaikkea ohjelmointikieleen liittyviä asioita ja luentoja pidettiin vain kaksi tuntia viikossa. Oppilaille annettiin vain hyviä käytänteitä ja keinoja, joidenka avulla he pystyvät selviämään tehtävistä. Kaikki oppimateriaali, mitä luennoilla näytettiin oli oppilaiden saatavissa netissä. Materiaali seuraa tehtävien rakennetta, joten oppilaat pystyvät seuraamaan materiaalia samalla, kun he tekevät tehtäviä. Materiaali antaa oppilaille oppimisen oikea-aikaista tukea, joka toimii yhteydessä oikean tekemisen kanssa. Materiaalin ja luentojen päätarkoitus on esittää oppilaille työstettyjä esimerkkejä, joissa esitetään askel askeleelta kuinka jokin ongelma ratkaistaan. Tämmöinen lähestymistapa auttoi oppilaita tunnistamaan hyviä tapoja ratkaista ohjelmointiin liittyviä ongelmia jo luentojen aikana. [15]

On selvää, että kurssilla olevilla opiskelijoilla suurin osa ajasta kuluu ohjelmointitehtävien tekemiseen. Suuri tehtäviin käytetty aika kehittää rutiinia. Tehtävissä käytettiin hyväksi myös kahta oppimisen oikea-aikaisen tukemisen keinoa Output- and Main-driven ohjelmointia. Nämä kaksi keinoa antavat ylimääräistä tukea oppilaille. [15]

Output-driven ohjelmointi on samantyylistä kuin Test-driven Development, missä testit on rakennettu ennen koodia. Tehtävät näyttävät oppilaalle syötteen, joka on tavoite saada aikaan ohjelman valmistuessa. Ohjelma voi kysyä esimerkiksi käyttäjältä korkeutta ja leveyttä, jonka jälkeen näytölle toimivassa ohjelmassa tulostuu ascii-merkeistä muodostuva neliö, jossa on syötetty leveys ja korkeus. Käyttäjä pystyy tulostuvaa syötettä tarkkailemalla näkemään toimiiko ohjelma oikein ja, missä on mahdollisia virheitä. Tehtävä itsessään antaa oppilaalle palautetta etenemisestä ja ohjaa oikeaan suuntaan. Main-driven Programming on sovellettu Output-driven ohjelmoinnista. Tehtäväpohja sisältää valmista koodia ja, jotta ohjelma saadaan toimimaan joutuu sitä ohjelmoiva tekemään ainakin yhden uuden luokan, jotta ohjelma toimii halutusti. [15]

Tehostettu oppipoikamalli on hyvä tapa opettaa asioita, joissa vaaditaan rutiinia ja hyvien käytänteiden opettelua mestarilta. Oppimisen oikea-aikainen tukeminen yhdistettynä määrättyihin arvoihin ja käytänteihin on tuottanut onnistuneita tuloksia Helsingin yliopistolla ohjelmoinnin perus- ja jatkokursseilla, missä tärkein ilmiö on kurssien keskeyttämismäärien laskeminen. Tehostetun oppipoikamallin uskotaan auttavan oppilaita, koska jatkuva palaute ja oppimisen oikea-aikainen tukeminen viedään äärimmäiselle tasolle. Tässä tilanteessa täysin uudet oppilaat, joilla työskentely ei ole tehokasta ja jotka normaalisti lopettaisivat kurssin kesken saavat tarpeeksi tukea ja motivaatiota suorittavat kurssin loppuun. Avainasemassa tässä opetuksen muodossa ovat tehtävät, kun tehtäviä on paljon, on niiden oltava

samalla mielekkäitä. Toisinaan tehtävät voivat vaikuttaa negatiivisesti motivaatioon, jos tehtävät eivät ole sidoksissa todellisiin asioihin ja ne eivät tue oppimisprosessia merkityksellisellä tavalla. Suurinosa nimettömästä oppilaitten antamasta palautteesta osoitti, että oppimistä pidettiin motivoivana ja palkitsevana. Eräs kommentti oppilaalta "The best thing on the course was the amount of exercises and exercise groups and the availability of teachers. It was very rewarding to be on a course where you could understand the course content by simply working diligently. Making mistakes also helped to learn things."[14]

