Tiina Mononen

OLIO-OHJELMOINNIN PERUSKÄSITTEIDEN OPETTAMINEN

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2012

TIIVISTELMÄ

Mononen, Tiina
Olio-ohjelmoinnin peruskäsitteiden opettaminen
Jyväskylä: Jyväskylän yliopisto, 2012, 34 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Hirvonen, Pertti

Olio-ohjelmointia pidetään vaikeana aiheena opettaa. Syytä on etsitty ohjelmointikielistä, opetukseen käytetyistä työkaluista, opetustavoista sekä järjestyksestä, missä oliot opetetaan suhteessa proseduraalisiin ohjelmointikäytäntöihin.

Tässä tutkielmassa perehdytään kirjallisuuskatsauksen avulla siihen, miten olio-ohjelmoinnin peruskäsitteet voitaisiin opettaa parhaiten. Jokainen edellä mainituista aihealueista käydään läpi, ja niille etsitään parhaat ratkaisut aiemman tutkimuskirjallisuuden avulla.

Tutkielman tuloksena on, että olio-ohjelmoinnin opettaminen on monisyinen ongelma, johon ei löydy mitään helppoa ratkaisua. Edes parasta ratkaisua millään osa-alueella ei voida löytää, vaan kyse on aina siitä, mikä tietylle kurssille sopii.

Asiasanat: olio-ohjelmointi, opettaminen

ABSTRACT

Mononen, Tiina

Teaching basic concepts of object-oriented programming
Jyväskylä: University of Jyväskylä, 2012, 34 p.
Information Systems, Bachelor's Thesis
Supervisor: Hirvonen, Pertti

Teaching object-oriented programming is considered difficult. The reason for this has been searched from programming languages, teaching tools, teaching methods and the teaching order of the procedural and object-oriented programming.

This thesis researches how to best teach object-oriented programming. Used method is literature review. Every aspect mentioned as a reason for learning difficulties in object-oriented programming is inspected and the best solution for each of them is seeked with the help of research literature.

Result of this thesis is that teaching object-oriented programming is a complex question, where easy answer can not be found. Even the best solution for any of the aspects why the teaching is so difficult, is difficult to found. That is because for different courses suits best different methods.

Keywords: object-oriented programming, teaching

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Olio-ohjelmointia käytetään nykyään paljon. Tämän takia sen opettaminen tuleville IT-alan ammattilaisille on tärkeää. Opiskelijat törmäävätkin usein jo ensimmäisillä ohjelmointikursseillaan olio-ohjelmoinnin peruskäsitteisiin (Bennedson & Caspersen, 2007). Näiden asioiden opettaminen opiskelijoille ei ole kuitenkaan helppoa, sillä useiden tutkimusten mukaan opiskelijoilla on vaikeuksia oppia olio-ohjelmointia (Boustedt, Eckerdal, McCartney ym., 2007; Bennedson & Caspersen, 2004; Yuen & Liu, 2010). Tästä herääkin kysymys: miten olio-ohjelmointia olisi parasta opettaa ja missä vaiheessa aloittelevan ohjelmoijan opintoja sen opetus pitäisi aloittaa?

Tutkimusta tästä aiheesta on tehty niin kauan kuin olio-ohjelmointia on opetettu eli nyt jo monta kymmentä vuotta. Vaikka tutkimusta onkin ollut paljon, sitä ei ole vielä saatu kunnolla siirrettyä käytäntöön (Pears, Seidman, Malmi ym., 2007). Esimerkiksi Taina Tikansalon (2008) pro gradu -tutkielmassa tutkittiin mm. Jyväskylän yliopiston opiskelijoiden olioiden oppimista ja todettiin, ettei se ole riittävällä tasolla. Useissa muissakin tutkimuksissa on todettu, että monet opiskelijat eivät ole sisäistäneet olio-ohjelmoinnin peruskäsitteitä (esim. Turner, Quintana-Castillo, Pérez-Quiñones & Edwards, 2008; Stamouli & Huggard, 2006). Ongelma olio-ohjelmoinnin oppimisessa on siis vahvasti yhä olemassa.

Tässä tutkielmassa selvitetään aikaisemman kirjallisuuden perusteella, miten olio-ohjelmoinnin peruskäsitteet voisi parhaiten opettaa. Olio-ohjelmoinnin opettamiseen on etsitty apukeinoja ohjelmointiympäristöistä, ohjelmointikielistä, opetustavoista sekä järjestyksestä, jossa olio-ohjelmointi opetetaan suhteessa perinteiseen proseduraaliseen ohjelmointiin. Tutkimuksissa tuodaan monesti esiin uusi oma opetustapa tai -väline, jota pidetään parhaana, mutta näitä eri opetusta parantavia asioita kokoavaa ja vertailevaa tutkimusta on vähän. Tarkoituksena tässä tutkielmassa onkin koota yhteen näitä erilaisia olio-ohjelmoinnin peruskäsitteiden opettamisessa auttavia tekijöitä. Lähteinä käytetään kaikkia olio-ohjelmoinnin opettamiseen keskittyviä tieteellisiä julkaisuja, vaikka pääpaino on tietenkin uudemmassa, 2000-luvulla julkaistussa materiaalissa.

Olio-ohjelmointia opetetaan monesti jo ohjelmoinnin alkeita opettavilla kursseilla, joten tutkielma keskittyy pääosin siihen, miten opettaa peruskäsitteet opiskelijoille, joilla ei ole juurikaan aikaisempaa kokemusta ohjelmoinnista. Erityisesti paino on yliopiston kandidaatin tutkinnon tai vastaavien opintojen tasoisessa opetuksessa.

Helppoa ratkaisua ei ole löydetty olio-ohjelmoinnin käsitteiden opettamisen ongelmiin. Millään yksittäisellä tekijällä ongelmia ei voida korjata vaan kysymys on monen asian kokonaisuudesta. Asiaan voivat vaikuttaa niin ohjelmointiympäristö, käytetty kieli, asioiden opettamisjärjestys kuin myös opetustavat. Toisaalta mistään näistä alueista ei voida löytää ehdottomasti parasta ratkaisua vaan kysymys on aina kokonaisuudesta. Erilaisille kursseille, opettajille ja opiskelijoille sopivat parhaiten hiukan eri lähestymistavat. Tämä tutkielma kerää yhteen tietoa olio-ohjelmoinnin opettamisesta ja antaa näin pohjaa jatkotutkimukselle aiheesta. Toisaalta se antaa olio-ohjelmointia opettaville kuvan erilaisista asioista, jotka vaikuttavat olio-ohjelmoinnin opettamiseen tai voivat edesauttaa sitä.

Seuraavassa kappaleessa käydään läpi, mitä olio-ohjelmoinnilla ja sen peruskäsitteillä tarkoitetaan. Tämän jälkeen käsitellään, miksi olio-ohjelmointia on niin vaikea opettaa. Sen jälkeen paneudutaan ongelmien ratkaisijoiksi esitettyihin asioihin, joita ovat tulisiko oliot esitellä opiskelijoilla heti ensimmäisen ohjelmointikurssin alussa vai ei, ohjelmointikielet, olio-ohjelmoinnin opetuksessa käytettävät työkalut sekä millä opetustavoilla voisi parantaa opiskelijoiden olio-ohjelmoinnin peruskäsitteiden oppimista.

2 KÄSITTEIDEN MÄÄRITTELY

2.1 Olio-ohjelmointi

Olio-ohjelmointi on yksi ohjelmointiparadigma eli tapa kuvata erilaisten järjestelmien rakennetta ja toimintaa (Koskimies, 2000, s. 21). Se muokkaa ajattelutapaa, jolla ongelmasta muodostetaan algoritminen malli. Tämä malli voidaan toteuttaa ohjelmoimalla, ja se määrittää ohjelman rakenteen. (Kölling, 1999a.)

Olio-ohjelmointi liittää ohjelmoinnin mallintamiseen ja simulointiin. Oliot nimittäin tunnistetaan ohjelman sovellusalueen pohjalta ja niitä käytetään mallintamaan tuon kohdealueen toimintoja. (Craig, 2007, s. 1.) Ohjelmistot kuvataan näiden tunnistettujen keskenään kommunikoivien olioiden avulla (Koskimies, 2000, s. 21). Olioparadigmassa pidetäänkin keskeisenä sen yhteyttä oikeaan maailmaan ja ihmisten luontaiseen tapaan hahmottaa todellisuus (Koskimies, 2000, s. 21; Zhu & Zhou, 2003).

Olio-ohjelmoinnin katsotaan alkaneen jo 60-luvun lopussa Simula-kielen luomisesta, vaikkakin se alkoi levitä laajemmin käyttöön vasta 1980-luvun puolivälissä (Koskimies, 2000, s. 26). Nykyään olio-ohjelmointia käytetään paljon. Erään useita maita kattavan tutkimuksen perusteella lähes puolet ohjelmoinnin perusteita opettavista kursseista opetetaankin jo olio-ohjelmointia käyttäen (Bennedson & Caspersen, 2007). Onkin tullut selkeäksi, että olio-ohjelmointia pitää opettaa opiskelijoille (Kölling, 1999a). ACM/IEEE-CS Joint Task Force on tehnyt suositukset, mitä tietojenkäsittelytieteiden opiskelijan pitäisi vähintäänkin oppia. Olio-ohjelmointi mainitaan näissä suosituksissa aiheena, josta kaikilla alan opiskelijoilla pitäisi olla tietoa. (Computing Curricula 2001, 2001, s. 115–116; Computer Science Curricula 2013, 2012, s. 129.)

2.2 Olio-ohjelmoinnin peruskäsitteet

Olio-ohjelmoinnin peruskäsitteiden pitäisi siis tulla tutuiksi kaikille tietojenkäsittelytiedettä opiskeleville. Seuraavassa määritellään nämä kyseiset peruskäsitteet. Koskimies (2000) laskee olio-ohjelmointia opettavan kirjansa kappalejaossa olion, luokan ja suojauksen peruskäsitteiksi. Muita isoja aihealueita teoksessa ovat periytyminen, myöhäinen sidonta ja polymorfismi. Ne ovatkin olio-ohjelmoinnin erikoisuuksia verrattuna muihin ohjelmointiparadigmoihin (Koskimies, 2000, s. 21). Nämä samat käsitteet toistuvat useissa olio-ohjelmoinnin opettamista käsittelevissä tutkimuksissa, vaikka niissä ei suoraan määritelläkään, mitkä ovat olio-ohjelmoinnin peruskäsitteet, jotka opiskelijoiden tulisi oppia (esim. Berges & Hubwieser, 2011; Hu, 2004; Sanders, Boustedt, Eckerdal ym., 2008; Golwasser & Letscher, 2008).

