Leevi Simula

JOUKKOISTAMINEN OHJELMISTOTUOTANNOSSA

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2019

TIIVISTELMÄ

Simula, Leevi
Joukkoistaminen ohjelmistotuotannossa
Jyväskylä: Jyväskylän yliopisto, 2019, 35 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Palonen, Teija

Kovaa vauhtia digitalisoituva yhteiskunta kasvattaa ohjelmistojen tarvetta aina infrastruktuurijärjestelmistä yritysten ja organisaatioiden tietojärjestelmiin. Yksi keino tuottaa ohjelmistoja tehokkaammin on joukkoistaminen, jossa osaamiseltaan monimuotoisen joukon osaamista hyödynnetään ongelmien ratkaisemisessa. Tämän tutkimuksen tarkoituksena oli tutkia kirjallisuuskatsauksen muodossa, miten joukkoistamista voidaan hyödyntää ohjelmistotuotannon eri vaiheissa. Tutkielman tavoitteena oli myös kerätä yhteen tietoa joukkoistamisen hyödyntämisen mahdollisuuksista, jotta esimerkiksi joukkoistamista liiketoiminnassaan harkitsevat yritykset saisivat tutkielmasta kuvan, miten he voisivat hyödyntää joukkoistamista ohjelmistotuotannossaan. Tutkielman perusteella todettiin, että joukkoistamista hyödynnetään erityisesti ohjelmistoprojektin ohjelmointi- ja testausvaiheessa, joista on julkaistu monipuolisesti tieteellisiä artikkeleita. Maailmalla on myös kaupallisia markkinapaikkoja, jotka tarjoavat joukkoistamista näihin ohjelmistoprojektin vaiheisiin. Joukkoistamista voidaan hyödyntää myös ohjelmistoprojektin määrittely-, suunnittelu- sekä käyttöönotto- ja ylläpitovaiheissa, mutta tieteellisiä artikkeleita oli julkaistu näistä vaiheista melko vähän. Lisäksi markkinoilla ei juurikaan ole yrityksiä, jotka tarjoaisivat joukkoistamista näihin ohjelmistoprojektin vaiheisiin. Tutkielman perusteella joukkoistetun ohjelmistotuotannon suurimmiksi eduiksi voidaan sanoa nopeus, edullisuus ja joustavuus. Suurimmiksi haasteiksi puolestaan kuuluvat joukon motivoiminen ja joukkoistetun työn laatu. Joukkoistettu ohjelmistotuotanto on kasvattanut viime vuosina suosiotaan, ja laajempi tutkimus sen hyödyistä ja haasteita kannustaa sen käytön lisääntymiseen myös tulevaisuudessa.

Asiasanat: Joukkoistaminen, ohjelmistotuotanto, joukkoistettu ohjelmistotuotanto

ABSTRACT

Simula, Leevi
Crowdsourcing in software development
Jyväskylä: University of Jyväskylä, 2019, 35 p.
Information Systems, Bachelor's Thesis
Supervisor: Palonen, Teija

In the rapidly digitizing society, the need for software from infrastructure systems to enterprise and organization information systems will increase. One way to develop software more efficiently is crowdsourcing which means that the know-how of a diverse group of professionals is utilized to solve problems. This thesis was conducted as a literature review. The purpose was to study how crowdsourcing can be utilized in the different phases of software engineering. The aim of the thesis was also to collect information on the potential of exploiting crowdsourcing, so that companies considering the use of crowdsourcing would get an idea on how to use crowdsourcing in software engineering. Based on the thesis, it was found that crowdsourcing is utilized especially during the programming and testing phases of a software process, and this has been the subject of various scientific articles. There are also commercial crowdsourcing platforms offering crowdsourcing services for these phases. Crowdsourcing can also be utilized in the following phases of the software process: requirements engineering, design, and implementation and maintenance. However, there were relatively few scientific articles covering these topics. In addition, there are hardly any companies offering crowdsourcing possibilities for these phases. According to the thesis, the greatest benefits of Crowdsourced Software Engineering are speed, affordability and flexibility. The biggest challenges, however, include motivating the team and the quality of work. Crowdsourced Software Engineering has gained popularity in recent years, and a growing body of research on its benefits and challenges will encourage its use also in the future.

Keywords: crowdsourcing, software engineering, Crowdsourced Software Engineering

SISÄLLYS

1 JOHDANTO

Moderni maailma on riippuvainen ohjelmistoista, jotka ovat tärkeässä roolissa valtionhallinnon, yhteiskunnan, organisaatioiden ja yritysten toiminnassa. Valtioiden rahajärjestelmät, infrastruktuurit ja teollisuus toimivat ohjelmistojen varassa ja maailma digitalisoituu yhä kovempaa vauhtia. Ohjelmistotuotannon tavoitteena on tukea ja tehostaa ammattilaisia ohjelmistosuunnittelijoita ohjelmistojen valmistamisessa. Ohjelmistotuotannolla (engl. software engineering) tarkoitetaan tietokoneohjelmien rakentamista sekä sitä tukevia tekniikoita, työkaluja, menettelytapoja ja periaatteita. Ohjelmistoprojekti on osa ohjelmistotuotantoa, joka koostuu joukosta peräkkäisiä toimia, joiden tavoitteena on ohjelmiston tuottaminen. (Sommerville, 2016, s. 18-23.) Ohjelmistoprojektin voidaan jakaa viiteen toisistaan erottuvaan vaiheeseen: määrittely, suunnittelu, ohjelmointi, testaus sekä käyttöönotto ja ylläpito. Vaikka erilaiset ohjelmistoprojektit eroavat toisistaan huomattavasti toistuvat edellä mainitut vaiheet lähes jokaisessa ohjelmistoprojektissa. Eri vaiheissa käytetään erilaisia toimintatapoja ja työkaluja, joiden avulla pyritään tukemaan ohjelmiston kehittämistä. (Haikala & Mikkonen, 2011, s. 30.)

Mediassa on viime aikoina puhuttu paljon alustataloudesta ja sen vaikutuksista tulevaisuuden työntekoon. Joukkoistaminen (engl. crowdsourcing) on yksi alustatalouden muodoista ja se mahdollistaa uudenlaisia työnteon tapoja ja vaikuttaa siksi koko yhteiskuntaan. Joukkoistamisen ideana on jakaa tehtäviä, kuten ideointia, ongelmanratkaisua, arviointia tai pieniä työtehtäviä, suurelle joukolle ihmisiä (Howe, 2006a). Monimuotoista osaamista sisältävä joukko onnistuu ratkaisemaan ongelman tehokkaammin kuin samankaltaista osaamista sisältävä joukko (Hong & Page, 2004.). Joukkoistamisen idea perustuu suuren joukon ideoiden, luovuuden ja kykyjen hyödyntämiseen. Mitä suurempi on joukko, sitä todennäköisemmin siinä on monimuotoisempaa osaamista ja näin ollen se suoriutuu paremmin ongelmanratkaisutilanteessa. Internet on mahdollistanut suurien massojen keräämisen yhteen, jolloin joukkoistamisen toteuttaminen on helpottunut. (Howe, 2006a.)

Digitalisoituva maailma tarvitsee yhä enemmän erilaisia ohjelmistoja sulautetuista järjestelmistä ja monimutkaisiin maailmanlaajuisesti toimiviin tieto-

järjestelmiin, joten onkin tärkeää tutkia erilaisia tapoja, joilla ohjelmistoja voidaan tuottaa tehokkaammin. Yksi näistä keinoista on joukkoistaminen ja sen käyttö ohjelmistotuotannossa onkin lisääntynyt viime vuosia ja käytön odotetaan kasvavan myös tulevaisuudessa (Mao, Capra, Harman & Jia 2017). Suomenkielistä tutkimusta joukkoistamisesta on vähänlaisesti, joten kirjallisuuskatsauksen tekeminen aiheesta tulee tarpeeseen. Yritykset, jotka voisivat hyödyntää joukkoistamista ohjelmistotuotannossaan eivät välttämättä tiedä, millaisin eri keinoin se on mahdollista. Tämän tutkimuksen tavoitteena onkin selventää lukijalle, millaisin eri tavoin joukkoistamista voidaan hyödyntää ohjelmistotuotannossa. Tämän tutkielman tavoitteena on vastata seuraavaan tutkimuskysymykseen:

- Miten joukkoistamista voidaan hyödyntää ohjelmistotuotannossa?

Tämä tutkielma on toteutettu kirjallisuuskatsauksena. Tutkielmassa keskitytään tarkastelemaan, miten joukkoistamista sovelletaan ohjelmistoprojektin eri vaiheissa. Tässä tutkielmassa esitellään ohjelmistotuotantoa, joukkoistamista ja joukkoistettua ohjelmistotuotantoa käsittelevää tieteellistä kirjallisuutta. Lähteiden valintaprosessissa pyrittiin valitsemaan kaikista laadukkaimmat ja relevanteimmat lähteet. Lähteiden valintaan vaikutti se, kuinka monta kertaa niihin oli viitattu Google Scholar-palvelun mukaan. Tutkimukseen ei valittu lähteitä, joihin oli viitattu vain vähän tai ei laisinkaan. Suurin osa tutkielmassa käytetyistä lähteistä on valittu julkaisukanavista, jotka on arvioitu Julkaisufoorumissa johtavaksi tasoksi tai korkeimmaksi tasoksi. Tutkielmassa on myös mukana lähteitä, jotka ovat Julkaisufoorumin mukaan perustasoa sekä muutamia verkkosivuja, jotka eivät täytä tieteellisen lähteen kriteereitä. Tutkielman lähdemateriaalia etsittiin tietokannoista lähinnä englannin kielellä muun muassa seuraavilla hakusanoilla: crowdsourcing, software engineering, software development ja crowdsourced software engineering. Lisäksi lähteitä löydettiin hakusanoilla löydettyjen, useita viittauksia saaneiden, lähteiden lähdeluetteloista. Lisäksi lähdekirjallisuutta löydettiin Jyväskylän yliopiston kirjastosta.

Tutkielma on jaettu viiteen lukuun, joista ensimmäinen on johdanto. Johdannon jälkeen siirrytään toiseen lukuun, jossa käsitellään ohjelmistotuotannon erivaiheita ja käydään läpi, mitä työtehtäviä eri vaiheet sisältävät, ja minkälaisia työkaluja niissä käytettään. Kolmannessa luvussa esitellään erilaisia joukkoistamisen määritelmiä sekä joukkoistamisen erilaisia ilmenemismuotoja. Neljännessä luvussa esitellään, miten joukkoistamista voidaan hyödyntää ohjelmistoprojektin eri vaiheiden aikana. Lisäksi luvussa esitellään erilaisia joukkoistamista hyödyntäviä työkaluja, joita tutkijat ovat kehittäneet ohjelmistotuotannon tueksi. Viimeisessä luvussa tiivistetään tutkimuksen tulokset ja esitellään jatkotutkimusaiheita.

2 OHJELMISTOTUOTANNON VAIHEET

Ohjelmistoprojekti sisältää joukon toimia, jotka johtavat ohjelmiston tuottamiseen. Koska ohjelmistoja on olemassa paljon erilaisia, ei ole olemassa kaikkiin tilanteisiin sopivaa toimintamallia (Sommerville, 2016, s. 44). Ohjelmistoprojektiin katsotaan kuitenkin liittyvän aina määrittelyyn, suunnitteluun, ohjelmointiin, testaustaukseen sekä käyttöönottoon ja ylläpitoon liittyviä toimia (Haikala & Mikkonen, 2011, s. 30; Sommerville, 2016 s. 44). Toimet ovat useimmiten erotettavissa ohjelmistoprojektin vaiheina, joilla on alku- ja loppuajankohdat. Tässä tutkielmassa ohjelmistoprojekti jaotellaan Haikalan ja Mikkolan (2011, s. 30) esittämän jaottelun mukaan määrittelyyn, suunnitteluun, ohjelmointiin, testaukseen sekä käyttöönottoon ja ylläpitoon. Tässä luvussa käydään läpi, mitä toimia kussakin ohjelmistoprojektin vaiheessa tehdään ja mitä työkaluja näissä toimissa vaaditaan. Riippuen käytettävästä projektimallista ja kehitettävästä ohjelmistosta, ohjelmistoprojektin vaiheiden järjestys voi muuttua ja niitä voidaan myös suorittaa päällekkäin (Sommerville, 2016, s. 54).

2.1 Määrittely

Määrittely on prosessi, jossa pyritään ymmärtämään ja määrittelemään, mitä ohjelmiston halutaan tekevän ja tunnistamaan rajoitteet ohjelmiston toiminnassa ja kehittämisessä. Asiakkaalla, jolle ohjelmistoprojektia tehdään, on usein asiakasvaatimuksia, joiden pohjalta pyritään dokumentoimaan ohjelmistojen toimintoja asiakasnäkökulmasta. Prosessia, jonka tavoitteena on luoda vaatimukset kattava sopimus, joka vastaa sidosryhmien tarpeita, kutsutaan vaatimuskäsittelyksi. (Sommerville, 2016, s. 54; Haikala & Mikkonen, 2011, s. 30.) Vaatimuskäsittely on oleellinen osa ohjelmistoprojektia. Leffingwellin ja Widrigin (2003, s. 7-13) mukaan suurin osa ohjelmistoprojektin epäonnistumisista johtuu huonosti tehdystä vaatimusmäärittelystä, koska vaatimusmäärittelyssä tehdyt virheet ovat yleensä kalleimpia ja vaikeimpia korjata.

