Johanna Virkkunen

Ohjelmoijana menestymistä edesauttavat tekijät

Tietotekniikan kandidaatintutkielma

18. toukokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Johanna Virkkunen

Yhteystiedot: johanna.m.virkkunen@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Ohjelmoijana menestymistä edesauttavat tekijät

Title in English: Factors that contribute to success as a programmer

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 24+0

Tiivistelmä: Tämä kirjallisuuskatsauksena toteutettu kandidaatintutkielma kartoittaa ohjelmoijana menestymistä edesauttavia tekijöitä. Ohjelmoijana menestymistä tarkastellaan useasta, kuten ohjelmistoalan ammattilaisena sekä koulutuksen tarjoamissa ohjelmointikursseissa menestymisen, näkökulmasta. Tutkielmassa avataan ohjelmoijan ja ohjelmoijana menestymisen käsitteet sekä käsitellään ohjelmoijana menestymistä edesauttavia tekijöitä viiden pääkäsitteen avulla: tekniset tietotaidot, kokemus, kognitiivinen kyvykkyys ja matemaattiset valmiudet sekä pehmeät taidot.

Avainsanat: ohjelmoijana menestyminen, tekniset tietotaidot, laadukas ohjelmakoodi, kokemus, kognitiivinen kyvykkyys, matemaattiset valmiudet, pehmeät taidot

Abstract: This Bachelor's Thesis, implemented as a literature review, identifies the factors that contribute to success as a programmer. Success as a programmer is viewed from several perspectives, such as success as a software professional and success in programming courses offered by education. The Thesis introduces the concepts of programmer and success as a programmer and discusses the factors contributing to success as a programmer using five main concepts: technical know-how, experience, cognitive ability and mathematical ability, and soft skills.

Keywords: success as a programmer, technical know-how, high quality program code, experience, cognitive ability, mathematical ability, soft skills

Sisällys

1 Johdanto

Menestyvän ohjelmoijan silmät liukuvat ohjelman lähdekoodin yli, ja muutamassa sekunnissa hän voi saada syvällisen ymmärryksen ohjelmatiedostoihin järjestetyistä abstrakteista symboleista ja tekstistä. Menestyvällä ohjelmoijalla on kyky seurata tietovirran suhteita ohjelmointielementtien välillä sekä havaita virheitä nopeasti. (Parnin, Siegmund ja Peitek 2017). Ohjelmakoodin kanssa työskentelyn lisäksi menestyvä ohjelmoija työskentelee tehokkaasti toisten ihmisten kanssa, koska harva ohjelmoija työskentelee yksin (Goodliffe 2014).

Tämän kirjallisuuskatsauksena toteutetun kandidaatintutkielman tavoitteena on kartoittaa aiemman kirjallisuuden avulla ohjelmoijana menestymistä edesauttavia tekijöitä. Ohjelmoijana menestymistä edesauttavia tekijöitä on tärkeää tutkia auttaakseen ihmisiä tekemään parempia päätöksiä opintojensa tai uriensa suhteen (Evans ja Simkin 1989). Tietokoneteollisuus korostaa yhä enemmän ohjelmoijana menestymistä edesauttavien tekijöiden tunnistamista. Näiden tekijöiden tunnistamisen avulla voidaan keskittyä niihin opiskelijoihin, ketkä ovat alttiita saavuttamaan menestystä tällä alalla. (Bubica ja Boljat 2014). Kilpailun tietokoneohjelmoinnin työpaikoista lisääntyessä, ohjelmoijana menestymistä edesauttavien tekijöiden tunnistaminen ja kehittäminen ovat tulleet yhä tärkeämmiksi alan työnantajille (Bailey ja Mitchell 2006).

Luvussa 2 käsitellään ohjelmoijana menestymistä ja siinä edesauttavia tekijöitä jaettuna aiheittain omiin alalukuihinsa. Luvussa 2.1 määritellään ohjelmoijan ja ohjelmoijana menestymisen käsitteet sekä kerrotaan erilaisia ohjelmoijan tehtäviä. Luvussa 2.2 käsitellään ohjelmoijana menestymistä edesauttavia teknisiä tietotaitoja, kuten laadukasta ohjelmakoodia ja ohjelmoijien tärkeiksi kokemia aiheita, sekä uuden opettelemisen merkitystä. Luvussa 2.3 käsitellään kokemuksen merkitystä ohjelmoijana menestymisessä. Luvussa 2.4 käsitellään ohjelmoijana menestymistä edesauttavaa kognitiivista kyvykkyyttä ja matemaattisia valmiuksia. Luvussa 2.5 käsitellään pehmeitä taitoja kolmen eri pääkäsitteen avulla. Luku 3 on yhteenveto.

2 Ohjelmoijana menestyminen

Ohjelmoijana menestymistä sekä siinä edesauttavia tekijöitä käsitellään tässä luvussa useasta näkökulmasta. Näiden eri ulottuvuuksien avulla pyritään saamaan monipuolinen ja kattava kuva ohjelmoijana menestymisen taustalla olevista tekijöistä. Huomion arvoista on, että ohjelmoijana menestymisen merkitys voi vaihdella riippuen ohjelmoijan tehtävästä, ja että erilaiset tekijät ovat tarpeellisia erilaisissa tietokoneohjelmoinnin tehtävissä (Bishop-Clark 1995).

2.1 Ohjelmoijan ja ohjelmoijana menestymisen käsitteet

Treude ym. (2012) tuovat esille tyypillisen tietokoneohjelmoijan stereotyypin muuttuneen eristyneestä tietokonekirjallisuudesta oppivasta yksilöstä henkilöksi, joka hallitsee persoonaansa pitkin sosiaalisia koodaussivustoja. Jotkin lähteet, kuten Cambridge University Press (2021b), määrittelevät ohjelmoijan käsitteen tarkoittamaan erityisesti ohjelmoijan ammatissa toimimista. Tässä kandidaatintutkielmassa käytetään Kielikone Oy:n (2020) mukaista määritelmää ohjelmoijan käsitteestä yleisesti tietokoneohjelmien laatijana. Täten ohjelmoijana menestymisellä tarkoitetaan tässä kirjallisuuskatsauksessa yleisesti menestymistä tietokoneohjelmien laatimisessa. Huomattavaa on, että ohjelmoijana menestymisen käsite elää ajan ja ohjelmoijan käsitteen elämisen mukana. Ohjelmoijana menestymistä tarkastellaan useasta, kuten ohjelmistoalan ammattilaisena sekä koulutuksen tarjoamissa ohjelmointikursseissa menestymisen, näkökulmasta.

Ohjelmoijan tehtäväkirjo on laaja. Ohjelmoija kirjoittaa ja testaa ohjelmakoodia, joka mahdollistaa tietokonesovellusten ja ohjelmistojen tarkoituksenmukaisen toiminnan. Ohjelmoija muuttaa ohjelmistokehittäjien ja -insinöörien luomat suunnitelmat ohjelmasta tietokoneen ymmärtämiksi toimintaohjeiksi. Lisäksi ohjelmoijan tehtäviin kuuluu virheellisten ohjelmakoodirivien löytämistä ja korjaamista. Toisinaan ohjelmoijan tehtäviin voi kuulua myös suunnittelutehtäviä ohjelmistoon ja ohjelmakoodin toteuttamiseen liittyen. (Bureau of Labor Statistics 2021).

