# Kynnyskäsitteet ohjelmoinnin oppimisessa

Pessi Moilanen

Pro gradu -tutkielma

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 27. marraskuuta 2017

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

| Tiedekunta — Fakultet — Faculty | Laitos — Institution — Department |
|---|---|
| Matemaattis-luonnontieteellinen | Tietojenkäsittelytieteen laitos |

Tekijä — Författare — Author
Pessi Moilanen

Työn nimi — Arbetets titel — Title
Kynnyskäsitteet ohjelmoinnin oppimisessa

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

| Työn laji — Arbetets art — Level | Aika — Datum — Month and year | Sivumäärä — Sidoantal — Number of pages |
|---|---|---|
| Pro gradu -tutkielma | 27. marraskuuta 2017 | 66 |

Tiivistelmä — Referat — Abstract

Tutkimuksen tavoitteena oli tutkia kynnyskäsitteitä (threshold concepts) ja tunnistaa niitä ohjelmoinnin oppimisessa. Kynnyskäsitteet ovat oppimisen kannalta tärkeitä paalukiviä, ja ne toimivat portteina uudenlaiseen kehittyneempään ajatteluun tietyssä aihealueessa. Ne liittyvät esimerkiksi johonkin opittavaan taitoon, tietoon tai asiaan. Kynnyskäsitteiden omaksuminen mahdollistaa laajentuneen ymmärryksen jostakin aihealueesta, koska ne sitovat useita eri käsitteitä yhteen luoden niiden välille syvemmän yhteyden. Jos opiskelija ei omaksu kynnyskäsitettä, saattaa hänen oppimisensa pysähtyä tai hän saattaa kehittää väärinkäsityksiä siihen liittyen.

Kynnyskäsitteiden tutkiminen on tärkeää, koska se mahdollistaa tehokkaamman tavan tunnistaa ja ennaltaehkäistä tilanteita, joissa opiskelija jää jumiin aihealueessa. Kynnyskäsitteiden avulla pystytään rakentamaan uudenlaisia opetussuunnitelmia, jotka keskittyvät kynnyskäsitteiden ympärille.

Tietoa mahdollisista kynnyskäsitteistä kerättiin kirjallisuuskatsauksella. Tämän lisäksi tehtiin empiiristä tutkimusta, jossa sovellettiin puolistrukturoitua haastattelua. Tutkimusta varten haastateltiin Helsingin yliopiston opiskelijoita, jotka olivat suorittaneet ohjelmoinnin perus- ja jatkokurssit. Haastatteluista tehtiin yhteenveto, joka sisälsi analyysin sekä listan käsitteistä, joihin liittyi haasteellisuutta, muuttavia kokemuksia tai väärinkäsityksiä. Yksi tutkimuksen tavoitteista oli löytää kurssien aikana ilmaantuvia mahdollisia kynnyskäsitteitä ja verrata niitä olemassaolevaan tutkimukseen.

Tutkimuksesta saadut tulokset vahvistivat olettamusta siitä, että ohjelmointiin liittyy useita mahdollisia kynnyskäsitteitä. Omasta empiirisestä tutkimuksesta saatuja käsitteitä olivat muun muassa olio-ohjelmointi, algoritminen ajattelu, perintä ja rajapinnat. Kaikki nämä käsitteet ovat myös aiemmissa tutkimuksissa esitetty kynnyskäsitteiksi. Kynnyskäsitteitä on hankala tunnistaa, sillä ne ovat määrittelyltään epämääräisiä. Kynnyskäsitteitä on tutkittu laajasti, mutta tarkempaa tietoa niiden soveltamisesta opetuksessa ei löydy tutkimuksista.

ACM Computing Classification System (CCS):
**Social and professional topics → Computer science education**

Avainsanat — Nyckelord — Keywords
kynnyskäsite, käsitteellinen muutos, tietojenkäsittelytiede, opetus, tehostettu oppipoikamalli

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

# Sisältö

# 1 Johdanto

Tämän tutkielman tarkoituksena oli selvittää, miten kynnyskäsitteet (threshold concepts) ilmenevät ohjelmoinnin oppimisen yhteydessä. Käsite (concept) on kielellinen ilmaisu jollekin asialle, ilmiölle, abstraktiolle tai yleistykselle. Käsitteet toimivat rakennuspalikoina tiedolle, ja niitä käytetään tiedon jäsentämiseen. Meyer ja Land ovat määritelleet kynnyskäsitteen käsitteenä, joka esittää muuttunutta tapaa ymmärtää, tulkita tai tarkastella jotakin ilmiötä, jota omaksumatta oppija ei pysty etenemään aihealueessa [ML03]. Kynnyskäsitteiden tutkiminen ja niiden tunnistaminen tarjoaa uudenlaisen lähestymistavan ohjelmoinnin oppimiseen ja sen opettamiseen [SM16].

Ideat kynnyskäsitteistä kehitettiin, kun mietittiin kuinka yliopistotason opetusta tietyissä aihealueissa pystyttäisiin parantamaan [SM16]. Huomattiin, että ne ovat monesti haasteellisia omaksua ja ne toimivat samalla kynnyksenä uusien asioiden oppimiselle [ML03]. Kynnyskäsitteet ovat oppimisen kannalta tärkeitä paalukiviä ja ne voidaan nähdä portteina uudenlaiseen ajatteluun [ML03]. Niiden omaksuminen mahdollistaa uusien käsitteiden oppimisen aihealueesta, sillä niiden ymmärtämisen seurauksena ajattelu ja ymmärrys asioista muuttuu huomattavasti (conceptual change) [ML03]. Teoria kynnyskäsitteistä on melko uusi ja se pohjautuu konstruktiiviseen oppimisteoriaan (constructive learning theory), jonka mukaan tiedon rakentuminen tapahtuu kokemusten ja havaintojen perusteella henkilön aiemman tiedon päälle. Ohjelmoinnin oppimiseen liittyvistä kynnyskäsitteistä on myös aiemmin tehty tutkimusta, minkä analysointiin tässä tutkielmassa paneudutaan.

Oppimisen näkökulmasta kynnyskäsitteitä voidaan tarkastella seuraavan esimerkin kautta: tarkastellaan tilannetta, kuinka pieni lapsi oppii tunnistamaan ihmiset muista elollisista olennoista. Jossakin vaiheessa kehitystään lapsi havaitsee ja muodostaa käsitteellisiä rakenteita seuraavista ihmiseen liittyvistä käsitteistä: ulkonäkö, liikkuminen sekä tapa äännellä ja käyttäy-

tyä. Näistä käsitteistä jokin saattaa toimia mahdollisena kynnyskäsitteenä, jonka avulla lapsi kykenee muodostamaan laajemman ymmärryksen siitä, kuinka tunnistaa ihmiset muista eläimistä. Kynnyskäsitteiden omaksuminen muuttaa lapsen ajattelutapaa pysyvästi ja luo lapselle uudenlaisen tavan tarkastella ympäröivää maailmaa. Kun lapsi sisäistää lisää käsitteitä ja yhdistää ne aiempaan tietämykseensä, pystyy hän tunnistamaan erot selkärangallisten ja selkärangattomien eläimien välillä. Käsitteet ja niihin liittyvä ymmärrys kehittyvät uuden tiedon myötä ja hiljalleen muokkaavat ja luovat ihmisen tietämystä.

Tutkielman empiirisen osion tarkastelukohteena ovat Helsingin yliopiston tietojenkäsittelytieteen laitoksella ohjelmoinnin perus- ja jatkokurssit suorittaneet opiskelijat ajankohtana, jolloin tehostettu oppipoikamalli (extreme apprenticeship) on ollut käytössä. Tehostetussa oppipoikamallissa yhdistyy kolme peruselementtiä: perinteiset luennot, aktiivinen ongelmien ratkaiseminen ja oikein ajoitettu tuki oppimiselle (scaffolding). Nämä kurssit toimivat alkuaskeleena ohjelmoinnin oppimiseen Helsingin yliopistolla ja ne ovat tärkeimmät perusopintojen kurssit opintojen alkuvaiheessa. Molemmat kursseista ovat viiden opintopisteen arvoisia ja kestävät yhden periodin. Kurssien aikana opiskelijat oppivat Java-ohjelmointikieltä, jota hyödynnetään myös opintojen jatkuessa.

Ohjelmointiin liittyvän ajattelutavan merkitys kasvaa, koska siihen sidottu tekniikka kehittyy eteenpäin jatkuvasti. Kaikkien ei tarvitse osata ohjelmoida, mutta ohjelmointiin liittyvää ajattelutapaa (computational thinking) on hyödyllistä omaksua, koska ohjelmointi ja tietokoneet ovat lopulta vain eräänlaisia työkaluja. Ohjelmointi on samanlaista perusosaamista kuin lukeminen, kirjoittaminen tai laskuoppi [Win06]. Ohjelmointiin liittyvä ajattelutapa auttaa ymmärtämään ympärillämme esiintyviä systeemejä sekä niihin liittyvää toimintalogiikkaa [Win06].

Perinteiset opetusmenetelmät pohjautuvat behaviorismiin. Behaviorismis-

sa oppiminen nähdään reaktiona ympäristöstä tuleviin ärsykkeisiin. Kyseisessä ajattelumallissa ei oteta huomioon opiskelijoiden aiempaa osaamista tai kykyä ymmärtää opetettavaa asiaa, vaan keskitytään ennalta määriteltyyn opetukseen, jossa olennaista on tiedon siirtäminen opettajalta opiskelijalle [Vag06]. Oppiminen nähdään siinä yksittäisten tiedon palasten ja taitojen vastaanottamisena. Perinteisessä tavassa opettaa käsitteet käydään läpi ennalta suunnitellussa järjestyksessä olettaen, että opiskelijat ovat jo omaksuneet uusiin käsitteisiin liittyvän perustiedon ja niihin liittyvän käsitteellisen osaamisen [Vag06].

Ohjelmoinnin oppiminen on haasteellista, sillä se vaatii uudenlaisen ajattelutavan omaksumista ja ohjelmointiin liittyvää ajattelutapaa on hankala yhdistää aiempaan tietämykseen [GM10]. Ohjelmointiin on hankala päästä sisälle, koska se ei ole yksittäinen opeteltava taito, vaan se on monimutkaista kognitiivista toimintaa [RR09]. Ohjelmointia oppii parhaiten tekemällä ja sitä voidaan ajatella nykyajan kädentaitona.

Ohjelmoinnin opettamiseen käytettäviä erilaisia opetusmenetelmiä on syytä tutkia. Monissa yliopistoissa perus- ja jatkokurssien opetuksessa käytetään edelleen pelkästään perinteisiä opetusmenetelmiä, jotka eivät välttämättä toimi optimaalisesti ohjelmoinnin opettamisen yhteydessä [MGH+06]. Huolta nykyisten opetusmenetelmien toimivuuden suhteen aiheuttaa se, että keskeytysprosentti ohjelmoinnin perus- ja jatkokursseilla (CS1) on ollut kansainvälisesti noin 33% [BC07]. Kynnyskäsitteiden ja niihin liittyvän tutkimuksen toivotaan antavan uusia työkaluja ohjelmoinnin opetukseen.

## 1.1 Tutkimuskysymykset

Tässä osiossa käsitellään tutkielman tutkimustavoitteet ja tutkimuskysymykset, jotka antavat yleiskuvan tutkielman sisällölle. Ensimmäinen tutkimuskysymys käsittelee teoriaa kynnyskäsitteistä ja käsitteellisestä muutoksesta, jotka ovat oleellisia teoriapohjan ja tutkielman empiirisen osion kannalta. Toinen tutkimuskysymys on, mitkä ohjelmointiin liittyvistä käsitteistä ovat mahdollisia kynnyskäsitteitä ohjelmoinnissa. Kolmannessa tutkimuskysymyksessä selvitetään sitä, mitkä käsitteet ovat mahdollisia kynnyskäsitteitä Helsingin yliopiston ohjelmoinnin perus- ja jatkokursseilla käytetyn opetusmenetelmän ollessa tehostettu oppipoikamalli (extreme apprenticeship).

**RQ1: Miten kynnyskäsitteet näkyvät olemassaolevassa tutkimuksessa?** Kynnyskäsitteiden piirteitä ja niiden ominaisuuksia analysoidaan olemassaolevaan tutkimukseen perustuen. Tarkoituksena on selvittää näistä tutkimuksista, kuinka kynnyskäsitteet vaikuttavat yleisesti oppimiseen, ja miten ne ilmenevät oppimisen yhteydessä.

**RQ2: Mitkä ovat mahdollisia kynnyskäsitteitä ohjelmointia opittaessa?** Ohjelmoinnin oppimiseen liittyy paljon vaativaa ajattelua, jossa erilaisia kognitiivisia rakenteita on hyödynnettävä samanaikaisesti. Ohjelmoinnin perus- ja jatkokursseilla uusia käsitteitä tulee valtava määrä, ja niiden omaksumiseen ja niiden välillä vallitsevien yhteyksien rakentamiseen tarvitaan paljon työtä. Kaikki ohjelmointiin liittyvät käsitteet eivät kuitenkaan ole opiskelijoille kynnyskäsitteitä ja ne poikkeavat eri henkilöillä toisistaan. Aiempia tutkimuksia analysoimalla pyritään saamaan käsitys yleisimmistä ohjelmointiin liittyvistä kynnyskäsitteistä, sekä millaisia haasteita niiden oppimiseen liittyy. Kynnyskäsitteiden tunnistaminen ja niihin liittyvien piirteiden analysointi antaa erilaisia näkökulmia opetuksen tehostamiseen.

**RQ3: Mitkä käsitteet ovat mahdollisia kynnyskäsitteitä ohjelmoinnin perus- ja jatkokursseilla?** Tutkimuksen empiirisen osan kohteena ovat Helsingin yliopiston tietojenkäsittelytieteen laitoksella ohjelmoinnin perus- ja jatkokurssin suorittaneet opiskelijat. Tutkimuksessa tarkastellaan opiskelijoiden kynnyskäsitteitä ja miten niiden omaksumisesta syntyvät käsitteelliset muutokset vaikuttavat ymmärrykseen ohjelmoinnista. Mielenkiintoista on selvittää, ovatko tutkimuksessa esiintyvät kynnyskäsitteet ja niihin liittyvät havainnot samoja kuin olemassaolevissa tutkimuksissa.

## 1.2 Laajuus ja aihealueen rajaus

Tämä tutkielma keskittyy kynnyskäsitteisiin ja niihin liittyviin ilmiöihin ohjelmoinnin oppimisen yhteydessä. Tutkielman empiirinen osio pohjautuu haastatteluihin, jotka tehtiin Helsingin yliopiston tietojenkäsittelytieteen laitoksen opiskelijoille. Tehtyä tutkimusta analysoidaan ja vertaillaan jo olemassaolevaan tutkimukseen kynnyskäsitteistä. Haastatteluiden kohteena olivat opiskelijat, jotka olivat suorittaneet ohjelmoinnin perus- ja jatkokurssit vuonna 2010 tai myöhemmin. Opiskelijoiden opintojen aloitusvuosi oli tutkimuksen kannalta olennainen, sillä kyseisten kurssien opetuksessa on ollut vuoden 2009 jälkeen käytössä tehostettu oppipoikamalli. Tehostettu oppipoikamalli poikkeaa yleisesti käytetyistä opetusmenetelmistä ja se saattaa vaikuttaa opiskelijoiden kynnyskäsitteiden erilaiseen kehittymiseen.

Rajoituksia tutkimuksessa aiheuttavat kynnyskäsitteiden epämääräinen määrittely kirjallisuudessa, joka tekee niiden tunnistamisesta ja tarkastelemisesta hankalaa. Yksittäisten kynnyskäsitteiden laajuudesta on käyty myös paljon keskustelua, sillä kynnyskäsitteet sitovat useita eri käsitteitä yhteen. Kynnyskäsitteet eivät myöskään ole kaikille henkilöille samat, joten tavoitteena on löytää käsitteitä, jotka omaavat kynnyskäsitteiden piirteitä.

Tutkielman rakenne on seuraava. Ensimmäinen luku sisältää tutkielman

johdannon, yleiskuvan ja aihealueen rajauksen. Toinen luku sisältää opetus- ja oppimisteorioita, jotka toimivat teoreettisena pohjana kynnyskäsitteille. Kolmas luku sisältää kynnyskäsitteisiin ja käsitteelliseen muutokseen liittyviä määritelmiä, jotka ovat tutkittavan aihealueen kannalta tärkeitä. Neljäs luku sisältää tietoa ohjelmoinnin oppimisesta ja opettamisesta; siinä käydään läpi ohjelmoinnin perus- ja jatkokurssien opetussisällöt ja esitellään aiemmissa tutkimuksissa tunnistettuja kynnyskäsitteitä. Nämä neljä lukua sisältävät teoreettisen pohjan tutkielmalle, ja niissä kasataan tietoa olemassaolevista tutkimuksista kynnyskäsitteisiin liittyen. Viides luku sisältää empiiriseen tutkimukseen käytettyjen menetelmien esittelyn ja tutkimusdatan esittelyn. Kuudes luku sisältää empiirisestä tutkimuksesta kerätyt tulokset ja pohdinnan. Seitsemäs luku kattaa yhteenvedon kynnyskäsitteistä ja tehdystä tutkimuksesta.

# 2 Opetus- ja oppimisteorioita

Tässä luvussa esitellään kynnyskäsitteiden kannalta tärkeitä opetus- ja oppimisteorioita. Konstruktiivisen oppimisteorian mukaan käytetyt opetusmenetelmät vaikuttavat oppilaiden tietämyksen muodostumiseen ja käsitteellisen ymmärryksen kehittymiseen. Teoria kynnyskäsitteistä pohjautuu konstruktiiviseen oppimisteoriaan, minkä takia opetus- ja oppimisteorioiden tarkastelu on tärkeää.

## 2.1 Konstruktiivinen oppimisteoria

Konstruktiivinen oppimisteoria (constructivist learning theory) on filosofinen paradigma. Sen mukaan ihminen ei ole tyhjä taulu (tabula rasa), vaan ihmisellä on jo syntyessään toimintamalleja, jotka vahvistuvat ja muovautuvat yksilöllisten kokemusten ja ympäristöllisten tekijöiden kautta. Konstruktiivisessa oppimisteoriassa oppiminen tulkitaan henkilökohtaisten kognitiivisten

rakenteiden kehityksenä. Tiedon rakentaminen tapahtuu hankkimalla uutta tietoa esimerkiksi kirjoista, tehtäviä tekemällä tai käymällä luennoilla [EMM+06].

Taito rakentaa uutta tietoa ja ymmärtää opetusta riippuu henkilön olemassaolevasta tiedosta ja kognitiivisista rakenteista [MGH+06]. Tämä uusi tieto synnyttää aiemman tiedon avulla ennakkokäsityksiä (preconceptions), jotka toimivat perustana uudelle tiedolle [Vag06]. Kun henkilö oppii jotakin uutta, hän muuttaa tai laajentaa olemassaolevia kognitiivisia rakenteita kehittyneiden ennakkokäsitysten avulla [EMM+06]. Erot kognitiivisissa rakenteissa selittävät sen, miksi eri henkilöt saattavat oppia opetetun asian eri tavoilla, vaikka opiskeluun käytetyt materiaalit ovat kaikille samat.

Tietoa voidaan omaksua eri tavoilla. Sitä voidaan muokata olemassaolevan tietämyksen yhteyteen (assimilation) [MGH+06]. Tässä tilanteessa omaksuttu tieto löytää perustan henkilön aiemmasta tietämyksestä ja mukautuu siihen. Joskus tapahtunut oppimiskokemus pystyy olemaan niin erilainen aiempiin kokemuksiin verrattuna, ettei sitä pysty mukauttamaan, vaan se pitää sopeuttaa (accomodation) [MGH+06]. Tällöin syntyy täysin uusi kognitiivinen rakenne. Jos henkilöllä ei ole tarpeeksi kehittyneitä kognitiivisia rakenteita uuteen tietoon liittyen, ei tiedon omaksuminen ole mahdollista tai tämän seurauksena saattaa kehittyä oppimista vaikeuttavia väärinkäsityksiä [Vag06].