Vaatimuskäsittelyssä kartoitetaan vaatimuksia, joita ohjelmistolla on oltava. Haikala ja Mikkonen (2013, s. 61) luokittelevat vaatimukset kolmeen eri luokkaan: toiminnalliset vaatimukset, ei-toiminnalliset vaatimukset ja reunaehdot. Toiminnallisia vaatimuksia ovat toiminnot, joita ohjelmistolla pitää pystyä tekemään. Ei-toiminnalliset vaatimukset määrittelevät millainen ohjelmisto on ja miten se toteuttaa vaatimukset. reunaehdot määrittelevät puitteet, jonka rajoissa ohjelmiston on toimittava. Reunaehtona voi olla esimerkiksi se, mitä ohjelmointikielellä ohjelmisto on toteutettava. Vaatimuskäsittely voidaan jakaa kahteen osaan: vaatimusmäärittelyyn ja vaatimustenhallintaan. Vaatimusmäärittelyllä (engl. requirements engineering) tarkoitetaan vaatimustenkäsittelyn niitä prosesseja, jotka pyrkivät selvittämään millaisia vaatimuksia ohjelmistolle on. Vaatimustenhallinnalla taasen tarkoitetaan hyväksyttyjen vaatimusten muuttamiseen liittyviä prosesseja. (Haikala & Mikkonen, 2011, s. 61-67.)

Myös vaatimusmäärittely voidaan jakaa neljään pienempään vaiheeseen: esille saaminen (engl. elicitation), analyysi, tarkka määrittely ja laadunvarmistus. Ensimmäisessä vaiheessa selvitetään sidosryhmien tarpeet ja olettamukset siitä, miten ohjelmiston pitäisi toimia ja hahmotellaan, minkälaiselle ohjelmistolle olisi tarvetta. Toisessa tehdään tietoon perustuvia päätöksiä ensimmäiseen vaiheen hahmotelmaa apuna käyttäen ja kolmannessa vaiheessa tehdään tarkka määritelmä ohjelmistosta hahmotelman ja analyysin perusteella. Neljännessä vaiheessa varmistetaan, että ohjelmisto vastaa tarpeisiin, joita varten se kehitettiin. (Van Lamsweerde, 2009, s. 31-33.)

Ennen vaatimuskäsittelyn aloittamista, ohjelmiston tarpeellisuudesta voidaan tehdä markkinatutkimus, jonka avulla saadaan selville, onko kyseisen kaltaiselle ohjelmistolle tarvetta tai markkinakysyntää ja onko ohjelmisto realistisesti toteutettavissa teknisesti ja taloudellisesti (Sommerville, 2016, s. 54). Vaatimuskäsittelyssä tehtävien määrittelydokumenttien tekemiseen käytetään sekä luonnollista kieltä että määrittelynotaatioita. Luonnollinen kieli on tärkein työkalu määrittelydokumenttien tekemisessä. Mitä ylemmällä tasolla määrittely tehdään, sitä parempi työkalu luonnollinen kieli yleensä on. Luonnollisen kielen avulla on helppo kuvata abstrakteja ja yleisiä määrittelyjä ja luonnollinen kieli auttaa myös tekemään määrittelydokumentista asiakkaan kannalta helpommin ymmärrettävän. Kun määrittelyssä kaivataan suurempaa tarkkuutta, käytetään määrittelynotaatioita, joiden avulla on luonnollista kieltä helpompaa kuvata ohjelmiston rakennetta ja toimintaa. Käytetyin määrittelynotaatio on UML-standardi, jossa esitellään kaaviotekniikoita, joiden avulla voidaan kuvailla rakennetta, käyttäytymistä ja vuorovaikutusta. (Haikala & Mikkonen, 2011, s. 69-72.)

2.2 Suunnittelu

Kun ohjelmisto on määritelty kunnolla, tiedetään ohjelmistoon kohdistuvat vaatimukset. Ohjelmistosuunnittelun avulla kuvataan ohjelmiston rakenne, jollaiseksi se halutaan toteuttaa, datamallit ja rakenteet, joita ohjelmisto käyttää,

järjestelmän komponenttien väliset rajapinnat, joilla ohjelmisto toimii sekä joskus myös ohjelmiston käyttämät algoritmit. Koko ohjelmistoa ei suunnitella kerralla, vaan sitä suunnitellaan vaihe kerrallaan aloittaen suuremmista kokonaisuuksista edeten pienempiin yksityiskohtiin. (Sommerville, 2016, s. 56.)

Ohjelmistosuunnittelun keskeisin keino on arkkitehtuurisuunnittelu (Haikala & Mikkonen, 2011, s. 177). Ohjelmistoarkkitehtuurin avulla määritellään ne rajat, joiden puitteissa ohjelmisto on rakennettava ja ylläpidettävä. Arkkitehtuurilla tarkoitetaan myös joukkoa päätöksiä liittyen koko ohjelmistoon tai sen pienempiin osiin, jotka rajoittavat ohjelmoijien ja ylläpitäjien oman liiallisen luovuuden käyttöä. (D'souza & Wills, 1998, s. 481-482.) Arkkitehtuurisuunnittelun avulla edetään määritellystä ohjelmistosta tekniseen kuvaukseen, jonka avulla ongelma ratkaistaan. Ohjelmistoarkkitehtuuri ei ole ohjelmiston rakenteen tarkka kuvaus vaan tarjoaa puitteet ohjelmiston kehitykselle, testaukselle ja ylläpidolle. Arkkitehtuuri on siis abstrakti kuvaus ohjelmistosta, joka ei muutu normaalin ylläpidon aikana. (Haikala & Mikkonen, 2011, s. 177-178.) Ohjelmistosuunnittelussa voidaan käyttää apuna myös sovelluskehystä. Sovelluskehys on integroitu joukko luokkia, komponentteja ja rajapintoja, jotka toimivat yhteistyössä mahdollistaen arkkitehtuurin uudelleenkäytön samankaltaisissa ohjelmistoissa. (Schmidt, Gokhale & Natarajan, 2004.) Sovelluskehys toimii ohjelmistokehyksenä kehitettävälle ohjelmistolle. Se ei yleensä ole suorituskelpoinen ohjelmisto vaan joukko ohjelmiston osia, jotka toimivat yhteen toistensa kanssa. Lisäämällä sovelluskehykseen uutta koodia, kehys erikoistetaan, jolloin siitä syntyy uusi ohjelmisto. (Haikala & Mikkonen, 2011, s. 189.)

Ohjelmistosuunnittelu ja ohjelmointi tapahtuvat usein päällekkäin toistensa kanssa varsinkin silloin, jos projektimallina käytetään ketterää kehitystä. Tällöin ohjelmoinnin aikana ei luoda suunnitteludokumentteja vaan suunnittelu punoutuu yhteen implementoinnin kanssa, jolloin suunnittelu tapahtuu pelkästään ohjelmoijien henkilökohtaisessa dokumentaatiossa. Sommerville (2016, s. 47) ei erottelekaan suunnittelu- ja ohjelmointivaihetta, vaan esittää ne yhtenä ohjelmiston suunnittelu- ja implementointivaiheena.

2.3 Ohjelmointi

Ohjelmiston suunnittelua seuraa luonnollisesti suunnitelmien toteuttaminen kirjoittamalla ohjelmiston lähdekoodi. Ohjelmointivaiheen tuloksena syntyy varsinainen ohjelmisto sekä koodiin liittyvä dokumentaatio. Ohjelmointi itsessään on yksilöllistä toimintaa, eikä ole olemassa yleisiä prosesseja, joita ohjelmoijat yleisesti käyttäisivät. (Sommerville, 2016, s. 58.)

Ohjelmointia ei tarvitse aina aloittaa tyhjältä pöydältä. Useat ohjelmistot muistuttavat toisiaan, jolloin niiden ohjelmoinnissa on mahdollista käyttää aiemmin tehtyjen ohjelmistojen lähdekoodia (Haikala & Mikkonen, 2011, s. 190). Ohjelmistokoodia uudelleenkäyttämällä ohjelmistoa on mahdollista kehittää nopeammin ja halvemmalla. Kaikenlaista ohjelmistokoodia ei kuitenkaan pystytä uudelleenkäyttämään, vaan koodin on oltava suunniteltu uudelleenkäytet-

täväksi. Kun ohjelmistokoodista tehdään uudelleenkäytettävää, luodaan edellisessä alaluvussa mainittu sovelluskehys, jota voidaan käyttää uusien ohjelmistojen kehittämisen apuna. Sovelluskehyksen luominen on kuitenkin kalliimpaa, kuin tavallisen yksittäisen ohjelmiston kehittäminen. Siksi sovelluskehyksen luominen onkin järkevää vain, kun tiedetään sille olevan käyttöä tulevaisuuden ohjelmistoprojekteissa. (Frakes & Kang, 2005.)

Ohjelmistoa ei pystytä koodaamaan kerralla täysin toimivaksi, vaan ohjelmisto vaatii seuraavassa aliluvussa tarkemmin läpikäytävää testausta, jotta ohjelmistosta saadaan tehtyä käyttökelpoinen. Kun testauksella on löydetty virheitä ohjelmistosta, ohjelmoijat jatkavat kehitystyötään korjaamalla havaitut virheet. Virheiden paikantamiseen käytetään debuggausta eli virheenjäljitystä (engl. debugging), jolla tarkoitetaan virheiden jäljittämistä ja korjaamista. (Sommerville, 2016, s. 58.)

2.4 Testaus

Ohjelmointivaiheen jälkeen ohjelmistosta on olemassa versio, joka toimii kutakuinkin halutulla tavalla. Testausvaiheessa ohjelmiston toiminta varmennetaan sille tasolle, että se vastaa asiakkaan odotuksia (Sommerville, 2016, s. 56). Ohjelmistotestauksessa tutkitaan testattavan ohjelmiston laatua ja ominaisuuksia empiirisesti. Ohjelmistotestauksen tavoitteena on täysin virheetön ohjelmisto, joka on kuitenkin mahdotonta saavuttaa, varsinkin jos kyseessä on monimutkainen ohjelmisto. Virheettömyyden sijaan ohjelmistotestauksessa kannattaakin keskittyä ohjelmiston käyttäytymiseen, eli siihen, toimiiko ohjelmisto hyväksyttävästi, kun sitä tarkastellaan esimerkiksi luotettavuuden, ohjelmiston suorituskyvyn, ylläpidettävyyden, tietoturvan ja käytettävyyden kannalta. (Ammann & Offutt, 2016, s. 48.)

Sommerville (2016, s. 59) jakaa testausprosessin kolmeen vaiheeseen: komponenttitestaus (engl. component testing), järjestelmätestaus (engl. system testing) ja asiakastestaus (engl. customer testing). Komponenttitestauksessa ohjelmistokehittäjät testaavat ohjelmiston jokaisen komponentin erikseen, apuna voidaan käyttää automaattisia työkaluja. Kun komponentit on kerätty yhteen ja ne muodostavat ohjelmiston, ohjelmistosta etsitään virheitä järjestelmätestauksen avulla. Järjestelmätestauksessa keskitytään etsimään simuloidun testidatan avulla komponenttien yhteen liittämisestä seuranneita virheitä. Järjestelmätestauksessa kartoitetaan myös, vastaako ohjelmisto toiminnallisilta ja ei-toiminnallisilta vaatimuksiltaan määrittelyvaiheessa sovittuja ehtoja. Asiakastestauksessa ohjelmisto annetaan asiakkaiden tai potentiaalisten asiakkaiden testattavaksi. Ohjelmistossa esiintyvien virheiden lisäksi asiakastestauksessa saatetaan havaita, että ohjelmisto ei tyydytäkään asiakkaisen tarpeita. (Sommerville, 2016, s. 59.)

Ohjelmistojen testaustavat voidaan jakaa myös black box -testaukseen ja white box -testaukseen sen mukaan, onko testaajalla pääsy ohjelmiston lähdekoodiin. Blach box -testauksessa testaaja ei pääse käsiksi lähdekoodiin vaan tar-

kastelee ohjelmistoa kuin ”isoa mustaa laatikkoa”, jonka sisälle testaaja ei pysty näkemään. Testaaja voikin vain syöttää tietoa ohjelmistoon ja vastaanottaa tietoa ohjelmistolta. White box -testauksessa testaajalla on pääsy ohjelmiston lähdekoodiin, jonka avulla testaaja suunnittelee testitapauksia ja suorittaa ohjelmiston metodeja tietyillä parametreillä. Edellisessä kappaleissa mainituissa vaiheista ensimmäisessä, eli komponenttitestauksessa on kyse white box -testauksesta. Järjestelmätestauksessa kyse voi olla kummastakin testaustavasta ja asiakastestauksessa kyse on black box -testauksesta. (Nidhra & Dondeti, 2012.)