Kumar (2008) mainitsee ohjelmoijan tehtävän yleistämisen vaikeuden, ja toteaakin, että oh-

jelmoijat ovat ennemminkin erikoisasiantuntijoita. Ohjelmoija voidaan luokitella esimerkiksi keskittymisalueensa, käyttämänsä ohjelmointikielen ja koodausalustansa perusteella. Voidaan nähdä erilaisia ohjelmoijia, kuten sovelluskehittäjä; Web-ohjelmoija; tietokantaohjelmoija; verkko-ohjelmoija; kielikohtainen, kuten Java-ohjelmoija; alustakohtainen, kuten Linux-ohjelmoija sekä toiminnanohjausjärjestelmäohjelmoija (engl. *ERP Programmer*) (Kumar 2008). Osittain juuri tehtäväkirjon laajuuden vuoksi on monenlaisia ohjelmoijana menestymistä edesauttavia tekijöitä.

2.2 Tekniset tietotaidot

Tekniset tietotaidot edustavat merkittävästi ohjelmoijana menestymistä edesauttavia tekijöitä. Tekniset taidot käsittävät laajoja sekä suppeampia kokonaisuuksia, mukaan lukien seuraavat osaamisalueet: ohjelmointikielet, käyttöjärjestelmät, ohjelmointiympäristö, laitteisto, tietoverkko, tekstinkäsittely, tietokoneidenkäyttö, tulostimet ja taulukkolaskenta. Uusien taitojen opetteleminen on tärkeää sekä hyvät ohjelmoijat oppivat nopeasti. (Kumar 2008). Hyvän ohjelmoijan tietämyksen Kumar (2008) toteaa olevan syvää ja käsittävän tietoa monipuolisesti niin käyttöjärjestelmistä, prosessoreista, kielistä kuin ominaisuuksistakin. Tietämys on ennenkaikkea niin laajaa, että kaiken tarkistamista ulkoisista lähteistä, kuten Internetistä ja dokumentaatioista, ei tarvita (Kumar 2008). Lisäksi menestyksekkäällä ohjelmoijalla on tasapainoisesti käytännön ja teoreettista tietoa sekä kyltymätön uteliaisuus uusia ohjelmoinnin asioita kohtaan (Kumar 2008). Seuraavassa luvussa käsitellään laadukasta ohjelmakoodia ja ohjelmakoodin ongelmia.

2.2.1 Laadukas ohjelmakoodi

Laadukkaan ohjelmakoodin tuottaminen on yksi olennaisimmista ohjelmoijan tehtävistä, ja siten myös yksi olennaisimmista ohjelmoijana menestymistä edesauttavista tekijöistä. Kumar (2008) mainitsee hyvän ohjelmoijan toivottujen ominaisuuksien listauksessaan kyvyn analysoida ja kehittää monimutkaista koodia sovelluksia varten. Menestyksekästä ohjelmakoodia laatiessa on tärkeää olla laatutietoinen, jolla tarkoitetaan tietoisuutta kehitettävän koodin laadusta, sekä olla tinkimättä siitä. Laatutietoisuuteen sisältyy myös yksityiskohtien huomioiminen, perusteellinen testaaminen, laadukas kommentointi ja dokumentointi sekä vir-

heetön koodi (Kumar 2008).

Ohjelmakoodin laadun mittareita on useita, ja niitä on käytetty jo viiden vuosikymmenen ajan. Mittareita on kehitetty ohjelmakoodin laadun ominaisuuksille, kuten koheesiolle, kytkennälle (engl. *coupling*), kompleksisuudelle ja luettavuudelle. (Sharma ja Spinellis 2020; Pantiuchina, Lanza ja Bavota 2018). Tämänhetkiset ohjelmakoodin laadun mittarit kuitenkin kärsivät monenlaisista heikkouksista, kuten mittarin puutteellisesta kyvystä tehdä lupaamiaan asioita (Sharma ja Spinellis 2020). Koska yleisesti hyväksyttyjä mittareita koodin laadun arvioimiseksi ei ole, hyvän ja laadukkaan ohjelmakoodin määritteleminen ja selittäminen on haaste (Börstler ym. 2018).

Ohjelmakoodin laadulla on kuitenkin useita puolia, joita voidaan teemoittain jakaa esimerkiksi seuraaviin kategorioihin: luettavuus, rakenne, ymmärrettävyys, ylläpitokyky, oikeellisuus, dokumentointi, dynaaminen käyttäytyminen, testattavuus, virheellisyys, ylläpidettävyys ja monipuolisuus. Näistä kategorioista tärkeimpinä pidettiin Börstlerin (2018) tutkimuksessa luettavuutta ja rakennetta. Aiemmassa kappaleessa mainitun laatutietoisuuden sisältämien tekijöiden lisäksi laadukas ohjelmakoodi onkin helppolukuista, sisennettyä ja nimeämiskäytännöltään laadukasta, eikä se sisällä hyödytöntä koodia. Rakenteeltaan laadukas ohjelmakoodi on hyvin jäsenneltyä, modulaarista ja eriteltyä (engl. *decomposition*) (Börstler ym. 2018).

Seuraavaksi ohjelmakoodiesimerkit Java-ohjelmointikielellä tehdystä aliohjelmasta, joka laskee henkilön painoindeksin. Ensimmäisen esimerkin tarkoitus on ilmentää laadukasta ohjelmakoodia näiden aiemmin kuvailluin ominaisuuksin; aliohjelma on kattavasti kommentoitu, sen nimi ja parametrit ovat kuvaavuudellaan nimeämiskäytännöiltään laadukkaat sekä aliohjelma on sisennetty (ks. esimerkki 1). Toisen esimerkin tarkoituksena on ilmentää laadutonta ohjelmakoodia rikkomalla seuraavia edellämainittuja ominaisuuksia; aliohjelmaa ei ole kommentoitu, sen nimi ja parametrit eivät epäkuvaavuudellaan ilmennä laadukasta nimeämiskäytäntöä, eikä sitä ole sisennetty. Lisäksi helppolukuisuutta on heikennetty lisäämällä ja poistamalla välilyöntejä laskutoimitusriviltä (ks. esimerkki 2).

Esimerkki 1

```
/**
```

```
 * Ohjelma laskee painoindeksin ohjelman syötteenä saadun pituuden
 * ja painon avulla
 * @param pituus sen henkilön pituus, jonka painoindeksiä lasketaan
 * @param paino sen henkilön paino, jonka painoindeksiä lasketaan
 * @return henkilön painoindeksi
 */
public static double laskePainoindeksi(double pituus, double paino) {
    double painoindeksi = paino / (pituus * pituus);
    return painoindeksi;
}
```

Esimerkki 2

```
public static double laskuri(double x, double y) {
double z =y/(x *x);
return z;
}
```

Testaaminen on tärkeässä roolissa laadukkaan ohjelmakoodin laatimisessa. Ohjelmistotestaus on termi, joka kattaa laajan kirjon erilaisia toimintoja, kuten yksikkötestauksen (engl. *unit testing*) sekä hyväksymistestauksen (engl. *acceptance testing*) (Bertolino 2007). Bertolino (2007) tiivistää testauksen tarkoittavan ohjelmistojärjestelmän tarkkailua sen tarkoituksellisen käyttäytymisen todentamiseksi sekä mahdollisten toimintahäiriöiden tunnistamiseksi. Testausta käytetäänkin laajalti teollisuudessa laadunvarmistuksessa, sen tarjotessa realistista palautetta ohjelmiston käyttäytymisestä (Bertolino 2007). Athanasiou ym. (2014) toteavat testikoodilla olevan samanlaisia vaatimuksia ylläpitoon liittyen kuin tuotantokoodillakin; on tärkeää varmistaa koodin olevan helppo lukea ja ymmärtää, sen muokattavuuden helpottamiseksi.

