WWW.JAMK.FI

”SIINÄ ON SPEKSIT – MIKSET JO KOODAA?”

Ajatuksia ohjelmoinnin oppimisesta ja opettamisesta

Olli Vertanen

Kehittämishankeraportti
Joulukuu 2010

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Ammatillinen opettajakorkeakoulu

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t)	Julkaisun laji	Päivämäärä
VERTANEN, Olli	Kehittämishankeraportti	03.12.2010
	Sivumäärä 52	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
”SIINÄ ON SPEKSIT – MIKSET JO KOODAA?” Ajatuksia ohjelmoinnin oppimisesta ja opettamisesta

Koulutusohjelma
Ammatillinen opettajankoulutus/ammatillinen erityisopettajankoulutus/opinto-ohjaajankoulutus

Työn ohjaaja(t)
WEISSMANN, Kirsti

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyössä tutustuttiin ohjelmoinnin oppimisen ja opettamisen problematiikkaan. Työ tehtiin alan tutkimuksen ja julkaisujen pohjalta.

Työssä tuodaan esiin ensinnäkin yleisiä oppimiseen liittyviä asioita. Näitä ovat oppimisympäristö, opiskeluorientaatio, oppimistyyli, motivaatio, kognitiiviset toimintot ja muisti. Omana kokonaisuutenaan tuodaan esiin, mitä on ohjelmointitaito, mitä osa-alueita siihen kuuluu ja millaiset ovat tänä päivänä työlämävaatimukset IT-alalla. Työssä käsitellään myös ohjelmoinnin oppimisen ongelmia ja uusia opetusmenetelmiä mm. pariohjelmointia ja testivetoista kehitystä.

Avainsanat (asiasanat)
Ohjelmointi, ohjelmoinnin opettaminen

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s) VERTANEN, Olli	Type of publication Development project report	Date 03.12.2010
	Pages 52	Language Finnish
	Confidential ( ) Until	Permission for web publication (X)

Title ”HERE ARE THE SPECS – AREN’T YOU CODING ALREADY?” Some thoughts about leaning and teaching programming
Degree Programme Vocational Teacher Education
Tutor(s) WEISSMANN, Kirsti
Assigned by

Abstract

In this study we address the questions of learning and teaching programming. The study is made on the basis of recent publications from computer science education field.

First we take a look at what elements are involved in learning in general. These include for example learning environment, learning strategy, learning style, motivation, coginitive processes and human memory. These are partly individual properties of a learner, partly products of the social context where learners and teachers operate.

The structure and contents of programming discipline is explained, in addition to current qualification requirements of the profession. Finally, some common problems of novice learners are analyzed, and some modern teaching methods, like pair programming and test driven development, high-lighted.

Keywords

Programming, Teaching programming

Miscellaneous

SISÄLTÖ

1 JOHDANTO....3

2 OPPIMISEN EVÄÄT - MISSÄ, MITEN, KUKA?....5

2.1 Missä – oppiminen ja oppimisympäristö....5

2.2 Miten – oppimisorientaatio, lähestymistapa ja motivaatio....8

2.2.1 Lähestymistapa oppimiseen....8

2.2.2 Opiskeluorientaatio....11

2.2.3 Oppimistyyli....11

2.2.4 Motivaatio....13

2.3 Kuka – kognitiiviset toiminnot ja emootiot....13

2.3.1 Metakognitio....14

2.3.2 Muisti ja oppiminen....15

3 OHJELMOINTITAITO....18

3.1 Ohjelmoinnin osa-alueet....19

3.2 Ohjelmointitieto....22

3.3 Työelämävaatimukset ja ammatillinen osaaminen....23

4 OHJELMOINNIN OPPIMISESTA JA OPETTAMISESTA....27

4.1 Ohjelmoinnin oppimisen sietämätön raskaus....30

4.1.1 Erilaisia oppimisstrategioita....32

4.1.2 Ohjelmoijien oppimistyylit 33

4.2 Ohjelmoinnin oppimisympäristö....35

4.2.1 Opetusvälineiden valinnasta....35

4.2.2 Opetuskielen valinnasta 39

4.3 Ohjelmoinnin opetuksen ongelmia....40

4.4 Opetusmenetelmistä......41

4.4.1 Kognitiivinen oppipoikamalli......41

4.4.2 Testivetoinen kehitys......43

4.4.3 Pariohjelmointi......44

5 POHDINTA......48

LÄHTEET......50

KUVIOT

KUVIO 1. Dynaaminen oppimisympäristö (Rytkönen 2009, s 7)......6

KUVIO 2. Erilaiset lähestymistavat opiskeluun ja niiden sisältämät oppimissuuntautumiset (Lindblom-Ylänne ym. 2002, 122)......10

KUVIO 3. Kolbin oppimistyylit (Galpin yms. 2007)......12

KUVIO 4. Muistin monivarastomalli (Kalakoski 2007, 15)......16

KUVIO 5. Ohjelmointitehtävään liittyviä tietoja ja taitoja (Ala-Mutka 2005)......20

KUVIO 6. Ammattikorkeakouluosaaminen integroituna kokonaisuuteen (Raij, 2003). 24

KUVIO 7. Ohjelmointikäsitteiden oppiminen (Ala-Mutka, 2005)......29

KUVIO 8. Opiskelijoiden jakautuminen Kolbin oppimistyyleihin (Galpin 2007)......34

KUVIO 9. Ruudunkaappaus opetuskäyttöön suunnitellusta BlueJ-ohjelmankehitysympäristöstä......37

KUVIO 10. Ruudunkaappaus ammattikäyttöön soveltuvasta Eclipse-ohjelmankehitysympäristöstä......38

TAULUKOT

TAULUKKO 1. Erilaisten oppimisympäristöjen aiheuttamat jännitteet (Lindblom-Ylänne ym. 2002)......7

TAULUKKO 2. Ohjelmointikielet ja kognitiivinen kehitys/tyyli (White ym. 2005)......30

1 JOHDANTO

“Programming is an unnatural act”
Alan J. Perlis, 1982

Alunperin tarkoitukseni oli kehityshankkeessani tutkia pariohjelmointia pedagogisena menetelmänä ohjelmoinnin opetuksessa. Käytännön osuuden piti tapahtua opetus-harjoitteluni yhteydessä ja kehityshankeraportin piti syntyä jouhevasti opetusharjoittelun kokemuksen pohjalta. Matkalle sattui kuitenkin muutamia karikkoja. En osannut kokemattomuuttani varautua siihen, miten tarkkaan uusi opetus- ja oppimismenetelmä tulisi opiskelijoille ajaa sisään. Toiseksi, harjoitteluryhmässäni oli vain kuusi opiskelijaa, joista kaksi kieltäytyi järjestelmällisesti parityöskentelystä. Kahdesta parista ei juurikaan vertailevaa aineistoa saanut aikaan. Kolmanneksi, ehkä ensimmäiseen kohtaan kiinteästi liittyen, aktiivisetkin parit halusivat siirtyä itsenäiseen työskentelyyn kurssin puolivälissä – pareittain tekemällä ei kuulemma oppinut asioita riittävän hyvin. Olin lannistettu.

Olin saanut opettavakseni huonosti motivoituneen ryhmän eikä se mitenkään loistanut osaamisellaan. Olin opettanut ohjelmointia aiemminkin, kiinnittämättä isommin huomiota opetusmenetelmiin taikka opiskelijoiden erityispiirteisiin. Kurssit olivat onnistuneet joskus paremmin, joskus huonommin, mutta aina käyttäen yliopistossa jo 25 vuotta sitten omaksumaani kaavaa. Sehän on minulle *Se Oikea Tapa* opettaa ja oppia tietokoneen ohjelmointia. Toisaalta olin kyllä joskus ihmetellyt, miksi jotkut opiskelijat tuntuivat oppivan kaiken ilman ohjausta, ja toisiin ei ohjelmointi 'kolahtanut' kaikesta vaivannäöstäni huolimatta. Ehkä pitäisikin todeta, että lahjakkaimmat oppivat opetuksesta huolimatta.

Sysäyksenä tälle työlle oli kuitenkin tuo penkin alle luisunut opetusharjoitteluryhmä. Aloin tarkemmin pohtimaan, miten ohjelmointia pitäisi oikeasti opettaa? Onko oppimani tapa opettaa sittenkään se ainoa oikea kaikille tai edes kenellekään? Millaisia menetelmiä ohjelmoinnin opettamisessa tulisi käyttää? Pitääkö sitä lainkaan opettaa vai oppivatko opiskelijat aivan itsekseen? Ja miten ohjelmointia ylipäänsä opitaan? Jotain tutkimusta tästäkin aiheesta täytyisi olla olemassa! Kaipasin toisin sanoen aine-

didaktista täydennystä opettajankoulutuksen aiheisiin. Opettajakoulutuksen opinto-oppaan sanoin: “Kun opiskelija on alkamassa tai vasta alkanut opettajan työn, (kehittämis)hanke voi olla omaa pedagogista osaamista kehittävä tutkimuksellinen hanke”.

Toinen lähtökohta tälle työlle on tarkoitus toimia minulle itselleni tietovarastona ja muistina. Työskentelen teollisuuden palveluksessa ja ohjelmoinnin realiteetit ovat itselleni jokapäiväisiä. Siirtyminen opetustehtäviin tapahtuu sitten joskus – jos koskaan. Jos se joskus tapahtuu, toivon voivani kaivaa tämän raportin esille ja käyttää sitä muistin virkistämiseen, mistä opetustyössä olikaan kyse.

Pääotsikko työlle “Siinä on speksit – mikset jo koodaa?” (speksi, spesifikaatio eli määritelmä tai suunnitelma) on napattu noin kymmenen vuotta sitten Internetissä levinneestä räpistä “Koodaus on ihanaa”. Biisissä karroikoidaan humoristiseen sävyyn ohjelmoijan eli “koodaajan” arkipäivää. Otsikosta tuli lentävä lause monenkin projektin yhteydessä. Tämän työn otsikkoon se päätyi, koska mielestäni siinä korostetaan sitä valmiutta, jonka koulutuksen tulisi tuoreelle ammattilaiselle antaa. Työhön voi tarttua empimättä, jos on saanut oppilaitoksesta riittävät eväät matkaansa.

Tämä työ on tehty kirjallisuuden pohjalta. Kun lopulta aloitin tutkimaan, mitä ohjelmoinnin opettamisesta on kirjoitettu, edessä aukeni kokonaan uusi maailma. Tietojenkäsittelyn opetuksen tutkimus (Computer Science Education) on alana suhteellisen tuore, mutta varhaisimmat kirjoitukset löytyvät kuitenkin jo 1980-luvulta, ja ala kasvaa jatkuvasti. Tässä työssä esittelen joitain tuloksia, jotka tutkimusurakastani jäivät päällimmäiseksi ajatuksiin.

Luvun 2 nimesin “Oppimisen eväät” eli siinä käsittelen yleistä oppimiseen ja oppijaan liittyviä tekijöitä. Ohjelmoinnin opiskelijatkin ovat vain ihmisiä, ja toimivat sosiaalisessa kontekstissa. Opettajan on hyvä pitää mielessä, kuinka monet seikat oppimiseen lopulta vaikuttavat. Luvussa 3 keskityn pohtimaan, mitä on ohjelmointitaito. Mitä osa-alueita ohjelmointiin kuuluu ja millaisia taitoja ohjelmoijan tulisi hallita? Nämä pitäisi olla ohjelmoinnin opettajalla kirkkaana mielessä. Samoin kuin se, mihin työelämätodellisuuteen opiskelijoita valmennetaan. Luku 4 keskittyy lopulta ohjelmoinnin opettamisen ja oppimisen problematiikkaan. Raportin lopuksi pohdin, mitä yritinkään sanoa ja miksi.

2 OPPIMISEN EVÄÄT - MISSÄ, MITEN, KUKA?

"Most people find the concept of programming obvious,
but the doing impossible."
Alan J. Perlis, 1982

Oppiminen ei tapahdu umpiossa vaan aina jonkinlaisessa sosiaalisessa kontekstissa. Oppija – ihminen, astuu tähän kontekstiin mukanaan oma historiansa. Ja kohtaa toisia ihmisiä: opettajan ja opiskelijatovereitaan. Tämä luvussa otan esiin muutamia tähän problematiikkaan liittyviä asioita – missä, miten, kuka?

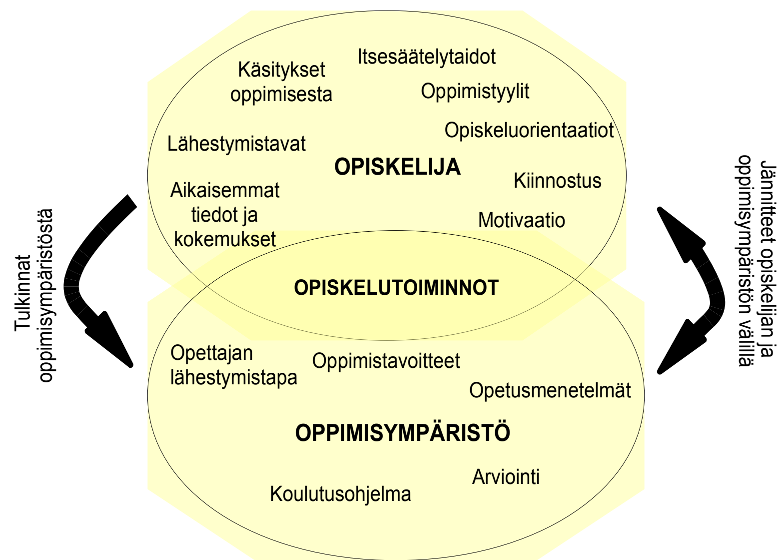
2.1 Missä – oppiminen ja oppimisympäristö

Kaikki oppiminen tapahtuu jossain oppimisympäristössä. Oppimisympäristöllä tarkoitetaan opettajan, opiskelijoiden, opiskeltavan aineen ja opetusmenetelmän muodostamaa kokonaisuutta (Lindblom-Ylänne ym. 2002, 54). Oppimisympäristö syntyy sosiaalisen ilmapiirin ja opetuksellisen lähestymistavan myötä. Fyysisen oppimisympäristön (esim. luokkahuone) lisäksi nykyisin virtuaaliset, verkko-opetukseen liittyvät oppimisympäristöt ovat yleisiä. Oppimisympäristön merkitystä on pyritty korostamaan kognitiivisen psykologian ja konstruktivistisen oppimiskäsityksen myötä. Opettajan tehtäväksi nähdään sopivan oppimisympäristön luominen opiskelijoille, jolloin opiskelijoilla on mahdollisuus vaikuttaa sekä opetuksen sisältöön ja opiskelumenetelmiin. Opiskelijakeskeisessä oppimisympäristössä opettaja ei tähtää suoraan tiedon jakamiseen, vaan aktivoimaan opiskelijoita tiedon rakentamiseen ja omien käsitysten muodostamiseen.

Oppimisympäristö on dynaaminen kokonaisuus, jonka kaikki osa-alueet vaikuttavat toisiinsa (KUVIO 1). Opettajan kontribuutio ympäristöön on hänen opetusmenetelmänsä ja -taitonsa, arviointimenetelmänsä ja käsityksensä opettamisesta ja oppimisesta. Oppimisympäristöön vaikuttaa myös oppilaitoksen kulttuuri, koulutusohjelma ja oppimiselle asetetut tavoitteet. Opiskelija tuo oppimisympäristöön mukanaan aikaisemmat tietonsa ja kokemuksensa, opiskelutaitonsa ja käsityksensä oppimisesta. Merkittävästi opiskeluympäristöön opiskelijan kannalta vaikuttaa hänen lähestymistapansa, motivaationsa, itsesäätelytaitonsa sekä opiskeluorientaationsa ja -strate-

giansa.

Opettajan ja opiskelijoiden tulkinnat oppimisympäristöstä ovat aina subjektiivisia ja voivat siten poiketa huomattavasti toisistaan. Esimerkiksi käsitykset opetussisällöstä, tavoitteista tai arvioinnista voivat olla erilaisia opiskelijoiden ja opettajan kesken.



KUVIO 1. Dynaaminen oppimisympäristö (Rytkönen 2009, s 7)

Opiskelija elää jatkuvassa vuorovaikutuksessa oppimisympäristönsä kanssa, ja näin hänen ja ympäristön kanssa syntyy väistämättä jännitteitä. Jännitteet saavat opiskelijan sopeutumaan ympäristöönsä, josta syystä ne voivat olla rakentavia tai tuhoisia. Rakentavat jännitteet saavat opiskelijan kehittämään opiskelutaitojaan sopeutukseen ympäristöönsä. Tuhoavat jännitteet puolestaan estävät opiskelijaa käyttämästä aikaisemmin oppimiaan taitojaan. Tuhoavien jännitteiden syntymisen voi estää opiskelijan jo valmiiksi kehittynyt oppimistyyli. Hän jatkaa opiskelua omalla tyylillään oppimisympäristöstä huolimatta. Opiskelija on tällöin immuuni oppimisympäristönsä vaikutuksille.

