Joonas Puuppo

GitHub Copilotin käyttö pariohjelmoinnissa

Tietotekniikan kandidaatintutkielma

26. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Puuppo

Yhteystiedot: joonas.a.j.puuppo@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: GitHub Copilotin käyttö pariohjelmoinnissa

Title in English: Use of GitHub Copilot in Pair Programming

Työ: Kandidaatintutkielma

Sivumäärä: 20+0

Tiivistelmä: Vuonna 2021 julkaistua ohjelmointityökalu GitHub Copilotia on markkinoitu ”tekoälypariohjelmoijana”. Perinteisesti pariohjelmointi perustuu prosessiin, jossa kaksi ihmisohjelmoijaa työskentelee saman tehtävän parissa kahdella eri abstraktion tasolla. Tässä tutkielmassa selvitetään, missä määrin Copilot voi toimia pariohjelmointiprosessissa toisen ihmisohjelmoijan paikalla. Kirjallisuuskatsauksen tuloksena havaitaan, että nykyisellään Copilot soveltuu pariohjelmointiin heikosti. Pariohjelmoinnille ominainen roolijako ei toteudu, ja kommunikaatio kahden ohjelmoijan välillä jää puuttumaan.

Avainsanat: pariohjelmointi, tekoälyavusteinen ohjelmointi, GitHub Copilot

Abstract: GitHub Copilot, released in 2021, has been marketed as an ”AI pair programmer”. Traditionally, pair programming is based on a process in which two human programmers work on the same task on two different levels of abstraction. The aim of this thesis is to find out to what extent can Copilot function in the pair programming process in place of one human programmer. The results of the literature review indicate that the current version of Copilot does not work well as a pair programmer. The characteristic division of roles in pair programming cannot be applied properly and the communication between two programmers does not occur.

Keywords: pair programming, AI-assisted programming, GitHub Copilot

Kuviot

Sisällys

1 Johdanto

Erilaiset tekoälysovellukset ovat arkipäiväistyneet kovaa vauhtia viime vuosina. Tekstikehotteen pohjalta kuvia generoivat työkalut, kuten DALL-E, Midjourney ja Stable Diffusion, sekä tekstimuotoisia vastauksia tuottava ChatGPT-chatbotti, ovat herättäneet paljon julkista keskustelua tekoälysovellusten mahdollisuuksista, haasteista ja uhkista. Yksi suosittu puheenaihe on se, miten tekoäly tulee mullistamaan työelämää. IT-alan ja käytännön ohjelmointityön kannalta olennainen tekoälysovellus on vuonna 2021 julkaistu ja 2022 laajemmin käyttöön otettu GitHub Copilot.

Tässä tutkielmassa tulen tarkastelemaan GitHub Copilotin käyttöä pariohjelmoinnin näkökulmasta. Tutkielmani rajautuu siis kahden tutkimusalueen, tekoälyavusteisen ohjelmoinnin ja pariohjelmoinnin risteyskohtaan. Perinteisen pariohjelmointitekniikan hyötyjä on aiemmassa tutkimuksessa havaittu mm. tuotetun ohjelmiston laadussa, saaduissa oppimistuloksissa ja työskentelyn tehokkuudessa (Alves de Lima Salge ja Berente 2016). Vuonna 2021 julkaistua GitHub Copilotia on markkinoitu ilmauksella ”your AI pair programmer” (Sarkar ym. 2022, s. 15), mikä luo otollisen asetelman sen tutkimiselle, miten Copilotin käyttö todellisuudessa vertautuu perinteiseen pariohjelmointiin. Voidaanko pariohjelmoinnilla saavutettuja hyötyjä havaita ihmisen ja tekoälytyökalun yhteistyössä? Tutkielmassani kartoitan tästä aiheesta tehdyn tutkimuksen nykytilaa.

Kirjallisuuskatsauksen pohjalta pyrin vastaamaan kahteen tutkimuskysymykseen. Ensinnäkin, miten GitHub Copilotin avustama ohjelmointi vertautuu perinteiseen pariohjelmointiprosessiin? Toiseksi, millaisia mahdollisuuksia ja toisaalta haasteita Copilotin käyttö tuo ohjelmointityöhön pariohjelmoinnin näkökulmasta?

Tutkielman seuraavassa luvussa tulen esittelemään sekä perinteisen pariohjelmointitekniikan että GitHub Copilotin toimintaa. Kolmannessa luvussa käsittelen Copilotin käyttöä pariohjelmoinnissa ja luon katsauksen aihepiirin tutkimukseen. Lopuksi vastaan kirjallisuuskatsauksen pohjalta esittämiini tutkimuskysymyksiin ja pohdin tutkielmani rajoitteita sekä ehdotuksia aiheen jatkotutkimukseen.

2 Teoreettinen tausta

Tässä luvussa esittelen tutkielman kannalta olennaiset käsitteet ja niistä tehtyä tutkimusta. Pariohjelmointia käsittelen sen toimintaperiaatteen kuvauksen, sen hyötyjen arvioinnin ja käytännön toteutuksen tarkastelun kautta. GitHub Copilotin toimintaa havainnollistan yksinkertaisten esimerkkien avulla, minkä jälkeen arvioin Copilotin tuottamien koodiehdotusten laatua aiempaan tutkimukseen nojaten.