Laadukas ohjelmakoodi sisältää vähän virheitä. Ohjelmoija kuitenkin tekee virheitä, ja niiden laatu sekä määrä vaihtelevat. Ray ym. (2014) tutkivat ohjelmointikielen vaikutusta ohjelmiston laatuun keräämällä dataa GitHubista 729 projektista, 80 miljoonasta lähdekoodirivistä, 29 000 tekijältä, 17 ohjelmointikielessä. Tutkimuksessa saatiin selville, että 88,53% ohjelmistovioista (engl. *software bugs*) liittyivät yleisiin ohjelmointivirheisiin (Ray ym. 2014). Ray ym. (2014) mainitsevat yleisiin ohjelmointivirheisiin sisältyvän muun muassa virheellisen virheidenkäsittelyn, tyyppivirheet, kirjoitusvirheet, käännösvirheet ja datanalustusvir-

heet. Lopuista ohjelmistovioista 5,44% liittyy virheelliseen muistinhallintaan; 1,99% samanaikaisuuden vikoihin ja 0,11% algoritmisiin virheisiin.

Keuning, Heeren ja Jeuring (2017) tutkivat ohjelmakoodin laatuongelmia hyödyntäen Blackbox-tietokannasta otettua dataa, sisältäen yli kaksi miljoonaa aloittelevien ohjelmoijien Java-ohjelmaa neljän viikon ajanjaksolta. Keuning, Heeren ja Jeuring (2017, suomennos minun) toteavat, että ”aloittelevat ohjelmoijat kehittävät ohjelmia, joissa on merkittävä määrä ohjelmakoodin laadun ongelmia”. Rayn ym. (2014) GitHubin käyttäjistä kerätystä datasta saaduista tuloksista poiketen, haastavimmiksi kategorioiksi opiskelijoille Keuning, Heeren ja Jeuring (2017) mainitsevat ne, jotka käsittelevät algoritmeja ja rakennetta. Näihin liittyviä ohjelman laatuongelmia ovat muun muassa koodin monistaminen (engl. *code duplication*), saavuttamaton koodi, ohjausrakenteiden epäsopiva valinta, kirjastofunktioiden käyttämättömyys, liian monimutkaiset lausekkeet ja sopimattomien tietotyyppien käyttö, liian pitkät metodit sekä luokat ilman selkeää tarkoitusta (Keuning, Heeren ja Jeuring 2017).

Ohjelmakoodin virheisiin liittyy olennaisesti käsite koodihajut (engl. *code smells*). Koodihajut on hyvin tunnettu käsite kuvaamaan ohjelmakoodin hajoamisen oireita tai muita ohjelmakoodin laatuun liittyviä ongelmia, jotka voivat haitata ohjelmakoodin ylläpidon kannalta olennaisia tekijöitä, kuten koodin ymmärrettävyyttä ja muokattavuutta (Yamashita ja Moonen 2013). Yamashita ja Moonen (2013) tutkivat 73 ohjelmistoammattilaisen tietämystä koodihajuista sekä koodihajuihin liittyvien työkalujen hyödyllisyyttä. 32% vastaajista totesivat, etteivät olleet koskaan kuulleet koodihajuista. Vastaajien arvioimista koodihajuista neljänä yleisimpänä pidettiin monistunutta koodia, pitkää metodia, vahingollista monimutkaisuutta ja suurta luokkaa. Vain 4% 50 vastaajasta käyttivät varsinaisesti koodihajuihin liittyviä työkaluja, kuten erityisiä koodihajujen tunnnistustyökaluja sekä näiden yhteydessä koodihajujen poistamiseksi refaktorointityökaluja.

Ohjelmakoodin koodihajujen ja virheiden tiedostaminen sekä välttäminen auttavat ohjelmoijana menestymisessä parantaessaan ohjelmakoodin laatua. Bell (1976, suomennos minun) toteaakin: ”Jopa asiantuntijaohjelmoijat kirjoittavat ohjelmia, jotka sisältävät virheitä, ja kyky etsiä ja poistaa virheitä säilyy tärkeänä koko ohjelmointiaikana.”. Seuraavassa luvussa käsitellään ohjelmoijien tärkeiksi kokemia taitoja.

2.2.2 Ohjelmoijien tärkeiksi kokemat taidot

Surakka (2007) on tutkinut, mitkä aiheet ja taidot ovat tärkeitä ohjelmistokehittäjille. Ohjelmistokehittäjä on termi, joka käsittää sekä ohjelmoijat että ohjelmistoinsinöörit (Surakka 2007). Surakan (2007) tutkimuksen pääkontribuutiona on ollut päivittää Lethbridgen tulokset vuodelta 1998. Lethbridgen (2000) tutkimuksessa (n=186) selvitettiin, mitkä koulutukselliset aiheet ovat osoittautuneet vastaajille eniten tärkeiksi heidän urallaan, ja tunnistaa aiheet, joilla heidän koulutusta tai tämänhetkistä tietämystä voitaisiin parantaa. Lethbridgen (2007) tulosten mukaan viisi tärkeintä aihetta olivat tietorakenteet, spesifit ohjelmointikielet, ohjelmistosuunnittelu- ja mallit, vaatimusten kerääminen ja analysointi sekä ohjelmistoarkkitehtuuri.

Surakan (2007) tutkimuksessa esitellään kolmen eri ryhmän; ohjelmistokehittäjien (n=11), professorien ja lehtorien (n=19) sekä maisteriopiskelijoiden (n=24) vastauksista saadut tulokset. Ryhmien jäsenet arvioivat 42:n eri aiheiden ja taitojen tärkeyttä, yhtenä tarkoituksena selvittää, mitkä niistä ovat tärkeimpiä ohjelmistokehittäjille (Surakka 2007). Suurin osa tuloksista olivat odotettuja; esimerkiksi Webiin liittyvät aiheet ja taidot ovat arvioitu tärkeämmäksi kuin 1998, jonka jälkeen Webin käyttö on lisääntynyt dramaattisesti (Surakka 2007). Tilastollisesti merkittävät erot osioissa tietokone- ja tietoturvallisuus sekä hajautetut järjestelmät, johtuvat mahdollisesti myös Webin lisääntyneestä käytöstä (Surakka 2007). Surakan (2007) tuloksista selviää, että tietorakenteet ja algoritmit olivat tärkeimpinä pidettyjä aiheita sekä ohjelmistokehittäjien että professoreiden ja lehtoreiden ryhmissä. Toisella sijalla on ohjelmistokehittäjien ryhmässä proseduraalinen ohjelmointi, kun taas professoreiden ja lehtoreiden ryhmässä olio-ohjelmointi. Molemmissa ryhmissä viiden tärkeimpänä pidetyn aiheen joukossa ovat suunnittelu ja implementointi.