Se, johtavatko jännitteet opiskelijan kannalta uusien taitojen kehittymiseen, vaiko peräti taitojen suhteen taantumaan, riippuu sekä opiskelijan omista itsesäätelytaidoista, ja myös siitä, miten oppimisympäristössä oppimista säädellään (TAULUKKO 1). Heikoilla itsesäätelytaidoilla varustettu opiskelija kehittyy parhaiten, jos oppimisen säätely on pääasiassa opettajan vastuulla (opettajakeskeinen oppimisympäristö). Toisaalta hyvät itsesäätelyvalmiudet omaava opiskelija menestyy parhaiten, jos hän saa itse aktiivisesti osallistua oppimisympäristönsä kehittämiseen, oppimistavoitteiden asettamiseen ja tavoitteiden saavuttamisen seuraamiseen (opiskelijakeskeinen oppimisympäristö). Opettajakeskeiseen oppimisympäristöön sopeutuminen saattaa tällaisella opiskelijalla johtaa taantumaan. Opiskelijoiden kokemukset oppimisympäristöstä ovat siis voimakkaasti yhteydessä oppimiseen. Esimerkiksi opiskelijat säätelevät usein oppimistaan sen mukaan, millaisiksi he kokevat opettajan odotukset oppimisesta (Rytkönen 2009, 21). Tämä korostaa arvioinnin merkitystä oppimisessa.

TAULUKKO 1. Erilaisten oppimisympäristöjen aiheuttamat jännitteet (Lindblom-Ylänne ym. 2002)

Opiskelijan itsesäätelytaidot	**Oppimisen säätelyn vastuu**		
	opettajalla	**jaettu**	**opiskelijalla**
Hyvät	Tuhoava jännite	Jännite vaihtelee	Lievä rakentava jännite
Kohtalaiset	Tuhoava jännite	Lievä rakentava jännite	Rakentava jännite
Heikot	Yhteensopivuus	Rakentava jännite	Tuhoava jännite

Opettajan kannalta tilanne ei ole helppo, koska jännitteiden syntyminen opiskelijan ja ympäristön välillä on yksilöllistä, eikä niiden syntymistä voi välttää opettipa millä tyylillä tahansa. Opettajan tulisi toisaalta olla hyvin tietoinen omista käsityksistään opettamisesta ja oppimisesta, ja toisaalta oppia tunnistamaan opiskelijansa ja heidän valmiutensa opiskelun itsesäätelyyn. Tällä tavoin voidaan ehkäistä tuhoavien jännitteiden syntyä ja törmäyskurssille joutumista opiskelijoiden kanssa.

Opettajan roolin hyvässä oppimisympäristössä voisi kiteyttää (Rytkönen 2009, 21)

1. Opettajan pitää määritellä keskeiset asiat opetettavasta aineksesta ja suunnitella opetus niiden ympärille.
2. Kurssin tavoitteet täytyy tehdä selkeäksi opiskelijoille
3. Opiskelijan oppimista tulee tukea myös toimintaa vaativilla tehtävillä, jotka edellyttävät kokonaiskuvan ymmärrystä.
4. Opettajan tulee tukea opiskeluprosessia jatkuvan, rakentavan arvioinnin ja palautteen avulla, jotka ovat yhtenevät tavoitteiden kanssa.

2.2 Miten – oppimisorientaatio, lähestymistapa ja motivaatio

Edellisessä luvussa todettiin, että oppiminen tapahtuu aina jossain oppimisympäristössä, jonka muovautumiseen opettajalla ja hänen oppimiskäsityksellään on suuri merkitys. Opettajan kannalta on toisaalta erittäinkin tähdellistä ymmärtää opiskelua opiskelijan lähtökohdista. Nykyisin painotetaan sanaa oppiminen opettamisen sijaan. Opiskelijakin on muuttunut oppijaksi. Mutta mistä osatekijöistä oppiminen koostuu? Miten ja miksi oppija oppii?

Kysymykseen tuskin löytyy täydellistä vastausta sen monitasoisuuden ja -vivahteisuuden takia. Seuraavassa tuon esille joitain ongelmaan liittyviä näkökohtia.

2.2.1 Lähestymistapa oppimiseen

Lähestymistavalla oppimiseen tarkoitetaan tapaa, miten opiskelija kokee, ymmärtää ja tulkitsee oppimistehtävät (Lindblom-Ylänne ym. 2002, 54). Laadullisesti lähestymistavat on jaettavissa kahteen pääryhmään: pinta- ja syväsuuntautuneet lähestymistavat.

Pintasuuntautunut opiskelija keskittyy lähinnä kurssivaatimuksista suoriutumiseen. Opiskelussa pääpaino on opiskeltavan materiaalin toistamisella. Opiskelijalla on oma käsityksensä, minkälaista osaamista heiltä edellytetään, jolloin pintasuuntautuneen opiskelijan tarkoitus on keskittyä tehtävän vaatimuksiin ymmärtämisen sijaan (Rytkönen 2009, 16). Pintasuuntautuneisuus näkyy esim. asioiden listaamisena argumentaation sijaan.

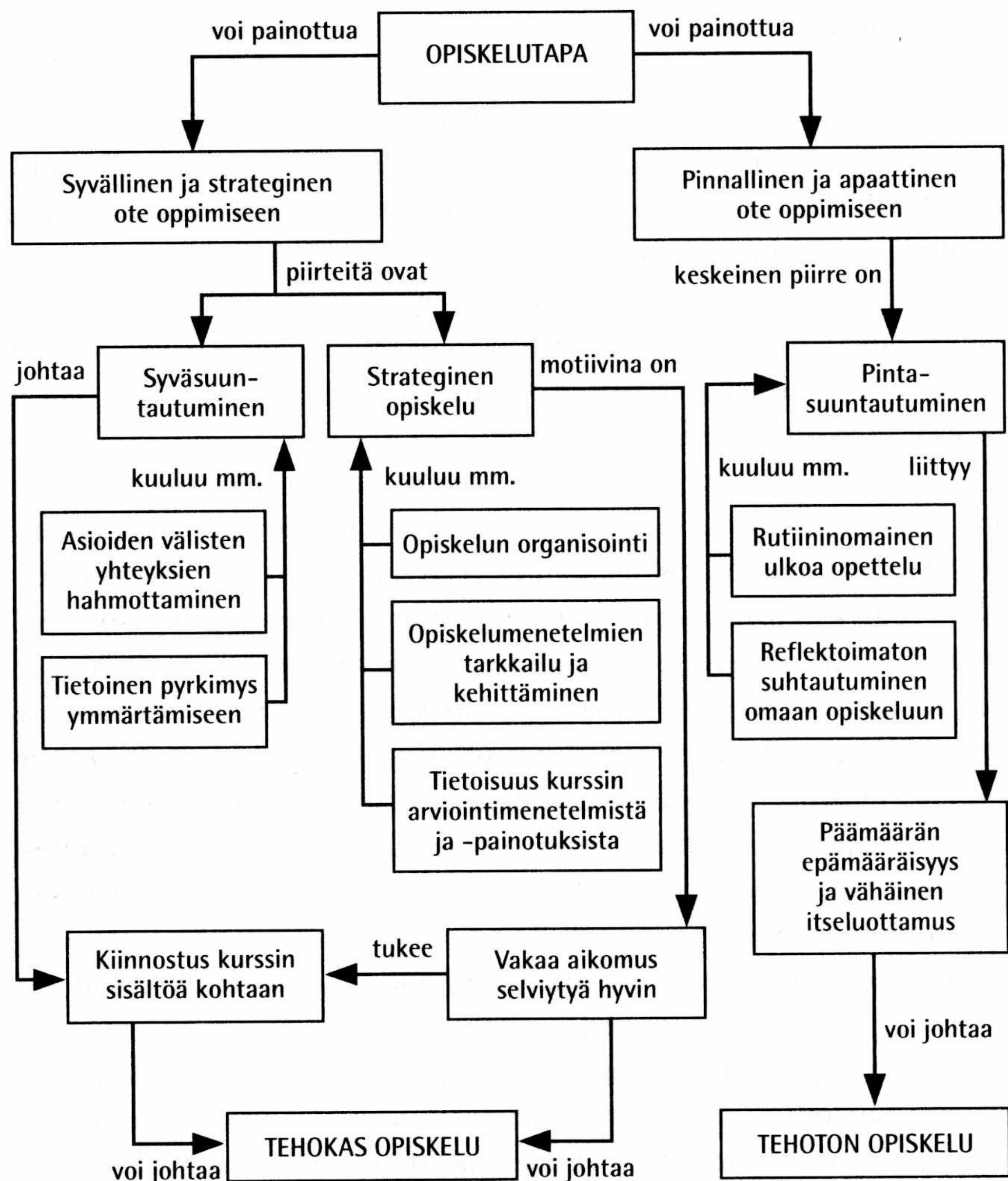
Syväsuuntautuneet opiskelijat pyrkivät puolestaan ensisijaisesti ymmärtämään opiskeltavan materiaalin. Oppimista ohjaa yleensä sisäinen kiinnostus opittavaan asiaan. Syväsuuntautunut opiskelija pyrkii käyttämään aikaisempaa tietoaan aiheesta, ymmärtämään asioiden kokonaiskuvan ja eri tekijöiden väliset yhteydet (holistinen strategia). Syväsuuntautuneisuuteen kuuluu myös yksityiskohtiin perehtyminen (serialistinen strategia). Tämän perusteella ei liene suuri yllätys, että syväsuuntautunut lähestymistapa on yhdistetty laadullisesti parempiin oppimistuloksiin kuin pintasuuntautunut.

Tutkimuksissa on löydetty myös lähestymistapoihin toinen ulottuvuus, strateginen lähestymistapa. Siinä opiskelijan painopiste on opiskelun organisoinnissa tähtäimenä mahdollisimman hyvä opintomenestys. Kurssin arviointi ohjaa merkittävästi strategisesti suuntautuneen opiskelijan työpanosta. Strateginen lähestymistapa liittyy enemmän opiskelijan itsesäätelyyn kuin itse oppimiseen. Siten se yhdistyy joko syvä- tai pintasuuntautuneen lähestymistavan kanssa - opiskelijan motivaatiosta riippuen. Tästä syystä lähestymistavasta käytetään myös nimityksiä systemaattinen tai tavoitteellinen lähestymistapa. Tämän lähestymistavan omaksuneet opiskelijat ovat taitavia ajankäyttönsä suhteen, tietoisia arviointikriteereistä ja vaatimuksista ja osaavat seurata omaa kehitystään.

Huomionarvoista strategisessa lähestymistavassa on myös, että sen on todettu useimminten olevan yhteydessä syväsuuntautuneeseen lähestymistapaan, oppimisen sisäiseen säätelyyn ja tiedon omaehtoiseen rakentamiseen. Toisin kuin pintasuuntautunut lähestyminen yhdistetään ulkoiseen säätelyyn, itsesäätelyn ongelmiin ja käsitykseen oppisena tiedon tallettamisena. Määräävät piirteet näistä kolmesta lähestymistavasta on koottu kuvion 2 käsitekarttaan.

Huomattavaa opettajan ja oppiaineen kannalta on, että oppimisen lähestymistapa on vahvasti kontekstispesifinen. Tämä tarkoittaa, että se on riippuvainen niin opiskeltavasta sisällöstä, tieteenalasta kuin oppimisympäristöstäkin. Intuitiivisesti tämä on hyvin selkeä tulos: jokaiseen opetusohjelmaan mahtuu opiskelijan kannalta 'heittopussiaineita', jotka suoritetaan vain opintopisteiden takia, todellisen mielenkiinnon kohdistuessa muihin aineryhmiin. Myös opiskelijoiden iän on todettu vaikuttavan suuntautuneisuuteen – vanhemmat opiskelijat ovat nuoria useammin syväsuuntau-

tuneita (Rytkönen 2009, 19). Syväsuuntautuneet oppimisprosessit tulisikin määritellä jokaisella ammattialalla erikseen (Lindblom-Ylänne ym. 2002, 121), jotta varmistetaan kunkin opiskelualan käsitteellinen ymmärtäminen.



KUVIO 2. Erilaiset lähestymistavat opiskeluun ja niiden sisältämät oppimissuuntautumiset (Lindblom-Ylänne ym. 2002, 122)

2.2.2 Opiskeluorientaatio

Kun lähestymistapa opiskeluun määriteltiin opiskelukontekstista riippuvaksi, niin opiskeluorientaatiolla viitataan enemmän pysyviin, opiskelijan luonteeseen liittyviin mieltymyksiin valita tietty opiskelutapa. Opiskeluorientaatio on siten taipumus omaksua tietty lähestymistapa opiskeluun kontekstista riippumatta. Lähestymistavan lisäksi opiskeluorientaatioon liittyy myös oppimistyyli, motivaatio ja itsesäätelytaidot, joita käsittelen seuraavissa luvuissa.

Esimerkkinä erilaisista orientaatioista on yliopisto-opiskelijoilla tehty tutkimus (Lindblom-Ylänne ym. 2002, 121), jossa opiskelijat jaettiin orientaationsa mukaan neljään ryhmään. Ryhmittely kuvaa myös syitä, joiden takia opiskelijat ovat alansa valinneet. Akateemisesti orientoituneet olivat kiinnostuneet ensi sijassa tutkimuksesta, ammatillisesti orientoituneiden tavoite oli saada ammatti ja työpaikka. Omaan kehitykseensä orientoituneet olivat ensi sijassa kiinnostuneet omien tietojen ja taitojen kehittämisestä. Sosiaalisesti orientoituneiden kiinnostus oli sen sijaan enemmän opiskelijaelämän kuin opiskelun puolella.

Orientaatio vaikutti selvästi opintomenestykseen siten, että ammatillisesti orientoituneet menestyivät parhaiten, ja sosiaalisesti orientoituneet heikoiten. Ammatillisesti orientoituneilla oli myös selkeästi paras opiskelumotivaatio, olivat selvästi muita eteenpäinpyrkivämpiä ja kilpailuhaluisimpia.

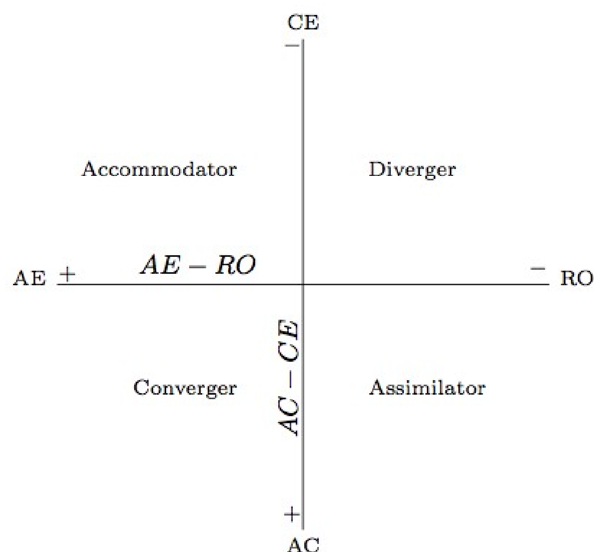
2.2.3 Oppimistyyli

Oppimistyyli on jossain määrin pysyvä, persoonallisuuteen liittyvä ominaisuus. Käsitteenä se on kaikkea muuta kuin yksinkertainen, ja sitä onkin määritelty ja mitattu eri tavoin. Otan tässä esiin tunnetun Kolbin oppimistyylien nelijaon, koska se on varsin usein viitattu, ja sitä on myös käytetty myöhemmin viitattavissa tutkimuksissa (Galpin yms. 2007).

Kolb erotti oppimisesta kaksi ulottuvuutta (KUVIO 3). Toinen ulottuvuus liittyy ajattelun ja tunteiden suhteeseen, ja toinen reflektoinnin ja toiminnan väliseen suhteeseen. Ensimmäisen dimension ääripäät ovat abstrakti käsitteellistäminen (abstract conceptualization, AC) ja välitön kokeminen (concrete experience, CE), ja toisen vastaavasti reflektoiva havaitseminen (reflective observation, RO) ja aktiivinen kokei-

leminen (active experimentation, AE). Näistä kahdesta ulottuvuudesta saadaan aikaan nelikenttä. Kaikkien ihmisten oppimisprosessiin sisältyy jonkin verran kaikkia em. aspekteja, ja niiden suhde vaikuttaa siihen, mihin kohtaan nelikentässä oppijan oppimistyyli sijoittuu (ks. esimerkki sivu 34). Nelikentän osat jaetaan karkeasti:

1. Päättelijä (Converger): Painottaa ajattelua ja toimintaa. Vahvuudet ovat ongelmanratkaisussa, päätöksenteossa ja teknisesti suuntautuneissa aloissa.
2. Tarkkailija (Assimilator): Painottaa ajattelua ja reflektointia. Fokus on mallintamisessa ja informaation käsittelyssä eikä niinkään ihmisissä.
3. Osallistuja (Diverger): Painottaa tunteita ja reflektointia. Tämä oppimistyyli assosioituu mielikuvitukseen, uusiin ideoihin, ja on ihmisläheinen.
4. Toteuttaja (Accomodator): Painottaa tunteita ja toimintaa, sopeutuu hyvin muuttuviin tilanteisiin ja käyttää ihmisiä tiedon lähteinä.