Helsinginyliopiston tietojenkäsittelytieteen laitoksella tehostettu oppipoikamalli otettiin ensimmäisen kerran käyttöön kevätlukukaudella 2010, jolloin sitä käytettiin opetuksessa ohjelmoinnin perusteet- ja ohjelmoinnin jatkokursseilla. Kurssikoe ja kurssin opetettavien asioiden sisältö oli sama kuin edeltävinä vuosina ja kurssin tuloksia verrattiin edeltäviin kahdeksaan vuoteen. Molempien kurssien lopussa oli kurssikoe, joka oli paperikoodausta ja siitä oli saatava 50% kokonaispisteistä läpäistäkseen kurssin. Edellä mainittujen seikkojen ansiosta tulosten pitäisi olla verrattavissa aiempiin kurssitoteutuksiin. Pitkän ajan keskiarvo kurssinläpäisystä sisältäen kevään 2010 oli syksyisin 58.5% ja keväisin 43.7%. Yksi mahdollisista eroavaisuuksista syksyn ja kevään välillä on se, että pääaineopiskelijoita on kurssilla enemmän prosentuaalisesti syksyllä kuin keväällä, ja pääaineopiskelijoilla on oletettavasti vahvempi tausta ohjelmoinnista. Läpäisy prosentti ohjelmoinninperusteissa oli 2010 keväällä korkeammalla kuin koskaan aiemmin eli 70.1% osallistuneista, toiseksi isoimman läpipääsyprosentin ollessa 65.4%. Samoin ohjelmoinninjatkokurssilla syksyisisin keskiarvo kurssin läpipääsyprosenteista oli 60.1% ja keväisin 45.3%. Keväällä 2010 kurssinläpäisi 86.4% oppilaista, mikä oli selvästi korkeampi kuin aiemmin. [14]

3.3 Kognitiivinen oppipoikamalli lasten opetuksessa

Olen toiminut ohjaajana Helsingin yliopiston tietojenkäsittelytieteen laitoksen järjestämillä lapsille tarkoitetuissa peliohjelmointikerhoissa ja leireillä vuoden 2012 kesästä lähtien. Syksyllä 2012 ja keväällä 2013 kerho järjestettiin kerran viikossa. Syksyllä 2013 ja keväällä 2014 kerhoja oli kahdelle erilliselle ryhmälle. Yksittäinen kerho on kestoltaan kaksi tuntia. Kerhoissa on käynyt vaihtelevasti 20-25 lasta. Peliohjelmointileirejä olin ohjaamassa 2012 kesällä kaksi kappaletta, 2013 kesällä viisi kappaletta ja 2014 kesällä olen ohjaamassa viidellä leirillä.

Kerho-ohjaajat ovat opiskelijoita, jotka ovat suorittaneet tietojenkäsittelytieteen opintoja. Ohjaajilla ei ole aiempaa kokemusta opettamisesta, joten opettaminen ei ole entuudestaan tuttua. Toimintaan osallistuneet ohjaajat ovat pysyneet samoina. Ohjaajat saavat palkkaa työskentelystään.

Lasten opetus on tapahtunut pääosin 2-D ympäristöllä nimeltään Scratch ja se on suunnattu 8-16 -vuotiaille [7]. Scratch on maksuton ja sii-

nä luodut projektit tallentuvat verkkoon, joten lapset pystyvät jatkamaan projektejaan vapaa-ajallaan. Sen toiminta pohjautuu skripteihin ja siinä käytetään "vedä ja pudota" -tekniikkaa (Drag and Drop), joten lasten ei tarvitse keskittyä syntaksiin [7]. Syntaksin välttäminen motivoi kiinnostusta ohjelmointiin [7]. Ohjelma kääntyy aina ja koodia ei tarvitse itse kirjoittaa. Scratchissa pystyy vaihtamaan eri kielille, joten vieraiden kielten osaaminen ei ole tarpeellista ja se vaatii vain vähäistä matematiikan ymmärtämistä [7].

Toimintamallina kerhossa on se, että yksittäistä peliä lapset tekevät aina kaksi viikkoa kerhohjaajien ohjaamana. Ohjaajat suunnittelevat etukäteen, mikä peli tehdään milloinkin. Pelien teemat vaihtelevat ja niissä opetettavat asiat pyritään pitämään monipuolisina, jotta jokaisella kerralla lapset oppisivat jotakin uutta. Kun yksittäisen pelin tekemistä on ohjattu kaksi viikkoa pidetään lyhyt esittelytilaisuus, jossa lapset voivat halutessaan esitellä toisilleen tuotoksiaan, jotka poikkeavat toisistaan hieman heidän oman panostuksensa ja aktiivisuudensa mukaan.

