Ohjelmoinnin oppimisen haasteet ja ratkaisut

Oulun yliopisto
Tietojenkäsittelytieteen laitos
Kandidaatin tutkielma
Sebastian Körkkö
11.5.2016

Tiivistelmä

Ohjelmoinnin oppimisen haasteet ovat johtaneet ohjelmointikurssien valtaviin keskeytysprosentteihin ja ohjelmointia pidetäänkin yhtenä haastavimmista tietojenkäsittelytieteen aihepiireistä. Tämän kuvailevan kirjallisuuskatsauksen tarkoituksena oli hahmottaa, mitä ohjelmoinnin oppimisen haasteita opiskelijat kohtaavat ja löytää ratkaisuja löydettyihin ongelmiin. Haasteet olivat moninaisia ja vahvasti kognitiiviseen psykologiaan liittyviä.

Ohjelmoinnin opetus on vuosikaudet ollut perinteistä luentopainotteista opetusta. Opiskelijat eivät ole motivoituneita passiiviseen ohjelmoinnin opiskeluun. Opiskelijan motivaation puute on johtanut useisiin opiskeluhaasteisiin, jotka voidaan välttää itseohjatulla, aktiivisella ja yhteisöllistävällä ohjelmoinnin opetuksella.

Asiasanat
proseduraalinen ohjelmointi, olio-ohjelmointi, ohjelmoinnin oppiminen, kognitiiviset haasteet, ongelmanratkaisukyky, kognitiivinen kuormitusteoria, motivaation vaikutus oppimiseen, merkityksellinen oppiminen, extreme apprenticeship, itseohjattu opiskelu, pariohjelmointi, skeemat

Ohjaaja
Filosofian lisensiaatti, yliopisto-opettaja, Jouni Lappalainen

Sisällysluettelo

1. Johdanto

Ohjelmoinnin oppiminen on perinteisesti yksi tietojenkäsittelytieteen haastavimmista aihealueista opiskelijoille, mikä johtaa korkeisiin keskeytysprosentteihin kursseilla ja koulutusohjelmissa (Lahtinen, Ala-Mutka & Järvinen, 2005). Keskeytysten suuri määrä on huomattu myös tutkijoiden keskuudessa ja ohjelmoinnin oppimisen vaikeuden juurisyyt ja ratkaisumenetelmät ovat alan tutkijoiden keskeisiä tutkimusaiheita (Tan, Ting & Ling 2009). Noviisien ohjelmoinnin oppimista voidaan tarkastella parhaiten kognitiivisen psykologian avulla (Robins, Rountree & Rountree, 2003).

Yksi suurimmista vaikuttimista ohjelmoinnin oppimisen vaikeuteen on se, että ohjelmointi ei ole yksittäinen taito. Ohjelmointi on kokonaisuus useita erilaisia ja eritasoisia taitoja, joiden tulee olla aktiivisina samaan aikaan. Osittain aiemmista seikoista johtuen ohjelmointi saattaa jopa pelottaa joitain ihmisiä (Moser, 1997). Myös opettamisen nojautuminen liiaksi opeteltavan kielen syntaksiin johtaa lähinnä opiskelijoiden taisteluun kääntäjää vastaan (Fincher, 1999) ja samalla opiskelijat eivät opi ymmärtämään muiden kirjoittamaa lähdekoodia (Wiedenbeck & Ramalingam, 1999).

Tämän kirjallisuuskatsauksen tavoitteena on selvittää, millaisia haasteita ohjelmoinnin oppiminen aiheuttaa noviiseille ja löytää mahdollisia ratkaisuja haasteisiin. Vaikka ongelmat ilmenevätkin joissakin tapauksissa ohjelmointikurssien alhaisina läpäisyprosentteina, on tämän kirjallisuuskatsauksen päämotiivina ennen kaikkea opiskelijan merkityksellinen oppiminen. Tutkimusmenetelmänä käytetään kuvailevaa kirjallisuuskatsausta.

Aihe sisältää keskeisiä psykologisia käsitteitä ohjelmoinnista ja oppimisesta, joita esitetään kappaleessa kaksi. Ohjelmoinnin ja oppimisen psykologiaa tarkastellaan nimenomaan oppijan ja ohjelmoijan näkökulmasta, joita erityisesti kognitiivinen psykologia pyrkii selittämään. Oppijan näkökulma on konstruktiivinen, jolloin skeemateoria ja skeeman hankkiminen ovat keskeisessä osassa. Skeeman hankkiminen tapahtuu ongelmanratkaisun tarjoamassa muotissa ACT-teorian mukaisesti. Tehokasta ja merkityksellistä oppimista, eli kognition optimointia tarkastellaan kognitiivisen kuormitusteorian avulla. Oppiminen ei kuitenkaan tapahdu tyhjiössä, sillä motivaatiolla on suuri vaikutus merkitykselliseen oppimiseen. Ohjelmointia voidaan suoraan verrata ongelmanratkaisuun, joka puolestaan korostaa ongelmanratkaisun roolia tässä kirjallisuustutkimuksessa. Tämän lisäksi on tärkeää ymmärtää ohjelmoinnin kognitiivinen rakenne.

Kolmas kappale käsittelee kirjallisuuskatsauksen keskeisintä aihetta, eli ohjelmoinnin oppimisen haasteita. Tutkimuskysymykselle oleellisia näistä haasteista ovat ongelmanratkaisutaitojen heikkous, puutteellinen käsitys ohjelmassa tapahtuvista asioista sitä käännettäessä ja ajettaessa (Winslow, 1996), sekä motivaation vaikutus ohjelmoinnin oppimiseen (Ambrosio, Almeida, Franco, Martins & Georges, 2012). Neljäs kappale esittelee löydettyjä, mahdollisia ehdotuksia ja keinoja ohjelmoinnin oppimisen haasteiden välttämiseksi. Viidennessä kappaleessa pohditaan kirjallisuuskatsauksen löydöksiä ja ohjelmointiopiskelun haasteellisuutta opiskelijoille, sekä mahdollisia vaihtoehtoja, joilla välttää haasteet. Kirjallisuuskatsauksen kuudennessa ja viimeisessä kappaleessa esitellään pohdinnasta seuraavat johtopäätökset.

2. Ohjelmoinnin oppimisen psykologia

Ohjelmoinnin tarkoituksena on yhdistellä ohjelmointikielen käskyjä siten, että niillä saavutetaan jokin tavoite, useimmiten ratkaistaan jokin ongelma. (Pennington & Grabowski, 1990). Ohjelmoinnin psykologia liittyy pääasiassa kognitiiviseen psykologiaan. Ormerodin (1990) mukaan ohjelmoinnin psykologian keskeisiä kysymyksiä ovat tietoon perustuva esitysmuoto (Knowledge representation), skeemat (Schema), tuottamissäännöt (Production rules), proseduraalinen tieto (Procedural knowledge), deklaratiivinen tieto (Declarative knowledge), tarkkaavaisuusmuisti (Attentional memory), muistiresurssit (Memory resources), semanttinen muisti (Semantic memory), ongelmanratkaisu (Problem solving), taidonhankinta (Skill acquisition) ja mentaalimallit (Mental models).

Ohjelmoinnista voidaan löytää kaksi keskeistä aktiviteettia, jotka ovat kokoaminen (Composition) ja ymmärrys (Comprehension). Kokoaminen tarkoittaa reaalimaailman ongelman kuvaamista ja siirtämistä ohjelmoinnin piiriin, jossa se ratkaistaan. Ymmärrys voidaan nähdä käänteisenä aktiviteettina kokoamiselle. Molemmat aktiviteetit ovat psykologisesti monimutkaisia, koska ne koostuvat alitehtävistä, jotka vaativat useiden tietotoimialueiden (Knowledge domain) hallitsemista ja yhtäaikaisia kognitiivisia prosesseja. (Pennington & Grabowski, 1990.) Shneidermanin ja Mayerin (1975) mukaan kokoamisen ja ymmärryksen lisäksi voidaan tunnistaa kolme muuta keskeistä ohjelmoinnin aktiviteettia, jotka ovat testaus (Debugging), modifiointi (Modification) ja oppiminen (Learning).

2.1 Oppimisprosessi ja kognitiiviset muistikomponentit

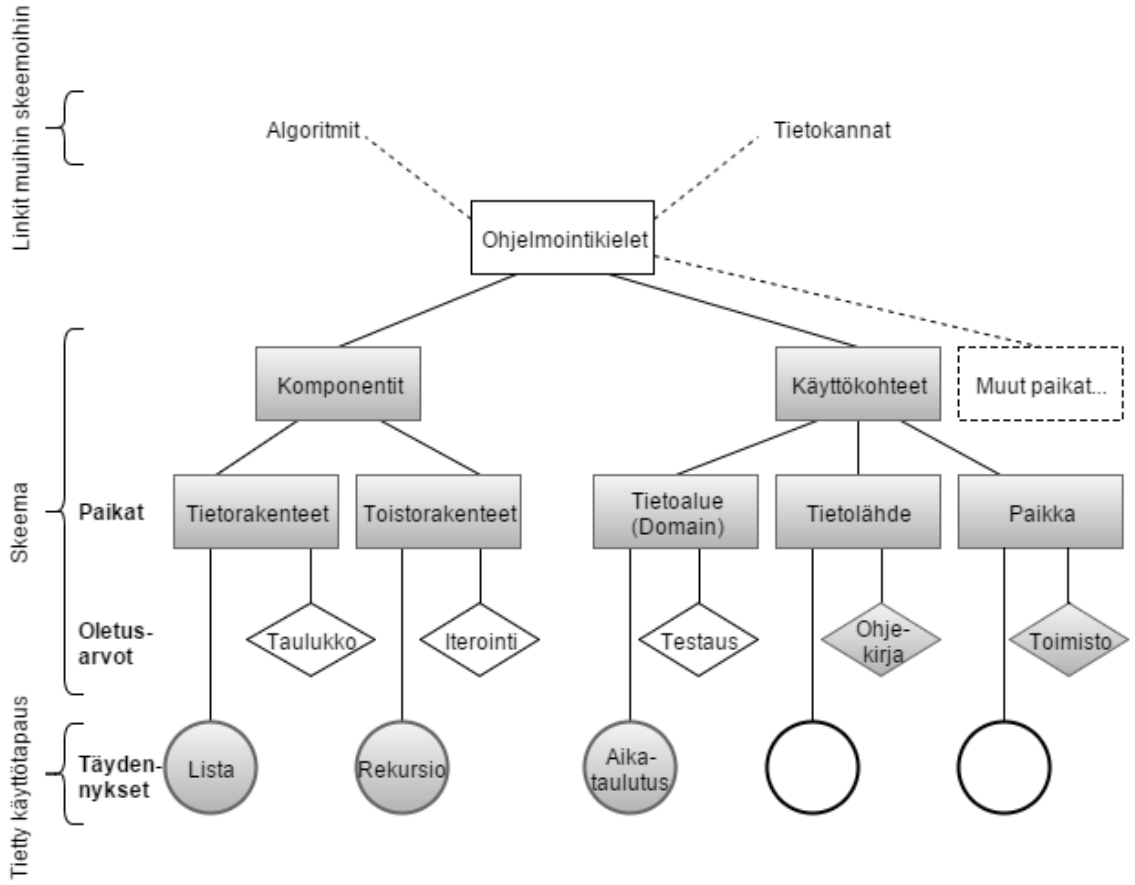
Oppiminen perustuu työmuistiin, pitkäaikaiseen muistiin ja niiden välillä tapahtuvaan keskinäiseen yhteistyöhön (Caspersen & Bennedsen, 2007). Jotta oppiminen olisi merkityksellistä, tulee uuden informaation käydä läpi kolme vaihetta. Ensimmäisessä vaiheessa eli reaktiossa (Reception) oppijan tulee kiinnittää huomio uuteen informaatioon. Miellettävyys-vaiheessa (Availability) oppijalla tulee olla ennalta vaadittavat esitiedot pitkäaikaismuistissa informaation omaksumista varten. Viimeinen vaihe on aktivointivaihe (Activation), jossa oppijan tulee aktiivisesti käyttää pitkäaikaismuistista palautettuja esitietoja, jotta uusi informaatio voidaan liittää siihen. (Mayer, 1981.) Näitä jo hallussa olevia esitietoja sanotaan skeemaksi (Mayer, 1981) ja merkityksellistä oppimista voidaankin kuvata skeeman hankkimiseksi (Schema acquisition) (Caspersen & Bennedsen, 2007).

Miller (1955) erottelee informaation osiin (Bit) ja paloihin (Chunk). Palat koostuvat osista, mutta palassa olevien osien lukumäärällä ei ole vaikutusta. Merkittävää on sen sijaan Millerin tekemä havainto siitä, että työmuistin kapasiteetti on kerrallaan ”7 +/- 2 -palaa”. (Miller, 1955.) Työmuisti uudelleenkoodaa (Recode) tai paloittelee (Chunking) informaation osia ja paloja siten, että osien määrä pienenee ja käytettävissä oleva työmuistin määrä lisääntyy (Caspersen & Bennedsen, 2007). EPAM-teorian (Elementary Perceiver and Memorizer) mukaan muisti koostuisi lyhyt-aikaisesta muistista, työmuistista ja pitkäaikaisesta muistista. Mallissa lyhyt-aikainen muisti toimii havainnosta seuraavan syötteen vastaanottajana ja hyvin pienen muistikapasiteettinsa takia lähinnä välittää tietoa eteenpäin työmuistille. (Feigenbaum, 1967.)