2.1 Pariohjelmointi

Pariohjelmointi on ketterän ohjelmistokehityksen metodi, jossa kaksi ihmisohjelmoijaa työskentelee samanaikaisesti, toinen ”kuljettajan” (engl. *driver*) ja toinen ”navigaattorin” (engl. *navigator*) roolissa. Kuljettajan roolissa oleva tuottaa koodia, kun taas navigaattorin tehtävänä on pohtia laajempaa kokonaisuutta, tarkkaillen tuotetun koodin vastaavuutta haluttuun lopputulokseen, huomioiden syntaktiset virheet ja tarjoten uusia näkökulmia ratkaistavaan ongelmaan. Jonkin ajan kuluttua roolit vaihtuvat. (Chen ja Rea 2018, s. 53.)

2.1.1 Pariohjelmoinnin hyödyt

Pariohjelmoinnin juuret ulottuvat 1970-luvulle, mutta sen tunnettuus kasvoi merkittävästi ketterän ohjelmistokehityksen yleistyessä vuosituhannen vaihteessa (Alves de Lima Salge ja Berente 2016, s. 1). Vuonna 2000 ohjelmistokehittäjä Kent Beck sisällytti pariohjelmoinnin eXtreme Programming -metodologiansa yhdeksi pääperiaatteeksi. Beck (2000) esitti pariohjelmoinnin hyötyjen näkyvän ohjelmiston laadussa, ohjelmoijien oppimistuloksissa ja ohjelmointityön nopeudessa. Tarkastelen seuraavaksi näitä Beckin kolmea väitettä yksi kerrallaan, ja kartoitan, millaisia tutkimustuloksia niihin liittyen on vuosien mittaan saatu.

Beckin väite ohjelmiston **laadun** kohoamisesta pariohjelmoinnin myötä pohjautuu sille perusolettamukselle, että kahden ihmisen yhteistyöllä voidaan ratkaista monimutkaisempia ongelmia ja saavuttaa parempia tuloksia kuin yksittäisen ihmisen työpanoksella. Lisäksi Beckin mukaan pariohjelmointi altistaa ohjelmointityön jatkuvalle tarkastelulle ja arvioinnille, jolloin työskentely on huolellisempaa. eXtreme Programming -menetelmän yksi kulmakivi

on jatkuva ohjelmiston testaaminen, ja Beck ehdottaakin, että ohjelmoijat keskittyvät testien kirjoittamiseen ennen halutun toiminnallisuuden kehittämistä. Tällöin molemmille ohjelmoijille muodostuu yhtenäinen ymmärrys siitä, mihin lopputulokseen halutaan päästä. (Beck 2000, s. 57.) Tutkimuksissa pariohjelmoinnin on havaittu tuottavan luotettavampaa ja virheettömämpää koodia verrattuna yksittäisen ohjelmoijan työhön (Lemos ym. 2012; Sison 2008). Arisholmin ym. mukaan kahden junioritasoisen ohjelmoijan pariohjelmointityö voi yltää samaan virheettömyyden tasoon kuin yhden senioritasoisen ohjelmoijan ohjelmointityö (Arisholm ym. 2007, s. 17).

Toinen Beckin argumenteista pariohjelmoinnin puolesta oli ohjelmointityöstä saadut paremmat **oppimistulokset**. Yhteistyössä ohjelmoijat voivat oppia toisiltaan, ja oppimista vauhdittaa se, että pariohjelmointi ohjaa ohjelmoijia jatkuvaan kommunikaatioon toistensa kanssa. Erityisesti osana Beckin eXtreme Programming -metodologiaa pariohjelmoinnissa käytettävät parit voivat koostua eri taito- tai kokemustason omaavista ohjelmoijista, jolloin kokemattomammille tarjoutuu mahdollisuus oppia kokeneemmilta ohjelmoijilta. Parit vaihtuvat säännöllisesti, jolloin kukin ohjelmoija pääsee oppimaan ja jakamaan oppimaansa monen muun kanssa. (Beck 2000, s. 78–79.) Sisonin tutkimuksessa pariohjelmointia tehneet koehenkilöt raportoivat oppineensa paljon, ja oppimista tapahtui kuljettajan ja navigaattorin välillä molemminsuuntaisesti (Sison 2008, s. 5).

Väitteellään pariohjelmoinnin **nopeudesta** Beck halusi haastaa sitä kritiikkiä, jota hän epäili pariohjelmointiin kohdistuvan tehokkuusnäkökulmasta. Beck esitti, että yhteistyön ja suunnittelukäytäntöjen myötä pariohjelmointi on nopeampaa kuin se, että työ jaettaisiin kahden ohjelmoijan kesken ja lopuksi työn tulokset yhdistettäisiin (Beck 2000, s. 79). Arisholm ym. (2007, s. 11) havaitsivat, että pariohjelmoijat suoriutuivat annetusta tehtävästä keskimäärin 8 % nopeammin kuin yksittäinen ohjelmoija. Toisaalta Lemos ym. (2012, s. 8) totesivat tutkimuksessaan, että kahden ohjelmoijan yhteenlaskettuna työaikana pariohjelmoinnissa kului samaan tehtävään keskimäärin 48 % enemmän aikaa kuin yksittäisellä ohjelmoijalla.