ACM/IEEE-CS Joint Task Force määrittelee näistä kapseloinnin, luokat ja niiden ilmentymät, perinnän ja polymorfismin asioiksi, jotka kaikkien tietojenkäsittelytieteiden opiskelijoiden pitäisi osata (Computing Curricula 2001, 2001, s. 115–116; Computer Science Curriculum 2008, 2008, s. 66–67). Seuraavassa käydään läpi tarkemmin nämä olio-ohjelmoinnin käsitteet, joita tässä tutkielmassa tarkoitetaan puhuttaessa olio-ohjelmoinnin peruskäsitteistä.

Olio on tietenkin olio-ohjelmoinnin hyvin perustava käsite. Olio on ohjelman rakenteen perusyksikkö. Siihen voidaan tallentaa tietoa attribuutteihin ja se pystyy suorittamaan toimintoja eli operaatioita. (Koskimies, 2000, s. 30). Yleensä toiminnot käsittelevät tietoa, jota olio sisältää. Olio on itsenäinen kokonaisuus, jolla on muista olioista ja ympäristöstä erottuva identiteetti. (Craig, 2007, s. 3.)

Suurin osa olio-ohjelmointikielistä perustuu luokan ja ilmentymän käsitteille (Craig, 2007, s. 13). Jokaisella oliolla on yksikäsitteinen luokka, jonka ohjeen mukaan se on luotu. Olio on tämän luokan ilmentymä. Luokka määrittelee olion piirteet. Se siis kertoo, miten sen ilmentymät toimivat ja millaista tietoa ne sisältävät. (Koskimies, 2000, s. 37.)

Olio-ohjelmoinnin yksi tärkeimmistä hyödyistä on tiedon kapselointi (Wick, Stevenson & Phillips, 2004). Kapselointi tarkoittaa, että tieto ja siihen liittyvät toiminnot pakataan yhdeksi suojatuksi kokonaisuudeksi. Olio on tällainen kokonaisuus, jota voi käyttää vain tietyillä tavoilla. Ulkopuoliset eivät näe olion sisäistä toteutusta tai sen tietoja vaan vain ulkoisen rajapinnan, jonka kautta oliota voidaan käyttää. (Koskimies, 2000, s. 30; Craig, 2007, s. 3-4.)

Olioiden yksi keskeinen ominaisuus on perintä (Craig, 2007, s. 3). Se tarkoittaa yhden luokan ominaisuuksien siirtymistä toiselle luokalle (Koskimies, 2000, s. 68). Luokka, joka perii, voi käyttää toiselta luokalta perimiään piirteitä kuin omiaan (Koskimies, 2000, s. 69).

Olio-ohjelmoinnin tyypillisenä ominaisuutena ja hyvänä puolena pidetään usein polymorfismia (Koskimies, 2000, s. 96). Se tarkoittaa, että ohjelmassa oleva kohde voi ottaa vastaan tai palauttaa useamman tyyppisiä asioita (Koskimies, 2000, s. 96; Craig, 2007, s. 4). Esimerkiksi muuttujalla on tietty luokka, mutta se

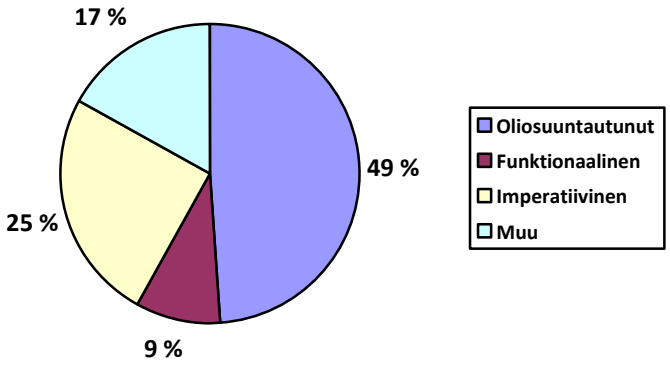
voi ajon aikana kuitenkin viitata myös olioon, jonka luokka perii luokan, joka muuttujalle on annettu (Koskimies, 2000, s. 96).

Ohjelmointi kielessä voidaan käyttää aikaista sidontaa eli sitoa staattisesti aliohjelma kutsut kutsuttavan aliohjelman määrittelyyn. Oliokielten keskeinen ominaisuus kuitenkin on, että aikaista sidontaa ei käytetä aina. Tämä johtuu polymorfismista. Oliota voidaan käyttää sen perimän luokan edustajan paikalla, ja toisaalta perivässä luokassa voidaan muuttaa perityn luokan operaatioita. Näin ei voida olla etukäteen varmoja mihin aliohjelmaan aliohjelma kutsu tulee viittaamaan. Tällöin käytetään myöhäistä sidontaa. (Koskimies, 2000, 49.)

3 OLIO-OHJELMOINNIN OPETTAMINEN

3.1 Olio-ohjelmoinnin opettamisen vaikeus

Aikaisemmin olio-ohjelmointia opetettiin vasta opintojen myöhemmässä vaiheessa, mutta jo 2000-luvun vaihteessa se oli siirtymässä ensimmäisille ohjelmointikursseille, jotka opiskelijoille tulivat eteen (Kölling, 1999a). 2000-luvulla kasvava määrä instituutioita onkin esittellyt ohjelmoinnin opiskelijoille olio-ohjelmointi kielen kautta (Stamouli & Huggard 2006). Vuonna 2007 tehdyn tutkimuksen mukaan jo lähes puolet ohjelmoinnin perusteita opettavista kursseista opetetaan olio-ohjelmointia käyttäen, kuten kuviosta 1 näkyy (Bennedson & Caspersen, 2007).



KUVIO 1 Ensimmäisellä ohjelmointikurssilla käytetyt ohjelmointiparadigmat

Vaikka olio-ohjelmointia opetetaan ensimmäisiltä ohjelmointikursseilta alkaen, pidetään sitä yhä vaikeana aiheena sekä oppia että opettaa (Boustedt, Eckerdal, McCartney ym., 2007; Bennedson & Caspersen, 2004; Yuen & Liu, 2010). Monissa tutkimuksissa onkin huomattu, että opiskelijat eivät ole kursseilla täysin sisäistäneet olio-ohjelmoinnin peruskäsitteitä (esim. Turner, Quintana-Castillo, Pérez-Quiñones & Edwards, 2008; Stamouli & Huggard, 2006).

Yksi syy, miksi oppilailla on vaikeuksia oppia olio-ohjelmointia voi olla opetuksessa yhä säilynyt proseduraalinen ajatusmalli (Bennedson & Caspersen,

2004). Köllingin (1999a) mukaan, kun on aluksi tottunut proseduraaliseen ohjelmointiin, on vaikea siirtyä olio-ohjelmoinnin ajatusmalliin. Toisinpäin vaihdos on sen sijaan helpompi.

Köllingin (1999a) mielestä onkin virhe pitää olio-ohjelmointia vain kielellisenä rakenteena, joka voidaan opettaa muiden rakenteiden jälkeen. Olio-ohjelmointi on hänen mukaansa paradigma, joka vaikuttaa tapaan, jolla ongelma ratkaistaan. Se määrää, millainen ohjelman rakenne on ja korvaa proseduraalisen ohjelmoinnin rakenteet sen sijaan, että ainoastaan lisäisi niiden päälle uutta.

Kölling (1999a) esittää myös, että ongelmia ei tuottaisikaan olio-ohjelmointi itsessään vaan siihen käytettävät välineet. Kieli ja ohjelmointiympäristö ovat tärkeitä ohjelmoinnin opetuksessa. Käytetyt ohjelmointikielet ovat kuitenkin hänen mielestään liian monimutkaisia ja ohjelmointiympäristöt liian hämmentäviä. Sopivan olio-ohjelmointi kielen valitseminen onkin tärkeää, jotta opiskelijat saavat parhaat mahdolliset taidot olio-ohjelmoinnista (Ekuobase, Akwukwuma & Egbokhare, 2009).

Epätehokkaat opetustavat voivat olla myös yksi pääsyistä siihen, että oppilaat oppivat huonosti olio-ohjelmointia (Sivasakthi & Rajendran, 2011). Opetus saattaa esimerkiksi olla syntaksilähtöistä eli keskitytään opetettavaan kieleen eikä yleisempiin ohjelmointitekniikkoihin. Tämä ei johda parhaaseen oppimistulokseen, koska pelkkä ohjelmointikielen osaaminen ei riitä tekemään hyvää ohjelmoijaa (Bennedson & Caspersen, 2004). Kokonaiskuvan opettaminen on kuitenkin haasteellista, sillä opiskelijat jumittuvat helposti kielen yksityiskohtiin ja kadottavan käsityksen kokonaisuudesta (Sanders, Boustedt, Eckerdal ym., 2008).

On myös esitetty, että olio-ohjelmointi on yksinkertaisesti liian monimutkainen ja abstrakti asia ensimmäiselle ohjelmointikurssille. Opiskelijat, joilla ei ole juurikaan aiempaa ohjelmointikokemusta, eivät ole valmiita opiskelemaan näin vaikeita asioita, eivätkä siksi kykene niitä oppimaan hyvin. (Hu, 2004.)

Toisaalta Berges ja Hubwieser (2011) totesivat tutkimuksessaan, että opiskelijoiden on mahdollista lähteä nollatasolta ja kirjoittaa parissa päivässä olioita käyttävä toimiva ohjelma parin ohjepaperin avulla. Kovin syvällistä osaamista tällä ei tietenkään voitu saavuttaa, mutta opiskelijat saivat kuitenkin jonkinlaisen kuvan olio-ohjelmoinnista nopeasti. Vaikka olio-ohjelmoinnin peruskäsitteiden opettaminen onkin vaikeaa, niiden ei siis pitäisi olla ylitsepääsemättömiä.