Pitkäaikaismuistia voidaan jaotella semanttiseen ja syntaktiseen tietoon. Semanttinen tieto koostuu yleisistä ohjelmointikonsepteista ja ohjelmointistrategioista. Syntaktinen muisti on sen sijaan esimerkiksi tarkkaa tietoa tietyn ohjelmointikielen syntaksista. Tutkimusten mukaan ihmiselle on helpompaa oppia uutta jo olemassa olevaan semanttiseen tietoon liittyvää syntaktista tietoa, kuin oppia täysin uusi semanttinen tieto. (Shneiderman & Mayer, 1979.) Semanttinen tieto on deklaratiivista tietoa ja sitä voidaan kuvata skeemoilla (Ormerod, 1990).

2.2 Skeemat

Skeema on pitkäaikaismuistissa säilytettävä muistirakenne, jonka avulla isompaa, useammasta palasta koostuvaa kokonaisuutta voidaan käsitellä työmuistissa yhtenä elementtinä (Caspersen & Bennedsen, 2007). Skeeman rakenne muodostuu semanttisen sisällön perusteella. Skeema sisältää abstraktia tietoa, minkä takia se ei liity mihinkään yksittäiseen tapahtumaan ja tämän lisäksi rajoittaa ja ohjaa työmuistin kognitiivista prosessia. Kuvassa 1 on esimerkki ohjelmointikieliä koskevasta skeemasta ja sen rakenteesta. Skeema voi sisältää myös linkkejä toisiin skeemoihin. (Ormerod, 1990.)



Kuva 1. Bartletin skeema Ormerodia (1990) mukaillen

Informaatioprosessoinnin näkökulmasta skeemateoriasta käytetään usein nimeä kognitiivinen skeemateoria (Cognitive Schema Theory, CST). Skeemateorian juuret ovat Kantilaisessa filosofiassa ja Gestaltilaisessa psykologiassa. Moderni konstruktiivis-kognitiivinen skeemateoria tunnistaa kolme tyypillistä skeemaluokkaa, jotka ovat muistiobjekti (Memory object), mentaalimalli ja kognitiivinen kenttä (Cognitive field) (Derry, 1996).

2.3 Kognitiivinen ohjelmarakenne

Ohjelma voidaan jakaa neljään tasoon. Ensimmäinen taso on tarkin ja koostuu yhdestä rivistä lähdekoodia. Toisella tasolla yksittäiset rivit muodostavat yksinkertaisen suunnitelman. Kolmannella tasolla yksinkertaiset suunnitelmat yhdistyvät kompleksisemmaksi suunnitelmaksi, joka ratkaisee jonkin osaongelman. Lopulta viimeisellä abstraktilla tasolla on koko ohjelma, joka ratkaisee jonkin kokonaisen ongelman. Ohjelmoija tarvitsee osaamista kaikilla neljällä tasolla, mikä tekee ohjelmoinnista kognitiivisesti kompleksia. (Rist, 1989).

Ohjelmoinnin näkökulmasta rakenne muodostuu tyypillisimmin käänteisesti ylhäältä alas (Top-down approach) askelittain jalostumalla (Stepwise refinement). Ohjelma rakentuu siten, että alkuperäistä ongelmaa pilkotaan pienemmiksi osatehtäviksi jokaisella askeleella. (Wirth, 1971.) Ristin (1989) mukaan on myös poikkeuksellisesti mahdollista, että ohjelmoinnin ammattilaiset kykenevät lähestymään ohjelmointia alhaalta ylös tilanteissa, joissa se on hyödyllistä.

Tyypillisiä yksinkertaisia suunnitelmia ovat esimerkiksi toistorakenteet (Loop) ja toistolaskuri (Running total). Yksinkertaisia suunnitelmia yhdistelemällä saadaan aikaiseksi komplekseja suunnitelmia (Rist, 1989). Suunnitelmat ovat keskeisin kognitiivinen osa ohjelmien kokoamisessa ja ymmärtämisessä, vaikkakin suunnitelman määritelmä tässä kontekstissa riippuu tutkijasta (Rist, 1995). Abstrakteimmalla tasolla puhutaan kokonaisesta ohjelmasta (Rist, 1989).

2.4 Ongelmanratkaisu

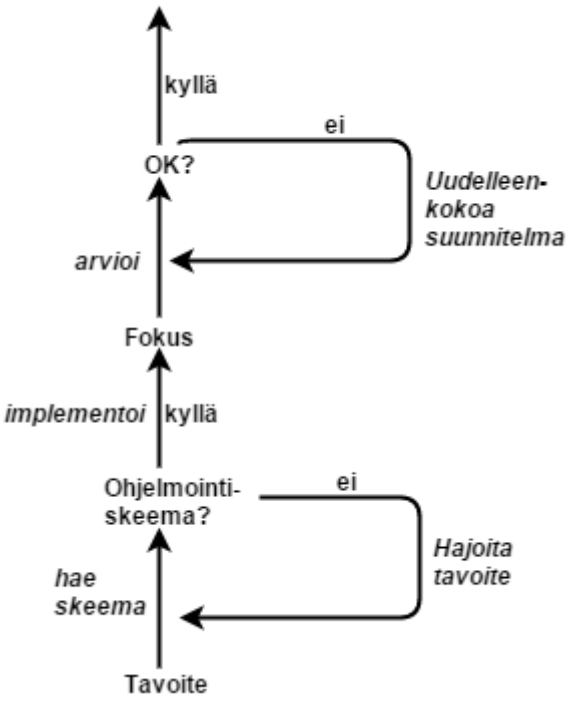
Feigenbaumin ja Simonin EPAM-teoria kehitettiin erityisesti esittämään ongelmanratkaisussa tarvittavia muistikomponentteja. Ongelmanratkaisua voidaan pitää yhtenä merkittävimmistä ohjelmointiin liittyvistä kognitiivisista taidoista. Shneiderman & Mayer muun muassa käyttävät ongelmanratkaisua synonyymin tavoin kirjoittaessaan ohjelmoinnin kokoamisaktiviteetista. (Shneiderman & Mayer, 1979.) Sekä Newell (1979), että Anderson (1982) näkevät ongelmanratkaisun kognition ”perustilana” (Basic mode).

Newellin ja Simonin mallissa ongelmanratkaisu tapahtuu ongelma-avaruudessa (Problem space). Ongelma-avaruus sisältää nykyisen tilan, välitilat sekä tavoitetilan. Operaattori (Operator) tarkoittaa toimintaa, joka johtaa tilan vaihtumiseen. Koska ongelma-avaruudet voivat olla suuria ja työmuistin määrä rajattua, tarvitsee ongelmanratkoja tapoja navigoidakseen ongelma-avaruudessa. Ongelma-avaruus-mallissa navigointi tapahtuu hakukontrollitiedon (Search control knowledge) avulla. Hakukontrollitietometodeista keskeisimmät ovat heuristinen haku (Heuristic search) ja keino-päämäärä-analyysi (Means-end analysis), jossa verrataan nykytilaa haluttuun tilaan ja kyetään valitsemaan oikea operaattori. (Newell, 1979.)

Ohjelmoinnin kontekstissa ongelmanratkaisu tarkoittaa oikean maailman ongelman siirtämistä ohjelmoinnin ongelman ratkaisun piiriin. Winslowin (1996) mallin mukaan ongelmanratkaisu ohjelmoinnissa on yksinkertainen: (1) Ymmärrä ongelma, (2a) Selvitä, miten ongelma ratkaistaan jossakin muodossa, (2b) Selvitä, miten ongelma ratkaistaan tietokoneelle ymmärrettävässä muodossa, (3) Käännä ratkaisu ohjelmointikielelle ja (4) Testaa ja jäljitä virheet.

Skeemoihin perustuvalla tavoitteen hajottamis- ja uudelleenkokoamismallilla (Goal decomposition and recomposition model) voidaan tarkastella tarkemmin reaalimaailman

ongelman siirtämistä ohjelmoinnin piiriin. Mallissa haetaan ongelmaan sellaista ratkaisua, jolla päästään tavoitteeseen. Jos tällainen löytyy, se implementoidaan. Jos ongelman ratkaisuun ei löydy soveltuvaa skeemaa, hajotetaan tavoitetta pienemmäksi, kunnes siihen soveltuva skeematieto löydetään. Tällaisessa tapauksessa ohjelmasuunnitelmasta tulee ylhäältä alas-mallin mukainen. Jos ylhäältä alas-mallilla ei löydetä ongelmanratkaisuun soveltuvaa skeemaa, käytetään fokaalista laajentumista (Focal expansion). Fokaalisessa laajentumisessa lähdetään rakentamaan ratkaisua pienemmistä ohjelmoijan jo tuntemista osista. Fokaalinen laajentuminen johtaa alhaalta ylös-mallin mukaiseen ohjelmasuunnitelmaan. (Rist 1989.) Kuvassa 2 esitetään tavoitteen hajottamis- ja uudelleenkokoamismallin yleistä toimintaa.



Kuva 2. Tavoitteen hajottamis- ja uudelleenkokoamismalli Ristiä (1989) mukaillen

2.5 Kognitiivisen kyvyn oppiminen

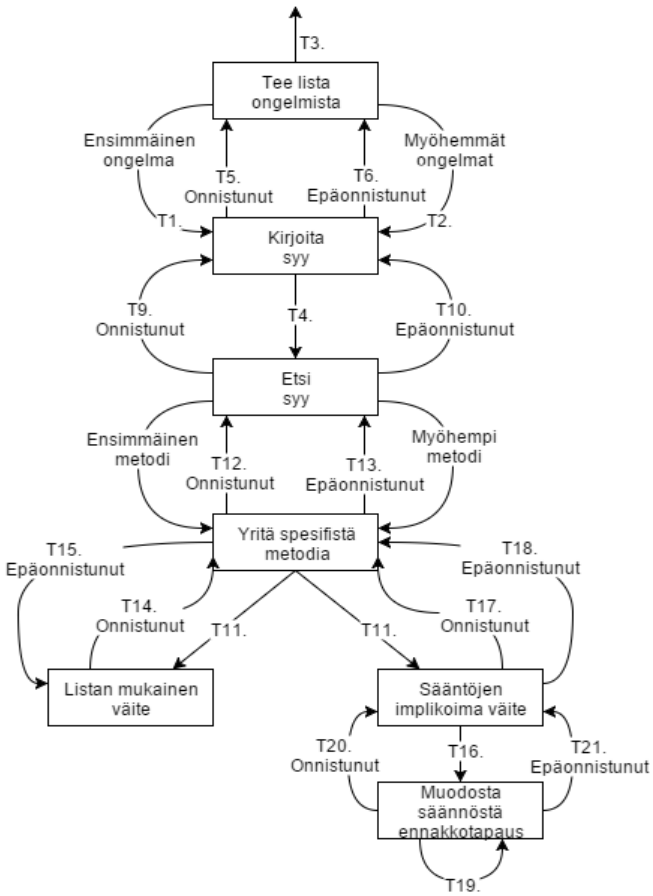
Kognitiivisen kyvyn oppimiseen on ehdotettu useita erilaisia malleja. Näillä kaikilla teorioilla on kuitenkin paljon yhteisiä piirteitä. Erityisesti niin sanotuille perinteisille malleille on tyypillistä olettaa, että oppiminen perustuu jo omattuun yleiseen tietoon, joka kokemusten kautta kykenee erikoistumaan. (Taatgen, 2008.) Perinteisillä malleilla tässä tapauksessa viitataan niin sanotun ”Pittsburghin koulukunnan” kognitiivisen psykologian tutkimukseen ja erityisesti nykyäänkin hallitseviin ACT-teorioihin (Adaptive Control of Thought) (Derry, 1996).

ACT-teoriat soveltavat Fittsin kehittämää kolmivaiheista taidon oppimismallia (Anderson, 1982). Kognitiivinen vaihe (Cognitive stage) on vaiheista ensimmäinen ja voi olla hyvinkin lyhyt. Vaihetta voidaan verrata lähes täysin ulkolukuopetteluun (Rote learning) ja vaiheelle tyypillistä on taidon toteuttaminen verbaalisen itselle selittämisen avulla (Verbal mediation). Assosiatiivisessa vaiheessa (Associative stage) itselle selittäminen ja virheiden määrä vähenee. Autonominen vaihe (Autonomous stage) on kolmivaiheisen mallin viimeinen vaihe, joka on niin sanottu jatkuvan kehityksen vaihe.

(Fitts, 1964.) ACT-teorioissa kognitiivinen vaihe on deklaratiivinen vaihe (Declarative stage), assosiatiivinen vaihe on siirtymätila, eli tiedon kokoaminen (Knowledge compilation) ja autonominen vaihe on proseduraalinen vaihe (Procedural stage) (Anderson, 1982).

