Kognitiivisen oppipoikamallin menetelmien soveltaminen ohjelmoinnin opetuksessa

Pessi Moilanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 24. huhtikuuta 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	24. huhtikuuta 2014	21

Tiivistelmä — Referat — Abstract

:D

Avainsanat — Nyckelord — Keywords
jee, jepa, jeejee

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, kognitiivisen oppipoikamallin erilaiset menetelmät ja kuinka niitä pystyy soveltamaan käytännössä. Samassa yhteydessä käsittelemme muutaman tilanteen, joissa kognitiivista oppipoikamallia on käytetty tietojenkäsittelytieteen opettamisessa. Käymme läpi myös ohjelmoinnin opetusta yleisesti ja tarkastelemme kognitiivisen oppipoikamallin toimivuutta. Lopuksi kerromme kuinka Helsingin yliopistolla

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.

Mestari on henkilö, joka on alansa ammattilainen eli erittäin kokenut tietyssä asiassa. Tämä kokemus on seurausta pitkäaikaisesta ja kovasta harjoittelusta. Oppipoika on henkilö, joka on noviisi opetettavassa asiassa ja pyrkii oppimaan mestarin opastuksella. Mestarilla voi olla useampia oppipoikia samaan aikaan. Mestari pyrkii opettamaan ja antamaan oppipojille töitä suoritettavaksi, joidenka seurauksena oppipojille karttuisi ammattitaitoa harjoiteltavasta asiasta.

Perinteinen oppipoikamalli (apprenticeship) on peräisin vanhoilta ajoilta, siinä mestari ja oppipoika tekevät töitä yhdessä. Perinteisessä oppipoikamallissa on mestarin suorittama työ jotakin konkreettista, josta seuraa jokin lopputuote. Työtehtävä voi olla esimerkiksi haarniskan takominen, kakun leipominen tai paidan kutominen. Mestari ohjeistaa oppipoikaa työn suorituksessa soveltaen erilaisia opetustekniikoita suoritettavasta työstä riippuen.

Oppipoika saa suoritettavakseen mestarin avustuksella pienempiä tehtäviä, jotka ovat sopivia oppipojan sen hetkiselle taitotasolle. Syynä tähän on monissa tilanteissa tehtävien tärkeys ja vaikeus, mikä ei jätä tilaa virheille. Täysin uusi oppipoika tuskin saa tarpeeksi luottoa mestarilta suorittaakseen tehtäviä, joista epäonnistumisen yhteydessä koitua suuria vahinkoja. Kisällille on myös mielekkäämpää tehdä pienempiä tehtäviä, koska tällöin hän näkee oman työnsä jäljen ja pystyy nauttimaan onnistumisesta useammin. Pienten tehtävien myötä oppipojalle kertyy kokemusta, jota hän pystyy hyödyntämään laajemmissa tehtävissä. Oppipoika voi oppia kuitenkin vain rajallisen määrän mestarin alaisena. Tarpeeksi opittuaan hän on valmis täysin itsenäiseen työskentelyyn.

Oppipojalle yksi oppimistavoista on vierestä tarkkaileminen, kun mestari suorittaa jotakin työtehtävää. Mestari pystyy antamaan työnvaiheissa erilaisia neuvoja, miten ja miksi asiat tapahtuvat. Tarpeeksi harjoiteltuaan ja opittuaan oppipoika ei pysty kartuttamaan ammattitaitoaan saman mestarin

alaisena. Tällöin oppipojasta on tullut kisälli.

1.1 Perinteinen opetus

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.

Suurin osa yliopistoita käyttävät edelleen perinteistä tapaa opettaa ohjelmoinnin peruskurssia. Ohjelmoinnin peruskurssilla on ollut kaikkialla taipumuksena sisältää suuria kurssin keskeytys prosentteja, kuten Helsingin yliopistolla keskeytys prosentin keskiarvo pidemmältä aikaväliltä on ollut 45%. Perinteinen tapa opettaa kattaa luentoja, kotona suoritettavia tehtäviä ja mahdollisia harjoitustilaisuuksia, joissa tehtävien malliratkaisut esitetään ja käydään läpi.

Luennoilla on taipumus 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 oikeasti 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. Opetus psykologiassa tunnetusti kyseiset tilanteet eivät ole parhaimpia aloittelijoille, jotka yrittävät oppia kognitiivisesti haastavia asioita. [14]

Kun ohjelmointia aletaan opettamaan 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 on alettu myöntämään, 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 opettelemaan 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. [5][13]

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

Oppilaat saattavat oppia huonoja ohjelmointikäytänteitä jos he joutuvat tekemään ohjelmointitehtävät täysin itsenäisesti. [14]

Tehokkain tapa kerätä tietoa ja taitoa kognitiivisen tieteellisen tutkimuksen mukaan tulee aktiivisen ongelmanratkaisun aikana. [1]

Liian moni oppilas ei osaa tehdä järkeviä ohjelmia opiskeltuaan vuoden tai kaksi ohjelmointia. [4] Tekstikirjat eivät paljasta prosessia kuinka ongelmia pitää lähteä ratkaisemaan ja miten niissä pitää ajatella. [4]

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]

2 Kognitiivinen oppipoikamalli

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. Tämmöisiä 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 tavalla, mikä on oppipojalle ymmärrettävissä. 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. Kognitiivisessä oppipoikamallissa on suuri painotus itse työskentelyssä lopputuotteen valmistumisen sijaan [14]*katso tästä 9,10. Kognitiivinen oppipoikamalli optimoidaan valmennuksen ja ohjauksen saantia oppilaille [14]. Kognitiivinen oppipoikamalli vaikuttaa myönteisesti opiskelijan motivaatioon ja opiskelun mukavuuteen, mikä puolestaan vaikuttaa huomattavasti oppimiseen [15].