Kerhojen ohjaus tapahtuu pajaluokassa, jota käytetään yleisesti yliopistolla pajaohjelmointiin. Ohjaajia on vaihtelevasti kaksi tai kolme. Ohjaajilla on kolme roolia ja ohjaajat vaihtelevat rooleja keskenään säännöllisesti.

Ensimmäisen ohjaajan tehtävä on kierrellä, katsoa lasten työskentelyn etenemistä, kysellä ja avustaa heitä ja pitää heidän huomionsa opetuksessa. Tämä ohjaaja keskittyy valmentamiseen, oppimisen oikea-aikaiseen tukemiseen ja oppilaiden artikuloinnin harjoittamiseen. Hän antaa muille ohjaajille palautetta, jos jokin asia on jäänyt lapsille epäselväksi.

Toinen ohjaaja toimii luokan edessä ja mallintaa kyseisen kerran pelin ohjelmointia lapsille. Mallintaminen tapahtuu kyselemällä lapsilta, mitä peli vaatii toiminnallisilta ominaisuuksiltaan seuraavaksi ja kuinka sellainen saadaan aikaan skriptien avulla. Tilanteessa, jossa lapset eivät osaa vastata kysymykseen suoraan, ohjaaja kysyy johdattelevia kysymyksiä ja ohjaa ajattelua oikeaan suuntaan kunnes lapset saavat ymmärryksen asiasta edes jollakin tasolla. Peliä tehdään pienissä osissa eteenpäin, kunnes peli on valmis eli siinä on edetty pisteeseen, jonka ohjaajat ovat etukäteen määritelleet.

Kolmas ohjaaja tekee peliä tietokoneella sitä mukaa kuin toinen ohjaaja mallintaa sitä. Tämä kolmannen ohjaajan tekemä peli näkyy lapsille videotykin kautta, jonka kaikki lapset näkevät ja pystyvät seuraamaan mitä peliin lisätään. Kun pelin mallintaminen luokan edessä ei ole käynnissä, kaikki kolme ohjaajaa kiertelevät luokassa ja tekevät saman tyyppistä tarkkailua kuin ensimmäisen ohjaajan rooliin kuuluu.

Olemme keväällä 2014 ottaneet käyttöön harjoitteen, jossa lapset joutuvat artikuloimaan projektiinsa kommenttityökalulla kuinka peli toimii ja mitä niissä oikein tapahtuu. Näin lapset harjoittelevat artikulointia kirjoittamisen muodossa ja kertaavat mahdollisesti skripteistä, mitä pelin skripteissä oikein tapahtuu. Ohjaajat pystyvät jälkikäteen tarkastelemaan lasten kommentteja ja tekemään niistä johtopäätöksiä. Näitä kommentteja ei ole vielä tarkasteltu, mutta ne ovat arvokasta tietoa oppimisen tutkimiseen.

4 Käytännön esimerkkejä opetustapojen soveltamisesta

Kognitiivista oppipoikamallia pystytään soveltamaan monilla eri menetelmillä. Oikea menetelmä on valittava tilannekohtaisesti, jotta se tukisi oppimista tehokkaasti. Käymme läpi keinoina pariohjelmoinnin, pajaohjelmoinnin, työstetyt esimerkit ja kysymysten muotoilun.

4.1 Pariohjelmointi

Pariohjelmoinnissa kaksi henkilöä työskentelee yhdessä samalla työpisteellä. Henkilöistä toinen on toteuttaja(driver), joka kirjoittaa koodia ja toinen henkilö on varmentaja(navigator), joka tarkastelee koodia sitä mukaa, kuin ajaja sitä kirjoittaa. Ajaja ja tarkkailija vaihtelevat vuoroja keskenään säännöllisesti eli kummatkin joutuvat olemaan eri rooleissa. He miettivät yhdessä, miten ongelmia pitää ratkaista ja miten asiat tulee tehdä. [17]

Pariohjelmoinnissa on kognitiivisen oppipoikamallin kannalta tärkeitä aspekteja. Siinä tulee vahvasti esille varsinkin mallinnus, peilaaminen ja artikulointi. Williams & Kessler [17] ovat esittäneet pariohjelmoinnin suurimpia hyötyjä, jotka muodostuvat yhteistyön seurauksena.

