Tiia Rantanen

”Salaa ajattelen, että se on tärkeintä” – ohjelmistoarkkitehtuurin tila suomalaisissa yrityksissä

Tietotekniikan pro gradu -tutkielma

19. marraskuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Tiia Rantanen

Yhteystiedot: tiia.k.rantanen@jyu.fi

Ohjaajat: Jonne Itkonen ja Antti-Jussi Lakanen

Työn nimi: *”Salaa ajattelen, että se on tärkeintä”* - ohjelmistoarkkitehtuurin tila suomalaisissa yrityksissä

Title in English: The state of software architecture in Finnish companies

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmistotekniikka

Sivumäärä: 59+9

Tiivistelmä: Ohjelmistokehitys on muuttunut ja monimutkaistunut edellisen 20 vuoden aikana merkittävästi. Uusia järjestelmiä rakennetaan uudelleenkäyttämällä ja yhdistelemällä olemassa olevia komponentteja ja niiden arkkitehtuureja. Ketterät menetelmät ovat hallitseva lähestymistapa suunnitelmalähtöisten menetelmien sijaan. Tutkimuksen tavoitteena oli kartoittaa arkkitehtuurisuunnittelun tilaa avoimen lähdekoodin ja ketterien menetelmien tulokulmista ja selvittää, miten yrityksissä käytetään ohjelmistoarkkitehtuuria tuotannon tukena ja ohjauksessa. Tutkimuksen teoriaosassa käsitellään ohjelmistoarkkitehtuuria avoimen lähdekoodin, koodin uudelleenkäytön ja ketterien menetelmien näkökulmista. Tutkimus toteutettiin kyselytutkimuksena. Kyselytutkimuksen kohderyhmänä olivat IT-alalla työskentelevät henkilöt, jotka päivittäisessä työssään ovat osa arkkitehtonisten suunnittelupäätöksien tekoa. Tutkimustulokset osoittivat, että ohjelmistoarkkitehtuuri on kehittynyt kaavioista ja dokumenteista myös osaksi lähdekoodia. Ohjelmistoarkkitehtuuri on tärkeä apuväline kehityksessä, mikäli sitä osataan käyttää ja se ymmärretään oikein.

Avainsanat: avoin lähdekoodi, koodin uudelleenkäyttäminen, arkkitehtoninen tekninen velka, arkkitehtoninen yhteensopimattomuus, ketterä kehitys, ketterä arkkitehtuuri, dokumentaatio

Abstract: Software development has evolved and become more complex over the past 20

years. New systems are built by reusing and merging existing components and their architectures. Agile methods are the dominant approach to software development. The goal of the thesis was to establish an understanding how Finnish companies use software architecture to support communication and development. The thesis focused on how software architecture related to open source, code reuse and agile software development. The theoretical part of the study deals with software architecture from the perspectives of open source, code reuse, and agile methods. The research was conducted with an online survey. The target group of the survey were people working in IT involved in the design decisions related to software architecture. The research findings showed that software architecture has evolved from diagrams and documents into source code as well. Software architecture is an important tool in development if it is used and understood appropriately.

Keywords: open souce, code reuse, architectural technical debt, architectural mismatch, agile software development, agile architecture, documentation

Kuviot

Taulukot

Sisältö

1 Johdanto

Digitalisaatio on vauhdittanut tietotekniikka-alan kasvua (Rajala 2019; Korpimies 2017). Alan kasvu oli vuosina 2014-2015 voimakkaampaa kuin muilla tilastoiduilla toimialoilla (Korpimies 2017). Ala on yksi menestyneimmistä toimialoista Suomessa ja alan yritysten liikevaihto vuonna 2018 oli 13,9 miljardia euroa (Rajala 2019). Alaa piinaa vahva osaajapula (Rajala 2019; Korpimies 2017). Suomalaiset it-alan yritykset ovat keskikooltaan pieniä ja niitä on paljon, vaikkakin ala työllistää jo yli 60 000 työntekijää (Korpimies 2017; Korhonen 2017).