KUVIO 3. Kolbin oppimistyylit (Galpin yms. 2007)

2.2.4 Motivaatio

Motivaatio tarkoittaa kiinnostusta, tarkoitusta tai vaikuttajaa, joka suuntaa ja ohjaa opiskelijan opiskelua (Rytkönen 2009, s 9). Motivaatiotekijät jaetaan yleisesti sisäisiin ja ulkoisiin. Sisäisesti motivoitunut opiskelija on aidosti kiinnostunut opiskeltavasta asiasta ja pyrkii sen ymmärtämiseen. Ulkoisesti motivoitunut opiskelija on lähinnä kiinnostunut opiskelun tuottamasta palkinnosta, joka voi olla opintopisteet tai mahdollisuus työpaikan saamiseen. Ihanteellinen opiskelumotivaatio on sisäisen ja ulkoisen motivaation yhdistelmä (Lindblom-Ylänne ym. 2002, 131).

Lindblon-Ylänteen ym. mukaan motivaatio on hyvän opetuksen seuraus – ei sen edellytys. Opettajalla on merkittävä rooli kiinnostuksen herättämisessä opiskeltavaa ainetta kohtaan. Kiinnostus on tärkeä osa menestyksekästä opiskelua, ja opettaja voi toimia motivaation herättäjänä. Kiven mukaan (2000, 42) tehtävänsisäisen opiskelumotivaation herättäminen opiskelijoissa on opettajan ammattitaidon keskeinen tehtävä. Opiskelijalla itsellään on toisaalta avainasema opiskelumotivaation ylläpidossa. Motivaatiolla on suora vaikutus opiskelijan tavoitteisiin, oppimisen laatuun ja itsesäätelyyn.

Tahtotekijöllä on keskeinen osa oppimaan motivoitumisessa ja oppimisen säätelyssä (Kivi 2000, 41). Tahto auttaa opiskelijaa suuntaamaan motivaatioitaan ja myös emootioitaan, ja hallitsemaan häiritseviä tunteita. Jotta motivaatio- ja tahtotekijöiden kontrollointi onnistuu, opiskelija tarvitsee hyvän käsityksen itsestään oppijana. Motivoitumisen yksi keskeisiä tekijöitä on myös, että opiskelija kokee itsensä hyväksi oppijaksi.

2.3 Kuka – kognitiiviset toiminnot ja emootiot

Ihmisen kognitiivisilla toiminnoilla tarkoitetaan tiedonkäsittelyyn liittyviä prosesseja – tiedon vastaanottamista, käsittelyä ja käyttöä. Nämä toiminnot ovat yhteydessä havaitsemiseen, tunnistamiseen, kielen tuottamiseen ja ymmärtämiseen, ajatteluun, ongelmanratkaisuun ja muistamiseen. Kognitiiviset toiminnot liittyvät siihen, miten yksilö mieltää ympäröivän todellisuutensa, ja miten hän on ympäristönsä kanssa vuorovaikutuksessa. Kognitiiviset toiminnot liittyvät myös vahvasti oppimiseen.

Affektiiviset toiminnot liittyvät tunteisiin eli emootioihin. Kognitiiviset ja affektiiviset alueet ovat ihmisen kokonaispersoonassa läheisessä keskinäisessä vuorovaikutuksessa. Esimerkiksi pelko, ahdistus ja jännitystila estävät oppimista. Toisaalta taas myönteiset kokemukset oppimistilanteissa parantavat opiskelijan käsitystä itsestään oppijana ja vahvistavat itseluottamusta.

2.3.1 Metakognitio

Kognitiiviset taidot ovat avainasemassa oppimisen suhteen, ja niiden riittämätön hallinta saattaa johtaa epäonnistumiseen oppimisessa. Voiko sitten kognitiivisia prosessejaan voi parantaa?

Metakognitio on yksilön tietoisuutta omista kognitiivisista prosesseistaan, 'kognitiota kognitiosta' (lat. cognoscere = tietää, tunnistaa), kykyä ohjata, säädellä ja arvioida omaa ajatteluaan (Kivi 2000, 36). Metakognitiossa on kyse aktiivisesta oman toiminnan tarkkailusta jonkin tehtävän suorittamisen aikana ja tarkoituksena on yleensä jonkin päämäärän saavuttaminen. Metakognitiota voidaan jossain mielessä kutsua myös toiminnan ohjaukseksi.

Niistä kontrollistrategioista, joita ihminen käyttää tietoiseen tiedonkäsittelyn ja ongelmatilanteiden ratkaisun ohjaukseen, käytetään yhteisnimitystä metakognitiot. Nämä tunnistavat, valitsevat, analysoivat ja hylkäävät ratkaisuun johtavia strategioita. Strategiavarasto myös täydentyy ajan ja kokemuksen myötä.

Omien metakognitioiden tunnistaminen on vaikeaa. Kuitenkin hyvin yksinkertainen tapa seurata ajattelun ja metakognitiivisen toiminnan etenemistä työskentelyn aikana on ajatella ääneen (Kivi 2000, 37). Ääneen puhuminen on kokeissa osoittautunut hyvin tehokkaaksi ja hyödylliseksi. Usein pelkästään sanat ’kyllä’ tai ’ei’ riittävät. Useimmilla meistä on myös omakohtaisia kokemuksia ääneen ajattelun hyödyistä.

Vielä alakouluikäisille tämä menettely on luonnollista, mutta katoaa jostain syystä useimmilla myöhemmässä vaiheessa. Ääneen ajattelua voidaan pitää metakognitioiden kehittymisen esimuotona. Opetustilanteessa opettaja voi pyytää oppilaita sanelemaan ylös ratkaisuprosessinsa, kirjaamaan sen ylös tai rohkaisemaan ratkaisijaa puhumaan samanaikaisesti muille opiskelijoille ratkaisuprosessin edetessä. Erilaisia ratkaisumalleja voi myös verrata oppilaiden kesken. Metakognitiivisen

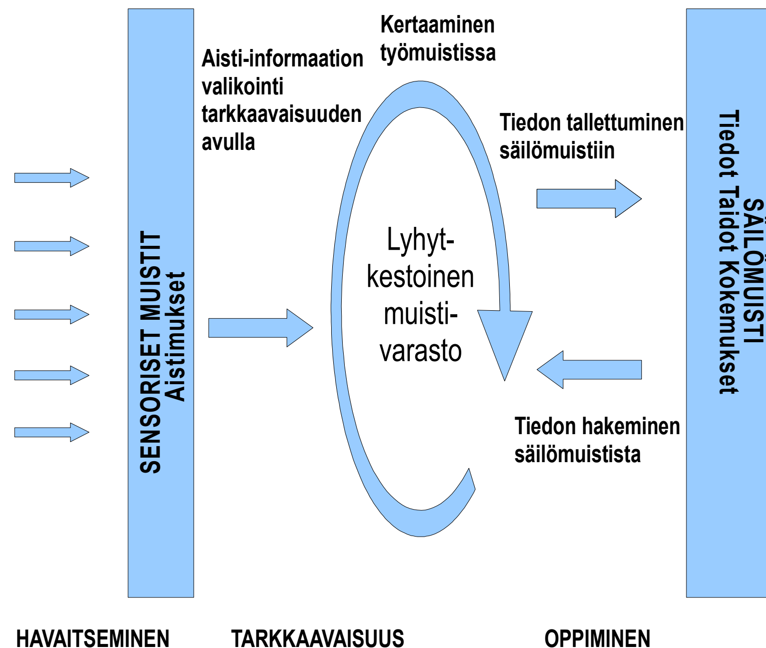
prosessin ymmärtäminen on tärkeää ja opettaa oppijaa kontrolloimaan oppimistaan ja itsesäätelytaitojaan.

2.3.2 Muisti ja oppiminen

Naivisti käsitettynä oppiminen on asioiden ulkoa muistamista. Sitähän monesti koulussakin harrastetaan. Oppimista on kuitenkin monenlaista ja muisti on tärkeä osa kaikenlaista oppimista.

Nykyisin vallalla olevan käsityksen mukaan ihmisen muisti koostuu useista kestoltaan ja kapasiteetiltaan erilaisista varastoista (ns. monivarastomalli). Keskeisin jako on lyhytkestoiseen työmuistijärjestelmään ja pitkäkestoiseen säilömuistijärjestelmään (KUVIO 4) (Kalakoski, 2007). Havaintotoiminta on tiedonkäsittelyn lähtökohta. Ihminen on jatkuvasti valveillaoloaikanaan aistiärsykkeiden pommituksen kohteena. Sensoriset muistit keräävät havaintoinformaatioita ja tallettavat sen sekunnin murto-osien ajaksi. Osa tästä informaatiosta, henkilön kulloisenkin tarkkaavaisuuden kohteen ja tason mukaan, siirtyy työmuistiin. Riittävän työmuistissa kertaamisen avulla informaatio saadaan siirrettyä säilömuistiin.

Työmuisti on ainoa muistivarasto, jota ihminen pystyy tietoisesti tarkkailemaan. Työmuistille on olennaista lyhytkestoisuus ja rajallinen koko. Tieto säilyy työmuistissa kertaamatta n. 20 sekuntia ja muistiin mahtuu kaikkiaan n. 5-9 tietoyksikköä. Koska työmuisti samalla myös muokkaa, kertaa, järjestää ja vertaa tietoja, on käytännössä sinne kerralla mahtuvien asioiden määrä n. 2-4. Työmuisti on kuitenkin keskeisessä roolissa tehtäviä suoritettaessa, jolloin keskeiseksi eroksi yksilöiden välillä nousee muistin käytön strategiat. Tiedon ylläpito työmuistissa palvelee vaativia kognitiivisia toimintoja kuten ajattelua mielikuvien avulla, ongelmanratkaisua ja päätöksentekoa.



KUVIO 4. Muistin monivarastomalli (Kalakoski 2007, 15)

Oppimisen kannalta merkittävä on työmuistin ongelmanratkaisun aikana kohdistuva kuorma. Tätä kutsutaan kognitiiviseksi kuormaksi. Kognitiivisen kuormittavuuden teorian mukaan oppiminen on tehokkainta kun työmuisti on optimaalisessa käytössä tiedon varastoinnin kannalta (Caspersen ym. 2007). Liian vähäinen kuormittavuus, esim. rutiinitehtävät, eivät johda uuden oppimiseen, mutta toisaalta liian vaikeat tehtävät eivät enää mahdollista työmuistissa olevan tiedon kognitiivista prosessointia. Työmuisti kuormittuu liikaa eikä oppimista tapahdu.

Työmuistin kuormitukseen liittyy termi kenttäsidonnaisuus (field dependency). Henkilöä voidaan pitää kenttäriippumattomana, jos hän pystyy erottelemaan havaintokentästään juuri haluamansa signaalit havaintokentän epärelevantista informaatiosta huolimatta (Mancy ym. 2004). Erittely on välttämätöntä, jotta informaatioita voidaan uudelleenjärjestellä ja muokata. Kenttäsidonnaiselle henkilölle taasen erilaisten informaation erittely kentästä on vaikeaa.

Toisin sanoen, kenttäriippumaton henkilö pystyy hyödyntämään työmuistiaan järkevämmin ja kenttäriippuvallla henkilöllä työmuisti helpommin ylikuormittuu liiasta informaatiosta. Kuitenkin on huomattava, ettö kenttäsidonnaisuus ja -riippumattomuus ovat liukuvan asteikon ääripäät. Asteikon keskivaiheille sijoittuvia sanotaan kenttäneutraaleiksi.

Säilömuisti on ihmisellä käytännössä rajaton. Säilömuistista voidaan erotella eri osa-alueita esim. deklaratiivinen asiamuisti, proseduraalinen taitomuisti, semanttinen tietomuisti ja episodinen tapahtumamuisti (Kalakoski 2007, 97). Oppisen kannalta merkittävää on miten tietoa saadaan talletettua säilömuistiin ja myös miten säilössä olevat tieto voidaan palauttaa. Muistaminenhan on varsinaisesti talletetun tiedon palauttamista työmuistiin.

Tieto jäsentyy säilömuistissa merkityksensä mukaan erilaisina mentaalimalleina eli skeemoina. Oppiminen on sitä, että näitä sisäisiä malleja kehitetään ja kasvatetaan jatkuvasti laadullisesti paremmiksi (Linblom-Ylänne ym. 2002). Tämä tapahtuu aktiivisesti rakentamalla ja prosessoimalla. Tässä prosessissa säilömuistissa olevia malleja integroidaan työmuistissa prosessoituun tietoon (schema acquisition) . Merkityksen analysoiminen asioista on oppimisen kannalta paljon tehokkaampaa kuin pelkkä toistaminen (Kalakoski 2007, 153), ja tällöin oppiminen voi tapahtua tahattomasti. Kaikille opettajille on myös tuttua, että itse tuotettu materiaali jää paremmin mieleen kuin toisten tekemän materiaalin toistaminen.

3 OHJELMOINTITAITO

"To understand a program you must become
both the machine and the program."
Alan J. Perlis, 1982

Ohjelmointi on, lyhyesti määriteltynä, erilaisten tietojenkäsittelyongelmien ratkaisemista algoritmisesti ja näiden ratkaisujen toteuttamista jollakin ohjelmointikielellä tietokoneen ymmärtämään muotoon. Ohjelmointiin liittyy siis toisaalta abstrakti ongelmanratkaisuprosessi ja toisaalta käytännöllisempi ratkaisun toteuttamiseen liittyvä aspekti. Ohjelmointi on keskeinen taito monella tietojenkäsittelyn osa-alueella.

ACM:n (Association of Computer Machinery) ja IEEE Computing Societyn yhdessä julkaisem suositus tietojenkäsittelyalojen opetussuunnitelmiksi (Shackelford ym. 2005) ovat kansainvälisesti laajassa käytössä ja monet tietojenkäsittelyn opetusohjelmat pohjaavat opetussuunnitelmansa niihin. Suositukset kattavat viisi koulutusalaa, jotka ovat tietotekniikka (computer engineering), tietojenkäsittelytiede (computer science), ohjelmistotuotanto (software engineering), informaatioteknologia (information technology) ja tietojärjestelmätiede (information systems).

Ohjelmoinnilla on kullakin koulutusalalla tärkeä rooli. Yllä mainittu suositus listaa 40 keskeisintä tietojenkäsittelyn osaamisaluetta ja arvioi kunkin alueen suhteellisen tärkeyden kullakin koulutusalalla. Ohjelmointi on ainoa alue, joka painoarvo on jokaisella koulutusalalla keskimäärin 3 tai enemmän (asteikolla 0-5). Kaikkien osaamisalueiden keskiarvo ohjelmoinnin osalta oli korkein (3,9). Ohjelmoinnin perustaidot on listattu myös yhdeksi kaikkia em. koulutusaloja selkeimmi yhdistäväksi tekijäksi. Ohjelmointitaito on myös edellytys usean muun osa-alueen omaksumiselle.

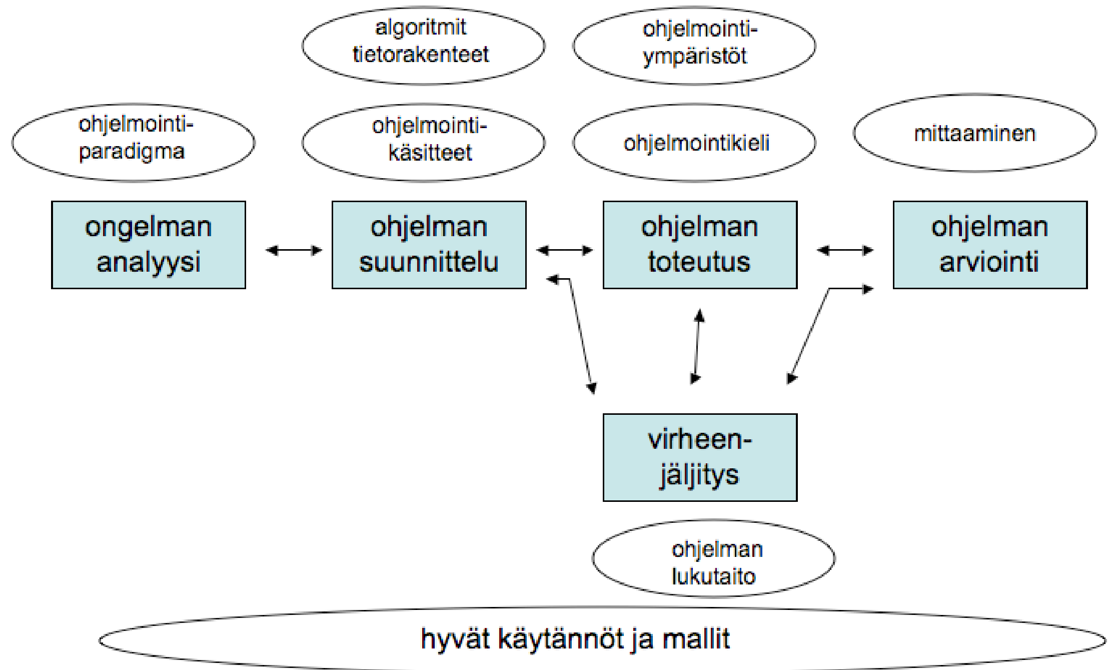
Opetuksen kannalta ensisijaisen tärkeää on miettiä, mitä ohjelmointi itse asiassa on ja millaisista osa-alueista ohjelmointitaito koostuu? Ilman tällaista analyysiä koulutusohjelmaan on vaikea laatia laadukasta ohjelmoinnin opetussuunnitelmaa, tai yksittäisen kurssin tavoitteet eivät välttämättä vastaa käytännön työelämän tarpeita. Seuraavassa tarkastelenkin lyhyesti, mitä ohjelmointitaito pitää sisällään.

