Kognitiivinen oppipoikamalli tietojenkäsittelytieteessä

Pessi Moilanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 20. maaliskuuta 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	20. maaliskuuta 2014	13

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 Kognitiivinen kisälliopetus teoreettisen tietojenkäsittelytieteen opetuksessa 4
4.2 Työstetyt esimerkit 5
4.3 Kysymysten muotoilu 7
4.4 Pariohjelmointi 8
4.5 Pajaohjelmointi 10
4.6 Perinteiset luennot 11
4.7 Koodausdojot 11

5 Yhteenveto **11**

Lähteet **11**

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

Opettamiseen on olemassa monia erilaisia tekniikoita ja menetelmiä. Tässä aineessa käymme läpi perinteisen 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 käytännössä.

Mestari on henkilö, joka on alansa ammattilainen eli erittäin kokenut tietyssä asiassa. Mestarilla voi olla oppipoikia, jotka hänen opastuksellaan pyrkivät saamaan kokemusta tästä mestarin hallitsemasta asiasta. Oppipoika eli kisälli on henkilö, joka pyrkii oppimaan mestarin opastuksella. Mestari ohjeistaa kisälliä työn suorituksessa soveltaen erilaisia opetustekniikoita. Tarpeeksi harjoiteltuaan ja opittuaan kisälli ei pysty kartuttamaan ammattitaitoaan saman mestarin alaisena.

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

Kisälli saa suoritettavakseen mestarin avustuksella pienempiä tehtäviä, jotka ovat sopivia kisällin sen hetkiselle taitotasolle. Syynä tähän on monissa tilanteissa tehtävien tärkeys ja vaikeus, mikä ei jätä tilaa virheille. Täysin uusi kisälli tuskin saa tarpeeksi luottoa mestarilta suorittaakseen tehtäviä, joista voi epäonnistumisen yhteydessä koittua suuria vahinkoja. Kisällille on myös mielekkäämpää tehdä pienempiä tehtäviä, koska tällöin hän näkee oman työnsä jäljen ja pystyy nauttimaan onnistumisesta useammin. Pienten tehtävien myötä kisällille 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.

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

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.

3 Ohjelmoinnin opetus

Ohjelmointi on taito, jota pystyy oppimaan tehokkaasti henkilöltä, joka on ammattilainen ohjelmoinnissa [10]. Opetuksessa käytettävien keinojen valitseminen on oleellista, koska kaikki tavat eivät tue tehokasta oppimista.

Kun ohjelmointia aletaan opettamaan täysin kokemattomille, ei tavoit-

teena 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 [10].

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. [4][9]

Teoreettisessa tietojenkäsittelytieteessä suuret epäonnistumis määrät ovat yleinen ongelma ainakin Euroopan ja Pohjois-Amerikan yliopistoissa [6]. Oppilailla on vaikeuksia sisältää teoreettisten kurssien asiaa niitten abstraktin ja teoreettisen luonteen vuoksi [6].

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 Kognitiivinen kisälliopetus teoreettisen tietojenkäsittelytieteen opetuksessa

Opetuksen muuttaminen perinteisestä opetuksesta kognitiivisen kisälliopetuksen suuntaan on tuottanut positiivisia tuloksia muun muassa Potsdamin yliopistolla, Saksassa, missä erään teoreettisen kurssin läpäisyprosentti on kasvanut kognitiivisten kisälliopetusmenetelmien 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. [6]

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. [6]

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. [6]

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. [6]

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 osoittivat, 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 kisälliopetuksen keinoja, joten tuloksissa on parantamisen varaa. [6]

4.2 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. [3]

Helsingin yliopiston tietorakenteet kurssin yhteydessä tehdyt kyselyt näyttävät, että työstetyt esimerkit ovat suositumpia oppimisen lähteenä, kuin perinteiset luennot. [8].

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 [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 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 kisällioppimisen 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.

4.3 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 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 kisällioppimisen ja oppimisen tilannesidonnaisuudesta.

Kysymysten muotoilulla on siis merkitystä siihen, kuinka oppilas joutuu muuttamaan ajatteluaan lähestyessään ongelmaa. Oppilas joutuu artikuloi-

maan 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.

4.4 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 [11] 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 [11] 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"[5] 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.

4.5 Pajaohjelmointi

Jotenki järkevästi näitte kahe viitteen käyttö [10][7]. 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 kisälliopetuksen 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 kisällioppimisen 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.

4.6 Perinteiset luennot

4.7 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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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