Avoimien ja ilmaisten ohjelmien ja työkalujen määrän kasvu on johtanut uusiin mahdollisuuksiin, mutta myös haasteisiin. Jokaista kirjoitettua koodiriviä kohden uudelleenkäytetään tuhansia rivejä jonkun toisen kirjoittamaa koodia (Garlan, Allen ja Ockerbloom 1995). Nykypäivän ohjelmointi onkin osittain olemassa olevien komponenttien yhdistämistä sekä yhteiskäyttöä kaupallisia ja avoimen lähdekoodin alustoja ja kirjastoja hyödyntämällä.

Ohjelmistokehitys on kehittynyt ja monimutkaistunut edellisen 20 vuoden aikana merkittävästi (Waterman 2018a). Uusia järjestelmiä rakennetaan uudelleenkäyttämällä ja yhdistelemällä komponentteja, joita ei ole suunniteltu toimimaan yhdessä (Mikkonen ja Taivalsaari 2019). Google Trendien mukaan ohjelmistoarkkitehtuuri ei enää ole niin kiinnostava aihe kuin aikaisemmin (“Google Trends” 2020) siitäkään huolimatta, että arkkitehtuuriin liittyy edelleen paljon ongelmia. Arkkitehtuurin liittyvien suunnittelupäätöksien muuttaminen on kallista ja aikaavievää, joskus jopa mahdotonta. Jatkokehitys ja muutokset tuotantotiimeissä lisäävät teknistä velkaa arkkitehtuurin osalta. Arkkitehtuuri on yleisin teknisen velan syy (Ampatzoglou ym. 2016; Martini, Stray ja Moe 2019). Koodin uudelleenkäyttö avoimen lähdekoodin kautta on lyhentänyt tuotantoon tarvittavaa aikaa ja laskenut kustannuksia, mutta myös aiheuttanut arkkitehtonista yhteensopimattomuutta ja altistanut tietoturvaongelmille.

Ketterät kehitysmallit ovat arkipäivää valtaosassa alan yrityksistä. Ketterän kehityksen mallit eivät itsessään tarjoa ratkaisuja arkkitehtuurisuunnitteluun tai sisällä työvaiheita suunnittelutyön tekemiselle. Ohjelmistoarkkitehtuuri nähdään suunnitelmalähtöisten (plan-driven development) kehitysmenetelmien kautta raskaana etukäteissuunnitteluna (big design upfront),

eikä lisäarvoa tuovana työkaluna.

Arkkitehtuurisuunnittelun määrä ja tarve on vähentynyt. Pilvipohjaisten alustapalveluiden yleistyminen sekä modernit, yhteistyötä lisäävät, kehitystyökalut vähentävät arkkitehtonisten suunnittelupäätöksien tarvetta (Hohpe ym. 2016). Ketterät menetelmät ovat vähentäneet tarvetta tehdä peruuttamattomia suunnittelupäätöksiä heti projektin alkaessa keskittyen yksinkertaiseen minimiarkkitehtuuriin, jolla voidaan tuoda asiakkaalle lisäarvoa nopeasti (Abrahamsson, Babar ja Kruchten 2010).

Tutkimuksen tavoitteena on kartoittaa arkkitehtuurisuunnittelun tilaa suomalaisissa yrityksissä ja selvittää, miten yrityksissä käytetään ohjelmistoarkkitehtuuria. Tutkielmassa käsitellään ohjelmistoarkkitehtuurin käsitettä, tarpeellisuutta ja käytänteitä Suomalaisten yrityksien näkökulmasta. Tutkielman tulokset painottavat ohjelmistoarkkitehtuurin käyttöä avoimen lähdekoodin sekä ketterien menetelmien kontekstissa. Teoriaosio keskittyy avoimen lähdekoodin osalta koodin uudelleenkäyttöön ja arkkitehtoniseen yhteensopivuuteen, mitkä ovat avoimeen lähdekoodiin liittyviä ilmiöitä. Lisäksi keskitytään arkkitehtoniseen tekniseen velkaan, jonka yksi aiheuttaja on koodin uudelleenkäyttö. Teoriaosio keskittyy ketterien menetelmien osalta menetelmiin yleisesti sekä arkkitehtuurin syntymiseen ja dokumentointiin ketterissä menetelmissä sekä arkkitehdin roolin kehittymiseen.

