Kognitiivinen oppipoikamalli tietojenkäsittelytieteessä

Pessi Moilanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 28. helmikuuta 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

Kognitiivinen oppipoikamalli tietojenkäsittelytieteessä

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	28. helmikuuta 2014	9

Tiivistelmä — Referat — Abstract

:D

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto **1**

2 Kognitiivinen kisällioppiminen **2**

3 Ohjelmoinnin opetus **3**

4 Opettamistapojen soveltaminen käytännössä **4**
4.1 Muotoillut esimerkit 4
4.2 Kysymysten muotoilu 5
4.3 Pariohjelmointi 6
4.4 Pajaohjelmointi 8
4.5 Perinteiset luennot 8
4.6 Koodausdojot 8

5 Yhteenveto **8**

Lähteet **8**

1 Johdanto

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 enemmän aikaa. Ihmisellä on rajallinen aika oppia asioita, joten oppiminen on tehtävä mahdollisimman tehokkaasti ajankäytön suhteen.

Opettamiseen on olemassa monia erilaisia tekniikoita. Tässä aineessa käymme läpi kisällioppimisen ja kognitiivisen kisällioppimisen perusideat, kognitiivisen kisällioppimisen erilaiset menetelmät ja kuinka niitä pystyy soveltamaan käytännössä. Samassa yhteydessä käsittelemme muutaman tilanteen, joissa kognitiivista kisällioppimista on käytetty tietojenkäsittelytieteen opettamisessa.

Mestari on henkilö, joka on alansa ammattilainen eli erittäin kokenut0 tietyssä asiassa. Mestarilla voi olla oppipoikia, jotka hänen opastuksellaan pyrkivät saamaan kokemusta. Oppipoika eli kisälli on henkilö, joka pyrkii oppimaan mestarilta. Mestari ohjeistaa kisälliä työn suorituksessa soveltaen erilaisia opetustekniikoita. Tarpeeksi harjoiteltuaan ja opittuaan kisällistä tulee mestari.

Perinteisessä kisällioppimisessa (apprenticeship) on mestarin suorittaman työn eri vaiheita konkreettisesti näkyvissä oppipojalle. Oppipojalle yksi oppimistavoista on vierestä tarkkaileminen, kun mestari suorittaa jotakin työtehtävää. Työtehtävä voi olla esimerkiksi haarniskan takominen, kakun leipominen tai paidan kutominen. Mestari pystyy antamaan työnvaiheissa erilaisia neuvoja, miten ja miksi asiat tapahtuvat. Oppilas pystyy tekemään näistä vaiheista muistiinpanoja, joita hän hyödyntää harjoittelussaan.

Kisälli voi saada suoritettavakseen mestarin avustuksella pienempiä tehtäviä, jotka ovat sopivia kisällin sen hetkiselle taitotasolle. Syynä tähän on monissa tilanteissa tehtävien tärkeys, mikä ei jätä tilaa virheille. Täysin uusi kisälli tuskin saa tarpeeksi luottoa mestarilta suorittaakseen tehtäviä, joista voi epäonnistumisen yhteydessä koittua suuria vahinkoja. Pienistä tehtävistä kisälli kerryttää kokemusta. Kokemuksen karttuessa kisälli alkaa tekemään laajempia ja haastavampia tehtäviä.

Oppipoika voi oppia kuitenkin vain rajallisen määrän mestarin alaisena. Tarpeeksi opittuaan on hän valmis täysin itsenäiseen työskentelyyn. Alansa mestariksi päästäkseen on hänen saatava oman alansa killalta tunnustus mestarin taidoista. Tämä tunnustus vaatii työn esittämistä killalle, jonka perusteella arvioidaan onko kisälli valmis mestariksi.