Toinen oleellinen osa ACT-teorioita on tuotantojärjestelmä (Production system), joka kuvaa taidosta omattua tietoa jos-niin-lausekkeilla (If-then statements) (Derry, 1996). Kuvassa 3 esitetään esimerkki tuotantojärjestelmän kontrollivirtauksesta (Flow of control) ja kuvassa 4 kuvataan tuotantojärjestelmän toimintaa jos-niin-lausekkeilla. Tuotantojärjestelmä esittää proseduraalista tietoa tuotantona (Production) ja deklaratiivista tietoa propositioverkostona (Proposition network), joka on semanttisesti järjestäytynyttä deklaratiivista tietoa (Anderson, 1982). Tuotantojärjestelmästä voidaan käyttää myös nimitystä tuotantosäännöt (Production rules). Ormerodin (1990) mukaan on tyypillistä kuvata proseduraalista tietoa tuotantosäännöillä ja deklaratiivista tietoa propositioverkostojen sijaan skeemoilla.

ACT-teorioiden pohjalla on Newellin ja Simonin ongelma-avaruus-malli (Taatgen, 2008). ACT-tuotantojärjestelmä sisältää joukon tuotantoja, jotka toteutuvat perustuen deklaratiivisiin faktoihin. ACT-teoriat perustuvat siihen, että tuotantojärjestelmä on hierarkkinen, tavoitepohjainen ja systemaattista ongelmanratkaisua tukeva malli, jossa uuden taidon oppimisen alussa sovelletaan yleisempää ongelmanratkaisuun soveltuvaa tuotantojärjestelmää. Taidon kehittyessä tuotantojärjestelmä virittyy spesifisemmän taidon tuotantojärjestelmäksi. (Anderson, 1982.)



Kuva 3. Tuotantojärjestelmän toiminnan kontrollivirtaus Andersonia (1982) mukaillen

T1.	JOS	tavoite on ratkaista useita ongelmia
	NIIN	aseta alitavoite, jossa ratkaistaan ensimmäinen ongelma
T2.	JOS	tavoite on ratkaista useita ongelmia ja yksi ongelma on juuri ratkaistu
	NIIN	aseta seuraava ongelma alitavoitteeksi
T3.	JOS	tavoite on ratkaista useita ongelmia ja kaikki ongelmat on ratkaistu
	NIIN	POISTA tavoite onnistuneena
T4.	JOS	tavoite on kirjoittaa relaation nimi argumentille
	NIIN	aseta alitavoite, jossa etsitään relaatio argumentille
T5.	JOS	tavoite on kirjoittaa relaation nimi argumentille ja nimi on löydetty
	NIIN	kirjoita nimi ja POISTA tavoite onnistuneena
T6.	JOS	tavoite on kirjoittaa relaation nimi argumentille ja nimeä ei löydetty
	NIIN	POISTA tavoite epäonnistuneena
T7.	JOS	tavoite on löytää relaatio ja käytössä on lista metodeista relaation löytämiseksi
	NIIN	aseta alitavoite, jolla yritetään listan ensimmäistä metodia
T8.	JOS	tavoite on löytää relaatio ja käytössä on lista metodeista relaation löytämiseksi ja metodi on epäonnistunut
	NIIN	aseta alitavoite, jolla yritetään listan seuraavaa metodia
T9.	JOS	tavoite on löytää relaatio ja käytössä on lista metodeista relaation löytämiseksi ja metodi on onnistunut
	NIIN	POISTA tavoite onnistuneena
T10.	JOS	tavoite on löytää relaatio ja käytössä on lista metodeista relaation löytämiseksi ja kaikki metodit ovat epäonnistuneet
	NIIN	POISTA tavoite epäonnistuunena
T11.	JOS	tavoite on yrittää metodia, joka vaatii yhteyden muodostamista
	NIIN	aseta alitavoite "muodosta yhteys"
T12.	JOS	tavoite on yrittää metodia ja alitavoite on onnistunut
	NIIN	POISTA tavoite onnistuneena
T13.	JOS	tavoite on yrittää metodia ja alitavoite on epäonnistunut
	NIIN	POISTA tavoite epäonnistuneena
T14.	JOS	tavoite on vahvistaa toteamuksen kuuluvan listaan ja lista sisältää toteamuksen
	NIIN	POISTA tavoite onnistuneena
T15.	JOS	tavoite on vahvistaa toteamuksen kuuluvan listaan ja lista ei sisällä toteamusta
	NIIN	POISTA tavoite epäonnistuneena
T16.	JOS	tavoite on vahvistaa, että rivi on säännön edellyttämässä joukossa ja joukko sisältää säännön muotoa "ennakkotapaus, jos johtuu x:stä" ja johtuu x:stä vastaa riviä
	NIIN	aseta alitavoite, jossa selvitetään vastaako ennakkotapaus vahvisttetuja toteamuksia ja merkkaa sääntö yritetyksi
T17.	OS	tavoite on vahvistaa, että rivi on säännön edellyttämässä joukossa ja joukko sisältää säännön muotoa "ennakkotapaus, jos johtuu x:stä" ja johtuu x:stä vastaa riviä ja ennakkotapaus on vahvistettu
	NIIN	POISTA tavoite onnistuneena
T18.	JOS	tavoite on vahvistaa, että rivi on säännön edellyttämässä joukossa ja ei ole olemassa yrittämätöntä sääntöä joukossa, joka vastaa riviä
	NIIN	POISTA tavoite epäonnistuneena
T19.	JOS	tavoite on selvittää, jos ennakkotapaus vastaa vahvistettuja toteamuksia ja on olemassa vahvistamattomia ennakkotapausehtoja ja ehto vastaa vahvistettua väitettä
	NIIN	merkkaa ehto vahvistetuksi
T20.	JOS	tavoite on selvittää, jos ennakkotapaus vastaa vahvistettuja toteamuksia ja vahvistamattomia ennakkotapausehtoja ei ole
	NIIN	POISTA tavoite onnistuneena
T21.	JOS	tavoite on selvittää, jos ennakkotapaus vastaa vahvistettuja toteamuksia ja on olemassa vahvistamaton ennakkotapausehto ja se ei vastaa vahvistettua toteamusta
	NIIN	POISTA tavoite epäonnistuneena

Kuva 4. Kuvan 3 tuotantojärjestelmän jos-niin-lausekkeet Andersonia mukaillen (Anderson, 1982)

ACT-R (Adaptive Control of Thought – Rational) on uusin ja tarkin ACT-malli, jossa teoriaan lisättiin käsite tuotannon kokoamisesta (Production compilation). Tuotannon kokoaminen on mekanismi, jossa kahdesta peräkkäin käytetystä tuotantojärjestelmästä kootaan uusi tuotantojärjestelmä (Taatgen, 2008). Derry (1996) luokittelee ACT-teoriat rajoittuneiksi konstruktiivisiksi malleiksi.

2.6 Kognitiivinen kuormitusteoria

Kognitiivisen kuormitusteorian (Cognitive Load Theory, CLT) mukaan ihmisellä on rajoittamaton pitkäaikaismuistin kapasiteetti. Teoria käsittelee työmuistin kognitiivista kuormaa ongelmanratkaisun ja oppimisen aikana sekä kuorman optimointia. (Caspersen & Bennedsen, 2007.) Tavanomainen ongelmanratkaisu tietotaitoalueilla, joilta oppija ei ole omaksunut skeemaa, on useimmiten keino-päämäärä-analyysin mukaista. Keino-päämäärä-analyysi on kognitiiviselta kuormitukseltaan hyvin raskasta, mikä johtaa oppimisen heikentymiseen. (Sweller, 1988.)

Skeema sen sijaan voi sisältää valtavankin määrän tietoa ja tämän lisäksi työmuisti kykenee käsittelemään sitä yksittäisenä elementtinä. Skeeman käyttäminen voi myös usein johtaa automaatioon, joka vähentää työmuistin kognitiivista kuormitusta entisestään (Kirschner, 2002). Oppiminen ja ongelmanratkaisu ovatkin Swellerin (1988) mukaan jossakin määrin ristiriidassa keskenään, sillä tavanomainen ongelmanratkaisu johtaa korkeaan kuormitukseen ja skeeman hankkimisen heikkenemiseen.

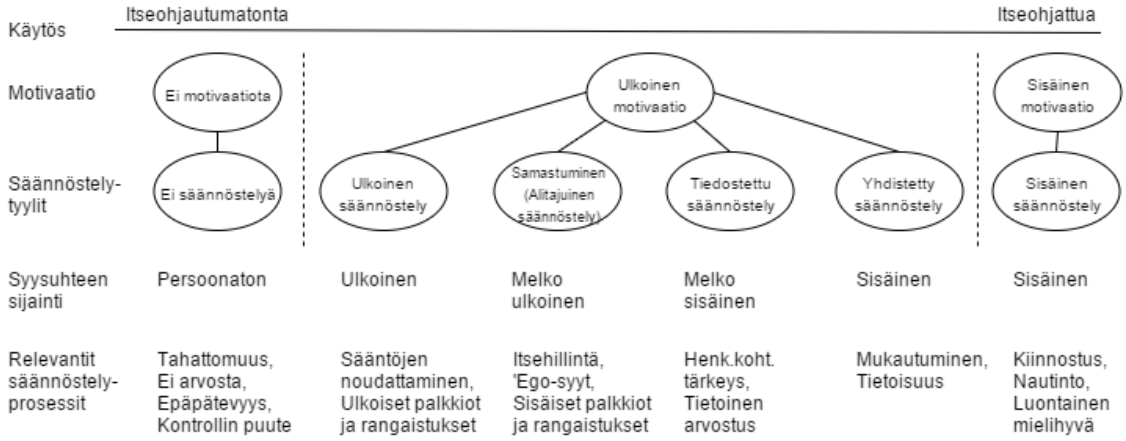
Kognitiivista kuormitusta tulee olla teorian mukaan oikea määrä ja sen tulee olla myös oikeanlaista, jotta oppiminen olisi merkityksellistä. Kognitiivinen kuormitus voidaan jakaa kolmeen erilaiseen kuormitukseen. (Caspersen & Bennedsen, 2007.) Asiaankuulumaton (Extraneous) kognitiivinen kuormitus johtuu huonon ohjeistuksen ja opetuksen tulkitsemiseen kuluvasta kapasiteetista. Luontaiseen (Intrisic) kognitiiviseen kuormitukseen ei voida vaikuttaa ulkopuolelta, sillä se liittyy opittavan asian ja oppijan aikaisemman osaamisen tuomaan kuormitukseen. (Kirschner, 2002.) Asiaankuuluva (Germane) kognitiivinen kuorma on oleellista nimenomaan oppimisen kannalta, sillä sen avulla työmuisti ratkaisee ongelmia, luo uusia skeemoja ja yhdistää opittavaa tietoa jo olemassa oleviin skeemoihin. Kognitiivinen kuormitus kokonaisuudessaan koostuu asiaankuulumattomasta, asiaankuuluvasta ja luontaisesta kuormituksesta. Merkityksellinen oppiminen perustuukin kuormituksen maksimoinnin sijaan enemmän kognitiivisen kuormituksen balanssin optimointiin. Tämä tarkoittaa sitä, että merkityksellisen oppimisen toteutumiseksi minimoidaan oppimiseen liittyvä asiaankuulumaton kuormitus ja maksimoidaan asiaankuuluva kuormitus. (Caspersen & Bennedsen, 2007.)

2.7 Motivaatio

Motivaatioteorian tarkoituksena on pyrkiä selittämään, miksi ihminen toimii, kuten toimii tai miksi ihminen ajattelee, kuten ajattelee. Esimerkiksi opiskelussa motivaatiota tutkiva voisi miettiä, miksi ylipäätään opiskellaan tai miksi opiskellaan juuri jotain tiettyä aihetta, kauanko opiskelun aloittamiseen menee aikaa, kauanko se kestää, miten kovasti todellisuudessa opiskellaan ja mitä ihminen miettii opiskellessaan (Graham & Weiner, 1996). Motivaatioteorioita on olemassa useita. Opiskelunäkökulman kannalta merkittäviä teorioita on neljä. Näitä ovat odotusarvoteoria (Expectancy-Value Theory), päämäärän saavutusteoria (Achievement Goal Theory), itseohjautuvuusteoria (Self-Determination Theory, SDT) ja sosiaaliskognitiivinen teoria (Social Cognitive Theory). (Anderman & Gray, 2001.)