Tutkimus pohjautuu seuraaviin kysymyksiin:

1. **Arkkitehtuurin käyttö yrityksissä**: Miten arkkitehtuuria käytetään yrityksissä?
2. **Avoin lähdekoodi**: Vaikuttaako avoimen ja uudelleenkäytettävän lähdekoodin käyttö arkkitehtuuriin?
3. **Ketterät menetelmät**: Miten arkkitehtuuri suunnitellaan ja dokumentoidaan, kun käytetään ketteriä malleja?

Tutkimuksen teoriaosuuden pohjana on käytetty alan kirjallisuutta ja tutkimuksia. Tutkimusaineisto kerättiin verkkolomakkeella, johon on voinut vastata maaliskuun 2018 ajan. Tutkimukseen vastasi 38 henkilöä.

Tutkimus jakautuu 7 lukuun. Luvussa 2 esitetään mitä ohjelmistoarkkitehtuuri tarkoittaa. Luvussa 3 käydään läpi avoimen lähdekoodin ja koodin uudelleenkäytön vaikutus ohjel-

mistoarkkitehtuuriin. Luvussa 4 käsitellään ketteriä menetelmiä ja niiden vaikutusta ohjelmistoarkkitehtuurin suunnitteluun ja dokumentointiin sekä arkkitehdin rooliin. Luvussa 5 käydään läpi tutkimuksen toteuttamistapa ja menetelmät. Luvussa 6 esitetään tutkimuksen tulokset. Luvussa 7 on pohdinta tutkimuksen tuloksista, luotettavuudesta ja jatkotutkimuksesta.

2 Ohjelmistoarkkitehtuuri

Kruchten (2004) mukaan ohjelmistoarkkitehtuuri tarkoittaa järjestelmän perustavanlaatuista rakennetta ja sitä, miten komponentit ovat kytköksissä toisiinsa korkealla tasolla, sekä näiden käyttäytymistä. Fowler (2003) mukaan ohjelmistoarkkitehtuuri tarkoittaa niitä suunnittelupäätöksiä, jotka täytyy tehdä aikaisin tai joita on hankala muuttaa myöhemmin. Suunnittelupäätös on kuvaus muutoksista, perusteluista ja suunnitterelurajoituksista, jotka toteuttavat osittain tai kokonaan yhden tai useamman vaatimuksen (Jansen ja Bosch 2005). Ohjelmistoarkkitehtuuri on siis kokoonpano suunnittelupäätöksiä, jotka määrittävät järjestelmän rakenteen ja laadulliset ominaisuudet (Jansen ja Bosch 2005).

Rechtin (1994) mukaan arkkitehtuuri sisältää kaksi osaa: i) osiointistrategian ja ii) koordinointistrategian. Osiointistrategia koostuu järjestelmän osituksista, erillisistä ja päällekkäisistä osioista tai komponenteista. Koordinointistrategia määrittää osioiden ja komponenttien väliset rajapinnat.

IEEE:n arkkitehtuurien kuvaamista koskeva standardi määrittelee ohjelmistoarkkitehtuurin järjestelmän perusorganisaatioksi, joka sisältää järjestelmän osat, niiden keskinäiset suhteet ja niiden suhteet ympäristöön, sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja kehittämistä (“ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description” 2011).

Arkkitehtuurin voidaan siis ajatella koostuvan neljästä osasta:

- ohjelman tai järjestelmän rakenteesta, eli *elementeistä*,
- niiden välisistä suhteista sekä
- niiden ominaisuuksista ja
- käyttäytymisestä.