Parit yhdistävät yksilöllisiä taitojaan ja hyödyntävät yhteisesti ideoitaan ja kokemuksiaan ongelmien ratkaisemiseen, prosessi tunnetaan myös nimellä "pair brainstorming". He ilmaisevat mallintamisen ja artikuloinnin avulla ajatuksensa toiselle, minkä kautta he oppivat toisiltaan uusia tapoja lähestyä ongelmia ja tehdä asioita. Ajatukset on saatava toiselle ymmärrettävään muotoon. Samalla kun omia ajatuksiaan muotoilee sanoiksi toiselle tapahtuu oppimista. Ajatusten muotoilua sanoiksi on harjoiteltava, koska osapuolten on päästävä ymmärrykseen siitä, mitä toinen tarkoittaa selittäessään asioita. Tarkastellessaan toisen ajatuksia ja ideoita tapahtuu peilaamista. [17]

Parien välillä syntyy myönteisesti vaikuttavaa suorituspainetta. Pareina työskentelevillä henkilöillä on taipumuksena työskennellä nopeammin ja järkevämmin, koska he eivät halua pettää kumppaninsa luottamusta. Samalla koodi noudattaa todennäköisemmin määrättyjä standardeja. He oppivat luottamaan toisiinsa, joten he pystyvät myöntämään virheensä ja kysymään toiselta apua tarvittaessa. He pystyvät kysymään toisiltaan toimiiko jokin osa koodista niin kuin pitääkin. [17]

Pariohjelmoinnissa syntyy koodia, joka sisältää vähemmän virheitä, koska kirjoitettavaa koodia on tarkkailemassa kaksi silmäparia yhden sijaan. Molemmat osapuolet esittävät toisilleen kysymyksiä ja vastailevat koodin toiminnasta, joten ongelmia tarkastellaan useammasta näkökulmasta. Mahdolliset virheet huomataan tehokkaammin kuin yksin ohjelmoidessa. [17]

Parit jakavat tietämystään keskenään. He joutuvat kokoajan kommentoimaan ja jakamaan näkökulmia erilaisiin tilanteisiin. Luottamusta syntyy sitä mukaa, mitä enemmän parit oppivat toisistaan. Toisen tunteminen kehittää

ilmapiiriä ja motivaatiota työskennellä. Parit pystyvät jakamaan tietonsa avoimin mielin toiselle. [17]

Pariohjelmoinnissa tapahtuu tutkimista, kun parit yrittävät miettiä keskenään millaiset lähestymiskeinot olisivat parhaita eri ongelmiin ja he joutuvat muotoilemaan toisilleen lähestymistapoja ongelmiin. Siinä tapahtuu samalla valmennusta molemminpuolisen tiedonvaihtamisen muodossa. [17]

Yksi pariohjelmoinnin tarkoituksista on parantaa taitoja, joten virheitä ei tarvitse hävetä ja ne pystyy myöntämään. Parit voivat korjata toistensa huonoja tekniikoita ja tapoja sekä samalla korvata ne paremmilla. Kommunikointi on oleellista ja parien välillä ei saa kulua pitkiä aikoja ilman suullista kommunikointia, koska ohjelmointi tehdään yhteistyössä. Pitää kuunnella ja ymmärtää mitä parilla on sanottavana ja on tehtävä töitä toisen ymmärtämisen eteen. Työn lopputulos on yhteinen, joten molemmat henkilöt ovat vastuussa koodista, joten osapuolten on oltava tarkkaavaisia, kun uutta koodia tuotetaan. [17]

4.2 Pajaohjelmointi

Pajaohjelmoinnissa opiskelijoille on varattu ajankohtia, jolloin he pystyvät mennä tekemään ohjelmointitehtäviä sosiaaliseen ympäristöön pajaluokkaan. Pajaluokassa on läsnä muita opiskelijoita ja vähintään yksi pajaohjaaja eli kisälli, joilla on ohjelmointitehtävien tekemiseen vaaditut tiedot ja taidot hallussa. [15]