2.1 Roolit

Opettaja

Oppilas

2.2 Kognitiivisen oppipoikamallin opetusmenetelmät

Kognitiiviseen oppipoikamalliin on määritelty kuusi erilaista opetusmenetelmää. Opetusmenetelmät tukevat, hyödyntävät ja ovat vahvasti sidoksissa toisiinsa. Opetusmenetelmiä ovat mallintaminen, valmentaminen, oppimisen oikea-aikainen tukeminen, artikulointi, peilaaminen ja tutkiminen.

Mallintamisessa (Modelling) työn suorittamisen eri vaiheet pyritään saamaan näkyviin oppilaalle. Työn suorittaa normaalisti 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.

Valmentamisessa (Coaching) opettaja tarkkailee oppilaan työskentelyä ja antaa hänelle henkilökohtaisia neuvoja ja pyrkii avustamaan oppilasta kriittisillä. Opettaja pyrkii samaistumaan oppilaan osaamiseen ja pyrkii ohjaamaan tätä oikeaan suuntaan. Opettaja voi rakentaa oppilaalle tehtäviä, jotka ovat oppilaan osaamiselle sopivia.

Oppimisen oikea-aikainen tukeminen (scaffolding) käsittää erilaisia tekniikoita ja strategioita oppilaan oppimisen tukemiseen. Oppilaalle voidaan antaa tehtäviä, jotka sisältävät tekniikoita, joita oppilas ei entuudestaan osaa.

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 pystyy antamaan sekä suullista ja kirjoitettua palautetta oppilaalle, jonka perusteella oppilas pystyy parantamaan työskentelyään [1]. Opettajan täytyy suhteuttaa käytettävissä oleva aika opetettavian asioiden laajuuten. Pieniä yksityiskohtia ei pysty hiomaan loputtomiin, jos aikaa on rajallisesti. Annetun avun määrän ja neuvojen on oltava sopivia eivätkä ne saa pilata tehtävän tuomaa oppimismahdollisuutta.

Opettaja vähentää antamaansa tukea hiljalleen (fading), jättäen oppilaalle enemmän vastuuta. Nämä keinot kehittävät oppilaan itsetietoisuutta ja oppilas oppii korjaamaan itse virheitään, joidenka seurauksena hän ei tarvitse enää niin paljoa ulkopuolista apua.

Artikuloinnissa (articulation) oppilaiden on muutettava ajatuksensa, tie-

tonsa 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. Opettaja voi pyytää oppilaita puhumaan ääneen oppilaan suorittaessa jotakin tehtävää, näin oppilas joutuu muotoilemaan ajatuksensa sanoiksi.

Kognitiivinen oppipoikamalli jakaa opettamisen kolmeen eri vaiheeseen: mallintamiseen, oppimisen oikea-aikaiseen tukemiseen ja tuen vähentämiseen [14].

Peilaamisessa (reflection) 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 paremmiksi. Oppilas voi muistella aiempaa työskentelyään ja miettiä, mitä hän on oppinut. Peilaamisen seurauksena oppilaan itsetietoisuus ja itsekriittisyys kehittyvät.

Tutkimisessa (exploration) 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.

2.3 Tilanteet

2.4 Vaiheet

3 Kognitiivinen oppipoikamalli ohjelmoinnin opetuksessa

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

3.1 Kognitiivinen oppipoikamalli teoreettisen tietojenkäsittelytieteen opetuksessa

Opetuksen muuttaminen 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 kuuluu ainakin säännölliset kielet, konteksti vapaat kielet, eri tyyppisiä 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 viikottaisia kotitehtäviä, jotka oli ratkaistava henkilökohtaisesti ja palautettava tarkastettavaksi. Se sisälsi 90 minuuttia laskaritilaisuuksia, joka toinen viikko, niissä oli paikalla 25-30 opiskelijaa. Laskaritilaisuuksissa käytiin edellis viikon tehtävät läpi, joka mahdollistaa oppilaiden tekemien ratkaisujen oikeellisuuden tarkastuksen. Kurssin lopussa oli kirjallinen tentti, joka ratkaisee kurssin läpäisyn. [8]

Tehostaakseen opetusta ottivat kurssivastaavat käyttöön laskaritilaisuuksissa ratkaistavat tehtävät. Oppilailla oli tarkoitus suorittaa nämä ylimääräiset harjoitukset laskaritilaisuuden aikana. Tuutorit pyrkivät rohkaisemaan paikalla olevia opiskelijoita keskustelemaan erilaisista aiheista ja he myös kyselivät kysymyksiä. Oppilaat saivat viikottain noin viisi oikein-väärin tietovisa kysymystä liittyen edellisviikon asioihin. Näiden kysymysten avulla oppilaiden on tarkoitus tarkistaa ymmärryksensä käytyyn asiaan ja ne toimivat samalla lämmittelynä oikeiden harjoitusten parissa työskentelyyn. [8]

Kurssia järjestettäessä on huomattua, että on todella tärkeää laittaa kotitehtävien palauttaminen pakolliseksi, koska muulloin 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 siis 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 siis tekemään näkyvämpiä oppilaille. Mallintamista tuotiin opetuksiin opastus sessioissa, joissa oppilaita tuettiin oikeaaikaisestipitämällä vahva yhteys harjoitusten, kotitehtävien ja loppukokeen välillä. Oppilaita myös pyrittiin valmentamaan harjoitustilaisuuksissa ja antamalla heille palautetta palautetuista kotitehtävistä. Samaan aikaan loppukokeiden taso pidettiin samallatasolla, kuin ennenkin ja epäonnistumis prosentti pysyi kokoajan alle 10% kaksi vuotta putkeen. Tulokset osoitti-

vat, että on mahdollista pienentää epäonnistumis prosentteja teoreettisen tietojenkäsittelytieteen kursseissa muuttamalla opettamistapoja ja samalla pitämällä vaatimustason normaalilla tasolla. Uusien opetuskäytäntöjen käyttöön ottamisen jälkeen epäonnistumis prosentit pienenivät keskimääräisesti noin 10 prosentilla. Kurssilla ei ole kuitenkaan käytössä kaikkia kognitiivisen oppipoikamallin keinoja, joten tuloksissa on parantamisen varaa. [8]