2.2.3 Uuden opetteleminen

Ohjelmoijana menestymistä edesauttava tietotekninen osaaminen ei pysy vakiona ajan kuluessa. Kumar (2008, suomennos minun) toteaakin, että ”Paljon tietokoneteknologiaa on vanhentumassa ja uutta teknologiaa on tulossa ulos päivittäin.”. Perusosaamisen hankkimisen jälkeen tärkeää ohjelmoijana menestymisen kannalta on jatkuva uusien tietoteknisten

asioiden opetteleminen.

Ohjelmoijana, kuten kaikilla muillakin aloilla menestyminen, vaatii vuosia jatkuvaa ja tarkoituksellista harjoittelua. Verrattuna muihin asiantuntija-aloihin, ohjelmoijana menestyminen perustuu epävakaisiin, lyhytaikaisiin ja usein siirtämättömiin elementteihin. (Parnin, Siegmund ja Peitek 2017). Ohjelmoijana menestyäkseen vuosien harjoittelun tulisikin sisältää monipuolisti uusien asioiden opettelemista.

Kumar (2008) käsittelee asioita, joita ohjelmoijat voivat kehittää tullakseen hyviksi. Kumar (2008) mainitsee jatkuvan oppimisen, harjoittelun ja kovan työn tärkeyden. Hyväksi ohjelmoijaksi tullakseen tulisi jatkaa oppimista eri lähteistä sekä käyttää aikaa jatkokoulutukseen- ja kursseihin (Kumar 2008). Kumar (2008) nostaa esille myös liiallisen kiintymyksen välttämisen; ohjelmoijalla voi usein olla taipumus kiintyä liikaa esimerkiksi käyttämäänsä tietokoneeseen, ohjelmointikieleen tai käyttöjärjestelmään. Tämä voi Kumarin (2008) mielestä aiheuttaa mukavuusvyöhykkeen, jolta poistuminen tuottaa mielipahaa ja vaikeuttaa sopeutumista uuteen työympäristöön. Kumar (2008, suomennos minun) ottaakin kantaa ajan hermoilla olemiseen: ”On erittäin tärkeää olla tietoinen asioista, jotka saattavat muuttua lyhyellä aikavälillä ja olla valmistautunut siihen.”.

2.3 Kokemus

Kumar (2008) kysyy, täytyykö tuhansia rivejä ohjelmaa kirjoittanutta kutsua hyväksi ohjelmoijaksi. Kokemusta onkin tutkittu eri alueilla ohjelmistotuotannossa, erityisesti ohjelmoinnissa; voidaan melkein yleismaailmallisesti todeta, että asiantuntijat suoriutuvat aloittelijoita paremmin erilaisissa tehtävissä ja tilanteissa (Dieste ym. 2017). Dieste ym. (2017) tuovat kuitenkin esille, että kokemuksen ympärillä on paljon epävarmuutta siitä, onko kokemus yhteydessä parempaan suorituskykyyn. Seuraavassa kappaleessa käsitellään tutkimusta, jossa kokemus oli yhteydessä parempaan suoritukseen.

Hagan ja Markham (2000) tutkivat opiskelijoilla aiemman ohjelmointikielikokemuksen vaikutusta johdantotason ohjelmointikurssissa menestymiseen. Ohjelmointikurssi toteutettiin Java-ohjelmointikielellä, ja siihen kuului viisi arviointikomponenttia. Arviointikomponenteihin sisältyi useita luokkia hyödyntävän pelin toteuttamista sekä testejä, jotka sisälsivät

muun muassa monivalintatehtäviä, ohjelmakoodin täydentämistä, luokkakaavioiden piirtämistä sekä testin kirjoittamista. Tulosten perusteella aiempi kokemus hyödytti merkittävästi opiskelijoiden menestymistä johdantotason ohjelmointikursseissa. Lisäksi suurimmassa osassa arviointikomponenteissa menestyminen oli sitä parempaa, mitä useampaa ohjelmointikieltä opiskelija oli käyttänyt tai opiskellut. (Hagan ja Markham 2000).

Dieste ym. (2017) kertovat, että on raportoitu tapauksia, joissa kokemuksella ei ole positiivista vaikutusta. Curtis ym. (1979) havaitsivat, että vuosien ohjelmointikokemus ei korreloi ohjelman rekonstruointitehtävässä menestymisen kanssa. Lisäksi Muller ja Padberg (2004) raportoivat, että kokemustaso ja toteutukseen käytetty aika eivät korreloi koehenkilöiden suorittaessa koodaustehtävää testivetoisessa (engl. *Test-Driven Development*) kehityksessä.

Kokemuksen merkitys ohjelmoijana menestymistä edesauttavana tekijänä vaihtelee myös sen mukaan, mistä se on hankittu. Dieste ym. (2017) toteavatkin tutkimuksensa tuloksissa, että teollisuudessa hankitulla ohjelmointikokemuksella ei näytä olevan minkäänlaista vaikutusta laatuun ja tuottavuuteen. Toisin sanoen; teollisuudessa työskentelevät saavuttavat korkeamman suorituskyvyn itsenäisesti riippumatta heidän kokemuksestaan teollisuudessa (Dieste ym. 2017). Korkeakoulussa hankitun ohjelmointikokemuksen merkitys taas on huomattavaa; prosentuaalisesti jokainen harjoitteluvuosi lisää 4% kasvua laatuun ja tuottavuuteen (Dieste ym. 2017).

Aiemman tutkimuksen perusteella voidaan havaita kokemuksen merkitys ohjelmoijana menestymistä edesauttavana tekijänä, mutta ei kaikilla sen alueilla. Kokemuksen merkitys voi vaihdella myös yksilöittäin, sillä toiset ohjelmoijat omaksuvat ohjelmoinnin konsepteja nopeammin kuin ikätoverinsa (Parnin, Siegmund ja Peitek 2017). Kuten Nichols (2019) toteaa, kokemuksen merkitys on selkeästi tärkeä, mutta sen arvo on rajallinen.

2.4 Kognitiivinen kyvykkyys ja matemaattiset valmiudet

Kognitiivisella kyvykkyydellä tarkoitetaan taitoja, jotka liittyvät havaitsemiseen, oppimiseen, muistiin, ymmärtämiseen, tietoisuuteen, päättelyyn, arviointiin, intuitioon ja kieleen liittyvien tehtävien suorittamiseen (American Psychological Association 2020). Kognitiivinen kyvykkyys on pätevä suorituskyvyn ennustaja yleisesti sekä tietokoneohjelmoinnin tie-

tämyksen piirissä (Evans ja Simkin 1989). Erilaisia kognitiivisia kyvykkyyksiä tarvitaan ohjelmoijana menestymisessä sekä siinä edesauttavien matemaattisten valmiuksien saavuttamisessa (Männamaa ym. 2012). Tässä luvussa käsitellään ohjelmoijana menestymistä edesauttavia kognitiivisia kyvykkyyksiä ja matemaattisia valmiuksia.