2 Kognitiivinen kisällioppiminen

Kognitiivisessa kisällioppiminen (cognitive apprenticeship) on teoria prosessista, jossa mestari opettaa taitoa kisällille [2]. 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 vaiheet ovat kaavojen pyörittelyt ja ratkaisun ilmestyminen paperille. Mestarin on saatava ajatuksensa näkyviin. Toisinaan ajattelu on kuitenkin abstrahoitunut ja yhdistynyt muihin käsitteisiin, jotka eivät ole oppilaalle vielä tuttuja. Tällöin opettajan on selitettävä asia kisällille, tasolla mikä on kisällille ymmärrettävissä. Ajatusten näkyviin tuontiin on erilaisia tekniikoita, joita voidaan soveltaa tilanteesta riippuen. Kognitiivisessä kisällioppimisessa on samat periaatteet kuin perinteisessä kisällioppimisessa, poikkeavuutena on opetettavien asioiden luonne ja kuinka oppi saadaan perille oppilaalle.

Kognitiiviseen kisällioppimiseen on määritelty kuusi erilaista opetusmenetelmää. Opetusmenetelmät sulautuvat toisiinsa eli ne hyödyntävät ja ovat vahvasti sidoksissa toisiinsa. Mallintamisessa (Modelling) työn suorittamisen eri vaiheet pyritään saamaan näkyviin oppilaalle. Työn suorittaa normaalisti opettaja tai muu oppilasta kokeneempi henkilö. Oppilas pyrkii saamaan kokemusta ja muodostamaan käsitteellissen mallin suoritetuista vaiheista. Oppilas näkee syy seuraussuhteen ja saa käsityksen miten suoritetut vaiheet johtavat onnistuneeseen lopputulokseen.

Kognitiivisessä kisällioppimisessa ongelmana on ajatustyön esille saanti oppilaalle, joten on pyrittävä käyttämään keinoja, joilla oppilas hahmottaa ne. Ongelmanratkaisuprosessia mallintaessa mestari kertoo kussakin työn vaiheessa ääneen mitä hän tekee jamiksi, jolloin oppilaat voivat kirjoittaa ongelmanratkaisuprosessin vaiheet muistiin. Valmentamisessa (Coaching) opettaja tarkkailee oppilaan työskentelyä ja antaa hänelle henkilökohtaisia neuvoja ja pystyy avustamaan oppilasta kriittisillä hetkillä. 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 osaa arvioida oppilaan taitotasoa, kykyä oppia ja käytettävissä olevaa 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 tehtävässä etenemiseen estääkseen jumiin jääminen. 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]. Annetun avun määrä ja neuvojen on oltava sopiva, ne eivät saa pilata tehtävän tuomaa oppimismahdollisuutta, eivätkä ne saa olla liian vähäistä. 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, tietonsa ja ajatusprosessinsa sanoiksi. Artikuloinnin seurauksena oppilaat voivat vertailla paremmin omaa ajatteluaan opettajan ajatteluun ja opettaja pääsee käsiksi oppilaan ongelmanratkaisuprosessiin. Opettaja voi pyytää oppilaita puhumaan ääneen oppilaan suorittaessa jotakin tehtävää, näin oppilas joutuu muotoilemaan ajatuksensa sanoiksi.

Oppilaan voi laittaa pariohjelmoimaan toisen henkilön kanssa, tässä tilanteessa oppilaan on kommunikoitava toisen osapuolen kanssa ja esitettävä tälle ajatuksensa.

Oppilaalle voidaan esittää muotoiltuja kysymyksiä. Kuinka-kysymysten sijaan oppilaille kannattaa esittää miksi-kysymyksiä. Oppilas voidaa pyytää ohjelmoimaan ratkaisu johonkin tehtävään ja sen lisäksi kysyä miksi kyseinen koodi toimii. Oppilas joutuu tehtävän ratkaisun ohella miettimään omaa työskentelyään syvemmin. [4] Peilaamisessa (reflection) oppilas vertailee työskentelyään opettajaan, kokeneempaan henkilöön tai toiseen oppilaasen. 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 itetietoisuus ja itsekriittisyys kehittyy ja hän pystyy vertailemaan ymmärrystään toisten ymmärrykseen. 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. Muodostaessaan omia ongelmia kehittyy oppilaan taito määritellä, käydä läpi ongelman vaatimat ominaisuudet. Samalla oppilaan taito itsenäiseen työskentelyyn kehittyy.