3.2 Missä vaiheessa olio-ohjelmointi tulisi opettaa?

Yksi keskeinen kysymys olio-ohjelmoinnin peruskäsitteiden opettamisessa on ollut, pitäisikö ne opettaa ennen proseduraalisia ohjelmointikäytänteitä vai vasta niiden jälkeen. Kuten aikaisemmin tässä tutkielmassa on todettu, on olio-ohjelmoinnin opettamisesta jo ensimmäisellä ohjelmointikurssilla tullut yleisempää. Tästä huolimatta aiheesta kiistellään yhä. Pääsyy kiistelyn jatkumiselle on väittelyn monimutkaisuus. Mukana keskustelussa on monia opetukseen liit-

tyviä ulottuvuuksia, kuten oppimistavoitteet, käytetyt esimerkit, kursseilla olevat tehtävät ja kokeet, työvälineet ja arvostelutekniikat. (Ehlert & Schulte, 2009.)

Oliot ensin -lähestymistavassa korostetaan olio-ohjelmoinnin ja oliosuunnittelun periaatteita alusta asti. Ensimmäinen ohjelmointikurssi aloitetaan olion ja perinnän käsitteillä, jolloin ne tulevat tutuiksi opiskelijoille kurssin alusta lähtien. Oppilaiden kokeiltua niitä yksinkertaisilla interaktiivisilla ohjelmilla, kurssilla esitellään perinteiset kontrollirakenteet. Mukana opetuksessa pyritään kuitenkin koko ajan pitämään myös olio-ohjelmoinnin periaatteet. (Computer Curricula 2001, 2001, s. 30.)

Suurin etu oliot ensin -strategiassa on oppilaiden varhainen tutustuminen olio-ohjelmointiin, josta on tullut yhä tärkeämpi sekä akateemisessa maailmassa että yrityksissä. Jos oppilaat aloittavat jo varhaisessa vaiheessa olio-ohjelmoinnin opiskelun, työskentelevät he sen parissa enemmän, mikä voi auttaa hallitsemaan tämän vaikeana pidetyn asian. (Computer Curricula 2001, 2001, s. 30.) Tämä strategia voi auttaa olio-ohjelmoinnin oppimisessa myös sen takia, että joidenkin mielestä opiskelijoilla, jotka ovat oppineet proseduraalisen ohjelmoinnin ennen olio-ohjelmointia, on suurempia vaikeuksia oppia olionäkökulmaa. (Kölling 1999a; Computer Curricula 2001, 2001, s. 30.)

Jos olio-ohjelmointia ei opeteta ohjelmoinnin perusteita esittelevällä kurssilla aluksi, keskittyy kurssi kontrollirakenteisiin, funktioihin ja muihin perinteisen proseduraalisen ohjelmoinnin keskeisiin elementteihin (Computer Curricula 2001, 2001, s. 30). Näillä kursseilla opetetaan tyypillisesti aluksi muuttujat ja sijoittaminen. Tämän jälkeen tulevat valinta, iteraatio ja aliohjelmat. (Ehlert & Schulte, 2009.) Tällöin olio-ohjelmoinnin tekniikat jätetään yleensä vasta toiselle ohjelmointikurssille (Computer Curricula 2001, 2001, s. 30).

Olio-ohjelmoinnin opettaminen vasta myöhemmin on hyvä asia, koska olio-ohjelmointi voi olla liian monimutkainen, laaja ja abstrakti aihe ensimmäiselle ohjelmointikurssille (Hu, 2004). Oliokielen monimutkaisuus voi musertaa aloittelevan opiskelija. Lisäksi opiskelijat, jotka ovat tottuneet työskentelemään oliokielillä, eivät välttämättä ole halukkaita oppimaan työskentelemään ilman piirteitä, jotka tekevät olio-ohjelmoinnista niin tehokkaan. (Computer Curricula 2001, 2001, s. 30.) Etuna proseduraalisten tekniikoiden opettamisessa ensinnä on myös se, että opiskelijat voivat oppia ne paremmin. Jos aikaa käytetään olio-ohjelmointiin, opiskelijat eivät välttämättä ehdi tai halua syventyä niin hyvin proseduraaliseen ohjelmointiin. (Computer Curricula 2001, 2001, s. 30; Hu, 2004.)

Bennedson ja Caspersen (2007) tekivät useita maita kattaneen kyselytutkimuksen siitä, kuinka suuri osa opiskelijoista pääsee kandidaatin opintojen tasoisen ohjelmoinnin perusteita opettavan kurssin läpi. Noin puolella kursseista opetettiin oliot ensin, kuten kuviosta 1 näkyy. Kaikissa kursseissa oli kuitenkin samanlaiset läpäisyprosentit, joten tutkimus ei tue sitä, että kurssin helppouteen vaikuttaisi, missä vaiheessa olio-ohjelmointi opetetaan kurssilla.

Ehlert ja Schulte (2009) tekivät puolestaan empiirisen tutkimuksen siitä, vaikuttaako opiskelijoiden oppimiseen opetetaanko oliot aluksi vai vasta myöhemmin ohjelmoinnin perusteita opettavalla kurssilla. He eivät löytäneet mi-

tään eroa oppimistuloksessa pidemmällä aikavälillä. Oppimisen kannalta ei siis ole oleellista opetetaanko oliot aikaisemmin vai myöhemmin.

Tuloksen syyksi arveltiin sitä, että ohjelmoinnissa vaikeita aiheita on sekä olio-ohjelmoinnin piiriin kuuluvissa että siihen kuulumattomissa aihealueissa. Opiskelijat kokivat esimerkiksi taulukot vaikeammiksi kuin perinnän. Näin ollen olio-ohjelmointi ei itse asiassa ole vaikeampaa kuin muu ohjelmointi. Siksipä olio-ohjelmoinnin opettamisen järjestys ei vaikuta lopulliseen oppimistulokseen. (Ehlert & Schulte, 2009.)

Myöskään ACM/IEEE-CS Joint Task Forcen tekemät ohjeistukset siitä, mitä tietojenkäsittelytieteen opiskelijan pitää oppia, eivät suosittele mitään tiettyä opetusjärjestystä ohjelmoinnin perusteiden esittelyyn. Vuoden 2001 ohjeistuksessa esitellään tasa-arvoisina kuusi eri näkökulmaa ohjelmoinnin opettamiseen (Computing Curricula 2001, 2001, s. 22). Vaikka asiat opetetaan eri järjestyksessä, opetetaan kaikissa olio-ohjelmointi opiskelijoille kolmen ensimmäisen lukukauden aikana (Computing Curricula 2001, 2001, s. 29–34). Näitä kaikkia on käytetty onnistuneesti useammassa tiedekunnassa (Computing Curricula 2001, 2001, s. 29). Samoin vuoden 2013 kommentoinnille avoinna olevassa raakaversiossa jatketaan samaa linjaa. Olio-ohjelmointi on kaikille pakollinen, mutta alun opinnot voidaan opettaa eri järjestyksessä. (Computer Science Curricula 2013, 2012, s. 23 ja s. 129.) Järjestystä ei voi määrätä, sillä eri koulutusohjelmiin sopii eri järjestys (Computer Science Curricula 2008, 2008, s. 26).

Loppujen lopuksi uusimman tutkimuksen valossa ja myös ottaen huomioon ACM/IEEE-CS Joint Task Forcen lukujärjestyssuositukset ei ole väliä, opetetaanko oliot ennen vai jälkeen suhteessa kielten perinteisiin kontrollirakenteisiin. Olio-ohjelmointi on tärkeä asia, joka kaikkien tulisi oppia, mutta ei ole oleellista tuleeko se ensimmäisellä ohjelmointikurssilla vai vasta sen jälkeen. Jokainen opettaja voi valita oman tyylinsä, sillä oppilaat oppivat pitkällä tähtäimellä asiat yhtä hyvin.

3.3 Kielen valinta olio-ohjelmoinnin opetukseen

Olio-ohjelmoinnin opettamiseen käytetyt ohjelmointikielet voivat olla liian monimutkaisia ohjelmoinnin alkeita opiskeleville (Kölling, 1999a). Jos opettaja ei erityisesti paneudu esittelemään materiaalia niin, että se rajoittaa kielen monimutkaisuuden esilletuloa, kielen yksityiskohdat voivat helposti musertaa aloittelevan opiskelijan (Computer Curricula 2001, 2001, s. 30.). Opetuksessa käytetään useita alalla suosittuja kieliä, mutta tämä saattaa olla huono käytäntö, sillä ne eivät ole välttämättä parhaita kieliä olio-ohjelmoinnin opettamiseen (Kölling, 1999a; Mannila & deRaad, 2006). Parhaan mahdollisen kielen käyttäminen opetuksessa onkin kriittistä (Ekuobase, Akwukwuma & Egbokhare, 2009).

Köllingin (1999a) mielestä olio-ohjelmoinnin opetukseen käytettävän kielen tärkeitä piirteitä ovat seuraavat:

- opetettavien käsitteiden tuominen esille selkeästi
- puhdas oliopohjaisuus

- turvallisuus: mahdollisimman aikainen virheiden havaitseminen ja niistä selkeästi kertominen
- korkean tason kieli
- ohjelman ajaminen yksinkertaista ja helppoa
- luettava syntaksi
- ei toistoa: kaiken tekemiseen on vain yksi tapa
- pieni kieli: oppilaille on tärkeää, että he voivat oppia kaiken
- helppo siirtyä muihin kieliin
- oikeellisuuden takaaminen: esimerkiksi esi- ja jälkiehdot
- sopiva ohjelmointiympäristö.

Monet näistä ominaisuuksista ovat samoja, jotka pitää ottaa huomioon ensimmäiselle ohjelmointikurssille kieltä valittaessa (Mannila & de Raad, 2006; Black, Bruce & Noble, 2010). Ne kaikki ovat kuitenkin oleellisia, jos halutaan opettaa olioita ensin. Olio-ohjelmoinnin opetus kielen valinta ei ehkä ole samalla tavalla oleellinen, jos olioita ei opeteta ensimmäisellä ohjelmointikurssilla, koska asiasta ei löydy yhtä paljon tutkimusta.

Kielen vaatimukset ovat pysyneet samoina aikojen saatossa, sillä Köllingin vuosituhannen vaihteessa esittämät ominaisuudet toistuvat yhä uusissa tutkimuksissa. Uutena mukaan on tullut opetusympäristöön liittyvät tekijät kuten opetusmateriaalin saatavuus ja opettajien kielen osaaminen (Ekuobase, Akwukwuma & Egbokhare, 2009; Mannila & deRaad, 2006). Nämä tekevät kielen päättämisestä ympäristöstä riippuvan, jolloin kaikille sopivinta kieltä ei voida löytää. Muita uusia vaatimuksia ovat esimerkiksi multimedian helppo tuominen ohjelmointiin, kielen käyttämisen kustannukset, mahdollisuus käyttää kieltä eri käyttöjärjestelmissä ja kielen käyttö muuallakin kuin opetuksessa. (Mannila & deRaad, 2006; Ekuobase, Akwukwuma & Egbokhare, 2009; Black, Bruce & Noble, 2010).