Testaus on tehokkain tapa ohjelmiston laadun määrittelyyn ja sen myötä ohjelmiston laadun parantamiseen (Orso & Rothermel, 2014). Ohjelmiston testaamisella voidaankin vähentää ohjelmiston käyttöönottoon liittyviä riskejä, sillä testausvaiheessa havaittavat virheet ovat helpompia korjata, kuin tuotantokäytössä havaitut virheet. Ammann ja Offutt, (2016, s. 38-39) analysoivat suuria valtionhallinnon teettämiä ohjelmistoprojekteja ja havaitsivat, että mitä aikaisemmassa vaiheessa ohjelmistossa olevat virheet havaitaan, sitä halvemmaksi niiden korjaaminen tulee. Ohjelmiston käyttöönoton jälkeen havaitut virheet tulevat noin 50 kertaa kalliimmiksi kuin ohjelmiston aikaisemmassa vaiheessa havaittujen virheiden korjaaminen. (Ammann & Offutt, 2016, s. 39.)

2.5 Käyttöönotto ja ylläpito

Ohjelmiston alustavan julkaisun jälkeen on vuorossa käyttöönotto- ja ylläpitovaihe, jossa kehitetty ohjelmisto otetaan käyttöön ja sen toimintaa ylläpidetään. Ohjelmiston ylläpidolla tarkoitetaan kaikkea sitä työtä, mitä ohjelmiston parissa tehdään julkaisun jälkeen. Julkaisun jälkeisiä töitä ovat esimerkiksi vikojen korjaaminen, suorituskyvyn parantaminen ja ohjelmiston mukauttaminen muuttuneeseen toimintaympäristöön. (Bennett & Rajlich, 2000.)

Käyttöönotto- ja ylläpitovaihe on usein pitkäkestoisin ohjelmistoprojektin viidestä vaiheesta. Varsinkin suurien infrastruktuuriohjelmistojen, kuten lentoliikenteen ohjausjärjestelmien elinikä voi olla yli 30 vuotta. Yritysten ohjelmistojen elinikä on usein yli 10 vuotta, koska uusien järjestelmien kehittäminen ja käyttöönotto on yrityksille kallista. Yritykset myös mukauttavat ohjelmistojaan muuttuvan liiketoiminnan mukana, koska ohjelmistot vastaavat usein yrityksen kriittisten liiketoimintojen onnistumisesta. Suuret yritykset kuluttavatkin enemmän rahaa ohjelmistojensa mukauttamiseen kuin uusien järjestelmien kehittämiseen. (Sommerville, 2016, s. 256.) Jonesin (2006) mukaan vuonna 2006 Yhdysvalloissa 75 prosenttia ohjelmistokehityksen parissa työskentelevistä olivat mukana ohjelmistojen ylläpitoprosessissa eikä Jones uskonut määrän vähenevän tulevaisuudessa. Ohjelmistojen ylläpito työllistää paljon ohjelmistokehittäjiä, koska ohjelmistot ikääntyvät vauhdilla ja suuret massapäivitykset vaativat paljon työtä (Jones, 2006).

Niessink ja Van Vliet (2000) näkevät ohjelmiston ylläpidon palveluna ja ohjelmistokehityksessä syntyneet ohjelmistot tuotteina. Näin ollen asiakkaat arvioivat ohjelmiston ylläpidon laatua eri tavalla, kuin ohjelmistokehitystä. Oh-

jelmiston ylläpidon laatua voidaankin arvioida kahdessa eri laatuulottuvuudessa: toiminnallinen laatu ja tekninen laatu. Toiminnallisella laadulla tarkoitetaan sitä, kuinka palvelu toimitetaan ja teknisellä palvelun tarjoamia tuloksia. Laadukkaan ylläpidon tarjoamiseksi on tarjottava erilaisia ja täydentäviä prosesseja, mitä ohjelmistokehityksessä tarjotaan. Palvelun arvo syntyy sen tarjoamista hyödyistä, jotka ylläpidossa ovat esimerkiksi virheiden korjauksia tai uusia ominaisuuksia toisin kuin ohjelmistokehityksessä syntyneen ohjelmiston tuottama arvo, jossa hyöty ilmenee suoraan ohjelmiston kautta. (Niessink & Van Vliet, 2000.)

Ohjelmiston käyttöönotto- ja ylläpitovaihe voidaan luokitella sen elinkaaren mukaan erilaisiin vaiheisiin, jotka määräytyvät ohjelmistoon kohdistettavien toimien mukaan. Vaiheet ovat varhaislapsuus (engl. infancy), murrosikä (engl. adolescence), aikuisuus (engl. adulthood) ja seniiliys (engl. senility). Tuotteen elinkaaren vaiheiden pituus vaihtelee ohjelmistokohtaisesti. Varhaislapsuus on vaihe, jossa ohjelmiston käyttäjäkunta on pieni ja yleisin ohjelmiston vaatima toimi on virheiden korjaus. Murrosiässä ohjelmiston käyttäjäkunta kasvaa ja virheiden korjaamisen lisäksi ohjelmisto saattaa vaatia uudelleen määrittelyä. Aikuisuusvaiheessa ohjelmisto on melko vapaa vioista ja sen käyttäjäkunta on suurimmillaan. Suuren käyttäjäkunnan takia ohjelmistoon kohdistuu paljon uusia toiminnallisuuteen ja käytettävyyteen kohdistuvia vaatimuksia. Muutosten määrän kasvaessa ohjelmistoa voidaan uudistaa laajemmin ja sen koodirakennetta voidaan parannella. Seniiliysvaiheessa olevia ohjelmistoja kutsutaan myös legacy-ohjelmistoiksi. Tässä vaiheessa ohjelmistoille tyypillistä on vähentyvä käyttäjäkunta, jolloin ohjelmistolle tehdään vain välttämättömät virheiden korjaukset. (Kitchenham ym., 1999.) Ohjelmisto voi olla legacy-vaiheessa useita kymmeniä vuosia. Varsinkin finanssialan yrityksissä on edelleen käytössä 1960-1990-luvuilla COBOL-ohjelmointikielellä tehtyjä ohjelmistoja. Ohjelmistot toimivat vielä hyvin, mutta niiden ylläpitoon on hankala löytää osaavaa henkilökuntaa, koska vanhat COBOL-ammattilaiset ovat eläköityneet eikä uusia enää kouluteta. (Mitchell, 2012.)

3 JOUKKOISTAMINEN

Joukkoistamisen juuret ovat avoimen lähdekoodin kehityksessä, jossa ohjelmistoharrastajien yhteisö halusi luoda parempaa lähdekoodia kuin suuret ohjelmistoyritykset (Howe, 2008). Joukkoistaminen on melko tuore termi ja tässä luvussa esitellään sen erilaisia määritelmiä sekä erilaisia hyödyntämismahdollisuuksia. Luvussa esitellään myös joukkoistamisen erilaisia ilmentymismuotoja.

3.1 Joukkoistamisen määrittely

Joukkoistaminen esiteltiin ensimmäistä kertaa suurelle yleisölle vuonna 2006 Wired-lehdessä julkaistussa artikkelissa (Howe, 2006a). Termiä oli kuitenkin käyttänyt jo aikaisemmin anonyymi käyttäjä keskustelupalstalla internetissä (Schenk & Guittard, 2011). Howe määrittelee joukkoistamisen tavallisesti yrityksen sisällä tehdyn työn ulkoistamiseksi avoimella kutsulla ennalta määrittelemättömälle, yleensä suurelle, joukolle ihmisiä (Howe, 2006b). Vuodesta 2008 joukkoistamista on yritetty määritellä useissa eri tieteellisissä julkaisuissa. Erilaiset määritelmät ovat johtaneet tilanteisiin, jossa joukkoistamiseksi tulkittaan erilaisia ilmiöitä riippuen siitä, mitä määritelmää käytetään. (Brabham, 2013, s. 27.)

Estellés-Arolas ja González-Ladrón-de-Guevara (2012) tekivät joukkoistamisen määrittelystä kirjallisuuskatsauksen, jossa he analysoivat suurimman osan siihen mennessä joukkoistamisesta julkaistut tieteelliset artikkelit. He löysivät 209 dokumentin joukosta 40 erilaista tulkintaa joukkoistamisesta. Tulkintojen systemaattisella analysoinnilla ja validoinnilla avulla Estellés-Arolas ja González-Ladrón-de-Guevara (2012) määrittelivät joukkoistamisen osallistavaksi toiminnaksi internetissä, jossa yksilö tai organisaatio ehdottaa työtehtävää joukolle yksilöitä. Joukon osaaminen, heterogeenisyys ja lukumäärä vaihtelevat. Työtehtävään osallistuminen on vapaaehtoista ja työtehtävän suorittaminen tuottaa molemminpuolista hyötyä. Osallistujan saama hyöty voi olla taloudellista, sosiaalista tunnustusta, itsetuntoa kohottavaa tai omien taitojensa kehit-

tämistä. Työtehtävän joukkoistanut taho säilyttää oikeudet osallistujan tekemään työhön. (Estellés-Arolas & González-Ladrón-de-Guevara, 2012.)

Toisin kuin Howen (2006b) määritelmää käyttämällä, Estellés-Arolasin ja González-Ladrón-de-Guevaran (2012) määritelmää tulkitsemalla esimerkiksi Wikipedia ei ole joukkoistamista. Wikipedian tapauksessa joukkoistajaa (engl. crowdsourcer) ei voida selvästi identifioida eikä joukolle annettavaa korvausta ole selkeästi määritelty, jonka vuoksi Estellés-Arolasin ja González-Ladrón-de-Guevaran (2012) mukaan kyseessä ei ole joukkoistaminen. Merkittävä ero on myös siinä, vaatiiko joukkoistaminen internetiä. Howen (2008) mukaan joukkoistaminen ei vaadi internetin käyttöä, mutta se on tehnyt joukkoistamisesta paljon helpompaa (Howe, 2008, s. 11). Estellés-Arolasin ja González-Ladrón-de-Guevaran (2012) katsovat internetin olevan väline, jolla joukkoistaminen tapahtuu. Joidenkin tutkijoiden mukaan Web 2.0 on teknologinen perusta joukkoistamisen kehittymiselle ja toiminnalle (Vukovic & Bartolini, 2010). Erilaiset määritelmät johtavat tilanteeseen, jossa on vaikea määritellä mitkä erilaiset toimet voidaan lukea joukkoistamiseksi ja mitkä eivät.

Vukovicin ja Bartolinin (2010) mukaan joukkoistamisessa voidaan hyödyntää myös yrityksen omia työntekijöitä. Tällöin joukkoistettava joukko voidaan jaotella sisäiseksi joukoksi, ulkoiseksi joukoksi ja hybridijoukoksi. Sisäinen joukko muodostuu joukkoistavan yrityksen omista työntekijöistä ja ulkoinen joukko koostuu muista kuin joukkoistavan yrityksen työntekijöistä. Hybridijoukko on luonnollisesti sekoitus ulkoisista ja sisäisistä työntekijöistä. Kannustimet ovat usein helpompia asettaa ulkoiselle joukolle, koska sisäisen joukon palkitseminen nähdään usein turhana, koska he saavat muutenkin palkkaa yritykseltä työnantajalleen tekemästään työstä. Sisäinen ryhmä kuitenkin tarvitsee jonkinlaisen palkinnon osallistumisestaan joukkoistamiseen tai muuten heidän motivaationsa työskennellä voi olla heikko. Ulkoisen joukon käyttö joukkoistamisessa saattaa aiheuttaa kuitenkin ongelmia tekijänoikeuksissa ja tietoturvassa. Toisin kuin sisäisellä joukolla, ulkoisella joukolla ei ole työsuhdetta joukkoistavaan yritykseen, jolloin tekijänoikeudelliset seikat on sovittava tarkkaan ennen joukkoistamista. Myös yrityksen tietoturva on otettava huomioon käytettäessä ulkoista joukkoa. On huomioitava, että joitain työtehtäviä ei välttämättä voi joukkoistaa ulkoiselle joukolle, koska tieto voi olla salassa pidettävää tai sen ei muuten haluta joutuvan yrityksen ulkopuolisten tahojen tietoon. (Vukovic & Bartolini, 2010.)

Joukkoistamista hyödyntäviä yrityksiä on maailmalla monia, joista yksi on usein joukkoistamista käsittelevissä lähteissä mainittu Threadless. Threadless on vaatetusalan yritys, joka on perustettu vuonna 2000, eli jo ennen, kuin Howe esitteli termin joukkoistaminen vuonna 2006. Yritys myy painatettuja T-paitoja, joiden suunnittelun se on joukkoistanut omalle virtuaaliyhteisölleen. Kuka tahansa saa liittyä Threadlessin virtuaaliyhteisöön ja suunnitella painatuksia T-paitoihin. Sama virtuaaliyhteisö arvioi joukkoistamalla luotuja T-paitoja ja eniten positiivia arvioita saaneet T-paidat painatetaan ja tuodaan myyntiin Threadlessin verkkosivuille. Painetuiksi päätyneiden T-paitojen suunnittelijat saavat 2000 dollarin palkkion sekä 500 dollarin lahjakortin Threadlessin verkkokaup-

paan. Joukkoistamisen avulla Threadless on onnistunut luomaan tuottoisan ja vähäriskisen liiketoimintamallin, jossa se saa verkkoyhteisöltään tuoteehdotuksia, joille se teettää markkinatutkimuksen, eikä Threadless joudu painattamaan T-paitoja, joille ei olisi jo kysyntää valmiiksi. (Brabham, 2013, s. xix-xx.)

