Lassi Mustonen

Ohjelmoijan tehokkuuteen vaikuttavat tekijät

Tietotekniikan kandidaatintutkielma

3. kesäkuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Lassi Mustonen

Yhteystiedot: lassi@lassimustonen.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Ohjelmoijan tehokkuuteen vaikuttavat tekijät

Title in English: Factors that affect programmers productivity

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 20+0

Tiivistelmä: Tässä tutkielmassa kartoitetaan aikaisemmasta tutkimuksesta löytynyttä tehokkuuteen liittyvää kirjallisuutta eri näkökulmista. Määritellään keskeisimmät termit ja tarkastellaan, miten ohjelmoijan tehokkuutta voidaan mitata ja mitä haasteita mittaamisessa on. Tarkasteltavia näkökulmia ovat ohjelmoijan tyytyväisyys, kokemus, ajan käyttö, työympäristö, persoonallisuus ja ohjelmointityyli ja pariohjelmointi.

Ohjelmoijan tehokkuuteen vaikuttavia tekijöitä arvioidaan eri näkökulmista ja selvitetään, miten voidaan parantaa ohjelmoijan tehokkuutta vaikuttamalla ohjelmoijan työtapoihin ja työympäristöön. Tehokkuuden parantaminen vaikuttaa positiivisesti yksittäiseen ohjelmoijaan. Ohjelmoijan tehokkuuden paraneminen heijastuu myös työnantajan tehokkuuden parantumiseksi.

Avainsanat: Ohjelmoijan tehokkuus, tietotyöntekijä, tehokkuus, efektiivisyys

Abstract: This thesis charts the literature found in the previous studies about efficiency from different perspectives. Defining key terms and inspection of how efficiency can be measured and what the challenges there are in measuring productivity. The aspects considered are programmer satisfaction, experience, use of time, work environment, personality and programming style, and pair programming.

Factors influencing programmer efficiency are evaluated from different perspectives and

explored how to improve programmer efficiency by influencing the programmer's working methods and work environment. The Reflection section aims to clarify the efficiency factors found and how they can be modified to achieve maximum efficiency.

Keywords: Programmer efficiency, programmer productivity, knowledge-worker, productivity, effectivity

Sisältö

1 Johdanto

Ohjelmoijien tehokkuus on tällä hetkellä merkittävä aihe, sillä yksittäisen ohjelmoijan tehokkuus heijastuu projektin, yrityksen ja loppukäyttäjän kokemukseen. Tehokkaammalla ohjelmoimisella voidaan tuottaa ohjelmistoja pienemmillä kustannuksilla, jotta kaikki sidosryhmät hyötyvät tästä tavoitteesta.

Tässä tutkimuksessa on tavoitteena löytää, mitkä ovat merkittävimmät tekijät yksittäisen ohjelmoijan tehokkuudessa. Tekijät joihin ohjelmoija voi itse vaikuttaa ja tekijät, jotka tulevat ympäristöstä. Tutkimuksessa on otettu ohjelmoinnin tehokkuuteen liittyviä tekijöitä laajasti kirjallisuuden mukaan ja keskitytään ei teknisiin asioihin, eli työkalut ja teknologiat on sivuutettu. Lisäksi tarkastellaan tehokkuutta pariohjelmoinnin näkökulmasta, koska sillä on suora vaikutus yksittäisen ohjelmoijan saavuttamaan tehokkuuteen.

Ympäristö ja ohjelmoijan tunteet vaikuttavat molemmat merkittävästi ohjelmoijan tehokkuuteen, joten molempia tulisi tarkastella, kun pyritään maksimaaliseen tehokkuuteen. Yksittäisen ohjelmoijan on hyvä tiedostaa myös ympäristön tekijät, jotka vaikuttavat tehokkuuteen, sillä hän itse voi myös vaikuttaa tehokkuuteen muuttamalla tai vaihtamalla ympäristöä.

Tässä tutkielmassa kartoitetaan aikaisemmasta tutkimuksessa löytynyttä tehokkuuteen liittyvää kirjallisuutta eri pääkategorioissa, jonka jälkeen tarkastellaan, mitkä ovat tavat tehostaa ohjelmoijan tehokkuutta. Tutkimuksessa löydettiin monia eri tekijöitä, jotka vaikuttavat tehokkuuteen huomattavasti. Erilaisilla toimintatavoilla ja ympäristön muutoksilla ohjelmoija ja organisaatio voi tehostaa omaa ohjelmiston tuottamiskykyään.

2 Tietotyöntekijä ja ohjelmoijien tehokkuus

Tässä luvussa tutkitaan ohjelmoijien tehokkuuden tutkimusta erilaisten näkökulmien kautta. Luvussa käydään läpi keskeisimmät termit, jotka liittyvät tehokkuuteen. Pyritään selventämään, mitkä ovat tekijöitä, jotka vaikuttavat yksittäisen ohjelmoijan tehokkuuteen ja millä eri tavoilla tätä voidaan hahmottaa ja mitata.