Monet ovat olleet sitä mieltä, että olemassa olevat kielet eivät ole olleet parhaita mahdollisia olio-ohjelmoinnin opettamiseen, joten he ovat kehittäneet opetukseen sopivia kieiä. Näitä ovat esimerkiksi Blue sekä the Initial Object-oriented Programming Language eli IOPL (Kölling & Rosenberg, 1996; Harrison, Sallabi & Eldridge, 2005). Opetukseen tehdyt kielet eivät kuitenkaan ole saaneet laajempaa suosiota. Tämä voi johtua siitä, että opettajat pitävät tärkeänä kriteerinä kielen valinnassa, että se on alalla käytetty kieli (Mannila & de Raadt, 2006; Mason, Cooper & de Raadt, 2012). Nykyään kielen valintakriteereissä esiintyykin usein sen käyttö opetuksen ulkopuolella (Mannila & deRaad, 2006; Ekuobase, Akwukwuma & Egbokhare, 2009)

Koska sopivan kielen valitseminen on hankalaa, on valinnan helpottamiseksi kehitetty erilaisia menetelmiä, joilla sopivimman kielen voi päättää. Näillä menetelmillä ei pyritä löytämään parasta mahdollista kieltä kaikissa tapauksessa vaan paras mahdollinen kieli, joka sopii tiettyyn tilanteeseen. Ekuobase, Akwukwuma ja Egbokhare (2009) ovat kehittäneet tällaisen menetelmän parhaan mahdollisen kielen löytämiseen olio-ohjelmoinnin opetukseen. He saivat tulokseksi, että C# sopii parhaiten opetukseen, mutta heti sen perässä tuli Java. Muut kielet (C++ ja Visual Basic) olivat paljon huonompia. Tuloksia ei voi kuitenkaan

yleistää, vaan jokaisen pitäisi itse löytää kaavaa käyttämällä paras kieli omiin olosuhteisiinsa. Tutkimus kuitenkin antaa yhden näkökulman parhaista kielistä.

Java on käytetyin kieli ohjelmoinnin perusteita opettavilla kursseilla ja samalla myös olio-ohjelmoinnin perusteita opettavilla kursseilla (Bennedson & Caspersen, 2007; Mason, Cooper & de Raadt, 2012; Davies, Polack-Wahl, & Anewalt, 2011). Javan suurena etuna pidetään sitä, että se on ammattilaisten paljon käyttämä kieli (Bruce, 2005; Mason, Cooper & de Raadt, 2012). Sitä ei ole kuitenkaan tarkoitettu aloitteleville ohjelmoijille, joten siinä on omat ongelmansa, jos sen avulla esitellään olio-ohjelmoinnin peruskäsitteitä aloittelevalle opiskelijalle. Nämä ongelmat eivät ole samalla tavalla oleellisia, jos olio-ohjelmoinnin peruskäsitteet opetetaan vasta myöhemmin opinnoissa, kun opiskelijat ovat jo tutustuneet Javaan.

Java ei ole aloittelevalle ohjelmoijalle helpoin mahdollinen kieli. Esimerkiksi heti aluksi opiskelija törmää main-metodiin, jossa on monta asiaa, joita opiskelija ei voi heti ymmärtää. Toinen ongelma Javassa ovat käyttöliittymät, sillä ohjelman pitäisi näyttää tulos ja ottaa vastaan syötteitä, mutta syötteiden vastaan ottaminen on Javassa monimutkaista. (Kölling & Rosenberg, 2001.) Nämä ongelmat on kuitenkin pyritty kiertämään kehittämällä erilaisia apuvälineitä opetukseen, joten nämäkään asiat eivät välttämättä ole ongelmia. Näihin apuvälineisiin paneudutaan enemmän seuraavassa luvussa.

Olio-ohjelmointikielten monimutkaisuutta on pyritty vähentämään myös sillä, että ei esitellä koko kieltä heti, vaan käytetään siitä vain osia. Esimerkiksi Hsia, Simpson, Smith ja Cartwright (2005) ovat suunnitelleet avuksi Javan hierarkkiset tasot, jotka auttavat keskittymään oliosuunnitteluun kielen mekaniikan sijaan. Malli muodostuu kolmesta tasosta ja lisäksi koko Java-kielestä. Jokaisella tasolla on edellistä rikkaampi ja monimutkaisempi kieli datan määrittelyyn ja laskennan suorittamiseen datalla. Ensimmäisellä tasolla käytetään vain paria avainsanaa, mutta niiden määrä lisääntyy joka tasolla. Koodia siis generoidaan opiskelijan näkymättömissä niin, että opiskelija voi keskittyä vain olennaisiin asioihin. Tällaiset tasot on käytännössä toteutettu esimerkiksi opetukseen suunnitellussa ProfessorJ-ohjelmointiympäristössä (Georgantaki & Retalis, 2007).

Myös Pythonin käyttämisen puolesta on puhuttu olio-ohjelmoinnin perusteiden opettamiseen. Python on myös kaupallisesti käytetty kieli, mutta syntaksiltaan yksinkertaisempi kuin esimerkiksi Java. Kieli tukee myös kaikkia tutkielman alussa määriteltyjä olio-ohjelmoinnin peruskäsitteitä. (Golwasser & Letscher, 2008). Tosin Pythonissa ei ole suoraan kapselointia, vaan kapseloinnin osoittamiseen käytetään nimeämiskonventioita. Python on helppo syntaksiltaan, mikä auttaa keskittymään olio-ohjelmoinnin peruskäsitteisiin. Lisäksi aloittelijan on helppo tehdä sillä nopeasti mielenkiintoisia projekteja. (Golwasser & Letscher, 2008; Oldham, 2005.) Eräässä tutkimuksessa kaikki Pythonin opetukseen valinneet opettajat pitivätkin tärkeimpänä kriteerinä valinnalleen, että kieli sopii hyvin opetukseen. Tämä siitä huolimatta, että kielen käyttö lisääntyy koko ajan myös ammattilaiskäytössä ja on näin ollen alalla käytetty kieli. (Mason, Cooper & de Raadt, 2012.) Huonoja puolia esimerkiksi Javaan verrattuna ovat

Pythonin dynaaminen tyypitys, joka saattaa tuottaa ongelmia, sekä kapseloinnin olemassa olo vain nimeämiskonventiona (Oldham, 2005).

Jo aikaisemmin hyvänä kielenä mainittu C# on hyvin lähellä Javaa syntaksiltaan, mutta se ei ole vielä ainakaan noussut paljon käytetyksi kieleksi ohjelmoinnin opettamisessa. Esimerkiksi Pythonia opetetaan ensimmäisellä ohjelmointikurssilla useammin. (Mason, Cooper & de Raadt, 2012; Davies, Polack-Wahl, & Anewalt, 2011.) Koska C#:n syntaksi on hyvin lähellä Javan syntaksia, on siinä ongelmana sama monimutkaisuus, jonka aloittelija kohtaa Javassa. Huonona puolena C#:ssa Pythoniin ja Javaan verrattuna onkin pääasiassa se, että sillä voi kehittää vain Windowsissa toimivia ohjelmia (Giangrande, 2007). Java on tällä hetkellä ohjelmoinnissa ylivoimaisesti suosituin oliokieli, mikä on saattanut pitää sen myös eniten opetuksessa käytettynä kielenä (TIOBE software, 2012).

Kölling (1999a) vertailee edellä mainittujen periaatteidensa perusteella C++:aa, Eiffeliä, Smalltalkia ja Javaa keskenään. Yksikään näistä kielistä ei ole hänen mielestään kovin hyvä opetuksessa, vaikkakin Eiffel on paras näistä. Myös Mannilan ja de Raadtin tutkimuksen mukaan, jossa yritettiin etsiä parasta kieltä ylipäänsä ohjelmoinnin opetukseen, Eiffel nousi kärkeen Pythonin kanssa. Heti näiden jälkeen tuli Java. Tutkimuksessa tosin ei painotettu niinkään olio-ohjelmoinnin opettamista vaan yleensä ottaen ohjelmoinnin alkeiden opettamista. (Mannila & de Raadt, 2006.) Nämä kielet tulevat kuitenkin esiin myös erityisesti olio-ohjelmoinnin opettamisesta puhuttaessa, joten tulos antaa jonkinlaista apua kielen valintaan.

Eiffel ei kuitenkaan ole yleensä ottaen suosittu kieli, joten se menee opetuksen kannalta ehkä samaan kategoriaan kuin opetukseen suunnitellut kielet (TIOBE software, 2012). C#, Python ja Java ovat siis aika samalla viivalla ja kielen valinta riippuu ehkä eniten siitä, mitä opettaja haluaa painottaa. Taulukkoon 1 on koottu edellä mainittujen tutkimusten avulla eri kielten hyviä ja huonoja puolia kielen valinnan selkeyttämiseksi.

TAULUKKO 1 Eri kielten sopivuus olio-ohjelmoinnin opettamiseen, erityisesti oliot ensin näkökulmasta.

Kieli	**Hyvät puolet**	**Huonot puolet**
Java	- Alalla paljon käytetty kieli - Paljon erilaisia apuvälineitä huonojen puolien kiertämiseen	- Aloittelijalle hankala - I/O:n käyttö hankala
Python	- Koko ajan enempi käytetty - Helpompi aloittelijalle kuin esimerkiksi Java	- Dynaaminen tyypitys - Kapselointi vain nimeämiskäytäntönä
C#	- Hyvin lähellä Javaa	- Aloittelijalle hankala kuten Java - Ohjelmat toimivat vain Windowsissa
C++		- Ei puhdas oliokieli

		- Ei turvallinen - Ei standardi tukea graafisen käyttöliittymän kehittämistä
Eiffeil	- Opetukseen suunniteltu kieli	- Ei ammattilaisten paljon käyttämä
Opetukseen suunnitellut kielet	- Suunniteltu opetukseen	- Ei suoraan käyttöä työelämässä (voi vähentää opiskelijoiden motivaatiota)

Mikään yleisesti käytössä oleva olio-ohjelmointikieli ei ole yksi selitteisesti paras opetukseen. Olio-ohjelmoinnin opetukseen suunnitellut kielet olisivat varmaankin parhaita opetukseen, ovathan ne siihen suunniteltuja, mutta ne eivät ole saaneet laajempaa menestystä. Luultavasti, koska kielen valintaan vaikuttavat monet muutkin tekijät kuten se, kuinka hyödyllinen se on opiskelijan tulevan työllistymisen kannalta. Kielen valitseminen olio-ohjelmoinnin opetuksessa on erityisen olennaista, kun olio-ohjelmointia opetetaan ohjelmoinnin perusteita opettavalla kurssilla. Muuten on loogistakin käyttää olioiden opettamiseen samaa kieltä, jota on aikaisemmilla kursseilla jo käytetty.