Itseohjautuvuusteoria käsittää karkeasti kahdenlaista motivaatiota. Sisäinen motivaatio (Intrinsic motivation) pohjautuu henkilön aitoon mielenkiintoon tai nautintoon tehdä jotakin. Ulkoista motivaatiota (Extrinsic motivation) ovat esimerkiksi ulkoiset palkkiot ja rangaistukset, sisäiset palkkiot ja rangaistukset, itsehillintä, egoon liittyvät syyt tai toiminnan tiedostettu henkilökohtainen tärkeys. Itseohjautuvuuden tasoilla täydellinen motivaation puute tarkoittaa toimintaa, jota ei suoriteta tai se suoritetaan ilman

määrätietoisuutta. Motivaation puute voi johtua esimerkiksi henkilön vähäisestä arvostuksesta aktiviteettia kohtaan, vähäisestä kompetenssin tunteesta tai oletuksesta, ettei toiminta johda haluttuun lopputulokseen. Puhdas ulkoinen motivaatio eli ulkoisesti säännöstelty motivaatio on yksilön näkökulmasta kontrolloivaa tai vieraannuttavaa ja perustuu täysin ulkoisiin seurauksiin. Samastumisen säännöstelyssä (Introjected regulation) toiminta suoritetaan ahdistuksen tai syyllisyydentunteen välttämiseksi tai ylpeyden saavuttamiseksi ja tätä alitajuista itsetunnon säännöstelyä voidaan pitää melko ulkoisena. Tiedostettu säännöstely (Regulation through identification) on melko sisäistä motivaatiota ja sisältää tietoista arvostusta, jonka takia toiminta on tärkeää. Eniten autonominen ulkoisen motivaation taso on yhdistetty säännöstely (Integrated regulation), jonka aiheuttamaa toimintaa voidaan suurilta osin verrata sisäisestä motivaatiosta johdettuun toimintaan. Erona sisäiseen motivaatioon, yhdistetyssä säännöstelyssä on kuitenkin muita syitä toiminnalle kuin luontainen ilo. (Deci, Koestner & Ryan, 1999.) Kuvassa 5 havainnollistetaan itseohjautuvuuden ja motivaation vaikutussuhdetta.



Kuva 5. Itseohjautuvuuden jatkumo Ryania & Deciä (2000) mukaillen

Itseohjautuvuusteorian mukaan ihmisen perustavanlaatuisia tarpeita ovat omaehtoisuus (Autonomy), kyvykkyys (Competence) ja yhteisöllisyys (Relatedness). (Ryan & Deci, 2000.) Douglassin & Morrisin (2014) mukaan opiskelija tulee sisäisesti motivoituneeksi, kun oppimistilanne antaa opiskelijalle tunteen omaehtoisuudesta, kyvykkyydestä, yhteisöllisyydestä tai tarkoituksellisuudesta. Deci et al. (1999) tekivät sisäisestä ja ulkoisesta motivaatiosta metatutkimuksen 182:sta aiemmin julkaistusta motivaatiotutkimuksesta. Metatutkimuksessa selvisi, että eritysesti konkreettisella ulkoisilla palkkioilla on merkittävä negatiivinen vaikutus sisäiseen motivaatioon. (Deci et al., 1999.)

3. Ohjelmoinnin oppimisen haasteet

Useat tutkimukset ovat todistaneet, että aloittelevat ohjelmoijat osaavat opeteltavan ohjelmointikielen syntaksin ja yksittäisten lausekkeiden semantiikan, mutta eivät osaa yhdistellä lausekkeita toimiviksi ohjelmiksi. Tapauksissa joissa ongelmanratkaisu onnistuu paperilla, ongelmaksi muodostuu ratkaisun muuttaminen ohjelmaksi. (Winslow, 1996.) Ohjelmointi vaatii osaajaltaan useita erilaisia taitoja ja tämä johtaa aloittelijoiden kohdalla siihen, että oppimisen haasteitakin on olemassa useita. Osaongelmien ja ongelmien ratkaisuun vaaditaan yksittäisten rivien yhdistelyä, mikä viittaa vaikeuksiin erityisesti ongelman ohjelmointikontekstiin siirtämisessä.

Gomes & Mendes (2007) listaavat tyypillisimpiä opiskelijoiden oppimisvaikeuksiin vaikuttavia syitä ohjelmoinnissa, kuten opiskelijan väärä opiskelumetodologia, opiskelijoiden liian vähäinen työmäärä, ongelmanratkaisutaitojen puute, matemaattisen ja loogisen osaamisen puute, ohjelmointitietojen heikkous, ohjelmoinnin vaativa abstraktiotaso, ohjelmointikielien kompleksi syntaksi ja opiskelijan motivaation puute. Gomesin ja Mendesin (2007) ohella myös Winslow (1996) korostaa erityisesti opiskelijoiden ongelmanratkaisutaitojen puutteellisuutta. Oppimisen tulisi myös aina olla jossain määrin itseohjattua, jotta oppijalla olisi riittävä motivaatiotaso (Jenkins, 2001).

3.1 Haasteet opiskelijoiden ja opettajien perspektiivistä

Vuonna 2005 toteutettiin kansainvälinen tutkimus kuudessa eri yliopistossa ja ammattikorkeakoulussa, jossa tutkittiin 559 opiskelijan ja 34 opettajan käsityksiä ohjelmoinnin oppimisen haasteista. Vastaajat koostuivat sekä pelkästään CS1-kurssin suorittaneista, että myös CS2-kurssin suorittaneista ja ohjelmointikielinä oli käytetty Javaa tai C++:aa. Suurimpia vaikeuksia tuottivat virheiden löytäminen ohjelmakoodista, funktionaalisuuden jakaminen pienempiin proseduureihin ja ohjelman suunnittelu niin, että se ratkaisee jonkin tietyn ongelman. Vaikeimpia aihealueita olivat puolestaan osoittimet ja viittaukset, virheiden käsittely, rekursio, kielen mukana olevien kirjastojen käyttö ja abstraktit tietotyypit. Tämän lisäksi C++ koettiin vaikeammaksi kieleksi kuin Java. Tutkimuksen perusteella voitiin todeta, että opiskelijoille vaikeimpia aihealueita ovat sellaiset aihealueet, jotka vaativat koko ohjelman toiminnan ymmärtämistä. Tämän lisäksi opiskelijat yliarvioivat taitojaan jokaisessa kategoriassa, sillä opettajat arvioivat kautta linjan kaikki aihealueet ja haasteet ongelmallisemmiksi kuin opiskelijat. (Lahtinen et al., 2005.)

Milne ja Rowe toteuttivat vuonna 2002 Dundeessa kyselytutkimuksen 66:lle ensimmäisen ja toisen vuoden C- ja C++-opiskelijoille. Tämän lisäksi erillinen kysely toteutettiin elektronisesti opettajille ja luennoitsijoille ympäri Iso-Britanniaa. Kyselyssä opettajat näkivät haastavimmiksi aihealueiksi osoittimet, virtuaaliset funktiot, dynaamisen muistin osoittamisen (malloc), muunneltavuuden (Polymorphism) ja rekursion. Opiskelijoiden mukaan haastavimpia aihealueita olivat kopiointimuodostimet (Copy constructor), operaattorin ylikuormitus (Operator overloading), template-toiminnallisuus (C++:ssa), dynaaminen muistin osoittaminen (malloc) ja osoittimet. Tutkimuksen mukaan vaikeimmat aihealueet liittyvät ohjelman muistiin ja sen

toimintaan, eikä opiskelijoilla ole mentaalimallia siitä, miten ohjelma toimii. (Milne & Rowe, 2002.)

Kolmas esiteltävä kyselytutkimus toteutettiin Malesiassa vuonna 2009, siihen osallistui 180 CS1-kurssin opiskelijaa ja kysely pohjautui kahdessa aiemmin esitellyssä tutkimuksessa käytettyihin kyselyihin. Kyselyssä opiskelijoiden mukaan vaikeimpia aihealueita ohjelmoinnissa olivat monimuotoisuus, periytyminen, kapselointi, abstraktit tietotyypit, osoittimet ja viittaukset. Oppimisvaikeuksia oli opiskelijoiden mukaan erityisesti ohjelman suunnittelussa siten, että se ratkaisee jonkin tietyn ongelman, funktionaalisuuden jakamisessa proseduureihin ja ohjelmointikielen syntaksin oppimisessa. Tutkimuksen tulosten mukaan merkittävin ongelma opiskelijoiden oppimisessa on puutteellinen ymmärrys siitä, mitä ohjelmassa tapahtuu suorituksen aikana. (Tan et al., 2009.)

Kokonaisuutena tutkimuksista ilmenee, että suurimmat vaikeudet liittyvät ongelmanratkaisuun ohjelmointikontekstissa ja opiskelijoiden puutteelliseen ymmärrykseen tietokoneen toiminnasta. Tämän lisäksi merkittävä osa haasteista liittyi ohjelmointikielien abstrakteihin toimintoihin.

3.2 Ohjelmointikielen ymmärrys

Ohjelmointikielen ymmärrys on yksi merkittävimmistä ohjelmoinnin oppimisen haasteista. Adelson (1981) käyttää nimitystä konseptuaalinen kategoria, johon säilötään elementtejä. Nämä kategoriat voivat olla syntaktisia tai semanttisia. Eksperttien konseptuaaliset kategoriat ovat semanttisia ja vahvasti järjesteltyjä, mikä viittaa skeemoihin. Noviisien tapauksessa rakenne on heikosti järjesteltyä ja konseptuaaliset kategoriat syntaktisia. Tyypillinen tieto tällaisessa konseptuaalisessa kategoriassa voi olla esimerkiksi yhden rivin komento. (Adelson, 1981.) Winslowin (1996) mukaan noviiseilta puuttuu ohjelmointiin tarpeellisia mentaalimalleja, osaavat vain pintatietoutta ohjelmoinnista ja omaavat heikkoa tietoutta ohjelmoinnista, millä tarkoitetaan tietoa, joka tiedetään, mutta jota ei osata hyödyntää oikeassa tilanteessa. Robins et al. (2003) näkevät myös, että noviisien väärinkäsitykset ovat yksi ongelma-alue. Sekaannuksia voi tuottaa esimerkiksi luonnollisessa kielessä esiintyvän sanan tarkoitus ja ohjelmointikielessä saman nimisen komennon toiminnan sekoittaminen keskenään (Robins et al., 2003).

Ymmärryksen tutkimus on siirtynyt 2000-luvulla enemmän tutkimuksiin, joissa verrataan noviisien osaamista ohjelmointikielen lukemisessa, kirjoittamisessa ja jäljityksessä. McCracken et al. (2001) tutkivat CS1-opiskelijoita ja huomasivat, että suuri osa opiskelijoista ei osannut ohjelmoida kurssin päätyttyä. Suurimmaksi ongelmaksi muodostui tehtävänannossa esitetyn ongelman abstrahointi ja siirtäminen ohjelmoinnin kontekstiin. Lister et al. (2004) tunnustivat McCracken et al:in löydökset, mutta olivat huolestuneita erityisesti siitä, että ohjelman kirjoittamisen lisäksi suuri osa opiskelijoista ei osannut myöskään lukea tai jäljittää lähdekoodia. Lopez, Whalley, Robbins & Lister (2008) löysivät vahvan yhteyden ohjelmointikielen jäljityskyvyissä ja kirjoituskyvyissä. Yhteys oli erityisen vahva, kun tarkasteltiin toistorakenteita. Tämän lisäksi löydettiin mahdollinen yhteys lähdekoodin lukemistaidon ja lähdekoodin kirjoitustaidon välille. Venables, Tan & Lister (2009) tekivät vastaavan löydöksen omasta tutkimuksestaan, jossa löydettiin vahva yhteys ohjelmointikielen lukemistaidon, jäljitystaidon ja kirjoitustaidon välille.

3.3 Ongelmanratkaisutaidot

Toinen merkittävimmistä ohjelmoinnin oppimisen haasteita aiheuttavista ongelmista liittyy ongelmanratkaisuun. Luova, looginen ja deduktiivinen päättely liittyvät suurilta osin matemaattisiin kykyihin (Pacheco, Gomes, Henriques, de Almeida & Mendes, 2008). Byrne & Lyons (2001) löysivät yhteyden opiskelijan ohjelmointimenestyksestä ja opiskelijan aiemmasta menestyksestä matematiikassa ja muissa tiedeoppiaineissa. Tutkimuksen mukaan yhteyden syy on ohjelmoinnin ja matemaattisten aineiden vaatimien kognitiivisten kykyjen samankaltaisuus. Sen sijaan menestys muissa luonnollisissa kielissä ei vaikuttanut merkittävästi ohjelmointikielien oppimisessa. (Byrne & Lyons, 2001.)

Pacheco et al. (2008) esittivät kolme tutkimusta matemaattisten taitojen, ongelmanratkaisun ja ohjelmoinnin liittymisestä toisiinsa, joissa kaikissa löydettiin positiivinen korrelaatio heikkojen matemaattisten taitojen ja heikon ongelmanratkaisukyvyn välille. Tämän lisäksi yhdessä tutkimuksista löydettiin positiivinen korrelaatio myös heikkojen matemaattisten taitojen ja heikon ohjelmointimenestyksen välille (Pacheco et al., 2008). Bergin & Reilly (2005) löysivät vahvan positiivisen korrelaation matemaattisten kykyjen ja ohjelmointikykyjen välille. Cantwell-Wilson & Shrock (2001) löysivät myös positiivisen korrelaation matemaattisten taitojen ja ohjelmointimenestyksen välille. Pillay & Jugoo (2005) löysivät positiivisen korrelaation ongelmanratkaisutaitojen ja ohjelmointitaitojen välille.