Joukkoistaminen aiheuttaa yrityksille myös haasteita, joista puhutaan tieteellisissä julkaisuissa vähemmän kuin joukkoistamisen hyödyistä ja onnistumisista. Simula (2013) tarkastelee tutkimuksessaan joukkoistamista kriittisestä näkökulmasta. Yleisiä ongelmia joukkoistamisessa ovat, kuinka joukko tehdään tietoiseksi mahdollisuudestaan osallistua joukkoistamiseen, miten houkutella joukko osallistumaan joukkoistamiseen ja kuinka sitouttaa joukko. Joukkoistamisen voima piilee suuressa joukossa ihmisiä eli jos ihmiset eivät ole tietoisia mahdollisuudesta osallistua joukkoistamiseen tai he eivät koe hyötyvänsä osallistumisesta tarpeeksi, ei tarvittavan suurta joukkoa välttämättä saada kasaan, jotta joukkoistaminen onnistuisi. Myös joukkoistamisalustat kilpailevat siitä, kuka saa sitoutettua joukon omaan palveluunsa, koska yksi ihminen voi osallistua vain rajatusti joukkoistamisprojekteihin. Ihmisten alkuinnostuksen lopahtaminen saattaa johtaa työntekijöiden katoon, jos heidän innostustaan ei saada ylläpidettyä pidempään. Voidaan myös väittää, että joukkoistaminen on työntekijöiden hyväksikäyttöä. Kun järjestetään kilpailu, jossa voittajaksi valitaan vain paras vaihtoehto muiden joukosta, muut osallistujat eivät saa tekemästään työstä minkäänlaista korvausta. Tämä on johtanut jo pienimuotoisiin protesteihin joukkoistamista vastaan, joissa lahjakkaita henkilöitä kannustetaan jättämään osallistumatta joukkoistamista hyödyntäviin kilpailuihin. (Simula, 2013.)

3.2 Joukkoistamisen muodot

Howen (2008, s. 280) mukaan joukkoistaminen ei ole yksittäinen strategia vaan sateenvarjotermi, joka kattaa allensa laajan ryhmän lähestymistapoja, jotka kaikki ovat riippuvaisia joukosta. Tutkijat ovatkin jaotelleet joukkoistamista erilaisiin kategorioihin käyttäen erilaisia jaottelutapoja. Brabham (2013, s. 45) jaottelee joukkoistamisen neljään kategoriaan ongelmatyyppien mukaan, joista ensimmäinen soveltuu informaation keräämiseen, organisointiin ja ongelmien raportointiin. Toinen kategoria keskittyy ongelmiin, joihin on empiirisesti todistettavia ratkaisuja ja kolmannessa etsitään ratkaisuja ongelmiin, joissa ratkaisut ovat makuasioita tai markkinatukitoimenpiteitä. Viimeisessä kategoriassa ihmiset tekevät suuren mittaluokan data-analyysiä, koska ovat siinä tehokkaampia kuin tietokoneet. (Brabham, 2013, s. 44.) Kleemann, Voß ja Rieder (2008) taasen jakavat joukkoistamisen viiteen eri kategoriaan: kuluttajien osallistuminen tuotekehittelyyn, kuluttajien osallistuminen tuotteen suunnitteluun, kilpailut tietyn ongelman ratkaisemiseksi, pysyvät avoimet kutsut, yhteisöraportointi, tuotearvioinnit ja asiakkaiden keskinäinen asiakastuki.

Joukkoistaminen voidaan jaotella myös sen perusteella, kuinka joukkoistettu työ toteutetaan. Vukovic ja Bartolini (2010) jakavat joukkoistamisen jouk-

koälyyn ja kilpailuun tai markkinapaikaan. Joukkoälyä hyödynnettäessä, joukko työskentelee yhdessä saavuttaakseen tavoitellun päämäärän. Kilpailussa joukko kilpailee toisiaan vastaan, jolloin joukon tuottamien ratkaisujen joukosta valitaan parhaimmat ja vain he saavat työstään korvauksen. Markkinapaikassa joukkoistettavat työt jaetaan erilaisin kriteerein joukolle ja jokaista työtä suorittaa vain yksi henkilö. Joukkoistaminen voidaan toteuttaa myös edellisiä vaihtoehtoja yhdistelemällä. Joukkoistettava projekti voidaan jakaa pienempiin osiin ja antaa joukkoälyä hyödyntävien ryhmien ratkottavaksi. Kun kaikki projektin osat ovat valmiit, ne yhdistetään, jolloin projekti on valmis. Kaikki ryhmät voivat myös toteuttaa samaa projektia, jolloin he kilpailevat keskenään siitä, mikä ryhmistä tuottaa parhaimman lopputuloksen. (Vukovic & Bartolini, 2010.)

Tässä tutkielmassa esitellään tarkemmin Howen (2008) esittelemä joukkoistamisen kategorisointi, jossa joukkoistamisen jaetaan neljään pääluokkaan, jotka ovat joukkoäly (engl. crowd wisdom), joukkotuotanto (engl. crowd creation), joukkoarviointi (engl. crowd voting) ja joukkorahoitus (engl. crowdfunding).

3.2.1 Joukkoäly

Joukkoälyn periaatteena on, että joukoissa on enemmän älyä ja tietämystä, kuin yksilöissä. Joukkoistamisen tehtävänä on luoda olosuhteet, jossa joukko voi tuoda esille tietämystään. (Howe, 2008, s. 280.) Joukon vahvuus on sen monimuotoisuudessa. Monimuotoinen joukko ongelmanratkaisijoita suoriutuu paremmin kuin yhdenmukaista osaajista koostuva joukko ongelmanratkaisijoita. Mitä suurempi joukko on, sitä monimuotoisempaa sen osaaminen on, joka johtaa parempaan suoriutumiseen ongelmanratkaisutilanteessa. (Hong & Page, 2004.)

Howe (2008, s. 133) jakaa joukkoälyn käyttötavat kolmeen pääkategoriaan. Ensimmäinen on ennustemarkkinat (eng. prediction market), jotka hyödyntävät joukkoälyä tulevaisuuden tapahtumien ennustamiseen, joiden avulla voi ennustaa esimerkiksi vaalien tuloksen tarkemmin kuin perinteisillä kannatuskyselyillä. Toinen kategoria on ongelmanratkaisu (engl. crowdcasting), jossa jokin ongelma välitetään määrittelemättömälle verkostolle potentiaalisia ongelmanratkaisijoita. Kolmantena on niin kutsuttu idea jam, joka on internetin välityksellä toteutettu iso aivoriihitapahtuma, joka kestää ennemminkin viikkoja kuin päiviä. Erona ongelmanratkaisuun, idea jamissa ei keskitytä tiettyyn ongelmaan, vaan tuotetaan kaikenlaisia uusia ideoita. (Howe, 2008, s. 133-134.)

3.2.2 Joukkotuotanto

Howen (2008, s. 281) mukaan joukkotuotannolla tarkoitetaan joukon luovan energian hyödyntämistä sisällön tuottamisessa. Joukkoistaminen on eri asia kuin käyttäjien tuottama sisältö, vaikka käyttäjien tuottamaa sisältöä hyödynnetäänkin usein joukkoistamista hyödyntävässä liiketoiminnassa. Kun suuri joukko tuottaa suuren määrän erilaisia tuotoksia, on niiden diversiteetti suuri. Suu-

rin osa tuotoksista on heikkolaatuisia, mutta joukkoon mahtuu myös hyviä tuotoksia. Joukkoistamisessa tärkeää onkin poimia parhaat tuotokset suuresta joukosta huonoja. (Howe, 2008, s. 177-180.) Tutkielmassa aiemmin esitelty Threadless hyödyntää joukkotuotantoa T-paitojensa suunnittelun joukkoistamisessa. Threadlessin verkkoyhteisö suunnittelee T-paitojen painatuksia, joista osa päätyy yrityksen toteutettavaksi.

3.2.3 Joukkoarviointi

Joukkoarvioinnilla tarkoitetaan suuren tietomäärän organisointia joukkoa hyödyntämällä. Joukon ei tarvitse aina toimia, jotta tietoa voidaan järjestellä, vaan tietoa voidaan kerätä seuraamalla joukon käyttäytymistä (Howe, 2008, s. 281). Joukkoarviointi on hyödyllistä varsinkin sellaisissa palveluissa, joissa on valtava määrä käyttäjien tuottamaa sisältöä. Käyttävät arvioivat sisältöä, jota ovat kuluttaneet ja tuovat näin esiin parhaan palvelussa olevan sisällön valtavasta määrästä heikkolaatuisempaa sisältöä. (Howe, 2008, s. 226.)

Esimerkiksi videopalvelu Youtube käyttää joukkoarviointia videosisältönsä järjestelemiseen. Videoita etsittäessä Youtube järjestelee videoita joukkojen tuottaman datan avulla, jota on muun muassa katsojamäärän sekä käyttäjien antama arvio videosta. (Howe, 2008, s. 225.) Myös aiemmin tutkielmassa esitellyssä Threadlessissa hyödynnetään joukkoarviointia. Threadlessin verkkoyhteisö arvioi suunniteltua T-paitoja ja äänestää niistä parhaimmat toteutettaviksi. Tällöin yritys tietää, että sen valmistamille tuotteille on kysyntää jo ennen tuotteiden valmistusta.

3.2.4 Joukkorahoitus

Joukkoistamisen muodoista joukkorahoitus on ollut eniten esillä viime vuosien aikana. Joukkorahoituksessa suuret ihmisjoukot korvaavat pankit ja muut perinteiset rahoituslaitokset rahoittajina (Howe, 2008, s. 281). Joukkorahoitus madaltaa hierarkioita yhdistämällä ihmiset, joilla on ylimääräistä rahaa niiden kanssa, jotka tarvitsevat rahaa (Howe, 2008, s. 247). Joukkorahoitus voi olla muodoltaan lahjoitus, vaihtokauppa tulevaisuudessa julkaistavaan tuotteeseen tai jonkinlainen tuki tiettyä tarkoitusta varten aloitetulle hankkeelle. Joukkorahoitukseen osallistuneen rahoittajan saama hyöty voi olla rahallista tai muutoin kuin rahana saatava etuus, kuten tunnustus tai äänestysoikeus. (Belleflamme, Lambert & Schwienbacher, 2014.)

4 JOUKKOISTAMINEN OHJELMISTOTUOTANNOSSA

Neljännessä luvussa yhdistetään joukkoistaminen ja ohjelmistotuotanto tarkastelemalla, miten joukkoistamista hyödynnetään ohjelmistotuotannossa. Luvussa esitellään joukkoistetun ohjelmistotuotanto ja tarkastellaan sen etuja ja haasteita. Luvussa käydään myös läpi, miten eri tavoin tieteellisissä artikkeleissa on esitelty joukkoistamisen hyödyntämistä ohjelmistoprojektin viidessä eri vaiheessa: määrittelyssä, suunnittelussa, ohjelmoinnissa, testauksessa ja käytössä ja ylläpidossa. Lisäksi luvussa esitellään yrityksiä, joiden liiketoimintamalli perustuu joukkoistamiselle.

4.1 Joukkoistettu ohjelmistotuotanto

Mao ym. (2017) määrittelevät joukkoistetun ohjelmistotuotannon (engl. crowdsourced software engineering) työtehtäväksi, joka on suunnattu määräämättömälle ihmisjoukolle avoimella kutsulla. Heidän määritelmänsä polveutuu suoraan Howen (2006b) joukkoistamisen määritelmästä. Maon ym. (2017) mukaan koko ohjelmistoa ei tarvitse tehdä joukkoistamalla, jotta sitä voitaisiin kutsua joukkoistetuksi ohjelmistokehitykseksi, vaan joukkoistamista voidaan hyödyntää vain pienemmissä ohjelmistotuotannon osatehtävissä. Joukkoistetun ohjelmistotuotannon avulla voidaan rekrytoida nopeasti työvoimaa suorittamaan erilaisia ohjelmistotuotannon työtehtäviä, kuten vaatimusmäärittelyä, suunnittelua, ohjelmointia ja testausta. (Mao ym., 2017.)

Joukkoistettu ohjelmistotuotanto vaatii tavallisesti kolme toimijaa: työnantaja (engl. requester), alusta (engl. platform) ja työntekijä (engl. worker). Työnantaja on se taho, joka haluaa saada ohjelmistotuotantotyötä tehdyksi ja työntekijä on se toimija, joka osallistuu ohjelmistontuotantoon alustoiden kautta. Alusta on markkinapaikka, jossa työnantajat löytävät työntekijät. Työnantajat ilmoittavat alustalla työtehtäviä, joiden suorittamisessa he tarvitsevat joukkoistettua työvoimaa. Työntekijät valitsevat mieleisensä työtehtävät ja suoritettuaan

työn, he jättävät tekemänsä työn alustalle, joka välittää sen työnantajalle. (Mao ym., 2017.)