Opettamisesta tekee haasteellisesta se, että opiskelijoilla on kursseille mentäessä toisistaan poikkeavia kognitiivisia rakenteita aihealueesta. Opettajien on otettava huomioon opiskelijoiden kognitiivisten rakenteiden eroavaisuudet, eli tiedon rakentumisen pitää tapahtua opiskelijan aiemman tiedon kontekstissa. Opettaessa tulee auttaa ja aktivoida opiskelijoita uuden tiedon rakentamisessa passiivisen tiedon siirtämisen sijaan. Opettajien on myös huomioitava, että heidän omat kognitiiviset rakenteensa ovat paljon korkeammalla tasolla aihealueessa kuin opiskelijoiden [MGH+06].

## 2.2 Skeemateoria

Skeemateoria (schema theory) pohjautuu konstruktiiviseen oppimisteoriaan; siinä henkilön tietämys nähdään kognitiivisina rakenteina eli skeemoina (schemas), jotka auttavat omaksumaan, organisoimaan ja ymmärtämään tietoa. Skeemat ovat abstraktioita joukosta kokemuksia ja niitä on mahdollista soveltaa aihealueeseen liittyvässä kontekstissa [MGH+06]. Skeemat muodostuvat vaiheittain, ja niissä yhdistyvät uudesta kokemuksesta hankittu informaatio ja henkilökohtaiset aiemmat kokemukset. Skeemoja ei ajatella pelkästään hierarkkisen tiedon muodostumisena, vaan ne ovat abstraktioita, jotka pitävät sisällään tiedon tarkoituksen ja riippuvuudet. Osa skeemoista vaatii ajattelua aktivoituakseen, kuten isojen numeroiden kertolasku. Tällaisessa tilanteessa henkilö joutuu käyttämään osaa työmuististaan skeemojen aktivointiin. Jotkut skeemat tulevat puolestaan täysin alitajunnasta eivätkä ne vaadi ylimääräistä vaivannäköä toimiakseen, esimerkiksi käveleminen tai puhuminen [MGH+06].

Driscoll [Dri05] listaa kolme eri mekanismia, kuinka skeemoja pystyy muodostumaan ja kehittymään uusien kokemusten kautta. Kasvu (accretion): uusi kokemus sopii hyvin olemassaolevaan skeemaan. Kokemus muistuttaa jotakin aiempaa skeemaa niin hyvin, ettei tätä olemassaolevaa skeemaa tarvitse muokata. Virittäminen (tuning): uutta kokemusta ei pysty ymmärtämään täysin olemassaolevan skeeman perusteella, joten skeema kehittyy ja muokkautuu sopimaan paremmin uuteen kokemukseen. Uudelleenrakentaminen (restructuring): uusi kokemus on tarpeeksi poikkeava, eikä minkään aiemman skeeman sisältö sovellu käsiteltävän tiedon tallentamiseen. Tällöin on muodostettava täysin uusi skeema tiedon hallitsemiseksi. Yleisin reaktio tilanteissa, joissa uusi tieto ei yhdisty aiempiin skeemoihin tarpeeksi hyvin, on tiedon käsittelemättä jättäminen tai nopea unohtaminen.

Skeemat selittävät eroavaisuudet ammattilaisten ja aloittelijoiden välillä. Ammattilaisilla on pitkälle kehittyneitä skeemoja, jotka hoitavat tiedon

järjestämistä ja käsittelemistä. Ammattilaiset pystyvät tunnistamaan yhtäläisyyksiä aiemmista kokemuksistaan ja hyödyntämään oikeita skeemoja ongelman ratkaisuun. Aloittelijan on puolestaan turvauduttava usein yritykseen ja erehdykseen, koska hänen skeemansa eivät ole niin kehittyneitä ja sisältävät vain hajanaista tietoa eri aihealueista. Aiemman tietämyksen ollessa hajanaista on henkilön turvauduttava moniin eri skeemoihin ja kokeiltava useita eri lähestymistapoja ongelman ratkaisemiseksi. [MGH+06]

## 2.3 Kognitiivinen kuormitus

Kognitiivinen kuorma (cognitive load theory) on teoria siitä, kuinka oppiminen tapahtuu ja miten uusi tieto omaksutaan pysyvästi. Sen mukaan ihmisillä on kolme piirteiltään erilaista muistia. Aistimuisti (sensory) pitää tallessa nähtyä tietoa noin 0,5 sekuntia ja kuultua tietoa noin 3 sekuntia. Työmuisti (working) saa uutta tietoa aistimuistista ja soveltaa siihen säilömuistista (long-term) löytyvää tietoa. Työmuistissa tapahtuu aktiivista ja kuormittavaa tiedon prosessointia. Työmuistilla on pieni kapasiteetti ja se pystyy käsittelemään ainoastaan 4-9 asiaa samanaikaisesti. Säilömuisti on puolestaan kapasiteetiltaan lähes rajoittamaton tila ja sinne tallentuu työmuistista siirretty käsitelty tieto. [MGH+06]

Työmuistin kuormitukseen vaikuttaa seuraava kolmen piirteen summa [MGH+06]. Tiedon luontaisuus (intrinsic) pitää sisällään opetettavan aihealueen vaikeuden. Jos aihealue on haastava ja siinä on paljon toisiinsa sidottuja käsitteitä, vaikuttaa se kuormittavasti työmuistiin, sillä opiskelijan on otettava eri asioita huomioon kerralla ja mietittävä niiden välisiä yhteyksiä. Oppimisen luontaisuuteen ei pystytä kunnolla vaikuttamaan pedagogisilla valinnoilla, sillä jotkin käsitteet on pakko sitoa toisiinsa opetettavassa kontekstissa. Tiedon asiaankuulumattomuus (extraneous) eli opetettavan asian kannalta epäolennaisen tiedon määrä, jonka erottamiseen tarvitaan työmuistin käyttöä. Epäolennainen tieto kuormittaa opiskelijoita turhaan,

sillä he joutuvat käyttämään työmuistiaan tunnistaakseen olennaisen tiedon epäolennaisesta. Epäolennaiseen tietoon pystytään vaikuttamaan karsimalla opetuksesta ja materiaaleista aihealueeseen liittymätöntä tietoa. Tiedon olennaisuus (germane) eli kuinka paljon työmuistia tarvitaan prosessoimaan olennaista tietoa skeeman muodostamiseksi.

Opetuksessa on otettava huomioon, kuinka uusia käsitteitä opetetaan ja kuinka ne sidotaan asiayhteyteen, sillä työmuistin ylirasittuessa suuresta kognitiivisesta kuormituksesta taito prosessoida tietoa rajoittuu ja oppiminen on tehottomampaa [MS05]. Tiedon olennaisuus on pyrittävä pitämään suurena, kun taas tiedon asiaankuulumattomuus on pidettävä vähäisenä. Opetettavan asian olennaisuuteen ja asiaankuulumattomuuteen voidaan vaikuttaa muokkaamalla materiaaleja ja esimerkkejä, hyödyntämällä oppimisen oikeaaikaista tukemista ja ajoittamalla uusien käsitteiden opettamista paremmin [MGH+06]. Pedagogisilla valinnoilla pystytään vaikuttamaan kognitiivisen kuormituksen määrään.

Opettajille vaikeuksia tuottavat tilanteet, joissa opiskelijoilla on hankaluuksia yhdistää uudet käsitteet aiempaan tietoon. Nämä tilanteet aiheuttavat opiskelijoille paljon kognitiivista kuormitusta, mikä hankaloittaa skeemojen muodostumista [MGH+06]. Tätä ongelmaa korostaa opetuksessa se, että opettajat ovat sisäistäneet asian ymmärtämiseen vaaditut skeemat. He eivät kuitenkaan aina osaa ottaa huomioon, että opiskelijoilla näitä skeemoja ei ole vielä muodostunut. Opetuksessa on onnistuttava löytämään opetettavan asian kannalta tärkeimmät pääideat ja pyrittävä luomaan niistä opiskelijoille skeemoja, joiden avulla uutta tietoa on helpompaa omaksua.

## 2.4 Tehostettu oppipoikamalli

Konstruktiivisen oppimisteorian mukaan käytetyillä opetusmenetelmillä on vaikutusta oppimiseen ja sen takia tutkielman empiirisessä osiossa otetaan huomioon opetuksessa käytetyt opetusmenetelmät. Tässä osiossa selitetään,

mitä ovat kognitiivinen oppipoikamalli (cognitive apprenticeship) ja sen laajennus tehostettu oppipoikamalli (extreme apprenticeship).

Kognitiivinen oppipoikamalli (cognitive apprenticeship) on teoria prosessista, jossa mestari opettaa jotakin taitoa oppipojalle [BCN89]. Kognitiivisessa oppipoikamallissa on suuri painotus työskentelyssä ja sen eri vaiheissa pelkän lopputuotteen valmistumisen sijaan [VPL11]. Kyseisen opetusmenetelmän on todettu vaikuttavan myönteisesti opiskelijan opiskelumotivaatioon ja opiskelumukavuuteen, jotka puolestaan vaikuttavat huomattavasti oppimiseen [VPLK11]. Kyseinen malli soveltuu hyvin asioiden opettamiseen opiskelujen varhaisessa vaiheessa, koska sen avulla pystytään syventymään opetettaviin aihealueisiin [Bla06, LMRC13]. Se auttaa opiskelijoita rakentamaan kokemusta sekä parempaa ymmärrystä käsitteistä, ja samalla heistä tulee itsenäisempiä verrattuna perinteisiin opetusmenetelmiin [Bla06]. Siinä käsitellään opettamista tilanteissa, joissa työnvaiheet opetettavasta asiasta eivät usein ole konkreettisesti näkyvillä oppipojalle. Ohjelmoinnin näkökulmasta tällaisia asioita voivat olla esimerkiksi yleinen ongelmanratkaisu, oikean algoritmin valinta tai luetun koodin ymmärtäminen. Oppipojan on vaikea oppia mestarilta, jos ainoa näkyvä työnvaihe on ratkaisun ilmestyminen paperille ilman vaadittua ajatusprosessia; tämän takia mestarin on saatava ajatusprosessinsa näkyviin oppipojalle [CBH91]. Ajatusten näkyville tuontiin on erilaisia opetusmenetelmiä, joita voidaan soveltaa tilanteesta riippuen.

Kognitiiviseen oppipoikamalliin on määritelty kuusi opetusmenetelmää, jotka ovat mallintaminen (modelling), valmentaminen (coaching), oppimisen oikea-aikainen tukeminen (scaffolding), artikulointi (articulation), peilaaminen (reflection) ja tutkiminen (exploration). Menetelmät tukevat ja hyödyntävät toisiaan sekä ovat vahvasti sidoksissa keskenään. [CBH91]

Mallintamisessa suoritettavan työn eri vaiheet pyritään saamaan opiskelijalle näkyviin. Mallintamisen suorittaa yleensä opettaja tai joku muu

oppilasta kokeneempi henkilö ja se tehdään tavalla, jossa työn vaiheet havainnollistetaan oppilaan ymmärrettävällä tasolla. Oppilas pyrkii muodostamaan käsitteellisen mallin suoritetuista työn vaiheista ja hän näkee työn eri vaiheiden merkityksen sen valmistumisen kannalta. Mallintamisen tavoitteena on, että oppilas saa käsityksen suoritettavan työn eri vaiheista, jotka johtavat onnistuneeseen lopputulokseen. Esimerkiksi jotakin ongelmanratkaisuprosessia mallintaessaan mestari kertoo jokaisessa työn vaiheessa ääneen ajatteluprosessistaan: mitä hän tekee ja miksi. Tämän prosessin avulla oppilas saa ymmärrystä mestarilta ongelmanratkaisuprosessin eri vaiheista ja tarkoituksista. [CBH91]

Valmentamisessa opettaja tarkkailee oppilaan työskentelyä, antaa hänelle henkilökohtaisia neuvoja ja pyrkii avustamaan oppilasta kriittisillä hetkillä. Opettaja pyrkii samaistumaan oppilaan osaamiseen ja pyrkii ohjaamaan tätä oikeaan suuntaan. Opettaja suunnittelee ja rakentaa oppilaalle tehtäviä, jotka ovat oppilaan osaamistasolle sopivia. [CBH91]

Oppimisen oikea-aikainen tukeminen käsittää erilaisia tekniikoita ja strategioita [CBH91]. Opettajalle on tärkeää, että hän pystyy arvioimaan oppilaan taitotasoa, kykyä oppia ja opettamiseen varattua aikaa. Opettajan on tarkkailtava oppilaan työskentelyä varmistaakseen, että oppilas ei jää jumiin työssään, josta saattaa seurata oppilaan turhautumista ja motivaation menettämistä [BR10a, Sor10]. Opettaja antaa oppilaalle apua tehtävän niissä vaiheissa, joita oppilas ei vielä itsenäisesti osaa suorittaa. Opettaja voi antaa sekä suullista että kirjoitettua palautetta oppilaalle, jonka perusteella oppilas pystyy parantamaan työskentelyään [BR10a]. Lisäksi oppilaalle voidaan antaa tehtäviä, jotka sisältävät oppilaalle entuudestaan tuntemattomia tekniikoita opettajan ollessa samalla hänen tukenaan [CBH91]. Apua ja neuvoja ei saa kuitenkaan antaa liikaa, jotta tehtävän tuoma oppimismahdollisuus ei mene pilalle [VVLP13]. Opettajan täytyy suhteuttaa opetukseen käytettävissä oleva aika opetettavien asioiden laajuuteen ja sisältöön. Pieniä

yksityiskohtia ei pysty hiomaan loputtomiin, jos aikaa on rajallisesti.

Opettaja vähentää oppimisen oikea-aikaista tukemista (fading) sitä mukaa, kun oppilaalle kertyy kokemusta työstettävästä asiasta, jolloin oppilaalle jää enemmän vastuuta. Henkilökohtaista apua ei tarvitse olla paljon, kunhan oppilasta silloin tällöin ohjataan oikeaan suuntaan [VPLK11]. Nämä keinot kehittävät oppilaan itsetietoisuutta ja oppilas oppii korjaamaan virheitään itsenäisesti, minkä seurauksena hän ei tarvitse enää niin paljoa tukea [CBH91].

Artikuloinnissa oppilaan on muutettava ajatuksensa, tietonsa ja ajatusprosessinsa sanoiksi. Artikuloinnin seurauksena oppilas pystyy vertailemaan paremmin omaa ajatteluaan opettajan ajatteluun. Opettaja pääsee käsiksi oppilaan ongelmanratkaisuprosessiin, kun hän saa selville miten oppilas ajattelee erilaisissa tilanteissa. Esimerkiksi opettaja voi pyytää oppilasta selittämään ääneen ajatuksia samalla kun tämä suorittaa jonkin tehtävän eri vaiheita. [CBH91]

Peilaamisessa oppilas vertailee työskentelyään opettajaan, kokeneempaan henkilöön tai toiseen oppilaaseen. Oppilas saa tätä kautta tietoa siitä, miten hänen työskentelynsä poikkeaa toisen henkilön työskentelystä. Näin oppilas oppii huomaamaan asiat, joihin hänen on panostettava tullakseen paremmaksi. Oppilas pystyy muistelemaan aiempaa työskentelyään ja miettimään, mitä hän on oppinut. Peilaamisen seurauksena oppilaan itsetietoisuus ja itsekriittisyys kehittyvät. [CBH91]

Tutkimisessa oppilas joutuu keksimään uusia tapoja, strategioita ja keinoja lähestyä aihealuetta. Oppilas joutuu kehittämään itse kysymyksiä ja ongelmia, joita hän pyrkii sen jälkeen ratkaisemaan. Tämän seurauksena hän joutuu miettimään asioita eri näkökulmasta kuin tilanteessa, jossa hän saa tehtävän mestarilta. Oppilaan muodostaessa omia ongelmia ratkaistavakseen oppilaan taito määritellä ja mallintaa tehtävän vaatimia askeleita kehittyy. Hän joutuu käymään läpi ongelman vaatimat ominaisuudet ja sen, mitä

kyseisen ongelman ratkaisuun tulee osata. Tutkimisen seurauksena oppilaan taito itsenäiseen työskentelyyn kehittyy. [CBH91]

Mallintaminen, valmentaminen ja oppimisen oikea-aikainen tukeminen ovat kognitiivisen oppipoikamallin päätekniikat. Ne on suunniteltu niin, että opiskelija saa rakennettua käsitteellisen mallin ongelman tilasta ja pystyy kehittämään kognitiivisia taitojaan tarkkailun ja harjoittelun kautta. Artikulointi ja peilaaminen pyrkivät saamaan opiskelijan sisäistämään havainnot ja kokemukset samalla kehittäen uutta tietoa ja ongelmanratkaisutaitoja. Tutkiminen edistää opiskelijan itsenäistä ongelmien muodostamista ja ratkaisemista. [LMRC13]

Tehostettu oppipoikamalli (extreme apprenticeship) on kognitiivisen oppipoikamallin laajennus. Tehostetussa oppipoikamallissa on monia tärkeitä perusarvoja, joita painotetaan kaiken opetuksen yhteydessä. Tehostetun oppipoikamallin käyttö voi helposti kärsiä, jos perusarvoja ei noudateta [VPL11]. Siinä ovat vahvassa asemassa jatkuva palaute, tekemisen painottaminen, sekä opettajan ja oppilaan välinen yhteistyö. Lisäksi tehostetussa oppipoikamallissa kyseenalaistetaan perinteisten luentojen tärkeys. Lopullinen tavoite kaikella neuvonnalla, opetuksella ja harjoittelulla on se, että opiskelijasta tulee lopulta mestari. Sitä voidaan soveltaa opiskelijamääriltään isoillakin kursseilla [VPL11] ja online-ympäristöjen kautta esimerkiksi MOOC:ssa (Massive Open Online Course) [VLK12].

Kaksisuuntainen jatkuva palaute tekee oppimisprosessista merkityksellistä ja tehokasta [VPL11]. Opiskelijan on saatava jatkuvaa palautetta opettajalta ja palautteen on toimittava molempiin suuntiin. Opettajan antama palaute kannustaa opiskelijaa ja auttaa häntä saamaan varmistuksen siitä, että hän etenee oikeaan suuntaan työskentelyssään [VPL11]. Opiskelija puolestaan antaa palautetta opettajalle, jotta opettaja pystyy kehittymään tiedon ja taitojen eteenpäin siirtämisessä. Opettaja pystyy saamaan annetun palautteen avulla myös tiedon opiskelijan tekemisistä eli onnistumisista ja kohtaamista

haasteista, joidenka avulla hän pystyy muokkaamaan opetusta soveltumaan paremmin opiskelijan tarpeisiin. Helsingin yliopiston tietojenkäsittelytieteen laitoksella tehostettu oppipoikamalli otettiin ensimmäisen kerran käyttöön kevätlukukaudella 2010 ohjelmoinnin perus- ja jatkokursseilla [VPLK11].