2.1 Tehokkuuden ja efektiivisyyden ero

Tehokkuudella tarkoitetaan jonkin kyseisen tehtävän tekemistä tehokkaasti suhteessa johonkin resurssiin, ja ohjelmoinnista puhuttaessa tässä kontekstissa resurssi on yleensä aika. Pelkän tehokkuuden mittaaminen ei ole välttämättä järkevää vaan on otettava huomioon myös muita asioita, kuten efektiivisyys. Efektiivisyydellä tarkoitetaan oikeiden asioiden tekemistä oikeaan aikaan (Drucker 1999). Tehokkuudella tarkoitetaan ohjelmoinnin kontekstissa yleensä tehtävän suorittamiseen käytettyä aikaa suhteessa tehtävän kokoon. Efektiivisyydellä taas pyritään vastaamaan, että mikä tehtävä on nyt tärkein tavoitteen saavuttamiseksi? Efektiivisyyttä tarvitaan siihe, että voimme panostaa tehokkuutta oikeaan tehtävään.

Tietotyöntekijä on työntekijä, jonka on kyettävä itse tekemään päätöksiä työssään ja mietittävä mihin, työpanosta tulisi tällä hetkellä asettaa tiimin tai organisaation tavoitteen saavuttamiseksi (Drucker 1999). Ollakseen tehokas, tietotyöntekijän on oltava myös efektiivinen. Ohjelmointi on melko manuaalista työtä, jos ohjelmoija tietää, mitä pitää tehdä ja osaa toteuttaa tämän tehtävän. Ohjelmoijan on kuitenkin päätettävä, mikä on tällä hetkellä kriittisin asia tavoitteen saavuttamiseksi, joten tästä syystä ohjelmoijat ovat tietotyöntekijöitä.

Drucker (1999) nostaa esiin kuusi suurta tekijää tietotyöntekijän tehokkuudesta:

- Tietotyöntekijän tehokkuus vaatii, että kysytään seuraava kysymys: "Mikä on tehtävä?"
- Se vaatii myös, että langetamme vastuun tehokkuudesta yksittäiselle tietotyöntekijälle itselleen. Tietotyöntekijän täytyy hallinnoida itseään ja heidän täytyy olla autonomisia.
- Jatkuvan innovoinnin täytyy olla osa työtä, se on tietotyöntekijän tehtävä ja vastuu.

- Tietotyön tekeminen vaatii jatkuvaa oppimista tietotyöntekijältä mutta yhtä paljon opettamista.
- Tietotyöntekijän tehokkuus ei ole ainakaan määrällisessä ulosannissa päätavoite. Laatu on vähintäänkin yhtä tärkeää.
- Viimeisenä tietotyöntekijän tehokkuus vaatii, että tietotyöntekijä nähdään ja häntä kohdellaan voimavarana, eikä kustannuksena. Lisäksi se vaatii, että tietotyöntekijä haluaa työskennellä organisaatiolle muiden mahdollisuuksien sijaan.

2.2 Mittaaminen ja haasteet

Yleisesti tiedossa oleva ongelma on tietotyöntekijän tehokkuuden ja efektiivisyyden mittaaminen, sillä ongelma ei ole niinkään laadun mittauksessa vaan siinä, että tietotyöntekijä suorittaa oikeaa tehtävää tavoitteen saavuttamiseksi. (Drucker 1999). Ongelmana on siis mitata työntekijän tehokkuutta merkittävällä tavalla, sillä aina kun työhön sisältyy päätöksiä, on myös huomioitava efektiivisyys eli oikeiden asioiden tekeminen.

Ohjelmoijan tehokkuutta on mitattu erilaisilla mittareilla, joista yksi perinteinen mittaustapa on koodirivien mittaaminen. Koodirivejä mitatessa mitataan ohjelmoijan tuottamaa koodia rivien määrässä. Koodirivimittaus on yksinkertainen ja ohjelmointikielestä riippumaton tapa mitata ohjelmoijan tehokkuutta.

Koodirivien mittaamisessa on perinteisellä tavalla useita ongelmia, sillä tiiviin koodin kirjoittaminen voi olla huomattavasti tehokkaampaa, ja kokeneet ohjelmoijat voivat kirjoittaa yhdelle riville saman funktionaalisuuden kuin kokematon ohjelmoija neljälle riville. Tähän on kuitenkin tarjottu ratkaisua muuttamalla koodirivimittausta hieman. Poistamalla mittauksista toistettu koodi, koodi jota ei lueta, testauskoodi ja muita vastaavia saadaan melko tehokas tapa mitata ohjelmoinnin tehokkuutta universaalisti kaikilla ohjelmointikielillä (Parareda ja Pizka 2007).