3.4 Työvälineet opetuksen avuksi

Olio-ohjelmoinnin opetukseen käytettyjen työkalujen yksi keskeinen piirre on, että ne tekevät ajatuksen ja koodin tasolla abstraktit oliot näkyviksi ja ymmärrettäviksi. Visuaaliset elementit tarjoavat konkreettisia ja oikeaan elämään perustuvia skenaarioita oliokäsitteistä. (Yuen & Liu, 2010.) Multimedia voi esittää vertauskuvia olioista, jolloin opiskelija voi nähdä koodinsa heräävän eloon uudella tavalla. Interaktiivisen multimedian apuna käyttämisen opetuksessa on tarkoitus lisätä erilaisten esitystasojen kautta ymmärrystä olio-ohjelmoinnista ja auttaa sen oppimisessa. (Bierre, Ventura, Phelps & Egert, 2006.)

Toinen tehtävä työkaluilla on antaa opiskelijoille motivoiva ympäristö (Bierre, Ventura, Phelps & Egert, 2006). Interaktiivisuus ja välitön palaute, jonka nämä työkalut voivat antaa opiskelijalle on palkitsevaa ja tekee oppimisesta hauskaa. Tämä tukee ohjelmoinnin oppimista suuresti. (Kölling, 1999b.) Monet opettajat uskovatkin, että abstraktien ohjelmointikäsitteiden oppimista kannattaa tukea visuaalisilla työkaluilla. Eteenkin jos oliot opetetaan ensin, ovat työkalut oleellisia, jotta olio-ohjelmoinnin käsitteet saadaan konkreettisimmiksi jo kurssin alussa. Kiistelyä on kuitenkin siitä, mikä on paras tapa. (Bruce, 2005; Ragonis & Ben-Ari, 2005.)

Olio-ohjelmoinnin käsitteiden tutuksi tekemiseen onkin suunniteltu monenlaisia eri ohjelmia avuksi. Olio-ohjelmoinnin opetuksessa auttavat työvälineet voidaan Georgetakin ja Retaliksen (2007) mukaan jakaa viiteen kategoriaan. Nämä kategoriat esimerkkeineen ja ominaisuuksineen esitellään taulukossa 2.

TAULUKKO 2 Eri työvälinekategoriat olio-ohjelmoinnin opettamiseen.

Kategoria	**Esimerkki**	**Ominaisuudet**
Ohjelmoitavat mikromaailmat	- Karel the robot (eri versioita eri kielille) - Alice	Visuaalinen maailma, jossa näkyviin asioihin voidaan vaikuttaa. Yksinkertainen ohjelmointikieli tai valikoiden ja kuvakkeiden avulla ohjelmointi.
Peliympäristöt	- Robocode	Peli, jota pelataa ohjelmoimalla. Esim. Robocodessa ohjelmoidaan olio-ohjelmoinnin käsitteitä käyttäen robotteja, jotka taistelevat keskenään.
Opetukseen suunniteltuihin kieliin liittyvät välineet	- IOPL:n ohjelmointiympäristö	Olio-ohjelmoinnin opettamiseen sunniteltujen kielien ohjelmointiympäristöt
Opetukseen suunnitellut ohjelmointiympäristöt	- BlueJ - DrJava	Voivat esittää olio-ohjelmoinnin elementtejä graafisesti (esim. UML). Käyttävät ammattilaisten käyttämiä kieliä tai yksinkertaistettuja versioita niistä.
Oliot ensin -opetustapaa tukevat välineet	- Monet edellä mainituista esim. BlueJ ja Alice. - Olio-ohjelmointia tukevat grafiikka kirjastot	Piilottaa olio-ohjelmointikielen yksityiskohdat. Mahdollistaa käsitteiden esittelemisen pikkuhiljaa yksinkertaisista vaikeampiin.

Ohjelmointiympäristöt ovat tärkein näistä työkaluista, sillä niitä käytetään eniten (Georgantaki & Retalis, 2007). Lisäksi ohjelmointiympäristö on tärkeä tekijä olio-ohjelmointia opetettaessa, erityisesti, jos olio-ohjelmointia opetetaan ohjelmoinnin alkeita opettavalla kurssilla. Ohjelmointiympäristön valinta liittyy myös kielen valintaan, sillä eri kielille on eri ympäristöt. (Kölling, 1999b.)

Kölling (1999b) määrittelee olio-ohjelmoinnin opettamiseen sopivan ohjelmointiympäristön tärkeimmäksi piirteeksi käytön helppouden, sillä liian tehokkaat ja kattavat työkalut ovat hankalia aloittelijoille. Graafinen käyttöliittymä on myös lähes välttämätön. Työkalujen pitäisi olla integroituja, sillä yhden ohjelman pitäisi riittää kaikkiin tarpeisiin. Ympäristön tulisi tukea olioita, mieluiten niin, että olioita voidaan luoda interaktiivisesti. Ympäristön pitäisi tukea myös koodin uudelleenkäyttöä, opiskelua ja ryhmätyötä, eikä se saa olla liian kallis. (Kölling, 1999b.) Mielenkiintoista on, että toisin kuin kielistä puhuttaessa, yksi kriteeri hyvälle ohjelmointiympäristölle ei ole, että se on IT-alalle oleellinen ja siitä on helppo siirtyä ammattilaisten käyttämiin ympäristöihin.

Java on suosituin opetuskieli, joten myös ohjelmointiympäristöjä käsittelevää tutkimusta on eniten Javasta (Dale, 2005; Olan, 2004; Mason, Cooper & de Raadt, 2012). Opetusalan ammattilaiset ovat kehittäneet pedagogisia ohjelmointiympäristöjä tehdäkseen Javasta helpommin lähestyttävän aloittelijoille. Tämä

on hyödyllistä erityisesti, jos halutaan keskittyä alusta asti olio-ohjelmoinnin käsitteisiin. (Hsia, Simpson, Smith & Cartwright, 2005; Olan, 2004).

Käytetyin näistä opetukseen suunnitelluista ohjelmointiympäristöistä on BlueJ (Olan, 2004; Mason, Cooper & de Raadt, 2012). Siinä on yksinkertaisempi ja intuitiivisempi käyttöliittymä kuin ammattilaiskäyttöön tarkoitetuissa ympäristöissä. Se mahdollistaa interaktiivisen kokeilun Javan rakenteilla, sillä se tekee pääohjelman määrittämisen tarpeettomaksi. Näin yksittäisillä olioilla voidaan tehdä kokeiluja ilman kokonaisen ohjelman kirjoittamista. (Hsia, Simpson, Smith & Cartwright, 2005; Olan, 2004.)

BlueJ tarjoaa graafisen ympäristön, jossa opiskelijat voivat luoda olioita ja suorittaa metodeja ilman koodin kirjoittamista (Hsia, Simpson, Smith & Cartwright, 2005). Se tukeekin vahvasti olioiden opettamista ensimmäiseksi ohjelmointikurssilla. BlueJ antaa mahdollisuuden luoda UML-kaavioita, joiden pohjalta koodi luodaan. Luokan kuviota klikkaamalla auki saa editorin, jolla voi kirjoittaa luokkaan liittyvää koodia. Parametrit ja metodien tulokset esitellään dialogi-ikkunoiden avulla. (Olan, 2004; Kölling & Rosenberg, 2001.)

BlueJ:n ongelmaksi on kuitenkin todettu, että koska siinä käytetään staattisia visualisaatioita olioista, voi opiskelijoilla olla vaikeuksia siirtyä kokonaisten ohjelmien kirjoittamiseen. BlueJ ei nimittäin auta ymmärtämään, kuinka ohjelma etenee käytännössä. Opiskelijoilla oli BlueJ:llä käydyllä kurssilla ongelmia ymmärtää esimerkiksi metodin vaikutusta olion tilaan, metodien kutsumista, mistä parametrit tai paluuarvot tulevat, konstruktoreita ja ohjelman suorittamisen kokonaiskuvaa. Hyödyt siitä, että opiskelun alussa käytetään staattista visualisaatiota voivatkin muuttua haitoiksi opiskeluissa pidemmälle mentäessä. (Ragonis, & Ben-Ari, 2005.) Pelkkä opetukseen kehitetty ympäristö ei riitäkään, vaan optimaalista on, että opiskelijat käyttäisivät sen lisäksi myös ammattilaisille tarkoitettua ohjelmointiympäristöä (Georgantaki, & Retalis, 2007).

Myös parasta ohjelmointiympäristöä olio-ohjelmoinnin opetukseen etsivissä tutkimuksissa painotus on niiden sopivuudessa ohjelmoinnin alkeita opettaville kursseille. Kun olio-ohjelmointia opetetaan vasta myöhemmin opinnoissa, ei ympäristö ole ehkä niin oleellinen, koska opiskelijoilla on jo jonkinlainen käsitys ohjelmointiympäristöistä.

Objektiivista tutkimusta eri työkalujen vaikutuksista olio-ohjelmoinnin oppimiseen ei ole juuri tehty. Ehdottomasti parasta työkalua opettamiseen onkin luultavasti vaikea sanoa, sillä työkalun valinta riippuu aina monista tekijöistä kuten käytetystä kielestä ja kurssin rakenteesta. Pelkät opetustyökalut eivät kuitenkaan riitä, vaan opiskelijoiden tulisi käyttää myös ammattilaisten työkaluja ohjelmointikursseilla.