Whiten & Sivitanidesin (2005) kirjallisuustutkimuksessa esitellään tutkimuksia, joissa on verrattu tutkittavien ohjelmointimenestystä ja luokiteltu tutkittavia Piagetin kognitiivisen kehitysteorian perusteella. Tutkimusten perusteella proseduraalisten ja objekti-orientoituneiden kielten hallintaan vaaditaan formaali kognitiokehityksen taso. Valtaosassa tutkimuksista huomattiin kuitenkin, ettei suurin osa aikuisista ja yliopisto-opiskelijoista koskaan kehity Piagetin asteikolla formaalin kognition tasolle. Kirjallisuustutkimuksen perusteella suuri osa tutkimuksista löysi myös yhteyden heikon kognition kehityksen ja heikon ohjelmointimenestyksen välille. (White & Sivitanides, 2005.)

Winslowin (1996) mukaan erityisesti noviisien ongelmanratkaisutaidot ovat heikot ja suurin syy ohjelmoinnin oppimisen vaikeudelle. Koska noviiseilla ei ole riittävää osaamista ohjelmoinnista, noviisit päätyvät käyttämään yleisiä ongelmanratkaisustrategioita, kuten keino-päämäärä-analyysia, ongelmissa, joihin se soveltuu huonosti (Winslow, 1996).

3.4 Motivaation vaikutus ohjelmoinnin oppimiseen

Motivaation vaikutusta ohjelmointiin on tutkittu varsin rajallisesti. Jenkinsin (2001) mukaan motivaation tieteellinen mittaaminen on vaikeata. Itseohjautuvuusteorian motivaatiomääritelmä on yleispätevä, mutta Jenkins (2001) tunnistaa ohjelmointikurssin kontekstissa neljää erityyppistä motivaatiota. Ulkoisessa motivaatiossa päämotiivi on kurssin ja koulutusohjelman jälkeinen ura, sisäisessä motivaatiossa päämotiivi on vahva henkilökohtainen kiinnostus tietojenkäsittelytieteisiin tai ohjelmointiin. Sosiaalisessa motivaatiossa päämotiivi on miellyttää kolmatta osapuolta ja saavuttamisen motivaatiossa tavoitteena on menestyä hyvin henkilökohtaisen mielihyvän saavuttamiseksi. Tutkimuksessa tehtiin hälyttävä huomio, sillä 48 prosentilla opiskelijoista ensisijainen motivoiva tekijä oli ohjelmointikurssin pakollisuus. Vain 20 prosenttia opiskelijoista ilmoitti olevansa aidosti kiinnostunut aiheesta. (Jenkins, 2001.)

Gomes, Santos & Mendes (2012) tutkivat opiskelijoiden motivaation tasoa suhteessa ohjelmointimenestykseen ja löysivät korrelaation motivaation ja loppuarvosanan välille. Merkittävämpi korrelaatio löytyi kuitenkin ohjelmointimenestyksestä ja opiskelijan omasta käsityksestä ohjelmointitaidoistaan (Gomes et al., 2012). Bergin & Reilly (2005) tutkivat motivaation ja mukavuusasteen vaikutusta ohjelmointimenestykseen ja huomasivat, että sisäisesti motivoituneet opiskelijat menestyivät ohjelmoinnissa ulkoisesti motivoituneita opiskelijoita paremmin. Ambrosio et al. (2012) pitävät motivaatiota yhtenä tärkeimmistä oppimiseen vaikuttavista tekijöistä ja näkevät, että perinteiset ohjelmoinnin opetusmetodit eivät herätä erityisesti innostuneisuutta opiskelijoissa. Rountree, Rountree & Robins (2002) näkevät, että opiskelijan positiivinen asennoituminen on suurin ohjelmoinnin oppimiseen vaikuttava tekijä.

Guiffrida, Lynch, Wall & Abel (2013) näkevät, että motivaation vaikutus opiskelumenestykseen on mahdollisesti hyvin tärkeä. Tutkimuksessa selvisi myös, että sisäisesti motivoituneet opiskelijat olivat keskiarvoltaan keskitasoa parempia opiskelijoita. Nämä sisäisesti motivoituneet oppijat osoittivat sisäistä tarvetta omaehtoisuudelle ja kyvykkyydelle. (Guiffrida et al., 2013.) Vastaavia tuloksia on saatu lapsilla tehdyissä tutkimuksissa. Cordova & Lepper (1996) pyrkivät motivoimaan oppilaitaan erityisesti sisäisellä motivaatiolla ja tuloksena oppilaat saatiin osallistumaan aktiivisemmin ja suorittamaan ja yrittämään monimutkaisempia operaatioita. Niemiec & Ryan (2009) huomauttavat, että useat tutkimukset ovat vahvistaneet opiskelijoiden omaehtoisuuden, kyvykkyyden ja yhteisöllisyyden olevan kriittisiä akateemisen motivaation sisäistämisessä. Tutkimuksen mukaan on selvää, että sisäinen motivaatio ja ulkoisen motivaation omaehtoisimmat tyypit vaikuttavat positiivisesti akateemisiin tuloksiin. (Niemiec & Ryan, 2009.)

4. Oppimisen tukeminen

Ohjelmoinnin oppimisen haasteista tunnistettuja merkittäviä haasteita olivat heikko ohjelmoinnin ja ohjelman suorituksen aikainen ymmärrys, ongelmanratkaisutaitojen heikkous ja opiskelijoiden motivaation puute. Näiden ongelmakohtien korjaamiseen on olemassa mahdollisia ratkaisuja, joista tutkimuskysymykselle olennaisimpia esitellään tässä kappaleessa. Vihavainen, Paksula & Luukkainen (2011) ihmettelevät, minkä takia edelleen suurin osa kursseista on järjestelty ohjelmointikielen rakenteen mukaan, vaikka tutkimus osoittaa ongelman olevan ohjelmointikielen syntaksin ja semanttisen sisällön ymmärtämisen sijaan opiskelijoiden kyvyssä rakentaa merkityksellisiä ohjelmia.

4.1 Käsitteellinen tietokone ohjelmoinnin ymmärryksen helpottamiseksi

Ohjelmoinnin vaatima korkea abstraktiotaso, kognitiivinen kompleksisuus ja kognitiivinen kuormitus ovat merkittäviä tekijöitä opiskelijoiden ohjelmointiin liittyvässä ymmärryksessä. Ohjelmoinnin ja tietokoneen ymmärtämistä voidaan tarkastella esimerkiksi käsitteellisen tietokoneen (Notional machine) avulla. Käsitteellinen tietokone on abstraktio (Sorva, 2013), idealisoitu konsepti (du Boulay, O'Shea & Monk, 1981) tietokoneen ominaisuuksista, joita käyttäjä opettelee kontrolloimaan. Käsitteellinen tietokone kuvaa tietokonetta ohjelman suorittajan roolissa jonkin ohjelmointikielen kontekstissa (du Boulay et al., 1981). Käsitteellinen tietokone ei tarkoita mentaalimallia, vaikkakin käsitteellisestä tietokoneesta voi muodostaa mentaalimallin. Käsitteellinen tietokone ei ole myöskään määritelmä tai visualisointi tietokoneesta. (Sorva, 2013.)

Noviiseille tärkeintä on käsitteellisen tietokoneen ymmärtäminen, mikä tarkoittaa ohjelmointikielen valinnassa loogista ja syntaktista yksinkertaisuutta ja hyvää näkyvyyttä ohjelman toimintaan. Tällainen ohjelmointikieli on suunniteltu enemmän noviisien oppimisen helpottamiseksi, kuin todelliseen ohjelmointiin. Komentojen tulisi olla yksinkertaisia ja kuvaavia. Näkyvyyttä parantavat muun muassa virheviestit ja ohjelmointiympäristöjen (IDE) kommenttikentät, jotka ovat esimerkki käsitteellisen tietokoneen tarjoamasta ikkunasta ohjelman suorituksen aikaisiin tapahtumiin. (du Boulay et al., 1981.)

Mayer (1976) tutki mallien vaikutusta ohjelmoinnin oppimiseen ja huomasi, että tietokoneen ja ohjelmointikielen toiminnan selittäminen ja mallintaminen edistää merkittävästi oppimista. Smith & Webb (1995) käyttävät sen sijaan termiä tietokoneen toimintaa kuvaava mentaalimalli. Noviiseilla tulisi olla virheiden etsimiseen ja poistamiseen visuaalinen työkalu (Smith & Webb, 1995). Sekä du Boulay et al., (1981), että Smith & Webb (1995) määrittelevät, että visualisoinnin tulee olla noviisille oikealla abstraktiotasolla, eikä esimerkiksi binäärikoodia. Naps et al. (2003) pitävät selvänä, että opiskelijan aktiivinen osallistuminen avustettuna ohjelman suorituksen visualisoinnilla johtaa merkityksellisempään oppimiseen.

Bennedsen & Schulte (2010) tutkivat manuaalisen jäljityksen ja visuaalisen työkalun (BlueJ) vaikutuksia objektit ensin-tyyppisessä ohjelmointiopetuksessa ja totesivat, ettei visuaalinen työkalu tarjonnut mitään etua manuaaliseen jäljitykseen verrattuna. Kölling (2008) näkee sen sijaan BlueJ:n edistäneen opiskelijoiden oppimista omassa

opetuksessaan, mutta kaikkia oppimisen ongelmia BlueJ ei pysty ratkaisemaan. UUhistle on toinen käsitteellisen tietokoneen ymmärrystä tukemaan kehitetty visuaalinen työkalu Python-ohjelmointikielelle. UUhistle poikkeaa muista työkaluista siten, että se on visuaalinen simulointiohjelma, joka tukee automaattisesti arvioitavia tehtäviä. Sorva & Sirkiä (2010) ovat tehneet ohjelman käytöstä laadullista analyysia yli tuhannelle opiskelijalle ja näkevät UUhistlen käyttämisen CS1-kursseilla lupaavana. UUhistle harjoituksineen ja visualisointeineen on lisännyt opiskelijan aktiivista osallistumista ja ohjelman käyttö on saanut myönteistä palautetta myös opiskelijoilta. (Sorva & Sirkiä, 2010.)

4.2 Objektit ensin-ajatusmalli ohjelmoinnin ymmärryksen helpottamiseksi

Yksi merkittävimmistä ongelmista ohjelmoinnin oppimisessa liittyi ongelmanratkaisuun ja erityisesti ongelman siirtämiseen alkuperäiseltä tietoalueelta ohjelmointikontekstiin. Viime aikoina tähän ongelmaan ovat useat tutkijat ja opettajat suositelleet niin sanottua objektit ensin-ajatusmallia (Objects first-paradigm). Objektit ensin-ohjelmointitekstikirjat lähtevät aihealueista, kuten objekti, luokka ja metodi, mutta eivät sisällä alussa juuri ollenkaan esimerkkejä lähdekoodista. (Sajaniemi & Kuittinen, 2008.) Tyypillisesti opetus alkeiskursseilla on järjestetty proseduraalisen ohjelmoinnin periaatteilla. Proseduraalisessa ohjelmoinnissa on tyypillistä erottelu aliohjelmiin, jossa aktiiviset proseduurit manipuloivat passiivisia tietorakenteita. Puhtaassa objektiorientoituneessa ohjelmoinnissa (Olio-ohjelmointi) kaikki elementit ovat objekteja. Objekteilla on niille itselleen kuuluvaa dataa ja metodeja. Tyypillisesti data on objektille yksityistä, osa metodeista on yksityisiä, kun taas joitakin metodeja voidaan tarjota muiden objektien kutsuttavaksi. (Rosson & Albert, 1990.)

Objektit ensin-lähtökohdan tukijat pitävät selvänä, että olio-ohjelmointi on ratkaisu ohjelmoinnin oppimisen haasteisiin. Väitteiden mukaan olio-ohjelmointi yksinkertaistaa huomattavasti ongelman siirtämistä alkuperäiseltä tietoalueelta ohjelmoinnin tietoalueelle. Tällä tarkoitetaan sitä, että asiaankuuluvien objektien löytäminen on luonnollista ja ongelmatietoalueen kartoittaminen siten helpompaa kuin proseduraalisessa ohjelmointiympäristössä. (Detienne, 1997.) Sajaniemen & Kuittisen (2008) mukaan objektit ensin-ajatusmalli palvelee enemmän ohjelmointitoimialaa ja pyrkii miellyttämään opiskelijoita, mutta objektit ensin-ajatusmallia tukevia metodologisesti oikeita tutkimuksia ei juurikaan ole olemassa. Suurimpia huolia olio-ohjelmoinnin opettamiseen ennen proseduraalista ohjelmointia ovat Hun (2004) mukaan olio-ohjelmoinnin liian korkea abstraktiotaso CS1-opiskelijoille, liiallinen materiaalin määrä CS1-kurssille, algoritmien heikompi osaaminen ja proseduraalisen ohjelmoinnin opetuksen huomiotta jättäminen. Sajaniemi & Kuittinen (2008) pitivät ongelmallisena erityisesti CS1-opiskelijoiden kannalta olio-ohjelmoinnin huomattavasti monimutkaisempaa käsitteellistä tietokonetta.