Tutkimukset siis laajalti tukevat Kent Beckin väitteitä pariohjelmoinnin hyödyistä etenkin mitä tulee ohjelmointityön tulosten laatuun, mutta tästä linjasta poikkeaviakin tutkimustuloksia on saatu. Alves de Lima Salge ja Berente (2016, s. 6–7) huomauttavat, että ristiriitaisia tuloksia voidaan osittain selittää moderaattorimuuttujilla: pariohjelmoinnilla vaikuttaisi

olevan laatua ja oppimistuloksia kohottava vaikutus monimutkaisissa ohjelmointitehtävissä muttei välttämättä yksinkertaisissa ohjelmointitehtävissä. Myös erilaiset ohjelmiston laadun ja ohjelmointityön tehokkuuden mittausmenetelmät ja määritelmät voivat aiheuttaa hajontaa tutkimustuloksissa.

2.1.2 Pariohjelmointi käytännössä

Kuinka tarkasti Beckin määrittelemä pariohjelmoinnin metodologia sitten välittyy käytännön ohjelmointityöhön? Chong ja Hurlbutt (2007, s. 8) kyseenalaistavat etnografisessa tutkimuksessaan pariohjelmoinnin yleisesti hyväksytyn kuljettaja-navigaattori-roolijaon hyödyllisyyden ja yleisyyden käytännön pariohjelmointityössä. Siinä missä pariohjelmoinnin pääperiaatteena on ajateltu olevan kahden ohjelmoijan samanaikainen ajattelu ja työskentely kahdella eri abstraktion tasolla, Chong ja Hurlbutt eivät havainneet tutkimuksessaan tällaista selkeää jakoa, vaan ohjelmoijat liikkuivat yhdessä kommunikoiden kuljettajan ja navigaattorin roolien välillä. Freudenberg, Romero ja du Boulay (2007, s. 7–8) vahvistavat tämän havainnon ja esittävät, että kahden ohjelmoijan verbaalisella kommunikaatiolla on merkittävä rooli pariohjelmoinnin hyötyjen saavuttamisessa.

Pariohjelmointia tekevien ohjelmoijien kokemustaso näyttäisi vaikuttavan pariohjelmoinnista saataviin laatu-, oppimis- ja tehokkuushyötyihin (Alves de Lima Salge ja Berente 2016, s. 6–7). Joidenkin tutkimustulosten mukaan taas jo junioritasoiset ohjelmoijat voivat pariohjelmoinnin aikana saavuttaa samankaltaisia tuloksia kuin kahdesta kokeneemmasta ohjelmoijasta koostuva pari. Tätä voi selittää jo mainittu ajatusten sanallistaminen, joka auttaa ohjelmoijia ymmärtämään koodin toimintaa (Hannay ym. 2009, s. 12). Mahdolliset erot pariohjelmointia tekevän parin kokemustasossa voivat ohjata vuorovaikutuksen painopisteitä. Chong ja Hurlbutt (2007, s. 3) havaitsivat, että kun parin molempien osapuolien taitotaso oli samankaltainen, kommunikaatio oli tasapuolista, kun taas eri taitotason omaavissa pareissa kokeneempi ohjelmoija dominoi vuorovaikutusta.

2.2 GitHub Copilot

GitHub Copilot on GitHubin ja OpenAIn kehittämä, Codex-kielimalliin perustuva kaupallinen tekoälytyökalu, jonka tarkoituksena on auttaa ohjelmoijaa ohjelmointityössä generoimalla koodia luonnollisella kielellä annetun kuvauksen pohjalta ja ehdottamalla täydennyksiä jo kirjoitettuun koodiin (*About GitHub Copilot* 2023). Copilotista ja tekoälyavusteisesta ohjelmoinnista laajemminkin on tehty tutkimusta. Nguyen ja Nadi (2022) tarkastelevat empiirisessä tutkimuksessaan Copilotin koodiehdotusten oikeellisuutta ja Copilotin tuottaman koodin ymmärrettävyyttä. Barke, James ja Polikarpova (2022) ovat selvittäneet, millä tavoin ohjelmoijat toimivat vuorovaikutuksessa Copilotin kanssa. Muitakin tekoälyavusteisen ohjelmoinnin työkaluja on kehitetty: Sarkar ym. (2022, s. 6) nostavat tutkimuksessaan Copilotin rinnalle Tabninen sekä Visual Studio IntelliCoden vaihtoehtoisina tekoälyä hyödyntävinä ohjelmointityökaluina.

2.2.1 Copilotin käyttö

Copilotia käytetään koodieditoriin, kuten Visual Studio Codeen, JetBrainsiin tai Neovimiin asennettavan lisäosan kautta. Tässä luvussa esittelen Copilotin toimintaa Visual Studio Coden lisäosana, mutta toimintaperiaate on samankaltainen muissakin tuetuissa editoreissa. Copilotin käyttö muistuttaa yleisellä tasolla monen modernin ohjelmointiympäristön tarjoamaa automaattista koodin täydennystä. Copilot voi kuitenkin generoida kerralla paljon laajempia kokonaisuuksia, ja sen koodiehdotukset ovat tarkemmin annettuun kontekstiin sovitettuja (Sarkar ym. 2022, s. 1–2).