Kognitiivista kyvkkyyttä ja matemaattista valmiutta on hyödynnetty ohjelmointimenestymistä tutkittaessa. Erdoganin, Aydinin ja Kabacan (2008) tutkimuksen päätarkoituksena oli tutkia tekijöitä, jotka ennustavat saavutuksia ohjelmoinnissa. Tutkimuksessa selvitettiin 48 lukioikäisen opiskelijan C-ohjelmointikielisissä ohjelmointikursseissa menestymisen ja luovuuden, ongelmanratkaisun, yleisen soveltuvuuden, tietokoneasenteiden ja matemaattisten saavutusten välistä suhdetta. Merkittävimmät suhteet löydettiin ohjelmointisaavutusten ja yleisen soveltuvuuden (selitti 87,2% vaihtelusta opiskelijoiden suorituksissa), ja matemaattisten saavutusten (selitti 19,8% vaihtelusta opiskelijoiden suorituksissa) välillä. Yleistä soveltuvuutta mitattiin suorituskykytestillä, joka mittaa kognitiiviseen kyvykkyyteen liitettyjä tekijöitä, kuten analyyttistä ajattelua, abstraktia ajattelua ja tilakäsitystä. Matemaattisia saavutuksia mitattiin ensimmäisen lukukauden arvosanojen avulla. Tuloksista voidaan nähdä kognitiivisen kyvykkyyden ja matemaattisten valmiuksien merkitys ohjelmoijana menestymisessä.

Tietokoneohjelmointia pidetään yleisesti älyllisenä toimintana, ja tietokoneohjelman kirjoittaminen vaatii erityisiä kognitiivisia kykyjä (Kumar 2008; Bell 1976). Niitä tarvitaan useassa vaiheessa tietokoneohjelmaa kirjoittaessa; ohjelmointiongelman muotoilemisessa, sen jakamisessa pienempiin osaongelmiin, suunnitelman luomisessa sen ratkaisemiseksi sekä minimoidessa, havaitessa ja korjatessa loogisia virheitä (Bell 1976). Lisäksi ohjelmoijalle tärkeässä laskennallisessa ajattelussa (engl. *Computational Thinking*) tarvitaan erilaisia kognitiivisia prosesseja, kuten avaruudellista hahmotuskykyä sekä yleistä älykkyyttä (Ambrosio ym. 2014). Laskennallisella ajattelulla tarkoitetaan kykyä tulkita maailmaa algoritmisesti kontrolloituna sisääntulojen (engl. *input*) muuttumisena ulostuloiksi (engl. *output*) (Denning 2009).

Ambrosio ym. (2014) mainitsevat useita kognitiivisia kyvkkyyksiä, jotka ovat arvokkaita tietokoneohjelmointia opetellessa. Näitä ovat muun muassa luetunymmärtäminen, kriittinen ja systeemiajattelu, suunnitteleminen ja ongelmanratkaisu, luovuus, älyllinen uteliaisuus sekä

useat erilaiset päättelyn muodot. Etenkin ongelmanratkaisu on useissa lähteissä liitetty ohjelmointiin ja ohjelmointitaitoihin (Niemelä ja Valmari 2018; Parnin, Siegmund ja Peitek 2017; Ambrosio ym. 2014; Gomes ym. 2006). Asiantuntijaohjelmoijat hyödyntävätkin erilaisia ongelmanratkaisustrategioita paremmin kuin aloittelevat ohjelmoijat, sekä mukauttavat näitä sujuvammin erilaisiin tilanteisiin (Parnin, Siegmund ja Peitek 2017).

”Niin ikään yhteys matemaattisen kyvykkyyden ja ohjelmoinnin välillä on yleisesti hyväksytty.” (Erdogan, Aydin ja Kabaca 2008, suomennos minun). Gomes ym. (2006) toteavat kyvyn lukea, kirjoittaa, kuunnella, keskustella ja tulkita matematiikkaa, olevan tärkeä osa ohjelmoinnin opettelemista. Heidän tutkimuksessaan tuli esille, että ohjelmointitaitojen puutteeseen liittyi syvä puute matemaattisesta tiedosta ja taidosta, ja että usein jälkimmäinen oli suurin syy ensimmäiseen. Lisäksi Gomes ym. (2006) mainitsevat, että esteeksi ohjelmointiongelman ratkaisemiseksi voi muodostua tekstuaalisen ratkaisun muuttaminen matemaattiseen kieleen.

Ohjelmoinnissa menestymisen kannalta olennaisia matemaattisia taitoja ja aiheita on useita. Näistä Gomes ym. (2006) mainitsevat seuraavia asioita: matemaattiset yhteydet, numeeriset järjestelmät ja numeroteoria, laskeminen ja arviointi, kuviot ja funktiot, algebra, geometria, tilastot, todennäköisyydet ja mittaaminen. Matemaattisilla yhteyksillä tarkoitetaan kykyä luoda yhteyksiä jokapäiväisen elämän konkreettisten tilanteiden, ja matematiikan ja ohjelmoinnin välille (Gomes ym. 2006). Numeroteoriaan liittyy käsitteet, kuten vajaat luvut, täydelliset luvut, kolmioluvut ja neliöt. Kuvioihin ja funktioihin liittyää kyky ilmaista ratkaisu todelliseen ongelmaan funktion avulla. Geometrista osaamista ja geometristen mallien soveltamista tarvitaan useiden ohjelmointiongelmien ratkaisemiseksi (Gomes ym. 2006).

Hallitakseen ohjelmistoja ja parantaakseen niiden laatua, tietojenkäsittelijät ovat ehdottaneet matemaattisten aiheiden, kuten logiikan, kielioppien ja joukko-opin, olevan sopiva matemaattinen perusta (Niemelä ja Valmari 2018). Myös Surakan (2007) tutkimuksesta ilmenee logiikan tärkeys muiden tärkeiden aiheiden, kuten automaattien, diskreetin matematiikan ja algoritmien, lisäksi. Niemelä ja Valmari (2018) toteavat, että oppimiskohteiden ohjelmointia varten tulisi siirtyä enemmän kohti diskreettiä matematiikkaa, johon liittyy esimerkiksi algoritmit, tietorakenteet ja logiikka.

2.5 Pehmeät taidot

Pehmeät taidot (engl. *soft skills*) täydentävät teknisiä taitoja, ja erilaisia pehmeitä taitoja tarvitaan erilaisissa ohjelmoijan tehtävissä, kuten suunnittelussa, ohjelmakokoodin kirjoittamisessa ja testaamisessa (Ahmed, Capretz ja Campbell 2012; Lewis ym. 2008). Pehmeillä taidoilla tarkoitetaan persoonallisuudenpiirteitä ja asenteita, jotka ohjaavat ihmisen käyttäytymistä. Lisäksi pehmeitä taitoja ovat esimerkiksi organisointitaidot sekä kyky vuorovaikuttaa toisten ihmisten kanssa (Ahmed, Capretz ja Campbell 2012; Roan ja Whitehouse 2007). Tässä luvussa käsitellään ohjelmoijana menestymistä edesauttavia pehmeitä taitoja kolmen pääkäsitteen avulla, jotka ovat ihmissuhdetaidot, arvot ja persoonallisuus.