3.2 Tehostettu oppipoikamalli

Tehostettu oppipoikamalli (Extreme Apprenticeship) on kognitiivisen oppipoikamallin laajennus. Se painottaa arvoja, joihin kuuluvat tekeminen, luentojen hyödyllisyyden kyseellistäminen ja jatkuva palaute mestarin ja oppipojan välillä [14].

Kaksisuuntainen jatkuva palaute tekee oppimisprosessista merkityksellistä ja tehokasta. Se on huomattavasti tehokkaampaa, jos oppilas saa palautetta, että hän edistyy ja työ etenee oikeaan suuntaan. Jotta oppilas saa palautetta täytyy mestarin olla tietoinen oppilaan tekemisistä eli oppilaan onnistumisista ja kohtaamista haasteista.

Parhaita tehostetun oppipoikamallin ominaisuuksiin kuuluu, että sitä pystyy soveltamaan opiskelijamääriltään isoillakin kursseilla. Sitä on sovellettu ainakin 67, 44, 192 ja 147 oppilaan tietojenkäsittely kursseilla ja se on toiminut onnistuneesti [15].

Tehostetussa oppipoikamallissa on monia tärkeitä arvoja, joita painotetaan kaikissa kurssin toimintojen yhteydessä. Oppilaat oppivat tekemällä, kuten yleensäkkin harjoittelu tekee mestariksi. 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 omaksuttaan ja poikkeuksia ei tehdä yksilöiden kohdalla. Lopullinen tavoite kaikella neuvonnalla ja opetuksella on se, että oppilaasta lopulta tulee mestari. [14]

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

Tehostetun oppipoikamallin käyttö voi helposti kärsiä, jos perusarvoja ei noudateta [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 2 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 siis 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. Software Craftsmanship yhteisön idea "Code Katas", jossa pienet tehtävät edistävät ohjelmoijia kehittämään taitojaan harjoittelun ja toiston kautta. ”practising the solution to a Kata un- til the steps and keystrokes became like second nature, and you could do them without thinking. In this way, you can internalize the process/technique you are practicing until it is under your fingers”*artikkelin 13 lähde*. Kaikissa tehtävissä oli lyhyt tekstillinen selitys kuinka ohjelman tulee toimia. 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 siis oppilaalle palautetta etenemisestä ja ohjaa oikeaan suuntaan. [15]

Main-driven Programming on laajennus Output-driven ohjelmoinnille. Tehtäväpohja sisältää valmista koodia ja, jotta ohjelma saadaan toimimaan joutuu sitä ohjelmoiva tekemään ainakin yhden uuden luokan, jotta ohjelma toimii halutusti. [15]

Jatkuva palaute on tärkeää, koska se mahdollistaa nopean arvioinnin ja jatkuvan etenemisen tunteen.

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 diligenttly. Making mistakes also helped to learn things."[14]

XA:ta voi soveltaa myös online ympäristöön eli MOOCISSA. Suurin osa yliopisto tason ohjelmointikursseista ovat sekoitus nettimateriaaleja ja paikallisia luentoja. [13]

3.3 Tutkimustuloksia tehostetun oppipoikamallin toimivuudesta

Helsinginyliopiston tietojenkäsittelytieteen laitoksella tehostettu oppipoikamalli otettiin ensimmäistä kertaa käyttöön kevätlukukaudella 2010, jolloin sitä käytettiin opetuksessa ohjelmoinnin perus- ja jatkokursseilla. Kurssikoe ja kurssin opetettavien asioiden sisältö oli sama kuin edeltävinä vuosina ja kurssin tuloksia verrattiin edeltäviin kahdeksaan vuoteen. Molempien kurs-

sien lopussa oli kurssikoe, joka oli paperikoodausta ja siitä oli saatava 50% pisteistä läpäistäkseen kurssi. Edellä mainittujen seikkojen ansiosta tulosten pitäisi olla verrattavissa aiempiin kurssitoteutuksiin. Pitkänajan 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 aiemmin eli 70.1% toiseksi isoimman läpäisy prosentin ollessa 65.4%. Samoin ohjelmoinninjatkokurssilla syksyisisin keskiarvo kurssinläpäisystä oli syksyisin 60.1% ja keväisin 45.3%. Keväällä 2010 kurssinläpäisi 86.4% oppilaista, mikä oli selvästi korkeampi kuin aiemmin korkein läpäisy m ä 67.3%. [14]

4 Ohjelmoinnin opetus

5 Käytännön esimerkkejä opetustapojen soveltamisesta

5.1 Pariohjelmointi

Pariohjelmoinnissa kaksi henkilöä työskentelee yhdessä samalla työpisteellä. Henkilöistä toinen on ajaja(driver), joka kirjoittaa koodia ja toinen henkilö on tarkkailija(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 pitäisi ratkaista ja miten asiat tulisi tehdä.

Williams & Kessler [17] ovat määritelleet pariohjelmoinnin suurimpia hyötyjä, jotka muodostuvat yhteistyön seurauksena verrattuna normaaliin ohjelmointiin.

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.