Ohjelmoija kommunikoi Copilotin kanssa kehotteiden (engl. *prompts*) avulla. Kehotteet ovat tekstiä, jota ohjelmoija kirjoittaa suoraan koodieditorissa tarkasteltavan tiedoston siihen kohtaan, johon koodiehdotus halutaan generoida. Kehote voi olla kirjoitettu joko avatussa tiedostossa käytettävällä ohjelmointikielellä tai luonnollisella kielellä koodikommenttina. Copilot havaitsee kontekstin analysoimalla avattua tiedostoa sekä siihen liittyviä muita tiedostoja ja generoi koodiehdotuksen kursorin kohdalle automaattisen täydennyksen tapaan. Ohjelmoija voi hyväksyä koodiehdotuksen `Tab`-näppäintä painamalla. Vaihtoehtoisia koodiehdotuksia ohjelmoija voi selata suoraan kooditiedostossa tai avaamalla listan ehdotuksista erilliseen

välilehteen. (*Getting Started with GitHub Copilot in Visual Studio Code* 2023)

Esimerkkinä Copilotin toiminnasta kokeillaan yksinkertaisen Python-kielisen funktion generointia, ensin koodikehotteen ja sitten luonnollisen kielen kehotteen pohjalta. Tavoitteena on luoda funktio, joka palauttaa syötteenä annetun listan pienimmän arvon. Kuviossa 1 aloitetaan funktion määrittely antamalla funktiolle sen haluttua toimintaa kuvaava nimi `min` ja funktion parametriksi `list`-niminen muuttuja. Tämän koodikehotteen pohjalta Copilot generoi koodiehdotuksen, joka täydentää funktion määrittelyn.

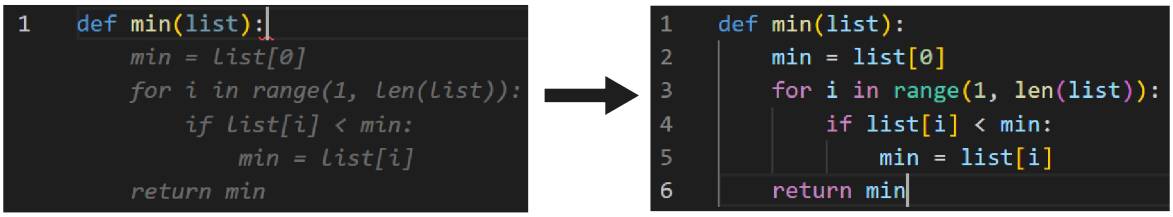
Kokeillaan seuraavaksi päästä samaan lopputulokseen antamalla Copilotille kehotteeksi koodikommentti: ”Function that returns the minimum value of a given list”. Kommentissa kuvaillaan luonnollisella kielellä se funktio, joka halutaan toteuttaa. Kuviossa 2 huomataan, että Copilot ehdottaa ensin funktion esittelyriviä. Hyväksyminen tapahtuu `Tab`-näppäintä painamalla. Seuraava ehdotus täydentää funktion sisäisen toiminnan. Tämäkin vaatii ohjelmoijalta hyväksynnän. Huomionarvoista on, että aloittamalla generointi luonnollisen kielen kommentilla saatiin lopputuloksena funktion lisäksi myös koodikommentti, joka kuvaa funktion toimintaa.

2.2.2 Copilotin koodiehdotusten laatu

Copilot perustuu OpenAI:n Codex-kielimalliin (Chen ym. 2021), jonka kouluttamiseen on käytetty julkisissa GitHub-repositorioissa olevaa koodia. Siinä missä Copilotin käyttämä Codex-malli kykenee tuottamaan koodia eri ohjelmointikielillä, Codexin varhaiset versiot kehitettiin tuottamaan vain Python-koodia. Tällainen 12 miljardin parametrin laajuinen kielimalli onnistui Chenin ym. tutkimuksessa ratkaisemaan ensimmäisellä tuottamallaan koodiehdotuksella 28,8 % annetuista ohjelmointitehtävistä. Kun yhtä tehtävää kohden generoitiin sata koodiehdotusta, niiden joukosta löytyi oikea ratkaisu 72,3 % todennäköisyydellä. (Chen ym. 2021, s. 7.)

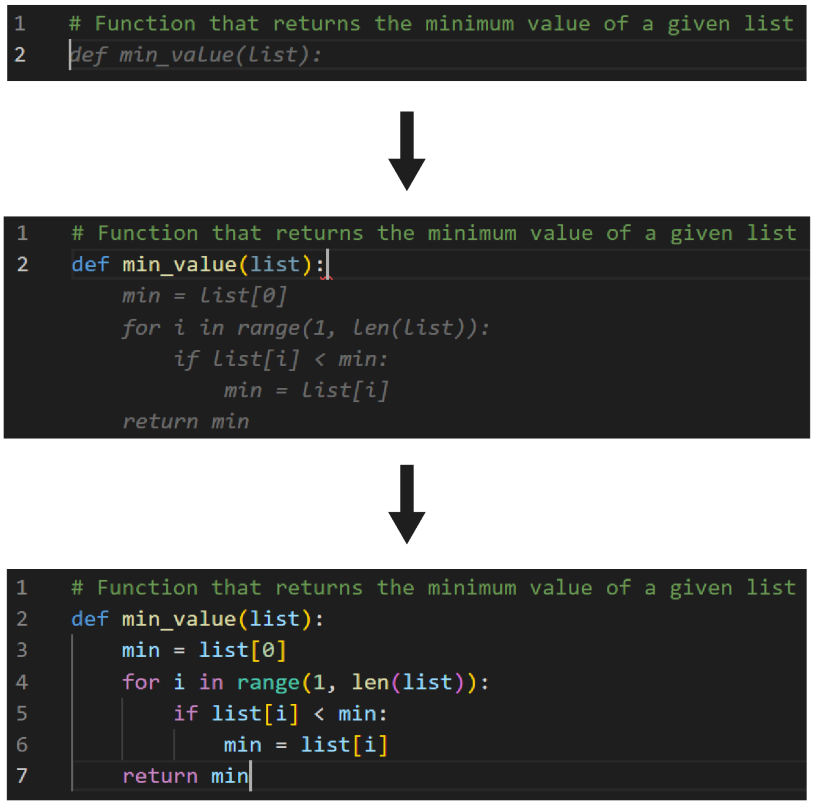
On hankalaa arvioida, missä määrin näiden tulosten pohjalta voidaan vetää johtopäätöksiä Copilotin kyvykkyydestä vastaavanlaisissa tehtävissä. Copilot generoi koodiehdotuksia lukuisilla eri ohjelmointikielillä, mutta koodiehdotusten laatuun vaikuttaa käytetyn ohjelmointikielen esiintyvyys Copilotin koulutusdatassa (*About GitHub Copilot* 2023).