3.1 Ohjelmoinnin osa-alueet

Kuviossa 5 on esitetty ohjelmointitehtävään liittyviä vaiheita ja niihin liittyviä tietoja ja taitoja (Ala-Mutka 2005). Vaiheet on esitetty suorakaitein ja tiedot ja taidot soikioina. Ohjelmointitehtävä jaetaan yleensä analyysi-, suunnittelu-, toteutus- ja testausvaiheisiin. Testauksen tarkoitus on löytää ohjelmointivirheet mahdollisimman tarkkaan. Havaitut virheet on korjattava. Tämä saattaa muuttaa ohjelman rakennetta vielä merkittävästi. Usein toteutus etenee iteratiivisesti siten, että testauksesta palataan virheenjäljityksen kautta toteutukseen tai jopa uudelleensuunnitteluun. Eri vaiheet pitävät sisällään monenlaisia ohjelmoijalta vaadittavia taitoja. Toisaalta kehitys on nykyään yleensä inkrementaalista, jolloin jo toteutettuja ominaisuuksia laajennetaan ja ohjelmaan lisätään uusia ominaisuuksia.

Ohjelmointikieli on väline, jolla ohjelmat kirjoitetaan. Yleensä ohjelmoinnista puhuttaessa ensimmäisenä viitataan juuri ohjelmointikieleen – 'millä kielellä ohjelma on toteutettu?'. Myös ohjelmointikurssit nimetään yleensä sen mukaan, mitä kieltä kurssilla käytetään tai opetellaan. Kielen hallintaan liittyy ensinnäkin sen syntaksi, eli kielen muotosäännöt. Syntaksi kertoo, miten millaisia tunnuksia, lauseita ja symboleja kielessä käytetään, ja mikä on kielen lauseiden oikea kirjoitusasu. Toinen tärkeä kielen aspekti on kielen semantiikka, eli mitä lauseet ja symbolit merkitsevät.

Kieli on kuitenkin vain ohjelman ulkoinen esitysmuoto – keino ohjelman toteuttamiseen. Kielen syntaksin ja semantiikan hallinta ei vielä takaa käytännön ohjelmointitaitoa, vaikka ovatkin välttämättömiä osa-alueita. Huomattavasti oleellisempaa on tietää, miten kieltä ja sen rakenteita käytetään erilaisten ongelmatilanteiden ratkaisuun. Tämä miten-kysymys liittyy toisaalta algoritmien ja tietorakenteiden tuntemukseen, toisaalta algoritmiseen ajatteluun ja ongelmanratkaisutaitoon. Tietorakenteilla tarkoitetaan tietojenkäsittelyn yhteydessä sitä, miten tietoalkiot järjestetään tietokoneen muistiin, ja algoritmeilla sitä, miten tietoalkoita muistissa muokataan.



KUVIO 5. Ohjelmointitehtävään liittyviä tietoja ja taitoja (Ala-Mutka 2005)

Käytännössä jokainen tietotekniikan opetussuunnitelma sisältää erillisen algoritmeja ja tietorakenteita käsittelevän erikoiskurssin. Algoritmista ajattelua ja ongelmanratkaisua sen sijaan ei opeteta erikseen vaan se sisältyy sivujuonteena jokaiselle ohjelmointia käsittelevälle kurssille. Tärkeää on ymmärtää algoritmien ja tietorakenteiden ohjelmointikielestä riippumaton universaali luonne.

Ohjelmoijan on hallittava myös suuri joukko yleisiä, kielestä riippumattomia ohjelmointikäsitteitä ohjelman suunnitteluun ja jäsentämiseen liittyen. Esimerkiksi miten ongelma (ja sitä kautta ohjelma) kannattaa pilkkoa pienempiin komponentteihin, miten komponentit vaihtavat tietoa keskenään, ja mikä on tehtyjen ratkaisujen vaikutus ohjelman suorituskykyyn, algoritmien aika- ja muistivaatimukset jne. Osa käsitteistä liittyy valittuun ohjelmointiparadigmaan. Paradigmalla tarkoitetaan fundamentaalia ohjelmointityyliä, jota ohjelman toteutus noudattaa. Pääparadigmoina yleisesti pidetään imperatiivista ohjelmointia, oliosuuntautunutta ohjelmointia, funktionaalista ohjelmointia ja logiikkaohjelmointia. Paradigman valinta ohjelman toteutuksessa on tehtävä hyvin varhaisessa vaiheessa ja sen vaikutus toteutukseen on hyvin merkittävä. Paradigmasta toiseen siirtyminen kesken toteutuksen vaatii suunnitteluprosessin uudelleenaloittamista. Ohjelmointikieli on usein sidoksissa johonkin

paradigmaan, joten valinta tehdään usein implisiittisesti ohjelmointikieltä valittaessa. Myös moniparadigmakieliä on olemassa.

Ohjelma ei ole suinkaan valmis, kun se on saatu kirjoitettua ohjelmointikielellä koneelle. Ohjelmoijan tulee vielä varmistua, että ohjelma toimii toivotulla tavalla. Tämä tarkoittaa ohjelman testaamista ja suorituskyvyn mittaamista. Näihin liittyy omat haasteensa. Miten suunnitellaan riittävän kattavat testitapaukset, jotta ohjelman kaikki polut saadaan testattua? Miten testit toteutetaan? Virheenjäljitysvaiheessa vaaditaan hyvää ymmärrystä, miten ohjelma lopulta toimii. Silloin korostuu ohjelman lukutaito hyvien tehokkaiden virheenjäljitysstrategioiden lisäksi.

Ohjelmointiympäristöllä tarkoitetaan työkaluja, joilla ohjelma kirjoitetaan, käännetään ja sovitetaan ajoympäristöönsä. Virheenjäljitykseen (engl. debugging) on olemassa omat työkalunsa ns. debuggerit. Nykyisin käytetään pääsääntöisesti integroituja kehitysympäristöjä (IDE, integrated development environment), joilla voidaan tehdä ohjelmien editointi, käännös, linkitys ja virheenjäljitys. Monesti ympäristöt osaavat myös analysoida ohjelman suorituskykyä ja muistinkulutusta. Vaikka ne saattavat helpottaakin aloittelijan kynnystä, niiden tehokas asiantuntijakäyttö vaatii kuitenkin ohjelmakehitykseen liittyvien käsitteiden ja menetelmien syvällistä ymmärrystä.

Erittäin tärkeässä roolissa tämä päivänä ohjelmistokehityksessä ovat erilaiset funktiot- ja luokkakirjastot. Nämä ovat valmiita ohjelmaosia, joita käytetään hyväksi uutta ohjelmaa toteutettaessa. Kirjastot liittyvät joko ohjelmointikieleen sen lisänä, tai johonkin erityisalaan. Ohjelmointikieleen sidotuista kirjastoista esimerkkinä ovat esim. C++ STL (Standard library) tai erilaiset Java APIt (application programming interface). Erityiskirjastoista voi mainita esim. grafiikkaohjelmointiin liittyvä OpenGL kirjasto tai graafisen käyttöliittymän toteuttava Qt.

Näiden lisäksi ohjelmointiympäristöön liittyy nykypäivänä yleinen tietokoneen käyttötaito ja myös konekirjoitustaito. Ohjelmointitehtävä vaikeutuu huomattavasti, jos opiskelija ei hallitse tietokoneen käyttöä, tai käyttää näppäimistöä yhdellä sormella jokaista kirjainta katseella erikseen näppäimistöltä hakien. Ohjelmointitehtävän ratkaiseminen hankaloituu, jos sen kirjoittaminen koneelle tuskastuttaa ja korjauksien tekeminen vie ylen määrin aikaa.

Kuviosta 5 puuttuu pallukka ohjelman kohdealueen hallinnalle. Ohjelman tekijällä tulee kuitenkin olla jonkinlainen käsitys siitä erikoisalueesta, jota toteutettava ohjelma toimii (kirjanpito, automaatioprosessi, tms.). Jos tämä tietämys puuttuu, on se hankittava ohjelmointiprojektin edetessä.

3.2 Ohjelmointitieto

Haapasalo (1994, 55-60) jakaa matemaattisen tiedon käsitteelliseen eli konseptuaaliseen tietoon ja proseduraaliseen tietoon. Tämä jako sopii hyvin myös ohjelmointitietoon. Konseptuaalinen tieto liittyy paitsi yksittäisiin käsitteisiin myös niiden välisiin riippuvuuksiin. Käsitteet muodostavat semanttisen verkon erilaisine tulkintoineen. Konseptuaalisesta tiedosta on toisissa yhteyksissä käytetty nimitystä deklaratiivinen tieto. Konseptuaalinen tieto liitetään usein ymmärrykseen:

> *“Oppilaan syvällinen tiedonmuodostus riippuu riittävän konseptuaalisen tiedon hallinnasta. Koska ei ole olemassakaan käsitteitä, jotka esiintyisivät täysin eristettyinä muista käsitteistä, semanttisten struktuurien ymmärtäminen on olennaista.”* (Haapasalo 1994)

Proseduraaliseen tietoon kuuluu formaalin kielen (esim. ohjelmointikielen) symboliset esitykset, säännöt ja niiden ymmärtäminen, ja erilaiset toimintatavat ja algoritmit ongelmien ratkaisemiseksi. Proseduraalinen tieto on eräänlaista käyttötaitoa. Sen avulla voidaan ongelmat ratkaista, mutta se ei vielä välttämättä riitä ratkaisun ymmärtämiseen.

> *“Ymmärtävä ja pysyvä oppiminen ... on mahdollista vain silloin, kun konseptuaalinen ja proseduraalinen tieto liittyvät kiinteästi toisiinsa ja tukevat toistensa muodostumista.”* (Haapasalo 1994)

Ohjelmointikäsitteet ovat pääsääntöisesti abstrakteja. Myös käsitteiden hierarkia on tärkeä kuten matematiikassa, jossa lauseet ja todistukset rakennetaan aksioomien varaan. Hierarkian pohjalla on teoreettinen malli tietokoneen toiminnasta. Ohjelmointikielet määritellään tämän mallin avulla. Ohjelmointi käsittelee siten artefakteja, joiden toiminta on tarkkaan mallinnettu ja niihin liittyvä tieto on eksaktia eikä 'sosiaalisesti neuvoteltavissa'. Esimerkiksi ohjelmointikielen syntaksi on ontologinen realiteetti, joka on sama tulkitsijastaan riippumatta.

Esimerkiksi Java-kielistä ohjelmaa kirjoitettaessa konseptuaalista tietoa, jota ohjelmoija joutuu käyttämään, on mm. oliot, luokat, metodit, tietotyypit. Proseduraalista tietoaan ym. käsitteitä soveltamalla hän pystyy kuvaamaan ongelmansa ratkaisun ohjelmointikielellä. Lopullinen ratkaisu tehdään Java-kielen syntaktisia ja semanttisia sääntöjä käyttäen. Päästäkseen ratkaisuun on ohjelmoijalla oltava selkeä käsitys koneesta, jolle ohjelmia toteutetaan. Kyse ei ole fyysistä tietokoneesta, vaan Java virtuaalikoneesta, jossa ohjelmaa suoritetaan.

3.3 Työelämävaatimukset ja ammatillinen osaaminen

Koulujen tulisi tuottaa työelämään osaavia ammattilaisia. Varsinkin ammattikorkeakouluissa ja myös teknisissä korkeakouluissa painotetaan työelämän taitoja. Tosin myös yliopistoista, joiden koulutus on edellisiä enemmän tutkimuspainotteista, tietojenkäsittelytieteen opiskelijat sijoittuvat valtaosaltaan yrityksiin töihin.

Raij (2003) tunnistaa osaamisesta neljä komponenttia (KUVIO 6). Osaamisessa yhdistyy sekä tieteellinen tietäminen että tekemisen osaaminen. Nämä tekijät on varsin helppo huomioida opetuksessa. Kuvion oikealla puolella mainitut asiat, kontekstin ymmärtäminen ja erilaisten tilanteiden hallintakyky ovat haasteellisempia. Kyse on työelämän eri tilanteiden hallitsemisesta, ja toimintakulttuurin ja ilmiöiden tajuamisesta. Kontekstiin, toimintaympäristöön, kuuluu keskeisinä ihmiset toimijoina ja osaajina. Ihmisillä on erilaisia työrooleja, mutta he ovat läsnä myös yksilöinä. Eri työelämän tilanteisiin opitaan reagoimaan harjaantumisen kautta, oman itsensä ohjaamisen ja tutkivan työotteen avulla.

Ohjelmoinnin voi ajatella olevan pitkälle käsityötä, koska vielä toistaiseksi ohjelmien tuottaminen on vaikeasti automatisoitavissa. Perinteisistä käsityöammateista poiketen tuotos, jonka ammattilainen tuottaa on täysin abstrakti. Sitten 1970- ja 80-lukujen alan ammattinimikkeet ovat muuttuneet. Aikaisempia nimikkeitä ohjelmoija ja atk-suunnittelija tuskin enää tapaa. Tilalle on tullut sellaisia nimikkeitä kuin ohjelmistoinsinööri (software engineer), ohjelmistosuunnittelija (software designer) tai ohjelmistokehittäjä (software developer). Nimikkeiden englanninkieliset versiot ovat suosiossa. Tehtävänkuvaan kuuluu kiinteästi, ohjelmien suunnittelun lisäksi, ohjelmistojen toteuttaminen ja testaaminen. Käytännössä työ on kuitenkin hyvin pitkälle

ohjelmointia ja ohjelmointiin liittyvien ongelmien ratkaisemista.

Ohjelmistotuotanto on myös hyvin pitkälle palveluala. Tuotanto keskittyy yhä enemmän palveluja tuottaviin yrityksiin, ja yritykset ostavat tarvitsemansa ohjelmistot tilaustyönä. Kansainvälisyys on usein vahvasti kuvassa mukana – varsinkin isommat ohjelmistofirmat toimivat useassa maassa. Tavoitteena on tuottaa toimivia ohjelmistoja asiakkaille mahdollisimman tehokkaasti ja edullisesti. Tehokkuus ja tuottavuusvaatimukset kansainvälisestä kilpailutilanteesta johtuen jatkuvasti kiristyvät, mutta laatuvaatimuksista ei kuitenkaan voida samalla tinkiä, oikeastaan päinvastoin.

KUVIO 6. Ammattikorkeakouluosaaminen integroituna kokonaisuuteen (Raij, 2003)

Ohjelmistoala on suhteellisen nuori ala verrattuna moneen muuhun teollisuuden alaan. Tietotekniikka kehittyy jatkuvasti, ja myös ohjelmistojen tuotantomenetelmät ovat koko ajan muutosten kohteena. Nykyisin suosiossa on ns. ketterät tuotantomenetelmät, joilla pyritään toisaalta projektien riskien, jotka ovat viime kädessä taloudellisia, parempaan hallintaan, ja toisaalta myös laadullisesti parempiin ohjelmistoihin.

Ketterissä menetelmissä on ideana kehittää ohjelmistosta ensin pieni toimivat malli, jota sitten täydennetään projektin edetessä. Suunnittelu, toteutus ja testaus suoritetaan nopeina sykleinä, ja syklin lopussa tulisi olla aina toimiva, hieman ominaisuuksiltaan täydennetty ohjelma olemassa. Näin pyritään myös dynaamiseen muutosten hallintaan projektin aikana.

Projektiryhmän työskentelyssä korostuu kommunikointi- ja yhteistyötaidot. Projektin etenemistä seurataan päivittäisissä palavereissa, ja työtä hidastavat ongelmat tulee tunnistaa mahdollisimman aikaisessa vaiheessa ja saattaa ryhmän tietoon. Ryhmän vastuulla on reagoida ongelmiin ja etsiä niihin ratkaisu. Ohjelmoinnin osuus on korostunut, koska ohjelmointityö jakaantuu koko projektin elinkaaren ajalle. Myös ohjelmien testaamiseen käytettävien ohjelmien koodaamiseen panostetaan enemmän kuin aiemmin. Huomattavaa on, että projektin tuottama ohjelmakoodi on tavallaan yhteistä eli kaikkien tiimin jäsenten tulisi periaatteessa pystyä muokkaamaan ja korjaamaan kenen muun tahansa projektityöntekijän toteuttamaa koodia. Kouluympäristössä opiskelija yleensä joutuu lukemaan, ymmärtämään ja korjaamaan vain itse tekemäänsä koodia.