Parit yhdistävät yksilöllisiä taitojaan ja hyödyntävät yhteisesti ideoitaan ja kokemustaan ongelmien ratkaisuun. Prosessi tunnetaan myös nimellä "pair brainstorming".

Parit oppivat luottamaan toisiinsa, joten he pystyvät myöntämään virheensä ja kysymään toiselta apua tarvittaessa. He pystyvät kysymään toisiltaan toimiiko jokin osa koodista niin, kuin pitääkin.

Pariohjelmoinnissa syntyy koodia, joka sisältää vähemmän virheitä. Kirjoitettavaa koodia on tarkkailemassa kaksi silmäparia yhden sijaan, joten mahdolliset virheet huomataan tehokkaammin kuin yksin ohjelmoidessa.

Parina koodissa olevien virheiden etsiminen johtaa yleensä ratkaisun

löytymiseen. Parit joutuvat artikuloinnin avulla ilmaisemaan ajatuksensa toiselle, jota kautta vastaus ongelmaan mahdollisesti löytyy. Molemmat esittävät toisilleen kysymyksiä ja vastailevat koodin toiminnasta, joten ongelmaa tarkastellaan useammasta näkökulmasta.

Parit jakavat tietämystään keskenään. He joutuvat kokoajan kommentoimaan ja jakamaan näkökulmia erilaisiin tilanteisiin.

Parien välille syntyy luottamusta sitä mukaa, mitä enemmän parit oppivat toisistaan. Toisen tunteminen kehittää ilmapiiriä ja motivaatiota työskennellä. Parit pystyvät jakamaan tietonsa avoimin mielin toiselle.

Tiedon siirtäminen. Tilanteissa, joissa pari vaihtuu säännöllisesti tietävät ohjelmoijat tarkemmin systeemin kokonaisvaltaisesta toiminnasta.

Williams & Kessler [17] ovat määritelleet seitsemän erilaista tapaa joilla edistää pariohjelmointina työskentelyä. Taukojen pitäminen on kannattavaa, koska välillä pitää rentoutua ja saada ajatukset nollattua. Ohjelmointiin liittyvät tehtävät voivat olla henkisesti todella väsyttäviä, joten pienet tauot ovat suositeltavia. Nöyryyden harjoittelu on tärkeää, koska yhdessä työskennellessä on pystyttävä myöntämään omat virheensä ja pystyttävä katsomaan toisen lähestymistapoja avoimin mielin. Pariohjelmoinnin yksi tarkoituksista on parantaa taitoja, joten virheitä ei tarvitse hävetä ja ne voi myöntää. Kommunikointi on oleellista. Parien välillä ei saisi kulua kovin pitkiä aikoja ilman suullista kommunikointia, koska ohjelmointi tehdään yhteistyössä. Kommunikoinnissa on hyödynnettävä peilaamista ja artikulointia. Pitää kuunnella ja ymmärtää mitä parilla on sanottavana, on tehtävä töitä toisen ymmärtämisen eteen. Työn lopputulos on yhteinen, joten molemmat henkilöt ovat vastuussa koodista, joten molempien on oltava tarkkaavaisia, kun uutta koodia tuotetaan.

Pariohjelmontia pystyy tekemään etäntyönä, jolloin henkilöt eivät ole samassa työpisteessä. Luodakseen pariohjelmointimaisen tilanteen, on oppilaiden käytettäviä ohjelmia, joilla he pystyvät jakamaan näytön kuvaa toiselle. "Assessing the Effectiveness of Distributed Pair Programming for an Online Informatics Curriculum"[6] artikkelissa kerrottiin, että oppilaat eivät olleet niin tyytyväisiä etänä toteutettuun pariohjelmointiin kuin perinteisiin pariohjelmointiin.

Artikulointi tulee ilmi pariohjelmoinnissa, kun parien on ilmaistava itseään ja jakaa ajatuksensa parilleen, koska ohjelmointi tehdään yhteistyössä. Ajatusten muotoilua sanoiksi on harjoiteltava, koska parien on päästävä ymmärrykseen siitä, mitä toinen yrittää tarkoittaa selittäessään asioita. Samalla kun omia ajatuksiaan muotoilee sanoiksi toiselle tapahtuu oppimista. He mallintavat asiat toiselle ymmärrettävään muotoon, pilkkomalla ne tarpeeksi pieniksi palasiksi, jotta toinen pystyy ne ymmärtämään. Parit valmentavat toisiaan tietyssä mielessä, he seuraavat toistensa toimintoja ja tukevat toista tarvittaessa. Tarkastellessaan toisen ajatuksia, pystyy pari peilaamaan niitä omiin tietoihin ja taitoihin. Peilaamalla pari oppii uusia tekniikoita ja saa uusia näkökulmia lähestyä erilaisia tilanteita. Parit pystyvät korjaamaan

toistensa huonoja tekniikoita ja tapoja ja samalla korvaamaan ne paremmilla. Pariohjelmoinnissa tapahtuu tietynlaista tutkimista, kun parit yrittävät miettiä keskenään millaiset lähestymiskeinot olisivat parhaita eri ongelmiin ja he joutuvat muotoilemaan erilaisia ongelmatilanteita. Pariohjelmoinnissa tapahtuu siis tiedon vaihtoa ja siirtoa, parit opettavat toisiaan.

5.2 Pajaohjelmointi

Jotenki järkevästi näitte kahe viitteen käyttö [15][9]. Pajaohjelmoinnissa opiskelijoille on varattu erillisiä ajankohtia, jolloin he pystyvät mennä tekemään ohjelmointitehtäviä pajaluokkaan. Pajaluokassa on aina läsnä yksi tai useampi kokeneempi henkilö, joilla on tehtävien sisältämät tiedot ja taidot hallussa.