Joukkoistamista hyödynnetään laajasti yrityksissä ja organisaatioissa aina sotilaallisista- ja akateemisista organisaatioista isoihin IT-yrityksiin. Muun muassa Yhdysvaltain asevoimien tutkimusorganisaatio DARPA kehitti joukkoistamista hyödyntävän verifiointijärjestelmän ohjelmistojen formaalia verifiointia varten. Verifiointijärjestelmän tavoitteena on tehdä verifioimisesta aloittelijaystävällistä luomalla pelejä, joiden avulla amatöörit voivat auttaa verifiointityökaluja suorittamaan ohjelmistojen verifiointeja (DARPA, 2015). Joukkoistettua ohjelmistotuotantoa hyödyntää myös avaruusjärjestö NASA ja Harvard Business School, jotka järjestivät joukkoistamista hyödyntävän kilpailun, jolla etsittiin parasta ohjelmistokoodia NASAn järjestelmiin (NASA, 2011). Myös ohjelmistoalan yritys Microsoft hyödyntää joukkoistamista Windows 10 -käyttöjärjestelmässä, jossa käyttäjiä hyödynnetään käyttöjärjestelmän virheiden etsimisessä sekä antamalla käyttäjille mahdollisuuden palautteen antamiseen käyttöjärjestelmän kehittämiseksi (Segev, 2014).

Joukkoistettua ohjelmistotuotantoa tarjoavien alustojen määrä on kasvussa. Alustat työllistävät työntekijöitä erilaisilla tavoilla, kuten kilpailuilla, tarjouskaupoilla ja valikoimalla työntekijöitä alustalle rekisteröityneiden työntekijöiden joukosta. Yleensä alustat erikoistuvat tarjoamaan joukkoistamista eri ohjelmistotuotannon työtehtäviin. Joukkoistettua ohjelmistotuotantoa tarjoavia alustoja ovat muun muassa TopCoder, AppStori, uTest ja Amazon Mechanical Turk. TopCoder on perustettu vuonna 2001, ja on näin ollen joukkoistetun ohjelmistotuotannon edelläkävijä. TopCoder tarjoaa ohjelmistotuotannon ratkaisuja laajalla skaalalla, ja joukkoistaminen toteutetaan kilpailujen avulla. AppStori tarjoaa alustan, jossa joukkorahoituksella hankituilla rahoituksella tuetaan kuluttajien ideoiden perusteella kehitettäviä ohjelmistoja. uTest on alusta, jonka avulla voi joukkoistaa ohjelmistotestauksen, joka toteutetaan sekä kilpailuilla että työntekijöitä valikoimalla. Amazon Mechanical Turk -alustan avulla voi joukkoistaa pieniä työtehtäviä, joita toteuttamaan ihmiset ovat tehokkaampia kuin tietokoneet. (Mao ym., 2017.)

Joukkoistettu ohjelmistotuotanto kasvattaa suosiotaan nopeutensa, laadukkaan työjäljen, halvan hinnan ja joustavuuden ansiosta. Joukkoistettua ohjelmistotuotantoa hyödyntävä TopCoder pystyi suorittamaan saman ohjelmistovirheiden etsintäkilpailun kolmessa päivässä, jonka suorittamisessa perinteisellä ohjelmistoyrityksellä kesti 10 päivää. TopCoder -alustalla joukkoistamalla ohjelmoitu koodi on myös melko virheetöntä verrattuna perinteisen ohjelmistoyrityksen tuottamaan lähdekoodiin. TopCoderin tuottaman ohjelmiston hinta on myös joissain tapauksissa noin puolet pienempi, kuin johtavilla ohjelmistoyrityksillä. TopCoderin yritysrakenne on myös joustava, sillä yritys tarvitsee vain vähän vakituisia työntekijöitä, verrattuna perinteisiin ohjelmistoyrityksiin. Vähäinen vakituisten työntekijöiden määrä helpottaa yrityksen rekrytointipaineita yrityksen kasvaessa, eikä työntekijöitä tarvitse irtisanoa työn vähentyessä. (Lakhani, Garvin & Lonstein, 2010.)

LaToza, Towne, Van Der Hoek ja Herbsleb (2013) esittävät joukkoistetun ohjelmistotuotannon eduiksi lyhyemmän markkinoilletuontiajan, monipuolisemman ohjelmistokehittäjäjoukon, ohjelmoijien lisääntyneen tuottavuuden, vähentyneet ohjelmistotuotantokustannukset, ohjelmointityön mielekkyyden lisäämisen ja erilaisten käytäntöjen helpomman kokeilun. Joukkoistetussa ohjelmistotuotannossa käytettävissä oleva ohjelmoijien määrä on usein perinteistä ohjelmistotuotantoa suurempi, jolloin työtehtäviä voidaan suorittaa rinnakkain, jolloin ohjelmisto saadaan usein tuotettua markkinoille nopeammin. Joukkoistamista hyödyntämällä ohjelmistotuotannossa voidaan hyödyntää myös muidenkin kuin ohjelmoijien työpanosta, kuten aiemmin mainitussa DARPAn kehittämässä amatöörien työpanosta hyödyntävässä verifiointityökalussa. Ohjelmoijien tuottavuutta voidaan lisätä kehittämällä parempia kanavia, joissa asiantuntevat ohjelmoijat voivat jakaa osaamistaan aloitteleville ohjelmoijille. Myös ohjelmistojen tuotantokustannuksia on mahdollista alentaa joukkoistamisen avulla. Joukkoistamisella teetetylle työlle ei tarvitse maksaa rahallista korvausta, vaan korvaus voi olla esimerkiksi työntekijän omien taitojen kehittäminen, jolloin kustannukset voivat olla hyvinkin huokeat. Joukkoistettu ohjelmistokehitys mahdollistaa myös työn mielekkyyden ylläpitämisen paremmin kuin perinteinen ohjelmistotuotanto. Kun ohjelmoijat valitsevat heille sopivan haastavia ja mieluisia työtehtäviä, on työn tekeminen mielekkäämpää. Joukkoistamisen laskee myös kynnystä kokeilla uudenlaisia ohjelmistotuotannon tekniikoita, jolloin voidaan löytää entistä tehokkaampia tapoja tehdä ohjelmistoja. (LaToza ym., 2013.)

Joukkoistetulla ohjelmistotuotannolla voidaan ratkaista myös muita, kuin ohjelmistotuotannon ongelmia. Lakhani ym. (2013) esittävät tutkimuksessaan, miten Harvard Medical School hyödynsi TopCoderin joukkoistamisalustaa uusien algoritmien kehittämiseksi DNA:n sekvensointia varten. Työvoimaa hankittiin järjestämällä kilpailu, jossa jaettiin palkintorahoja yhteensä 6000 dollaria. Kukaan kilpailuun osallistuneesta ei ollut bioinformatiikan ammattilainen, mutta silti 30 esitettyä ratkaisua olivat tarkempia ja nopeampia kuin alakohtaiset verrokit. (Lakhani ym., 2013.)

Joukkoistamisen etujen vastapainoksi se aiheuttaa myös monia haasteita. Stol ja Fitzgerald (2014) luettelevat tutkimuksessaan joukkoistamisen kuusi suurinta haastetta, jotka ovat työtehtävien jakaminen, koordinointi ja kommunikointi, suunnittelu ja aikataulutus, laadunvarmistus, osaaminen ja immateriaalioikeudet sekä motivaatio ja palkitseminen. Vaikka työtehtävien jakamisella joukolle saatetaankin saavuttaa tehokkuuden paranemista työskentelyyn, saattaa sillä olla myös päinvastainen vaikutus. Työtehtävien jakaminen liian pieniksi osiksi saattaa johtaa päällekkäisten töiden tekoon. Myös työtehtävien huono koordinointi saattaa aiheuttaa päällekkäisyyksiä työtehtävissä ja lisäksi kasvanut työntekijöiden määrä aiheuttaa usein kommunikaatio-ongelmia ohjelmistotuotantoprojekteissa. Ongelma saattaa esiintyä myös joukkoistamisessa, koska myös joukkoistetun työvoiman kanssa on yleensä kommunikoitava. Joukkoistaminen voi aiheuttaa ongelmia myös suunnittelun ja aikataulutuksen kannalta. Kun työvaiheita ulkoistetaan ennestään tuntemattomille työntekijöille työn

lopputulos tai aikataulussa pysyminen on epävarmempaa, kuin vakituisilla työntekijöillä teetettynä. Jos joukkoistetulta työvoimalta vaaditaan jonkinlaista ammattitaitoa, ei voida myöskään olla aina varmoja, onko kyseistä ammattitaitoa tarjoavia työntekijöitä saatavilla juuri silloin kuin kyseinen työtehtävä on suoritettava. Laatu on yksi ohjelmistojen tärkeimmistä ominaisuuksista. Joukkoistetun työtehtävän laatu riippuu työn tekijästä ja saattaa vaihdella suuresti. Työn laatu riippuukin usein joukkoistamistavasta ja siitä kuinka paljon työntekijöitä on saatavilla. Jos koko ohjelmisto teetetään joukkoistamalla, ei yrityksellä itsellään ole vahvaa osaamista ohjelmiston toiminnasta. Tämä saattaa johtaa ongelmiin ohjelmiston ylläpidon ja jatkokehityksen kannalta. Joukkoistettaessa työtehtäviä joudutaan usein antamaan työntekijälle käyttöön yrityksen ei-julkisia tietoja tai pääsy yrityksen järjestelmiin. Tämä saattaa johtaa tietojen vuotamiseen ja tietoturvaongelmiin. Jotta joukkoistaminen onnistuu, on työntekijöiden motivaation oltava kohdillaan. Yleensä motivaattorina toimii palkkio tehdystä työstä. Jos palkkio on kuitenkin liian suuri voi työ jäädä tekemättä. (Stol & Fritzgerald, 2014.)

Maon ym. (2017) toteuttamassa kirjallisuuskatsauksessa tutkittiin joukkoistetusta ohjelmistotuotantoa käsitteleviä artikkeleita. Niistä artikkeleista, jotka käsittelivät joukkoistamisen soveltamista ohjelmistotuotannon eri vaiheissa, noin puolet käsitteli ohjelmointi- ja testausvaihetta. Käyttöönotto- ja ylläpitovaihetta sekä määrittelyvaihetta käsitteli noin neljännes artikkeleista. Tässä tutkielmassa esitellyistä ohjelmistotuotannon vaiheista vähiten artikkeleita oli julkaistu suunnittelusta (Mao ym., 2017.) Seuraavaksi tutkielmassa käydään läpi, miten ohjelmistotuotannon eri vaiheissa voidaan hyödyntää joukkoistamista ohjelmistotuotannon tukena.

4.2 Joukkoistaminen ohjelmistotuotannon eri vaiheissa

4.2.1 Määrittely

Määrittelyvaiheessa joukkoistamisen käytön motiivina ovat kustannussäästöt, yritysten ulkopuolisten henkilöiden aihealueen tuntemuksen hyödyntäminen, käyttäjien tarpeiden selvittäminen, automaatio ja ohjelmiston parempi laatu. Joukkoistetun määrittelyn tueksi on luotu erilaisia työkaluja, kuten StakeSource ja StakeRare, joiden avulla saadaan esille, mitä ominaisuuksia käyttäjät ohjelmistolta haluavat. (Mao ym., 2017.)

Lim ja Finkelstein (2012) kehittivät StakeRare-työkalun, jonka avulla pystytään tunnistamaan ja priorisoimaan ohjelmiston vaatimuksia sosiaalisella verkostoanalyysillä ja yhteistoiminnallisen suodattamisen avulla. StakeRare on tarkoitettu suurille ohjelmistoprojekteille, joita vaivaavat liian suuri tietomäärä, riittämätön sidosryhmien panos ja puolueellinen vaatimusten priorisointi (Lim & Finkelstein, 2012). StakeSource on työkalu, jonka avulla vaatimusmäärittelyyn otetaan mukaan kaikki sidosryhmät eikä vain pientä joukkoa ammattilai-

sia. Näin saadaan vähennettyä asiantuntijoiden työtaakkaa ja sidosryhmien vaatimukset otetaan todennäköisemmin huomioon. (Lim, Quercia & Finkelstein, 2010.) StakeSource2.0 on kehittyneempi versio StakeSource-työkalusta ja se on julkisesti saatavilla internetissä (StakeSource, 2019). Lim, Damian, Ishikawa ja Finkelstein (2013) esittelivät tutkimuksessaan kymmenen määrittelyprojektia, joissa on hyödynnetty StakeSource-työkalua, joka todettiin tehokkaaksi, jos sidosryhmät ovat tarpeeksi sitoutuneita projektiin.

Adepetu, Khaja, Al Abd, Al Zaadi ja Svetinovic (2012) esittävät toteutettavaksi CrowdREquire-alustan, jonka avulla vaatimusmäärittelyn vaiheet ovat oikeissa olosuhteissa toteutettavissa joukkoistamalla. Vaatimusmäärittely on työläs prosessi, jossa vaaditaan ammattimaista työvoimaa, jota ei kuitenkaan aina ole saatavilla. CrowdREquire-alustan avulla vaatimusmäärittely voitaisiin teettää joukolla, jolla on monimuotoinen ammattitaito, osaaminen ja kokemus. CrowdREquire toimisi markkinapaikkana, jossa rekisteröityneet käyttäjät kilpailevat siitä, kenen vaatimusmääritelmän projektin joukkoistaja valitsee. CrowdREquire ei tarjoa pelkästään markkinapaikkaa, jonka välityksellä työnantaja ja työntekijä ovat yhteydessä, vaan se tarjoaa myös rakenteen, joka ohjaa sekä työnantajaa, että työntekijöitä haluttuun suuntaa. Vaatimusmäärittelyssä onkin tärkeää, että joukkoistettavan projektin sidosryhmät osaavat kuvailla ongelmansa selkeästi ja työntekijöitä ohjataan suorittamaan haluttuja tehtäviä. Alustan toimivuuden kannalta tärkeintä olisikin saada mukaan mahdollisimman paljon rekisteröityneitä työntekijöitä, joilla on perusosaaminen vaatimusmäärittelystä sekä työnantajia, jotka haluavat joukkoistaa projektejaan.