3 Ohjelmoinnin opetus

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

Jakaessa iso kokonaisuus pienempiin osiin saavat oppilaat niistä enem-

män irti, koska he oppivat miksi jokaisella osalla tulee olemaan merkitys lopputuloksen kannalta.[4]

Kun oppilaat alkavat opettelemaan ohjelmointia, on tärkeää tuoda esiin, kuinka ohjelmat muodostuvat pienissä paloissa [7].

4 Opettamistapojen soveltaminen käytännössä

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

4.1 Muotoillut esimerkit

Muotoiltu esimerkki on askel askeleelta läpikäynti, kuinka suorittaa jokin tehtävä tai kuinka ratkaista jokin ongelma eli opettaja mallintaa tehtävän. Opetettava asia havainnollistetaan oppilaalle asian eri vaiheissa, niin että oppilaat ymmärtävät vaiheiden merkitykset ja tärkeydet. Muotoiltujen esimerkkien avulla voidaan opettaa monimutkaisia ongelmanratkaisu taitoja ja ne auttavat skeemojen muodostamisessa ja oppimisen kehittämisessä. [3]

Helsingin yliopiston tietorakenteet kurssin yhteydessä tehdyt kyselyt näyttäisivät, että muotoillut esimerkit olisivat suositumpia oppimisen lähteenä, kuin perinteiset luennot. [6].

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

Muotoillut esimerkit soveltuvat opiskelukohteisiin, joissa taidonhankkiminen on oleellista kuten musiikissa, shakissa ja ohjelmoinnissa *hae viite Instructional Design of a Programming Course... viite n.2*. Hunosti menestyvät oppilaat tarkastelevat monesti muotoillut esimerkit ainoastaan jos ne ovat tavanomaisista ongelmista, 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 [3]*.

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 muotoillut esimerkit vaikuttavat.

Muotoiltuja 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 kisällioppimisen kannalta muotoilluissa esimerkeissä suurimpaan asemaan tulevat mallintaminen (modelling) ja oppimisen oikea-aikainen tukeminen (scaffolding). Opettaja mallintaa muotoilluissa esimerkeissä työn ammattilaisen näkökulmasta ja pyrkii aukaisemaan sen oppilaille, niin että he ymmärtäisivät sen. Oppilas pääsee näkemään nämä vaiheet ja kuinka pienistä askelista edetään onnistuneeseen lopputulokseen ja hän pystyy rakentamaan mielikuvan kokonaisuuteen johtavista askelista. Kun oppilas tekee samoja taitoja vaativaa työtä, pystyy hän peilaamaan(reflection) suoritustaan opettajan suoritukseen ja ha hakea näistä muotoilluista esimerkeistä tukea. Muotoilluissa esimerkeissä pyritään myös luomaan esimerkit niin, että kaikki vaiheet olisi selitetty tarpeeksi hyvin auki suhteessa oppilaiden taitotasoon. Vaiheiden aikana oppilaita pyritään oikea-aikaisesti tukemaan (scaffolding) sen verran, että oppimiskokemuksesta tulee mielekästä.

4.2 Kysymysten muotoilu

Tämä aliluku pohjautuu artikkeliin The Abstraction Transition Taxonomy-Developing Desired Learning Outcomes through the Lens of Situated Cognition[4]. Kerromme kuinka kyseisessä artikkelissa käydään läpi miten kysymysten muotoilua pystyy käyttämään ohjelmoinnin opetuksessa.