Opiskelijat voivat halutessaan kysyä ohjaajilta apua tehtäviin, jos niissä tulee vastaan ongelmakohtia. He pystyvät myös vertailemaan vastauksiaan kavereittensa tai toisten oppilaiden vastauksiin. Pajaohjelmoidessa ollaan sosiaalisessa kanssakäymisessä muiden ihmisten kanssa. Ohjelmoidessaan pajassa oppilas saa tukea tehtävien tekemiseen, eikä niistä tule ylipääsemätöntä estettä, koska apua ja neuvoja on aina saatavilla.

Helsingin yliopistolla ohjelmoinnin perusteet ja ohjelmoinnin jatkokurssia varten pajaohjaajille on annettu selkeät toimintaohjeet. Kaikkia henkilöitä joilla on ongelmia tehtävissä tulisi auttaa, tämä vaatii ohjaajien kiertelyä luokassa ja oppilaiden työskentelyn tarkkailua. Ohjaajat voivat alkaa suosimaan tai karttamaan tiettyjä oppilaita. Kaikki henkilöt eivät välttämättä uskalla tai halua pyytää apua syystä tai toisesta, vaikka olisivatkin 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 itse löytämään ratkaisun 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.

Pajaohjaajat ovat yleensä alkuvaiheissa opintojaan ja heidän opettamiskokemuksensa rajoittuvat monesti vain oppilaiden tuutorointiin. Osalla pajaohjaajista ei ole muuta kokemusta kuin ohjelmoinnin peruskurssin tiedot. Yhteistä pajaohjaajille on kuitenkin asenne, he haluavat ja ovat valmiita auttamaan toisa oppilaita. He ovat tietyssä mielessä kisällejä, jotka ovat maailmalla etsimässä lisää kokemusta. Pajaohjaajat oppivat itse neuvoessaa oppilaita. He saavat lisää kokemusta perustehtävistä ja mahdollisesti uusia

näkökulmia lähestyä ongelmia. He saavat myös kokemusta opettamisesta ja neuvomisesta ja heidän kommunikaatiotaidot paranevat. Pajaohjaajat saavat myös pientä korvausta tekemästään työstä, mikä motivoi heitä opettamisessa. Osalla pajaohjaajista alkaa loppua taidot kurssien loppuakohti mentäessä, tällöin kokeneemmat pajaohjaajat ottavat enemmän vastuuta. Pajaohjaajat pystyvät tarvittaessa kysymään myös apua IRC-kanavalla.

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 ja näyttämään erilaisia tapoja ratkaista tietyntyyppisiä ongelmia, näistä tilanteista oppilas pystyy peilaamaan omaa ratkaisuaan ja ajatteluaan tämän kokeneemman henkilön ajatusmaailmaan. Pajaohjaaja mallintaa nämä ajatukset oppilaalle tasolla, jolla tehtävän oivalluskokemus ei mene pilalle, mutta auttaa oppilasta kuitenkin eteenpäin. Oppilas joutuu myös artikuloimaan omat ajatuksensa omin sanoin, kun hän selittää jotakin pajaohjaajalle. Artikuloidessaan ajatuksiaan pajaohjaaja saa paremman kuvan, minkä tyyppistä apua oppilas kaipaa ongelmatilanteessa. Tutkiminen tulee esiin siinä, kun pajaohjaaja pystyy halutessaan käyttää keinona opettamisessa itsekeksimiään esimerkkejä. Pajaohjaaja mallintaa tällöin tehtävän ja millaista ajattelua sen ratkaisu vaatii.

5.3 Työstetyt esimerkit

Työstetty esimerkki on askel askeleelta jonkin läpikäynti, kuinka suorittaa jokin tehtävä tai kuinka ratkaista jokin ongelma. Tehtävä toisinsanoen mallinnetaan vaiheissa, tavalla joista kokonaisuus pyritään ymmärtämään. Opetettava asia havainnollistetaan oppilaalle asian eri vaiheissa, niin että oppilaat ymmärtävät vaiheiden merkitykset ja tärkeydet. Työstettyjen esimerkkien avulla voidaan opettaa monimutkaisia ongelmanratkaisutaitoja ja ne auttavat skeemojen muodostamisessa ja oppimisen kehittämisessä. [4]

Helsingin yliopiston tietorakenteet kurssin yhteydessä tehdyt kyselyt näyttävät, että työstetyt esimerkit ovat suositumpia oppimisen lähteenä, kuin perinteiset luennot. [11].

Opiskelijat oppivat enemmän tutkimalla esimerkkejä, kuin tekemällä samat tehtävät itse. He pystyvät tarkastelemaan ratkaisun erilaiset vaiheet ja muodostamaan niistä skeemoja.

Työstetyt esimerkit soveltuvat opiskelukohteisiin, joissa taidon hankkiminen on oleellista kuten musiikissa, shakissa ja ohjelmoinnissa *hae viite Instructional Design of a Programming Course... viite n.2*. Huonosti menestyvät oppilaat tarkastelevat monesti työstetyt esimerkit ainoastaan jos ne ovat tavanomaisista ongelmista, koska he voivat kokea nämä tehtävät

pelottavina ja etäisinä. Joten jos esimerkki ei ole tälläinen voi käydä niin, että pelkästään paremmat oppilaat katsovat ne läpi *hae viite Instructional Design of a Programming Course... viite n.13 ja [4]*.

Kaikkiin tilanteisiin kyseinen opetusmuoto ei kuitenkaan sovi. Sitä on testattu myös henkilöihin, jotka ovat jo saaneet jonkin verran kokemusta opetetusta asiasta. Näillä henkilöillä opetuksen vaikutukset eivät enää olleet kovin tehokkaita *viite instructional design of a "29"*. Mitä enemmän oppilas tietää opetettavasta asiasta, sitä vähemmän työstetyt esimerkit vaikuttavat.