Kölling (1999) huomauttaa, että objektit-ensin ajatusmalli ei ole itse ongelma, vaan käytössä olevat opetusmetodit ja oppimisvälineet. Ongelmia ovat Köllingin mukaan erityisesti C++:n käyttäminen olio-ohjelmoinnin oppikirjoissa ja kursseilla, sekä olio-ohjelmointia huonosti tukevat ohjelmointiympäristöt. (Kölling, 1999.) Börstler, Nordström ja Paterson (2011) analysoivat 191 opettajien tekemää arviota 21:stä Olio-esimerkistä, jotka esiintyivät yhdessätoista eri objektit ensin-oppikirjassa. Tulosten mukaan esimerkkien taso ei ollut erityisen hyvä. Suuri osa esimerkeistä epäonnistui muun muassa objekti-ajattelutavan tukemisessa. (Börstler et al., 2011.) Köllingin (1999) mukaan Java olisi paras olemassa olevista olio-ohjelmointikielistä noviiseille,

mutta edelleen kaukana ideaalista. Tärkein syy Köllingin mukaan objektit ensin-ajatusmallissa liittyy siihen, että ajattelumallin muutos (Paradigm shift) olio-ohjelmoinnista proseduraaliseen ohjelmointiin on huomattavasti nopeampi, kuin toisinpäin. (Kölling, 1999.) Gries (2008) kertoo opettaneensa ohjelmointia objektit ensin-ajatusmallilla menestyksekkäästi jo kahdeksan vuoden ajan ja pyytääkin huomioimaan opetusmetodien tärkeyden objektit ensin-keskustelussa. Griesin (2008) mukaan objektit ensin-ajatusmalli on ainoa oikea lähtökohta ohjelmoinnin opetukseen.

Sajaniemi & Kuittinen (2008) ehdottavat objektit ensin-ajatusmallin systemaattista tutkimusta, jotta saataisiin selvitettyä olio-ohjelmoijan mentaalimallit, erot eksperttien ja noviisien taidoissa, strategioissa ja mentaalimalleissa sekä tutkittaisiin olio-ohjelmoijanoviisien kognitiivista kehitystä. Sajaniemi & Hu (2006) sen sijaan ehdottavat objektit ensin-ajatusmallin sijaan muuttujat ensin-ajatusmallia (Variables-first approach), jossa CS1 aloitettaisiin sekä proseduraaliselle, että olio-ohjelmoinnille tyypillisistä tietorakenteista, muuttujista ja yleisistä kontrollirakenteista. Detiennen (1997) mukaan on kuitenkin näyttöä siitä, että olio-ohjelmoinnista on hyötyä ammattilaisille.

4.3 Opiskelumotivaation kohentaminen

Opiskelumotivaatiolla on merkittävä rooli merkityksellisessä oppimisessa. Sisäisesti motivoituneet opiskelijat menestyvät opiskeluissaan ulkoisesti motivoituneita paremmin. Niemiec & Ryan (2009) näkevät tärkeänä opetusstrategiat, joissa opiskelijoille annetaan vaihtoehtoja, tunnustetaan opiskelijoiden tuntemukset aiheista ja minimoidaan opiskelijoihin kohdistettua painetta ja kontrollia. Yksi ehdotetuista malleista on minimaalisen ohjauksen lähestymistapa itseohjattu opiskelu (Self-directed learning). Muita minimaalisen ohjauksen lähestymistapoja ovat löydöspohjainen oppiminen (Discovery-based learning), tutkimuslähtökohtainen oppiminen (Inquiry-based learning), ongelmalähtökohtainen oppiminen (Problem-based learning), kokemusperäinen oppiminen (Experiential learning) ja konstruktiivinen oppiminen (Constructivist learning). Lähestymistapoja ei ole tarpeen eritellä tässä yhteydessä, sillä ne kaikki kuuluvat jossakin määrin konstruktiivisen näkemyksen piiriin, joka korostaa oppijan roolia. (Kirschner, Sweller & Clark, 2006.)

Keskustelua itseohjatusta opiskelusta ja sen tehokkuudesta on käyty paljon. Kirschner et al. (2006) huomauttavat, että konstruktiivinen itseohjattu opiskelu on todettu epäonnistuneeksi useissa tutkimuksissa erityisesti minimaalisen ohjauksen aiheuttaman korkean kognitiivisen kuorman takia. Hmelo-Silver, Duncan & Chinn (2007) näkevät ongelmallisena sen, että Kirschner et al. lokeroivat selvästi eri lähestymistapoja minimaalisen ohjauksen lähestymistapojen alle ja että tutkimus jättää huomioimatta useat empiiriset todisteet ongelmalähtökohtaista ja tutkimuslähtökohtaista oppimista tukevista tutkimuksista. Schmidt, Loyens, van Gog & Paas (2007) eivät tue Kirschner et al.:in näkemystä vahvan ohjeistamisen ylivertaisuudesta, vaan näkevät myös kognitiivisen kuormitusteorian kannalta ongelmalähtökohtaisen oppimisen paremmaksi vaihtoehdoksi. Sweller, Kirschner & Clark (2007) toteavat Hmelo-Silver et al:in esittämät empiiriset todisteet metodologisesti heikoiksi. Sekä Hmelo-Silver et al. (2007) että Schmidt et al. (2007) pitävät selvänä, että onnistuakseen, itseohjatun opiskelun tulee olla ohjattua ja sisältää oikea-aikaista opiskelun tukemista (Scaffolding).

4.3.1 Extreme Apprenticeship-metodi motivaation ja ongelmanratkaisutaitojen kohottamiseksi

Extreme Apprenticeship-metodi on suunniteltu ohjelmointiammattilaisten Extreme Programming-metodien ja Cognitive Apprenticeship-mallin pohjalta. Cognitive Apprenticeship-mallin mukaan opetus on jaettu kolmeen vaiheeseen. Mallintamisvaiheessa (Modeling) voidaan esimerkiksi käyttää työstettyä esimerkkiä (Worked example), jossa opettaja näyttää ohjelmointitehtävän tekemisen alusta loppuun. Seuraava vaihe on oikea-aikainen opiskelun tukeminen, jossa opiskelijat harjoittelevat ohjelmointia itse ja ohjeistus on lähinnä riittävien vihjeiden antamista opiskelijoille. Kolmas vaihe on häivytysvaihe (Fading), jossa opiskelija on saavuttanut riittävän tason ja opiskelun tukeminen lopetetaan. (Vihavainen, Paksula & Luukkainen, 2011.)

Metodin tarkoituksena on korostaa aktiivista oppimista ja kohottaa opiskelijoiden sisäistä motivaatiota. Oikea-aikainen opiskelun tukeminen, jossa keskustellaan opiskelijoiden ongelmanratkaisumalleista ja annetaan hienovaraisia vihjeitä, myös tukee sisäistä motivaatiota ja vähentää kognitiivista kuormaa. Keskeistä metodissa on muun muassa tekemällä oppiminen, jatkuva palaute, saarnaamisen välttäminen, oikea-aikainen avustaminen, itsenäisen tiedonhaun tukeminen ja harjoitusten pakollisuus. Metodia kokeiltiin CS1-kurssilla ja CS2-kurssilla Helsingin yliopiston tietojenkäsittelytieteen laitoksella vuonna 2010. Vuosien 2002-2009 syksyn CS1-kurssien keskimääräinen läpäisyprosentti oli 58,5 prosenttia ja keväisten CS1-kurssien keskimääräinen läpäisyprosentti 43,7 prosenttia. Extreme Apprenticeship-metodilla keväällä 2010 toteutetun CS1-kurssin läpäisyprosentti oli 70,1 prosenttia. CS2-kurssilla keskimääräiset läpäisyprosentit vuosilta 2002-2009 olivat syksyisin 60,1 prosenttia ja keväisin 45,3 prosenttia. Keväällä 2010 Extreme Apprenticeship-metodilla toteutetun CS2-kurssin läpäisyprosentti oli 86,4 prosenttia. Suurin osa palautteesta piti kurssia motivoivana ja palkitsevana. (Vihavainen et al., 2011.)

Vuoden 2010 onnistumisen jälkeen Extreme Apprenticeship-metodia laajennettiin ja kokeilua jatkettiin Helsingin yliopiston tietojenkäsittelytieteen CS1- ja CS2-kursseilla. Kursseihin on ensimmäisen toteutuksen jälkeen implementoitu niin sanottu Test My Code (TMC)-ympäristö, joka tarkastaa opiskelijoiden ohjelmointiharjoituksia NetBeans-ohjelmointiympäristön kautta automaattisesti. Vihavainen & Luukkainen (2013) näkevät, että TMC on vaikuttanut nimenomaan myönteisesti opiskeluun, sillä ohjaajilla on enemmän aikaa opiskelijoiden tukemiseen. Kolmen vuoden siirtymävaiheen jälkeen läpäisyprosentit ovat pysyneet huomattavan korkeina sekä CS1-, että CS2-kursseilla. Ohjaajien mukaan sitäkin myönteisempää on opiskelijoiden kohonnut työmäärä ja palaute, joka on pysynyt erittäin positiivisena. (Vihavainen & Luukkainen, 2011.)

4.3.2 Itseohjattu opiskelu motivaation kohottamiseksi

Jyväskylän yliopistossa kehitettiin itseohjattu kurssimalli ohjelmoinnin CS3-kurssille. Kurssi on funktionaalisen ohjelmoinnin kurssi jo pidemmällä oleville opiskelijoille, mutta motivaation näkökulmasta kurssimallin uudelleensuunnittelu liittyy myös tämän kirjallisuuskatsauksen piiriin. Kurssin suunnittelussa huomioitiin erityisesti itseohjautuvuusteoriaa ja täten pyrittiin välttämään ulkoista palkitsemista, deadlineja, määräyksiä, painostavaa arviointia ja säädettyjä tavoitteita. Isomöttönen & Tirronen (2013) näkevät, että itseohjattuun opiskelu-keskusteluun voidaan relevanttina liittää Klugin (1976) näkemys ja keskustelunavaus arvioinnin tarpeettomuudesta. Klug (1976)

näkee, että ulkoinen arviointi johtaa opiskelijoiden opiskeluprosessien ohjautumiseen niihin asioihin, jotka vaikuttavat suoraan ulkoiseen arviointiin. Suurimpana kurssin uudelleensuunnittelun motivoijana nähtiin kuitenkin opetusresurssien valuminen hukkaan. Vanhassa mallissa luentoja järjestettiin aiheista, joita opiskelijat olisivat voineet selvittää helposti itsekin. Isomöttönen & Tirronen (2013) huomauttavat, että itseohjattu opiskelu ei heidän näkemyksensä mukaan tarkoita opiskelijoiden jättämistä yksin, kuten Kirschner et al. asian esittivät. (Isomöttönen & Tirronen, 2013.)

Itseohjattuun opiskeluun pohjautuvassa kurssimallissa ei järjestetty luentoja lainkaan. Kurssimallissa annettiin seuraavan viikon tehtävät viikon lopussa. Seuraavan viikon alussa järjestettiin kontaktiopetusta, jossa autettiin opiskelijoita ongelmissa. Tätä seurasi kaksi päivää itsenäistä työskentelyä, tehtävien palautus ja toinen kontaktiopetustapahtuma, jossa esitettiin mielenkiintoisia ratkaisuja ja vastaukset tehtäviin. Kurssimallissa käytettiin myös osittain ryhmätyöskentelyä. Uuden kurssimallin myötä läpäisyprosentti nousi merkittävästi ja oppilaiden palaute oli myönteisempää, mutta ongelmiakin löydettiin. Merkittävimmät ongelmat liittyivät opiskelijoiden itseohjautuvuuteen ryhmätyöskentelyssä, joidenkin opiskelijoiden oppimisprosessien sekaantumiseen kurssin liian nopeassa rytmissä ja teorian ymmärrykseen. (Isomöttönen & Tirronen, 2013.) Myöhemmässä tutkimuksessa kahden kurssitoteutuksen jälkeen Isomöttönen, Tirronen & Cochez (2013) näkevät itseohjatussa opiskelussa edelleen ongelmia. Ongelmia huomattiin muun muassa itseohjautuvuuden puutteessa, sillä opiskelijat eivät aloittaneet tehtävien tekemistä ajoissa tai itsenäisesti. Ryhmätyöskentelyssä ongelmaksi muodostuivat huomattavan eritasoiset opiskelijat, vaikka kurssin alussa tähän pyrittiin kiinnittämään huomiota. (Isomöttönen, Tirronen & Cochez, 2013.)

4.3.3 Pariohjelmointi opiskelussa opiskelijoiden motivaation parantamiseksi

Pariohjelmointi (Pair programming) liittyy terminä Extreme Programming-ajatusmalliin. Pariohjelmoinnin ideana on asettaa kaksi ohjelmoijaa saman koneen äärelle. Ohjelmoijista toinen on ”kuski” (Driver), joka tuottaa aktiivisesti lähdekoodia. Toisen roolina on seurata ohjelmointia ja havaita taktisia tai strategisia ongelmia ja virheellistä syntaksia sekä logiikkaa. (Williams & Kessler, 2000.) Nosek (1998) huomasi ensimmäisenä pariohjelmoinnin hyödyt ammattilaisille. Pariohjelmoinnin motivaatiota auttava piirre selittyy erityisesti ryhmäpaineella. Pariohjelmoinnin tapauksessa voidaan puhua ”paripaineesta”, jossa toisen opiskelijan läsnäolo saa molemmat opiskelijat huomattavan motivoituneiksi. Tämä on huomattavissa selvästi nopeammasta tehtävän aloittamiseen kuluvasta ajasta ja katkeamattomasta opiskelusta. (Williams & Kessler).