Ohjelmistokehitys on tänä päivänä suurimman osan ajasta tiimityötä, johon osallistuu useita erilaisia tehtäviä tekeviä henkilöitä. Lisäksi ohjelmistokehitys sisältää ongelmanratkaisua, jossa erilaiset ihmiset näkevät ongelman erilaisista näkökulmista (Capretz ja Ahmed 2010). Näin ollen siinä menestyäkseen ohjelmoijalle on hyödyksi kehittyneet ihmissuhdetaidot. Ihmissuhdetaidoilla tarkoitetaan kykyä olla vuorovaikutuksessa muiden ihmisten kanssa sekä suotuisissa että hankalissa olosuhteissa (Oxford University Press 2021; Ahmed, Capretz ja Campbell 2012). Ihmissuhdetaitoja tarvitaan myös tehokkaan asiakkaiden ja käyttäjien kanssa tapahtuvan yhteistyön mahdollistamiseksi (Capretz ja Ahmed 2010).

Ihmissuhdetaidoista sekä Ahmed, Capretz ja Campbell (2012) että Kumar (2008), tuovat esille kommunikoinnin merkityksen ohjelmoijalle. Ahmed, Capretz ja Campbell (2012) tutkivat 500 IT-alan työpaikkailmoituksen pehmeiden taitojen vaatimuksia. Näistä 140 oli tietokoneohjelmoijan työpaikkailmoituksia, joissa 90%:ssa mainittiin vaatimuksena kommunikointi. Kommunikoinnilla tarkoitettiin kykyä välittää tietoa siten, että se otetaan hyvin vastaan ja ymmärretään. Kumar (2008) toteaa hyvän ohjelmoijan kommunikoinnin olevan monipuolista, sisältäen hyvän puhumisen ja kuuntelemisen taidon. Lisäksi hän mainitsee kommunikoinnin sisältävän hyvän kehonkielen, kielen, eleet, kieliopin ja sanankäytön, sekä taidon keskustella kirjallisesti, suullisesti, sähköpostitse, puheluin, kasvotusten ja ryhmässä.

Ohjelmoijalla tulee olla kyky työskennellä ryhmässä erilaisissa rooleissa, kuten joukkuepelaajana ja johtajana (Capretz ja Ahmed 2010; Kumar 2008). Tähän kuuluu muiden ymmärtäminen ja kyky olla sympaattinen viestinnässään, sekä kyky käsitellä ihmisiä erilaisista taus-

toista. Lisäksi hyvä ohjelmoija kykenee ratkaisemaan konflikteja ja välttämään näitä. Ryhmässä toimiessaan hyvä ohjelmoija on myös valmis ottamaan vastuuta, ja tarvittaessa johtajan roolin kriittisiä ongelmia ratkaistaessa. Työpaikalla hyvä ohjelmoija on avulias ja halukas jakamaan tietoaan ilman hyvien vinkkien piilottamista tai niiden esittämistä epämääräisesti. (Kumar 2008).

Arvoilla tarkoitetaan ihmisten käytöstä kontrolloivia uskomuksia siitä, mikä on oikein, väärin ja tärkeintä heidän elämässään (Cambridge University Press 2021c). Ohjelmoijana menestymistä edesauttaviin arvoihin liittyy käsite teoreettinen arvouskomus (engl. *theoretical value belief*). Arvouskomukset ohjaavat yksilöitä reagoimaan eri tavoin eri tilanteissa sekä tarjoavat motivaation toiminnan taustalla syvällisesti henkilökohtaisella tasolla. Teoreettinen arvouskomus on joukko henkilökohtaisia standardeja, jotka ohjaavat yksilöä järkevien päätösten osalta. Henkilö, joka ilmentää tätä ominaisuutta, arvostaa järjestystä, ongelmanratkaisua ja todisteita, ja on motivoitunut totuuden löytämisestä. (Cegielski ja Hall 2006; Ajzen 1991).

Cegielski ja Hall (2006) tutkivat ohjelmointisuorituksessa menestymistä ennustavia tekijöitä. Ohjelmointisuoritusta mitattiin kolmella kirjallisella kokeella, kahdella ajastetulla laboratoriokokeella sekä kattavalla ohjelmointitehtävällä, joka ulottui ongelman muotoilusta toteuttamiseen. Teoreettista arvouskomusta mitattiin käyttämällä Allportin, Lindzeyn ja Vernonin (1970) arvomittaria. Tutkimuksen tuloksista selvisi, että teoreettinen arvouskomus on pätevä ohjelmointisuorituksen indikaattori.

Persoonallisuudella tarkoitetaan yksilön erityistä ominaisuuksien yhdistelmää, joka tekee kyseisen henkilön erilaiseksi muista, ja joka ilmenee henkilön käyttäytymisestä, tunteista ja ajatuksista (Cambridge University Press 2021a). Cegielskin ja Hallin (2006) tutkimuksen (ks. edellinen kappale) kolmantena tutkimuskysymyksenä oli, onko persoonallisuus pätevä ennustaja olio-ohjelmoinnin ohjelmointitehtävässä menestymisessä. Yksilön persoonallisuutta arvioitiin neljän komponentin avulla. Komponentit olivat itsetunto, yleinen minäpystyvyys, neuroottisuus ja hallintakäsitys (engl. *locus of control*), ja niitä mitattiin käyttämällä tunnettuja asteikkoja, kuten Rosenbergin (1989) itsetuntomittaria ja Judgesin (1998) yleistettyä minäpystyvyysasteikkoa. Mittauksista saadut tulokset yhdistettiin yhdeksi persoonallisuuspisteytykseksi. Tulosten mukaan persoonallisuus on voimakas ohjelmointisuorituksen en-

nustaja. Aiemmista tutkimuksista poiketen, persoonallisuusuudella oli suurempi ennustava voima kuin kognitiivisella kyvykkyydellä (Cegielski ja Hall 2006). Lisäksi tuloksista selvisi, että henkilö, jolla on vahva peroonallisuus, menestyy yhtä todennäköisesti ohjelmointisuorituksessa kuin henkilö, jolla on vahva teoreettinen arvouskomus. Vahvaan peroonallisuuteen liittyy korkea itsetunto, korkea minäpystyvyys, korkea hallintakäsitys ja matala neuroottisuus (Cegielski ja Hall 2006).

Persoonallisuuden piirteistä tietoisuuden ja avoimuuden on havaittu ennustavan ohjelmointitaitoja. Näiden kahden merkitys ohjelmointitehtävissä menestymisen kannalta on muuttunut vuosien saatossa; avoimuuden merkitys on kasvanut viime vuosina ja tietoisuuden merkitys on laskenut. Persoonallisuuden piirteistä introversion on havaittu merkittävimpänä ohjelmointitaitojen ennustajana. Introvertit ovat varautuneita yksilöitä, joilla on matala sosiaalisuuden taso sekä taipumus keskittyä sisäiseen minäänsä sosiaalisen ympäristönsä sijaan. Näiden ominaisuuksien seurauksena he saattavat olla osaavampia syvällistä tarkastelua edellyttävissä ohjelmistokehitystehtävissä. (Gnambs 2015).

Eri tieteenaloihin liittyvässä ongelmanratkaisussa menestyminen ei välttämättä riipu ainoastaan kognitiivisesta kyvykkyydestä, vaan siihen saattaa vaikuttaa persoonallisuustekijät, kuten itsetunto ja minäpystyvyys. Nämä tekijät voivat saada henkilön, jolla on vähemmän kognitiivista kyvykkyyttä, työskentelemään kovemmin ja ylittämään kognitiivisesti kyvykkäämmän henkilön. (Cegielski ja Hall 2006). Puuttuvaa ohjelmoijana menestymistä edesauttavaa kognitiivista kyvkkyyttä voidaan siis kompensoida erilaisilla persoonallisuustekijöillä.