Muutenkin olemassa olevan koodin ymmärtäminen, virheiden jäljitys ja testaus on huomattavan suuressa roolissa ohjelmointikursseihin verrattuna, joissa pääasiassa keskitytään uuden koodin tuottamiseen. Vanhoja ohjelmistoja joudutaan päivittämään, tehdyt muutokset saattavat levitä esim. ohjelmakirjastojen mukana hyvinkin laajalle, ja muutosten vaikutukset voivat olla kauaskantoisia. Laadunvarmistuksesta ei voida tinkiä.

Laadun takaamiseksi on olemassa erilaisia menetelmiä kuten pariohjelmointi ja testivetoinen kehitys. Nämä ovat menetelmiä, jotka ovat rantautuneet oppilaitoksiinkin. Käsittelen niitä hieman tarkemmin opetusmenetelmien yhteydessä (luku 4.4). Kukin kehittäjä vastaa myös kirjoittamansa koodin virheettömyydestä. Tähän liittyy koodiyksiköiden testauksen suunnittelu ja toteutus. Ohjelmistot kokonaisuutena testataan normaalisti testaukseen perehtyneiden asiantuntijoiden puolesta.

Yksittäinen kehittäjä on vastuussa hänelle annettujen tehtävien työmäärän arvioinnista. Työtehtävät on pystyttävä jakamaan sopiviin osatehtäviin, joista kukin arvioidaan erikseen. Kehittäjä itse tietää parhaiten, kuinka paljon minkäkinlaiseen työtehtävään häneltä itseltään menee aikaa. Jatkuva oman työn tarkkailu tietenkin kehittää ajan myötä työntekijän omaa käsitystä ammattitaidostaan. Keskittymällä työssä vain oleelliseen, voidaan myös projektin kokonaisaikatauluja lyhentää.

Erittäin huomattavaa työympäristössä on sen jatkuva teknologinen muutos. Ohjelmistotuotantoalalla muutoksen aiheuttavat markkinatilanteet, jotka tuovat uusia kehittyviä teknologioita pinnalle, ja vanhemmat teknologiat poistuvat käytöstä. Nämä aiheuttavat tarvetta osaamisen päivittämiseen. Yleensä kyse on helposti omaksuttavista muutoksista. Varsinaisia paradigman muutoksia tapahtuu harvoin, ehkä noin kerran kymmenessä vuodessa. Edellinen oli siirtyminen proseduraalisesta ohjelmoinnista olio-ohjelmointiin. Tällä hetkellä on käynnissä siirtyminen rinnakkaisohjelmointiin ja rinnakkaisalgoritmien käyttöön. Tämän muutoksen on mahdollistanut moniydinprosessorien esiinmarssi tavallisiin PC-tietokoneisiinkin.

Toinen ominaisuus on teknologinen diversiteetti. Rinnakkaisia teknologioita on hyvin paljon esimerkiksi ohjelmointikieliä, tietokantajärjestelmiä, ohjelmointiympäristöjä, käyttöjärjestelmiä. Ominaista näille on, että ne ovat perusperiaatteiltaan hyvin samankaltaisia, mutta eroavat teknisiltä yksityiskohdiltaan hyvinkin paljon. Tällaisessa ympäristössä tarvitaan ohjelmoijalta kykyä siirtää tietämyksensä teknologiasta toiseen. Tätä helpottaa periaatteiden tuntemus, ja kyky nähdä rajapintojen taakse eli tieto siitä, miten eri teknologiat on toteutettu.

Miten sitten parhaiten kouluelämässä voidaan valmistautua kohtaamaan työelämän vaatimukset? Ohjelmointialalle on tyypillistä jatkuva muutos. Parhaiten muutokseen voi valmistautua hankkimalla opiskeluaikana hyvät valmiudet elinikäiseen oppimiseen. Andrew S. Tanenbaum, arvostettu professori ja alan tietokirjailija, joka on julkaissut lukuisia ympäri maailmaa käytössä olevia tietokirjoja, esitti jo vuonna 1997 kymmenen kultaista sääntöään tietojenkäsittelytieteen opetukseen (Tanenbaum 1997). Säännöistä ensimmäinen kuului: “Ajattele pitkällä tähtäimellä”. Tänään koulusta valmistuvat ovat alalla töissä vielä neljänkymmenen vuoden päästä. Kuinka paljon ala ehtii muuttua siinä ajassa?

4 OHJELMOINNIN OPPIMISESTA JA OPETTAMISESTA

"Perhaps if we wrote programs from childhood on,
as adults we'd be able to read them."
Alan J. Perlis, 1982

Ohjelmoinnin opetuksen alkuvaiheet ovat yleensä ongelmallisia. Opiskelijalla pitää olla aika iso tietoperusta ennen kuin hän pystyy tekemään mielekästä ohjelmointia. Tähän kuuluu algoritmisen ongelmanratkaisun alkeet, opetuskielen syntaktinen ja semanttinen hallinta ja ohjelmointityökalujen ja -ympäristön hallinta.

Ennustettaessa opintomenestystä ohjelmointikurssilla on eri tutkimuksissa nostettu esille seuraavanlaisia valmiuksia (Shaffer 2005):

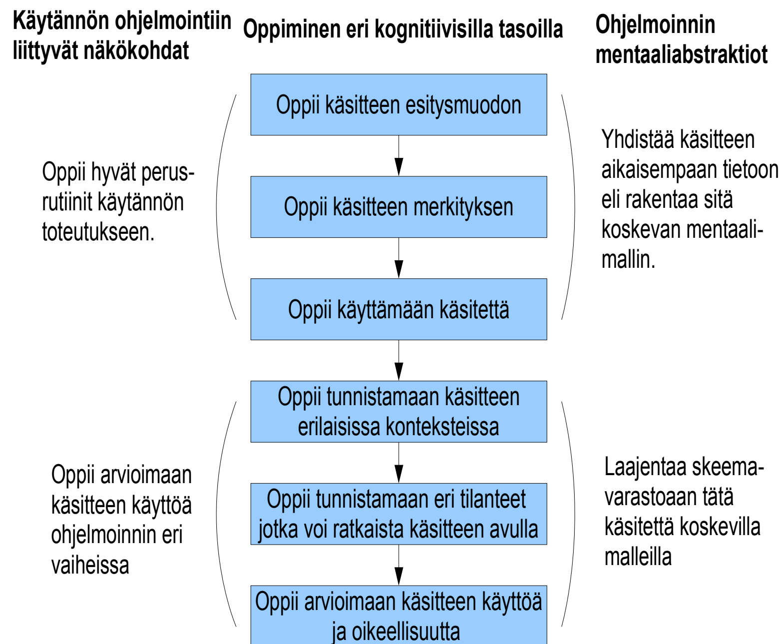
- Kyvykkyys ratkaista matematiikan sanallisia tehtäviä, ja kyvykkyys esittää epäformaali ongelmanasettelu formaalein menetelmin (esim. matemaattisena lausekkeena).
- Motivaatio, kiinnostus ja aikaisempi ohjelmointikokemus.
- Menestyminen koulumatematiikassa yleensä.
- Tehokkaat työmuistin käytön strategiat.
- Kenttäriippumattomuus, kyky strukturoida materiaalia uudella tavalla.

Toisaalta, joissain tutkimuksissa ei ole löydetty mitään selkää erittelevää kognitiivista tai persoonallista tekijää, joka ennustaisi menestymistä ohjelmointikursseilla (Robins yms. 2003), vaan niinkin yksinkertainen tekijä kuin opiskelijan omat odotukset arvosanasta ennustivat menestyksen parhaiten. Toivottava arvosana tietenkin kertoo jo opiskelijan omasta käsityksestä oppijana ja myös asenteesta ja motivaatiosta ohjelmointiin. Usein jopa ohjelmointipainotteisen opinto-ohjelman menestyksellistä suorittamista on pidetty osoituksena kyvystä omaksua ohjelmointitaito. Tosin tähän liittyvää 'luonnonvalintaa' (eli huonosti menestyvät keskeyttävät) voi helposti häiritä huonolla arviointipolitiikalla. Ulkoisten paineiden alla (esim. valmistuneiden määrän kasvatus), myös huonosti osaavat voidaan päästää kursseista läpi.

Yleinen tapa ohjelmoinnin alkeisopetukseen on kuitenkin lähestyä ongelmaa ohjelmointikielestä käsin. Ohjelmointikursseilla on tapana käydä kielen rakennetta läpi aihe kerrallaan (esim. tietotyypit, ehtolauseet, aliohjelmat, toistolauseet...) ja tietyssä järjestyksessä. Tämä lähestymistapa on, yleisyydestään päätellen, ilmeisen hyväksi havaittu ja kertautuu usein esimerkiksi oppikirjojen rakenteissa. Linn ja Dalbey (Winslow 1996) muotoilivat jo 15 vuotta sitten ns. ohjelmoinnin oppimisen ideaalisen ketjun, joka lähtee liikkeelle juuri kielen yksityiskohdista. Ketju voidaan ajatella vaiheittain etenevänä iteratiivisena prosessina:

1. Opeta/opettele yksi ohjelmointikielen ominaisuus sekä syntaksin että semantiikan kannalta.
2. Opeta/opettele yhdistämään opittu ominaisuus aiemmin opittuihin suunnittelu- ja ongelmanratkaisutaitoihin
3. Sulauta opittu taito yleisiin ongelmanratkaisutaitoihin.

Ala-Mutka (2005) on esittänyt vastaavan oppimisketjun Bloomin taksonomioita (Starr ym 2008) mukaillen (KUVIO 7). Oppiminen tapahtuu mielessä olevaa mentaalimallia (skeemaa) muokkaamalla ja parantamalla sen laatua. Yleisin käsitys siitä, miten ohjelmoinnin oppiminen tapahtuu, on juuri skeemojen, mentaalimallien, kasvattamisen kautta (Shaffer, 2005), ja rakentamalla eräänlainen 'työkalupakki' lähestymistavoista erityyppisiin ongelmiin. Eksperttiyden kasvaessa pakin sisältö muokkautuu eli skeemat muuttavat sisältöään. Tämä vaatii harjoittelua toinen toistaan seuraavien tehtävien avulla, jotka auttavat työstämään mentaalimalleja. Mallit ovat tärkeä osa ymmärtämyksen rakentamiseksi. Ohjelmointia koskevat mentaalimallit liittyvät tietorakenteisiin, tiedon esitysmuotoihin, kontrollirakenteisiin, ohjelman suunnitteluun ja ongelmaympäristöön. Ohjelmoinnin oppiminen sisältää myös mentaalimallin muodostamisen ohjelmoitavan tietokoneen toiminnasta.



KUVIO 7. Ohjelmointikäsitteiden oppiminen (Ala-Mutka, 2005)

Valmiudet ohjelmointikielten ja niihin liittyvien käsitteiden omaksumiseen riippuu oppijan kognitiivisen kehityksen tasosta (White ym. 2005). Piaget'n teoria jakaa ihmisen kognitiivisen kehityksen neljään vaiheeseen: sensomotoriseen, esioperatio-naaliseen, konkreettisten operaatioiden ja formaalien operaatioiden vaiheeseen. Formaalien operaatioiden vaihe kognitiivisessa kehityksessä saavutetaan normaalisti 11-12 vuoden iässä. Tässä vaiheessa mm. looginen ja abstrakti ajattelu kehittyy. Joil-lain ihmisillä kehitys tapahtuu myöhemmin, jos lainkaan. Formaalissa ajattelukyvys-säkin on havaittu eri tasoja. Esimerkiksi monet aikuiset ja toisen asteen opiskelijat eivät kykene suoriutumaan monista formaaleista operaatioista lainkaan(White ym. 2005).

Taulukossa 2 on yhteenveto eri tyyppisistä ohjelmointikielistä ja niiden vaatimista kognitiivisen kehityksen tasoista empiirisiin havaintoihin perustuen. Formaalien operaatioiden taso on jaettu taulukossa kahteen osaan perustuen juuri edellä mainittuihin havaintoihin formaalien operaatioiden kehittymisestä eri ihmisillä eri tavalla. Kognitiivinen tyyli viittaa siihen, kumpi aivopuolisko ajattelussa on hallitseva. Vasen aivopuolisko on yleensä liitetty loogiseen ajatteluun ja oikea konkreettiseen toimintaan. Mielenkiintoinen havainto on, että objektiorientoituneiden ja visuaalisten kielten hallinta on riippumaton kognitiivisesta tyylistä.

TAULUKKO 2. Ohjelmointikielet ja kognitiivinen kehitys/tyyli (White ym. 2005)

Ohjelmointikielen luokitus	Piaget:n kognitiivisen kehityksen taso				Kognitiivinen tyyli
	Esioperationaalinen	Konkreettinen	Esiformaalinen	Formaalinen	
Proseduraalinen (esim. C, Pascal)	Ylikuormittava	Ylikuormittava	Ylikuormittava	Tuottava ja motivoitunut	Vasen aivopuolisko
Objektiorientoitunut (C++, Java)	Ylikuormittava	Ylikuormittava	Ylikuormittava	Tuottava ja motivoitunut	Molemmat aivopuoliskot
Visuaalinen (Visual Basic)	Ylikuormittava	Ylikuormittava	Tuottava ja motivoitunut	Alikuormittava	Molemmat aivopuoliskot
Merkintäkieli (HTML, XML)	Ylikuormittava	Tuottava ja motivoitunut	Alikuormittava	Alikuormittava	Oikea aivopuolisko

4.1 Ohjelmoinnin oppimisen sietämätön raskaus

Mikä ohjelmoinnin oppimisessa sitten on vaikeaa? Lahtinen ym. (2005) tutkivat ensimmäisen vuoden yliopisto-opiskelijoiden omaa näkemystä ohjelmointikurssista ja sen sisällöstä. Tutkimus toteutettiin kyselytutkimuksena. Opiskelijat aloittivat ohjelmoinnin joko Java tai C++ kielellä, joista kumpikin on olio-ohjelmointikieli. Vaikeimpia ohjelmointiin liittyviä kokonaisuuksia opiskelijoiden mielestä olivat 1) miten laatia tietyn ongelman ratkaiseva ohjelma, 2) ohjelman toiminnallisuuden jako aliohjelmiin 3) virheiden löytäminen omista ohjelmista. Kaikki nämä vaativat isojen kokonaisuuksien hallintaa eikä vain yhden käsitteen tai pienen osa-alueen.

Vaikeimmat yksittäiset ohjelmointikäsitteet olivat 1) rekursio, 2) osoittimet ja viitteet, 3) virheenkäsittely ja 4) funktiokirjastojen käyttö. Toisaalta ongelmat eivät välttämättä liity käsitteiden ymmärtämiseen sinänsä, vaan lähinnä kykyyn käyttää niitä oikeissa kohdissa ohjelmaa eli käsitteiden soveltaminen ongelmanratkaisun apuna. Funktiokirjastojen tehokas käyttö vaatii itsenäisiä työskentely- ja tiedonhakutaitoja, mikä saattaa olla vaikeaa aloittelevalle opiskelijalle. Tutkimuksessa havaittiin myös, että opiskelijat yleensä yliarvioivat oman osaamisensa. Sinällään tämä on jo oppimisen este.

Tutkimuksessa havaittiin vahva korrelaatio ohjelmoinnin tiettyjen ydinosaamisalueiden oppimisen välillä. Näitä olivat 1) ohjelmarakenteiden ymmärtäminen, 2) miten laatia tietyn ongelman ratkaiseva ohjelma, 3) ohjelmointikielen syntaksin osaaminen ja 4) ohjelman toiminnallisuuden jako aliohjelmiin. Nämä joko opittiin kaikki tai jokaisen kanssa oli ongelmia. Näiden osa-alueiden hallinta korreloi myös vahvasti yksittäisten ohjelmointikäsitteiden oppimisen kanssa.

Myös Milne ym. (2002) saivat samanlaisia tuloksia tutkiessaan C++ kielen oppimista. He havaitsivat vaikeimmiksi aihealueiksi oppia aiheet, jotka vaativat ymmärtämystä siitä, mitä tietokoneen muistissa lopulta tapahtuu ohjelman suorituksen aikana (osoittimet, dynaaminen muistinvaraus, parametrien välitys, tietorakenteet). Epävarmuus aivan perusasioiden hallinnassa vaikuttaa monimutkaisempien käsitteiden ymmärtämiseen. Vaikeiksi listattiin myös oliosuuntautuneeseen ohjelmointiin liittyvät käsitteet (kopiorakentajat, polymorfismi, virtuaalifunktiot), jotka liittyvät muistinhallintaan. Myös Milnen tutkimuksessa tuli esille että opiskelijat arvioivat oman osaamisensa korkeammalle kuin opettajat.

Varsin monessa tutkimuksessa onkin huomautettu, että selkeän tietokoneen mentaalimallin olemassaolo on ohjelmoinnin omaksumisen edellytys (mm. Shaffer 2002, Ben-Ari 2001, Robins ym.2003). Vastaavasti voidaan ajatella, että sen puuttuminen on oppimisen este. Malli pitää opiskelijoilla eksplisiittisesti opettaa, koska se on yleensä ontologinen realiteetti, jonka konstruoiminen ilman etukäteistietämystä on mahdoton tehtävä. Toisaalta, opiskelijalle riittää, että opetuksessa käsitellään hänen omaa työskentelytasoaan yhtä kerrosta alempana olevaa abstraktiota. Toisin sanoen, ohjelmoijan tulee tuntea sen koneen rakenne, jolle hän ohjelmiaan tekee.