Ohjelmointi on taito, joka vaatii paljon harjoittelua. Tämän takia tehtävien tekeminen kursseilla on aloitettava mahdollisimman aikaisin. Rutiinin kerryttämisen kannalta myös tehtävien määrä pitää olla suuri ja niissä on oltava jonkin verran toistoa, jotta asiat jäävät kunnolla mieleen. Niissä on oltava myös selkeät ohjeistukset tehtävän ratkaisemisen aloittamiseksi. Tehtävät on jaettu pieniin osiin, jotka asettavat opetukseen välitavoitteita. Pienet askeleet takaavat sen, että opiskelijat tuntevat oppivansa ja edistyvänsä tehtävissä. Tehtävien tekemisen on oltava välttämättömyys kurssien läpipääsylle, sillä kursseilla oppiminen pohjautuu tehtävien tekoon. Opiskelijan on harjoiteltava taitoja, kunnes haluttu taito omaksutaan. [VPLK11]

Tehtäviä voi tehdä pajaluokassa, jossa on paikalla ohjaajia. Ohjaajina toimii opiskelijoita, jotka ovat suorittaneet kurssit ja omaksuneet kurssien sisällön. Ohjaajat pystyvät tarkkailemaan opiskelijan oppimista ja samalla edistämään opiskelijan oppimisprosessia oppimisen oikea-aikaisella tukemisella. Opiskelijoita rohkaistaan etsimään tietoa itsenäisesti, koska kaikkea mahdollista ei aina käsitellä luennoilla tai materiaalissa, joten on tärkeää osata etsiä tietoa muualtakin. Opiskelijoille muodostuu harjoittelun ansiosta vahva rutiini koodin kirjoittamisesta ja he saavat motivaatiota opiskeluun onnistumisen kokemuksista.

Kognitiivisen oppipoikamallin opetusmenetelmistä mallintaminen tulee näkyviin oppimateriaalien ja luentojen kautta, jotka ovat saatavilla netistä. Opetuksessa tulee välttää liiallista yksityiskohtiin takertumista luennoilla, sillä luentojen on hyvä kattaa vain sen verran tietoa, että tehtävissä pääsee alkuun. Oppilaille annetaan hyviä käytänteitä ja keinoja, joiden avulla he pystyvät selviämään tehtävistä. Opetusmateriaali seuraa annettujen tehtä-

vien rakennetta, joten opiskelijat pystyvät seuraamaan materiaalia samalla, kun he tekevät tehtäviä. Materiaali antaa opiskelijoille oppimiseen oikea-aikaista tukea, joka toimii yhteydessä oikean tekemisen kanssa. Materiaalien ja luentojen päätarkoitus on esittää opiskelijoille työstettyjä esimerkkejä siitä, kuinka jokin ongelma ratkaistaan askel askeleelta. Opettajien on pyrittävä näyttämään esimerkkejä, jotka liittyvät suoritettaviin tehtäviin. Tällöin opiskelijat pystyvät yhdistämään luennoilla oppimiaan malleja ja käytänteitä omiin ajatusprosesseihinsa. Tällainen lähestymistapa auttaa opiskelijoita tunnistamaan hyviä tapoja ratkaista ohjelmointiin liittyviä ongelmia jo luentojen aikana. [VPLK11]

Tehostettu oppipoikamalli on tuottanut onnistuneita tuloksia Helsingin yliopistolla ohjelmoinnin perus- ja jatkokursseilla, sillä kurssien keskeyttämismäärät ovat laskeneet huomattavasti. Tehostetun oppipoikamallin uskotaan auttavan opiskelijoita, koska jatkuva palaute ja oppimisen oikea-aikainen tukeminen viedään äärimmäiselle tasolle. Menetelmän avulla täysin uudet opiskelijat, joilla työskentely ei ole muuten tehokasta ja jotka normaalisti lopettavat kurssin kesken, saavat tarpeeksi tukea ja motivaatiota suorittaakseen kurssit loppuun. Suurin osa nimettömästä opiskelijoiden antamasta palautteesta osoittaa, että oppimista tehostetun oppipoikamallin avulla pidetään motivoivana ja palkitsevana. [VPL11]

# 3 Kynnyskäsitteet ja käsitteellinen muutos

Tässä osiossa esitellään kynnyskäsite, käsitteellinen muutos ja väärinkäsitys. Kyseiset käsitteet ovat tärkeitä, koska tutkielmaa varten tehty tutkimus pohjautuu näiden ilmiöiden ympärille.

## 3.1 Kynnyskäsite

Tietämyksen esittämiseen tarvitaan käsitteitä (concepts), jotka ovat kognitiivisia merkityssisältöjä ja niitä kuvataan yhdellä sanalla kuten: objekti, eläin, elossa, lämpö, paino ja aine [Car00]. Kynnyskäsitteet (threshold concepts) voivat olla jokin opittava taito, omaksuttava tieto tai asia [Vag06]. Kynnyskäsitteet edustavat osia aihealueesta, jotka ovat erityisen haastavia omaksua ja yhdistää hallittavaksi kokonaisuudeksi pienemmistä palasista. Ne voidaan nähdä portteina uudenlaiseen ajattelutapaan jostakin aihealueesta. Niiden omaksuminen sitoo useita eri käsitteitä yhteen ja mahdollistaa uuden tavan ymmärtää asioita aiheuttaen opiskelijan ajattelussa käsitteellisen muutoksen (conceptual change). Kynnyskäsitteen omaksuminen vaikuttaa henkilön ammattimaisuuteen aihealueessa ja todennäköisesti tapaan, kuinka henkilö siitä puhuu [SM16]. Teoria kynnyskäsitteistä pohjautuu konstruktiiviseen oppimisteoriaan, jonka mukaan oppiminen tapahtuu henkilön aiempaan tietämykseen perustuen. Henkilön tieto ja ymmärrys aihealueessa rakentuvat ulkoisten ärsykkeiden kautta hänen olemassaolevien kokemustensa ja henkilökohtaisten kognitiivisten rakenteiden päälle [Vag06]. Tämän takia oppimiseen vaikuttavat tekijät, kuten käytettävät opetusmenetelmät, on syytä ottaa huomioon kynnyskäsitteitä tarkastellessa [SF14]. Kynnyskäsitteiden tunnistaminen mahdollistaa opetussuunnitelmien rakentamisen, joissa painotetaan tiettyjen käsitteiden opettamiseen.

Meyer ja Land [ML03] ovat ensimmäisenä määritelleet kynnyskäsitteiden peruspiirteet, joita on viisi kappaletta. Kynnyskäsitteet pystyvät olemaan piirteiltään muokkaavia (transformative), peruuttamattomia (irreversible),

yhdistäviä (integrative), rajaavia (bounded) ja ongelmallisia (troublesome). Kaikkien piirteiden ei tarvitse olla voimassa samanaikaisesti, jotta jokin käsite pystyttäisiin määrittelemään kynnyskäsitteeksi. Kynnyskäsitteiden piirteet näkyvät oppilaan ajattelutavan muutoksena ja itse käsitteiden luonteessa. Tärkeimmät kaksi kynnyskäsitteiden piirrettä ovat muokkaavuus ja ongelmallisuus [RR09]. Kynnyskäsitteiden piirteet ovat seuraavat:

1. Muokkaavuus: kynnyskäsitteiden oppiminen muokkaa oppilaan tapaa ajatella ja ymmärtää jokin aihealue tai sen osa. Niiden omaksuminen saattaa joissakin tapauksissa jopa muokata oppilaan kokonaisvaltaista ajattelutapaa ja maailmankuvaa.

2. Peruuttamattomuus: kynnyskäsitteiden oppiminen kasaa aihealueeseen liittyviä käsitteitä yhteen muuttaen oppilaan ajattelutapaa todennäköisesti tavalla, jota oppilaan on vaikea tai mahdoton unohtaa tai muuttaa ilman suurta vaivannäköä.

3. Yhdistävyys: kynnyskäsitteiden oppiminen luo opiskelijalle uusia yhteyksiä aihealueeseen liittyvien käsitteiden välillä, yhdistäen tietoa ja sitoen käsitteitä yhtenäisiksi kokonaisuuksiksi. Opiskelija kykenee näkemään yhteyksiä, joita hän ei ole ennen huomannut.

4. Rajaavuus: kynnyskäsitteiden oppiminen määrittelee rajat käsitteelliselle alueelle tai harjoitteelle. Tietyn kynnyskäsitteen pystyy ymmärtämään vain henkilöt, jotka ovat “sisällä” aihealueessa.

5. Ongelmallisuus: kynnyskäsitteet voivat olla todella vaikeita omaksua. Ne sisältävät piirteitä, jotka ovat monimutkaisia ja vaikeita oppia ilman syvempää ymmärrystä aihealueesta ja siihen liittyvistä ilmiöistä.

Kynnyskäsitteiden määrittely on epämääräistä, mikä jättää hieman tulkinnan varaa niiden tunnistamiseen [Sor10]. Kynnyskäsitteitä pystytään tunnistamaan niiden viidestä eri piirteestä: muokkaavuudesta, peruuttamatto-

muudesta, yhdistävyydestä, rajaavuudesta ja ongelmallisuudesta [MGH+06]. Kynnyskäsitteen löytäminen on vaikeaa, mikäli halutaan löytää käsite, joka täyttää kaikki kynnyskäsitteiden viisi eri piirrettä [MGH+06]. Käsite voi olla kynnyskäsite, vaikka se ei sisälläkään kaikkia viittä eri piirrettä [MGH+06]. Shinners-Kennedy ja kumppanit [SKF13] väittävät, että osa kynnyskäsitteiden tunnistamisen hankaluudesta liittyy niiden jokapäiväisyyteen. Niitä tarkastellaan harvoin erillään jokapäiväisistä tilanteista, koska ne sulautuvat muihin ilmiöihin.

Yhteistä kynnyskäsitteiden teorialla, skeemateorialla, kognitiivisen kuormituksen teorialla ja konstruktiivisella oppimisteorialla on teoria siitä, kuinka uusi tieto rakentuu. Kynnyskäsitteiden näkökulmasta skeemateoriassa näkyy kynnyskäsitteisiin liittyviä muokkaavia ja yhdistäviä piirteitä. Skeemat pystytään yhdistämään luonteeltaan enemmän kuitenkin muokkaaviin piirteisiin, koska uusien skeemojen luonti sisältää muutakin kuin tiedon eri osien yhdistämistä [MGH+06]. Muita kolmea kynnyskäsitteiden piirteistä ei pahemmin näy skeemateorian yhteydessä [MGH+06]. Kynnyskäsitteet eivät ole kaikille ihmisille samat, ja se käy ilmi tarkasteltaessa konstruktiivista oppimisteoriaa ja skeemateoriaa. Henkilön tietämys rakentuu hänen aiempien kokemustensa päälle, mikä myös selittää sen, miksi jotkut ihmiset kokevat tietyn käsitteen kynnyskäsitteenä ja toiset eivät.

Kynnyskäsitteet ovat merkittäviä kohtia opiskelijan oppimisprosessissa, koska ne muokkaavat opiskelijan kognitiivisia rakenteita huomattavasti. Ne myös luovat keskeisen ymmärryksen aihealueeseen liittyvästä osaamisesta. Kynnyskäsitteiden avulla voidaan selittää, miksi jotkin oppilaat eivät tunnu oppivan mitään, kun taas toiset oppivat kaiken [Sor10]. Konstruktiivisen oppimisteorian avulla pystytään perustelemaan, miksi kynnyskäsitteet eivät ole kaikilla samat. Erään henkilön omaksuma kynnyskäsite ei välttämättä ole kynnyskäsite toiselle, koska oppiminen pohjautuu henkilön aiempiin kokemuksiin [Sor10]. Kynnyskäsitteitä tutkiessa on kuitenkin huomioitava,

että on myös olemassa käsitteitä, joiden on tapana olla yleisesti opiskelijoille hankalia oppia ja joiden oppiminen aiheuttaa muokkaavia kokemuksia [DG15].

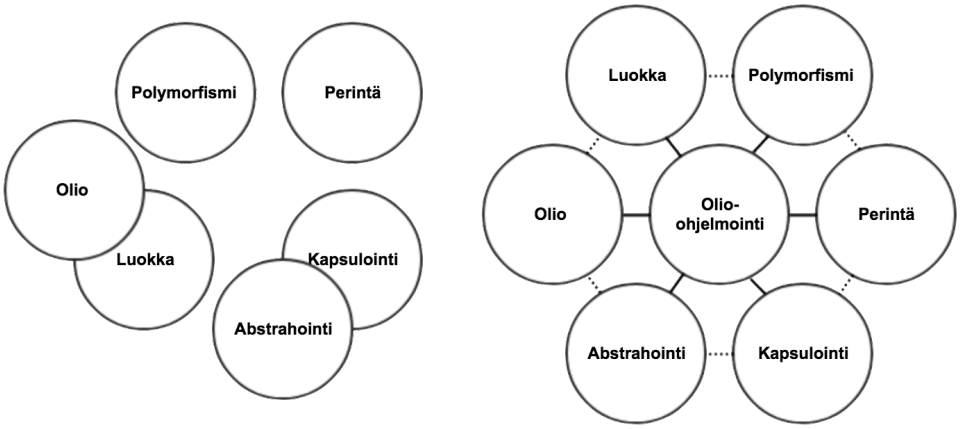
Kynnyskäsitteet jakavat samoja piirteitä kuin muutkin käsitteet, vaikka niillä onkin erillinen luokittelu [SKF13]. Yliopisto-opettajat kuvailevat kynnyskäsitteitä yleensä ydinkäsitteinä (core concept) jostakin tietystä aihealueesta [ML03]. Kynnyskäsitteet saattavat vaikuttaa samanlaiselta kuin käsiteltävän aihealueen perus- ja ydinkäsitteet, jotka toimivat rakennuspalikkoina opiskelijoiden ymmärrykseen aihealueesta, mutta ne eivät välttämättä johda käsitteelliseen muutokseen opiskelijoiden ajattelussa [ML03]. Kynnyskäsitteiden avulla pystytään selittämään, miksi opiskelijoiden ja opettajien välillä on tiedossa ja osaamisessa "muuri" [DG15]. Opettajat ovat jo omaksuneet kynnyskäsitteet ja kokeneet käsitteellisiä muutoksia, kun opiskelijoilla nämä ovat vielä edessä.

Ennen kuin tiettyyn aihealueeseen liittyvät kynnyskäsitteet on opittu, ne saattavat toimia esteenä opiskelijan oppimisprosessille ja opiskelija saattaa joutua välitilaan (liminal state) [SBE+12]. Tässä tilassa opiskelija jää jumiin kynnyskäsitteen ymmärtämisen rajamaille, jolloin hän saattaa sekoittaa uutta ja vanhaa ymmärrystä keskenään [SM16]. Jos oppilas ei onnistu omaksumaan aihealueeseen liittyviä kynnyskäsitteitä, niin opiskelija saattaa omaksua tietoa virheellisesti tai saada pelkästään pintapuolisen ymmärryksen aihealueesta [BEM+07, EMM+06]. Tällöin opiskelijan oppiminen kärsii ja hän saattaa turhautua ja menettää motivaation opiskeluun.

Tietojenkäsittelytieteen yhteydessä kynnyskäsitteen esimerkkinä voidaan käyttää olio-ohjelmointia. Olio-ohjelmointi on aihealueena laaja, sillä se sisältää paljon eri käsitteitä kuten luokka (class), olio (object), monimuotoisuus (polymorphism), kapselointi (encapsulation), perintä (inheritance) ja abstrahointi (abstraction). Jokin näistä käsitteistä saatta toimia opiskelijalle kynnyskäsitteenä. Ennen kynnyskäsitteen omaksumista opiskelijan

tieto olio-ohjelmoinnista on hajanaista, ja hän saattaa kokea siihen liittyvät käsitteet täysin erinäisinä kokonaisuuksina, joilla ei ole yhteyksiä keskenään. Lisäksi nämä yksittäiset käsitteet saattavat sisältää heikkoa ja väärää ymmärrystä, koska opiskelija ei saa yhdistettyä niitä kognitiivisiin rakenteisiinsa. Kun opiskelija oppii tarvittavan kynnyskäsitteen, muuttaa se opiskelijan ajattelutapaa olio-ohjelmoinnista yhdistäen samalla merkityksen eri käsitteiden välillä yhtenäiseksi kokonaisuudeksi luoden syvemmän ymmärryksen aihealueesta esimerkkinä (kts. Kuva 1).

**Kuva 1:** Kynnyskäsitteen oppimisen vaikutus




Idea kynnyskäsitteistä on vielä aika uusi ja se tarvitsee lisää empiiristä tukea. Kynnyskäsitteiden olemassaoloa on tutkittu tietojenkäsittelytieteen lisäksi muun muassa ainakin taloustieteissä [DM05], matematiikassa ja maantieteessä [SKC07]. Tietojenkäsittelytieteessä kynnyskäsitteiksi on ehdotettu muun muassa olio-ohjelmointia (object oriented programming), abstrahointia (abstraction), algoritmista ajattelua (algorithmic thinking), rekursiota (recursion), koodin uudelleenkäyttöä (code reuse) ja luokkaa (class) [SM16]. Kynnyskäsitteiksi on muissa aihealueissa esitetty seuraavia: matematiikassa raja-arvo (limit), kauppatieteissä vaihtoehtoiskustannus (opportunity cost), kirjallisuuskritiikissä ironia ja kirjanpidossa poisto (depreciation) [EMM+06].

## 3.2 Käsitteellinen muutos

Käsitteellinen muutos (conceptual change) on prosessi, jossa henkilön ajattelu jostakin käsitteestä tai käsitteistä muuttuu [RR09]. Se sisältää asteittaisen prosessin, jossa aihealueeseen liittyvät käsitteet ja niihin liittyvä ymmärrys rakentuu eteenpäin. Käsitteellisen muutoksen mahdollistaa jonkin tai joidenkin kynnyskäsitteiden ymmärtäminen, koska se mahdollistaa uusia kognitiivisia yhteyksiä eri käsitteiden välillä. Käsitteellisen muutosten edistämiseen pystytään hyödyntämään tarkoitusperäisiä oppimisstrategioita [CL10].

Esimerkkinä käsitteellisestä muutoksesta toimii se, että alle kouluikäiset lapset eivät aina kykene erottamaan elottomia asioita elollisista. Pienet lapset saattavat muun muassa erehtyä luulemaan, että auto on elollinen olento. Lapset perustavat ajattelunsa ja ymmärryksensä erilaisiin kognitiivisiin rakenteisin kuin aikuiset. Aikuiset ovat kokeneet käsitteellisiä muutoksia, jotka ovat auttaneet heitä ymmärtämään ja tunnistamaan eroavaisuudet elävien ja elottomien asioiden välillä. [Car00]

Oppimisen ja opetuksen näkökulmista on tarve saada parempi ymmärrys käsitteellisistä muutoksista ja luoda oppimisympäristöjä, jotka tukevat käsitteellisten muutosten kehittymistä. Chan ja muut [CL10] ovat tehneet tutkimusta käsitteellisten muutosten syntymisestä ja he havaitsivat, että käsitteellisiä muutoksia syntyy enemmän opetustilanteissa, joissa hyödynnetään oppimisen oikea-aikaista tukemista. He myös huomasivat, että oman toiminnan peilaaminen voi edistää käsitteellisten muutosten kehittymistä. Nämä havainnot antavat viitteitä siitä, että tehostettua oppipoikamallia pystytään hyödyntämään opetuksessa, jos tavoitteena on edistää käsitteellisiä muutoksia. Tehostettu oppipoikamalli pitää sisällään oppimisen oikea-aikaista tukemista ja peilaamista. Käsitteellisen muutoksen tarkastelu vanhemmilla henkilöillä on tarpeellista, koska siihen liittyvää tutkimusta on tehty lähinnä peruskouluikäisistä nuorista [CL10].

Janet ja Nathan Rountree [RR09] ovat keränneet opiskelijoiden omia

kertomuksia liittyen muuttaviin kokemuksiin ohjelmoinnissa. He keräsivät tietoa kynnyskäsitteistä, jotka olivat aiheuttaneet oppilaan ohjelmoinnin käsityksessä suuria muuttavia kokemuksia. Näitä käsitteitä olivat esimerkiksi seuraavat: modulaarisuus (modularity), tiedon abstrahointi (data abstraction), olio-ohjelmoinnin käsitteet (object-oriented concepts), koodin uudelleenkäyttö (code reuse), suunnittelumallit (design patterns) ja monimutkaisuus (complexity).