Toinen yleinen tapa mitata ohjelmoijan tehokkuutta on funkitiopisteet, jotka tulevat ketterän kehityksen prosessin mukana (Wagner ja Ruhe 2018). Funktiopisteet arvioidaan kehitystiimin kesken ja eri tehtäville asetetaan eri pistearvo. Vaativat tehtävät saavat suuremman arvon. Näiden pistearvioiden mukaan voidaan mitata ohjelmoijan tehokkuutta.

Funktiopistellä mittaaminen vaikuttaisi olevan tarkempaa kuin perinteinen koodirivien mittaaminen. Organisaation sisällä vaihtelu pistearvioissa tiimien välillä on noin 30 prosentin luokkaa, mutta organisaatioiden välillä vaihtelu voi olla suurempi. Käyttöönotto on myös yksi haaste funktiopisteissä, sillä pisteiden arvioiminen voi olla vaativaa. Erityisesti kokemattomissa tiimeissä vaihtelua pistearvioissa on havaittavissa enemmän. (Low ja Jeffery 1990)

Ohjelmoijille tehdyssä kyselyssä kysyttiin, mitkä mittarit auttaisivat oman tehokkuuden mittaamisessa eniten ja viisi suosituinta kohtaa olivat (Meyer ym. 2014):

- korjattujen bugien ja tehtyjen ohjelmointitehtävien määrä
- tiettyyn tehtävään käytetty aika
- koodin katselmointiin käytetty aika
- koodin kirjoittamiseen käytetty aika
- koodin katselmointeihin osallistumisen määrä

2.3 Ohjelmoijan tyytyväisyys ja onnellisuus

Ohjelmoijan onnellisuuden ja kustannustehokkaan ohjelmistotuotannon välillä on selkeä yhteys. Ohjelmoijat, jotka ovat onnellisempia ja tyytyväisempiä työhönsä, tuottavat tehokkaammin ja laadukkaammin koodia. Esimerkiksi tehokkuus ja motivaatio ovat parempia, jos ohjelmoija on onnellinen ja tyytyväinen. Onnellisuuden puute taas johtaa esimerkiksi huonompaan kognitiiviseen kykyyn, joka heijastaa suoraan tuotetun koodin määrään ja laatuun. (Graziotin ym. 2018)

Onnellisuuden puute vaikuttaa siis lähes kaikin puolin negatiivisesti ohjelmistotuotannon tehokkuuteen ja työntekijän hyvinvointiin, mutta kaikista eniten se näkyy ohjelmoijan tehokkuudessa (Graziotin ym. 2017). Osa ohjelmointi työstä vaatii luovuutta, joten hyvään tehokkuuteen pyrittäessä täytyy ohjelmoijan kyetä luoviin ratkaisuihin. Luovuus on korkeampi, jos ihminen on hyväntuulinen (Davis 2009).

Kuten Drucker (1999) totesi, tietotyötekijä täytyy nähdä voimavarana, eikä kustannuksena. Kun työntekijää kohdellaan voimavarana heillä on parempi mahdollisuus olla onnellisia työssään.

Ohjelmoinnissa on monia asioita, jotka tuottavat ohjelmoijille turhautumista. Ford ja Parnin (2015) löysivät useita eri kategorioita, joista seitsemän yleisintä olivat:

- Virheen aiheuttavan koodin löytäminen sovelluksesta
- Kokemattomuus ohjelmointityökaluissa
- Ohjelmointitehtävän suuri koko aiheuttaa turhautumista
- Uuteen projektiin asettuminen
- Tarvittavien resurssien puuttuminen, kuten dokumentaatio
- Ohjelmointi kokemuksen puute, esimerkiksi uudella ohjelmointi kielellä
- Tehtävän olettaminen yksinkertaiseksi, joka osoittautuu vaikeaksi

Myös kommunikointi tyylillä on merkitystä. Avoimen lähdekoodin projekteista on havaittu, että ystävällisempi tapa kommunikoida auttaa paremman tehokkuuden saavuttamisessa, sillä kehittäjät lähtevät mieluummin korjaamaa ongelmia, mitkä on kirjoitettu ystävällisesti (Destefanis ym. 2016).