4.1.1 Erilaisia oppimisstrategioita

Opiskelun alkuvaiheessa ensimmäisillä ohjelmointikursseilla turhautumisen syynä voi olla virheelliset opiskelustrategiat. Opiskelijat käyttävät aiemmin muissa aineissa oppimiaan strategioita, eivätkä yleensä ymmärrä heti ohjelmoinnin opiskelussa vaadittavaa tarkkuuden ja systemaattisuuden tasoa. Ohjelmat on kirjoitettava välimerkilleen oikein ennen kuin niitä pystytään kokeilemaan. Ohjelmien testaus on monimutkaista, ja virheiden jäljitys vaatii järjestelmällisyyttä ja vie aikaa vievää.

Opiskelijan suhtautuminen ohjelmointivirheisiin on merkittävä opintomotivaation indikaattori (Robins ym. 2003). Ohjelmointivirheet ovat hyvin yleisiä ja yksi opintojen päätavoitteista onkin oppia virheiden korjaaminen. Ohjelmien kehitystyökalut pystyvät antamaan automaattisesti hyvin suoraa ja informatiivista palautetta ohjelman virheistä. Toiset opiskelijat suhtautuvat virheilmoituksiin negatiivisesti, turhautuvat, eivätkä pysty ratkaisemaan ongelmaa enää itsenäisesti. Toiset opiskelijat käyttävät virheistä saamaansa palautetta (ohjelman tai kääntäjän antamaa) suoraan hyväkseen ja pystyvät jatkamaan ohjelmansa kehittämistä. On vielä kolmas ryhmä, joka yrittää tehdä ohjelmaan tavallaan satunnaisia korjauksia, koska eivät puutteellisten taitojen takia kykene jäljittämään virhettä. Heidän edistymisensä ei ole ensimmäistä ryhmää kummoisempi.

Esimerkki virheellisen strategian omaksumisesta on ns. leikkaa-liimaa -tyylinen ohjelmointi. Ohjelmointioppaiden ja -kirjojen, ja miksei myös opettajien, antamat esimerkit voivat johtaa tietyssä tapauksissa tähän tyyliin, jos opetuksessa epäonnistutaan herättämään oppilaan oma ajatustyöskentely. Ilmiötä voi myös kutsua ohjelmoinnin 'tarkoituksen menetykseksi' (loss of intentionality) (Boyer ym. 2008). Sen sijaan, että opiskelija yrittäisi itse ratkaista ongelman, hän yrittää käyttää hyväkseen valmista ratkaisua aikaisemmasta hieman samalta näyttävästä ongelmasta. Opiskelijan strategia on joko selvitä tehtävästä mahdollisimman helposti ja nopeasti (ilman motivaatiota ratkaisun ymmärtämisestä) tai sitten hän on jo aikaisemmin kurssilla tippunut ns. kärryiltä, ja tehtävä on aivan liian vaikea hänelle ratkaistavaksi. Ilman syvällisempää ymmärtämystä ratkaisun rakenteesta siihen tehdään jopa satunnaiselta tuntuvia muutoksia. Ohjelma pystytään nopeasti kääntämään ja kokeilemaan. Mikäli muutokset eivät ole mieleen, kokeillaan seuraavaa mahdollista ratkaisua. Ohjelmointi

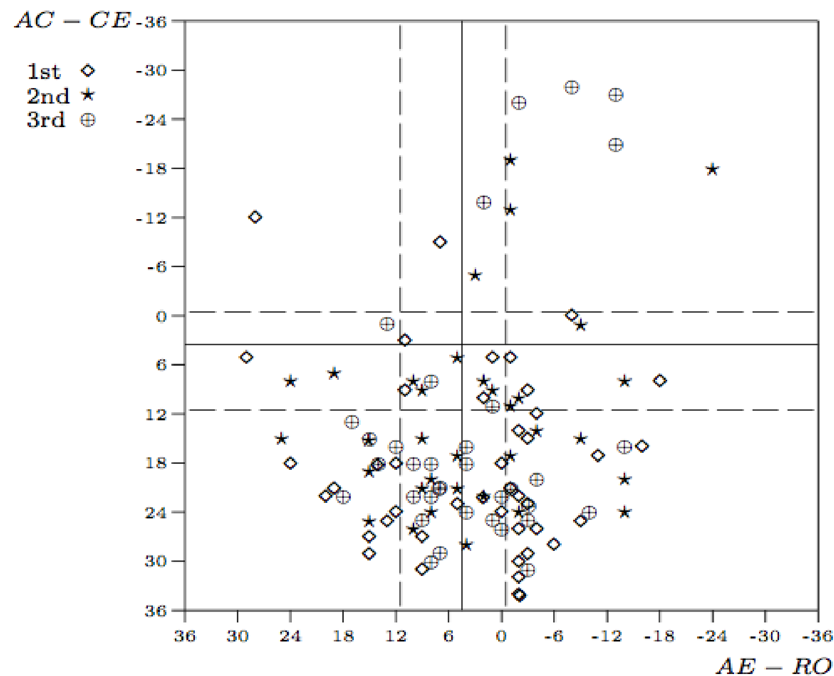
on ajatteluprosessin sijaan kuin palapelin rakentamista sokkona – kuin yksi palanen ei sovi, yritetään mekaanisesti seuraavaa jne.

Toisenlainen ongelma on ns. bricolage -tyyli (Ben-Ari 2001). Bricolage tarkoittaa, jonkin artefaktin kokoamista puhtaasti yritys-erehdys -menetelmällä. Ohjelmoinnissa tämä tarkoittaa ohjelman kirjoittamista ilman minkäänlaista etukäteissuunnittelua tai teoreettista taustatukea. Hyvin yksinkertaisten ohjelmien kasaaminen näin on etevältä ohjelmoijalta helppoa, mutta lähestymistapa johtaa väistämättä ongelmiin monimutkaisten tehtävien kanssa, jotka vaativat systemaattisia suunnittelu- ja analyysitaitoja.

Kysyttäessä ensimmäisen vuoden yliopisto-opiskelijoilta (Lahtinen ym. 2005), he pitivät yksin työskentelyä (sekä opiskelu että ohjelmointi) parhaana tapana oppia. Sitä suosittin luentojen ja ohjattujen käytännön sessioiden kustannuksella. Toisaalta ohjatusta opetuksesta arvostettiin eniten käytännön pienryhmäharjoittelua. Kaikista paras opiskelumateriaali oli opiskelijoiden mielestä esimerkkiohjelmat. Tulokset kertovat oppimisen tapahtuvan käytännön harjoituksen ja esimerkkien kautta. Mutta kuten yllä todettiin, esimerkeillä voi olla negatiivisiakin vaikutuksia.

4.1.2 Ohjelmoijien oppimistyylit

Ohjelmoinnin opiskelijoiden oppimistyylejä on myös tutkittu. Galpin yms. (2007) tutkivat etelä-afrikkalaisia yliopisto-opiskelijoita ensimmäisellä, toisella ja kolmannella vuosikurssilla Kolbin oppimistyylien kannalta (katso sivu 11 ja kuvio 3). Tulos oli pääsääntöisesti se, että suurin osa opiskelijoista sijoittui abstraktio-tunne -akselilla vahvasti abstrakti käsitteellistämisen puolelle (KUVIO 8), mikä on opiskelijoiden alan valintaa ajatellen oikeansuuntainen tulos. Vastaavan suuntaisia tuloksia on saatu myös länsimaisissa yliopistoissa. Huonoa oppimistulosta ei siis pääsääntöisesti voi laittaa persoonallisuuden kannalta epäonnistuneen uravalinnan syyksi.



KUVIO 8. Opiskelijoiden jakautuminen Kolbin oppimistyyleihin (Galpin 2007)

Mancy ym. (2004) tutkivat alkeiskurssin oppilaiden kognitiivisia ominaisuuksia, työmuistin määrää ja kenttäriippuvuutta. Matematiikassa ja tilastotieteessä työmuistin koolla ja kenttäriippumattomuudella voidaan ennustaa menestymistä oppiaineessa. Mancy ym. mukaan kuitenkin työmuistin määrällä on vain marginaalinen merkitys ohjelmointikurssilla menestymiseen. Sen sijaan kenttäriippumattomiksi luokitteleminen korreloi erittäin vahvasti arvosanan kanssa erityisesti ohjelmointikykyä mittaavassa kokeessa. Tästä voisi päätellä, että informaatiotulvat ohjelmointikursseilla on suuri. Toinen mahdollinen tulkinta asialle on, että kurssin opettajat ovat hyvin osanneet huomioida kognitiivisen kuormituksen opetusta suunnitellessaan. Tekijät pohtivat, pystyykö kenttäriippumattomuutta opettamaan ja kuinka helposti. Tätä voisi helpottaa avainkohtien painottaminen kunkin opeteltavan asian kohdalla. Toisaalta kenttäriippuvuuden testaus opiskelijoilta voisi auttaa opettajaa suunnittelemaan opetustaan.

4.2 Ohjelmoinnin oppimisympäristö

Oppimisympäristö vaikuttaa oppimiseen joko positiivisesti tai hidastavasti. Ohjelmointikurssilla oppimisympäristöön voidaan laskea kuuluvaksi, paitsi luvussa 2.1 kuvatut asiat (sivu 5), myös käytettävä ohjelmointikieli ja ohjelmien kehittämiseen käytettävä ohjelmisto (käyttöliittymä, editori, kääntäjä, virheenjäljitin). Aloittelevan ohjelmoijan on usein vaikea hahmottaa suorituskelpoisen ohjelman tuottamiseen tarvittavaa työkaluketjua.

Otan seuraavassa esimerkinomaisesti esiin nämä kaksi asiaa, ohjelmointiympäristön ja -kielen valinnan. Kehitysympäristön valinnasta teksti perustuu omiin havaintoihin eri kursseilla käytetyistä työkaluista sekä yliopisto- että ammattikorkeakouluopetuksessa. Ohjelmointikielen valinta perustuu Mannilan ym. (2008) eri kieliä vertaileviin tutkimuksiin.

Valinnoilla oppimisympäristön suhteen voidaan vaikuttaa ratkaisevasti kognitiiviseen kuormittavuuteen. Valinnat eivät tosin ole aina opettajasta kiinni vaan esim. ensimmäisen ohjelmointikielen valinta määrätään opetussuunnitelmassa. Paineita voi tulla mm. 'työelämäyhteensopivuudesta'. Samoin ohjelmointityökalut valitaan monesti sen mukaan, minkä toimittajan ohjelmat oppilaitokseen on saatu edullisimmalla sopimuksella, eikä suinkaan pedagogisten periaatteiden mukaan.

4.2.1 Opetusvälineiden valinnasta

Oikeansuuntaisilla työkaluvalinnoilla voidaan häivyttää ohjelmointiympäristön tuomaa ylimääräistä kognitiivista kuormaa oppimistilanteessa erityisesti ohjelmoinnin alkeisopetuksessa. Kuviossa 9 on esimerkki opetuskäyttöön suunnitellusta BlueJ-ohjelmankehitysympäristöstä (BlueJ 2010). Käyttöliittymä on hyvin pelkistetty sisältäen vain kaikkein oleellisimman. Vasemmassa ikkunassa opiskelija näkee ohjelman graafisessa esitysmuodossa, ja voi jopa muokata tätä lisäämällä ohjelmaan uusia luokkia ja niiden välisiä yhteyksiä graafisten elementtien avulla. Oikeanpuoleiseen ikkunaan on aukaistu yhden luokan (esimerkin Triangle) lähdekoodi ohjelmaeditoriin muokattavaksi. Erilaiset ohjelmointikielen syntaktiset elementit on koodattu eri väreillä, mikä on yleistä tämän päivän editoreissa. Aivan oikeassa laidassa on navigointipalkki, joka auttaa ohjelmoijaa hahmottamaan, mitä kohtaa ohjelmalistausta parhaillaan hän

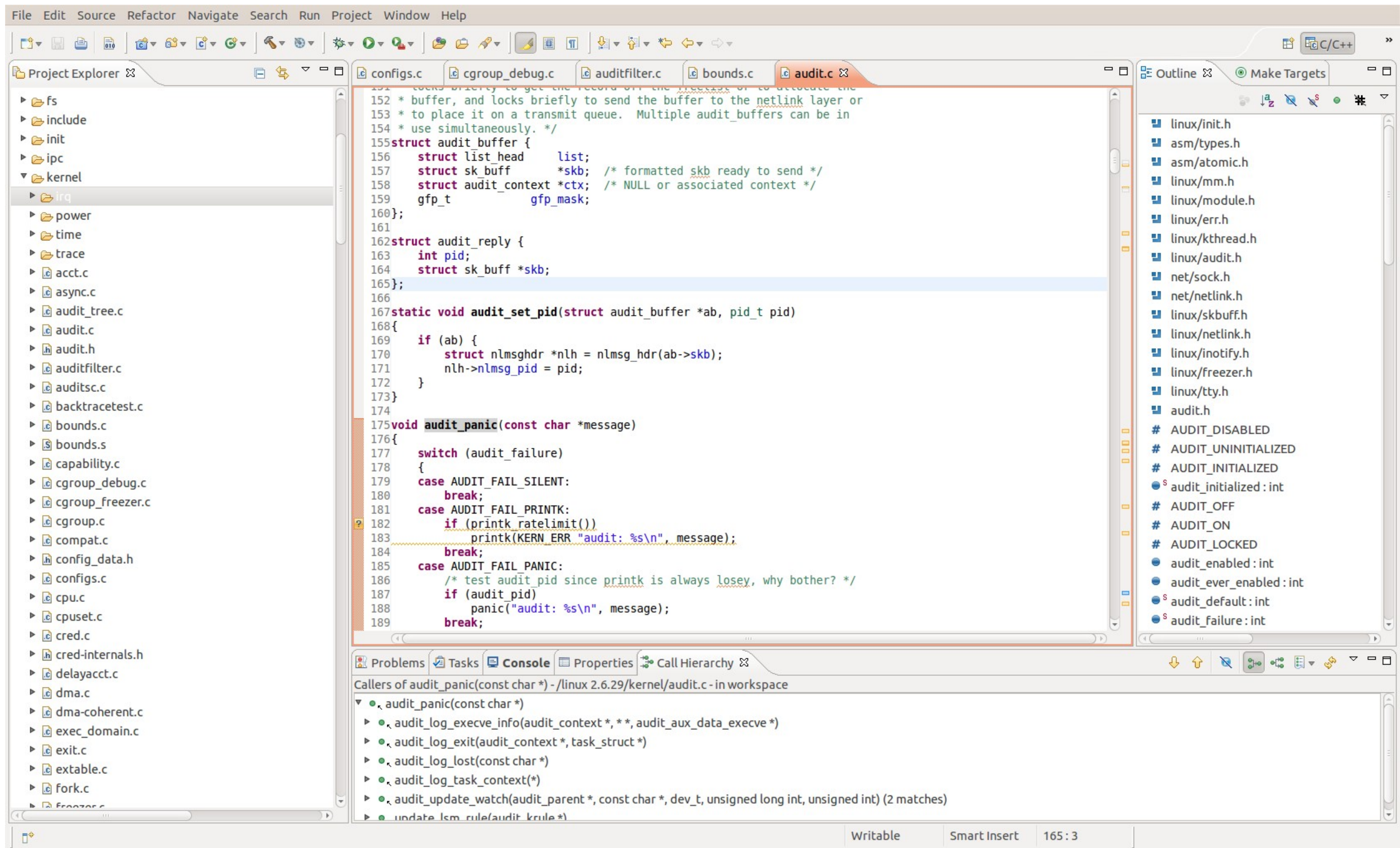
käsittelee. BlueJ itsessään on vain osa kokonaisuudesta, jonka tavoite on luoda selkeä pedagogiikka olio-ohjelmoinnin opettamiseen. Työkalu on kehitetty opetusmetodin tueksi.

Kuviossa 10 puolestaan on esimerkki ammattikäyttöön soveltuvasta Eclipse-ympäristöstä. Eclipse on erittäin monipuolinen, laajennettavissa oleva ja ohjelmoijan käyttötarpeen mukaan muokattava työkalu. Eclipse on ilmaisohjelmisto, mutta vastaavia kaupallisia ohjelmistoa on ammattilaisille ja myös oppilaitoskäytössä runsaasti tarjolla. Verrattuna BlueJ-ympäristöön, huomio kuvassa kiinnittyy käyttöliittymän yksityiskohtien runsauteen, lukuisiin ikkunoihin ja välilehtiin. Aloittelijan käytössä työkalu on avain liian monimutkainen, ja lukuisat ominaisuudet vievät huomiota itse opiskeltavalta asialta. Valitettavasti todellisuudessa moni aloittaa ohjelmointiopiskelunsa tämän tasoisilla työkaluilla. Kouluissa on yleistä hankkia jonkin kaupallisen tarjoajan ohjelmistoja, mutta on epäselvää kuinka tarkkaan näiden pedagogisia ominaisuuksia on oikeasti evaluoitu.