3 Yhteenveto

Tämän kandidaatintutkielman tarkoituksena oli aiemman tutkimuksen avulla kartoittaa ohjelmoijana menestymistä edesauttavia tekijöitä. Tässä tutkielmassa ohjelmoijana menestymistä edesauttavia tekijöitä kartoitettiin neljän eri pääkäsitteen avulla: tekniset tietotaidot, kokemus, kognitiivinen kyvykkyys ja matemaattiset valmiudet sekä pehmeät taidot. Yhteenvetona voidaan sanoa, että ohjelmoijana menestymistä edesauttavia tekijöitä on paljon, ja ne käsittävät asioita useilta eri tieteenaloilta.

Tutkielmassa esitetyn nojalla voidaan sanoa, että ohjelmoijana menestymistä edesauttavat teknisistä tietotaidoista useat tekijät, kuten laadukkaan ohjelmakoodin tuottaminen ja uuden opetteleminen. Laadukasta ohjelmakoodia tuottaakseen tulee olla tietoinen laadukkaan ohjelmakoodin ominaisuuksista, sekä kyetä etsimään ja poistamaan virheitä. Kokemuksen merkitys vaihtelee tilanteittain ja yksilöittän. Kognitiivisista kyvykkyyksistä esimerkiksi ongelmanratkaisu sekä älyllinen uteliaisuus ovat tärkeitä ohjelmoijana menestymistä edesauttavia tekijöitä. Matemaattiset aiheet, kuten logiikka, joukko-oppi ja diskreetti matematiikka ovat eduksi ohjelmoijana menestymiselle. Pehmeistä taidoista ohjelmoijana menestymistä edesauttavat ihmissuhdetaidot, arvot ja persoonallisuus. Persoonallisuuden piirteistä esimerkiksi introversio ja avoimuus ovat tärkeitä ohjelmoijana menestymisen kannalta.

Voidaan huomata, että ohjelmoijana voi menestyä monenlaiset henkilöt, ja että osa tekijöistä, kuten laadukkaan ohjelmakoodin tuottaminen, ovat ainakin osittain harjoiteltavissa. Ohjelmointikykyyn tarvittavien taitojen harjoitteluun ja kehittymiseen tarvitaan sitkeyttä, päättäväisyyttä ja pitkäjänteisyyttä (Gomes ym. 2006). Näin ollen yksilöllä on mahdollisuus tehdä aktiivisia toimia tullakseen paremmaksi ja menestyneemmäksi ohjelmoijaksi.

Tutkielman heikkoutena voidaan nähdä riittämätön aiheen rajaaminen. Ohjelmoijana menestymistä edesauttavia tekijöitä tutkittiin usealta eri tieteenalalta sekä useasta näkökulmasta. Näin ollen eri tieteenalojen ohjelmoijana menestymistä edesauttavien tekijöiden tarkastelu jäi jossain määrin pintapuoliseksi. Ohjelmoijana menestymistä edesauttavien tekijöiden kartoittaminen ainoastaan esimerkiksi ohjelmistoammattilaisen näkökulmasta olisi voinut eheyttää tutkielman sisältöä.

Lähteet

Ahmed, Faheem, Luiz Fernando Capretz ja Piers Campbell. 2012. “Evaluating the demand for soft skills in software development”. *It Professional* 14 (1): 44–49.

Ajzen, Icek. 1991. “The Theory of Planned Behavior”. *Organizational Behavior and Human Decision Processes* 50 (joulukuu): 179–211. https://doi.org/10.1016/0749-5978(91)90020-T.

Allport, Gordon W, Philip E Vernon ja Gardner Lindzey. 1970. “Study of Values”.

Ambrosio, Ana Paula, Leandro S Almeida, Joaquim Macedo ja Amanda Helena Rodrigues Franco. 2014. “Exploring core cognitive skills of computational thinking”.

American Psychological Association. 2020. *APA Dictionary of Psychology.* Saatavilla WWW-muodossa, https://dictionary.apa.org/, haettu sana ”cognitive ability”, viitattu 17.3.2021.

Athanasiou, D., A. Nugroho, J. Visser ja A. Zaidman. 2014. “Test Code Quality and Its Relation to Issue Handling Performance”. *IEEE Transactions on Software Engineering* 40 (11): 1100–1125. https://doi.org/10.1109/TSE.2014.2342227.

Bailey, Janet L., ja R. B. Mitchell. 2006. “Industry Perceptions of the Competencies Needed by Computer Programmers: Technical, Business, and Soft Skills”. *Journal of Computer Information Systems* 47:28–33.

Bell, D. 1976. “Programmer selection and programming errors”. *The Computer Journal* 19, numero 3 (tammikuu): 202–206. ISSN: 0010-4620. https://doi.org/10.1093/comjnl/19.3.202. eprint: https://academic.oup.com/comjnl/article-pdf/19/3/202/954049/190202.pdf. https://doi.org/10.1093/comjnl/19.3.202.

Bertolino, A. 2007. “Software Testing Research: Achievements, Challenges, Dreams”. Teoksessa *Future of Software Engineering (FOSE ’07),* 85–103. https://doi.org/10.1109/FOSE.2007.25.

Bishop-Clark, Catherine. 1995. “Cognitive style, personality, and computer programming”. *Computers in Human Behavior* 11 (2): 241–260. ISSN: 0747-5632. https://doi.org/https://doi.org/10.1016/0747-5632(94)00034-F.

Bubica, N., ja I. Boljat. 2014. “PREDICTORS OF NOVICES PROGRAMMERS’ PERFORMANCE”. Teoksessa *ICERI2014 Proceedings,* 1536–1545. 7th International Conference of Education, Research and Innovation. Seville, Spain: IATED, 17-19 November, 2014. ISBN: 978-84-617-2484-0.

Bureau of Labor Statistics. 2021. “Occupational Outlook Handbook”. Viitattu 30. huhtikuuta 2021. https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm.

Börstler, Jürgen, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner ja Bonnie MacKellar. 2018. “"I know it when I see it"Perceptions of Code Quality: ITiCSE’17 Working Group Report”. Teoksessa *Proceedings of the 2017 ITiCSE Conference on Working Group Reports,* 70–85.

Cambridge University Press. 2021a. *Cambridge Academic Content Dictionary.* Saatavilla WWW-muodossa, https://dictionary.cambridge.org/dictionary/english, haettu sana ”personality”, viitattu 26.3.2021.

———. 2021b. *Cambridge Advanced Learner’s Dictionary & Thesaurus.* Saatavilla WWW-muodossa, https://dictionary.cambridge.org/dictionary/english, haettu sana ”programmer”, viitattu 7.3.2021.

———. 2021c. *Cambridge Business English Dictionary.* Saatavilla WWW-muodossa, https://dictionary.cambridge.org/dictionary/english, haettu sana ”value”, viitattu 7.4.2021.