Opiskelijat voivat halutessaan kysyä ohjaajilta apua tehtäviin, jos niissä tulee vastaan ongelmia. Pajaohjelmoidessa ollaan sosiaalisessa kanssakäymisessä muiden ihmisten kanssa. Opiskelijat pystyvät vertailemaan vastauksiaan kavereittensa tai toisten oppilaiden vastauksiin, jotka ovat paikalla pajaluokassa. Ohjelmoidessaan pajassa oppilas saa tukea tehtävien tekemiseen, eikä tehtävissä tule ylipääsemättömiä esteitä, koska apua ja neuvoja on aina saatavilla. [9] [15]

Kisällit ovat yleensä alkuvaiheissa opintojaan ja heidän opettamiskokemuksensa rajoittuvat monesti vain oppilaiden tuutorointiin [15]. Kisällit kartuttavat kokemusta ja oppivat itse neuvoessaan oppilaita [15]. He saavat itselleen lisää rutiinia perustehtävistä ja uusia näkökulmia lähestyä ongelmia [15]. Kisällit saavat pientä korvausta tekemästään työstä, mikä motivoi heitä opettamisessa [9]. Osalla pajaohjaajista saattaa loppua taidot kurssien loppuakohti mentäessä, tällöin kokeneemmat pajaohjaajat ottavat enemmän vastuuta [15].

Pajaohjelmoinnissa kognitiivisen oppipoikamallin menetelmiä tulee esiin seuraavilla tavoilla. Valmentaminen tulee näkyviin, kun pajaohjaaja seuraa oppilaan etenemistä. Hän pystyy samaistumaan oppilaan tilanteeseen ja pyrkii ohjaamaan tätä kohti tehtävien ratkaisua ja puhtaampaa koodia. Samalla valmentajan roolissa oleva pajaohjaaja pyrkii antamaan oppilaalle oikea-aikaista tukea oppimisessa. Pajaohjaaja pystyy esittämään ideoita op-

pilaille ja näyttämään tapoja ratkaista ongelmia. Oppilas pystyy peilaamaan omaa ratkaisuaan ja ajatteluaan tämän kokeneemman henkilön ajatusmaailmaan. Pajaohjaaja mallintaa nämä ajatukset oppilaalle tasolla, jolla tehtävän oppimiskokemus on edelleen saatavissa [15]. Oppilas joutuu myös artikuloimaan omat ajatuksensa omin sanoin, kun hän selventää ratkaisuaan kisällille. Kuunnellessaan oppilasta pajaohjaaja saa paremman kuvan, minkä tyyppistä apua oppilas kaipaa ongelmatilanteessa. Tutkiminen tulee esiin siinä, kun pajaohjaaja köyttää opettamisessa itse keksimiään esimerkkejä.

Helsingin yliopistolla ohjelmoinnin perusteet ja ohjelmoinnin jatkokurssia varten pajaohjaajille on annettu selkeät toimintaohjeet. Kaikkia henkilöitä joilla on ongelmia tehtävissä tulee auttaa. Ohjaajien on kierreltävä luokassa ja tarkkailtava opiskelijoiden työskentelyä. Kaikki henkilöt eivät välttämättä uskalla tai halua pyytää apua syystä tai toisesta, vaikka ovatkin jääneet jumiin johonkin tehtävään. Ohjaajat eivät saa antaa tehtäviin suoria vastauksia, vaan heidän on ohjattava oppilaita kohti ongelman ratkaisua antamalla heille sopivan pieniä neuvoja, jotta tehtävän teko edistyy ja oppilaat pystyvät lopulta itse löytämään ratkaisun ja kokemaan onnistumisen. Tässä tilanteessa tulee näkyviin kognitiivisen oppipoikamallin aspektista vahvasti näkyviin oppimisen oikea-aikainen tukeminen. Ohjaajat tarkkailevat neuvomisen ohella oppilaiden koodin ulkoasua ja neuvovat oppilaita tekemään siistimpää koodia. Koodin ulkoasun siistiminen on vaikeaa, ilman esimerkkejä ja neuvoja. Ohjaajien on painotettava opiskelijoille, että oikea vastaus ei ole tarpeeksi, vaan ohjelmointityylistä on saatava ymmärrettävää ja ylläpidettävää koodia. Ohjaaja voi opastaa oppilaita oikeaan suuntaan. Vaikka pajassa olisi hiljaista, on silti pyrittävä olemaan aktiivinen ja käyttää aika oppilaiden neuvomiseen. [15]