Mielenkiintoinen kysymys on, jos ohjelmointi aloitetaan yksinkertaisella työkalulla, niin kuinka helppoa on opitun siirtäminen ammattityökaluihin. Jälleen on kysymys siitä, miten hyvin opiskelija on oppinut periaatteet, eikä vain pinnalle näkyvän näppäintekniikan. Toinen opiskelija kysyy uudessa ympäristössä: “Miten saan käännettyä ohjelmani?” ja toisen kysymys on: “Missä on se nappi, jota tarvitsen, ja joka aikaisemmassa ympäristössä oli tuossa?”.

Toisilla opiskelijoilla oppimisen esteenä saattaa olla yksinkertaisesti se, että tekstin tuottaminen tietokoneella on hidasta. Kun lisäksi ohjelmointiharjoituksissa tuotettavan koodin muotovaatimukset ovat tiukat eli ohjelmien on oltava syntaktisesti välimerkilleen oikein, voi editointiin ja koodin korjaukseen mennä kohtuuttomasti aikaa, ennen kuin opiskelija saa ohjelmansa edes kerran käännettyä suoritettavaan muotoon, puhumattakaan toimivaksi korjaamisesta. Sopiva editori pystyy auttamaan ohjelman kirjoittamisessa esimerkiksi värikoodaamalla syntaktiset elementit ja tarkistamalla jatkuvasti kirjoitetun ohjelman syntaksia. Ja sujuvan konekirjoitustaidon hankkiminen opiskelujen alkuvaiheessa voisi olla hyödyllinen panostus.

KUVIO 9. Ruudunkaappaus opetuskäyttöön suunnitellusta BlueJ-ohjelmankehitysympäristöstä.



KUVIO 10. Ruudunkaappaus ammattikäyttöön soveltuvasta Eclipse-ohjelmankehitysympäristöstä.

4.2.2 Opetuskielen valinnasta

Mannila ym. tutkivat ohjelmointikielen valinnan vaikutusta oppimiseen ohjelmoinnin alkeisopetuksessa. Hypoteesina oli käyttää mahdollisimman yksinkertaista kieltä. Tutkijat valitsivat kieleksi Pythonin. Python on suhteellisen uusi tulkattava ns. skripti-kieli, joka on syntaksiltaan pelkistetty perinteisiin teollisuuskieliin verrattuna. Skripti-kieliset ohjelmat suoritetaan ohjelmatulkin avulla, jolloin käännös-linkitys -vaihe jää ohjelmakehityksessä pois. Kääntöpuolena on, että tulkattavat ohjelmat ovat hitaampia suorittaa, mutta ohjelman suorituskyvyn merkitys on oppimisvaiheessa häviävän pieni.

Ohjelmoinnin alkeiskurssilla tulisi ennen kaikkea valmistaa opiskelijoita algoritmiseen ajatteluun ja opettaa ohjelmoinnin perustaitoja. Nämä taidot ovat käytännössä samoja kielestä riippumatta. Kehitettävien mentaalimallien tulisi liittyä 'ohjelmoinnin ideaan' eikä ohjelmointikieleen. Kuten aiemmin on tullut esille, kieli on vain työkalu. (Toki kursseilla joutuu opettamaan puhtaasti opetuskieleen liittyviä käsitteitä). Syntaktisesti yksinkertainen Pythonin tapainen kieli sopii tähän tarkoitukseen erittäin hyvin. Oppilas voi paremmin keskittyä olennaiseen, eikä kuluta aikaa toissijaisiin ongelmiin. Pythonista puuttuu muutamia perinteisiä ohjelmointikielen osia, kuten muuttujien ennakkomäärittely ja staattinen tyypitys, mikä saattaa tehdä siitä arvelutt-avan valinnan (joidenkin mielestä) opetuskieleksi.

Mannilan tutkimuksessa oli kaksi vertailuryhmää. Toinen ryhmä opetteli ohjelmointia Pythonilla ja toinen Javalla, joka on tällä hetkellä yleisimpiä teollisuudessa käytettyjä kieliä. Kurssien lopputuloksena tehtyjä harjoitustöitä analysoitiin sekä logiikkavir-heiden kannalta että toiminnallisuuden kannalta. Työt eivät olleet täysin identtisiä, mutta laajuudeltaan toisiaan vastaavia. Analyysissä havaittiin, että Python-ryhmän ohjelmissa oli keskimäärin vähemmän syntaksivirheitä (2 vs 19) ja logiikkavirheitä (17 vs 40). Python-ohjelmat myös useimmiten suorittuivat virheettömästi (63% vs 28%) ja vastasivat määritelmäänsä (70% vs 28%) eli toisin sanoen toimivat niinkuin oli alun-perin tarkoitettukin. Myös muilta osin tutkijat havaitsivat Python-ohjelmat laaduk-kaammiksi. Vaikka kokeen otos oli pieni - molemmissa ryhmissä 30 oppilasta, tutki-muksen tulokset olivat intuitiivisesti varsin helposti ymmärrettävissä.

Tutkimukseen kuului vielä toinenkin osa, jossa selvitettiin Python-opiskelijoiden sopeutumista jatkossa Java-kurssille. Siinä todettiin, että ohjelmointitietämys - 'ohjelmoinnin idea', joka oli omaksuttu Pythonilla ohjelmoitaessa, oli varsin hyvin siirrettävissä eteenpäin, ja opiskelijoilla oli hyvin valmiuksia oppia Javaan liittyvät erikoisuudet, ja Pythonista puuttuvat 'oikean' kielen ominaisuudet (tai rasitteet, näkökulmasta riippuen).

Tutkimuksessa tulee selkeästi esille kognitiivisen kuorman ja oppimisen välinen yhteys. Javan opettelu ensimmäisenä ohjelmointikielenä voi olettaa olevan selkeästi kognitiivisesti kuormittavampaa. Kun kuorma sovitetaan paremmin oppijan kykyihin ja valmiusasteeseen, on oppiminen tehokkaampaa. Tämä heijastuu suoraan tuottavuuteen. Tutkimuksen perusteella ei ainakaan ole perusteita aloittaa ohjelmoinnin opiskelua teollisuuskielellä, mikä varsinkin ammattikorkeakouluissa on 'yritysyhteensopivuuden' takia yleistä.

4.3 Ohjelmoinnin opetuksen ongelmia

Oppimisen ongelmat voivat johtua osittain siitä, että opiskelijat eivät alkuvaiheessa pysty vielä ymmärtämään, mistä ohjelmoinnissa on lopulta kyse. Opiskelijat helposti erehtyvät luulemaan, että ohjelmointi rajoittuu ohjelman kirjoittamiseen. Syytä ei välttämättä kannata etsiä opiskelijoista, vaan monesti kyse on ohjelmointikurssien sisällön suunnittelusta – niillä keskitytään ohjelmien tuottamiseen ja kielen opiskeluun. Sama näkökulma on usein myös alan oppikirjoissa. Muut ohjelmointiin liittyvät taidot opiskelijan oletetaan oppivan ikään kuin 'siinä sivussa'.

Analogiana voisi ajatella vieraan kielen oppimista koulussa. Jos kieltä harjoiteltaisiin vain kirjoittamalla, ja jätettäisiin lukeminen ja puhuminen pois, voisi kielen opetuksen ajatella olevan vajavaista. Vaikkakin analogian voi helposti havaita jossain kohti ontuvan, on se kuitenkin joiltain osin varsin osuva. Myös ohjelmointikielissä on ohjelman tuottamisen todettu vaativan erilaisia kognitiivisia taitoja kuin ohjelman ymmärtämisen ja lukemisen. Ohjelman ymmärtämiseen liittyvät taidot auttavat huomattavasti virheiden jäljityksessä ja suorituskyvyn analysoinnissa.

Ohjelmointikurssilla pitäisi tuoda esille koko kuviossa 5 (sivu 20) esitetty kokonaisuus tavalla tai toisella. Harjoitusten pitäisi kehittää paitsi ohjelman kirjoitustaitoa, myös suunnittelu, arviointi- ja lukutaitoa sekä virheiden jäljitystä.

Usein kursseilla opettaja tarjoaa tehtävistä malliratkaisut. Tämä on perusteltua mm. siksi, että oppilaat etenevät eri tahtia eivätkä kaikki ehdi tehdä kaikkia annettuja tehtäviä. Valmiista ratkaisusta puuttuu kuitenkin se ajatteluprosessi, jolla ratkaisuun päästiin. Ajatteluprosessi, ratkaisun saavuttamisen strategia on kuitenkin se tärkein osa lopullista ratkaisua. Sen ymmärtäminen auttaa pääsemään ratkaisun alkuun seuraavissakin tehtävissä. Jos ei koskaan pakota itseään epämukavuusalueelle pohtimaan itsenäisesti ratkaisua vaan odottaa malliratkaisua, oppiminen jää tapahtumatta. Ulkoa opiskelusta ei ole ohjelmoinnissa mitään hyötyä.

Malliratkaisun antaa myös opiskelijalle mielikuvan, että ohjelma on syntynyt kertakirjoittamalla. Tämä mielikuva on täysin vääristynyt. Todellisuudessa kaikki ohjelmat tehdään enemmän tai vähemmän inkrementaalisen metodin kautta eli aloittamalla yksinkertaisesta rungosta, ja sitten täydentämällä ja lisäämällä ohjelmaan ominaisuuksia. Malliratkaisussa jää siis paitsi ratkaisustrategia, myös ohjelman todellinen kirjoitusprosessi havainnollistamatta.

Lisäksi opetuksen suunnittelussa olisi huomioitava, että kurssilla opetetaan ohjelmointia, ei ohjelmointikieltä. Kieli on vain työkalu ohjelman toteuttamiseen. Opetuksen pitää auttaa opiskelijaa luomaan yleisiä ongelmanratkaisumalleja eikä pelkästään spesifisiä tiettyyn kieleen sidottuja malleja.

4.4 Opetusmenetelmistä

4.4.1 Kognitiivinen oppipoikamalli

Kognitiivinen oppipoikamalli (cognitive apprenticeship) yrittää tuoda eksperttien taidot esille opetuksessa. Siinä keskitytään niihin prosesseihin, joita ekspertit käyttävät monimutkaisia tehtäviä suorittaessaan. Kuten nimikin viittaa siinä keskitytään kognitiivisten tai metakognitiivisten prosessien oppimiseen ekspertin opastuksen kautta (Caspersen ym. 2007). Menetelmään kuuluu olennaisesti esimerkkien läpikäynti (worked examples). Ero malliratkaisuihin on se, että esimerkkiä käsiteltäessä

käydään läpi myös itse ongelmaratkaisuprosessi.

Esimerkkien läpikäynti on oleellista opetuksessa, jossa pyritään jonkin taidon omaksumiseen. Ohjelmoinnin opetuksessa järjestely voi olla esimerkiksi se, että opettaja ratkaisee ongelman ryhmän edessä, käyttäen tietokoneeseen liitettyä videotykkiä, ja selittää samalla oman ajatusprosessinsa etenemisen (ns. 'live coding') (Boyer ym. 2008). Opettajan johtamana toimintona tällä on suora yhteys oppimistavoitteisiin. Kun läpikäynti toistuu riittävän moneen kertaan, opiskelijat pystyvät 'matkimaan' prosessia ja ratkomaan itsenäisesti vastaavia ongelmia. Toisaalta lähestymistavassa on se ongelma, että esimerkin läpikäynnissä toistetaan opettajan (tai oppikirjan) omaksumia ajatusrakennelmia. Opiskelijat eivät välttämättä omaksu itsenäistä ajattelutapaa vaan oppivat matkimalla ja käyttämään analogioita.

Gaspar ym. (2009) kehittivät mallia eteenpäin siirtymällä opettajan demonstraatiosta opiskelijoiden demonstraatioon. Tällöin ohjauksen näkökulma muuttuu siten, että opettajan ajatusrakennelmien sijaan käsitellään opiskelijoiden ajatusprosessia. Näin päästään selvemmin oppimisen ongelmakohtien jäljille. Opetustilannejärjestely voi olla esim. sellainen, että yksi opiskelijoista (tai pari) ratkaisee ongelman opettajan ohjauksella. Oppilaan monitori heijastetaan taululle, josta muut voivat seurata ratkaisun kulkua. Varsinkin aivan vasta-alkajien kanssa tämä vaatii herkkää otetta, ja voi olla aikaa vievää. Myös muiden opiskelijoiden saaminen mukaan prosessiin voi olla haaste. Vaihtoehtoinen tapa onkin antaa koko ryhmän etsiä ratkaisua jonkin aikaa (yksittäin tai pareina), ja palata sitten yhteen ratkaisuun ja sen syntyprosessiin tarkemmin, jolloin muillakin opiskelijoilla kuin ratkaisun tekijällä tulisi olla annettavaa keskusteluun.

Jälkimmäinen menetelmä on ollut hieman toisenlaisessa muodossa yliopistoissa käytössä jo vuosikymmeniä. Luento-opetukseen on kuulunut kiinteänä osana harjoitustehtävät. Opiskelijat ratkaisevat tehtävät omalla ajallaan (yksin tai vapaasti ryhmäytyen) ja ne käsitellään viikottaisissa pienryhmäkokoontumisissa. Kunkin tehtävän ratkaisun esittelee yksi opiskelijoista vuorollaan. AMK-opetuksessa opiskelijoiden demonstraatioille voi olla vaikeampi löytää aikaa rajallisen opetusajan takia. Opiskelijoilla on oltava ensi itsenäistä aikaa ongelman ratkaisuun ja sitten vielä vähintään sama aika ratkaisun läpikäyntiin. Ratkaisevaa lähestymistavassa kuitenkin on,

että käsitellään opiskelijan omia ratkaisuja ja ennen kaikkea, miten ratkaisu on syntynyt. Ohjelmoinnissa, jos missä, pitää paikkansa vanha sanonta 'virheistä oppii'. Alkuvaiheessa epäonnistumisen ja virheiden pelko on yleensä suuri, josta syystä oman ratkaisun esittely voi olla vaikeaa ja tuntua epämiellyttävältä. Epämiellyttävät tunteet toimivat oppimisen esteenä.

Muistan itse omilta opiskeluajoiltani kuinka vaikeaa oman ratkaisun esittäminen alussa oli. Silloin käytettiin piirtoheittimiä ja säästösyistä vesiliuokoisia tusseja, koska kalvoja haluttiin kierrättää. Itselläni ei oman ratkaisun esittäminen ollut sinänsä ollut ongelma, vaan ratkaisun kirjoittaminen piirtoheittimelle. Vasenkätisyys ja vesiliukoiset tussit ovat huono yhdistelmä.

Gaspar ym. (2009) ehdottaa, että koko tuntiopetus tulisi rakentaa esimerkkien läpikäynnissä opiskelijoilta kertyvän aineiston pohjalta, koska menetelmä tuo esille virheet ajatusrakenteissa aikaisessa vaiheessa ja ensiluokkaista tietoa oppimisen esteistä. Asioiden teoreettinen läpikäynti tulisi painottua näin havaittuihin ongelmakohtiin ja samoin uusien tehtävien tulisi vahvistaa havaittua puutteellista tietämystä. Caspersen (2007) antaa paljon vinkkejä, miten läpikäytävät esimerkit tulisi suunnitella.

4.4.2 Testivetoinen kehitys

Testivetoinen kehitys (test-driven development, TDD) on ohjelmistokehitysmenetelmä, jossa ennen jonkin ominaisuuden toteuttamista luodaan ominaisuuteen liittyvät testitapaukset vaatimusmäärittelyn pohjalta, ja vasta sen jälkeen toteutetaan itse ohjelma. Jälkikäteen toteutettava ohjelma tai ominaisuus on vasta sitten toimiva, kun se läpäisee tehdyt testit.

Opetuksessa TDD:tä voi soveltaa monin tavoin. Opettaja voi alkuun toteuttaa testit, ja opiskelija voi käyttää niitä oman ohjelmansa testaamiseen. Menetelmä mahdollistaa jopa tehtävien automaattisen arvioinnin (Edwards 2003). Opiskelija saa suoran palautteen ohjelmansa valmiusasteesta, ja voi sen perusteella jatkaa kehitystä. Kognitiiviselta kannalta menetelmä auttaa opiskelijaa kehittämään omia mentaalimallejaan valmiin ratkaisun toistamisen sijaan. Opettaja ei korjaa virhettä suoraan tai osoita virheellistä kohtaa koodissa, vaan oppilaan on se itse löydettävä. Tämä kehittää

virheenjäljitystaitoja. Menetelmä myös opettaa siihen ajattelumalliin, että ohjelmat on testattava, ja vasta kunnolla testattu ohjelma on valmis.

Gaspar ym. (2009) on käyttänyt TDD yhteistoiminnalliseen oppimiseen. Tehtävät ratkaistiin pareittain. Parista kumpikin toteutti tehtävään liittyvät testit ja itse tehtävän. Tämän jälkeen kumpikin testasi parinsa tekemän ohjelman itse toteuttamallaan testijoukolla. Havaitut virheet kirjattiin ja kukin parista korjasi itse virheensä koodissa. Menetelmää iteroitiin kunnes molemmat ohjelmat läpäisivät testit. Parit eivät etsineet tai korjanneet virheitä suoraan toisen koodista vaan testien kautta, jolloin korjaaminen jäi ohjelman toteuttajan tehtäväksi.