Työstettyjä esimerkkejä kannattaa olla samaan opetettavaan asiaan liittyen enemmän kuin yksi. Yhden esimerkin kannattaa kuvata yksinkertaista tilannetta, josta opetettavan asian perusidean ymmärtää. Toinen esimerkki puolestaan voi olla monimutkaisempi kuin ensimmäinen. Oppilaat saavat näin enemmän irti opetettavasta asiasta, koska he saavat erilaisia näkökulmia *hae viite Instructional Design of a Programming.. viite 54,22,25*. Samalla ensimmäisen esimerkin ymmärtäneet henkilöt pystyvät syventymään asiaan tarkemmin.

Kognitiivisen oppipoikamallin kannalta työstetyissä esimerkeissä suurimpaan asemaan tulevat mallintaminen ja oppimisen oikea-aikainen tukeminen. Opettaja mallintaa työstetyissä esimerkeissä työn ammattilaisen näkökulmasta ja pyrkii aukaisemaan sen oppilaille, niin että he ymmärtävät sen. Oppilas pääsee näkemään nämä vaiheet ja kuinka pienistä askelista edetään onnistuneeseen lopputulokseen ja hän pystyy rakentamaan mielikuvan kokonaisuuteen johtavista askelista. Opettaja pyrkii valmentamaan oppilaita muodostamalla esimerkeistä oppilaiden kyseiselle taitotasolle sopivia ja ohjaamalla heitä oikeaan suuntaan tulevaisuuden tehtävien ratkaisussa. Kun oppilas tekee samoja taitoja vaativaa työtä, hän pystyy peilaamaan suoritustaan opettajan suoritukseen ja hakea näistä työstetyistä esimerkeistä tukea. Työstetyissä esimerkeissä pyritään luomaan esimerkit niin, että kaikki vaiheet on selitetty tarpeeksi hyvin auki suhteessa oppilaiden taitotasoon. Vaiheiden aikana oppilaita pyritään oikea-aikaisesti tukemaan sen verran, että oppimiskokemuksesta tulee mielekästä. Oppilas voi peilata omia vastauksiaan näihin työstettyihin esimerkkeihin ja miettimään, mitä on mahdollista parantaa ja mitä on tehty oikein.

5.4 Kysymysten muotoilu

Tämä aliluku pohjautuu artikkeliin The Abstraction Transition Taxonomy-Developing Desired Learning Outcomes through the Lens of Situated Cognition[5]. Kerromme kuinka kyseisessä artikkelissa käydään läpi miten kysymysten muotoilua pystyy käyttämään ohjelmoinnin opetuksessa.

Artikkelissa tarkasteltiin jonkin Yhdysvalloissa sijaitsevan yliopiston seitsemän eri CS0-, CS1- ja CS2-ohjelmointikurssien eli ohjelmoinnin peruskurssien opetusmateriaalejen sisältämien tehtävien rakennetta. Jokainen näistä kursseista arvosteltiin kurssikokeen perusteella. Kurssien oppimistavoitteet

pohjautuivat kurssikokeen sisältämiin kysymyksiin.

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 vielä vähemmän. Niitä oli 0%-15% välillä kokonaiskysymys määrästä.

Kuinka-kysymykset voivat luoda oppilaille harhaanjohtavan käsityksen millä tasolla asioita on tärkeää osata kokeessa. Oppilas opettelee muodostamaan vastaukset kysymyksiin ulkomuistista, ilman syvempää ymmärrystä miksi ja miten hänen vastauksensa oikeasti toimivat. Tämä kannustaa oppilaita ratkaisemaan ongelmia ymmärtämättä niitä, koska miksi nähdä ylimääräistä vaivaa ymmärryksen saamiseen, jos tehtävät on ratkaistavissa vähemmälläkin.

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 syvemmin, hän pystyy perustelemaan vastauksiaan ja muodostamaan yhtenäisen konseptin lopputulokseen johtavista askeleista. Miksi-kysymyksiin tehtyjä vastauksia on kuitenkin haastavampi arvostella, koska vastaukset poikkeavat oppilaiden välillä ja heidän ajatusprosessinsa ovat yksilöllisiä.

Johtopäätöksinä he totesivat miksi-kysymyksiä olevan liian vähän. Heillä ei ole konkreettisia todisteita muotoiltujen kysymyksien toimivuudesta, mutta monet tekijät *PITÄÄ ETSIÄ VIITTEET ASD* tukevat ideaa niiden toimivuudesta. Miksi-kysymyksiä tukevia tekijöitä löytyy kognitiivisen oppipoikamallin ja oppimisen tilannesidonnaisuudesta.

Kysymysten muotoilulla on siis merkitystä siihen, kuinka oppilas joutuu muuttamaan ajatteluaan lähestyessään ongelmaa. Oppilas joutuu artikuloimaan ratkaisuun johtavia vaiheita, jolloin hän käy läpi ratkaisuaan syvemmin omin sanoin. Oppilas käy läpi oman ongelmanratkaisuprosessinsa ja pystyy samalla peilaamaan sitä opettajan työskentelyyn. Samalla oppilas pystyy huomaamaan, mitkä vaiheet ovat johtaneet mahdolliseen onnistumiseen tai epäonnistumiseen, joten hän tietää missä on vielä parannettavaa. Oppilas joutuu mallintamaan ratkaisun eri vaiheet, jotta hän pystyy perustelemaan miksi asiat toimivat. Hän pystyy hyödyntämään vastausta muodostaessaan asioita, jotka hän on oppinut kokeneemmalta henkilöltä ja pyrkiä peilaamaan omaa vastaustaan tämän henkilön muodostamaan vastaukseen. Miksi-kysymykset ovat tärkeitä myös opettajan kannalta, koska tällöin opettaja pystyy näkemään oppilaan ajatusmaailman ja antamaan hänelle oikea-aikaista tukea ja ohjaamaan oppilasta oikeaan suuntaan, jos vastaukset ovat hapuilevia.