## 3.3 Väärinkäsitykset

Väärinkäsitykset (misconception) ovat virheellistä ajattelua ja tietämystä sisältäviä kognitiivisia rakenteita. Konstruktiivisen oppimisteorian mukaan väärinkäsityksiä kehittyy luonnollisesti tilanteissa, joissa oppilas yrittää laajentaa tai muokata tietämystään aihealueesta [EMM+06]. Yksilön aiempi tietämys voi johtaa väärinkäsityksiin, kun useita samankaltaisia termejä käytetään tuntemattomassa yhteydessä [EMM+06]. Mielenkiintoinen kysymys on, ilmeneekö väärinkäsityksiä enemmän kynnyskäsitteiden kuin muiden käsitteiden kohdalla. Kirjallisuutta tutkittaessa kyseistä tietoa ei löytynyt.

Kun opiskelijat tutustuvat uuteen aiheeseen, heidän osittainen tietämyksensä pyrkii ohjaamaan heitä omien sääntöjen muodostamiseen. Valitettavasti tietämyksen ollessa vajaavaista nämä itse rakentuneet säännöt saattavat johtaa väärinkäsityksiin, joita voi olla myöhemmin vaikea muuttaa oikeiksi [EMM+06]. Opiskelijoiden on nähtävä paljon vaivaa, jotta väärinkäsityksistä kehittynyt ajatusmalli voidaan muodostaa uudelleen [EMM+06]. Aloittelevilla ohjelmoijilla on havaittu kehittyvän väärinkäsityksiä joidenkin olio-ohjelmoinnin peruskäsitteiden suhteen [Dé06]. Tällaisia käsitteitä ovat varsinkin luokka ja perintä [Dé06].

Tutkimukset väärinkäsityksistä, kuten kynnyskäsitteistä, keskittyvät tutkimaan tapoja, jotka selittävät miten oppilaat mahdollisesti epäonnistuvat oppimisessa. Väärinkäsitykset ovat käsitteitä, joita oppilaiden ei haluta oppi-

van. Voidaan spekuloida, että väärinkäsitykset eivät ole työläitä omaksua toisin kuin kynnyskäsitteet [EMM+06]. Kynnyskäsitteiden ja väärinkäsitysten välillä on mahdollisesti yhteys, sillä ne jakavat samoja piirteitä keskenään. Väärinkäsitykset voivat olla tietoa yhdistäviä kuten kynnyskäsitteet, mutta väärinkäsityksen yhteydessä yhdistyminen perustuu väärään ymmärrykseen aihealueesta [EMM+06]. Väärinkäsityksiä voi olla vaikea unohtaa samoin kuin kynnyskäsitteitä, ja niiden korjaamiseksi joutuu näkemään paljon vaivaa [EMM+06].

# 4 Ohjelmoinnin opetus ja oppiminen

Tässä luvussa käydään läpi ohjelmoinnin oppimista ja opettamista yleisesti, sekä tarkastellaan näihin liittyviä haasteita. Tarkastelun kohteena on myös Helsingin yliopiston ohjelmoinnin perus- ja jatkokurssien opetussisältö. Lisäksi ohjelmoinnin oppimiseen liittyviä kynnyskäsitteitä tarkastellaan aiempien tutkimusten valossa.

## 4.1 Ohjelmoinnin oppiminen ja opettaminen

Ohjelmoinnin osaamisen merkitys kasvaa jatkuvasti, sillä siihen perustuvat ilmiöt ja palvelut lisääntyvät. Tämä tarkoittaa sitä, että yhä useammalla koulutusalalla opetussuunnitelmaan on sisällytetty ohjelmoinninopetusta. Peruskoulujen opetussuunnitelmaan ohjelmoinnin opetus otettiin käyttöön vuonna 2016. Lisääntyvän ohjelmoinnin opetuksen tarpeen takia ohjelmoinnin opetukseen on panostettava ja erilaisia opetusmenetelmiä tutkittava. Ohjelmointi on opetettavana aihealueena melko nuori, joten sen oppimiseen ja opettamiseen liittyvää empiiristä tutkimusta ei ole pitkältä aikajaksolta. Ohjelmointi on monille vaikea aihe, ja se on ymmärrettävästi avainalue ohjelmoinnin opetuksen tutkimuksessa [SSHL09]. Yleinen näkökulma tietokoneohjelmista opiskelijoilla ja loppukäyttäjillä on se, että ohjelma on

“taikaväline”, joka “tekee tarvittavan työn”. Usein kuullaan myös seuraavan tyylisiä kommentteja “se teki sen taas” tai “se ei toimi” [Vag06]. Tämä viittaa siihen, että “joku” tai “jokin” muu on vastuussa tapahtumista ohjelman suorituksen aikana kuin käyttäjä itse.

Ohjelmointi ei ole yksittäinen opeteltava taito, vaan se on monimutkaista kognitiivista toimintaa, jossa on samanaikaisesti rakennettava ja sovellettava useita korkeamman tason kognitiivisia taitoja [RR09]. Ohjelmointia ei myöskään nähdä ainoastaan yksinkertaisten syntaksirakenteiden hyödyntämisenä, vaan käytännön ongelmanratkaisutaitojen harjoittamisena tarkoituksellisessa kontekstissa [VVLP13]. Tehtyjen tutkimusten mukaan tehokkain tapa kartuttaa ohjelmointiin liittyvää tietoa ja taitoa on aktiivisen ongelmanratkaisuprosessin aikana [BR10a]. Ohjelmoinnin opettamiseen liittyy kolme peruskomponenttia: opetussuunnitelma (curriculum), opetus (pedagogy) ja tehtävät (assesment) [Car00].

Aloittelevien ohjelmoijien kohtaamista ongelmista on tehty paljon tutkimusta [Sor10, SM16]. Tutkimuksissa käy ilmi, että opiskelijat eivät kykene rakentamaan järkeviä ohjelmakokonaisuuksia opiskeltuaan pari vuotta ohjelmointia [CB07]. Black ja muut [Bla06] toteavat, että todellinen haaste ohjelmoinnin oppimisessa on opitun syntaksin hyödyntäminen ongelmien ratkaisemiseen. Sillä, kuinka monta eri käsitteen määritelmää opiskelijat pystyvät muistamaan tai kuinka montaa menetelmää he osaavat soveltaa, ei ole paljon merkitystä. Opiskelijat, jotka läpäisevät ohjelmointikurssit kehittämättä itselleen dynaamista näkökulmaa ohjelmointiin, eivät ole oppineet paljoa ohjelmoinnista [Sor10]. Tämä viittaa siihen, että syntaksin osaamisesta huolimatta ohjelmointiin liittyvän ajattelutavan skeemoja ei ole vielä kehittynyt. Opiskelijaa tulee ohjata suuntaan, jossa tällaiset skeemat pääsevät kehittymään ja yhdistymään opetettuihin käsitteisiin ja menetelmiin.

Ohjelmoinnin perus- ja jatkokursseilla (CS1) on ollut kansainvälisesti noin 33%:n keskeytysprosentti, joka on huolestuttavan suuri [BC07]. Viimeaikaiset

tutkimukset ovat osoittaneet, että pedagogiikan kehityksestä huolimatta keskeytysprosentit kursseilla eivät ole pienentyneet [WL14]. Pedagogiikan kehitys ei ole myöskään vaikuttanut ulkoisen opetuksen kontekstiin, kuten kursseilla opetettuun ohjelmointikieleen [WL14].

Korkeakouluissa ohjelmoinnin perus- ja jatkokurssit tarjoavat aloituspisteen ohjelmoinnin opinnoille. Empiiriset tutkimukset ovat osoittaneet, että opiskelijat kokevat nämä kurssit erityisen haastavina [Vag06]. Tähän vaikuttaa se, että ohjelmoinnin peruskursseilla oppilaat joutuvat kohtaamaan paljon uusia käsitteitä lyhyessä ajassa. Opeteltavia käsitteitä ovat esimerkiksi semantiikka, ohjelmointiympäristöt, looginen ajattelu, sekä ongelmanratkaisuun liittyvät strategiat. Osa näiden kurssien aikana kohdattavista käsitteistä saattaa toimia kynnyskäsitteinä opiskelijoille [SBE+12]. Kynnyskäsitteiden ja käsitteisiin liittyvän ymmärryksen kehittymistä kurssien aikana on kuitenkin haastavaa tutkia, sillä eri oppilaitoksissa ja kursseilla käytetään useita eri opetus- ja arviointimenetelmiä [SF14].

Perinteisiä opetusmenetelmiä ohjelmoinnin opetuksessa on kritisoitu paljon. Perinteisissä lähestymistavoissa ohjelmoinnin perus- ja jatkokursseilla opetuksessa ei ole tyypillisesti otettu huomioon psykologisia ja kokeellisia tutkimuksia liittyen ohjelmointitaitojen kehittymiseen [MGH+06]. Perinteiset oppimateriaalit eivät paljasta opiskelijoille prosessia siitä, kuinka erilaisissa ongelmatilanteissa pitää pyrkiä toimimaan [CB07]. Opiskelijoille on haastavaa havaita ja korjata itsenäisesti virheitään [XHB14, GM10]. Toisaalta on myös tutkimusta siitä, miten perinteisiin opetusmenetelmiin perustuvat sisäiset tekijät eivät ole pystyneet selittämään epäonnistumisia erilaisissa opetuskokonaisuuksissa [WLG14].

Ohjelmoinnin opetukseen liittyviä uusia vaihtoehtoisia opetusmenetelmiä on tutkittu paljon, mutta uudet opetusmenetelmät eivät ole vielä kunnolla löytäneet tietään opetukseen [MGH+06]. Yleisesti opetussuunnitelmien materiaalit rakentuvat tietämättömyyden ympärille ja keskittyvät siihen

mitä opiskelija ei vielä hallitse [Car00]. Tämän sijaan voitaisiin keskittyä opiskelijoiden aiempaan osaamiseen, koska tällöin pystytään panostamaan opiskelijoiden etenemisen kannalta olennaisiin kohtiin. Opiskelijat saattavat oppia huonoja ohjelmointikäytänteitä, jos he joutuvat tekemään ohjelmointitehtävät täysin itsenäisesti [VPL11]. Kehittävä palaute auttaa opiskelijoita kehittämään omia taitojaan ja pääsemään yli ongelmatilanteista.

Korkeamman tason opetuksessa ohjelmointi nähdään taitona, jossa hyödytään suorasta kontaktista ja kokeneempien henkilöiden antamasta tuesta ja palautteesta [KV11]. Opettajat saattavat helposti tehdä sen virheen, että he eivät ota huomioon opiskelijoiden ajatusprosessin olevan erilaisella tasolla kuin itsellään. Opettajat ovat omaksuneet ohjelmointiin liittyvät kynnyskäsitteet ja kokeneet useita käsitteellisiä muutoksia, minkä seurauksena heidän ajatusprosessinsa on abstrahoitunut korkeammalle tasolle. Opetuksessa on otettava ajattelumalleihin liittyvä ero huomioon ja käytettävä opetusmenetelmiä, jotka soveltuvat tuomaan opettajan ja opiskelijoiden ajattelua ja ymmärrystä lähemmäksi toisiaan. Hyvät opettajat ymmärtävät, että opettaminen on aloitettava opiskelijoiden sen hetkiseen tietämykseen perustuen. Opettajat luulevat monesti opiskelijoiden ongelmien ohjelmoinnissa johtuvan huonosta proseduraalisesta osaamisesta eli matemaattisista työkaluista [GM10]. Usein ongelma johtuu kuitenkin huonosta käsitteellisestä osaamisesta eli käsitteiden ja periaatteiden ymmärtämisestä, yhdistämisestä ja taidosta soveltaa niitä.

Monille opettajille on epäselvää, minkä tason ymmärrys oppilaiden tulee aihealueesta saavuttaa ohjelmoinnin perus- ja jatkokurssien aikana. Tähän vaikuttaa se, että tietojenkäsittelytieteen opettajilla ei ole vakituista mekanismia määritellä sitä, mille tasolle oppilaiden on päästävä ohjelmointitaidoiltaan. Ei ole olemassa yleisiä rajoja, joiden mukaan oppilaiden osaamista arvioidaan ensimmäisen tai viimeisen opiskeluvuoden jälkeen. [MGH+06]

Kynnyskäsitteitä pystytään ajattelemaan jalokivinä opetussuunnitelmas-

sa, koska ne tarjoavat painotuspisteitä opetukseen. Ne ovat kohtia, joiden opettamiseen kannattaa laittaa enemmän aikaa ja vaivannäköä. Opiskelijat, jotka omaksuvat kynnyskäsitteitä, ovat hyvässä asemassa ja pystyvät oppimaan lisää helpommin kuin henkilöt, jotka eivät ole omaksuneet niitä. Kynnyskäsitteiden avulla pystytään järjestämään aihealueen sisältämää tietämystä tehokkaammin verrattuna nykyisiin opetusmenetelmiin, ja niiden avulla pystytään mahdollisesti selittämään oppilaiden vaikeudet tietyissä vaiheissa opintosuunnitelmaa ja puuttumaan niihin ajoissa [SBE+12]. Niiden tunnistaminen mahdollistaa opetukseen tarkemmat sisällöt ja sen seurauksena opetus olisi siirrettävämpää eri koulutuslaitoksien välillä [MGH+06]. Vaikka kynnyskäsitteitä on haastavaa oppia ja opettaa, on niiden painotus opetuksessa kustannustehokasta [Sor10]. Kynnyskäsitteiden opettamiseen pitää panostaa enemmän, vaikka muiden käsitteiden oppiminen kärsisikin.

## 4.2 Ohjelmoinnin perus- ja jatkokurssien opetussisältö

Konstruktiivisen oppimisteorian mukaan oppimiseen liittyvät vahvasti käytetyt opetusmenetelmät. Olemassaolevissa tutkimuksissa ei ole mainittu kovin tarkasti tai ollenkaan ohjelmoinnin perus- ja jatkokurssien aikana käytettyjä opetusmenetelmiä tai niiden aikana opetettavia teemoja. Ohjelmoinnin kursseilla ilmenevät käsitteet vaihtuvat kursseittain ja riippuvat kurssien toteutuksesta eri yliopistoissa. Näistä syistä ei pystytä olemaan varmoja, kuinka hyvin eri tutkimuksista kerättyjä tuloksia pystytään vertailemaan keskenään. Tämän tutkielman empiirisen osion tutkimus sidotaan opiskelijoihin, jotka ovat suorittaneet ohjelmoinnin perus- ja jatkokurssit Helsingin yliopiston tietojenkäsittelytieteen laitoksella, kun tehostettu oppipoikamalli on ollut käytössä.

Helsingin yliopiston tietojenkäsittelytieteen laitoksen ohjelmoinnin perus- ja jatkokurssi sisältävät suuren määrän eri käsitteitä, ja niistä on julkaistu oppimistavoitematriisit (liite A ja B), joista saa hahmotelman kurssien kan-

nalta tärkeimmistä käsitteistä. Oppimistavoitematriisit näyttävät suuntaa opettajille ja opiskelijoille teemoista, joita kurssit pitävät sisällään. Oppimistavoitematriisit saattavat myös auttaa opiskelijoita ajankäytössä, ja ne auttavat laittamaan opiskeltavat asiat tärkeysjärjestykseen. Oppimistavoitematriisit näille kyseisille kursseille on luotu 17.8.2011. Kurssien opetuksen sisältö on muuttunut vuosien aikana, mutta oleelliset asiat ovat suunnilleen samat. Niissä ilmenee samoja painotuskohtia kuin olemassaolevasta kirjallisuudesta koskien kynnyskäsitteitä. Yhteisiä teemoja ovat muun muassa algoritminen ajattelu, olio-ohjelmointi, rajapinnat, perintä ja monimuotoisuus.

Ohjelmoinnin peruskurssin oppimistavoitematriisista löytyy neljä eri pääteemaa. Ensimmäinen pääteema on algoritmit ja ohjausrakenteet, ja sen esitietona toimii pelkkä peruskoulun matematiikka. Toinen pääteema on muuttujat ja tyypit, ja sen esitieto on algoritmin idea. Kolmas pääteema on aliohjelmat, ja sen esitietona on algoritmin idea. Neljäs pääteema on luokat, oliot ja kapselointi, ja sen esitietona ovat muuttujat, algoritmit, metodit ja parametrivälitys. Näitä pääteemoja ja esitietoja pystytään pitämään oppimistavoitematriisissa eräänlaisina kynnyksinä uusien asioiden opiskelemiselle ja niiden opettamiseen painotetaan ajallisesti. Pääteemoista luokat, oliot ja kapselointi, eli tiivistettynä olio-ohjelmointi, on kaikista yleisin mahdollisista kynnyskäsitteistä, joka tietojenkäsittelytieteeseen liittyvissä tutkimuksissa esiintyy.

Ohjelmoinnin jatkokurssin oppimistavoitematriisissa löytyy neljä eri pääteemaa. Melkein kaikki kurssin esitiedoista ovat peräisin ohjelmoinnin perusteet kurssilta. Ensimmäinen pääteema on luokkamäärittelyn tekniikat, ja sen esitietoina toimivat luokka, olio ja kapselointi. Toinen pääteema on periytyminen, jonka esitietona on luokka ja olio. Kolmas pääteema on virhetilanteiden käsittely, jonka esitietoina on taito laatia yksinkertaisia Java-ohjelmia. Neljäs pääteema on ohjelmointitekniikka, jonka esitietona on taito laatia yksinkertaisia Java-ohjelmia.

## 4.3 Mahdollisia kynnyskäsitteitä ohjelmoinnissa

Meyerin ja Landin [ML03] ensimmäisenä esittämä teoria kynnyskäsitteistä on ollut monien tutkijoiden tutkimuksen kohteena myös tietojenkäsittelytieteessä. Kynnyskäsitteiden tutkimiseen on käytetty useita erilaisia lähestymistapoja, sillä niiden tunnistaminen ja luokittelu aihealueessa on haastavaa.

Sanders ja kumppanit [SM16] kokosivat olemassaolevista tutkimuksista taulukon (kts. Taulukko 1), joka sisältää 32 tunnistettua mahdollista kynnyskäsitettä ja tiedon siitä kuinka empiiriset tulokset kerättiin. Opiskelijoihin sidottuja ja opiskelun aikana tapahtuvia tapoja ovat olleet luokkakaaviot, käsitekartat, ääneen miettiminen ja kuvakaappaukset. Kurssien yhteydessä yhtäaikaisesti ja toistuvasti tapahtuneet tavat ovat olleet käsitekartta ja oppimispäiväkirjat. Jälkikäteen tapahtuvissa tavoissa tutkimuksiin on käytetty sekä opiskelijoita ja opetushenkilökuntaa. Opiskelijoita on jälkikäteen pyydetty tekemään biografeja, peilaavia esseitä, heitä on haastateltu ja heille on pidetty puolistrukturoituja haastatteluja. Osa tutkimuksista on keskittynyt myös oppilaitoksen henkilökuntaan. Opettajilta on muun muassa kartoitettu haastatteluiden ja kyselyiden avulla opiskelijoiden kohtaamia kynnyskäsitteitä. Opettajilta on myös kerätty kynnyskäsitteitä tukevia todisteita, jotka ovat perustuneet oppilashavaintoihin, oppimateriaaleihin, opetussuunnitelmiin liittyviin dokumentteihin sekä tutkintovaatimuksiin.