4.3 Työstetyt esimerkit

Työstetyssä esimerkissä opettaja mallintaa tehtävän oppilaille vaiheittain, jolloin pienten askeleiden merkitys kokonaisuuden syntymiseen tulee ymmärrettäväksi [4][14]. Työstettyjen esimerkkien avulla voidaan tehokkaasti opettaa monimutkaisia ongelmanratkaisutaitoja ja ne auttavat sisäisesti organisoitujen muistirakenteiden muodostamisessa ja oppimisen kehittämisessä [4].

Kognitiivisen oppipoikamallin kannalta työstetyissä esimerkeissä tärkeimpään asemaan tulevat mallintaminen, peilaaminen ja oppimisen oikea-aikainen tukeminen. Opettaja pyrkii valmentamaan oppilaita muodostamalla esimerkeistä oppilaiden kyseiselle taitotasolle sopivia ja ohjaamalla heitä oikeaan suuntaan tulevaisuuden tehtävien ratkaisussa. Vaiheiden aikana oppilaita pyritään oikea-aikaisesti tukemaan sen verran, että oppimiskokemuksesta tulee mielekäs. Oppilas voi peilata omia vastauksiaan näihin työstettyihin esimerkkeihin ja miettiä, mitä on mahdollista parantaa ja mitä on tehty oikein.

Työstettyjä esimerkkejä kannattaa olla samaan opetettavaan asiaan liittyen enemmän kuin yksi. Yhden esimerkin kannattaa kuvata yksinkertaista tilannetta, josta ymmärtää opetettavan asian perusidean. Toinen esimerkki puolestaan voi olla monimutkaisempi kuin ensimmäinen. Oppilaat saavat näin enemmän irti opetettavasta asiasta, koska he saavat erilaisia näkökulmia. Samalla ensimmäisen esimerkin ymmärtäneet henkilöt pystyvät syventymään asiaan tarkemmin. Huonosti menestyvilläl oppilailla on taipumus jättää työstetyt esimerkit katsomatta, jos ne eivät ole tavanomaisista ongelmista, koska he kokevat yleensä nämä tehtävät pelottavina ja etäisinä. [4]

Helsingin yliopiston tietorakenteet-kurssin yhteydessä tehdyt kyselyt näyttävät, että työstetyt esimerkit ovat suositumpia oppimisen lähteenä kuin perinteiset luennot. Opiskelijat oppivat enemmän tutkimalla esimerkkejä kuin tekemällä samat tehtävät itse. [11]

Kaikkiin tilanteisiin työstetyt esimerkit eivät kuitenkaan sovi. Sitä on testattu henkilöihin, joilla on kokemusta opetettavasta asiasta. Näillä henkilöillä opetuksen vaikutukset eivät olleet kovin tehokkaita. Mitä enemmän oppilas tietää opetettavasta asiasta, sitä vähemmän työstetyt esimerkit vaikuttavat. [4]

4.4 Kysymysten muotoilu

Kysymysten muotoilua käytetään ohjelmoinnin opetuksessa. Sillä, onko kysymys miksi- vai kuinka-muodossa, on merkitystä oppilaan ongelmanratkaisuprosessin kannalta [6].

Kysymysten muotoilulla on merkitystä siihen, kuinka oppilas joutuu muuttamaan ajattelutapaansa ratkaistessaan ongelmaa. Esitämme kaksi esimerkkiä kahdesta erilaisesta kysymyksen muotoilusta, kuinka-kysymys: "ohjelmoi ohjelma, joka tulostaa halutun kokoisen neliön"ja miksi-kysymys: "ohjelmoi ohjelma, joka tulostaa halutun kokoisen neliön ja kerro miksi ohjelma toimii. [6]

Kuinka-kysymykset luovat oppilaalle harhaanjohtavan käsityksen millä tasolla asioita on tärkeää osata, koska oppilas pystyy kokeilemaan erilaisia keinoja toteuttaa tehtävää ja saada aikaan ratkaisun puolivahingossa ymmärtämättä miksi se toimii. Oppilas opettelee muodostamaan ratkaisun kysymyksiin ulkomuistista, ilman syvempää ymmärrystä miksi ja miten hänen vastauksensa oikeasti toimivat. Tämä kannustaa oppilaita ratkaisemaan ongelmia ymmärtämättä niitä, miksi nähdä ylimääräistä vaivaa kurssin läpipääsyn eteen, jos kurssin tehtävät on ratkaistavissa vähemmälläkin. [6]