Williams & Kessler (2001) tutkivat pariohjelmoinnin vaikutusta ohjelmistotekniikan kurssilla, jossa oli 41 opiskelijaa. Tutkimuksessa opiskelijat jaettiin kahteen ryhmään, joista toinen suoritti ohjelmointiharjoituksia itsenäisesti ja toinen pariohjelmoinnilla. Pariohjelmoijat käyttivät keskimäärin 15 prosenttia enemmän aikaa ohjelmointiharjoitusten tekemiseen, mutta mediaania tarkasteltaessa ero ei ollut tutkimuksen kannalta merkittävä. Pariohjelmointi vaikutti opiskelijoiden työskentelyn tasoon ja määrään myönteisesti ja pariohjelmoijat vaikuttivat olevan enemmän motivoituneita saamaan harjoitukset valmiiksi. Opiskelijoista 84 prosenttia ilmoitti nauttineensa tehtävien tekemisestä enemmän pariohjelmoinnin ansiosta ja opiskelijat olivat pariohjelmoidessa varmempia omista ratkaisuistaan. Opettajat myös huomasivat,

että opiskelijoilla oli pariohjelmoidessa vähemmän kysyttävää ja tehtävissä huijaaminen väheni. (Williams & Kessler, 2001.)

McDowell, Werner, Bullock & Fernald (2002) kokeilivat pariohjelmointia osana CS1-kurssia. Tutkimuksessa 172 opiskelijaa teki tehtäviä pariohjelmoinnin avulla ja 141 itsenäisesti. Kaikki tehtävät, luennot ja pistokkaat olivat verrannollisia, lopputentti identtinen ja kaikilla opiskelijoilla oli sama ohjaaja. Ohjelmointiharjoituksista pariohjelmoijat saivat keskimäärin 86,3 prosenttia oikein ja yksin ohjelmoineet 67 prosenttia. Pariohjelmoijien mediaani oli 88 prosenttia ja yksinohjelmoineiden 68 prosenttia, joten pariohjelmoijat menestyivät merkittävästi paremmin ohjelmointiharjoituksissa. Tutkimuksessa huomautetaan, että pariohjelmoijien arvosanat voivat olla vääristyneitä. Parin molemmat osapuolet saivat saman arvosanan, jolloin mahdollisesti paremman osapuolen arvosana on kohottanut heikomman osapuolen arvosanaa. Lopputentin tuloksissa pystyttiin arvioimaan paremmin pariohjelmoijien ymmärrystä ja kykyä ohjelmoida yksin. Pariohjelmoijat saivat keskimäärin 72,9 prosenttia oikein ja yksin ohjelmoineet 74,6 prosenttia oikein. Pariohjelmien mediaanitulos oli 79,2 prosenttia ja yksinohjelmoineiden 78,3 prosenttia, joten merkittävää eroa ei ollut. Tutkimuksessa merkittävin löydös oli kuitenkin keskeytysprosenteissa. Pariohjelmoijista kurssin loppuun suoritti 92,4 prosenttia, kun yksinohjelmoijista kurssin päätyi suorittamaan 75,9 prosenttia. (McDowell et al., 2002.)

5. Pohdinta

Ihmisellä on taipumus pyrkiä esittämään yksinkertaistettuja ratkaisuja monimutkaisiin ongelmiin ja tämän kirjallisuuskatsauksen esittämän aiemman tutkimuksen perusteella ohjelmoinnin oppimisen haasteisiin vastaaminen ei myöskään ole erityisen yksinkertaista. Oppiminen ja ohjelmointi ovat itsessään komplekseja kognitiivisia ilmiöitä. Oppimisen tukeminen ja motivointi ovat usein haasteellisia asioita jo helpommissakin aihealueissa. Ohjelmointi taas on korkeaa formaalia ajattelutapaa vaativa aktiviteetti. Abstraktiotaso on korkea ja päällekkäisiä samaan aikaan aktiivisia kognitiivisia kykyjä vaaditaan useita.

Voidaan kuitenkin perustellusti sanoa, että osa havainnoiduista haasteista on lähinnä seurausta perustavanlaatuisimmista ongelmista. Kaikille tässä kirjallisuuskatsauksessa esitetyille ongelmille on yhteistä se, että ne edustavat perinteisen luentopohjaisen ohjelmoinnin opetuksen opiskelijoiden haasteita. Vajaan neljän vuosikymmenen ajan empiiriset tutkimukset ovat osoittaneet mitä monimuotoisemman joukon valideja ongelmia opiskelijoiden ohjelmointitiedoissa ja – taidoissa. Tästä huolimatta perinteistä luentopohjaista ohjelmoinnin opetusta edelleen järjestetään useissa yliopistoissa. Voisi olla oikeutettua kysyä, onko perinteisellä luentopohjaisella opetuksella mitään paikkaa ohjelmoinnin opetuksessa. On kiistaton tosiasia, että ohjelmointia opitaan vain tekemällä.

Aikainen ohjelmoinnin haasteiden tutkimus on kohdentunut hyvin paljon noviisien virheisiin, väärinkäsityksiin ja erilaisten ohjelmointiongelmien löytämiseen. Motivaatiosta ja sen vaikutuksesta merkitykselliseen oppimiseen on alettu keskustelemaan toden teolla vasta 2000-luvulla. Opiskelijoilla ei vaikuttaisi olevan erityistä mielenkiintoa ohjelmointia kohtaan, eikä erityisesti syntaksiin keskittyvien luentojen seuraamiseen. Suuri osa opiskelijoista osallistuu ohjelmointikursseille, koska ne ovat pakollinen osa opintoja. Perinteinen luentopohjainen opetus ei myöskään innosta opiskelijoita. Opiskelijan passiivinen rooli tällaisessa kurssitoteutuksessa ei anna autonomian tunnetta, kohota yhteisöllisyyttä saatikka edistä kompetenssin tunnetta. Tämän myötä luentopohjainen kurssi ei salli sisäisen motivaation kehittymistä opiskelijoissa, joilla motivaatio oli puutteellista tai merkittävästi ulkoista jo ennestään.

On selvää, etteivätkö opiskelijoiden ongelmanratkaisutaidot ja käsitys ohjelmoinnista olisi puutteellisia. Nämä ovat kuitenkin juuri niitä asioita, joihin aktiivisella oppimisella mahdollisesti kyettäisiin vaikuttamaan. Itseohjatulla opiskelulla voitaisiin vaikuttaa sisäisen motivaation kehitykseen ja saada opiskelijasta aktiivinen tiedon hankkija. Jyväskylän yliopiston mallissa voi olla ongelmia, mutta toisaalta Helsingin yliopiston Extreme Apprenticeship-malli on onnistunut erittäin hyvin. Jyväskylän mallin soveltaminen suoraan CS1- ja CS2-kursseille on mahdotonta, sillä tutkimuksen mukaan CS3-opiskelijoilla oli ongelmia kurssisuorituksessa. Selvästikin kognitiivisesti haasteellisten aihepiirien oppiminen tarjoaa omat haasteensa itseohjatussa mallissa. Tähän vaikuttaa myös opiskelijoiden erilaiset kognitiiviset tyylit. Oletettavasti Extreme Apprenticeship-mallin tavoin oikea-aikaisella oppimisen tukemisella kyetään kuitenkin minimoimaan ongelmat.

Vaikka itseohjattu opiskelu on saanut kritiikkiä osakseen väitetystä korkeasta kognitiivisesta kuormastaan, voi tämä olla vältettävissä tarpeellisen ja laadukkaan

materiaalin tarjoamisella ja oikea-aikaisella opiskelun tukemisella. Itseohjatun opiskelun ei tarvitse tapahtua niin äärimmäisessä muodossa, kuin Jyväskylän yliopistossa sitä on kokeiltu, vaikka Jyväskylässäkään ideana ei missään nimessä ollut opiskelijoiden yksin jättäminen. Extreme Apprenticeship-malli ei vaikuta erityisen autonomiselta pakollisine harjoituksineen, mutta voi olla, että harjoitusten pakollisuus on ainoa pätevä tapa järjestää ohjelmoinnin opetusta. Exteme Apprenticeship-mallissa autonomian tunne luotiin tekemisen kautta, vapaalla ongelmanratkaisulla ja tiedon hankintaan kannustamisella. Tiedon hankintaan kannustaminen on toinen itseohjatun opiskelun vahvuus. Erityisesti teknologia-alalla, jossa kehitys on nopeaa, on opiskelijan tiedonhankintataidoilla korostunut merkitys. Tänä päivänä opittu taito voi olla muinainen ja tarpeeton taito kahdenkymmenen vuoden päästä ja silloin jokainen opiskelija kiittää opetustapaa, jossa on kannustettu ja ohjattu aktiiviseen tiedon hankintaan itse.

Ohjelmoinnin harjoitusten järjestäminen on toinen mielenkiintoinen keskustelun ansaitseva aihepiiri. Perinteisesti ohjelmointi on nähty yksin tekemisenä, mutta Extreme Programming-näkökulma on tuonut vaihtoehtoisen näkemyksen keskusteluun mukaan. Ammattilaisten menestyksekkäästi hyödyntämälle pariohjelmointi-tekniikalle on varsin nopeasti kyetty löytämään hyötyjä myös ohjelmoinnin opetuksessa. Esimerkiksi Jyväskylän yliopiston itseohjatussa mallissa ryhmätyöskentelyssä nähtiin ongelmia. Tutkimusten perusteella pariohjelmointi voisi olla yksi keino opiskelijoiden sisäisen motivaation kohottamiseen yhteisöllisyyden kautta ja varteenotettava vaihtoehto ryhmätyöskentelylle.

Objektit ensin-keskustelu on käynyt kuumana viimeiset vuodet ja mielipiteet ovat poikkeuksellisen vahvoja. Toisaalta osa olio-ohjelmoinnin tukijoista näkee proseduraalisen paradigman vanhanaikaisena ja syyllistävät opettajia laiskoiksi. Proseduraalisen paradigman tukijat sen sijaan näkevät olio-ohjelmoinnin liian abstraktina noviiseille. Keskustelu aiheesta on tärkeää, mutta tähän mennessä käyty keskustelu on ollut hedelmätöntä ja syyllistävää. Molemmissa ajattelumalleissa on puolensa, mutta huomioitavaa on myös olio-ohjelmoinnin tukijoiden huoli ajatusmallin siirtymisen hitaudesta proseduraalisesta ohjelmoinnista olio-ohjelmointiin. Voi olla, että ohjelmoinnin haasteet eivät tule ratkeamaan merkittävästi pelkällä ajattelumallin valinnalla. Keskeistä on lähinnä se, että edes jompikumpi ajattelumalleista opittaisiin kiitettävästi ennen toisen paradigman esittelyä. Ohjelmoinnin oppiminen sisältää jo valmiiksi valtavan määrän potentiaalisia väärinkäsityksen aiheita ja asiaa ei auta se, että opetuksessa siirrytään kiireellä käsittelemään toista paradigmaa ennen kuin opiskelijat ovat oppineet ensimmäistäkään ajattelumallia.

6. Johtopäätökset

Kuten jo pohdinnan alussa mainittiin, monimutkaisiin kysymyksiin harvoin on olemassa yksinkertaisia vastauksia. Mitä sitten ovat ohjelmoinnin oppimisen haasteet? Oletettavasti ohjelmoinnin oppimisen haastavuuteen merkittävästi vaikuttavia asioita on kolme. Näitä ovat opiskelijan motivaation puute, heikot ongelmanratkaisutaidot ja ohjelmoinnin yleinen ymmärtämättömyys. Kirjallisuustutkimuksen toinen keskeinen kysymys liittyykin siihen, miten näitä haasteita kyetään vähentämään tai välttämään. Voi olla, että avain näihin monimuotoisiin ohjelmoinnin oppimisen haasteisiin on pohjimmiltaan opiskelijan motivaatiossa. Ilman riittävää motivaatiota, yksikään haasteista ei ratkea. Riittävällä sisäisellä motivaatiolla sen sijaan jokainen opiskelija kykenee oppimaan haluamansa. Extreme Apprenticeship-malli voisi olla yksi niin sanotuista hyvistä vastauksista ohjelmoinnin oppimisen ongelmiin.

Pariohjelmoinnista saadut myönteiset tutkimukset myös tarjoavat mielenkiintoisen vaihtoehtoisen ajatuksen uudistetusta ja muokatusta Extreme Apprenticeship-mallista, jossa järjestettäisiin pariohjelmointiharjoituksia. Kirjallisuustutkimus herättää kysymyksiä, jotka liittyvät erityisesti Oulun yliopiston tietojenkäsittelytieteen laitoksen ohjelmoinnin opetuksen järjestämiseen. Kiinnostavaa voisi olla esimerkiksi motivaatiotutkimus Johdatus ohjelmointiin-kurssin opiskelijoille.

Lähdeluettelo

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*. 9(4). 422-433.