**Taulukko 1:** Kirjallisuudessa ehdotetut kynnyskäsitteet ja niiden keräykseen hyödynnetty kohderyhmä, jos se oli tiedossa (O = opiskelijat, H = oppilaitoksen henkilökunta) [SM16].

| **Concept** | **Data source** |
|---|---|
| Abstraction | OH |
| Assignment statement semantics | |
| Basic programming principles (interrelationship of syntax, algorithms, and programming logic) | O |
| Class | O |
| Class declaration | O |
| Class/object/instance distinction | H |
| Code reuse | O |
| Complexity | O |
| Data abstraction | O |
| Design patterns | O |
| Dot-object notation in Python | O |
| Information hiding | |
| Inheritance | O |
| Java syntax | |
| Logic | OH |
| Modularity | O |
| Object-interaction | O |
| Notional machine | |
| NP-completeness reductions | |
| Object-oriented programming | OH |
| Parameter passing (methods in Python) | O |
| Pointers | OH |
| Polymorphism | H |
| Procedural abstraction | H |
| Program dynamics | |
| Program-memory interaction | |
| Recursion | OH |
| Relative addressing in spreadsheets | |
| Seperation of content, presentation and behavior in web pages | H |
| Software object | O |
| State | O |
| Translation between representations | H |

Kynnyskäsitteiden tunnistamisessa saattaa auttaa käsitteiden luokitteleminen erilaisiin kategorioihin kuten ydinkäsitteet (core concepts), peruskäsitteet (fundamental concepts), avainkäsitteet (key concepts) ja vastaavat kategoriat. Luokittelut pystyvät tarjoamaan erilaisia näkökulmia opiskeltavan aihealueen olemuksesta, jota ei välttämättä muuten tule ajatelleeksi [SKF13]. Hyvä keino kynnyskäsitteiden tunnistamiseen on ydinkäsitteiden listaaminen aihealueesta, sillä kynnyskäsitteitä pystytyään ajattelemaan ydinkäsitteiden alijoukkona [EMM+06].

Käsitteitä on valtava määrä ohjelmoinnin yhteydessä ja monet näistä käsitteistä ovat myös todella tärkeitä ohjelmoinnin osaamisen kannalta. Rountree ja kumppanit [RR09] ovatkin todenneet, että kaikki kynnyskäsitteet ovat myös ydinkäsitteitä (core concepts) aihealueessa, mutta kaikki ydinkäsitteet eivät ole kynnyskäsitteitä. He myös esittivät seuraavan esimerkin havainnoimaan tilannetta. Poimimalla käsitteitä mielivaltaisesti ACM:n opetussuunnitelmaohjeistuksessa esitetyistä käsitteistä pystytään luomaan seuraava esimerkki: olio-ohjelmointi (object-oriented programming, OOP) saattaa olla kynnyskäsite, mutta "tietojenkäsittelytieteen historia" (history of computing) ei välttämättä ole kynnyskäsite, vaikka se tunnistetaan ydinkäsitteeksi. Tietojenkäsittelytieteen historia saattaa sisältää opiskelijalle avartavaa, uutta, monimutkaista ja mahdollisesti vaivannäköä vaativaa tietoa, mutta se ei ole luonteeltaan ongelmallista tai muuttavaa tietoa oppilaalle.

Olio-ohjelmointi (Object-oriented programming) on tapa rakentaa ohjelmia. Sitä hyödyntävissä systeemeissä pienemmät ohjelman osat ovat nimeltään olioita (objects). Oliot kapseloivat tietoa ja toiminnallisuutta. Kapseloinnilla tarkoitetaan toteutukseen liittyvien muuttujien ja metodien piilottamista muiden luokkien näkyvistä [FB09]. Oliot jaetaan luokkiin, joissa luokan ilmentymät pitävät sisällään samat rakenteet ja toiminnallisuudet, mutta saattavat pitää sisällään eri arvoja tietorakenteissaan. Tärkeä aspekti olio-ohjelmointia hyödyntävissä kielissä on monimuotoisuus, joka mahdollis-

taa laajennettujen luokkien käytön kaikkialla, missä alkuperäistä luokkaa pystyy käyttämään [FB09]. Uusia luokkia tehdessä olemassaolevia luokkia pystytään laajentamaan eri tietorakenteilla tai toiminnallisuuksilla. Tämä ominaisuus tekee olioissa sijaitsevan koodin uudelleenkäytöstä mahdollista. Olio-ohjelmointi pitää sisällään paljon käsitteitä: abstrahointi (abstraction), luokka (class), ilmentymä (instance), luokan määrittely (class declaration), koodin uudelleenkäyttö (code reuse), tiedon piilotus (information hiding), perintä (inheritance), olio (object), monimuotoisuus (polymorphism), aliluokka (subclass) ja yliluokka (superclass). Mössenböck (1993) [Mös93] mainitsee olio-ohjelmoinnin huonona puolena sen, että ohjelmoijan on opittava hyödyntämään ja ymmärtämään luokkien eri käsitteitä kuten perintää ja monimuotoisuutta. Yleisimpiä olio-ohjelmointia tukevat ohjelmointikieliä ovat esimerkiksi Java, Scala, Ruby, Python, C++, C#, Visual Basic, PHP, Smalltalk ja Object Pascal.

Olio-ohjelmointi on alueena erittäin laaja ja pitää sisällään useita mahdollisia kynnyskäsitteitä. Osa tutkijoista ei pidä olio-ohjelmointia itsessään kynnyskäsitteenä, vaan he pitävät sitä liian laajana aihealueena [BEM+07]. Jos olio-ohjelmointia tarkastellaan yksinkertaisimmillaan sisällyttäen oliot, luokat ja kapseloinnin, mutta jättäen pois käsitteet kuten perinnän ja monimuotoisuuden, niin tälläkin tasolla olio-ohjelmointi on ydinkäsite tietojenkäsittelytieteessä ja yksi ensimmäisistä asioista, joka opiskelijoiden on opittava ja on tarpeen ymmärtää [EMM+06]. Toisaalta osa tutkijoista pitää olio-ohjelmointia kynnyskäsitteenä, koska sen oppimiseen liittyviä ongelmia tutkiessa on havaittu, että opiskelijoilla on ongelmia erottaa luokan ja olion käsitteet toisistaan. Tämän takia olio-ohjelmointi käsitteenä yhdistää luokan ja olion käsitteet ja antaa perustelua sille, miksi olio-ohjelmointia pystytään ajattelemaan yksittäisenä kynnyskäsitteenä [EMM+06].

Kirjallisuudesta löytyy paljon todisteita siitä, että opiskelijat kokevat olio-ohjelmoinnin haastavana oppia [EMM+06]. Useita eri tutkimuksia on

kohdistettu ensimmäisen vuoden yliopisto opiskelijoille. Tutkimuksissa opiskelijoille on tehty haastatteluita, joiden perusteella on tehty erinäisiä havaintoja olio-ohjelmoinnin oppimiseen liittyen. Opiskelijat kokevat olio-ohjelmointiin liittyvät käsitteet hankalina oppia, vaikka niiden opiskeluun laitettaisiin paljon vaivannäköä [EMM+06]. Kapselointi ja koodin uudelleenkäytettävyyteen suhteen opiskelijat usein luulevat, että koodirivien ja luokkien vähentäminen on tärkeämpää kuin kapselointi [EMM+06]. Opiskelijat eivät myöskään ymmärrä ohjelmakokonaisuuksia, joissa on useampia luokkia [EMM+06]. Tutkimusten mukaan oppilaat eivät aina omaksu käsitteitä, joita olio-ohjelmointi pitää sisällään, kuten monimuotoisuus ja olioiden yhteistoiminta [RR09]. Tutkimuksissa on myös havaittu olio-ohjelmoinnin olevan muokkaavaa, ja Luker [Luk94] toteaa olio-ohjelmoinnin paradigman oppimisesta: "requires nothing less than complete change of the world view". Olio-ohjelmoinnin käsitteiden oppimiseen liittyy myös väärinkäsityksiä, jotka vaikeuttavat sen oppimista [EMM+06].

Abstrahointi (abstraction) terminä juontaa juurensa latinan kielen sanasta *abstrahare*. Abstrahare sanana viittaa prosessiin, jossa jostakin asiasta poistetaan piirteitä, kunnes jäljellä ovat vain olennaiset piirteet. Ohjelmointi pitää sisällään abstrahointien määrittelemistä ja hyödyntämistä. Abstrahointi ja taito luoda joustavuutta abstrahoinnin tasolta toiselle ovat avaintaitoja ohjelmoinnissa. Abstrahointi on ydinkäsite tietojenkäsittelytieteessä ja sitä on tutkittu laajasti [EMM+06, MBE+08]. Useat tutkimukset tukevat abstrahointia kynnyskäsitteenä, sillä se pitää sisällään selkeitä kynnyskäsitteiden piirteitä. Joissakin tutkimuksissa väitetään, että abstrahointi ei vaikuta kynnyskäsitteeltä itsessään, mutta tietyt käsitteet, joissa abstrahointi on läsnä, saattavat olla kynnyskäsitteitä [MBE+08].

Empiirisen tutkimusten mukaan oppilaat usein kuvailevat muuttavien oppimiskokemusten tapahtuvan oppimistilanteissa, joissa he hyödyntävät monenlaisia abstrahointeja, kuten modulaarisuutta (modularity), tiedon

abstrahointia (data abstraction), perintää (inheritance), monimuotoisuus (polymorfism), koodin uudelleenkäyttöä (code reuse), suunnittelumalleja (design patterns) ja monimutkaisuutta (complexity). Jotkut oppilaat kuvailevat abstrahoitujen käsitteiden tulevan vastaan ensin käsitetasolla ja vasta myöhemmin ohjelmissa. Toisaalta jotkut oppilaista ovat oppineet hyödyntämään näitä abstrahoituja käsitteitä, vaikka eivät olekaan joutuneet käyttämään niitä. [MBE+08]

Olio-ohjelmoinnissa abstrahointi on yksi kolmesta keskeisestä piirteestä kapseloinnin ja perinnän lisäksi. Abstrahoinnin avulla ohjelmoija pystyy piilottamaan oliossa kaiken paitsi olennaisen tiedon kompleksisuuden vähentämiseksi. Luokan määritteleminen siihen liittyvien muuttujien ja metodien kanssa on perinteisin vaadittu abstrahointi [OBL04].

Or-Bach ja Lavy [OBL04] ovat tehneet empiirisen tutkimuksen koskien abstrahointia olio-ohjelmoinnissa. Tutkimuksessa he käyttivät kognitiivista tehtäväanalyysiluokittelua (cognitive task analysis taxonomy) ja tulivat siihen johtopäätökseen, että abstrahointi on avainasemassa olio-ohjelmoinnissa. Se on taito, jonka oppimiseen vaaditaan paljon kognitiivista vaivannäköä. He olivat myös samaa mieltä muiden tutkijoiden kanssa siitä, että olio-ohjelmointiin liittyvää ohjelmien analysointia ja suunnittelua pitää opettaa jo ensimmäisen suoritettavan ohjelmointikurssin aikana.

Detienne [Dé06] on tehnyt samankaltaisia havaintoja omassa tutkimuksessaan, joka koski oppilaiden käyttämiä metodeja ja attribuutteja olio-ohjelmoinnissa. Hänen mukaansa aloittelevien ohjelmoijien yksi suurimmista vaikeuksista liittyy ratkaisun muodostamiseen deklaratiivisten ja proseduraalisten tapojen välillä varsinkin tilanteissa, joissa olio voidaan ajatella abstrahoituna tietotyyppinä. Hän havaitsi tuolloin ammattilaisen ja aloittelijan ajattelutavoissa vallitsevan eroavaisuuden.

Box ja Whitelaw [BW00] esittävät tutkimuksessaan, että abstrahointi auttaa selittämään vaikeutta oppia olio-ohjelmointia. Olio-ohjelmoinnissa

uusien käsitteiden oppiminen sisältää useita askeleita. Näistä askeleista abstrahointi on kaikista haastavin ja viimeinen. Oppilaille tuottaa vaikeuksia havaita asiat, jotka olioissa ryhmittyvät yhteen ja mitä muuttujia milloinkin pitää käyttää parametreina. He myös toteavat, että abstrahoinnin vaikeuden ymmärtäminen on ollut tärkeä osa olio-ohjelmoinnin opetuksen kehitystä.

# 5 Empiirinen tutkimus

Tässä luvussa käsitellään tutkielman empriisisen osion sisältö ja tutkimuksessa käytetty laadullinen tutkimusmenetelmä. Tutkimuksen kohdehenkilöihin liittyvät kriteerit käydään läpi ja heidän taustojaan aukaistaan hieman. Lopuksi käsitellään tarkemmin laadullisen tutkimuksen rakennetta.

## 5.1 Tutkimusmenetelmä

Tutkielman tapaustutkimuksessa käytetään puolistrukturoitua haastattelua. Puolistrukturoituja haastatteluita käytetään paljon laadullisessa tutkimuksessa. Se on myös yleisin tutkimusmenetelmä yhteiskuntatieteissä [Sal14]. Puolistrukturoidussa haastattelussa haastattelijalla on haastattelutilanteessa käytössään tyypillisesti valmiiksi suunniteltu haastatteluohje, jota hän seuraa [BR10b]. Haastatteluohje pitää usein sisällään joukon valmiita avoimia kysymyksiä, jotka pitävät haastattelun tietyn aihealueen sisällä, vaikka haastattelu saattaa poiketa haastattelijan ohjaamana. Haastattelu perinteisesti nauhoitetaan ja myöhemmin litteroidaan analyysia varten, sillä tulosten kirjoittaminen haastattelun aikana voi häiritä haastattelua [BR10b]. Puolistrukturoidut haastattelut pystyvät tarjoamaan luotettavaa ja vertailtavissa olevaa laadullista dataa [BR10b].

Haastattelua rakennettaessa on tärkeää miettiä, kuinka kysymykset muotoillaan kohteelle, jotta kysyttävä kysymys ei johdattele vastaamaan tietyllä tavalla. Esimerkiksi tilanteessa, jossa opiskelija kertoo jonkin kurssilla

esiintyneen aihealueen olleen vaikea, ei siihen liittyvistä väärinkäsityksistä haluta kysyä suoraan. Tämä saattaa johtaa opiskelijalla olettamukseen, että kyseiseen aihealueeseen täytyy kuulua väärinkäsityksiä.

Puolistrukturoitu haastattelu sopii tutkittavaan aihealueeseen, koska se antaa mahdollista liikkumavaraa haastattelussa. Avoimien kysymysten avulla pystytään perehtymään syvemmin opiskelijoiden henkilökohtaisiin ongelma-alueisiin oppimisessa ja tätä kautta mahdollisiin kynnyskäsitteisiin. Haastateltava henkilö pystyy ilmaisemaan omia näkökulmiaan aihealueesta, ja haastattelija pystyy perehtymään yksittäisiin kysymyksiin syvemmin, kun haastattelun kulku ei ole ennalta määritelty [BR10b]. Kyseinen menetelmä antaa mahdollisuuden muokata haastatteluun liittyviä kysymyksiä tarpeen mukaan. Menetelmä myös huomioi sen, että ihmiset ovat luonteeltaan erilaisia, ja että kaikki eivät ole yhtä puheliaita oma-aloitteisesti.

## 5.2 Tutkimustieto

Tutkimuksen lähtökohtana oli haastatella ohjelmoinnin perus- ja jatkokurssit suorittaneita opiskelijoita. Kurssit suorittaneet opiskelijat muistivat tarpeeksi hyvin haastattelun kannalta syyn siihen, miksi he eivät edenneet ohjelmointitehtävissä ja mitkä olivat heille haastavia asioita. Kaikki haastattelut pidettiin kevään 2016 aikana. Haastattelussa tiloina toimivat yliopiston omat tilat tai jokin muu ennalta sovittu paikka. Haastateltuja opiskelijoita oli yhteensä yhdeksän. Suuremman joukon haastattelu ei ollut tarpeellista, sillä kyseisellä määrällä haastatteluita vastaukset alkoivat toistamaan itseään ja haastattelut eivät antaneet enää uutta informaatiota.

Tutkimuksen tutkimushenkilöt (N=9) olivat Helsingin yliopiston opiskelijoita. Kaikki haastatelluista henkilöistä olivat käyneet ohjelmoinnin perus- ja jatkokurssin Helsingin yliopiston tietojenkäsittelytieteen laitoksella vuonna 2010 tai myöhemmin tehostetun oppipoikamallin ollessa käytössä. Tutkimushenkilöiksi valikoituivat satunnaiset, yllä mainitut kriteerit täyttävät

opiskelijat vapaaehtoisuuteen perustuen. Haastatelluista henkilöistä miehiä oli kahdeksan (88,9%) ja naisia yksi (11,1%). Tietojenkäsittelytieteen laitoksella suurin osa opiskelijoista on miehiä, mikä saattoi näkyä tutkimukseen osallistuvien henkilöiden sukupuolijakaumassa. Haastatellut valittiin satunnaisista henkilöistä yliopiston tiloissa, joten jakauma on myös tältä osin satunnainen. Haastateltaville henkilöille kerrottiin haastattelusta etukäteen vain haastattelun aihealue ja kesto.

Haastateltujen henkilöiden pääaine jakautui seuraavasti: tietojenkäsittelytiede (N=6), matematiikka (N=2) ja tilastotiede (N=1). Tietojenkäsittelytiedettä ei tarvitse opiskella Helsingin yliopistolla pääaineena, vaan sitä voi opiskella myös sivuaineena. Tietojenkäsittelytieteen opiskelijoille ohjelmoinnin perus- ja jatkokurssi ovat pakollinen osa opintoja. Muille tutkimukseen osallistuneille henkilöille kurssit ovat osa tietojenkäsittelytieteen sivuainekokonaisuutta tai vapaavalinnaisia opintoja. Tutkimushenkilöiden kurssien suoritusajankohdat jakautuivat vuosittain seuraavasti: 2010 (N=1), 2011 (N=3), 2012 (N=1), 2013 (N=1), 2014 (N=2), 2015 (N=1).

Haastateltujen opiskelijoiden ohjelmointikokemus ennen ohjelmoinnin perus- ja jatkokurssien suorittamista oli pääosin kovin vähäistä. Kaikilla haastatelluista henkilöistä (N=9) oli ohjelmointikokemusta ala- tai yläasteelta, jolloin oltiin tehty yksinkertaisia HTML-kotisivuja. Ainoastaan kolmella henkilöllä oli laajempaa ohjelmointikokemusta, joka sisälsi ohjelmointikurssien suorittamista lukiossa tai korkeammalla koulutusasteella. Nämä henkilöt olivat aiemmin perehtyneet C#-, C++-, PHP- tai Java-ohjelmointikieliin. Yksikään haastatelluista ei ollut ennen opintojensa aloittamista työskennellyt tehtävissä, joissa olisi vaadittu ohjelmoinnin osaamista.

## 5.3 Haastattelun rakenne

Puolistrukturoitu haastattelu rakennettiin sisältämään kolme selkeää osa-aluetta ja sitä varten rakennettiin haastattelupohja (kts. Taulukko 2).

**Taulukko 2:** Haastattelupohja

| **1. Johdanto** |
|---|
| Oletko suorittanut ohjelmoinnin perus- ja jatkokurssit Helsingin yliopistolla, kun tehostettu oppipoikamalli on ollut käytössä? |
| Minä vuonna? |
| Mikä on pääaineesi? |
| Oliko sinulla ennen opiskelujen aloittamista aiempaa ohjelmointikokemusta? Mitä, missä ja milloin? |
| Millainen mielikuva sinulla oli ohjelmoinnista ennen opiskelujen aloitusta? |
| **2. Kurssien aikana tapahtuneet asiat** |
| Millaiset tuntemukset sinulla oli tietojenkäsittelytieteen opiskeluista ensimmäisten viikkojen aikana? Oliko opiskeluilmapiiri mukava? |
| Millaiselta ohjelmointi tuntui ensimmäisten viikkojen aikana ohjelmoinnin perus- ja jatkokursseilla? |
| Muistatko opetuksesta sen, missä järjestyksessä käsitteiden suhteen edettiin? |
| Mitkä olivat kurssien aikana haastavia käsitteitä? |
| Liittyikö sinulla joihinkin näistä käsitteistä mahdollisia väärinkäsityksiä? |
| Vaikuttivatko haastavat aihealueet tai väärinkäsitykset muiden asioiden oppimista? |
| Tuliko sinulle ohjelmointiin liittyviä tajuntaa ja ymmärrystä laajentavia kokemuksia kurssien aikana? |
| **3. Suhtautuminen ohjelmointiin nykyään** |
| Mitä mieltä olet nykyään ohjelmoinnista? |
| Onko käsityksesi aihealueesta muuttunut, kun vertaat sitä aikaan ennen ohjelmoinnin opiskelujen aloittamista? |
| Oletko ohjelmoinut paljon ohjelmoinnin perus- ja jatkokurssien jälkeen? |
| Onko ohjelmoinnin oppiminen muuttanut arkipäiväistä tapaasi ajatella? |
| Hyödynnätkö nykyään ohjelmointiin tarvittavaa ajattelua myös muissa yhteyksissä? |

Jokaisella osiolla oli selkeä merkitys haastattelun kannalta. Osiot sisälsivät tietyn määrän ennalta määriteltyjä avoimia kysymyksiä, joiden lisäksi haastattelun edetessä esitettiin lisää kysymyksiä. Lisäkysymysten avulla sy-

vennyttiin opiskelijoiden ajatusmaailmaan ja mahdollisiin kynnyskäsitteisiin liittyviin piirteisiin, jotka joissakin tapauksissa antoivat lisää tietoa opiskelijan oppimisprosessista. Haastattelusta haluttiin samalla luoda mahdollisimman luonnollinen tilanne opiskelijoille, joka muistutti enemmän vuorovaikutteista keskustelua kuin tarkasti etenevää kyselyä.