Limin ja Flinkelsteinin (2012) kehittämästä StakeSource-työkalusta ei löydy viime aikoina julkaistuja tieteellisiä artikkeleita eikä sen verkkosivuilla mainita StakeSource-työkalulla viime aikoina toteutetuista projekteista. Myöskään Adepetun ym. (2012) toteutettavaksi ehdottamaa CrowdREquire-alustaa ei ole toteutettu. Vaikka joukkoistaminen on todettu käyttökelpoiseksi vaatimusmäärittelyn apuna, näyttää sen käyttö olevan kuitenkin melko vähäistä.

4.2.2 Suunnittelu

Monet kaupalliset joukkoistamisalustat, kuten 99designs, DesignCrowd ja crowdSping, tarjoavat joukkoistettua käyttöliittymäsuunnittelua (Mao ym., 2017). Ohjelmistoprojektin suunnitteluvaiheessa on kuitenkin kyse ohjelmiston teknisestä suunnittelusta eikä käyttöliittymän suunnittelusta. Joukkoistamisen hyödyntämisestä ohjelmiston suunnitteluvaiheessa onkin julkaistu vain muutamia tutkimusartikkeleita.

Lasecki ym. (2015) esittävät toteutettavaksi Apparition-järjestelmän, joka tarjoaa työkaluja ja tekniikoita interaktiivisten järjestelmien prototyyppien luomiseksi. Apparition hyödyntää joukkoistamista vaikeasti automatisoitavissa olevien toimintojen suorittamiseen, jonka avulla Apparition pystyy luomaan prototyyppejä, jotka vastaavat yli 90 prosenttisesti käyttäjien tarpeita. Aikaa prototyyppien luomiseen kuluu vain muutamia sekunteja, jonka ansiosta suun-

nittelijat pystyvät luomaan ja muokkaamaan prototyyppejä suunnittelutilaisuuden aikana. (Lasecki ym., 2015.)

Vaikka arkkitehtuurisuunnittelu on ohjelmistosuunnittelun tärkein työkalu, vain harva joukkoistamisalusta tukee arkkitehtuurisuunnittelun joukkoistamista. LaToza, Chen, Jiang, Zhao ja Van Der Hoek (2015) toteuttivat tutkimuksessaan arkkitehtuurisuunnittelun joukkoistamalla. Joukkoistaminen toteutettiin kilpailuna, mutta työntekijöille annettiin mahdollisuus antaa ja vastaanottaa palautetta omasta ja kilpailevista arkkitehtuurisuunnitelmista ja sen jälkeen parannella omaa arkkitehtuurisuunnitelmaansa. Mahdollisuudella tutustua kilpaileviin arkkitehtuurisuunnitelmiin paransi arkkitehtuurisuunnitelmien laatua merkittävästi. Tutkimus osoittaa, että joukkoistamisesta on mahdollisesti hyötyä arkkitehtuurisuunnittelussa, koska silloin suunnitelmaa arvioivat ulkopuoliset tahot, jolloin todennäköisyys tuottaa laadukas suunnitelma paranee. Myös kilpailevia arkkitehtuurisuunnitelmia voidaan yhdistellä, jolloin saavutetaan parempi lopputulos. (LaToza ym., 2015.) Myös Nebeling, Leone ja Norrie (2012) ehdottivat joukon hyödyntämistä web-sovelluksia suunniteltaessa tarjoamalla helppokäyttöisiä rajapintoja, joiden avulla suunnitteluun voivat osallistua myös vähemmän teknologiasta ymmärtävät ihmiset.

Laseckin ym. (2015) suunnittelemasta Apparition-järjestelmästä ei löydy viime aikoina julkaistuja tieteellisiä artikkeleita, eikä järjestelmää tiettävästi ole toteutettu kenenkään toimesta. Vaikka joukkoistamisesta on todettu olevan hyötyä ohjelmistoprojektin suunnitteluvaiheessa, sen käyttö vaikuttaa olevan hyvin vähäistä ja saatavilla olevan tieteellisen tutkimuksen rajallista.

4.2.3 Ohjelmointi

Mao ym. (2017) jakavat joukkoistamisen käytön ohjelmointivaiheessa kolmeen alakategoriaan: joukko-ohjelmointialustat, ohjelmiston optimointi ja ohjelmointiympäristön kehittäminen. Tavallisen ohjelmointityön lisäksi joukko-ohjelmointi ja ohjelmointiympäristön parannus -kategorioissa hyödynnetään aikaisemmin joukkoistamalla kerättyä tietoa ohjelmointityön tukemiseksi. Kaupallisista joukkoistamisalustoista ohjelmointivaiheen parissa työskentelevät muun muassa TopCoder, GetACoder, AppStori ja Bountify. TopCoder, GetACoder ja AppStori tarjoavat kokonaisvaltaista ohjelmistokehitystä, jonka lopputuloksena on käyttövalmis ohjelmisto. Bountify toimii markkinapaikkana pienille ohjelmointitöille, jossa käyttäjä voi teetättää työn joukolla tarjoamalla 1-100 dollarin palkkion parhaalle ratkaisulle. StackOverflow on kysymyksiin ja vastauksiin perustuva verkkosivu, jossa ohjelmoijat voivat kysyä apua ongelmiinsa ja auttaa muita avun tarvitsijoita vastaamalla heidän kysymyksiinsä. (Mao ym., 2017.)

Ohjelmointialustojen tehtävänä on hallinnoida ja koordinoida joukkoistettuja työntekijöitä, jotta työtehtävien suorittaminen onnistuisi tehokkaammin. Tieteellisissä artikkeleissa on esitelty malleja joukkoistamista hyödyntävien ohjelmointialustojen rakentamiseksi, joita ovat muun muassa Collabode ja CIDRE. (Mao ym., 2017.) Goldman (2011) toteutti väitöskirjaansa varten Collabode-

ohjelmointiympäristön, jonka tavoitteena oli tehdä ohjelmoijien yhteistyössä toimimisesta tehokkaampaa. Collaboden avulla ohjelmoija voi esimerkiksi joukkoistaa pieniä ohjelmointitehtäviä työtovereilleen, jolloin ohjelmoijan on mahdollista työskennellä flow-tilassa pidempään ohjelmistokoodin kriittisen osan parissa. (Goldman, 2011.) CIDRE (Cloud-based Integrated Development and Runtime Environment) on Ballin ym. (2015) kehittämä pilvipohjainen ohjelmointiympäristö, joka koostuu joukko-ohjelmointiyhteisöstä, internetissä toimivasta ohjelmointialustasta ja sovelluskaupasta, joiden avulla ohjelmistoympäristön suunnittelijat, ohjelmoijat ja käyttäjät voivat antaa toisilleen palautetta kehitysprosessin aikana.

Joukkoistamista voidaan käyttää myös ohjelmistojen optimointiin (Auler, Borin, de Halleux, Moskal & Tillmann, 2014) ja puoliautomaattiseen ohjelmien luomisen (Cochran, D'Antoni, Livshits, Molnar & Veanes 2015). Auler ym. (2014) kehittivät joukkoistamista hyödyntävän lähestymistavan JavaScript-komentokielellä toteutettujen ohjelmien optimoimiseksi. JavaScript-komentokielellä toteutetut ohjelmistot toimivat usein monenlaisessa eri ympäristössä, kuten eri verkkoselaimissa. Optimoinnilla varmistetaan, että JavaScript-ohjelmistot toimivat tehokkaasti monissa eri ympäristöissä. Joukkoistamisen avulla kerättiin pilveen tietoa yleisistä tehokkuutta rajoittavia pullonkauloja, joiden avulla JavaScript-ohjelmistot saatiin optimoitua paremmiksi. (Auler ym., 2014.) Cochran ym. (2015) toteuttivat CROWDBOOST-työkalun, jonka ohjelmointityötä voi tehostaa joukkoistamalla hankalia ohjelmointitöitä, kuten URL-osoitteiden vahvistaminen, kokeneille ja kokemattomille ohjelmoijille. CROWDBOOST-työkalun hyödyntämän joukkoälyn avulla saavutettiin yli 16 prosenttia virheettömämpi lopputulos verrattuna perinteiseen ohjelmointityöhön. (Auler ym., 2014.)

Ohjelmistoympäristöjen parantamista joukkoistamisen avulla on tutkittu laajasti vuodesta 2010 lähtien. Erilaisia työkaluja ja menetelmiä on ehdotettu auttamaan ohjelmoijia ohjelmointityössä ja virheenjäljityksessä. (Mao ym., 2017.) Monet aloittelevat ja kokeneemmatkin ohjelmoijat käyttävät internetin keskustelufoorumeja ja verkkosivustoja, kuten StackOverflow-verkkosivustoa, ohjelmoinnissa kohtaamiensa ongelmien ratkaisemiseen. Hartmann, MacDougall, Brandt ja Klemmer (2010) esittelivät HelpMeOut-järjestelmän, joka mahdollistaa joukkoistamalla kerättyjen ongelmanratkaisujen esittelemisen ohjelmoijalle suoraan ohjelmointiympäristössä. HelpMeOut-järjestelmällä on tietokanta, johon on tallennettu joukkoistamalla kerättyjä korjauksia virheellisen ohjelmistokoodin korjaamiseksi. Järjestelmä auttaa virheenjäljityksessä havaitsemalla automaattisesti virheellisen koodin ja ehdottamalla siihen parannusta tietokannasta löytämänsä ratkaisuehdotuksen perusteella. Järjestelmää testattiin aloittelevilla ohjelmoijilla ja sen huomattiin tuottavan hyödyllisiä korjausehdotuksia 47 prosentissa virhetapauksista. (Hartmann ym., 2010.) Ponzanellin, Bacchellin ja Lanzan (2013) tekemä Seahawk on HelpMeOut-järjestelmän tyylinen ohjelmoijia avustava Eclipse-ohjelmointiympäristön liitännäinen, joka auttaa virhetilanteiden selvittämisessä sekä dokumentoinnissa. Seahawk hyödyntää StackOverflow-verkkosivuston joukkoistamalla hankittuja ratkaisuja ohjelmoijien ongel-

matilanteiden selvittämiseksi ja näin ollen ohjelmoija saa tarvittavaa apua suoraan ohjelmointiympäristössä (Ponzanelli ym., 2013).

Joukkoistamisen hyödyntämisestä ohjelmointivaiheessa on julkaistu monia tieteellisiä artikkeleita ja markkinoilla on monia kaupallisia alustoja, jotka tarjoavat markkinapaikan työtehtävien joukkoistamiselle tai tukevat muuten ohjelmointityötä joukkoistamisen avulla. Ohjelmointivaihe onkin yksi eniten joukkoistamista hyödyntävä ohjelmistoprojektin vaihe.

4.2.4 Testaus

Joukkoistetun ohjelmistotuotannon tutkijat ovat tutkineet runsaasti joukkoistamisen hyödyntämistä ohjelmistojen testauksessa. Joukkoistamisen hyödyntämistä ohjelmiston testauksessa kutsutaan usein joukkoistetuksi testaukseksi (engl. crowdsourced testing). Joukkoistetun testauksen etuna perinteiseen testaukseen on käyttäjien osallistaminen tuotteen testaukseen ammattilaisten lisäksi. Joukkoistettua testausta on sovellettu monessa eri tyyppisessä testauksessa, kuten käytettävyystestauksessa, suorituskykytestauksessa, käyttöliittymätestauksessa ja testitapausten luomisessa. Markkinoilla toimii useita joukkoistettua testausta tarjoavaa yritystä, kuten uTest, Passbrains, 99Tests ja Pay4Bugs. Yritysten joukkoistamista tarjoavat alustat toimivat pääsääntöisesti markkinapaikkoina testauksessa tarvittavien työntekijöiden löytämiseksi. (Mao ym., 2017.)

Käytettävyystestauksella on tärkeä rooli verkkosivun menestymisen kannalta, mutta se on usein kallista ja vaatii paljon työvoimaa (Liu, Bias, Lease & Kuipers, 2012). Liu ym. (2012) totesivat tutkimuksessaan käytettävyystestauksen joukkoistamisen olevan nopeampaa, halvempaa ja helpompaa kuin laboratoriossa toteutettu käytettävyystestaus, mutta käytettävyystestauksen laatu ei ole joukkoistamalla tyypillisesti yhtä hyvä kuin laboratoriossa toteutetulla käytettävyystestauksella. Joukkoistamalla saatu palaute verkkosivuston käytettävyydestä oli lyhyempää ja vähemmän hyödyllistä kuin laboratoriossa toteutetussa testauksessa saadusta (Liu ym., 2012). Schneider ja Cheung (2011) totesivat kuitenkin joukkoistetun käytettävyystestauksen havaitsevan käytettävyysongelmia yhtä hyvin kuin ammattilaisten suorittama käytettävyystestaus.