Onkin kyseenalaistettu, voivatko opetustyökalut tai -ympäristöt edes ratkaista ongelmia olio-ohjelmoinnin käsitteiden oppimisessa (Ragonis & Ben-Ari, 2005; Hu, 2004). Esimerkiksi BlueJ:tä käyttäen on saatu tulokseksi, että sen käyttö ei olisi merkittävästi parantanut opiskelijoiden suorituksia ohjelmointikurssilla, mutta asia vaatii yhä lisää tutkimusta (Lee & Dalgarno, 2008). Jos opetukseen käytetyt työkalut olisivat pääsyy opiskelijoiden oppimisvaikeuksiin, olisi

asia saatu luultavasti jo ratkaistua (Hu, 2004). Työkalut ovatkin vain yksi osa monimutkaisessa ongelmassa.

3.5 Erilaisia tapoja opettaa olio-ohjelmointia

Opetukseen käytetty kieli ja työvälineet vaikuttavat oppimiseen, mutta hyvin paljon siihen vaikuttaa myös se, miten asiat opetetaan. Huonot opetuskäytänteet voivat nimittäin vaikeuttaa olio-ohjelmoinnin käsitteiden oppimista (Sivasakthi & Rajendran, 2011). Opetuskielen ja -ohjelmointiympäristön valinta ovat erityisen tärkeitä, kun oliot opetetaan ensimmäisen ohjelmointkurssin alussa. Opetustapa on sen sijaan yhtä lailla tärkeä huolimatta siitä, missä vaiheessa olio-ohjelmoinnin peruskäsitteet tuodaan esille.

Eckerdalin ja Thunén (2005) tutkimuksen mukaan opiskelijat voivat ymmärtää luokan ja olion käsitteet kolmella eri tasolla. Näistä jokainen taso syventää edellisen tason ymmärrystä ja lisää uuden ulottuvuuden edellisten päälle. Tasot on esitetty taulukossa 3. Ensimmäisellä tasolla olio ja luokka ymmärretään vain koodin tasolla tekstinä ja rakenteina. Syvimmällä, kolmannella, tasolla olio ja luokka puolestaan käsitetään jonkin oikean maailman ilmiön mallina.

TAULUKKO 3 Opiskelijoiden eri ymmärrystasot olioista ja luokista

Taso	**Ymmärrys olioista**	**Ymmärrys luokista**
1	Olio on koodia	Luokka on ohjelman osa, joka muodostaa koodin rakenteen.
2	Olio on jotain, joka on aktiivinen ohjelmassa.	Luokka kuvaa olion käyttäytymisen ja ominaisuudet.
3	Olio on jonkin oikean maailman ilmiön malli.	Luokka kuvaa jonkin oikean maailman ilmiön ominaisuudet ja käyttäytymisen.

Suurin osa opiskelijoista pääsee tasolle kaksi. Vain harvat pääsevät tasolle kolme. Olisi kuitenkin suotavaa, että huomattavasti suurempi määrä opiskelijoista pääsisi tälle tasolle, joka ilmentää syvää osaamista. (Eckerdal & Thuné, 2005.) Opiskelijoille siis pitäisi tuoda ilmi oikean maailman ja olio-ohjelmoinnin peruskäsitteiden yhteys.

Zhu ja Zhou (2003) haluavatkin tuoda yksittäisiä olio-ohjelmoinnin käsitteitä opetettaessa esille koko ajan niiden yhteyden todellisuuteen. Olio-ohjelmointi vastaa ihmisen luonnollista ajattelua, jossa on yleisiä kategorioita ja niistä erikoistapauksia, joten sitä voi hyvin selventää esimerkeillä todellisuudesta. Zhu ja Zhou antavat seuraavat viisi askelta olio-ohjelmoinnin peruskäsitteiden opettamiseen:

- Olio-ohjelmoinnin perusperiaatteiden yhteys tavalliseen ajatteluun tulee tuoda esille.

- Olio-käsite kannattaa esitellä oikean maailman havaintojen kautta. Luokan käsitteen puolestaan voi esitellä monien samanlaisten olioiden abstraktiona.
- Olion ilmentymän luomisen voi opettaa, kun luokan käsite on opittu.
- Aliluokat voi esitellä lisäämällä yksityiskohtia olemassa olevaan luokkaan. Yläluokat puolestaan voi tuoda esille löytämällä monille eri luokille yhteisiä ominaisuuksia.

Näillä ohjeilla opiskelijat voivat paremmin käsittää olio-ohjelmoinnin peruskäsitteet ja soveltaa niitä. Yleensä ottaen uudet käsitteet ja periaatteet tulisikin esitellä jokapäiväiseen elämään liittyvien esimerkkien avulla.

Luokan ja olion ero voi olla opiskelijoille erityisen vaikea asia hahmottaa, joten sitä tulisi korostaa niin usein kuin mahdollista. Kannattaa pysyä yhdessä selityksessä ja antaa paljon esimerkkejä. (Sanders, Boustedt, Eckerdal ym., 2008.) Opiskelijoille annettavien koodiesimerkkien tulee kuitenkin olla hyviä väärinkäsitysten välttämiseksi. Jotta vältettäisiin tutkimuksissa esille tulleet opiskelijoiden väärät käsitykset olioista ja luokista, pitää seuraavat asiat ottaa huomioon. Esimerkkeinä olevissa luokissa pitää olla useampi kuin yksi attribuutti. Olioiden tiedon säilytys puoli ei saa korostua, joten kun oliolle annetaan viesti, pitää olion antaman vastauksen muuttua olennaisesti olion tilasta riippuen. Esimerkeissä pitäisi olla useampi ilmentymä samasta luokasta, jolloin ero luokan ja olion välillä tulee selkeämmäksi. (Eckerdal & Thuné, 2005.)

Yksi tapa tuoda olio-ohjelmoinnin käsitteet paremmin opiskelijoille esille on käsikirjoitettujen roolileikkien avulla. Toiminta, joka saa opiskelijat osallistumaan sen sijaan, että he olisivat vain passiivisia katsojia, saa oppimaan paremmin. Olio-ohjelmoinnin käsitteet sopivat hyvin näytellyiksi, koska olioita voi verrata ihmiseen. Myös ihmiset ovat tiedon kätkeviä autonomisia toimijoita, jotka kommunikoivat keskenään. Esimerkiksi jo ensimmäisellä luennolla voidaan tuoda olioiden olemusta esille harjoituksella, jossa opettaja on pääohjelma ja käskee oppilaita nimeltä tekemään tiettyjä etukäteen sovittuja toimintoja. Tämä tuo esille olion, metodin, parametrin ja paluuarvon käsitteet. (Andrianoff & Levine, 2002.)

Näiden harjoituksien vaikutusta parantaa, jos kokemuksesta keskustellaan jälkeenpäin ja niihin viitataan myöhemmin opiskelijoiden oppiessa lisää. Nämä harjoitukset auttavat erityisesti hahmottamaan ohjelman etenemistä monimutkaisissa oliopohjaisissa järjestelmissä, sillä niissä voidaan hyvin tuoda esille järjestelmän dynaaminen toiminta. (Andrianoff & Levine, 2002.)

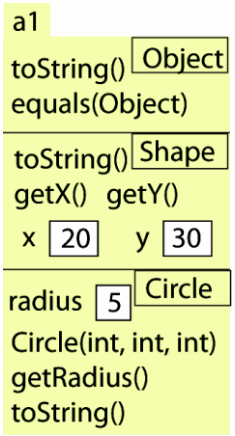
Toinen tapa yrittää selkeyttää olio-ohjelmoinnin käsitteitä on luoda vertauskuva, jota voidaan käyttää yhtenäisesti koko opetuksen ajan. Tällä tavalla käsitteet saadaan tehtyä opiskelijoille konkreettisemmiksi. Gries (2008) tarjoaa tällaiseksi vertauskuvaksi arkistokaapin, jossa jokaiselle luokalle on oma lokero. Seuraavassa kuvataan tätä Griesin vertauskuvaa tarkemmin.

Lokeron sisällä on kansioita, joista jokainen edustaa oliota. Jokaisessa kansiossa on olion sisältämät attribuutit ja metodit. Vertauskuvaa laajennetaan kuvaamaan metodeja kansioissa olevina ohjeina. Esimerkiksi hammaslääkärin

toimistossa on sihteerillä ja hammashoitajalla eri tehtävänsä, mutta laittamalla jokaiseen kansioon ohjeet kaikista kansiolle tehtävistä toiminnoista, kuka tahansa voi suorittaa nämä tehtävät.

Erityisesti Gries korostaa sitä kuinka olioilla on aina niille määrätty ainutlaatuinen nimi, jonka perusteella olio eli kansio voidaan etsiä arkistosta. Kaksi muuttujaa, joilla on eri nimet ja jotka viittaavat samaan olioon, vertautuvat hammaslääkärin toimiston kahteen eri työntekijään, joiden kummankin pitää päästä käsiksi tiettyyn kansioon. Näin saadaan vertauskuvalla esille muuttujaan sijoitetun olion käsite ilman tarvetta puhua esimerkiksi muistipaikoista. Kun tarkoituksena on opetella olio-ohjelmointia, muistipaikoista ja tietokoneen toiminnan yksityiskohdista puhuminen voi turhaan hämmentää opiskelijoita.

Tärkeää tässä vertauskuvassa on myös se, että oliot voidaan piirtää näkyville. Kuvassa 2 on näin piirretty olio, joka esittää myös perinnän ilmituomisen kuvassa. Ylhäällä vasemmalla on olion nimi. Vasemmassa reunassa laatikossa on puolestaan luokka. Jokainen peritty luokka metodeineen ja attribuutteineen on erotettuna toisista ja alimpana on luokka joka perii edelliset. Näin ollen kun olion metodia kutsutaan, aletaan sitä etsiä alemmasta luokasta ja noustaan ylöspäin kunnes se löytyy. Metodien järjestyksellä ei ole mitään väliä piirustuksessa. Tällä kuvalla voidaan tehdä opiskelijoille konkreettisemmaksi luokan ideaa.



KUVIO 2 Oliosta piirretty kuva

Annetun vertauskuvan pitää olla johdonmukainen ja antaa opiskelijoille tapa nähdä abstrakti olioiden käsite jollain konkreettisella tavalla. Suurin osa opiskelijoista, joille tämä vertauskuva esiteltiin, kokivat sen auttavan. Osa tosin ei kokenut tarvetta minkäänlaisille vertauskuville.