TDD:n soveltaminen vaatii opettajalta syvällistä paneutumista. Kollanus & Isomöttönen (2008) toteavat, että menetelmä pitää opettaa tarkasti jopa myös usean vuoden jo opiskelleille. Menetelmä kuulostaa varsin työläältä ja vaatii jo jonkinlaista perusohjelmointitaitoa jo pelkästään testien toteuttamisen kohdalta. Testien tekeminen on lisätyö, joka kuormittaa entisestään työläitä harjoituksia. Etukäteisvalmistelulla voidaan opiskelijan kuormaa helpottaa. Proulx (2010) on soveltanut TTD:tä usean vuoden ensimmäisillä ohjelmointikursseilla, ja on toteuttanut testien tekemiseen oman luokkakirjaston opiskelijoiden käyttöön.

4.4.3 Pariohjelmointi

Pariohjelmointi on menetelmä, jossa kaksi ohjelmoijaa työskentelee yhden näppäimistön ja hiiren kanssa. Pari noudattaa määrättyjä rooleja. Toinen on ’kuski’ (driver), jonka tehtävänä on käyttää näppäimistöä ja hiirtä, ja on siten vastuussa varsinaisesta ohjelman kirjoittamisesta koneelle. Toinen parista on ’kartturi’ (navigator). Kartturi osallistuu ohjelman kehittämiseen sanallisin keinoin. Kartturi voi olla esim. ’tarkastaja’, jolloin hän tarkkailee koodia ja sen oikeellisuutta. Kartturi voi olla myös ’esimies’, jolloin hän työskentelee hieman korkeammalla abstraktiotasolla tehden esim. ohjelman rakenteeseen kuuluvia päätöksiä ollen koko ajan ajatuksissa hieman kuskia edellä. Työnjako riippuu parin keskinäisestä sopimuksesta ja esim. aikaisemmasta kokemuksesta. Rooli voi vaihtua hyvinkin tiuhaan, jolloin näppäimistö vaihtaa käyttäjäänsä usein, tai roolit voivat olla kutakuinkin kiinteät. Ohjelma syntyy kahden henkilön kommunikaation ja yhteisvaikutuksen tuloksena.

Menetelmänä pariohjelmoinnin lähtökohdat ovat teollisuudessa, jossa sitä on hyödynnetty muodossa tai toisessa jo vuosikymmeniä. Erityisesti vaikeiden ongelmakohtien ratkaisussa se on ollut luonteva tapa työskennellä. Varsinaiseen suosioon pariohjelminti nousi ketterien menetelmien yleistymisen myötä 1990-luvulla. (Williams 2008). Tutkimuksissa on havaittu parin käyttävän tehtävien suorittamiseen tunteina hieman enemmän aikaa (4-15%), mutta toisaalta yksittäisen tehtävän suorittamiseen kuluva aika saadaan lähes puolitettua, ja myöskin lopputulos on yksin tehtyä laadukkaampi ja sisältää vähemmän virheitä.

Opetuksessa menetelmä on kiinnostava varsinkin sen yhteistoiminnallisen aspektin takia. Yhteistoiminnallisuus ja tiimityöskentely ovat usein kriittinen tekijä työelämätaidoissa nykypäivänä. Menetelmä kannustaa yhteistyöhön toisten opiskelijoiden kanssa ja se helpottaa vertaistuen käyttöä tehtävien ratkaisemisessa.

Parien muodostaminen on menetelmässä ratkaisevassa osassa. Parien valinta vapaasti ei ole välttämättä paras vaihtoehto, vaan parit kannattaa muodostaa etukäteisarvioinnin perusteella. (Williams 2008). Valintakriteerinä voi käyttää joko opiskelijoiden taitotasoa, jolloin parien tulisi olla tasoltaan suurin piirtein samanlaiset. Toinen vaihtoehto on valita työmoraaliltaan samantasoiset parit. Arviointiin riittää periaatteessa yksi kysymys, jossa opiskelijaa pyydetään vastaamaan asteikolla 1–9, kuinka paljon hän on valmis tekemään työtä tehtävänantojen eteen. Vastaus 1 vastaa vaihtoehtoa 'juuri ja juuri läpäisten' ja 9 on 'haluan parhaan mahdollisen arvosanan'. Parin tulisi vaihtaa rooleja riittävän usein ja jopa dynaamisesti oppituntien aikana. Sama pari tulisi säilyttää aina kunkin tehtävänannon ajan ja vähintään 1–3 viikkoa.

Kokeilin itse pariohjelmointia opetusharjoitteluryhmässäni. Kokeilu oli hyvin suppea, koska ryhmässäni oli vain kuusi opiskelijaa, joista kaksi alunperinkin halusi työskennellä yksin. Jäljelle jääneestä kahdesta parista molemmat halusivat siirtyä yksin työskentelyyn kurssin puolivälissä. Päällimmäisenä syynä oli, että kartturina toimessa oppiminen koettiin vähäiseksi, oppiminen liittyi työskentelyyn näppäimistöllä. Tähän pidän suurimpana syynä riittämätöntä perehdytystä menetelmään. Kuten edellä todettiin TDD-menetelmän yhteydessä, opiskelijat tulisi perehdyttää uuteen opetusmenetelmään huolella, ja myös seurata miten opiskelijat oikeasti menetelmää soveltavat. Lisäksi aiemmin omaksutut työskentelytavat voivat olla esteenä uuden oppimi-

selle. Oppilaitoksen opetuskulttuuri oli harjoittelutapauksessani hyvin erilainen, ja ohjelmointikursseilla tehtävät tehtiin pääsääntöisesti yksin. Myös oppilaitoksen omat opettajat suhtautuivat kokeiluun hieman epäillen. Suurin huolenaihe tuntui olevan vapaamatkustajat, jotka hankkisivat arvosanan parinsa kustannuksella.

Esimerkiksi Mentz ym. (2008) sisällyttivät pariohjelmointikokeiluunsa viisi selkää yhteistoiminnallisen oppimisen periaatetta. Näiden periaatteiden selkeä julkituominen työskentelyn alussa ja toteuttamisen valvonta paransivat kirjoittajien mukaan kokeilun tulosta verrattuna aikaisempiin tutkimuksiin. Oppilaat tunsivat parityöskentelyn mielekkäämmäksi eikä esim. ongelmaa 'epäreilujen' arvosanojen antamisesta esiintynyt lainkaan. Yhteistoiminnallisen oppimisen periaatteet olivat

1. Positiivinen riippuvuus: Parin molempien osapuolien tulisi ymmärtää, että he voivat onnistua ainoastaan yhdessä. Kummankin tulisi ymmärtää roolinsa ja myös hallita roolien vaihto. Parille tulisi myös tehdä selväksi, että työtä arvostellaan ja arvostelukriteerien tulee olla selvillä.
2. Luottamus yksilöön: Kummankin osapuolen tulee osallistua työhön yhtäläisesti. Yksilöllisen osallistumisen astetta tulee pystyä arvioimaan, esimerkiksi kummankin parin jäsenen tulisi pystyä kirjoittamaan toteutettu ohjelma tehtävän jälkeen myös yksin. Mikäli se ei onnistu, on toisen osallistuminen ollut selvästi heikompaa kuin toisen.
3. Kasvokkain työskentely: Parin tulee työskennellä fyysisesti lähekkäin, jolloin mahdollistetaan aito yhteistoiminta.
4. Sosiaalisten taitojen kehittäminen: Parin jäsenten tulisi kommunikoida säännöllisesti ja jatkuvasti (tehtävän suorittamisen aikana) toistensa kanssa. Vain näin he voivat kehittää keskinäistä luottamusta, antaa oikeaa kritiikkiä toisella ja myös oppia sietämään toisen kritiikkiä. Kommunikoimalla voi tuoda esille myös uusia ideoita, kannustaa kokeilemaan ja muotoilla ongelmanratkaisuja. Jotta taidot kehittyvät tasaisesti, tulisi parin jäsenten vaihtaa rooleja jatkuvasti.

5. Reflektointi: Parin tulisi seurata kehitystään säännöllisesti, ja miettiä miten suoritusta voisi vielä parantaa. Ohjaajan olisi hyvä osallistua tällaiseen itsearviointiin antaen omaa palautetta ja ohjausta.

5 POHDINTA

"Everyone can be taught to sculpt: Michelangelo would have had to be taught how not to. So it is with the great programmers."
Alan J. Perlis, 1982

Asetin alussa itselleni tavoitteeksi ymmärtää, miten ohjelmointi opitaan ja miten sitä tulisi opettaa. Pääsinkö tavoitteeseeni? Tuskin, sillä mitään lopullista totuutta asiassa ei ole olemassakaan. Mutta hieman salaisuuden verhoa tunnen pystyneeni raottamaan.

Alussa totesin, että kaikki oppiminen tapahtuu jossain oppimisympäristössä, ja sen muotoutumiseen sekä opiskelijat että opettajat vaikuttavat. Ohjelmoinnin oppiminen tuntuu kuitenkin tutkimusten valossa olevan hyvin henkilökohtainen prosessi. Tämä johtunee toisaalta aihepiirin abstraktista luonteesta ja toisaalta myös ohjelmointiin liittyvästä välineestä, tietokoneesta. Tietokoneiden käyttöliittymät on suunniteltu ennemmin henkilökohtaisiksi kuin kollaboratiivisiksi.

Motivaatiotekijät ovat keskeisessä asemassa oppimisen kannalta, ja opettajalla on keskeinen rooli motivaation luojana. Toisaalta voi ajatella, että jo jollekin alalle hakeutuminen ja kiinnostus alaan osoittaa tiettyä sisäistä motivaatiota. Ehkä opettajan rooli onkin ennemmin olla sammuttamatta orastavaa kiinnostusta. Tässä astuu kuvaan opetuksen suunnittelu siten, että opetus ei ole liian kuormittavaa, mutta kuitenkin liikutaan opiskelijan kannalta sopivasti 'lähikehityksen vyöhykkeellä'. Ohjelmointi ei siinä mielessä juuri eroa muista aloista. Koska kaikki opiskelijat ovat yksilöitä, tulisi opettajan tunnistaa opiskelijan itsesäätelyn taso ja osata säädellä omaa ohjaustaan sen mukaan.

Varsin monet läpikäymäni tutkimusraportit totesivat yhtä kantaa, että ohjelmoinnin oppiminen on vaikeaa, jatkaen samaan hengenvetoon, että ohjelmoinnin opettaminen on yhtä lailla vaikeaa. Opettajan kannalta on tärkeää tuntea ohjelmoinnin eri osa-alueet, ja suunnitella opetuksensa siten, että kurssilla harjoitetaan kaikkia alueita. Oppimisympäristöön liittyvillä valinnoilla, esimerkiksi ohjelmointikieli ja ohjelmakehitysympäristö, voidaan vaikuttaa kognitiiviseen kuormaan, ja välttää ylikuormitusta

opintojen alkuvaiheessa.

Ohjelmistoala elää jatkuvassa kehityksessä ja siitä syystä alalla työskentelevät ovat armoitettuja elinikäisiä oppijoita – halusivat sitä tai eivät. Yritysmaailmakin joutuu painiskelemaan työntekijöidensä kouluttamisen kanssa. Uutta on opittava ja samalla tehtävä työ tehokkaasti. Useimmiten työ opettaa parhaiten. Siksi työelämään tarvitaan oppimisvalmiita työntekijöitä.

Tästä syystä ohjelmistoalalla kehitetään paljon erilaisia käytäntöjä ja menetelmiä, joista jotkut löytävät tiensä yritysmaailmasta koulumaailmaan opetusmenetelmiksi. Tällaisia ovat mm. edellä käsitellyt pariohjelmointi ja testivetoinen kehitys. Näiden menetelmien käyttö opetuksessa on työelämälähtöisyyttä parhaimmillaan, kun samalla opitaan työelämän hyväksi havaittuja käytäntöjä. Samalla voidaan opetukseen tuoda yhteistoiminnallisuutta, joka on tärkeä taito työelämässä.

Työ on opiskelujen aikanakin paras opettaja. Ilman jatkuvaa käytännön harjoittelua ja itse tehtyjä harjoituksia ei ohjelmointia opi kukaan. Pintasuuntautuneella ulkoaoppimiseen tähtäävällä opiskelustrategialla ei kovin pitkälle pääse. Suuri haaste opiskelujen alkuvaiheessa onkin saada opiskelijat omaksumaan se tarkkuuden ja systemaattisuuden taso, jonka ohjelmointi ja sen opiskelu vaatii.

LÄHTEET

Ala-Mutka, K. 2005. Ohjelmoinnin opetuksen ongelmia ja ratkaisuja. Teoksessa ReflekTori 2005, Tekniikan opetuksen symposium, Toim. A. Yanar. Espoo: Teknillinen Korkeakoulu.

Ben-Ari, M. 2001. Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching 20, 1, 45-73.

BlueJ – The interactive Java environment. Viitattu 22.11.2010. http://www.bluej.org/.

Boyer, N. , Langevin, S. , and Gaspar, A. 2008. Computer Programming meets Education: The Intersection of Information Technology and Self-Directio. Teoksessa 22nd Annual International Self-Directed Learning Symposium, Courtyard by Marriott at Cocoa Beach , Florida:

Caspersen, M. E. and Bennedsen, J. 2007. Instructional design of a programming course: a learning theoretic approach. Teoksessa ICER '07: Proceedings of the third international workshop on Computing education research, New York, NY, USA: ACM, 111-122.

Edwards, S. H. 2003. Rethinking computer science education from a test-first perspective. Teoksessa Companion of the 18th annual ACM SIGPLAN conference on Object--oriented programming, systems, languages, and applications, New York, NY, USA: ACM, 148--155.

Galpin, V. C. , Sanders, I. D. , and Chen, P.-Y. 2007. Learning styles and personality types of computer science students at a South African university. Teoksessa ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, New York, NY, USA: ACM, 201-205.

Gaspar, A. , Langevin, S. , & Boyer, N. 2009. Encyclopedia of Information Science and Technology. 708-714.

Haapasalo, L. 1994. Oppiminen, tieto ja ongelmanratkaisu. Jyväskylä: Medusa-Software.

Kalakoski, V. 2007. Muistikirja. Edita Publishing Oy.

Kivi, T. 2000. Oppimisen Taidot. Opetushallitus.

Lahtinen, E. , Ala-Mutka, K. , Järvinen, H.-M. 2005. A study of the difficulties of novice programmers. SIGCSE Bull. 37, 3, 14-18.

Lindblom-Ylänne, S. & Nevgi, A. 2002. Yliopisto- ja korkeakouluopettajan käsikirja. WSOY.

Mannila, L. and De Raadt, M. 2008. An Objective Comparison of Languages for Teaching Introductory Programming. Teoksessa Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research, New York, NY, USA: ACM, 32-37.

Mancy, R. and Reid, N. 2004. Aspects of Cognitive Style and Programming. Teoksessa Proceedings of 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG), 1-9.

Mentz, E. , Van Der Walt, J. L. , & Goosen, L. 2008. The effect of incorporating cooperative learning principles in pair programming for student teachers. Computer Science Education 18, 4, 247-260.

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

Perlis, A.J. 1982, Epigrams on Programming, SIGPLAN Notices 17, 9, 7 -13.

Proulx, V. K. 2010. Test-first design pedagogy and support for introductory OO programming: tutorial presentation. J. Comput. Small Coll. 25, 70-72.

Raij, K. 2003. Osaamisen tuottaminen ammattikorkeakoulun päämääränä, Teoksessa Ammattikorkeakoulupedagogiikka, Toim. Kotila H.. 42-58.

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

Rytkönen, H. 2009. Oppiminen opintomenestyksen takana. Pro Gradu -työ. Helsingin yliopisto, Kasvatustieteen laitos.

Shackelford, R. , Mcgettrick, A. , Sloan, R. , Topi, H. , Davies, G. , Kamali, R. , Cross, J. , Impagliazzo, J. , Leblanc, R. , & Lunt, B. 2006. Computing Curricula 2005: The Overview Report. SIGCSE Bull. 38, 456-457.

Shaffer, S. C. 2005. A brief overview of theories of learning to program. PPIG Newsletter.

Starr, C. W. , Manaris, B. , & Stalvey, R. H. 2008. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. SIGCSE Bull. 40, 1, 261-265.

Tanenbaum, A. S. 1997. Ten golden rules for teaching computer science, ACM SIGCSE 1997, keynote talk, Viitattu 15.11.2010, http://www.few.vu.nl/~ast/talks/cse-97/cse-97.pdf.

White, G. & Sivitanides, M. 2005. Cognitive Differences Between Procedural Programming and Object Oriented Programming. Information Technology and Management 6, 4, 333--350.

Williams, L. 2008. Pair Programming. Taylor and Francis Group.

Winslow, L. E. 1996. Programming pedagogy – a psychological overview. SIGCSE Bull. 28, 3, 17-22.