2.4 Kokemus

Kokemuksella ja tehokkuudella ei välttämättä ole selkeää yhteyttä toistensa kanssa. Vaikka työhaastatteluissa monesti painotetaan kokemusta, se ei kuitenkaan ole niin merkityksellistä ohjelmoijan tehokkuuteen nähden. Merkityksellisempää on ohjelmoijan kyvykkyys suorittaa tehtävää. Pitkä työura alalla ei siis suoraan kerro ohjelmoijan tehokkuudesta. (Wagner ja Ruhe 2018)

Toisessa tutkimuksessa havaittiin sama ilmiö, eli kokemus työelämässä ei välttämättä nosta ohjelmoijan tehokkuutta, vaan se saavutetaan riippumatta kokemuksesta työelämässä, eli pienellä määrällä kokemusta voidaan päästä samaan tehokkuuteen kuin suurella. Tutkimuksessa havaittiin myös, että henkilöt, joilla on tutkinto tietotekniikasta, suoriutuvat paremmin tehokkuuden ja laadun osalta. (Dieste ym. 2017)

Tästä voidaan päätellä, että kokemuksella työelämässä ei ole vahvaa kausaliteettia tehokkuuden kanssa. Kokemusta ei tulisi siis arvostaa liikaa ohjelmoijan tehokkuuden arvioimisessa. On kuitenkin huomattu, että ohjelmoija, joilla on alle kahdeksan kuukauden kokemus

käytössä olevasta IDE-ympäristössä ovat tehottomampia. Tämä johtuu osittain siitä, että nykyiset IDE-ympäristöt ovat huomattavasti monimutkaisempia, kuin tekstieditorit. (Zayour ja Hajjdiab 2013).

Ohjelmoijan tehokkuus siis lisääntyy kriittisesti ohjelmointiuran alkuvaiheessa, sillä kokemus IDE-ympäristössä työskentelystä kasvaa ja kyky tuottaa koodia tehokkaasti siinä rinnalla. Suurella määrällä kokemusta ei kuitenkaan ole merkitystä tehokkuuteen. Kun tietty kompetenssitaso on saavutettu, alkaa kokemuksen merkitys vähentyä huomattavasti. Kirjallisuudesta ei kuitenkaan nouse esiin, minkä ajan jälkeen kokemus on riittävä maksimaalisen tehokkuuden saavuttamiseen.

2.5 Ajankäyttö

Yli kolmannes ohjelmoijien käyttämästä ajasta ei kulu teknisten asioiden parissa, vaan muissa työhön liittyvissä tehtävissä (Wagner ja Ruhe 2018). Tämä on yksi merkittävä tekijä efektiivisyyteen liittyen. Pelkkä tehokkuus ei siis riitä vaan ohjelmoijan on osattava myös päättää mikä on milloinkin tehokkain tapa edistää projektia. On kuitenkin huomattu, että erityisesti jos tiimin koko kasvaa kommunikoinnin lisääminen nostaa tehokkuutta (Wagner ja Ruhe 2018). Vaikka helposti saattaisi ajatella, että kommunikointiin käytetty aika on aikaa pois ohjelmoinnista tästä, voidaan havaita, että ohjelmoijan ja tiimin tulee käyttää aikaa myös kommunikointiin, jotta saadaan tehtyä oikeita asioita projektin kehityksen kannalta.

Myös ajankohta on merkittävä tehokkuuden kannalta. Myöhäisiltana tehdyt commitit (00:00 - 04:00) ovat bugisempia, kuin päivällä tehdyt commitit (Eyolfson, Tan ja Lam 2011). On siis syytä olettaa, että väsymyksellä on merkitystä ohjelmiston laadun kautta tehokkuuteen. Yöllä toteutetut commitit voivat olla kiireellisiä korjauksia rikkinäiseen ohjelmistoon, joten bugisuus voi johtua osittain myös tästä. Ohjelmoijan tulisi siis suorittaa tärkeät tehtävät, kun hän on pirteänä mahdollisuuksien mukaan.

Pidempi kuin neljänkymmenen tunnin työviikko vaikuttaa negatiivisesti sekä työntekijään, että työn antajaan. Työntekijöiden todennäköisyys tehdä virheitä kasvaa, kun työpäivän pituus ylittää tuntia. Lisäksi pitkillä työviikoilla on negatiivinen vaikutus työntekijöiden terveyteen. (Caruso ym. 2004)

2.6 Ympäristö ja keskeytykset

Ohjelmoijat kokevat, että heidän työpäivänsä oli tehokas, jos he saivat aikaan monta pientä tehtävää, tai yhden ison tehtävän ilman keskeytyksiä tai kontekstin vaihtoja. Samassa tutkimuksessa kuitenkin todettiin, että ohjelmoijat vaihtavat tehtävää 6,2 minuutin välein ja aktiviteettiä 1,6 minuutin välein ja tuntevat silti olevansa tehokkaita. Tämä voidaan selittää sillä, että kaikki keskeytykset, tai tehtävän vaihdokset eivät ole yhtä rasittavia tehokkuudelle.(Meyer ym. 2014). Rasittavimmat tehtävän vaihdokset ovat sellaisia, jotka vaativat ajattelutavan muutosta. Esimerkiksi koodauksen keskeyttäminen työkaverilta avunpyytämiseksi ei ole raskas tehtävän vaihdos sillä konteksti pysyy samana.