Ohjelmiston suorituskyvyn mittaus tosielämän tilanteissa on hankalaa, varsinkin jos laitteisto on riippuvainen ihmisestä tai verkkoyhteydestä. Suorituskykytestauksen tarkoituksena on varmistaa, että ohjelmisto toimii tehokkaasti kaikissa tilanteissa ja laitteissa. Perinteisesti suorituskykytestauksessa on yritetty parantaa menetelmiä, joiden avulla on yritetty jäljitellä tosielämän tilanteita. (Musson ym., 2013.) Musson ym. (2013) toteuttivat suorituskykytestauksen Lync-ohjelmistolle joukkoistamalla, johon osallistui 48 000 käyttäjää. Lync on Microsoftin isoille yrityksille suunnattu viestintäjärjestelmä, jonka tukee kommunikointia pikaviestien lähettämisestä suuriin videokonferensseihin. Joukkoistaminen toteutettiin muuntamalla Lync itsessään joukkoistamisalustaksi. Käyttäjien ei tarvinnut tehdä muuta kuin vastata avoimeen pyyntöön, eli siihen, saako heidän tuottamaansa dataa käyttää sovelluksen kehittämiseen.

Käyttäjät eivät siis tehneet mitään erityisiä työtehtäviä vaan antavat pelkästään heidän tuottamansa datan ohjelmiston kehittäjän käytettäväksi. (Musson ym., 2013.)

Joukkoistamista voidaan hyödyntää myös käyttöliittymien testauksessa. Käyttöliittymätestaus on vaikeaa, koska automaattisten testien luominen ja ylläpito on haastavaa ja manuaalisten testien teko vie paljon aikaa, on kallista ja se on hankalaa toteuttaa jatkuvassa testausprosessissa (Dolstra, Vliegendhart, Pouwelse, 2013). Dolstra ym. (2013) esittävät tutkimuksessaan, että käyttöliittymätestaus on mahdollista joukkoistaa asentamalla järjestelmä virtuaalikoneille ja päästämällä testaajat niihin käsiksi verkkoselaimiensa kautta. Näin saadaan toteutettua osittain jatkuva käyttöliittymätestausprosessi, joka on kustannuksiltaan maltillinen ja siihen osallistuu suuri joukko testaajia. Virheellisten tulosten määrä oli kuitenkin joukkoistamalla suurempi kuin perinteisessä käyttöliittymätestauksessa. (Dolstra ym., 2013.)

Chen ja Kim (2012) esittelivät tutkimuksessaan menetelmän, jolla voidaan parantaa automaattista testitapausten luomista, teettämällä ihmisillä tietokoneelle vaikeita tehtäviä. Tietokoneelle vaikeat tehtävät muunnettiin palapeleiksi, joita kokoamalla ihmiset voivat parantaa automaattisesti tietokoneella luotujen testitapausten kattavuutta. Testitapausten kattavuutta saadaan siis parannettua joukkoistamalla palapelejä kokoamista joukolle ihmisiä. Tutkimustulokset kahdessa avoimen lähdekoodin projekteissa osoittivat testien kattavuuden parantuneen 7 prosenttia ja 5,8 prosenttia verrattuna ilman ihmisten apua luotuihin testitapauksiin, joka ei ole vähäpätöinen parannus. (Chen & Kim, 2012.)

Joukkoistamisen hyödyntäminen ohjelmistojen testauksessa houkuttelee ohjelmistokehittäjiä, koska se vähentää yrityksen omaa työvoiman tarvetta ja nopeuttaa testausprosessia. Joukkoistettua ohjelmistotestausta on käsitelty laajasti joukkoistetun ohjelmistotutkimusta käsittelevissä tieteellisissä artikkeleissa. Myös markkinoilla on lukuisia yrityksiä, jotka tarjoavat joukkoistettua testausta asiakkailleen. Voidaankin todeta, että joukkoistamisen hyödyntäminen ohjelmistotestauksessa on ohjelmistoprojektin vaiheista yksi suosituimmista.

4.2.5 Käyttöönotto ja ylläpito

Käyttöönotto- ja ylläpitovaihe on usein ohjelmistoprojektin pitkäkestoisin vaihe ja se kattaa kaiken työn, jota ohjelmiston parissa tehdään sen julkaisun jälkeen. Useimmiten näitä töitä ovat ohjelmiston vikojen korjaus, suorituskyvyn parantaminen ja ohjelmiston mukauttaminen muuttuneeseen toimintaympäristöön. Käyttöönotto ja ylläpito on yksi ensimmäisistä ohjelmistoprojektin vaiheista, jotka ovat hyötyneet joukkoistamisen tuomista hyödyistä (Mao ym., 2017).

Ohjelmistot ovat nykypäivänä usein niin suuria ja monimutkaisia, ettei niiden toimivuutta pystytä täysin varmentamaan formaalein tai automaattisin tekniikoin (Bacon, Chen, Parkes & Rao, 2009). Bacon ym. (2009) ehdottavat markkinamekanismin hyödyntämistä valitessa niitä virheitä, joita ohjelmistosta kannattaa korjata. Tavoitteena ei siis ole virhevapaa ohjelmisto, vaan ohjelmistosta korjattaisiin vain ne virheet, jotka häiritsevät käyttäjiä ja joiden korjaami-

nen on taloudellisesti kannattavaa. Joukkoistamista hyödynnettäisiin sen tiedon hankkimisessa, mitkä ohjelmiston virheet ovat käyttäjien mielestä häiritsevimpiä tai mitä uusia ominaisuuksia käyttäjät ohjelmistolta haluaisivat. Joukkoistamiseen osallistuneet käyttäjät palkittaisiin osallistumisestaan ohjelmiston virheiden raportoinnista tai uusien ominaisuuksien ideoimisesta. (Bacon ym., 2009.)

Joukkoistamisen avulla ohjelmiston käyttäjiä voi myös hyödyntää ohjelmiston tarkkailussa. Ali ym. (2011) ehdottivat tutkimuksessaan Social Sensing -tekniikan, jonka tarkoituksena on hyödyntää käyttäjien havaintokykyä osana ohjelmistoa. Social Sensing -tekniikassa käyttäjät toimivat ohjelmiston valvojina lisäten ohjelmiston suunnittelijoiden kykyä hahmottaa ohjelmiston laatua ja validiteettia sekä ohjelmiston mahdollisesti tarvitsemia muutostarpeita. Social Sensing -tekniikkaa voidaan käyttää esimerkiksi liikenneruuhkia valvovaan ohjelmistoon. Ohjelmiston käyttäjät valvovat, onko ohjelmiston tarjoama ajankohtainen tieto liikeruuhkista oikeaa, vai antaako ohjelmisto väärää tietoa käyttäjille. Näin joukkoistamalla ohjelmiston ylläpito on helpompaa ja ohjelmisto toimii luotettavammin. (Ali ym., 2011.)

Suuret ja monimutkaiset ohjelmistot tuottavat käyttövaikeuksia myös käyttäjille, jotka joutuvat tekemään monia valintoja, jotta ohjelmisto vastaisi heidän tarpeitaan. Hamidi, Andritsos ja Liaskos (2014) esittelivät MDP-metodin (Markov Decision Process), jonka avulla on mahdollista konfiguroida ohjelmiston asetukset joukolta kerätyn datan avulla. Kun laajalta joukolta on kerätty tarpeeksi tietoa siitä, miten he konfiguroivat oman ohjelmistonsa voidaan heidän tuottamansa datan perusteella ehdottaa uusille käyttäjille valintoja, joihin he luultavasti päätyisivät oman pohdintansakin perusteella. MDP-metodi on käytössä Facebookin yksityisuusasetuksissa ja tutkimuksen mukaan sen avulla pystyttiin vähentämään käyttäjien konfiguraatiovaiheita 27,7 prosentilla ja metodi ennusti 75 prosenttisesti oikein käyttäjien valinnat. (Hamidi ym., 2014.)

Ohjelmistoprojektin käyttöönotto- ja ylläpitovaiheessa joukkona toimiikin yleensä ohjelmiston käyttäjäkunta, jonka tuottaman datan avulla yritetään parantaa ohjelmiston toimintaa. Joukkoistamisen hyödyntämistä käyttöönotto- ja ylläpitovaiheessa on tutkittu kohtalaisesti ja sitä on myös onnistuttu hyödyntämään käytännön toteutuksissa onnistuneesti.

5 YHTEENVETO

Tässä kandidaatintutkielmassa tarkasteltiin kirjallisuuskatsauksen keinoin ohjelmistotuotantoa ja joukkoistamista sekä sitä, miten joukkoistamista voidaan hyödyntää ohjelmistotuotannon tukena. Tavoitteena on ollut määritellä ohjelmistotuotanto ja esitellä ohjelmistoprojektin viisi vaihetta: määrittely, suunnittelu, ohjelmointi, testaus sekä käyttöönotto ja ylläpito. Lisäksi esiteltiin ja määriteltiin joukkoistaminen ilmiönä. Tutkimus pyrki vastaamaan tutkimuskysymykseen, *Miten joukkoistamista voidaan hyödyntää ohjelmistotuotannossa?* Tutkielmassa keskityttiin tarkastelemaan joukkoistamisen mahdollisuuksia ohjelmistotuotannossa keskittyen ohjelmistoprojektin viiteen eri vaiheeseen.

Yleisimmin tieteellisissä artikkeleissa käytetty Jeff Howen määritelmä joukkoistamisesta on hyvin laaja ja sitä käyttämällä joukkoistamiseksi voidaan lukea hyvin monenlainen toiminta. Samaa määritelmää käytetään myös joukkoistetussa ohjelmistotuotannossa, jolla tarkoitetaan joukkoistamisen hyödyntämistä ohjelmistotuotannossa. Joukkoistamiselle on esitetty myös muita määritelmiä, joista osa on poikkeavia esimerkiksi internetin osuudesta joukkoistamisessa. Howen määritelmän käyttö johtaa tilanteeseen, jossa monenlainen joukon hyödyntäminen ohjelmistotuotannon tukena voidaan lukea joukkoistamiseksi, vaikka muiden määritelmien mukaan kyseessä ei olisikaan joukkoistaminen. Erilaiset joukkoistamisen määritelmän johtavat joukkoistamisen tutkimisen vaikeutumiseen, koska eri tutkimukset voivat lukea eri keinot joukkoistetuksi ohjelmistotuotannoksi. Tässä tutkielmassa ei ole erikseen arvioitu, onko käytetyssä lähdemateriaalissa olevat joukkoistamiseksi kutsutut ongelmanratkaisutai tuotantomallit joukkoistamisen määritelmän mukaisia, vaan tutkielmassa on luotettu tieteellisen julkaisun omaan arvioon. Joukkoistettu ohjelmistotuotanto on kasvattanut suosiotaan sekä tutkimusaiheena että käytännön sovelluksina myös suurissa yrityksissä ja organisaatioissa. Markkinoilla on monia liiketoimintamallissaan joukkoistamista hyödyntäviä yrityksiä ja niiden määrä on ollut viime vuosina kasvussa.

Ohjelmistoprojektin eri vaiheissa on mahdollista hyödyntää vaihtelevasti joukkoistamista. Ohjelmiston määrittelyn avuksi on luotu tai ehdotettu toteutettavaksi joukkoistamista hyödyntäviä työkaluja, kuten StakeSource, StakeRare ja

crowdREquire, jotka keskittyvät vaatimusmäärittelyn joukkoistamiseen. Näiden työkalujen avulla vaatimusmäärittelyprosessissa voidaan hyödyntää yrityksen omaa henkilökuntaa tai yrityksen ulkopuolista työvoimaa joukkoistamalla. Joukkoistaminen hyödyntämistä ohjelmistoprojektin määrittelyvaiheessa käsittelevää tutkimusta on kuitenkin julkaistu melko vähän ja joukkoistamista näytetään hyödyntävän rajallisesti yritysten vaatimusmäärittelyprosesseissa. Myöskään joukkoistamisen hyödyntämisestä ohjelmistoprojektin suunnitteluvaiheessa ei ole julkaistu paljoakaan tutkimusta. Tutkimusten perusteella joukkoistamisella on saavutettu tehokkuuden paranemista suunnitteluvaiheessa interaktiivisten prototyyppien luomisessa ja ohjelmistojen arkkitehtuurisuunnittelussa. Kaupalliset toteutukset ohjelmistoprojektin joukkoistamista hyödyntävistä suunnitteluvaiheen toteutuksista ovat kuitenkin vähäisiä.

Eniten joukkoistamista tutkimustietoa ja käytännön toteutuksia on ohjelmistoprojektin ohjelmointivaiheesta ja testausvaiheesta. Ohjelmointivaiheessa joukkoistamista voi hyödyntää erityisesti pienten työtehtävien ohjelmoinnissa, ohjelmistojen optimoinnissa ja ohjelmointiympäristöjen kehittämisessä. Kun osa työtehtävistä tuotetaan joukkoistamalla, voidaan ohjelmointityötä teettää enemmän rinnakkain, jolloin ohjelmointi on nopeampaa ja tehokkaampaa, jolloin tuotteen markkinoilletuontiaika lyhenee. Ohjelmiston optimoinnissa on hyötyä laajasta joukosta, joka käyttää ohjelmistoa erilaisissa ympäristöissä, jolloin ohjelmistosta saadaan kehitettyä mahdollisimman tehokkaasti eri ympäristöissä toimiva. Ohjelmointityö tapahtuu ohjelmointiympäristöissä, joihin voidaan lisätä joukkoistamista hyödyntäviä ominaisuuksia, jotka auttavat ohjelmoijia työssään. Joukkoistamisen hyödyntäminen testausvaiheessa mahdollistaa ohjelmiston käyttäjien tuomisen mukaan testaukseen, jolloin ohjelmistosta saadaan helpommin käyttäjien tarpeita vastaava. Joukkoistetusta testauksesta löytyy paljon tutkimusta ja markkinoilla on myös monia sitä tarjoavia yrityksiä. Ohjelmistoprojektin pitkäkestoisimmassa käyttöönotto- ja ylläpitovaiheessa voidaan joukkoistaa ohjelmiston käyttäjäkunta, jolloin ohjelmistosta saadaan kehitettyä paremmin käyttäjiä palveleva.