Ensimmäinen osio pohjautui opiskelijoiden tietoon ohjelmoinnista ennen opiskelujen aloittamista, ja se sisälsi neljä ennalta suunniteltua kysymystä. Kyselyn alkuosalla varmistettiin, että opiskelija oli käynyt ohjelmoinnin perus- ja jatkokurssit Helsingin yliopistolla haluttuna ajankohtana, tehostetun oppipoikamallin ollessa käytössä opetuksessa. Alkuosan kysymyksillä oli pyrkimyksenä saada tietoa opiskelijoiden kokemuksista opiskelujen alkuvaiheilta, jolloin he olivat suorittaneet kyseiset kurssit. Kysymyksillä aktivoitiin opiskelijoita muistelemaan opintojen alkuvaiheeseen liittyviä tapahtumia, koska osalla opiskelijoista kurssien suorittamisesta oli kulunut useampi vuosi. Samalla yritettiin kartoittaa opiskelijoiden ennakko-osaamista ohjelmoinnista, ajalta ennen kuin he alkoivat suorittaa tietojenkäsittelytieteen kursseja. Myös heidän silloisia mielikuviaan ohjelmoinnista ja motivaatiotaan ohjelmointia kohtaan haluttiin selvittää.

Toisessa osiossa olivat tutkimuksen kannalta tärkeimmät kysymykset. Se piti sisällään seitsemän ennalta suunniteltua kysymystä, jotka pyrkivät selvittämään opiskelijoiden kohtaamia mahdollisia kynnyskäsitteitä, väärinkäsityksiä ja asennetta ohjelmoinnin oppimiseen. Opiskelijoita pyydettiin muistelemaan parhaansa mukaan järjestystä, jossa käsitteiden suhteen opetuksessa edettiin ohjelmoinnin perus- ja jatkokursseilla. Tämän kysymyksen tarkoituksena oli, että opiskelijat palauttavat mieleensä kursseihin liittyviä asioita. Haastattelun avulla haluttiin tietää, kokivatko opiskelijat ohjelmoinnin oppimisen haasteellisena ja mihin aihealueisiin nämä mahdolliset haasteet liittyivät. Opiskelijoilta kysyttiin myös, liittyikö joihinkin käsitteistä väärinkäsityksiä tai haasteita, ja vaikuttivatko ne muiden aihealueiden ymmär-

tämiseen. Opiskelijoita pyydettiin kertomaan jos heillä liittyi ohjelmoinnin oppimiseen tajuntaa ja ymmärrystä laajentavia kokemuksia. Osion lopuksi haastateltaville henkilöille selitettiin, mitä sana kynnyskäsite tarkoittaa. Lisäksi kysyttiin, pystyvätkö he luokittelemaan joitakin opiskelujensa aikana kohtaamistaan käsitteistä kynnyskäsitteiksi.

Kolmannessa osiossa oli viisi ennalta suunniteltua kysymystä, joiden avulla selvitettiin opiskelijoiden nykyistä suhtautumista ohjelmointiin. Opiskelijoilta haluttiin selvittää, ovatko he jatkaneet ohjelmointia kyseisten kurssien jälkeen. Haastattelun avulla kerättiin myös tietoa siitä, miten ohjelmointiin liittyvien käsitteiden omaksuminen on muuttanut opiskelijoiden tapaa nähdä ja ajatella erilaisia arkipäiväisiä tilanteita. Tämä osio toimi loppuyhteenvetona, jonka perusteella haastatellut opiskelijat pystyivät tiivistämään haastattelun tärkeimmät asiat.

# 6 Tulokset

Tässä luvussa käydään läpi puolistrukturoiduista haastatteluista (N=9) kootut tulokset. Tuloksia analysoidaan syvemmin ja samalla pohditaan niiden yhteyttä mahdollisiin kynnyskäsitteisiin. Analysoituja tuloksia verrataan olemassaolevaan tutkimukseen kynnyskäsitteistä. Lopuksi pohditaan tutkimukseen liittyviä rajoitteita ja mahdollisia lähestymistapoja tehdä jatkotutkimusta ohjelmoinnin oppimiseen liittyvistä kynnyskäsitteistä.

## 6.1 Tulosten analyysi

Saaduista tuloksista luotiin kaksi erillistä taulukkoa, joista nähdään haastatteluissa ilmi tulleet haastavat käsitteet, väärinkäsitykset ja muuttavat kokemukset. Ohjelmoinnin oppimiseen liittyviä mahdollisia kynnyskäsitteitä pyritään tunnistamaan ja analysoimaan näiden kasattujen tulosten valossa.

Ongelmallisuus on yksi kynnyskäsitteiden piirteistä. Tunnistamalla on-

gelmallisia käsitteitä on mahdollista tunnistaa mahdollisia kynnyskäsitteitä. Haastavuus ja väärinkäsitykset voidaan olettaa sisältyvän tähän piirteeseen. Taulukko 3 pitää sisällään tiedon siitä, kuinka monta kertaa erilliset haastatellut (N = 9 opiskelijaa) mainitsivat jonkin käsitteen haastavaksi ja kuinka moneen käsitteeseen liittyi väärinkäsityksiä.

**Taulukko 3:** Haastatteluista kerätyt haastavat käsitteet ja väärinkäsitykset opiskelijamäärittäin (N=9).

| **Käsite** | **Haastavat käsitteet** | **Väärinkäsitykset** |
|---|---|---|
| Algoritminen ajattelu | 4 | - |
| Debuggaus | 1 | 1 |
| Ehtolausekkeet | 1 | 1 |
| Isot ohjelmakokonaisuudet | 2 | - |
| Isot kirjastot | 1 | - |
| Olio-ohjelmointi | 4 | 1 |
| Perintä ja rajapinnat | 4 | 2 |
| Rekursio | 7 | 1 |
| Näkyvyys | 3 | 3 |
| Swing | 3 | - |
| Syntaksi | 1 | - |
| Taulukoiden läpikäynti | 2 | - |
| Testien ymmärtäminen | 1 | - |
| Toistolausekkeet | 4 | 1 |

Opiskelijat kykenivät muistamaan sujuvasti kurssien aikana kohtaamiaan haastavia käsitteitä, ja haastattelut antoivat hyvin tietoa opiskelijoiden kohtaamista haastavista käsitteistä. Taulukkoa 3 tarkastelemalla nähdään, että haastatellut opiskelijat kokivat haastavimpina käsitteinä rekursion, algoritmisen ajattelun, olio-ohjelmoinnin, perinnän ja rajapinnat, sekä toistolausekkeet.

Saaduista haastavista käsitteistä pystytään tekemään yleistyksiä laajempiin kategorioihin.

Algoritmisen ajattelun mainitsi haastavaksi neljä opiskelijaa, ja se voidaan ajatella pitämään sisällään kasatuista käsitteistä rekursion, taulukoiden läpikäynnin, toistolausekkeet ja ehtolausekkeet. Vaikka toistolausekkeet ja ehtolausekkeet voidaankin ajatella osaksi algoritmista ajattelua, ovat ne kuitenkin osa ohjelmointiin liittyviä peruskäsitteitä. Nämä ovat asioita, jotka jokaisen ohjelmoijan on omaksuttava, jotta he pystyvät selviytymään ohjelmoinnin perus- ja jatkokursseista. Rekursio oli selkeästi haastavin käsite tutkimuksen opiskelijoille ja sen mainitsi haastavaksi seitsemän henkilöä. Rekursio käsitteenä sisältää algoritmista ajattelua; nämä molemmat ovat tunnistettu mahdollisiksi kynnyskäsitteiksi olemassaolevissa tutkimuksissa [SM16]. Rekursio on käsitteenä hankala oppia, mutta se ei välttämättä toimi kynnyskäsitteenä ohjelmoinnin perus- ja jatkokurssien aikana. Monimutkaisemmat algoritmiset rakenteet vaativat monesti rekursiota toimiakseen, ja sen takia rekursiota pystytään ajattelemaan kynnyskäsitteenä, joka on pakko oppia esimerkiksi puiden läpikäyntiä varten.

Olio-ohjelmointi puolestaan pitää sisällään havaituista käsitteistä perinnän ja rajapinnat. Olio-ohjelmoinnin mainitsi haastavaksi käsitteeksi neljä opiskelijaa. Perintä ja rajapinnat mainittiin haastavaksi käsitteeksi neljästi. Mielenkiintoinen huomio oli se, että perinnän ja rajapinnat haastavaksi käsitteeksi tunnistaneet opiskelijat eivät välttämättä kokeneet olio-ohjelmointia haastavaksi.

Swing aihealueena oli haastava kolmelle opiskelijalle. Swing liittyy ohjelmoinnin jatkokurssin loppupuolen tehtäviin, joissa opiskelijat hyödyntävät Swing -kirjastoa yksinkertaisten graafisten elementtien luomiseen. Tämä on usealle opiskelijalle ensimmäinen kerta, kun he joutuvat perehtymään laajaan ulkoiseen kirjastoon ja soveltamaan laajemmin kursseilla aiemmin opittuja asioita. Swingiä itsessään ei voida pitää kynnyskäsitteenä, vaan on-

gelmat tähän liittyen pystytään selittämään isojen ohjelmakokonaisuuksien ja kirjastojen ymmärtämisellä.

Haastattelut eivät antaneet paljoa tietoa opiskelijoiden omaksumista väärinkäsityksistä kurssien aikana. Tähän ilmiöön saattaa olla syynä se, että opiskelijat eivät kykene kunnolla muistamaan kursseilla kokemiaan väärinkäsityksiä tai nämä väärinkäsitykset ovat muokkautuneet pois hiljalleen ilman erityistä mieleen jäävää muuttavaa kokemusta aihealueesta. Saattaa olla, että opiskelijat eivät myöskään tiedosta kurssien aikana tai jälkeen asioita, joita he eivät ole ymmärtäneet kunnolla.

Vaikuttaa siltä, että opiskelijoiden väärinkäsitykset eivät ole yhteydessä siihen, miten usein jokin käsite koetaan haastavaksi. On mahdollista, että jotkin ohjelmointiin liittyvistä käsitteistä saattavat aiheuttaa väärinkäsityksiä helpommin kuin toiset. Kaikki haastatellut henkilöt, jotka kokivat näkyvyyden haastavana käsitteenä, kokivat myös väärinkäsityksiä kyseiseen käsitteeseen liittyen.

Muokkaavuus on yksi kynnyskäsitteiden pääpiirteistä ja sitä on mahdollista tutkia muuttavien kokemusten perusteella. Kynnyskäsitteiden oppiminen muokkaa oppilaan tapaa ajatella ja ymmärtää jokin aihealue tai sen osa. Tunnistamalla muuttavia kokemuksia on mahdollista tunnistaa mahdollisia kynnyskäsitteitä, ja miten niiden omaksuminen vaikutti oppilaan ajatteluun aihealueesta. Taulukko 4 pitää sisällään tiedon siitä, kuinka moneen käsitteeseen opiskelijat (N=9) mainitsivat liittyneen muuttavia kokemuksia.

**Taulukko 4:** Haastatteluista kerätyt muuttavat kokemukset opiskelijamäärittäin (N=9).

| **Käsite** | **Muuttavat kokemukset** |
|---|---|
| Ehtolausekkeet | 1 |
| Hashmapit | 1 |
| Isot kirjastot | 1 |
| Jakojäännös | 1 |
| Ohjelmoinnin vaikutus | 2 |
| Olio-ohjelmointi | 3 |
| Perintä ja rajapinnat | 2 |
| Rekursio | 1 |
| Swing | 1 |
| Toistolausekkeet | 2 |

Opiskelijat kykenivät muistamaan ja kertomaan kokemistaan muuttavista kokemuksista. Tähän saattaa olla syynä se, että kyseiset kokemukset ovat voineet laajentaa opiskelijoiden ymmärrystä tavalla, joka on jättänyt heille kokonaan uudenlaisen näkemyksen aihealueesta ja ilmiöistä sen ympärillä [ZBM+09]. Muuttavia kokemuksia liittyi useisiin eri käsitteisiin, mutta niitä liittyi määrällisesti vähän yksittäisiin käsitteisiin. Tästä syystä muuttavista kokemuksista ei saada tehtyä tarkkoja johtopäätöksiä suhteessa kynnyskäsitteisiin. Tulokset antavat kuitenkin viitteitä kynnyskäsitteiden olemuksesta, sillä useat mainituista aihealueista joihin muuttavat kokemukset sisältyivät, kuuluivat myös listaan haastavista käsitteistä tai väärinkäsityksistä.

Käsitteellä ohjelmoinnin vaikutus tarkoitetaan sitä, että opiskelijat ymmärsivät kuinka voimakas työkalu ohjelmointi on. He eivät olleet aiemmin huomanneet, mitä kaikkea ohjelmoinnin avulla pystyy tekemään, millainen työkalu se on, ja millaisia rajoitteita se pitää sisällään.

Haastatteluista kerätyistä tuloksista pystytään tunnistamaan useampi kä-

site, jotka omaavat kynnyskäsitteiden piirteistä ongelmallisuutta ja muokkaavuutta. Näitä haasteellisuutta, väärinkäsityksiä ja muokkaavia kokemuksia sisältäneitä käsitteitä olivat olio-ohjelmointi, perintä ja rajapinnat, rekursio, sekä toisto- ja ehtolausekkeet. Jos tarkastelemme kynnyskäsitteiden piirteistä pelkästään ongelmallisuutta ja muokkaavuutta, on mahdollista pitää edellä mainittuja käsitteitä mahdollisina kynnyskäsitteinä.

## 6.2 Tulokset muiden tutkimusten valossa

Tämän tutkielman tutkimustuloksissa ilmaantuneet haastavat käsitteet ovat kaikki esiintyneet myös Sandersin ja kumppanien [SM16] tutkimuksessa, joka pitää sisällään listan (kts. Taulukko 1) eri käsitteistä, joita tutkimuksissa ollaan tunnistettu mahdollisiksi kynnyskäsitteiksi. Selkeimmät mahdolliset kynnyskäsitteet tutkimustulosten perusteella olivat rekursio, algoritminen ajattelu, olio-ohjelmointi, toistolausekkeet, perintä ja rajapinnat. Vaikka tämän tutkielman empiirisessä osiossa ilmi tulleet kynnyskäsitteet eivät ole nimellisesti samoja listassa olevien kanssa, niin ne voidaan kuitenkin yhdistää johonkin toiseen käsitteeseen. Ohjelmointiin liittyvät kynnyskäsitteet ja käsitteet ylipäätänsä ovat sidoksissa toisiinsa, minkä seurauksena opiskelijoilla saattaa olla vaikeaa käyttää oikeita termejä puhuessaan asioista. Opiskelijat saattavat myös yksinkertaistaa käsitteitä ilman, että jaottelevat käsitteitä pienempiin palasiin [MBE+08]. Tämä hankaloittaa kynnyskäsitteiden tunnistamista ja niiden erittelyä.

Useissa tehdyissä tutkimuksissa olio-ohjelmointi on tunnistettu mahdolliseksi kynnyskäsitteeksi [Sor10, Mös93, RR09]. Olio-ohjelmointi on aihealueena haastava oppia [EMM+06], lisäksi siihen liittyy myös muuttavia kokemuksia ja väärinkäsityksiä [Luk94, EMM+06]. Tässä tutkielmassa tehty tutkimus antaa myös tuloksia, jotka kertovat olio-ohjelmoinnin haastavuudesta ja siihen liittyvistä muuttavista kokemuksista. Useat tutkijat pitävät sitä kuitenkin liian suurena yksittäisenä käsitteenä [BEM+07], koska se pi-

tää sisällään useita erillisiä käsitteitä. Tutkimustuloksissa huomattiin, että opiskelijoilla oli vaikeuksia näkyvyyden, perinnän ja rajapintojen kanssa, jotka voidaan laskea osaksi olio-ohjelmointia. On melko varmaa, että olio-ohjelmointi aihealueena pitää sisällään useampia kynnyskäsitteitä, mutta niiden yksittäinen tunnistaminen on hankalaa.

Tutkimustuloksista käy ilmi, että opiskelijoilla oli hankaluuksia isojen ohjelmakokonaisuuksien kanssa. Isoissa ohjelmakokonaisuuksissa on yhdisteltävä useita eri olioita luokkineen ja määriteltävä niihin liittyvää näkyvyyttä. Nämä elementit liittyvät kapselointiin ja olio-ohjelmointiin, joita ollaan aikaisemmissa tutkimuksissa epäilty mahdollisiksi kynnyskäsitteiksi [SM16]. Tutkimuksissa on myös huomattu, että opiskelijoille on hankalaa rakentaa ohjelmakokonaisuuksia, joissa on useampia luokkia [EMM+06].

Moström ja kumppanit [MBE+08] ovat keränneet kasaan opiskelijoiden kertomuksia liittyen muuttaviin kokemuksiin ohjelmoinnissa. Näitä käsitteitä olivat esimerkiksi seuraavat: modulaarisuus (modularity), tiedon abstrahointi (data abstraction), olio-ohjelmoinnin käsitteet (object-oriented concepts), koodin uudelleenkäyttö (code reuse), suunnittelumallit (design patterns) ja monimutkaisuus (complexity). Saaduista tutkimustuloksista löytyy samoja tuloksia kuin olemassaolevista tutkimuksista, joissa opiskelijat olivat kokeneet muuttavia kokemuksia liittyen olio-ohjelmointiin, perintään ja rajapintoihin, algoritmisiin rakenteisiin sekä isoihin ohjelmakokonaisuuksiin.

Mielenkiintoinen havainto muuttaviin kokemuksiin liittyy jakojäännöksen käsitteeseen. Eräs opiskelijoista mainitsi kokeneensa muuttavan kokemuksen tehdessään jakojäännöksiin liittyviä ohjelmointitehtäviä. Hän kertoi, ettei ollut ymmärtänyt jakojäännöksen käsitettä matematiikassa, mutta tehdessään ohjelmointitehtäviä hän sai laajemman ymmärryksen kyseisestä käsitteestä ja palaset loksahtivat paikoilleen. Tämä havainto kertoo siitä, että kynnyskäsitteet ja muuttavat kokemukset ovat useita eri aihealueita yhdistäviä.

## 6.3 Tutkimuksen rajoitteet

Haastatteluissa keskityttiin haasteellisiin käsitteisiin ja muuttaviin kokemuksiin ohjelmoinnin perus- ja jatkokurssien aikana. Olemassaolevissa tutkimuksissa on yleensä keskitytty kynnyskäsitteiden piirteistä ongelmallisuuteen tai käsitteelliseen vaikeuteen, ja muut kynnyskäsitteiden ominaisuuksista jäävät vähemmälle huomiolle [SM16]. Tämän tutkimuksen empiirinen osio keskittyi ongelmallisuuteen, käsitteelliseen vaikeuteen ja käsitteelliseen muutokseen, joten tutkimus antaa hieman uutta näkökulmaa kynnyskäsitteiden ominaisuuksista.