Artikkelissa he tarkastelivat jonkin Yhdysvalloissa sijaitsevan yliopiston 7 eri CS0-, CS1- ja CS2-ohjelmointikurssien eli ohjelmoinnin peruskurssien opetusmateriaalejen sisältämien tehtävien rakennetta. Jokainen näistä kursseista arvosteltiin kurssikokeen perusteella, joten kurssien oppimistavoitteet pohjautuivat niihin.

Opetusmateriaalien tehtävistä noin viidesosa oli miksi-kysymyksiä ja loput kuinka-kysymyksiä. Kuinka-kysymyksissä opiskelija voi joutua kirjoittamaan toimivan koodin ohjelmointitehtävään, mutta miksi-kysymyksissä joutuu hän 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 pystyy hän 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 tukevat ideaa niiden toimivuudesta. Miksi-kysymyksiä tukevia tekijöitä löytyy kognitiivisen kisällioppimisen ja oppimisen tilannesidonnaisuudesta.

Kysymysten muotoilulla on siis merkitystä siihen, kuinka oppilas joutuu muuttamaan ajatteluaan lähestyessään ongelmaa. Oppilas joutuu artikuloimaan(articulation) ratkaisuun johtavia vaiheita, jolloin hän käy läpi ratkaisuaan syvemmin omin sanoin. Oppilas käy läpi oman ongelmanratkaisuprosessinsa ja pystyy samalla peilaamaan(reflection) sitä opettajan työskentelyyn. Samalla oppilas pystyy huomaamaan, mitkä vaiheet ovat johtaneet mahdolliseen onnistumiseen tai epäonnistumiseen, joten hän tietää missä on vielä parannettavaa. Miksi-kysymykset ovat tärkeitä myös opettajan kannalta, koska tällöin opettaja pystyy näkemään oppilaan ajatusmaailman ja antamaan hänelle oikeia-aikaista tukea (scaffolding) ja ohjaamaan oppilasta oikeaan suuntaan, jos vastaukset ovat hapuilevia.

4.3 Pariohjelmointi

Pariohjelmoinnissa kaksi henkilöä työskentelevät yhdessä samalla työpisteellä. Henkilöistä toinen on ajaja(driver), joka kirjoittaa koodia, kun toinen henkilö tarkkailija(navigator) 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 [8] ovat määritelleet pariohjelmoinnin Suurimpia yhteistyön seurauksena syntyviä hyötyjä 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 myös 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 myös kysymään toisiltaan toimiiko jokin osa koodista niinkuin 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 debuggaaminen 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 avoiminmielin toiselle.

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

Williams & Kessler [8] ovat määritelleet myös 7 erilaista tapaa joilla edistää pariohjelmointina työskentelyä. Pitää muistaa pitää taukoja. 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, minuuttia ei saisi kulua ilman suullista kommunikointia, 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 myös 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 [5] artikkelissa kerrottiin, että oppilaat eivät olleet niin tyytyväisiä tyytyväisiä etänä toteutettuun pariohjelmointiin, kuin perinteisiin pariohjelmointiin.

Kognitiivisen kisällioppimisen kannalta pariohjelmoinnissa, jossa molemmat henkilöistä ovat taisotasoltaan suunnilleen samoista lähtökohdista tulee tärkeään asemaan artikulointi ja peilaaminen. Pareilla on tärkeää ilmaista 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. Tarkastellessaan toisen ajatuksia, pystyy niitä vertailemaan omiin tietoihin ja taitoihin. Vertailua tekemällä oppii mahdollisesti uusia tekniikoita ja saa uusia näkökulmia lähestyä tilanteita. Parit pystyvät korjaamaan toistensa huonoja tekniikoita ja tapoja ja samalla korvaamaan ne paremmilla. Pariohjelmoinnissa tapahtuu siis tiedon vaihtoa ja siirtoa, parit opettavat

toisiaan.

4.4 Pajaohjelmointi

4.5 Perinteiset luennot

4.6 Koodausdojot

5 Yhteenveto

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] 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.

[3] 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`.

[4] 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`.

[5] 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`.

[6] 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.

[7] 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.

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