Joukkoistamista voidaankin hyödyntää monin eri tavoin kaikissa ohjelmistoprojektin vaiheissa. Eniten tutkimusta ja käytännön toteutuksia on kuitenkin joukkoistamisen hyödyntämisestä ohjelmistoprojektin ohjelmointivaiheesta ja testausvaiheesta. Joukkoistetun ohjelmistotuotannon suurimmat edut ovat nopeus, edullisuus ja joustavuus. Toisaalta suurimpia haasteita ovat joukon motivoiminen ja työn laatu. Joukkoistetun ohjelmistotuotannon kohdalla on eriäviä tutkimuksia joukkoistamalla kehitettyjen ohjelmistojen laadun osalta. Varsinkin kilpailuina järjestetyt joukkoistamiset ovat olleet laadultaan jopa perinteistä ohjelmistotuotantoa parempia, mutta muutoin joukkoistamalla teetetyt työtehtävät eivät ole vastanneet laadustaan perinteisiä menetelmiä. Yleisellä tasolla joukkoistetun ohjelmistotuotannon edut ovat suuremmat kuin haitat, joten joukkoistetun ohjelmistotuotannon hyödyntämistä harkitsevan yrityksen kannattaa tarkastella mahdollisuuksia joukkoistamisen hyödyntämiseen ainakin ohjelmistoprojektin ohjelmointivaiheessa ja testausvaiheessa.

Joukkoistettuun ohjelmistotuotantoon kehitetyistä työkaluista on julkaistu monipuolisesti tieteellisiä artikkeleita, mutta usein niitä on testattu vain vähän tai ei ollenkaan osana teollista ohjelmistotuotantoa. Yritysten kynnys ottaa käyttöön uusia työkaluja on suuri varsinkin silloin, jos työkalun suoriutumisesta käytännön ohjelmistotuotannosta ei löydy tutkimustietoa. Joukkoistettua ohjelmistotuotantoa tutkivia tapaustutkimuksia on julkaistu kohtalaisesti, mutta varsinkin yrityksille tärkeä taloudellinen kannattavuus on usein jäänyt niissä vähemmälle huomiolle. Suurempi määrä tapaustutkimuksia joukkoistetusta ohjelmistotuotannosta, jotka sisältäisivät kattavan arvion joukkoistamisen hyödyntämisen kannattavuudesta, auttaisi yrityksiä tekemään päätöksen, onko joukkoistamisen hyödyntäminen järkevää heidän ohjelmistoprojekteissaan. Yritysten näkökulmasta, myös tutkimus joukkoistamisella toteutettujen töiden hinnoittelumekanismeista, auttaisi joukkoistamisen kustannusten arvioinnissa ja näin ollen madaltaisi kynnystä joukkoistamisen hyödyntämisestä ohjelmistotuotannossa. Jatkotutkimusta joukkoistamisen hyödyntämisestä ohjelmistotuotannossa tarvitaankin erityisesti joukkoistettuun ohjelmistotuotantoon julkaistujen työkalujen soveltuvuudesta teolliseen ohjelmistotuotantoon, joukkoistamisen taloudellisista hyödyistä ohjelmistotuotannossa verrattuna perinteisiin menetelmiin ja joukkoistetun työn hinnoittelumekanismeista.

LÄHTEET

Adepetu, A., Khaja, A. A., Al Abd, Y., Al Zaabi, A. & Svetinovic, D. (2012). Crowdrequire: A requirements engineering crowdsourcing platform. Teoksessa *2012 AAAI Spring Symposium Series.* AAAI.

Ali, R., Solis, C., Salehie, M., Omoronyia, I., Nuseibeh, B. & Maalej, W. (2011). Social sensing: When users become monitors. Teoksessa *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (476-479) ACM.

Ammann, P. & Offutt, J. (2016). *Introduction to software testing*. New York: Cambridge University Press.

Auler, R., Borin, E., de Halleux, P., Moskal, M. & Tillmann, N. (2014). Addressing JavaScript JIT engines performance quirks: A crowdsourced adaptive compiler. Teoksessa *International Conference on Compiler Construction* (218-237) Springer.

Bacon, D. F., Chen, Y., Parkes, D. & Rao, M. (2009). A market-based approach to software evolution. Teoksessa *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (973-980) ACM.

Ball, T., Burckhardt, S., de Halleux, J., Moskal, M., Protzenko, J. & Tillmann, N. (2015). Beyond open source: The TouchDevelop cloud-based integrated development environment. Teoksessa *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems* (83-93) IEEE.

Belleflamme, P., Lambert, T. & Schwienbacher, A. (2014). Crowdfunding: Tapping the right crowd. *Journal of business venturing*, 29(5), 585-609

Bennett, K. H. & Rajlich, V. T. (2000). Software maintenance and evolution: A roadmap. Teoksessa *Proceedings of the Conference on the Future of Software Engineering* (73-87) ACM.

Brabham D. (2013). *Crowdsourcing*. Cambridge, Massachusetts: MIT Press.

Chen, N. & Kim, S. (2012). Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles. Teoksessa *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (140-149) IEEE.

Cochran, R. A., D'Antoni, L., Livshits, B., Molnar, D. & Veanes, M. (2015). Program boosting: Program synthesis via crowd-sourcing. *ACM SIGPLAN Notices*, 50(1), 677-688.

DARPA. (2015). Crowd Sourced Formal Verification (CSFV) (Archived). Haettu osoitteesta https://www.darpa.mil/program/crowd-sourced-formal-verification

Dolstra, E., Vliegendhart, R. & Pouwelse, J. (2013). Crowdsourcing gui tests. Teoksessa *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation* (332-341) IEEE.

D'souza, D. F. & Wills, A. C. (1998). *Objects, components, and frameworks with UML: The catalysis approach*. Reading: Addison-Wesley.

Estellés-Arolas, E., & González-Ladrón-de-Guevara, F. (2012). Towards an integrated crowdsourcing definition. *Journal of Information Science*, 38(2), 189–200.

Frakes, W. B. & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 31(7), 529-536.

Goldman, M. (2011). Role-based interfaces for collaborative software development. Teoksessa *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology* (23-26) ACM.

Haikala, I. & Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt*. Helsinki: Talentum.

Hamidi, S., Andritsos, P. & Liaskos, S. (2014). Constructing adaptive configuration dialogs using crowd data. Teoksessa *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering* (485-490) ACM.

Hartmann, B., MacDougall, D., Brandt, J. & Klemmer, S. R. (2010). What would other programmers do: Suggesting solutions to error messages. Teoksessa *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1019-1028) ACM.

Hong L. & Page, S. E. (2004). Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences of the United States of America, 101*(46), 16385-16389.

Howe, J. (2006a). The rise of crowdsourcing. *Wired Magazine*, 14(6), 1-4.

Howe, J. (2006b, 2. kesäkuuta). Crowdsourcing: A Definition. Haettu 6.2.2019 osoitteesta https://crowdsourcing.typepad.com/cs/2006/06/

Howe, J. (2008). Crowdsourcing: *Why the power of the crowd is driving the future of the business.* New York: Crown Publishing

Jones, C. (2006). The economics of software maintenance in the twenty first century. Haettu osoitteesta http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.7697&rep=rep1&type=pdf

Kitchenham, B. A., Travassos, G. H., Von Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., Takada, S., Vehvilainen, R. & Yang, H. (1999). Towards an ontology of software maintenance. *Journal of Software Maintenance: Research and Practice*, 11(6), 365-389

Kleemann, F., Voß, G. G., & Rieder, K. (2008). Un (der) paid innovators: The commercial utilization of consumer work through crowdsourcing. *Science, technology & innovation studies,* 4(1), 5-26

Lakhani, K. R., Boudreau, K. J., Loh, P., Backstrom, L., Baldwin, C., Lonstein, E., . . . Guinan, E. C. (2013). Prize-based contests can provide solutions to computational biology problems. *Nature Biotechnology*, 31(2), 108.

Lakhani, K. R., Garvin, D. A. & Lonstein, E. (2010). Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit Case*, (610-032)

Lasecki, W. S., Kim, J., Rafter, N., Sen, O., Bigham, J. P. & Bernstein, M. S. (2015). Apparition: Crowdsourced user interfaces that come to life as you sketch them. Teoksessa *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (1925-1934) ACM.

LaToza, T. D., Chen, M., Jiang, L., Zhao, M. & Van Der Hoek, A. (2015). Borrowing from the crowd: A study of recombination in software design competitions. Teoksessa *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (551-562) IEEE.

LaToza, T. D., Towne, W. B., Van Der Hoek, A. & Herbsleb, J. D. (2013). Crowd development. Teoksessa *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering* (85-88) IEEE.

Leffingwell, D. & WidRig, D. (2003) *Managing software requirements: a use case approach.* Boston: Addison-Wesley.

Lim, S. L. & Finkelstein, A. (2012). StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE Transactions on Software Engineering*, 38(3), 707-735.

Lim, S. L., Damian, D., Ishikawa, F. & Finkelstein, A. (2013). Using web 2.0 for stakeholder analysis: StakeSource and its application in ten industrial projects. Teoksessa *Managing requirements knowledge* (221-242) Springer

Lim, S. L., Quercia, D. & Finkelstein, A. (2010). StakeSource: Harnessing the power of crowdsourcing and social networks in stakeholder analysis. Teoksessa *2010 ACM/IEEE 32nd International Conference on Software Engineering* (239-242) IEEE.

Liu, D., Bias, R. G., Lease, M. & Kuipers, R. (2012). Crowdsourcing for usability testing. *Proceedings of the American Society for Information Science and Technology*, 49(1), 1-10.

Mao, K., Capra, L., Harman, M. & Jia, Y. (2017). A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software*, 126, 57-84.

Mitchell, R. (15.3.2012). Cobol on the mainframe: Does it have a future? Haettu osoitteesta https://www.techworld.com/apps-wearables/cobol-on-mainframe-does-it-have-future-3344704/

Musson, R., Richards, J., Fisher, D., Bird, C., Bussone, B. & Ganguly, S. (2013). Leveraging the crowd: How 48,000 users helped improve lync performance. *IEEE Software*, 30(4), 38-45.

NASA. (2011). Challenge Platform: NTL. Haettu osoitteesta https://www.nasa.gov/sites/default/files/files/ntl-overview-sheet.pdf

Nebeling, M., Leone, S. & Norrie, M. C. (2012). Crowdsourced web engineering and design. Teoksessa *International Conference on Web Engineering* (31-45) Springer.

Nidhra, S. & Dondeti, J. (2012). Black box and white box testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2), 29-50.

Niessink, F. & Van Vliet, H. (2000). Software maintenance from a service perspective. Journal of Software Maintenance: Research and Practice, 12(2), 103-120.

Orso, A. & Rothermel, G. (2014). Software testing: A research travelogue (2000–2014). Teoksessa *Proceedings of the on Future of Software Engineering* (117-132) ACM.

Ponzanelli, L., Bacchelli, A. & Lanza, M. (2013). Seahawk: Stack overflow in the ide. Teoksessa *Proceedings of the 2013 International Conference on Software Engineering* (1295-1298) IEEE.

Schenk, E. & Guittard, C. (2011). Towards a characterization of crowdsourcing practices. *Journal of Innovation Economics & Management, 7*(1), 93-107.

Schmidt, D. C., Gokhale, A. & Natarajan, B. (2004). Leveraging application frameworks. *Queue*, 2(5), 66.

Schneider, C. & Cheung, T. (2013). The power of the crowd: Performing usability testing using an on-demand workforce. Teoksessa *Information systems development* (551-560) Springer.

Segev, L. (1.10.2014). How Microsoft is cleverly crowdsourcing Windows 10 development from its customers. Haettu osoitteesta https://thetechieguy.com/how-microsoft-is-cleverly-crowdsourcing-windows-10-development-from-its-customers/

Simula, H. (2013). The rise and fall of crowdsourcing? Teoksessa *2013 46th Hawaii International Conference on System Sciences* (2783-2791). IEEE.

Sommerville, I. (2016). *Software Engineering* (Tenth edition, global edition). England: Pearson.

StakeSource. (10.4.2019) The leading social networking tool that automatically identifies and prioritises the stakeholders for your projects, engages with the stakeholders, and understands their needs. Haettu osoitteesta http://www.stakesource.co.uk/

Stol, K. & Fitzgerald, B. (2014). Two's company, three's a crowd: A case study of crowdsourcing software development. Teoksessa *Proceedings of the 36th International Conference on Software Engineering* (187-198) ACM.

Van Lamsweerde, A. (2009). *Requirements engineering: From system goals to UML models to software* Chichester: John Wiley & Sons.

Vukovic, M. & Bartolini, C. (2010). Towards a research agenda for enterprise crowdsourcing. Teoksessa *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation* (425-434) Springer.