Tutkimuksessa, jossa testattiin ohjelmoijien ajankäyttöä tietyn tehtävän suorittamiseksi, jaettiin parhaat ja huonoimmat 25 prosenttia ryhmiin ja kysyttiin heidän työympäristöönsä liittyviä kysymyksiä. Kysymykset koskivat esimerkiksi keskeytyksiä, työtilan kokoa ja työpisteen äänekkyyttä. Tehokkaimmat ohjelmoivat työskentelivät keskimäärin paremmissa olosuhteissa, kuin vähemmän tehokkaat. (DeMarco ja Lister 1985)

Avoimessa konttorissa työskentelevät ihmiset kokevat, että heidät keskeytetään useasti 65 prosenttia ajasta, kun taas omassa huoneessa työskentelevät vain 29 prosenttia ajasta (Haynes 2008). Keskeytyksen jälkeen virheen tekemisen todennäköisyys kasvaa (Brumby, Janssen ja Mark 2019).

Kotoa työskentely on noussut suosiossaan huomattavasti. Kotona työskentelystä on huomattu positiivisa vaikutuksia tehokkuuteen ja erityisesti työntekijöiden tyytyväisyyteen (Bloom ym. 2015). Ohjelmoijien työ on luonteeltaan kotona työskentelyyn sopivaa, joten kotoa käsin työskentelyä kannattaa pitää vaihtoehtona tehokkuuden nostamiselle.

Älypuhelimen käytöllä ja tehokkuudella on myös vahva korrelaatio. Ihmiset kokevat, että älypuhelimen käyttö heikentää produktiivisuutta, sekä työelämässä, että vapaa ajalla (Duke ja Montag 2017). Älypuhelimessa oleville sovelluksille on tyypillistä, että ne antavat käyttäjälle useasti erilaisia ilmoituksia esimerkiksi sähköpostin saapumisesta. Jos puhelimessa on äänet päällä tämä keskeyttää ohjelmoijan työskentelyn ja se vaikuttaa negatiivisesti ohjelmointityön nopeuteen.

2.7 Persoonallisuus ja ohjelmointityyli

Persoonallisuudella ja ohjelmointityylin valinnalla on merkitystä parhaimman tehokkuuden saavuttamiseen. Persoonallisuutta tarkastellaan monesti vakiintuneella viiden suuren persoonallisuuspiirteen viitekehyksellä. Piirteet ovat:

- neuroottisuus
- ekstroversio
- avoimuus uusille kokemuksille
- sovinnollisuus
- tunnollisuus

(Digman 1990).

Merkittävimmät persoonallisuuspiirteet ohjelmointityylin valinnassa ovat tunnollisuus ja avoimuus uusille kokemuksille (Karimi ym. 2016). Muiden persoonallisuuspiirteiden ja tehokkuuden välillä ei havaittu merkityksellistä yhteyttä.

Cox ja Fisher (2009) havaitsivat, että ohjelmointityylit voidaan jaotella neljään eri osaan. Ohjelmointi tyylejä ovat alhaalta-ylös, ylhäältä-alas, syvyys-ensin ja leveys-ensin. Syvyys ja leveys näkökulmat eroavat toisistaan sillä, että syvyydessä tehdään yksi ominaisuus kerralla valmiiksi ja leveydessä tehdään yksi osa ohjelmasta kerralla valmiiksi. Ylhäältä alaspäin rakennettaessa tehdään ominaisuuksia isommalla riskillä, jotka toimivat vasta, kun alemman tason koodi on valmista ja alhaalta ylös toteutettaessa rakennetaan asetelmasta, jonka tiedetään toimivan.

Ohjelmoijat, joiden tunnollisuus on korkea on havaittu käyttävän luonnostaan syvyys-ensin lähestymistapaa ja se on myös suositeltava tapa henkilöille, joilla on korkea tunnollisuus piirre. Tunnolliset ohjelmoijat ovat tehokkaita iteratiivisissa tehtävissä. Ohjelmoijat, joilla on korkea avoimuus uusille kokemuksille käyttävät yleensä levelys-ensin lähestymistapaa ja heidän kannattaa lähestyä ohjelman kehittämistä opportunistisemmasta asemasta saavuttaakseen tavoitteensa nopeammin. (Karimi ym. 2016)

Voidaan myös päätellä, että ylhäältä-alaspain suuntautuvassa toteuttamistavassa riskit ovat korkeat, joten matala neuroottisuus piirre ohjelmoijalla nostaa todennäköisyyttä käyttää tätä

mallia, sillä riskinsieto kyky on korkeampi. Korkea neuroottisuus sen sijaan painottaa riskien välttämistä, joten ohjelmoija kellä tämä piirre on korkea, on helpompi suorittaa työtä alhaalta ylöspäin.