6 Kognitiivinen oppipoikamalli lasten opetuksessa

Olen ollut yhtenä ohjaajana Helsingin yliopiston tietojenkäsittelytieteen laitoksen järjestäillä 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, molemmille kaksi tuntia per viikko. 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 eivät vaihtele erikoisemmin, vaan ohjaajat ovat ohjaustoiminnassa pitkiä aikoja. Ohjaajat saavat palkkaa työskentelystää.

Toimintamallina kerhossa on ollut 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 muille lapsille pelejään, jotka poikkeavat toisistaan hieman lapsien oman panostuksen ja aktiivisuuden mukaan.

Lasten opetus on tapahtunut pääosin visuaalisella ohjelmointiympäristöllä nimeltään Scratch, mikä on suunnattu 8-16 -vuotiaille. Se on täysin ilmainen ja siinä luodut projektit tallentuvat verkkoon, joten lapset pystyvät jatkamaan luomuksiaan myös vapaa-ajallaan. Scratchin toiminta pohjautuu skripteihin, joiden avulla pelimaailman hahmot saadaan liikkumaan. Skriptit muodostetaan yhdistelemällä erilaisia toiminnallisuuspalikoita, joita ei voi yhdistää toisiinsa jos rikkinäistä koodia syntyisi. Ohjelma siis kääntyy aina ja koodia ei tarvitse itse kirjoittaa. Palikoiden sisältämän tekstikentän kielenkin pystyy vaihtamaan halutulle kielelle, joten muun kuin äidinkielen ymmärtäminen ei ole tarpeellista.

Kerhojenohjaus tapahtuu pajaluokassa, jota käytetään yleisesti yliopistolla pajaohjelmointiin. Ohjaajia on vaihtelevasti kaksi tai kolme. Ohjaajat vaihtelevat rooleja, jotka mainitaan alla.

Ensimmäisen ohjaajan tehtävä on pelkästään kierrellä ympäriinsä, katsoa lasten etenemistä, kysellä lapsilta ohjelman toiminnasta, avustaa lapsia ja pitää lasten huomio opetuksessa. Tämä ohjaaja keskittyy valmentamiseen, oppimisen oikea-aikaiseen tukemiseen ja oppilaiden artikuloinnin harjoittamiseen.

Toinen ohjaaja on luokan edessä ja mallintaa kyseisen kerran pelin ohjelmointia lapsille. Mallintaminen tapahtuu kyselemällä lapsilta, mitä peli vaatii toiminnallisuuksiltaan seuraavaksi ja kuinka tämmöinen toiminnal-

lisuus saataisiin aikaa skriptien avulla. Tilanteessa jossa lapset eivät osaa vastata kysymykseen suoraan, ohjaaja kysyy johdattelevia kysymyksiä ja ohjaa ajattelua oikeaan suuntaan kunnes suurinosa lapsista vaikuttaa ymmärtäneen mistä on kyse. Tällä menetelmällä peliä tehdään pienissä osissa eteenpäin, kunnes peli on valmis eli siinä on edetty pisteeseen, jonka ohjaajat ovat etukäteen määritelleet.

Kolmas ohjaaja on tietokoneella ja tekee peliä sitä mukaan kuin toinen ohjaaja mallintaa sitä. Tämä kolmannen ohjaajan tekemä versio 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 ottaneet käyttöön uuden harjoitteen 2014 keväällä, jossa lapset kirjoittavat jokaisen kerran jälkeen mietteensä projektiin kommentin muodossa. Lapsia on ohjeistettu kirjoittamaan kommenttiin kuinka peli toimii ja mitä skripteissä tapahtuu. Näin lapset harjoittelevat artikulointi kirjoittamisen muodossa ja kertaavat mahdollisesti myös skripteistä, mitä ohjelmassa oikein tapahtuu. Ohjaajat pystyvät myös jälkikäteen tarkastelemaan lasten kommentteja ja tehdä johtopäätöksiä. Näitä kommentteja ei ole tosin vielä luettu käpi, mutta ne ovat varmasti arvokasta tietoa oppimisen tarkasteluun.

7 Yhteenveto

Suunnitellaan ohjelmointi ympäristöjä opetustyökaluiksi. Yleensä ne ovat visuaalisia ympäristöjä, joilla voi tehdä yksinkertaisia pelejä tai animoituja tarinoita. Scratch mahdollistaa vedä-ja-pudota tekniikan, joten oppilaiden ei tarvitse keskittyä syntaksiin. Syntaksin välttäminen motivoi oppilaiden kiinnostusta ohjelmointiin. Scratch on 2-D ympräistö, joka vaatii vain vähäistä matematiikan ymmärtämistä. [7]

Kognitiivinen oppipoikametelmä on hyvä työkalu asioiden opettamiseen varhaisessa vaiheessa ja sen avulla pystyy syventymään asioihin. Kognitiivinen oppipoikamenetelmä keskittyy kehittämään taitoja ja pätevyyttä ympäristössä, jossa on apua tarjolla. Mallintaminen, valmentaminen ja oppimisen oikea-aikainen tukeminen ovat pää tekniikat kognitiivisessa oppipoikamallissa. Ne on suunniteltu niin, että oppilaat saavat rakennettua käsitteellisen mallin ongelman tilasta ja he pystyvät kehittämään kognitiivisia taitoja tarkkailun ja harjoittelun kautta. Molemmat artikulointi ja peilaaminen pyrkivät saamaan oppilaat sisäistämään havainnot ja kokemukset, mutta myös samalla kehittävät uutta tietoa ja ongelmanratkaisutaitoja. Tutkiminen edistää itsenäisyyttä ja rohkaisee itsenäiseen ongelman muodostamiseen ja ratkaisemiseen. Seuraamalla vierestä miten jokin algoritmi toimii, pystyvät oppilaat tekemään samaa roskaa. [10]