Usealla haastatellulla opiskelijalla oli kulunut useampi vuosi ohjelmoinnin perus- ja jatkokurssien suorittamisesta, mikä saattoi vaikuttaa heidän kykyynsä muistaa asioita kursseilta. Kurssit sisältävät monia eri käsitteitä, jotka käsitellään kurssien aikana nopeaan tahtiin. Ohjelmointiin liittyvistä käsitteistä osa on haastavia, ja ne vaativat täysin uudenlaista tapaa ajatella verrattuna muihin perinteisiin oppiaineisiin. Oppiminen tapahtuu usein asteittain ja pitkän ajan kuluessa. Opiskelijat saattavat olla pitkän aikaa kynnyskäsitteen ymmärtämisen välitilassa (liminal space), joka ei välttämättä jätä opiskelijoille erillistä muistikuvaa oppimiseen liittyvistä tapahtumista [SM16]. Tämä puolestaan tarkoittaa sitä, että kynnyskäsitteen ymmärtämisen suhteen ei välttämättä ole mahdollista määritellä mitään tiettyä ajankohtaa, koska käsitteet sulautuvat hiljalleen yhteen ja oppiminen tapahtuu hiljalleen pienemmissä osissa. On myös mahdollista, että opiskelijoiden ymmärrys jostakin käsitteestä saattaa olla edelleen heikko. Tällöin opiskelijat ovat voineet tulla toimeen ymmärtämättä käsitettä kunnolla, eivätkä välttämättä koe jotakin käsitettä haasteellisena.

Osa opiskelijoista ei välttämättä halua myöntää ohjelmoinnin perus- ja jatkokurssien aikana kohtaamiaan ongelmia. He saattavat pelätä arvostelua tai vähätellä kohtaamiaan ongelmia. On vaikea uskoa, että opiskelija ei kohtaa minkäänlaisia haasteita ohjelmoinnin perus- ja jatkokurssien aikana.

Haastatteluita tehtiin rajattu määrä (N=9), mikä ei välttämättä anna oikeanlaista kuvaa tutkittavasta aihealueesta. Haastavia käsitteitä ja muuttavia kokemuksia nousi tehtyjen haastattelujen perusteella esille runsaasti. Tästä huolimatta käsitteisiin liittyviä väärinkäsityksiä ilmeni niukasti. Isommalla otoksella haastateltavia mahdollisia väärinkäsityksiä voisi löytyä enemmän ja tulokset olisivat tällöin tarkempia.

## 6.4 Jatkotutkimus

Tutkielman empiirisessä osiossa tutkimus kohdistui Helsingin yliopistossa tietojenkäsittelytiedettä opiskeleviin opiskelijoihin, jotka ovat käyttäneet ohjelmoinnin perus- ja jatkokursseilla tehostettua oppipoikamallia. Suurimmassa osassa kynnyskäsitteisiin liittyvästä tutkimuksesta opetuksen sisältöä ja opetusmenetelmiä ei ole mainittu ollenkaan [SM16]. Konstruktiivisen oppimisteorian mukaan oppimiseen vaikuttavat ympäristölliset tekijät. Tehdyn tutkimuksen avulla saatiin ideoita ja ymmärrystä siitä, kuinka kynnyskäsitteitä pystytään tutkimaan lisää. Tämän takia mahdollista jatkotutkimusta olisi hyvä tehdä sitomalla se opetusmenetelmiin, jotta tulosten vertailu olisi luotettavampaa.

Jatkotutkimusta varten on mahdollista haastatella tietojenkäsittelytieteen laitoksen opettajia tai ohjelmointikursseilla toimineita kisällejä. Heiltä pystyy saamaan tietoa käsitteistä, jotka ovat opiskelijoille haastavia kurssien aikana. Opiskelijat ja kisällit pääsevät seuraamaan ja edistämään opiskelijoiden ohjelmointia pajaluokissa. He ovat säännöllisesti läsnä ja pääsevät perille opiskelijoiden erilaisista yksilöllisistä ongelmakohdista, joita viikoittaisiin tehtäviin mahdollisesti liittyy. He pääsevät myös näkemään, onko opiskelijoilla ongelmia jossakin aihealueessa tai saavatko opiskelijat ajattelua laajentavia kokemuksia oppimisen oikea-aikaisen tukemisen seurauksena.

Opiskelijoiden kehittymistä kannattaa tutkia pitkin kursseja, eikä vasta kurssien suorittamisen jälkeen. Mahdollinen tapa jatkotutkimukseen on

suorittaa useampia etukäteen rakennettuja kyselyitä ohjelmoinnin perus- ja jatkokurssia suorittaville opiskelijoille kurssien edetessä. Tämänkaltaiset kyselyt mahdollistavat isomman populaation kynnyskäsitteiden kartoittamisen. Haasteena tässä lähestymistavassa on kyselyiden rakentaminen mielekkäiksi.

Tutkielman empiirisen osion tutkimus ei ota huomioon opiskelijoita, jotka jättävät ohjelmoinnin perus- ja jatkokurssit kesken. Haastatteluita kannattaisi suorittaa myös kurssin keskeyttäneille opiskelijoille, koska näin saataisiin käsitys siitä, miksi kurssia ei suoritettu loppuun. Kurssin keskeyttänyt opiskelija on mahdollisesti kohdannut ongelmallisen kynnyskäsitteen jossakin vaiheessa kurssia, ja se on estänyt muiden olennaisten asioiden oppimisen.

Ohjelmoinnin perus- ja jatkokurssit hyödyntävät ohjelmointitehtävien yhteydessä TMC-palvelua (Test My Code), jonka avulla opiskelijat palauttavat viikoittaiset tehtävänsä ja saavat heti palautetta ratkaisuistaan. Palautteessa käy ilmi, onko tehtävät suoritettu onnistuneesti. Samalla palautteessa voidaan antaa neuvoja väärän ratkaisun korjaamiseksi. Tämä palaute toimii oikea-aikaisena palautteena opiskelijoille, tukee opiskelijoiden oppimista ja ohjaa heitä oikeaan suuntaan. Palvelu ottaa opiskelijoiden suostumuksella ajoittaisia tilannekatsauksia (snapshots) heidän kirjoittamastaan koodista. Opiskelijoiden koodia ja sen muodostumista seuraamalla ja analysoimalla pystyy pääsemään sisälle opiskelijoiden ohjelmointiin liittyvän ajattelun kehittymiseen. Ihantola ja kumppanit [IVA+15] ovat tehneet listan tutkimustavoitteista liittyen opiskelijoiden oppimisen analysointiin TMC:n avulla. Kyseistä tutkimusta ja sen tuloksia pystytään mahdollisesti hyödyntämään tutkimukseen kynnyskäsitteistä.

# 7 Yhteenveto

Teorian mukaan kynnyskäsitteet toimivat portteina syvempään ajatteluun aihealueessa. Niiden omaksuminen aiheuttaa opiskelijalla muuttavan kokemuksen, jonka seurauksena ymmärrys jostakin aihealueesta kehittyy [ML03].

Kynnyskäsitteiden piirteitä ovat muokkaavuus, peruuttamattomuus, yhdistävyys ja rajaavuus, joista tärkeimmät ovat muokkaavuus ja ongelmalisuus [RR09]. Kynnyskäsitteet jakavat paljon samoja piirteitä skeemateorian, kognitiivisen kuormituksen teorian ja konstruktiivisen oppimisteorian kanssa. Teorioissa yhteistä on teoria siitä, kuinka uusi tieto rakentuu henkilön aiempien kokemusten päälle.

Kirjallisuudessa kynnyskäsitteiden tutkimiselle on tunnistettu kaksi selkeää motiivia. Kynnyskäsitteiden avulla opettajat pystyvät käyttämään aikansa strategisesti tehokkaammin [AWW15]. Kynnyskäsitteet auttavat ennakoimaan mahdollisia esteitä oppimisessa ja niiden tunnistaminen auttaa varmistamaan, että esteiden muodostumista oppimisen aikana ehkäistään [AWW15]. Opettajat pystyvät kynnyskäsitteiden avulla huomioimaan, että opiskelijat kohtaavat kynnyskäsitteitä useassa vaiheessa opintojaan ja saamaan vaikutteita niihin liittyvistä ominaisuuksista. Kynnyskäsitteiden suhteen uskotaan siihen, että opetussuunnitelma, joka on rakennettu perustumaan tunnistettuihin kynnyskäsitteisiin ja motiiveihin, auttaa oppilaita huomattavasti kehittymään aihealueessa ammattilaisiksi [AWW15].

Kynnyskäsitteitä kohtaan on esitetty myös negatiivista palautetta kirjallisuudessa. Kynnyskäsitteet ovat piirteidensä takia vaikeita tunnistaa ja ne vaikuttavat kovin harvinaisilta, joten niistä ei välttämättä yksinään ole apua opetusmenetelmien ja opetussuunnitelman muokkaamiseen [MGH$^+$06]. Jotkut tutkijat ovat keskustelleet siitä, miten jotkin kynnyskäsitteisiin liittyvistä aihealueista vaikuttavat liian yleisiltä tai sidotuilta tiettyihin ohjelmointikieliin, kuten olio-ohjelmointi [SM16]. Kynnyskäsitteiden etsiminen tietyissä aihealueissa saattaa olla ajanhukkaa, jos kynnyskäsitteitä ei löydy aihealueesta tarpeeksi. Kynnyskäsitteet toimivat myös henkilökohtaisina ilmentyminä, eivätkä ne ole kaikille samoja. Käsitteistä ei pystytä sanomaan yleisesti ovatko ne kynnyskäsitteitä vai eivät, mutta on kuitenkin mahdollista tunnistaa käsitteitä, jotka toimivat todennäköisesti kynnyskäsitteinä.

Olemassaolevista tutkimuksista ei juurikaan löytynyt esimerkkejä siitä, kuinka tietoa kynnyskäsitteistä olisi yritetty hyödyntää käytännössä. Ainoa löytynyt esimerkki oli lähestymistapa, jossa kynnyskäsitteiden piirteistä monimutkaisuutta pyrittiin vähentämään. Monimutkaisuutta on pyritty vähentämään ohjelmoinnin perus- ja jatkokurssien alussa visuaalisia ohjelmointiympäristöjä, kuten Scratchia hyödyntämällä tai ennen kursseja pidettävällä johdantokurssilla (CS0) [VAW14]. Tämä monimutkaisuutta vähentävä lähestymistapa voidaan ajatella kynnyskäsitteisiin perustuvana opetustapana, sillä siinä pyritään vaikuttamaan kynnyskäsitteiden piirteisiin liittyviin ominaisuuksiin.

Kynnyskäsitteitä käsittelevä olemassaoleva tutkimus jakautuu yleisesti kolmeen laajempaan pääkategoriaa [SM16]:

1. Tutkimuksiin, joissa jätetään pedagogiikka huomiotta, toistetaan yleisiä lausuntoja opetuksesta ja kynnyskäsitteistä käyttäen lähteenä muita papereita.

2. Tutkimuksiin, joissa ehdotetaan tiettyjä taktiikoita toimia kynnyskäsitteiden suhteen.

3. Tutkimuksiin, joissa ehdotetaan interventioita tai taktiikoita toimia tiettyjen kynnyskäsitteiden suhteen.

Tutkielman empiirinen osio sisälsi havaintoja Helsingin yliopiston ohjelmoinnin perus- ja jatkokursseilla esiintyvistä kynnyskäsitteistä. Kynnyskäsitteiden piirteistä tärkeimmät ovat haasteellisuus ja muokkaavuus, joita tutkimuksen avulla pyrittiin selvittämään. Opiskelijoilta selvitettiin käsitteitä, jotka olivat ongelmallisia, väärinkäsityksiä tai muuttavia kokemuksia aiheuttavia. Kaikkia kolmea piirrettä omaavia käsitteitä olivat esimerkiksi olio-ohjelmointi, algoritminen ajattelu, perintä ja rajapinnat. Kyseiset käsitteet saattavat toimia kynnyskäsitteinä tietojenkäsittelytieteessä, koska niitä ovat tukemassa myös muista tutkimuksista saadut tulokset [SM16].

Lisäksi muita mahdollisia tutkimuksessa tunnistettuja kynnyskäsitteitä olivat seuraavat: debuggaus, ehtolausekkeet, isot ohjelmakokonaisuudet, isot kirjastot, rekursio, näkyvyys, Swing, syntaksi, taulukoiden läpikäynti, testien ymmärtäminen ja toistolausekkeet. Kaikki nämä käsitteet sisälsivät yksittäisiä kynnyskäsitteiden piirteitä, mutta kaikkia tunnistetuista käsitteistä ei kuitenkaan erinäisistä syistä voida pitää kynnyskäsitteinä. Se, että jokin käsite sisältää kynnyskäsitteisiin liittyviä piirteitä ei suoraan tarkoita, että se olisi kynnyskäsite jos se ei ole tarpeeksi olennainen käsiteltävän aihealueen kannalta. Tutkimustulosten valossa pystytään vahvistamaan aikaisemmissa tutkimuksissa havaittuja mahdollisia kynnyskäsitteitä.

Kirjallisuuden ja tutkimuksen oman empiirisen osion perusteella on melko varmaa, että kynnyskäsitteitä on olemassa ohjelmoinnin yhteydessä. Niiden olemassaoloa ovat tukemassa useista tutkimuksista peräisin olevat havainnot sekä opettajien, että oppilaiden näkökulmista. On kuitenkin kyseenalaista kannattaako kynnyskäsitteitä yrittää tunnistaa ja tutkia, koska saatu hyöty käytännössä saattaa olla vähäistä. Kynnyskäsitteitä ei ole pyritty aktiivisesti hyödyntämään opetuksessa, joka antaisi lisää tietoa onko kynnyskäsitteet toimiva lähestymistapa ohjelmoinnin opettamiseen. Kynnyskäsitteisiin perustuvan opetuksen haasteena on se, että kynnyskäsitteet eivät ole kaikille samoja. Toisaalta jotkin käsitteet toimivat todennäköisemmin kynnyskäsitteinä kuin toiset. On pyrittävä kokeilemaan, kuinka kynnyskäsitteitä on mahdollista hyödyntää opetuksessa, koska ilman käytännön havaintoja on niiden tutkimisen tärkeyttä kyseenalaistettava.

# Liitteet

## A Ohjelmoinnin peruskurssin oppimismatriisi

https://www.cs.helsinki.fi/courses/581325/matriisi

| **Ohjelmoinnin perusteet** | | | | |
|---|---|---|---|---|
| **Pääteemat** | **Esitiedot** | **Lähestyy oppimistavoitetta** | **Saavuttaa oppimistavoitteet** | **Syventää oppimistavoitteita** |
| Algoritmit ja ohjausrakenteet | Ei esitietoja (peruskoulun matematiikka) | • Tuntee ja osaa selittää ohjelmointikielen, kääntämisen ja tulkitsemisen idean.<br>• Osaa selittää sijoitusoperaation merkityksen ja algoritmin suorituksen etenemisen ajassa.<br>• Osaa simuloida yksinkertaisia algoritmeja. | • Osaa laatia yksinkertaisia algoritmeja.<br>• Osaa selittää käsitteen "algoritmin tila".<br>• Tajuaa miten loogiset lausekkeet ovat väittämiä algoritmin tilasta.<br>• Taitaa peruskontrollirakenteiden käytön.<br>• Oivaltaa syöttötietoja kyselevän ja tulostietoja kirjoittavan ohjelman idean ja osaa toteuttaa tällaisia.<br>• Tuntee taulukosta etsimisen idean ja osaa ohjelmoida peräkkäishaun, binäärihaun ja jonkin tavan järjestää taulukon alkiot. | • Ymmärtää miksi peräkkäishaku on lineaarinen operaatio, binäärihaku logaritminen ja järjestäminen neliöinen.<br>• Osaa laatia logiikaltaan ja ulkoasultaan tyylikkäitä ohjelmia. |
| Muuttujat ja tyypit | Algoritmin idea | • Oivaltaa muuttujan tyypin ja arvon idean. | • Osaa käyttää muuttujia ja kirjoittaa lausekkeita, joiden tyyppi on int, double, boolean ja String.<br>• Tuntee alkeistyypin ja viittaustyypin eron.<br>• Tuntee sijoitusyhteensopivuuden merkityksen ohjelmoinnissa.<br>• Tajuaa muodollisten parametrien ja paikallisten muuttujien käyttäytymisen<br>• Osaa käyttää luokkia muuttujien tyyppeinä.<br>• Osaa indeksoiden viitata dynaamisesti taulukkomuuttujan komponentteihin. | • Tuntee hieman tyypityksen historiaa ja osaa arvioida erilaisten ratkaisujen seurauksia. |
| Aliohjelmat | Algoritmin idea | • Oivaltaa algoritmin nimeämisen ja kutsumisen periaatteen. | • Tuntee hieman tyypityksen historiaa ja osaa arvioida erilaisten ratkaisujen seurauksia. | • Tietää, että Javan arvoparametrivälitys on vain yksi vaihtoehto parametrivälitykselle: on olemassa esimerkiksi kieliä, joissa käytetään viiteparametreja.<br>• Tuntee nimettyjen aliohjelmien historiaa. |
| Luokat, oliot ja kapselointi | Muuttujat, algoritmit, metodit, parametrivälitys | • Hahmottaa luokkamääritellyn olion piirustuksina, joista voidaan luoda erillisiä olioita. | • Osaa määritellä yksityisiä ilmentymämuuttujia ja ohjelmoida aksessoreita.<br>• Tuntee kapseloinnin tekniikan ja osaa sitä myös ohjelmoinnissa soveltaa.<br>• Tuntee käsitteen "olion tila".<br>• Tietää millainen elinkaari olioilla on ja miten se eroaa metodin paikallisten muuttujien elinkaaresta.<br>• Osaa välittää olioita parametreina.<br>• Tuntee automaattisen roskienkeruun merkityksen. | • Ymmärtää kapseloinnin tärkeyden ohjelmistotuotannossa ja ohjelman oikeellisuuteen pyrittäessä.<br>• Ymmärtää mitä seurauksia automaattisella roskienkeruulla on Java-kielen sovellettavuuteen ja mihin Java tästä syystä on hyvä, mihin kerta kaikkiaan sopimaton. |