Nguyen ja Nadi (2022) tarkastelevat tutkimuksessaan Copilotin koodiehdotuksia Javalla, JavaScriptilla ja Pythonilla, jotka olivat GitHubin vuosiraportin mukaan vuoden 2022 kolme suosituinta ohjelmointikieltä avoimen lähdekoodin GitHub-repositorioissa, sekä C:llä, joka sijoittuu samassa vertailussa yhdeksänneksi (*The State of The Octoverse 2022: The Top Programming Languages* 2022). Copilotin ensimmäinen koodiehdotus tuottaa hyväksytyn ratkaisun 57 % annetuista tehtävistä, kun ohjelmointikielenä on Java. Pienin onnistumisprosentti on JavaScript-koodiehdotuksilla (27 %), ja Pythonin ja C:n vastaavat tulokset jäävät nekin alle 50 prosenttiin. Toisaalta tutkimuksen tuloksista ilmenee, että Copilotin koodiehdotus toimii vähintäänkin hyödyllisenä lähtöpisteenä toimivalle ratkaisulle ohjelmointikielestä riippuen 61–91 % tapauksista. (Nguyen ja Nadi 2022, s. 3–4.)



Kuvio 1: Copilotin käyttö koodikehotteella.

Kuvankaappaukset Visual Studio Codesta v1.74.2



Kuvio 2: Copilotin käyttö luonnollisen kielen kehotteella.

Kuvankaappaukset Visual Studio Codesta v1.74.2

3 GitHub Copilot pariohjelmoinnissa

Kuten johdannossa todettiin, GitHub markkinoi Copilot-työkalua eräänlaisena ”tekoälypariohjelmoijana”. Miten Copilotin soveltaminen pariohjelmointitekniikkaan sitten käytännössä voisi tapahtua? Aiheen tutkimuksissa yhteistä on se, että niissä tarkastellaan sellaista Copilot-avusteista ohjelmointia, jossa toinen pariohjelmoinnin ihmisohjelmoijista korvataan Copilotilla. Kuljettaja-navigaattori-roolijakoon pohjaten voidaan ajatella, että ihmisohjelmoija toimii navigaattorina ja Copilot kuljettajana.

Copilotin ja pariohjelmoinnin yhteyksiä on jo ehditty tutkia. Sarkarin ym. artikkelissa ”What is it like to program with artificial intelligence?” vertaillaan suurten kielimallien avustamaa ohjelmointia aiemmin kehitettyihin ohjelmoijaa avustaviin tekniikoihin, kuten pariohjelmointiin (Sarkar ym. 2022). Dakhel ym. taas vertaavat artikkelissaan ”GitHub Copilot AI pair programmer: Asset or Liability?” Copilotin tuottamaa koodia ihmisohjelmoijien tuottamaan koodiin erilaisissa ohjelmointitehtävissä (Dakhel ym. 2022).

3.1 Roolijako

Perinteisessä pariohjelmoinnissa roolit vaihtuvat tietyin väliajoin. Copilot-avusteisessa pariohjelmoinnissa ihminen siirtyy toistuvasti navigaattorin paikalta kuljettajaksi antaakseen Copilotille kehotteita, joiden pohjalta koodia voidaan generoida. Copilotin siirtymistä navigaattorin rooliin taas on vaikea hahmottaa, ja voidaan kysyä, tapahtuuko sitä ollenkaan. Copilotin ominaisuudet mahdollistavat koodin generoimisen, mutta sillä ei ole esimerkiksi ominaisuutta, jolla se huomauttaisi kuljettajaa ohjelmointivirheistä.

Sarkar ym. (2022, s. 15–16) pitävät pariohjelmoinnin roolijaon soveltamista tekoälyavusteiseen ohjelmointiin hankalana ja pinnallisena. Koodigeneroinnin hyödyntäminen ei vapauta ohjelmoijaa navigaattorin roolille keskeiseen kokonaisuuden tarkkailuun, vaan ohjelmoija joutuu käymään tekoälyn tuottamaa koodia läpi lausekkeiden ja aliohjelmien logiikan tasolla. Sarkarin ym. mukaan osa pariohjelmoinnin hyödyistä selittyy roolijaon sijaan sillä, että ohjelmoijat joutuvat jatkuvasti sanallistamaan ajatteluaan toisilleen, ja tätä hyötyä ei saavuteta, kun pariohjelmoijana on toisen ihmisen sijaan tekoälytyökalu.