2.8 Pariohjelmointi

Pariohjelmoinnilla tarkoitetaan, että kaksi ohjelmoijaa työskentelevät yhdellä tietokoneella yhden ongelman kanssa ja tällä toiminnalla voi olla erilaiset vaikutukset tehokkuuteen. Pariohjelmoinnin tehokkuuteen vaikuttavat tavoitteet, eli onko nopeus vai laatu korkein prioriteetti. Myös eri tasoisilla ohjelmoijilla on erilaiset vaikutukset, eli tulee ottaa huomioon myös ohjelmoijien kokemus kyseisen tehtävän toteuttamisessa.

Pariohjelmmoinnista on havaittu, että persoonallisuus piirteillä ei vaikuta olevan merkittäviä vaikutuksia tehokkuuteen (Hannay ym. 2009). Toisessa tutkimuksessa kuitenkin huomattiin, että seuraavat asiat vaikuttavat positiivisesti pariohjelmoinnin tehokkuuteen.

- efektiivinen kommunikointi
- koettu mukavuus muiden kanssa työskennellessä
- itsevarmuus omiin kykyihin
- kyky tehdä kompromisseja

(Dick ja Zarnett 2002).

Näissä tutkimuksissa tarkasteltiin persoonallisuutta eri tavalla, mikä saattaa vaikuttaa tulosten eriäväisyyteen. Hannay ym. (2009) tarkastelivat persoonallisuutta viiden suuren persoonallisuus piirteen näkökulmasta ja Dick ja Zarnett (2002) toivat esiin tarkempia ominaisuuksia persoonallisuudesta. Korkean tason persoonallisuus piirteillä ei siis ole merkitystä tehokkuudessa, mutta tarkemmat persoonallisuus ominaisuudet, kuten itsevarmuus omiin kykyihin vaikuttaa tehokkuuteen positiivisesti.

Yleisesti pariohjelmoinnista voidaan todeta, että tehokkuus ei kehity, kun pariohjelmoidaan, mutta laatu kasvaa riippuen ohjelmoijien tasosta ja tehtävän monimutkaisuudesta. Kaksi kokematonta ohjelmoijaa voi tuottaa laadukkaampaa koodia yhdessä, kuin tehdessään yksin. Sama pätee myös keskitason ohjelmoijiin, mutta vain jos tehtävä on vaativa. Kokeneet oh-

jelmoijat saava hyötyjä vain, jos tehtävä on erittäin vaativa ja uskotaan, että se ei ole yksin ratkaistavissa helposti. (Dybå ym. 2007)

3 Ohjelmoijan tehokkuuden lisääminen

Tässä luvussa kartoitetaan, miten ohjelmoijan tehokkuuteen voidaan vaikuttaa positiivisesti. Kuten aikaisemmin todettiin ohjelmoijat ovat tietotyöntekijöitä. Drucker (1999) nosti esiin, että tietotyöntekijän on kysyttävä, että mikä on tehtävä, jotta voidaan kohdistaa tehokkuus oikeisiin tehtäviin. Tehokkuus menee hukkaan ilman efektiivisyyttä.

Tavoitteiden asettaminen on yksi tehokas tapa nostaa tehokkuutta ja motivaatiota. Tämän toimintatavan vaikuttavuutta voidaan tehostaa, jos omaa toimintaa tarkkaillaan ja mitataan jollakin tavalla (Meyer ym. 2014). Ohjelmoijan tulisi kyetä asettamaan itselleen tavoitteita ja mittareita, joilla voidaan mitata oman toiminnan tehokkuutta. Tällä tavoin ohjelmoija pystyy havaitsemaan, mitkä toimintamallit hänellä toimivat ja mitkä eivät.

Ohjelmoijien tyytyväisyys ja onnellisuus on erittäin keskeinen tehokkuuteen vaikuttava tekijä (Graziotin ym. 2018). Työnantajien, tiiminvetäjien ja ohjelmoijien itsensä on hyvä tiedostaa tämä muuttuja tehokkuudessa. Ohjelmoijien tyytyväisyyteen panostetaan nykyään reilusti resursseja, ja tyytyväinen ohjelmoija on myös yritykselle hyvä asia. Näin ollen molemmat osapuolet hyötyvät siitä, että ohjelmoija nähdään voimavarana ja hänen onnellisuuteen panostetaan. Kotoa työskenteleminen lisää työntekijän tyytyväisyyttä ja tehokkuutta (Bloom ym. 2015). Tyytyväisyys ja tehokkuus korreloivat keskenään, joten on mahdollista, että kotoa käsin työskentelyn tehokkuuden lisääntyminen johtuu tyytyväisyyden kohentumisesta.