Miksi-kysymyksissä oppilas joutuu rakentamaan syvempää ymmärrystä opetettuun asiaan. Hän joutuu käsittelemään tehtävää syvemmin ja käymään läpi lopputulokseen johtavia vaiheita tarkemmin. Kun oppilas ymmärtää asian, hän pystyy perustelemaan vastauksiaan ja muodostamaan yhtenäisen konseptin lopputulokseen johtavista askeleista. Miksi-kysymyksiin tehtyjä vastauksia on kuitenkin haastavampi arvostella, koska vastaukset poikkea-

vat oppilaiden välillä ja heidän ajatusprosessinsa ovat yksilöllisiä. Oppilas joutuu perustelemaan vastauksensa eli hän joutuu artikuloimaan ratkaisuun johtavia vaiheita ja mallintamaan näitä vaiheita vaihe vaiheelta. Artikuloidessaan ratkaisuaan käy oppilas läpi sen vaiheita syvemmin. Oppilas käy läpi oman ongelmanratkaisuprosessinsa vaiheet ja pystyy samalla peilaamaan sitä muiden työskentelyyn. Peilaamisen kautta hän hahmottaa, mitkä vaiheet johtavat onnistumiseen tai epäonnistumiseen. [6]

Miksi-kysymykset ovat tärkeitä myös opettajan kannalta, koska tällöin opettaja pystyy näkemään oppilaan ajattelua ja antamaan hänelle oikea-aikaista tukea ja ohjaamaan oppilasta oikeaan suuntaan. Opettaja pystyy muuttamaan opetusta ja painottamaan enemmän asioita, joissa oppilaalla on puutteita. [6]

Artikkelissa "The Abstraction Transition Taxonomy- Developing Desired Learning Outcomes through the Lens of Situated Cognition"tarkasteltiin jonkin Yhdysvalloissa sijaitsevan yliopiston seitsemän erillisen tietojenkäsittelytieteen alkuvaiheen kurssien opetusmateriaaleja. Opetusmateriaalien tehtävistä noin viidesosa oli miksi-kysymyksiä ja loput kuinka-kysymyksiä. Kuinka-kysymyksissä opiskelija voi joutua kirjoittamaan toimivan koodin ohjelmointitehtävään, mutta miksi-kysymyksissä hän joutuu perustelemaan miksi koodi toimii kyseisessä tehtävässä. Kurssikokeissa puolestaan miksi-kysymyksiä oli ainoastaan 0%-15% kaikista kysymyksistä. [6]

Johtopäätöksinä he totesivat miksi-kysymyksiä olevan liian vähän. Heillä ei ole konkreettisia todisteita muotoiltujen kysymyksien toimivuudesta, mutta heidän mukaansa monet tekijät tukevat ideaa niiden toimivuudesta. Miksi-kysymyksiä tukevia tekijöitä löytyy kognitiivisen oppipoikamallin ja oppimisen tilannesidonnaisuudesta. [6]

5 Yhteenveto

Pitää kirjottaa tääki vielä

Lähteet

[1] Bareiss, Ray ja Radley, Martin: *Coaching via Cognitive Apprenticeship*. Teoksessa *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, sivut 162–166, New York, NY, USA, 2010. ACM, ISBN 978-1-4503-0006-3. `http://doi.acm.org/10.1145/1734263.1734319`.

[2] Black, Toni R.: *Helping Novice Programming Students Succeed*. J. Comput. Sci. Coll., 22(2):109–114, joulukuu 2006, ISSN 1937-4771. `http://dl.acm.org/citation.cfm?id=1181901.1181922`.

[3] Brown, John Seely, Collins, A ja Newman, SE: *Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics.* Knowing, learning, and instruction: Essays in honor of Robert Glaser, 487, 1989.

[4] Caspersen, Michael E. ja Bennedsen, Jens: *Instructional Design of a Programming Course: A Learning Theoretic Approach.* Teoksessa *Proceedings of the Third International Workshop on Computing Education Research*, ICER '07, sivut 111–122, New York, NY, USA, 2007. ACM, ISBN 978-1-59593-841-1. http://doi.acm.org/10.1145/1288580.1288595.