Olennaisena erona kahden ihmisen tekemään pariohjelmointiin voidaan huomioida se, että aktiivisia, aloitteen tekemiseen kykeneviä toimijoita on kahden sijaan yksi. Vuorovaikutus Copilotin kanssa alkaa aina ohjelmoijan aloitteesta, ja Copilot reagoi siihen tarjoamalla koodiehdotuksia. Barke, James ja Polikarpova (2022, s. 2) havaitsivat ohjelmoijan ja Copilotin välisessä vuorovaikutuksessa vaihtelua kahden erilaisen moodin välillä: kiihdytyksen (engl. *acceleration*) ja tutkimuksen (engl. *exploration*). Kiihdytysvaiheessa ohjelmoija tietää mitä aikoo tehdä seuraavaksi, ja Copilot auttaa ohjelmoijaa saavuttamaan tavoitteensa nopeammin. Tutkimusvaiheessa ohjelmoija taas harkitsee seuraavaa tehtäväänsä ja käyttää Copilotia kartoittaakseen vaihtoehtojaan tai päästäkseen tehtävässä alkuun. Barken ym. mukaan ohjelmoijan ja Copilotin vuorovaikutus kiihdytysvaiheessa on nopeaa ja päättäväistä, kun taas tutkimusvaiheessa ohjelmoija hitaasti ja harkitusti luo Copilotille kehotteita ja käy läpi niiden pohjalta generoitua koodia.

3.2 Copilot-koodiehdotusten ymmärrettävyys ja luotettavuus

Luvussa 2.2.2 tarkasteltiin Copilotin tuottamien koodiehdotusten laatua yksinkertaisesti sen perusteella, miten suurella todennäköisyydellä ne ratkaisevat annetun ohjelmointitehtävän. Pariohjelmoinnin näkökulmasta yhtä tärkeää Copilotin hyödyllisyyden kannalta on se, miten ymmärrettävää tuotettu koodi on. Jos Copilotin koodiehdotus sisältää ohjelmointivirheen, on se helppo huomata ja korjata, jos koodin logiikan ja etenemisen seuraaminen on ohjelmoijalle vaivatonta. Toisaalta koodin ymmärrettävyys auttaa ohjelmoijaa havaitsemaan sellaiset tilanteet, joissa Copilotin koodiehdotus tuottaa näennäisesti pätevän ratkaisun mutta sisältää ei-tarkoituksenmukaisia sivuvaikutuksia tai avaa ohjelmistoon tietoturvahaavoittuvuuden.

Dakhel ym. (2022) vertasivat tutkimuksessaan Copilotin koodiehdotuksia ihmisohjelmoijan tuottamaan koodiin ja selvittivät, miten helppoa Copilotin tuottama virheellinen koodiehdotus on korjata verrattuna ihmisen luomaan virheelliseen koodiin. Koehenkilöiden kehittämät ratkaisut annettuihin ohjelmointiongelmiin olivat useammin oikeita ja monimuotoisempia kuin Copilotin generoimat ratkaisut. Toisaalta Copilotin tuottama virheellinen koodi oli helpommin korjattavissa, ja oikeat ratkaisut olivat yksinkertaisempia verrattuna koehenkilöiden tuottamaan koodiin. Copilot ei aina huomioinut koodille asetettuja vaatimuksia, kun taas koehenkilöt ottivat ne huomioon ratkaisuissaan.

Dakhelin ym. tutkimuksessa havaittiin, että kehotteen muotoilu voi vaikuttaa huomattavasti Copilotin tuottaman koodin laatuun: Copilot ei aina ymmärrä ilmaisuja, jotka ihmiselle ovat ilmiselviä. Esimerkiksi pyydettäessä järjestämään Person-olioita vanhimmasta nuorimpaan, Copilot ei ymmärtänyt ilmaisua ”older people are at the front”. Kun taas ilmaisu muutettiin muotoon ”descending order”, Copilot tuotti oikean vastauksen huomattavasti useammin. (Dakhel ym. 2022, s. 17.) Tutkimustulosten perusteella Dakhel ym. esittävät, että Copilotin kykyä ymmärtää luonnollista kieltä tulee kehittää pidemmälle, jotta Copilot voisi toimia pariohjelmoinnin yhtenä osapuolena (Dakhel ym. 2022, s. 18).

Sarkar ym. (2022, s. 7) nostavat kirjallisuuskatsauksessaan esiin tekoälyavusteisen koodigeneroinnin vaikutukset ohjelmiston turvallisuuteen ja luotettavuuteen. Monissa koodia generoivia työkaluja tarkastelevissa tutkimuksissa käytetään tuotetun koodin laadun arviointiin yksikkötesteihin perustuvaa metriikkaa, joka jättää huomiotta koodin ymmärrettävyyteen ja luotettavuuteen liittyvät potentiaaliset ongelmakohdat. Chen ym. (2021, s. 11) pitävät merkittävänä riskinä liiallista luottamusta generoituun koodiin. Koodi voi näyttää validilta mutta aiheuttaa yllättäviä sivuvaikutuksia ja turvallisuusriskejä, joten tekoälytyökalua hyödyntävän ohjelmoijan on oltava tietoinen riskeistä ja käytävä läpi koodin toimintaa. Pariohjelmointinäkökulmasta tällainen aktiivinen tarkkailu ja koodin toiminnan läpikäynti muistuttaa navigaattorin roolia.