Kokemus ei vaikuta ohjelmoijan tehokkuuteen tietyn kompetenssitason saavuttamisen jälkeen (Wagner ja Ruhe 2018). Kokemattomien ohjelmoijien tulisi siis pyrkiä nopeasti riittävälle kompetenssitasolle, jotta he voisivat saavuttaa maksimaalisen tehokkuuden. Työnantajien tulisi ottaa tämä huomioon rekrytoinnissaan, eikä asettaa liiallista painoa hakijoiden kokemuksen määrälle.

Ajan käyttäminen projektin alkuvaiheessa asiakkaan tarpeiden ymmärtämiseen lisää kehitysprosessin nopeutta. Ohjelmoija, joka ymmärtää asiakkaan tarpeet heti projektin alkuvaiheessa, pääsee haluttuihin tavoitteisiin nopeammin. Prototyyppien tekeminen ja nopea palautesykli asiakkaalta pienentää uudelleenkirjoitettavan koodin määrää, joten prosessi on tehokkaampi. Lisäksi ohjelmoijat toimivat tehokkaammin pienemmissä tiimeissä. (Blackburn,

Scudder ja Van Wassenhove 1996)

Yksittäisen ohjelmoijan tulisi pyrkiä ymmärtämään asiakkaan tarpeita prosessin alkuvaiheessa ja tuottaa ohjelmistoa pieninä inkrementteinä. Tehokkuuden nostamiseksi ohjelmoijan on valittava pienempi tiimi isomman sijaan.

Ajankäytössä on otettava huomioon työviikon kesto. Työviikon ylittäessä neljäkymmentä tuntia vaikutukset ovat negatiivisia (Caruso ym. 2004). Organisaatioiden ja tiimien tulisi siis välttää työviikon keston lisäämistä yli neljänkymmenen tunnin. Lyhyemmän työviikon ja tehokkuuden välillä ei ole havaittu positiivista korrelaatiota.

Oikeanlainen työympäristö voi auttaa ohjelmoijaa saavuttamaan paremman tehokkuuden melko pienillä ja yleensä kustannustehokkailla ratkaisuilla. Ympäristön vaikutukset ovat tehokkuudessa huomattavat, kuten DeMarco ja Lister (1985) toteavat tutkimuksessaan. Pienemmällä määrälle keskeytyksiä voidaan saavuttaa parempi tehokkuus. On otettava myös huomioon, mikä on keskeytyksistä eroon pääsemisen hinta. Esimerkiksi kommunikoinnin vähentäminen aiheuttaa ongelmia erityisesti isoissa tiimeissä (Wagner ja Ruhe 2018). Keskeytysten välttämisen ja kommunikoinnin määrän välisen tasapainon löytäminen voi olla vaativaa.

Pariohjelmointi ei ole kaikissa tilanteissa tehokkuuden kannalta optimaalinen valinta. Dybå ym. (2007) havaitsivat, että ohjelmointitehtävän ollessa vaativa pariohjelmoinnista saadaan tehokkuuteen ja laatuun positiivisia vaikutuksia. Ohjelmoijien tulisi siis harkita pariohjelmoinnin toteuttamista aina, kun tehtävä on taitotasoon nähden vaativa.

4 Pohdinta

Ohjelmoijan tehokkuus on erittäin kompleksinen asia. On useita tekijöitä, jotka vaikuttavat ohjelmoijan tehokkuuteen. Tämä lisäksi suurin osa näistä tekijöistä on itsessään kompleksisia, kuten esimerkiksi työympäristö. Lisähaasteensa tähän tuo mittaamisen vaativuus. Tehokkuuden mittarit eivät yleensä ole tarkkoja, ja efektiivisyyden mittaaminen on aina tapauskohtaista.

On tärkeää muistaa, että ohjelmoinnin tehokkuus ei ole tavoite itsessään, vaan se on tapa auttaa ohjelmoijia käyttämään kykyjään tehokkaasti (Boehm 1987). Tässä huomataan Druckerin (1999) tekemä huomio tietotyöntekijöiden mittaamisesta efektiivisyyden ja laadun, eikä pelkän tehokkuuden mukaan. Oikeilla toimintatavoilla voidaan saavuttaa tavoitteet nopeasti, kun tehdään oikeita asioita tehokkaalla tavalla. Kun tehokkuus ja efektiivisyys kohtaavat, saavutetaan optimaalisin tapa tehdä ohjelmistoja.

Ohjelmoijien näkeminen voimavarana ja heidän tyytyväisyyteen, sekä työympäristön laatuun panostaminen ei ole nollasummapeliä, vaan siinä päästään parempaan lopputulokseen kaikkien osapuolien kannalta.

Ohjelmoijien tehokkuuden hahmottaminen on vaativaa ja monimuotoista. Ohjelmoijien tulisi tutkia omaa tehokkuuttaan ja yrittää löytää itselleen sopivimmat käytännöt. Lisäksi organisaatioiden ja tiimien kannattaa tukea yksittäisen ohjelmoijan tehokkuutta, sillä se on myös heille eduksi.