Koska Java on käytetyin kieli ohjelmoinnin opetuksessa, on myös sen käyttämisestä olio-ohjelmointiin kirjoitettu eniten (esim. Bennedson & Caspersen, 2007; Mason, Cooper & de Raadt, 2012; Davies, Polack-Wahl, & Anewalt, 2011). Kölling ja Rosenberg (2001) antavat seuraavia ohjeita olio-ohjelmoinnin opettamiseen Javalla, mutta ne pätevät myös paljolti muilla kielillä opetettaessa. Kurssin alussa opiskelijoiden ei pitäisi aloittaa puhtaalta pöydältä vaan muokata olemassa olevaa koodia. Hello world -ohjelmaa ei muutenkaan pitäisi esitellä opiskelijoille, sillä se ei tuo esille olio-ohjelmointia. (Kölling & Rosenberg, 2001.)

Kopiointi ja muokkaus ovatkin ensimmäisiä askelia, kun opetellaan ohjelmoimaan. Opettajien esittelemien ohjelmamallien pitäisi olla sopivia suhteessa opiskelijoille annettuihin tehtäviin, jotta opiskelijat voivat soveltaa niitä. Niiden pitäisi olla tarpeeksi yleisiä, että niillä voi tehdä monenlaisia tehtäviä, muttei kuitenkaan liian hienostuneita. (Berge & Borge, Fjuk, Kaasbøll & Samuelsen, 2003.) Opiskelijoiden pitäisikin lukea koodia, mutta esimerkkien tulisi olla hyviä (Kölling & Rosenberg, 2001).

Main-metodin käyttämisen välttäminen kurssin alussa tuo esille oliot paremmin, sillä main-metodi ei ole olio-ohjelmoinnissa keskeinen osa (Kölling & Rosenberg, 2001). Toisaalta sen opettamista ei pidä viivyttää liian myöhään, sillä muuten opiskelijat eivät ymmärrä ohjelman etenemistä ja olioiden dynaamista toimintaa. Jos käytetään työkalua, jossa esimerkiksi voi olioille määrätä suoraan arvoja, kuten BlueJ:ssä, pitää pääohjelma esitellä kuitenkin suhteellisen aikaisin. Muuten opiskelijoilla on vaikeuksia käsittää esimerkiksi, missä oliot luodaan ja mitä tapahtuu paluuarvolle. Olion tilan käsite pitää myös opettaa selkeästi ja siitä pitää tehdä harjoituksia. (Ragonis, & Ben-Ari, 2005.)

Käyttöliittymät ovat ongelma Javalla ohjelmoitaessa, sillä oppilaan pitäisi jotenkin saada tulos näkymään ja antaa syötteitä, mutta syötteiden antaminen Javassa on monimutkaista. Opiskelijat joutuvat helposti vain kopioimaan koodia, jota eivät ymmärrä, mikä ei ole hyvä asia. Tämän takia esimerkiksi edellisessä luvussa mainittu BlueJ mahdollistaa parametrien antamisen suoraan. Oppilaille voidaan myös antaa koodia, jossa käyttöliittymä on valmiiksi kirjoitettu, jolloin oppilaat voivat keskittyä olennaisiin asioihin. (Kölling & Rosenberg, 2001.)

Olio-ohjelmoinnin hyötyjä on vaikea tuoda esille pienillä ja yksinkertaisilla tehtävillä. Opetuksessa kannattaakin käyttää isoja projekteja, sillä vasta ne tuovat esille olioille perustuvien rakenteiden hyödyt kunnolla. Isoissa projekteissa on muitakin hyviä puolia, kuten koodin lukeminen, dokumentoinnin tärkeyden tajuaminen ja luokkien rajapintojen ymmärtäminen. (Kölling & Rosenberg, 2001.) Eteenkin jos olio-ohjelmointia ei opeteta ensimmäisellä ohjelmointikurssilla, suurempi projekti on hyvä lähestymistapa (Chen, 2007). Ensimmäisellä kurssilla, kun käytetään oliot ensin -lähestymistapaa, ei ole välttämättä käytännöllistä esitellä opiskelijoille isompia projekteja, ainakaan kurssin alussa (Gries, 2008).

Nykyään isompana projektina tarjotaan pelien tekemistä, sillä se saa opiskelijat motivoitumaan paremmin opiskeltavaan asiaan. Pelit ovat myös helposti lähestyttävä aihe, sillä ne ovat opiskelijoille tuttuja ja mielenkiintoisia. Peliprojekti on niin iso, että monien eri luokkien käyttäminen on luontevaa, joten olio-ohjelmoinnin käsitteet ja edut on helppo tuoda esille. (Chen, 2007.)

Kokonaisuudessaan viimeaikaiset tutkimukset korostavat kokonaisuuden opettamista pelkkien kielen yksityiskohtien sijaan. Myös käsitteiden kuvaamiseen tosielämän esimerkkien avulla kannattaa panostaa. Opiskelijoille vaikeita asioita, vaikka ne olisivat teoriassa yksinkertaisiakin, kuten luokan ja olion eroa, pitää korostaa erityisesti.

4 YHTEENVETO

Olio-ohjelmoinnin opettamista pidetään vaikeana asiana. Ratkaisua on etsitty opettavasta ohjelmointikielestä, ohjelmointiympäristöstä, opetusmetodeista sekä järjestyksestä, jossa asiat opetetaan. Mikään näistä ei ole kuitenkaan antanut yksinkertaista ja tehokasta ratkaisua opiskelijoiden oppimisongelmiin.

Olio-ohjelmoinnin opetuksessa kyse on näiden kaikkien osa-alueiden kokonaisuudesta. Mistään näistä osa-alueista ei voi löytää parasta ratkaisua, vaan kyse on aina kokonaisuuden rakentamisesta, sillä erilaisille kursseille, opettajille ja opiskelijoille sopivat parhaiten hiukan eri lähestymistavat.

Opiskelijoiden motivointi ja saaminen pysymään kurssilla on yksi tutkimuksissa esiin tuleva teema. Tähän tarjotaan erilaisia apuvälineitä ja esimerkiksi pelien esiin tuomista. Samalla toivotaan, että opiskelijat voivat oppia ymmärtämään syvemmin olio-ohjelmoinnin käsitteitä.

Tässä tutkielmassa käsiteltiin olio-ohjelmoinnin opettamista pääasiassa ensimmäisellä ohjelmointikurssilla. Ehkä opettaminen ei ole myöhemmillä kursseilla niin ongelmallista, että asiasta olisi tehty paljon tutkimusta. Toisaalta tutkimus opetuksesta ensimmäisellä ohjelmointikurssilla painottaa olio-ohjelmoinnin käsitteiden opettamista kurssin alusta asti. Olioiden opettaminen ensin on erityisen haastavaa, koska on jouduttu siirtymään opetuksessa pois vanhasta proseduraalisesta ohjelmointiparadigmasta, minkä takia siitä on erityisesti tutkimusta.

Olio-ohjelmoinnin opettaminen liittyykin vahvasti ohjelmoinnin opettamiseen ja siihen pätevät samat asiat kuin yleisesti ohjelmoinnin opettamiseen ja jopa opettamiseen yleensä ottaen. Tämän takia onkin vaikea rajata, mitä kaikkea ottaa olio-ohjelmointia käsittelevään tutkimukseen, koska aihealue liittyy niin kiinteästi ohjelmoinnin opetukseen kokonaisuudessaan. Tässä tutkielmassa rajaus on tehty aika laajasti ja sen olisi voinut tehdä tiukemminkin keskittyen asioihin, jotka liittyvät ainoastaan olio-ohjelmoinnin käsitteiden opettamiseen.

Olio-ohjelmointi kytkeytyy läheisesti myös olio-ohjelmien suunnitteluun. Tässä tutkielmassa ei ole otettu kantaa juurikaan ohjelmien suunnitteluun olioita käyttäen. Kuitenkin olio-ohjelma pitää suunnitella ennen kuin sen voi ohjel-

moida, joten olio-ohjelmointia opetettaessa pitää ottaa huomioon myös se, miten ja missä vaiheessa olio-ohjelmien suunnittelu tuoda esille.

Tämä tutkielma on vain pintaraapaisu hyvin laajasta aiheesta, joten jatkotutkimusta voisi tehdä syventyen yhteen osa-alueeseen monista tässä esitellyistä. Tällainen jatkotutkimus voisi olla kirjallisuuskatsaus liittyen ainoastaan ohjelmointikieliin, opetustyökaluihin tai opetustapoihin. Jatkotutkimusta voisi tehdä myös empiirisesti tutkimalla erilaisten opetustapojen tai -työkalujen vaikutuksia oppimistuloksiin. Tällainen tutkimus on vaikeaa, mutta antaisi silti jonkinlaisen kuvan siitä, mitkä voisivat olla parhaita ratkaisuja missäkin tilanteessa.

LÄHTEET

Andrianoff, S. K. & Levine, D. B. (2002). Role playing in an object-oriented world. Teoksessa Proceedings of the 33rd SIGCSE technical symposium on Computer science education Covington, KY, USA, February 26 - March 02 (s. 121-125). New York: ACM New York. Haettu 10.6.2012 osoitteesta dl.acm.org/citation.cfm?id=563386

Bennedsen, J. & Caspersen, M. E. (2003). A Model-First Approach to Teaching Introductory Object-Orientation. COOL Workshop on Learning and Teaching Object-Orientation -- Scandinavian Perspectives Oslo, October 2003. Haettu 10.6.2012 osoitteesta http://cs.au.dk/~mec/publications/workshop/00--cool2003.pdf

Bennedsen, J. & Caspersen, M. E. (2004). Teaching Object-Oriented Programming - Towards Teaching a Systematic Programming Process. Teoksessa Proceedings of the Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts Oslo, Norway, 14-18 June. Haettu 10.6.2012 osoitteesta http://cs.au.dk/~mec/publications/workshop/11--ecoop2004.pdf

Bennedson, J. & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. ACM SIGCSE Bulletin, 39, 2, 32-36. Haettu 10.6.2012 osoitteesta http://users-cs.au.dk/mec/publications/journal/25--bulletin2007.pdf

Berge, O., Borge, R. E., Fjuk, A., Kaasbøll, J. & Samuelsen, T. (2003). Learning Object-Oriented Programming. Teoksessa Norsk informatikkonferanse 2003, Oslo, Norvège, 24.-26. november. Haettu 10.6.2012 osoitteesta http://www.nik.no/2003/Bidrag/Berge.pdf

Berges, M. & Hubwieser, P. (2011). Minimally Invasive Programming Courses – Learning OOP With(out) Instruction. Teoksessa Proceedings of the 42nd ACM technical symposium on Computer science education Dallas, TX, USA, March 09 – 12 (s. 87-92). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1953195