4 Yhteenveto

Tässä tutkielmassa olen tarkastellut kirjallisuuskatsauksen muodossa GitHub Copilotin käyttöä pariohjelmoinnissa. Pohdin seuraavaksi, millaisia vastauksia aiheen tutkimus tarjoaa johdannossa esittämiini kahteen tutkimuskysymykseen. Lopuksi käsittelen vielä tutkielman rajoitteita ja ehdotuksia jatkotutkimukseen.

Miten Copilotin avustama ohjelmointi vertautuu perinteiseen pariohjelmointiprosessiin? Perinteisen pariohjelmoinnin kuljettaja-navigaattori-roolijako ja vaihto roolista toiseen ei toteudu Copilot-avusteisessa pariohjelmoinnissa, vaan ihmisohjelmoija liikkuu jatkuvasti abstraktion tasojen välillä. Kuljettajan roolissa ohjelmoija antaa Copilotille kehotteita ja korjaa mahdollisia generoidussa koodissa olevia ohjelmointivirheitä. Navigaattorin roolissa ohjelmoija taas käy läpi tuotetun koodin toimintaa ja sen suhdetta kokonaisuuteen sekä ohjelmistolle asetettuihin vaatimuksiin. Toisaalta on havaittu, että myös kahden ihmisen suorittamassa pariohjelmoinnissa voi käytännössä tapahtua jatkuvaa roolien välillä liikkumista. Tekoälytyökaluna Copilot on lukittu kuljettajan rooliin, ainoastaan reagoiden ihmisohjelmoijan antamiin kehotteisiin.

Millaisia mahdollisuuksia ja haasteita Copilot-avusteinen ohjelmointi tuo pariohjelmointiin? Aihepiirin tutkimuksessa on näyttöä siitä, että osa pariohjelmoinnin hyödyistä ei synny roolijaosta vaan siitä, että pariohjelmointi tekniikkana pakottaa ohjelmoijat sanallistamaan ajatuksiaan. Jatkuva kommunikaatio taas parantaa ohjelmoijien käsitystä koodin toiminnasta ja johtaa siten parempaan lopputulokseen sekä uuden oppimiseen. Kun pariohjelmointi tapahtuu ihmisen ja tekoälytyökalun välillä, tällaista vuorovaikutusta ei tapahdu, ja osa pariohjelmoinnin hyödyistä jää saavuttamatta. Lisäksi tekoälyavusteisen ohjelmoinnin haasteena pidetään liiallista luottamusta tekoälytyökalun generoimaan koodiin, mikä voi aiheuttaa riskejä ohjelmistoturvallisuuden näkökulmasta.

Kirjallisuuskatsaus onnistui siis vastaamaan asettamiini tutkimuskysymyksiin, mutta vastausten rinnalla on syytä huomioida tutkielmani rajoitteet. Siinä missä pariohjelmointia on tutkittu jo yli kahden vuosikymmenen ajan, GitHub Copilot on uusi työkalu, joten sen tutkimuskin on vielä alkutekijöissään. Tutkielmani voidaan nähdä kartoituksena Copilotin ja pa-

riohjelmoinnin tutkimuksen nykytilasta, mutta tarvetta uusille kirjallisuuskatsauksille epäilemättä ilmenee sitä mukaa, kun Copilot ja tekoälyavusteinen ohjelmointi laajemminkin kehittyvät. Copilotia kehitetään jatkuvasti, joten eri tutkimusten tulokset eivät välttämättä ole suoraan verrattavissa toisiinsa. Tämä tekee myös kirjallisuuskatsauksen tekemisestä haastavaa, koska vaikka Copilotin tutkimusta on kirjoitushetkellä vasta kahden vajaan kahden vuoden ajalta, voivat empiiriset tutkimustulokset olla riippuvaisia siitä, millä Copilotin versiolla tutkimusta on tehty.

Sinä aikana, kun olen tehnyt tätä tutkielmaa, OpenAI on julkaissut GPT-4-kielimallin ja esitellyt yhteistyössä Microsoftin kanssa kehitettävän GitHub Copilot X -työkalun. Copilot X tuo tekoälyn generoimien koodiehdotusten rinnalle ChatGPT:n kaltaisen chat-sovelluksen, joka on integroitu ohjelmointiympäristöön ja jonka tarkoitus on avustaa ohjelmoijaa ohjelmistokehityssyklin kaikissa vaiheissa (Dohmke 2023). Sitä mukaa kun ohjelmointityössä avustavat tekoälytyökalut kehittyvät, olisi syytä tutkia, miten niiden käyttömahdollisuudet pariohjelmoinnissa mahdollisesti laajenevat. Copilot X:n esittelyssä mainitun chatbotin voisi ajatella syventävän ihmisohjelmoijan ja tekoälytyökalun välistä vuorovaikutusta. Kuten luvussa 2.1 huomattiin, tällainen kommunikaatio on perinteisen pariohjelmoinnin keskiössä.

Copilot ajautuu joskus ongelmiin tietotyyppien kanssa ja esimerkiksi päättelee tietotyypin virheellisesti muuttujan nimen perusteella (Dakhel ym. 2022, s. 12). Jos kehotteena käytetään koodia, joka on kirjoitettu kielellä, jossa tyypit ilmaistaan eksplisiittisesti muuttujaa alustettaessa tai aliohjelmaa määriteltäessä, tekeekö Copilot silloin vähemmän tyypitykseen liittyviä virheitä? Toisaalta, voiko esimerkiksi tyyppiannotaatioiden käyttö Pythonissa auttaa Copilotia samalla tavalla? Tässäkin olisi aihetta jatkotutkimukselle.