Ambrosio, A. P., Almeida, L., Franco, A., Martins, S., & Georges, F. (2012). Assessment of Self-regulated Attitudes and Behaviors of Introductory Programming Students. *Frontiers in Education Conference in Seattle 2012*. 1-6. doi: 10.1109/FIE.2012.6462314

Anderman, E. M., & Gray D. (2015). Motivation, Learning and Instruction. In J. D. Wright (eds), International *Encyclopedia of the Social & Behavioral Sciences (Second Edition)*. (928-935). Elsevier.

Anderson, J.A. (1982). Acquisition of Cognitive Skill. *Psychological Review*. 89(4). 369-406.

Bennedsen, J., & Schulte, C. (2010). BlueJ Visual Debugger for Learning the Execution of Object-Oriented Programs? *ACM Transactions on Computing Education*. 10(2). Article 8.

Bergin, S., & Reilly, R. (2005). Programming: Factors that Influence Success. *ACM SIGCSE Bulletin*. 37(1). 411-415.

Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. *Proceedins of the 17th Workshop of the Psychology of Programming Interest Group (PPIG17)*. 293-304.

Byrne, P., & Lyons, G. (2001) The Effect of Student Attributes on Success in Programming. *ITiCSE '01 Proceedings of the 6th annual conference on Innovation and technology in computer science education*. 49-52. doi: 10.1145/507758.377467

Börstler, J., Nordström, M., & Paterson, J. (2011). On the Quality of Examples in Introductory Java Textbooks. *ACM Transactions on Computing Education*. 11(1). Article 3.

Cantwell Wilson, B., & Shrock, S. (2001). Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. *ACM SIGCSE Bulletin.* 33(1). 184-188.

Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course: a learning theoretic approach. *ICER '07 Proceedings of the third international workshop on Computing education research*. 112-122. doi: 10.1145/1288580.1288595

Cordova, D. I., & Lepper, M. R. (1996). Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization and Choice. *Journal of Educational Psychology*. 88(4). 715-730.

Deci, E. L., Koestner, R., & Ryan, R. M. (1999). A Meta-Analytical Review of Experiments Examining the Effects of Extrinsic Rewards on Intrinsic Motivation. *Psychological Bulletin*. 125(6). 627-668.

Derry, S. A. (1996). Cognitive schema theory in the constructivist debate. *Educational Psychologist*. 31(3-4). 163-174. doi: 10.1080/00461520.1996.9653264

Detienne, F. (1997). Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. *Interacting with Computers*. 9(1). 47-72.

Douglass, C., & Morris, S. R. (2014). Student perspectives on self-directed learning. *Journal of Scholarship of Teaching and Learning*. 14(1). 13-25.

du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*. 14(3). 237-249.

Feigenbaum, E. A. (1967). Information Processing and Memory. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 4*. 37-51.

Fincher, S. (1999). What are we doing when we teach programming? *29th Annual Conference on Frontiers in Education (FIE '99)*. IEEE. 12A4/1-12A4-5. doi: 10.1109/FIE.1999.839268

Fitts, P. M. (1964). Perceptual-Motor Skill Learning. In A.W. Melton (eds), *Categories of Human Learning*. 243-285. New York, Academic Press.

Gomes, A., & Mendes, A. J. (2007). Learning to program – difficulties and solutions. *Proceedings of the International Conference on Engineering Education, ICEE '07 Coimbra, Portugal.* CD-ROM.

Gomes, A., Santos, A., & Mendes, A. J. (2012). A study on students' behaviours and attitudes towards learning to program. *Proceedings of the 17th annual conference on Innovation and technology in computer science education (ITiCSE '12)*. 132-137. doi: 10.1145/2325296.2325331

Graham, S., & Weiner, B. (1996). Theories and principles of motivation. In D.C. Berliner & R. C. Calfee, *Handbook of Educational Psychology*. (63-84). New York, USA: Macmillan

Gries, D. (2008). A Princpled Approach to Teaching OO First. *ACM SIGCSE Bulletin*. 40(1). 31-35.

Guiffrida, D. A., Lynch, M. F., Wall, A. E., & Abel, D. S. (2013). Do Reasons for Attending College Affect Academic Outcomes? A Test of a Motivational

Model From a Self-Determination Theory Perspective. *Journal of College Student Development*. 54(2). 121-139.

Hmelo-Silver, C. E., Duncan, R. G., & Chinn, C. A. (2007). Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller & Clark (2006). *Educational Psychologist*. 42(2). 99-107.

Hu, C. (2004). Rethinking of Teaching Objects-First. *Education and Information Technologies*. 9(3). 209-218.

Isomöttönen, V., & Tirronen, V. (2013). Teaching Programming by Emphasizing Self-Direction: How Did Students React to the Active Role Required of Them? *ACM Transactions on Computing Education*. 13(2). Article 6.

Isomöttönen, V., Tirronen, V., & Cochez, M. (2013). Issues with a Course That Emphasizes Self-Direction. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 111-116.

Jenkins, T. (2001). The motivation of students of programming. *Proceedings of the 6th annual conference on Innovation and technology in computer science education (ITiCSE '01)*. 53-56. doi: 10.1145/507758.377472

Kirschner, P.A. (2002). Cognitive load theory: implications of cognitive load theory on the design of learning. *Learning and Instruction*. 12(1). 1-10. doi 10.1016/S0959-4752(01)00014-7

Kirschner, P.A., Sweller, J., & Clark R.E. (2006). Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constuctivist Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*. 41(2). 75-86.

Klug, B. (1976). To grade, or not to grade. *Studies in Higher Education*. 1(2). 197-207.

Kölling, M. (1999). The problem of teaching object-oriented programming. Part I: Languages. *Journal of Object-Oriented Programming*. 11(8). 8-15.

Kölling, M. (2008). Using BlueJ to Introduce Programming. In J. Bennedsen, M. E. Caspersen & M. Kölling (eds.), *Reflections on the Teaching of Programming*. (98-115). Berlin, Germany: Springer.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H-M. (2005). A study of the diffictulties of novice programmers. *ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education*. 14-18. doi: 10.1145/1151954.1067453

Lister, R., Fone, W., McCartney, R., Seppälä, O., Adams, E. A., Hamer, J., Moström, J. E., Simon, B., Firzgerald, S., Lindholm, M., Sanders, K., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ITiCSE-WGR '04 Working group reports from ITiCSE on*

Innovation and technology in computer science education. 119-150. doi: 10.1145/1044550.1041673

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. *ICER '08 Proceedings of the Fourth International Workshop on Computing Education Research*. 101-112. doi: 10.1145/1404520.1404531

Mayer, R. E. (1976). Some conditions of meaningful learning for computer programming: advance organizers and subject control of frame order. *Journal of Educational Psychology*. 68(2). 143-150.

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys (CSUR)* 13(1). 121-141. doi: 10.1145/356835.356841

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y, B-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ITiCSE-WGR '01 Working group reports from ITiCSE on Innovation and technology in computer science education*. 125-180. doi: 10.1145/572133.572137

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE'02)*. 38-42.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*. 101(2). 81-97.

Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming – Views of Students and Tutors. *Education and Information Technologies*. 7(1). 55-66.

Moser, R. (1997). A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it. *ITiCSE '97 Proceedings of the 2nd conference on integrating technology into computer science education*. 114-116. doi: 10.1145/268809.268853

Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velazquez-Iturbide, J.A. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*. 35(2). 131-152.

Newell, A. (1979). Reasoning, problem solving and decision processes: the problem space as a fundamental category. *Technical Report CMU-CS-79-133*. Department of Computer Science, Carnegie-Mellon University. Pittsburgh, Pennsylvania.

Niemiec C. P., & Ryan, R. M. (2009). Autonomy, competence and relatedness in the classroom – Applying self-determination theory to educational practice. *Theory and Research in Education*. 7(2). 133-144.

Nosek, J. T. (1998). The Case for Collaborative Programming. *Communications of the ACM*. 41(3). 105-108.

Ormerod, T. (1990). Human cognition and Programming. In J.M. Hoc, T.R.G. Green, R. Samurcay & D.J. Gilmore, *Psychology of Programming* (63-82). London, UK: Academic Press Ltd.

Pacheco, A., Gomes, A., Henriques, J., de Almeida, A. M., & Mendes, J. A. (2008) Mathematics and Programming: Some studies. *CompSysTech '08 Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*. Article 77. doi: 10.1145/1500879.1500963

Pennington, N., & Grabowski, B. (1990). The Tasks of Programming. In J. M. Hoc, T.R.G. Green, R. Samurcay & D. J. Gilmore, *Psychology of Programming* (45-62). London, UK: Academic Press Ltd.

Pillay, N., & Jugoo. V. R. (2005). An Investigation into Student Characteristics Affecting Novice Programming Performance. *ACM SIGCSE Bulletin*. 37(4). 107-110.

Rist, R.S. (1989). Schema Creation in Programming. *Cognitive Science.* 13(3). 389-414. doi: 10.1207/s15516709cog1303_3

Rist, R.S. (1995). Program Structure and Design. *Cognitive Science*. 19(4). 507-562. doi: 10.1207/s15516709cog1904_3

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13(2). 137-172.

Rountree, N., Rountree, J., & Robins, A. (2002). Predictors of Success and Failure in a CS1 Course. *ACM SIGCSE Bulletin*. 34(4). 121-124.

Rosson, M. B., & Alpert, A.R. (1990). The Cognitive Consequences of Object-Oriented Design. *Human-Computer Interaction*. 5(4). 345-379.

Ryan, R. M., & Deci, E. L. (2000). Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being. *American Psychologist*. 55(1). 68-78.

Sajaniemi, J., & Hu, C. (2006). Teaching Programming: Going beyond "Objects First". *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group (PPIG2006)*. 255-265.

Sajaniemi, J., & Kuittinen, M. (2008). From procedures to objects: A research agenda for the psychology of object-oriented programming education. *Human Technology*. 4(1). 75-91.

Schmidt, H. G., Loyens, S. M. M., van Gog, T., & Paas, F. (2007). Problem-Based Learning is Compatible with Human Cognitive Architecture: Commentary on Kirschner, Sweller, and Clark (2006). *Educational Psychologist*. 42(2). 91-97.

Shneiderman, B., & Mayer, R. E. (1975). Towards a Cognitive Model of Programmer Behavior. *Technical Report No. 37*. Department of Computer Science, Indiana University, Bloomington, Indiana.

Shneiderman, B., & Mayer, R. E. (1979). Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *International Journal of Computer & Information Sciences*. 8(3). 219-238.

Smith, P.A., & Webb, G. I. (1995). Reinforcing a generic computer model for novice programmers. *Proceedings of the 7th Australian Society for Computer in Learning in Tertiary Education Conference (ASCILITE'95)*.

Sorva, J. (2013). Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education*. 13(2). Article 8.

Sorva, J., & Sirkiä, T. (2010). UUhistle – A Software Tool for Visual Program Simulation. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. 49-54.

Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*. 12(2). 257-285. doi: 10.1207/s15516709cog1202_4

Sweller, J., Kirschner, P. A., & Clark, R. E. (2007). Why Minimally Guided Teaching Techniques Do Not Work: A Repty to Commentaries. *Educational Psychologist*. 42(2). 115-121.

Taatgen, N.A., Huss, D., Dickison, D., & Anderson J.R. (2008). The Acquisition of Robust and Flexible Cognitive Skills. *Journal of Experimental Psychology*. 137(3). 548-565. doi: 10.1037/0096-3445.137.3.548

Tan, P-H., Ting, C-Y., & Ling, S-W. (2009). Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. *ICCTD '09 Proceedings of the 2009 International Conference on Computer Technology and Development – Volume 01.* 42-46. doi: 10.1109/ICCTD.2009.188

Venables, A., Tan, G., & Lister, R. (2009). A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. *ICER '09 Proceedings of the fifth international workshop on Computing education research workshop*. 117-128. doi: 10.1145/1584322.1584336

White, G. L., & Sivitanides, M. P. (2005). A Theory of the Relationships between Cognitive Requirements of Computer Programming Languages and Programmers' Cognitive Characteristics. *Journal of Information Systems Education*. 13(1). 59-66.

Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies – Best of empirical studies of programmers 7.* 51(1). 71-87. doi: 10.1006/ijhc.1999.0269

Vihavainen, A., & Luukkainen, M. (2013). Results from a Three-Year Transition to the Extreme Apprenticeship Method. *Proceedings of the IEEE 13th International Conference on Advanced Learning Technologies (ICALT)*. 336-340.

Vihavainen, A., Paksula, M., & Luukkainen, M. (2011). Extreme Apprenticeship Method in Teaching Programming for Beginners. *Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE'11)*. 93-98.

Williams, L., & Kessler, B. (2000). The Effects of 'Pair-Pressure' and 'Pair-Learning' on Software Engineering Education. *Proceedings of the 13th Conference on Software Engineering Education & Training (CSEET'00)*. 59-65.

Williams, L., & Kessler, B. (2001). Experimenting with Industry's 'Pair-Programming' Model in the Computer Science Classroom. *Computer Science Education*. 11(1). 7-20.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM*. 14(4). 221-227. doi: 10.1145/362575.362577