Opiskelijalla joka aloittaa ohjelmoinnin on paljon opittavaa alkuun: uusi ohjelma ohjelmointiympäristön muodossa, oikea syntaksi, syntaksin sovellukset, loogiset askeleet, kääntäjän virheviestien ymmärtäminen, oikea ohjelmointi tyyli, toimintojen modularisointi, monimutkaiset konseptit kuten taulukot. Kognitiivinen oppipoikamalli on tehokas tapa opettaa noviiseille ohjelmointia. Se auttaa oppilaita rakentamaan parempaa ymmärrystä konsepteista ja samalla heistä tulee itsenäisempiä kokemuksen myötä. Se kuinka opitaan kirjoittamaan koodia on samantapaista oppimista kuin vieraan kielen oppiminen. Opiskelijan pitää oppia sanoja monille objekteille ja toiminnoille ja kieliopilliset säännöt lauseilla. Todellinen haaste on kuitenkin se kuinka oppilas osaa hyödyntää opittua syntaksia ongelmien ratkaisemiseen. Tässä pisteessä hän tarvitsee oppiakseen jonkin asteen ohjausta tai neuvontaa. Opettajan rooli on antaa neuvoja ja ohjeita tähän uuteen maailmaan. Oppilaiilta pitää myös kysyä keskustelemaan ja selittämään ongelmanratkaisu prosessinsa tai strategiansa ja vertailemaan niitä muiden vastaaviin. Tämän kaltainen peilaaminen auttaa oppilaita ymmärtämään konseptin. Mallintamisen yhteydessä oppilas tarkkailee ja rakentaa mallia konseptia prosessista, millä työ valmistuu. [2]

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]

Ohjelmointi ei ole pelkästään yksinkertaisten syntaksi rakenteiden ja niiden sovellusten tekemistä, vaan käytännön ongelmanratkaisutaitojen harjoittamista tarkoituksellisessa kontekstissa. Oppilaan menestyminen kurssin alkuvaiheessa kehittää oppilaan itsetuntoa ja rakentaa heille ohjelmoinnista rutiinin, mikä auttaa heitä syventymään muuhunkin kuin syntaksiin. Kurssin alkuvaiheen onnistuminen perustuu suureen määrään pieniä ja melko helppoja harjoituksia, joidenka tarkoitus on auttaa ymmärtämään kurssin oppimistavoitteet. Oppilaan oppimisen oikea-aikainen tukeminen on oltava ajoitettu oikein ja ei liian yksityiskohtaista. Kun pajassa on kymmeniä oppilaita on vaikeaa rakentaa kokonaiskuvaa oppilaiden edistymisestä. Jossain asiassa voidaan tulla mestariksi vain jos sitä oikeasti harjoitellaan, niin kauan kuin on tarpeellista. Jotta oppilas voi harjoitella, on oltava tarkoituksellisia aktiviteetteja, kuten harjoituksia. Jatkuva palaute oppijan ja neuvojan välillä, oppijan on saatava varmuus etenemisestään ja siitä, että työ menee haluttuun suuntaan. Jotenka neuvojan pitää olla perillä onnistumisista ja haasteita joita oppija kohtaa kurssin aikana. [16]

Korkeamman tason opetuksessa ohjelmointi nähdään taitona, joka hyötyy suorasta kontaktista, tuesta ja palautteesta henkilöiltä jotka ovat jo mestareita siinä. XA:n avain kohtia on tekemisen painottaminen ja luentojen kyseenalaistaminen ja se keskittyy opettaja-oppilas yhteistyöhön. [9]

Ohjelmoinnin oppiminen on vaikeaa. Nykyään monet myöntävät, että luennointi ei ole paras tapa opettaa ohjelmoimaan, mutta myös harjoitustilai-

suudetkin ovat huonohkoja. Harjoitustilaisuuksissa saa minimaalista ohjausta tehtävien parissa. Aloittelijoille kyseinen lähestymistapa ei ole hyvä. Fadingia tapahtuu kun oppilas alkaa omaksumaan suoritettavaa tehtävää. Henkilökohtaisen avun ei tarvitse olla hirveän paljoa, kunhan oppilasta silloin tällöin ohjattaisiin oikeaan suuntaan. [15]

Ohjelmoinnin johdatuskurssin droppaus luvut on isot, joten on aika selvää, että perinteistä lähestymistapaa opettaa on parannettava. CA laittaa suuren painotuksen valmentamisen optimointiin ja ohjauksen antamiseen oppilaille. Useat tutkimukset osoittavat, että molemmilla motivaatiolla ja mukavuustasolla on huomattava merkitys oppilaan oppimiseen. CA sisältää monia elementtejä, jotka kehittävät molempia, mutta ohjelmointiharjoitusten osuus on merkittävä. Worked examples yksi tehokas tapa mallintaa asioita. Liian vaikeat tehtävät tappavat motivaation. [14]

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] 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`.

[6] Edwards, Richard L., Stewart, Jennifer K. ja Ferati, Mexhid: *Assessing the Effectiveness of Distributed Pair Programming for an Online Informatics Curriculum.* ACM Inroads, 1(1):48–54, maaliskuu 2010, ISSN 2153-2184. http://doi.acm.org/10.1145/1721933.1721951.

[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.