Lähteet

Blackburn, Joseph D, Gary D Scudder ja Luk N Van Wassenhove. 1996. “Improving speed and productivity of software development: a global survey of software developers”. *IEEE transactions on software engineering* 22 (12): 875–885.

Bloom, Nicholas, James Liang, John Roberts ja Zhichun Jenny Ying. 2015. “Does working from home work? Evidence from a Chinese experiment”. *The Quarterly Journal of Economics* 130 (1): 165–218.

Boehm, Barry W. 1987. “Improving software productivity”. *Computer,* numero 9: 43–57.

Brumby, Duncan P, Christian P Janssen ja Gloria Mark. 2019. “How do interruptions affect productivity?” Teoksessa *Rethinking Productivity in Software Engineering,* 85–107. Springer.

Caruso, Claire C, Edward M Hitchcock, Robert B Dick, John M Russo ja Jennifer M Schmit. 2004. “Overtime and extended work shifts; recent findings on illnesses, injuries, and health behaviors”.

Davis, Mark A. 2009. “Understanding the relationship between mood and creativity: A meta-analysis”. *Organizational behavior and human decision processes* 108 (1): 25–38.

DeMarco, Tom, ja Tim Lister. 1985. “Programmer performance and the effects of the workplace”. Teoksessa *Proceedings of the 8th international conference on Software engineering,* 268–272. IEEE Computer Society Press.

Destefanis, Giuseppe, Marco Ortu, Steve Counsell, Stephen Swift, Michele Marchesi ja Roberto Tonelli. 2016. “Software development: do good manners matter?” *PeerJ Computer Science* 2:e73.

Dick, Andrew J, ja Bryan Zarnett. 2002. “Paired programming and personality traits”. *XP2002, Italy* 18.

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.

Digman, John M. 1990. “Personality structure: Emergence of the five-factor model”. *Annual review of psychology* 41 (1): 417–440.

Drucker, Peter F. 1999. “Knowledge-worker productivity: The biggest challenge”. *California management review* 41 (2): 79–94.

Duke, Éilish, ja Christian Montag. 2017. “Smartphone addiction, daily interruptions and self-reported productivity”. *Addictive behaviors reports* 6:90–95.

Dybå, Tore, Erik Arisholm, Dag IK Sjøberg, Jo E Hannay ja Forrest Shull. 2007. “Are two heads better than one? On the effectiveness of pair programming”. *IEEE software* 24 (6): 12–15.

Eyolfson, Jon, Lin Tan ja Patrick Lam. 2011. “Do time of day and developer experience affect commit bugginess?” Teoksessa *Proceedings of the 8th Working Conference on Mining Software Repositories,* 153–162.

Ford, Denae, ja Chris Parnin. 2015. “Exploring causes of frustration for software developers”. Teoksessa *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering,* 115–116. IEEE.

Graziotin, Daniel, Fabian Fagerholm, Xiaofeng Wang ja Pekka Abrahamsson. 2017. “Unhappy developers: Bad for themselves, bad for process, and bad for software product”. Teoksessa *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C),* 362–364. IEEE.

———. 2018. “What happens when software developers are (un) happy”. *Journal of Systems and Software* 140:32–47.

Hannay, Jo E, Erik Arisholm, Harald Engvik ja Dag IK Sjoberg. 2009. “Effects of personality on pair programming”. *IEEE Transactions on Software Engineering* 36 (1): 61–80.

Haynes, Barry P. 2008. “The impact of office layout on productivity”. *Journal of facilities Management.*

Karimi, Zahra, Ahmad Baraani-Dastjerdi, Nasser Ghasem-Aghaee ja Stefan Wagner. 2016. “Links between the personalities, styles and performance in computer programming”. *Journal of Systems and Software* 111:228–241.

Low, Graham C., ja D. Ross Jeffery. 1990. “Function points in the estimation and evaluation of the software process”. *IEEE transactions on Software Engineering* 16 (1): 64–71.

Meyer, André N, Thomas Fritz, Gail C Murphy ja Thomas Zimmermann. 2014. “Software developers’ perceptions of productivity”. Teoksessa *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 19–29.

Parareda, B, ja Markus Pizka. 2007. “Measuring productivity using the infamous lines of code metric”. Teoksessa *Proceedings of SPACE 2007 Workshop, Nagoya, Japan.*

Wagner, Stefan, ja Melanie Ruhe. 2018. “A systematic review of productivity factors in software development”. *arXiv preprint arXiv:1801.06475.*

Zayour, Iyad, ja Hassan Hajjdiab. 2013. “How much integrated development environments (ides) improve productivity?” *JSW* 8 (10): 2425–2431.