Capretz, Luiz Fernando, ja Faheem Ahmed. 2010. “Why do we need personality diversity in software engineering?” *ACM SIGSOFT Software Engineering Notes* 35 (2): 1–11.

Cegielski, Casey G., ja Dianne J. Hall. 2006. “What Makes a Good Programmer?” *Commun. ACM* (New York, NY, USA) 49, numero 10 (lokakuu): 73–75. ISSN: 0001-0782. https://doi.org/10.1145/1164394.1164397.

Curtis, B., T. Love, P. Milliman ja S. Sheppard. 1979. “Modern Coding Practices and Programmer Performance”. *Computer* (Los Alamitos, CA, USA) 12, numero 12 (joulukuu): 41–49. ISSN: 1558-0814. https://doi.org/10.1109/MC.1979.1658575.

Denning, Peter J. 2009. “The Profession of IT Beyond Computational Thinking”. *Commun. ACM* (New York, NY, USA) 52, numero 6 (kesäkuu): 28–30. ISSN: 0001-0782. https://doi.org/10.1145/1516046.1516054.

Dieste, Oscar, Alejandrina M Aranda, Fernando Uyaguari, Burak Turhan, Ayse Tosun, Davide Fucci, Markku Oivo ja Natalia Juristo. 2017. “Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study”. *Empirical Software Engineering* 22 (5): 2457–2542.

Erdogan, Yavuz, Emin Aydin ja Tolga Kabaca. 2008. “Exploring the psychological predictors of programming achievement.” *Journal of Instructional Psychology* 35 (3).

Evans, Gerald E., ja Mark G. Simkin. 1989. “What Best Predicts Computer Proficiency?” *Commun. ACM* (New York, NY, USA) 32, numero 11 (marraskuu): 1322–1327. ISSN: 0001-0782. https://doi.org/10.1145/68814.68817.

Gnambs, Timo. 2015. “What makes a computer wiz? Linking personality traits and programming aptitude”. *Journal of Research in Personality* 58:31–34.

Gomes, Anabela, Lilian Carmo, Emilia Bigotte ja António Mendes. 2006. “Mathematics and programming problem solving”. Teoksessa *3rd e-learning conference–computer science education,* 1–5. Citeseer.

Goodliffe, Pete. 2014. *Becoming a Better Programmer: A Handbook for People Who Care About Code.* "O'Reilly Media, Inc.".

Hagan, Dianne, ja Selby Markham. 2000. “Does it help to have some programming experience before beginning a computing degree program?” Teoksessa *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education,* 25–28.

Judge, Timothy A, Edwin A Locke, Cathy C Durham ja Avraham N Kluger. 1998. “Dispositional effects on job and life satisfaction: the role of core evaluations.” *Journal of applied psychology* 83 (1): 17.

Keuning, Hieke, Bastiaan Heeren ja Johan Jeuring. 2017. “Code quality issues in student programs”. Teoksessa *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education,* 110–115.

Kielikone Oy. 2020. *MOT Kielitoimiston Sanakirja.* Saatavilla WWW-muodossa, http://www.kielitoimistonsanakirja.fi/, haettu sana ”ohjelmoija”, viitattu 7.3.2021.

Kumar, Suthikshn. 2008. “The Rise and Fall of a Good Programmer”. *Ubiquity* (New York, NY, USA) 2008, numero April (huhtikuu). https://doi.org/10.1145/1376142.1366323.

Lethbridge, T. C. 2000. “What knowledge is important to a software professional?” *Computer* 33 (5): 44–50. https://doi.org/10.1109/2.841783.

Lewis, Tracy L., Wanda J. Smith, France Bélanger ja K. Vernard Harrington. 2008. “Are Technical and Soft Skills Required? The Use of Structural Equation Modeling to Examine Factors Leading to Retention in the Cs Major”. Teoksessa *Proceedings of the Fourth International Workshop on Computing Education Research,* 91–100. ICER ’08. Sydney, Australia: Association for Computing Machinery. ISBN: 9781605582160. https://doi.org/10.1145/1404520.1404530.

Muller, Matthias M., ja Frank Padberg. 2004. “An Empirical Study about the Feelgood Factor in Pair Programming”. Teoksessa *Proceedings of the Software Metrics, 10th International Symposium,* 151–158. METRICS ’04. USA: IEEE Computer Society. ISBN: 0769521290.

Männamaa, Mairi, Eve Kikas, Kätlin Peets ja Anu Palu. 2012. “Cognitive correlates of math skills in third-grade students”. *Educational Psychology* 32 (1): 21–44.

Nichols, William R. 2019. “The End to the Myth of Individual Programmer Productivity”. *IEEE Software* 36 (5): 71–75. https://doi.org/10.1109/MS.2019.2908576.

Niemelä, Pia, ja Antti Valmari. 2018. “Elementary Math to Close the Digital Skills Gap”. Teoksessa *Proceedings of the 10th International Conference on Computer Supported Education - Volume 1: CSEDU,* 154–165. INSTICC, SciTePress. ISBN: 978-989-758-291-2. https://doi.org/10.5220/0006800201540165.

Oxford University Press. 2021. *Lexico.com.* Saatavilla WWW-muodossa, https://www.lexico.com/definition, haettu sana ”interpersonal skills”, viitattu 8.4.2021.

Pantiuchina, Jevgenija, Michele Lanza ja Gabriele Bavota. 2018. “Improving code: The (mis) perception of quality metrics”. Teoksessa *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 80–91. IEEE.

Parnin, Chris, Janet Siegmund ja Norman Peitek. 2017. “On the Nature of Programmer Expertise”. Teoksessa *Proceedings of the 28th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2017, Delft, The Netherlands, July 1-3, 2017,* toimittanut Luke Church ja Felienne Hermans, 16. Psychology of Programming Interest Group. http://ppig.org/library/paper/nature-programmer-expertise.

Ray, Baishakhi, Daryl Posnett, Vladimir Filkov ja Premkumar Devanbu. 2014. “A large scale study of programming languages and code quality in github”. Teoksessa *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 155–165.

Roan, Amanda, ja Gillian Whitehouse. 2007. “Women, information technology and ‘waves of optimism’: Australian evidence on ‘mixed-skill’ jobs”. *New Technology, Work and Employment* 22 (1): 21–33. https://doi.org/https://doi.org/10.1111/j.1468-005X.2007.00181.x.

Rosenberg, Morris. 1989. *Society and the Adolescent Self-Image.* Wesleyan University Press, Middletown, CT.

Sharma, Tushar, ja Diomidis Spinellis. 2020. “Do We Need Improved Code Quality Metrics?” *arXiv preprint arXiv:2012.12324.*

Surakka, Sami. 2007. “What Subjects and Skills Are Important for Software Developers?” *Commun. ACM* (New York, NY, USA) 50, numero 1 (tammikuu): 73–78. ISSN: 0001-0782. https://doi.org/10.1145/1188913.1188920.

Treude, Christoph, Fernando Figueira Filho, Brendan Cleary ja Margaret-Anne Storey. 2012. “Programming in a socially networked world: the evolution of the social programmer”. *The Future of Collaborative Software Development,* 1–3.

Yamashita, Aiko, ja Leon Moonen. 2013. “Do developers care about code smells? An exploratory survey”. Teoksessa *2013 20th working conference on reverse engineering (WCRE),* 242–251. IEEE.