Bierre, K., Ventura, P., Phelps, A. & Egert, C. (2006) Motivating OOP by Blowing Things Up: An Exercise in Cooperation and Competition in an Introductory Java Programming Course. Teoksessa Proceedings of the 37th SIGCSE technical symposium on Computer science education. New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1121452

Black, A., Bruce, K. B & Noble, J. (2010). Panel: designing the next educational programming language. Teoksessa Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion Reno, NV, USA, October 17 – 21 (s. 201-204). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://gracelang.org/documents/panelAbstract.pdf

Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K. & Zander, C. (2007). Threshold Concepts in Computer Science: Do they exist and are they useful? Teoksessa Proceedings of the 38th SIGCSE technical symposium on Computer science education Covington, KY, USA, March 07 – 11 (s. 504 - 508). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1227482

Bruce, K. B. (2005). Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. ACM SIGCSE Bulletin, 37, 2, 111-117. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1083477

Chen, W. (2007). Teaching Object-Oriented Programming Laboratory With Computer Game Programming. IEEE Transactions on Education, 50, 3, 197 – 203. Haettu 10.6.2012 osoitteesta http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287104

Computing Curricula 2001: Final Report (2001). Haettu 10.6.2012 osoitteesta http://www.acm.org/education/curric_vols/cc2001.pdf

Computer Science Curricula 2013, Strawman Draft (2012). Haettu 10.6.2012 osoitteesta http://ai.stanford.edu/users/sahami/CS2013/strawman-draft/cs2013-strawman.pdf

Computer Science Curriculum 2008: An Interim Revision of CS 2001 (2008). Haettu 10.6.2012 osoitteesta http://www.acm.org/education/curricula/ComputerScience2008.pdf

Craig, I. D. (2007). Object-Oriented Programming Languages: Interpretation. London: Springer. Haettu 10.6.2012 osoitteesta

Dale, N. (2005). Content and emphasis in CS1. ACM SIGCSE Bulletin Homepage archive, 37, 4, 69-73. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1113880

Davies, S., Polack-Wahl, J. A. & Anewalt, K. (2011). A snapshot of current practices in teaching the introductory programming sequence. Teoksessa Proceedings of the 42nd ACM technical symposium on Computer science education Dallas, TX, USA, March 09 - 12 (s. 625-630). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1953339

Eckerdal, A. & Thuné, M. (2005). Novice Java programmers' conceptions of "object" and "class", and variation theory. Teoksessa Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education Caparica, Portugal, June 26 - 29 (s. 89 - 93). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1067473

Ekuobase, G.O., Akwukwuma, V.V.N., Egbokhare, F.A. (2009). Introducing Object Oriented Programming in Tertiary Institutions: Which Language Is Most Appropriate? Research Journal of Applied Sciences, 4, 1, 41-49. Haettu 10.6.2012 osoitteesta http://www.medwelljournals.com/fulltext/?doi=rjasci.2009.41.49

Ehlert, A., & Schulte, C. (2009). Empirical Comparison of Objects-First and Objects-Later. Teoksessa Proceedings of the fifth international workshop

on Computing education research workshop Berkeley, CA, USA, August 10 - 11 (s. 15-26). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1584326

Georgantaki, S., Retalis, S. (2007). Using Educational Tools for Teaching Object Oriented Design and Programming. Journal of Information Technology Impact, 7, 2, 111-130. Haettu 10.6.2012 osoitteesta www.jiti.net/v07/jiti.v7n2.111-130.pdf

Giangrande, E. (2007). CS1 programming language options. Journal of Computing Sciences in Colleges archive, 22, 3, 153-160. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1181881

Golwasser, M. H. & Letscher, D. (2008). Teaching an object-oriented CS1 -: with Python. Teoksessa Proceedings of the 13th annual conference on Innovation and technology in computer science education Madrid, Spain, June 30 - July 02 (s. 42-46). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1384285

Gries, D. (2008). A principled approach to teaching OO first. Teoksessa Proceedings of the 39th SIGCSE technical symposium on Computer science education Portland, OR, USA, March 12 - 15 (s. 31-35). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1352149

Harrison, C.J., Sallabi, O.M. & Eldridge, S.E. (2005). An initial object-oriented programming language (IOPL) and its implementation. IEEE Transactions on Education, 48, 1, 3 – 10. Haettu 10.6.2012 osoitteesta http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1393096

Hsia, J. I., Simpson, E., Smith, D. & Cartwright, R. (2005). Taming Java for the Classroom. Teoksessa Proceedings of the 36th SIGCSE technical symposium on Computer science education St. Louis, MO, USA, February 23 – 27 (s. 327-331). New York: ACM New York, NY. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1047459

Hu, C. (2004). Rethinking of Teaching Objects-First. Education and Information Technologies, 9, 3, 209-218. Haettu 10.6.2012 osoitteesta http://www.springerlink.com/content/p43x147385675607/

Koskimies, K. (2000). Oliokirja. Helsinki: Satku – Kauppakaari.

Kölling, M. & Rosenberg, J. (1996). Blue - A Language for Teaching Object-Oriented Programmin. Teoksessa Klee, K. J. (toim.), Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education Philadelphia, PA, USA, February 15 – 17 (s. 190-194). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=236462.236537

Kölling, M. (1999a). The problem of teaching object-oriented programming. Part I: Languages. Journal of Object-Oriented Programming, 11, 8, 8-15. Haettu 10.6.2012 osoitteesta http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.8492

Kölling, M. (1999b). The Problem of Teaching Object-Oriented Programming, Part 2: Environments. Journal of Object-Oriented Programming, 11, 9, 6-12.

Haettu 10.6.2012 osoitteesta http://kar.kent.ac.uk/21839/1/the_problem_of_teaching_object-oriented_kolling.pdf

Kölling, M. & Rosenberg J. (2001). Guidelines for teaching object orientation with Java. Teoksessa Proceedings of the 6th annual conference on Innovation and technology in computer science education (s. 33 - 36). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=507758.377461

Lee, M. J. W. & Dalgarno, B. (2008). The Effectiveness of Screencasts and Cognitive Tools as Scaffolding for Novice Object-Oriented Programmers. Journal of Information Technology Education, 7, 61-80. Haettu 10.6.2012 osoitteesta http://www.jite.org/documents/Vol7/JITEv7p061-080Lee332.pdf

Mannila, L. & de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. Teoksessa Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006 Koli, Joensuu, Finland, November 09 - 12 (s. 32-37). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1315811

Mason, R., Cooper, G. & de Raadt, M. (2012). Trends in Introductory Programming Courses in Australian Universities – Languages, Environments and Pedagogy. Teoksessa M. de Raadt & A. Carbone (toim.), Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012), Melbourne, Australia, 30 January - 3 February (s. 33-42). Melbourne, Australia: ACS. Haettu 10.6.2012 osoitteesta http://crpit.com/confpapers/CRPITV123Mason.pdf

Olan, M. (2004). Dr. J vs. the bird: Java IDE's one-on-one. Journal of Computing Sciences in Colleges archive, 19, 5, 44-52. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1060089

Oldham, J. D. (2005). What happens after Python in CS1? Journal of Computing Sciences in Colleges archive, 20, 6, 7-13. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1060408

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. Teoksessa J. Carter & J. Amillo (toim.), Working group reports on ITiCSE on Innovation and technology in computer science education Dundee, Scotland Uk, June 23 – 27 (s. 204-223). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1345375.1345441

Ragonis, N. & Ben-Ari, M. (2005). On Understanding the Statics and Dynamics of Object-Oriented Programs. Proceedings of the 36th SIGCSE technical symposium on Computer science education St. Louis, MO, February 23-27 (s. 226-230). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1047425

Sivasakthi, M. & Rajendran, R. (2011). Learning difficulties of object-oriented programming paradigm using Java: students' perspective. Indian Journal of Science and Technolog, 4, 8, 983-985. Haettu 10.6.2012 osoitteesta http://www.indjst.org/archive/vol.4.issue.8/24-aug11sivasakthi.pdf

Sanders, K., Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Thomas, L. & Zander, C. (2008) Student Understanding of Object-Oriented Programmingas Expressed in Concept Maps Proceeding. Teoksessa Proceedings of the 39th SIGCSE technical symposium on Computer science education Portland, Oregon USA, March 11-16 (s. 332-336). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1352251

Stamouli, I. & Huggard, M. (2006). Object oriented programming and program correctness: the students' perspective. Teoksessa Proceedings of the second international workshop on Computing education research Canterbury, UK, September 9-10 (s. 109 – 118). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1151605

Streib, J. & Soma, T. (2010). Using Contour Diagrams and JIVE to Illustrate ObjectOriented Semantics in the Java Programming Language. Teoksessa Proceedings of the 41st ACM technical symposium on Computer science education Milwaukee, WI USA, March 10-13 (s. 510-514). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1734435

TIOBE Software (2012). TIOBE Programming Community Index for May 2012. Haettu 10.6.2012 osoitteesta http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Turner, S. A., Quintana-Castillo, R., Pérez-Quiñones, M. A. & Edwards, S. H. (2008). Misunderstandings about Object-Oriented Design: Experiences Using Code Reviews. Proceedings of the 39th SIGCSE technical symposium on Computer science education Portland, Oregon USA, March 11-16 (s. 97-101). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1352169

Wick, M. R., Stevenson, D. E & Phillips, A. T. (2004). Seven Design Rules for Teaching Students Sound Encapsulation and Abstraction of Object Properties and Member Data. Teoksessa Proceedings of the 35th SIGCSE technical symposium on Computer science education Norfolk, VA, March 3-7 (s. 100-104). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=971339

Yuen, T. T. & Liu, M. (2010). How Interactive Multimedia Authoring Transforms Object Oriented Thinking. Teoksessa Proceedings of the 41st ACM technical symposium on Computer science education Milwaukee, WI USA, March 10 - 13 (s. 426-430). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=1734408

Zhu, H. & Zhou, M. (2003). Methodology first and language second: a way to teach object-oriented programming. Teoksessa Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming,

systems, languages, and applications Anaheim, CA, USA, October 26 – 30 (s. 140-147). New York: ACM New York. Haettu 10.6.2012 osoitteesta http://dl.acm.org/citation.cfm?id=949389

Painamattomat lähteet:

Tikansalo, T. (2008). Tutkimus noviisien olio-ohjelmien suoritusta koskevista mentaalisista malleista. Pro gradu -tutkielma. Joensuun yliopisto, tietojenkäsittelytiede.