Lähteet

Alves de Lima Salge, Carolina ja Nicholas Berente. 2016. “Pair Programming vs. Solo Programming: What Do We Know After 15 Years of Research?” Teoksessa *2016 49th Hawaii International Conference on System Sciences (HICSS),* 5398–5406. Tammikuu. https://doi.org/10.1109/HICSS.2016.667.

Arisholm, Erik, Hans Gallis, Tore Dyba ja Dag I.K. Sjoberg. 2007. “Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise”. *IEEE Transactions on Software Engineering* 33, numero 2 (helmikuu): 65–86. ISSN: 1939-3520. https://doi.org/10.1109/TSE.2007.17.

Barke, Shraddha, Michael B. James ja Nadia Polikarpova. 2022. *Grounded Copilot: How Programmers Interact with Code-Generating Models,* arXiv:2206.15000, lokakuu. Viitattu 9. helmikuuta 2023. https://doi.org/10.48550/arXiv.2206.15000. arXiv: arXiv:2206.15000.

Beck, Kent. 2000. *Extreme Programming eXplained: Embrace Change.* Reading, MA: Addison-Wesley. ISBN: 978-0-201-61641-5.

Chen, Kuanchin ja Alan Rea. 2018. “Do Pair Programming Approaches Transcend Coding? Measuring Agile Attitudes in Diverse Information Systems Courses”. *Journal of Information Systems Education* 29, numero 2 (tammikuu): 53–64. ISSN: 2574-3872.

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards ym. 2021. *Evaluating Large Language Models Trained on Code,* arXiv:2107.03374, heinäkuu. Viitattu 2. maaliskuuta 2023. arXiv: arXiv:2107.03374.

Chong, Jan ja Tom Hurlbutt. 2007. “The Social Dynamics of Pair Programming”. Teoksessa *29th International Conference on Software Engineering (ICSE’07),* 354–363. Toukokuu. https://doi.org/10.1109/ICSE.2007.87.

Dakhel, Arghavan Moradi, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming ja Jiang. 2022. *GitHub Copilot AI Pair Programmer: Asset or Liability?,* arXiv:2206.15331, kesäkuu. Viitattu 26. tammikuuta 2023. https://doi.org/10.48550/arXiv.2206.15331. arXiv: arXiv:2206.15331.

Dohmke, Thomas. 2023. *GitHub Copilot X: The AI-powered Developer Experience.* https://github.blog/2023-03-22-github-copilot-x-the-ai-powered-developer-experience/, maaliskuu. Viitattu 17. huhtikuuta 2023.

Freudenberg, S., Pablo Romero ja Benedict du Boulay. 2007. "Talking the Talk: Is Intermediate-Level Conversation the Key to the Pair Programming Success Story?" Teoksessa *Proceedings - AGILE 2007,* 84–91. Syyskuu. ISBN: 978-0-7695-2872-4. https://doi.org/10.1109/AGILE.2007.1.

About GitHub Copilot. 2023. https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals#about-github-copilot. Documentation. Viitattu 9. helmikuuta 2023.

Getting Started with GitHub Copilot in Visual Studio Code. 2023. https://docs.github.com/en/copilot/getting-started-with-github-copilot/getting-started-with-github-copilot-in-visual-studio-code. Viitattu 17. maaliskuuta 2023.

Hannay, Jo, Tore Dybå, Erik Arisholm ja Dag Sjøberg. 2009. "The Effectiveness of Pair Programming: A Meta-Analysis". *Information and Software Technology* 51 (heinäkuu): 1110–1122. https://doi.org/10.1016/j.infsof.2009.02.001.

Lemos, Otávio Auguste Lazzarini, Fabiano Cutigi Ferrari, Fábio Fagundes Silveira ja Alessandro Garcia. 2012. "Development of Auxiliary Functions: Should You Be Agile? An Empirical Assessment of Pair Programming and Test-First Programming". Teoksessa *2012 34th International Conference on Software Engineering (ICSE),* 529–539. Kesäkuu. https://doi.org/10.1109/ICSE.2012.6227163.

Nguyen, Nhan ja Sarah Nadi. 2022. "An Empirical Evaluation of GitHub Copilot's Code Suggestions". Teoksessa *Proceedings of the 19th International Conference on Mining Software Repositories,* 1–5. Pittsburgh Pennsylvania: ACM, toukokuu. ISBN: 978-1-4503-9303-4, viitattu 26. tammikuuta 2023. https://doi.org/10.1145/3524842.3528470.

Sarkar, Advait, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan ja Ben Zorn. 2022. *What Is It like to Program with Artificial Intelligence?,* arXiv:2208.06213, lokakuu. Viitattu 9. helmikuuta 2023. arXiv: arXiv:2208.06213.

Sison, Raymund. 2008. "Investigating Pair Programming in a Software Engineering Course in an Asian Setting". Teoksessa *2008 15th Asia-Pacific Software Engineering Conference,* 325–331. Joulukuu. https://doi.org/10.1109/APSEC.2008.61.

The State of The Octoverse 2022: The Top Programming Languages. 2022. https://octoverse.github.com/2022/top-programming-languages. Viitattu 17. maaliskuuta 2023.