[5] Collins, Allan, Brown, John Seely ja Holum, Ann: *Cognitive apprenticeship: making thinking visible.* American Educator, 6:38–46, 1991.

[6] Cutts, Quintin, Esper, Sarah, Fecho, Marlena, Foster, Stephen R. ja Simon, Beth: *The Abstraction Transition Taxonomy: Developing Desired Learning Outcomes Through the Lens of Situated Cognition.* Teoksessa *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, sivut 63–70, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1604-0. http://doi.acm.org/10.1145/2361276.2361290.

[7] Hulsey, Caitlin, Pence, Toni B. ja Hodges, Larry F.: *Camp CyberGirls: Using a Virtual World to Introduce Computing Concepts to Middle School Girls.* Teoksessa *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, sivut 331–336, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2605-6. http://doi.acm.org/10.1145/2538862.2538881.

[8] Knobelsdorf, Maria, Kreitz, Christoph ja Böhne, Sebastian: *Teaching Theoretical Computer Science Using a Cognitive Apprenticeship Approach.* Teoksessa *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, sivut 67–72, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2605-6. http://doi.acm.org/10.1145/2538862.2538944.

[9] Kurhila, Jaakko ja Vihavainen, Arto: *Management, Structures and Tools to Scale Up Personal Advising in Large Programming Courses.* Teoksessa *Proceedings of the 2011 Conference on Information Technology Education*, SIGITE '11, sivut 3–8, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-1017-8. http://doi.acm.org/10.1145/2047594.2047596.

[10] Larkins, D. Brian, Moore, J. Christopher, Rubbo, Louis J. ja Covington, Laura R.: *Application of the Cognitive Apprenticeship Framework to a Middle School Robotics Camp.* Teoksessa *Proceeding of the 44th ACM*

Technical Symposium on Computer Science Education, SIGCSE '13, sivut 89–94, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-1868-6. http://doi.acm.org/10.1145/2445196.2445226.

[11] Luukkainen, Matti, Vihavainen, Arto ja Vikberg, Thomas: *A Software Craftsman's Approach to Data Structures.* Teoksessa *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, sivut 439–444, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1098-7. http://doi.acm.org/10.1145/2157136.2157266.

[12] Tillmann, Nikolai, De Halleux, Jonathan, Xie, Tao, Gulwani, Sumit ja Bishop, Judith: *Teaching and Learning Programming and Software Engineering via Interactive Gaming.* Teoksessa *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, sivut 1117–1126, Piscataway, NJ, USA, 2013. IEEE Press, ISBN 978-1-4673-3076-3. http://dl.acm.org/citation.cfm?id=2486788.2486941.

[13] Vihavainen, Arto, Luukkainen, Matti ja Kurhila, Jaakko: *Multi-faceted Support for MOOC in Programming.* Teoksessa *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, sivut 171–176, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1464-0. http://doi.acm.org/10.1145/2380552.2380603.

[14] Vihavainen, Arto, Paksula, Matti ja Luukkainen, Matti: *Extreme Apprenticeship Method in Teaching Programming for Beginners.* Teoksessa *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, sivut 93–98, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0500-6. http://doi.acm.org/10.1145/1953163.1953196.

[15] Vihavainen, Arto, Paksula, Matti, Luukkainen, Matti ja Kurhila, Jaakko: *Extreme Apprenticeship Method: Key Practices and Upward Scalability.* Teoksessa *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, sivut 273–277, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0697-3. http://doi.acm.org/10.1145/1999747.1999824.

[16] Vihavainen, Arto, Vikberg, Thomas, Luukkainen, Matti ja Pärtel, Martin: *Scaffolding Students' Learning Using Test My Code.* Teoksessa *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, sivut 117–122, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2078-8. http://doi.acm.org/10.1145/2462476.2462501.

[17] Williams, Laurie ja Kessler, Robert: *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002, ISBN 0201745763.

[18] Xu, Anbang, Huang, Shih Wen ja Bailey, Brian: *Voyant: Generating Structured Feedback on Visual Designs Using a Crowd of Non-experts*. Teoksessa *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, sivut 1433–1444, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2540-0. http://doi.acm.org/10.1145/2531602.2531604.