# B Ohjelmoinnin jatkokurssin oppimismatriisi

https://www.cs.helsinki.fi/courses/582103/matriisi

| Ohjelmoinnin jatkokurssi | | | | |
|---|---|---|---|---|
| **Pääteemat** | **Esitiedot** | **Lähestyy oppimistavoitetta** | **Saavuttaa oppimistavoitteet** | **Syventää oppimistavoitteita** |
| Luokkamäärittelyn tekniikat | • Luokka, olio ja kapselointi (Ohjelmoinnin perusteet) | • Tietää että "static" liittyy luokkaan, "ei-static" ilmentymään. | • Osaa käyttää ohjelmoinnissa luokka- ja ilmentymämuuttujia sekä luokka- ja ilmentymämetodeita.<br>• Tuntee näkyvyyden säätelyn mahdollisuudet ja ongelmat: yksityinen kalusto, pakkaustason kalusto, näkyminen aliluokkaan, julkinen kalusto. | • Osaa jäsennellä ohjelman arkkitehtuuria luokka- ja oliotason tekniikoilla ja erilaisilla näkyvyyksillä. |
| Periytyminen | • Luokka, olio (Ohjelmoinnin perusteet) | • Alkaa tajuta, miten aliluokka perii yliluokan ominaisuudet, kentät ja metodit.<br>• Hahmottaa Objectluokasta alkavan luokkien puumaisen periytymishierarkian. | • Ymmärtää yliluokka-aliluokkasuhteen ja osaa myös ohjelmoida aliluokkia.<br>• Osaa arvostaa perittyjen kenttien kapselointia ja tietää miten ja milloin siitä voi poiketa.<br>• Tuntee ja osaa ohjelmoinnissa ottaa huomioon sen, että konstruktorit eivät periydy ja sen mitä tästä seuraa. Osaa käyttää ohjelmoinnissa ilmauksia this, super, this() ja super().<br>• Taitaa perittyjen metodien korvaamisen ja perittyjen kenttien peittämisen.<br>• Osaa käyttää viittaustyyppisille arvoille yhteisen yliluokan - abstraktin ja ei-abstraktin - tai rajapintaluokan tyyppistä muuttujaa ja tällä tavoin toteuttaa yleiskäyttöisiä luokkia ja metodeita. Tässä polymorfismin ymmärtäminen on välttämätöntä!<br>• Tietää millaisia lisäyksiä periytyminen tuo näkyvyyssääntöihin. | • Ymmärtää syvällisesti abstraktien luokkien, rajapintaluokkien ja polymorfismin käytön ohjelmien laadinnassa.<br>• Tuntee moniperinnän idean ja osaa kertoa, miten Javassa moniperinnän edut ilman haittoja pyritään saavuttamaan rajapintaluokkien avulla. Osaa osallistua väittelyyn tästä kysymyksestä. |
| Virhetilanteiden käsittely | • Taito laatia yksinkertaisia Java-ohjelmia (Ohjelmoinnin perusteet)<br>• On tavannut virhetilanteita | • Tietää, että virheet pitää jotenkin hoidella.<br>• Hahmottaa poikkeusten käsittelyn vaikutuksen ohjelman suoritukseen. | • Tuntee erilaisia tapoja käsitellä virhetilanteita.<br>• Tuntee tarkistettujen ja tarkistamattomien poikkeusten periaatteen ja osaa laatia ohjelmia, joissa poikkeuksia käsitellään Exception-tasolla | • Osaa ohjelmoida omia poikkeusluokkia ja arvioida, millaisin tavoin virhetilanteita kulloinkin on paras käsitellä. |
| Ohjelmointitekniikkaa | • Taito laatia yksinkertaisia Java-ohjelmia (Ohjelmoinnin perusteet) | • Oivaltaa tyyppimuunnoksen tarpeen. | • Tuntee alkeistyypit ja niiden väliset sijoitusyhteensopivuussäännöt sekä myös eksplisiittisen tyyppimuunnoksen laveammasta alkeistyypistä suppeampaan.<br>• Tuntee pakkausten periaatteen.<br>• Osaa lähdemateriaalia käyttäen laatia ohjelmia, jotka lukevat ja kirjoittavat tekstitiedostoja.<br>• On tutustunut pariin geneeriseen kokoelmaluokkaan, tajuaa niiden idean ja ja osaa materiaalia käyttäen myös ohjelmoida niillä.<br>• Tuntee erilaisia ohjelman toimintaperiaatteita: tietoja kyselevä ohjelma, komentotulkki, suodatin, tapahtumaohjattu ohjelma. Osaa myös ohjelmoida kolmella ensin mainitulla tyylillä. | • Osaa laatia rekursiivisia metodeita. Ymmärtää miksi rekursiivinen metodi Fibonaccin lukujen laskentaan on aikavaativuudeltaan eksponentiaalinen, binäärihaku logaritminen ja aavistelee ymmärtävänsä, miksi pikajärjestäminen on keskimääräisessä tapauksessa O(n*log n).<br>• Tuntee aktivaatiotietuepinon idean lohkorakenteisen kielen toteutuksessa.<br>• Kirjoittaa sujuvasti tiedostonkäsittelyohjelmia. Taitaa sarjallistamisen.<br>• Osaa ohjelmoida omia geneerisiä luokkia.<br>• Osaa laatia mallin mukaan yksinkertaisia GUI-ohjelmia. |

# C Haastattelupohja

**1. Johdanto**

Oletko suorittanut ohjelmoinnin perus- ja jatkokurssin Helsingin yliopistolla kun tehostettu oppipoikamalli on ollut käytössä?

Milloin?

Mikä on pääaineesi?

Oliko sinulla ennen opiskelujen aloittamista aiempaa ohjelmointikokemusta? Mitä, missä ja milloin?

Millainen mielikuva sinulla oli ohjelmoinnista ennen opiskelujen aloitusta?

**2. Kurssien aikana tapahtuneet asiat**

Millaiset tuntemukset sinulla oli ekojen viikkojen aikana tietojenkäsittelytieteen opiskeluissa ylipäätänsä, oliko opiskeluilmapiiri mukava?

Millaiselta ohjelmointi tuntui muutaman ensimmäisen viikon aikana ohjelmoinnin perus- ja jatkokursseilla?

Muistatko opetusta, missä järjestyksessä käsitteiden suhteen edettiin?

Mitkä olivat haastavia käsitteitä?

Liittyikö sinulla johonkin näistä käsitteistä mahdollisia väärinkäsityksiä?

Vaikuttivatko haastavat aihealueet tai väärinkäsitykset muiden asioiden oppimiseen?

Tuliko sinulle kurssien aikana tajuntaa ja ymmärrystä laajentavia kokemuksia, mitkä olisivat jääneet mieleesi?

**3. Suhtautuminen ohjelmointiin nykyään**

Mitä mieltä olet nykyään ohjelmoinnista, onko käsityksesi aihealueesta muuttunut paljon verrattuna aikaan ennen ohjelmoinnin aloittamista?

Oletko ohjelmoinut paljon ohjelmoinnin perus- ja jatkokurssien jälkeen?

Onko ohjelmoinnin oppiminen muuttanut arkipäiväistä tapaasi ajatella?

Hyödynnätkö nykyään ohjelmointiin tarvittavaa ajattelua myös muissa yhteyksissä?

# Lähteet


[AWW15] Alston, Peter, Walsh, David ja Westhead, Gary: *Uncovering "Threshold Concepts" in Web Development: An Instructor Perspective*. Trans. Comput. Educ., 15(1):2:1–2:18, maaliskuu 2015, ISSN 1946-6226. http://doi.acm.org/10.1145/2700513.

[BC07] Bennedsen, Jens ja Caspersen, Michael E.: *Failure Rates in Introductory Programming*. SIGCSE Bull., 39(2):32–36, kesäkuu 2007, ISSN 0097-8418. http://doi.acm.org/10.1145/1272848.1272879.

[BCN89] Brown, John Seely, Collins, A ja Newman, SE: *Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics*. Knowing, learning, and instruction: Essays in honor of Robert Glaser, 487, 1989.

[BEM+07] Boustedt, Jonas, Eckerdal, Anna, McCartney, Robert, Moström, Jan Erik, Ratcliffe, Mark, Sanders, Kate ja Zander, Carol: *Threshold Concepts in Computer Science: Do They Exist and Are They Useful?* Teoksessa *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, sivut 504–508, New York, NY, USA, 2007. ACM, ISBN 1-59593-361-1. http://doi.acm.org/10.1145/1227310.1227482.

[Bla06] Black, Toni R.: *Helping Novice Programming Students Succeed*. J. Comput. Sci. Coll., 22(2):109–114, joulukuu 2006, ISSN 1937-4771. http://dl.acm.org/citation.cfm?id=1181901.1181922.

[BR10a] Bareiss, Ray ja Radley, Martin: *Coaching via Cognitive Apprenticeship*. Teoksessa *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, sivut 162–

166, New York, NY, USA, 2010. ACM, ISBN 978-1-4503-0006-3. http://doi.acm.org/10.1145/1734263.1734319.

[BR10b] Bernard, H. Russell ja Ryan, Gery Wayne.: *Analyzing qualitative data : systematic approaches / H. Russell Bernard, Gery W. Ryan.* Sage Thousand Oaks, Calif, 2010, ISBN 9780761924906.

[BW00] Box, Roger ja Whitelaw, Michael: *Experiences when Migrating from Structured Analysis to Object-oriented Modelling.* Teoksessa *Proceedings of the Australasian Conference on Computing Education*, ACSE '00, sivut 12–18, New York, NY, USA, 2000. ACM, ISBN 1-58113-271-9. http://doi.acm.org/10.1145/359369.359372.

[Car00] Carey, Susan: *Science Education as Conceptual Change.* Journal of Applied Developmental Psychology, 21(1):13–19, 2000.

[CB07] Caspersen, Michael E. ja Bennedsen, Jens: *Instructional Design of a Programming Course: A Learning Theoretic Approach.* Teoksessa *Proceedings of the Third International Workshop on Computing Education Research*, ICER '07, sivut 111–122, New York, NY, USA, 2007. ACM, ISBN 978-1-59593-841-1. http://doi.acm.org/10.1145/1288580.1288595.

[CBH91] Collins, Allan, Brown, John Seely ja Holum, Ann: *Cognitive apprenticeship: making thinking visible.* American Educator, 6:38–46, 1991.

[CL10] Chan, Carol K. K. ja Lam, Ivan C. K.: *Conceptual Change and Epistemic Growth Through Reflective Assessment in Computer-supported Knowledge Building.* Teoksessa *Proceedings of the 9th International Conference of the Learning Sciences - Volume 1*,

ICLS '10, sivut 1063–1070. International Society of the Learning Sciences, 2010. http://dl.acm.org/citation.cfm?id=1854360.1854496.

[DG15] Diethelm, Ira ja Goschler, Juliana: *Questions on Spoken Language and Terminology for Teaching Computer Science*. Teoksessa *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, sivut 21–26, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-3440-2. http://doi.acm.org/10.1145/2729094.2742600.

[DM05] Davies, Peter ja Mangan, Jean: *Recognising Threshold Concepts: an Exploration of Different Approaches*. Teoksessa *European Association in Learning and Instruction Conference (EARLI)*, nide 23, 2005.

[Dri05] Driscoll, M.P.: *Psychology of Learning for Instruction*. Pearson Allyn and Bacon, 2005, ISBN 9780205441815. https://books.google.fi/books?id=6LiOAQAACAAJ.

[Dé06] Détienne, Françoise: *Assessing the cognitive consequences of the object-oriented approach: a survey of empirical research on object-oriented design by individuals and teams*. CoRR, abs/cs/0611154, 2006. http://dblp.uni-trier.de/db/journals/corr/corr0611.html#abs-cs-0611154.

[EMM+06] Eckerdal, Anna, McCartney, Robert, Moström, Jan Erik, Ratcliffe, Mark, Sanders, Kate ja Zander, Carol: *Putting Threshold Concepts into Context in Computer Science Education*. Teoksessa *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITICSE '06, sivut 103–107, New York, NY, USA, 2006. ACM,

ISBN 1-59593-055-8. http://doi.acm.org/10.1145/1140124.1140154.

[FB09] Fehér, Gábor ja Békés, András G.: *ECT: An Object-oriented Extension to Erlang*. Teoksessa *Proceedings of the 8th ACM SIGPLAN Workshop on ERLANG*, ERLANG '09, sivut 51–62, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-507-9. http://doi.acm.org/10.1145/1596600.1596608.

[GM10] Gomes, Anabela Jesus ja Mendes, António José: *A Study on Student Performance in First Year CS Courses*. Teoksessa *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, sivut 113–117, New York, NY, USA, 2010. ACM, ISBN 978-1-60558-820-9. http://doi.acm.org/10.1145/1822090.1822123.

[IVA+15] Ihantola, Petri, Vihavainen, Arto, Ahadi, Alireza, Butler, Matthew, Börstler, Jürgen, Edwards, Stephen H., Isohanni, Essi, Korhonen, Ari, Petersen, Andrew, Rivers, Kelly, Rubio, Miguel Ángel, Sheard, Judy, Skupas, Bronius, Spacco, Jaime, Szabo, Claudia ja Toll, Daniel: *Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies*. Teoksessa *Proceedings of the 2015 ITiCSE on Working Group Reports*, ITICSE-WGR '15, sivut 41–63, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-4146-2. http://doi.acm.org/10.1145/2858796.2858798.

[KV11] Kurhila, Jaakko ja Vihavainen, Arto: *Management, Structures and Tools to Scale Up Personal Advising in Large Programming Courses*. Teoksessa *Proceedings of the 2011 Conference on Information Technology Education*, SIGITE '11, sivut

3–8, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-1017-8. http://doi.acm.org/10.1145/2047594.2047596.

[LMRC13] Larkins, D. Brian, Moore, J. Christopher, Rubbo, Louis J. ja Covington, Laura R.: *Application of the Cognitive Apprenticeship Framework to a Middle School Robotics Camp.* Teoksessa *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, sivut 89–94, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-1868-6. http://doi.acm.org/10.1145/2445196.2445226.

[Luk94] Luker, Paul A.: *There's More to OOP Than Syntax!* Teoksessa *Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education*, SIGCSE '94, sivut 56–60, New York, NY, USA, 1994. ACM, ISBN 0-89791-646-8. http://doi.acm.org/10.1145/191029.191056.

[MBE+08] Moström, Jan Erik, Boustedt, Jonas, Eckerdal, Anna, McCartney, Robert, Sanders, Kate, Thomas, Lynda ja Zander, Carol: *Concrete Examples of Abstraction As Manifested in Students' Transformative Experiences.* Teoksessa *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, sivut 125–136, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-216-0. http://doi.acm.org/10.1145/1404520.1404533.

[MGH+06] Mead, Jerry, Gray, Simon, Hamer, John, James, Richard, Sorva, Juha, Clair, Caroline St. ja Thomas, Lynda: *A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition.* Teoksessa *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '06, sivut 182–194, New York, NY,

USA, 2006. ACM, ISBN 1-59593-603-3. http://doi.acm.org/10.1145/1189215.1189185.

[ML03] Meyer, Jan ja Land, Ray: *Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines.* University of Edinburgh Edinburgh, 2003. http://www.ed.ac.uk/etl/docs/ETLreport4.pdf.

[Mös93] Mössenböck, H.: *Object-Oriented Programming: In Oberon-2.* Springer-Verl, 1993, ISBN 9780387564111. https://books.google.fi/books?id=0yOzAAAAIAAJ.

[MS05] Merriënboer, Jeroen J. G. ja Sweller, John: *Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions.* Educational Psychology Review, 17(2):147–177, 2005, ISSN 1573-336X. http://dx.doi.org/10.1007/s10648-005-3951-0.

[OBL04] Or-Bach, Rachel ja Lavy, Ilana: *Cognitive Activities of Abstraction in Object Orientation: An Empirical Study.* SIGCSE Bull., 36(2):82–86, kesäkuu 2004, ISSN 0097-8418. http://doi.acm.org/10.1145/1024338.1024378.

[RR09] Rountree, Janet ja Rountree, Nathan: *Issues Regarding Threshold Concepts in Computer Science.* Teoksessa *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*, ACE '09, sivut 139–146, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc., ISBN 978-1-920682-76-7. http://dl.acm.org/citation.cfm?id=1862712.1862733.

[Sal14] Salmons, Janet E.: *Qualitative Online Interviews: Strategies, Design, and Skills*. Sage Publications, Inc., Thousand Oaks, CA, USA, 2nd painos, 2014, ISBN 1483332675, 9781483332673.

[SBE+12] Sanders, Kate, Boustedt, Jonas, Eckerdal, Anna, McCartney, Robert, Moström, Jan Erik, Thomas, Lynda ja Zander, Carol: *Threshold Concepts and Threshold Skills in Computing*. Teoksessa *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, sivut 23–30, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1604-0. http://doi.acm.org/10.1145/2361276.2361283.

[SF14] Szabo, Claudia ja Falkner, Katrina: *Neo-piagetian Theory As a Guide to Curriculum Analysis*. Teoksessa *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, sivut 115–120, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2605-6. http://doi.acm.org/10.1145/2538862.2538910.

[SKC07] Stokes, Alison, King, Helen ja C., Libarkin Julie: *Research in Science Education: Threshold Concepts*. Journal of Geoscience Education, 55(5):434–438, nov 2007.

[SKF13] Shinners-Kennedy, Dermot ja Fincher, Sally A.: *Identifying Threshold Concepts: From Dead End to a New Direction*. Teoksessa *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, sivut 9–18, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2243-0. http://doi.acm.org/10.1145/2493394.2493396.

[SM16] Sanders, Kate ja McCartney, Robert: *Threshold Concepts in Computing: Past, Present, and Future*. Teoksessa *Proceedings*

*of the 16th Koli Calling International Conference on Computing Education Research*, Koli Calling '16, sivut 91–100, New York, NY, USA, 2016. ACM, ISBN 978-1-4503-4770-9. http://doi.acm.org/10.1145/2999541.2999546.

[Sor10] Sorva, Juha: *Reflections on Threshold Concepts in Computer Programming and Beyond*. Teoksessa *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, sivut 21–30, New York, NY, USA, 2010. ACM, ISBN 978-1-4503-0520-4. http://doi.acm.org/10.1145/1930464.1930467.

[SSHL09] Sheard, Judy, Simon, S., Hamilton, Margaret ja Lönnberg, Jan: *Analysis of Research into the Teaching and Learning of Programming*. Teoksessa *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09, sivut 93–104, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-615-1. http://doi.acm.org/10.1145/1584322.1584334.

[Vag06] Vagianou, Evgenia: *Program Working Storage: A Beginner's Model*. Teoksessa *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, sivut 69–76, New York, NY, USA, 2006. ACM. http://doi.acm.org/10.1145/1315803.1315816.

[VAW14] Vihavainen, Arto, Airaksinen, Jonne ja Watson, Christopher: *A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success*. Teoksessa *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, sivut 19–26, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2755-8. http://doi.acm.org/10.1145/2632320.2632349.

[VLK12] Vihavainen, Arto, Luukkainen, Matti ja Kurhila, Jaakko: *Multi-faceted Support for MOOC in Programming.* Teoksessa *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, sivut 171–176, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1464-0. http://doi.acm.org/10.1145/2380552.2380603.

[VPL11] Vihavainen, Arto, Paksula, Matti ja Luukkainen, Matti: *Extreme Apprenticeship Method in Teaching Programming for Beginners.* Teoksessa *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, sivut 93–98, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0500-6. http://doi.acm.org/10.1145/1953163.1953196.

[VPLK11] Vihavainen, Arto, Paksula, Matti, Luukkainen, Matti ja Kurhila, Jaakko: *Extreme Apprenticeship Method: Key Practices and Upward Scalability.* Teoksessa *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, sivut 273–277, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0697-3. http://doi.acm.org/10.1145/1999747.1999824.

[VVLP13] Vihavainen, Arto, Vikberg, Thomas, Luukkainen, Matti ja Pärtel, Martin: *Scaffolding Students' Learning Using Test My Code.* Teoksessa *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, sivut 117–122, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2078-8. http://doi.acm.org/10.1145/2462476.2462501.

[Win06] Wing, Jeannette M.: *Computational Thinking.* Commun. ACM, 49(3):33–35, maaliskuu 2006, ISSN 0001-0782. http://doi.acm.

org/10.1145/1118178.1118215.

[WL14] Watson, Christopher ja Li, Frederick W.B.: *Failure Rates in Introductory Programming Revisited*. Teoksessa *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, sivut 39–44, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2833-3. http://doi.acm.org/10.1145/2591708.2591749.

[WLG14] Watson, Christopher, Li, Frederick W.B. ja Godwin, Jamie L.: *No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success*. Teoksessa *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, sivut 469–474, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2605-6. http://doi.acm.org/10.1145/2538862.2538930.

[XHB14] Xu, Anbang, Huang, Shih Wen ja Bailey, Brian: *Voyant: Generating Structured Feedback on Visual Designs Using a Crowd of Non-experts*. Teoksessa *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, sivut 1433–1444, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2540-0. http://doi.acm.org/10.1145/2531602.2531604.

[ZBM+09] Zander, Carol, Boustedt, Jonas, McCartney, Robert, Moström, Jan Erik, Sanders, Kate ja Thomas, Lynda: *Student Transformations: Are They Computer Scientists Yet?* Teoksessa *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09, sivut 129–140, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-615-1. http://doi.acm.org/10.1145/1584322.1584337.