Ohjelmistoarkkitehtuuria on tutkittu vuodesta 1968 (Dijkstra 1968). Tästä huolimatta ei ole konsensusta siitä, mitä ohjelmistoarkkitehtuuri tarkalleen ottaen on. Vaikka määritelmästä on käyty keskustelua jo pitkään, arkkitehtuurista on syntynyt liuta toisistaan poikkeavia määrityksiä ja näkemyksiä (“What is your definition of software architecture?” 2010). Jopa käsit-

teen määrittelystä on tehty omaa tutkimusta (Baragry ja Reed 1998).

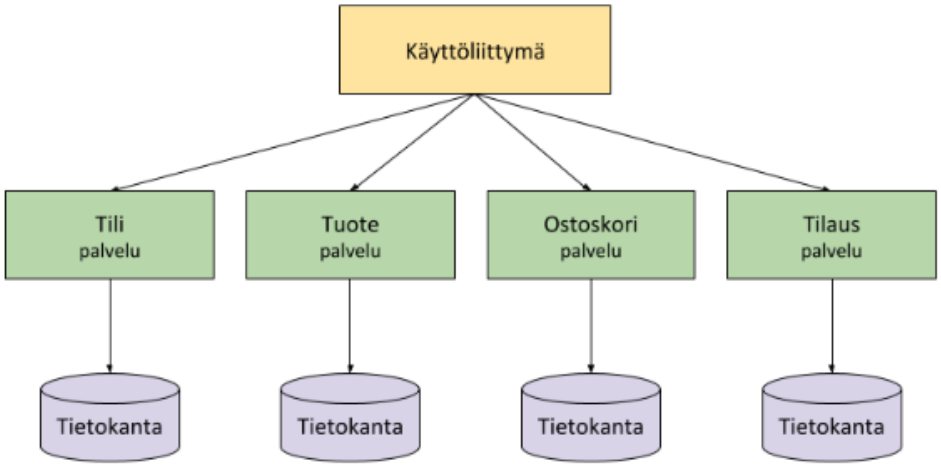
Arkkitehtuuri tarjoaa järjestelmälle pohjan ja säännöt, joiden mukaan ohjelmistoa rakennetaan ja kehitetään. Arkkitehtuuri on korkean tason yksinkertaistus tai abstraktio monimutkaisesta järjestelmästä (Bass, Clements ja Kazman 2013; Koskimies ja Mikkonen 2005). Korkean tason kuvaaminen tarkoittaa sitä, että siitä ei käy ilmi elementtien sisäiset toiminnot, eli lähdekoodin tasolla olevat tekniset ratkaisut.

Jokaisella järjestelmällä on arkkitehtuuri, vaikka sitä ei erikseen suunniteltaisi (Miyachi 2011; Ambler 2020). Arkkitehtuurin suunnitteleminen tai suunnittelematta jättäminen ei itsessään takaa, että arkkitehtuuri toteuttaa sille asetetut vaatimukset ja tavoitteet (Bass, Clements ja Kazman 2013). Arkkitehtuuri määrittää miten järjestelmä toteuttaa annetut laadulliset vaatimukset (Jaiswal 2019). Arkkitehtuurisuunnittelun avulla järjestelmän laadullisten vaatimusten toteutumista ja käyttäytymistä voidaan arvioida aikaisessa vaiheessa ennen järjestelmän toteuttamista (Perry ja Wolf 1992). Mahdollisuus varmistaa, että järjestelmä toteuttaa sidosryhmien tarpeet ennen järjestelmän rakentamista vähentää kustannuksia ja riskejä (Obbink ym. 2020; Fairbanks 2010; Poort ja van Vliet 2012). Keuler, Wagner ja Winkler (2012) mukaan arkkitehtuurin tulee olla yksiselitteinen, jotta sitä on helppo seurata ja hankala rikkoa.

Arkkitehtuurisuunnittelu mahdollistaa viestinnän sidosryhmien kanssa siten, että sidosryhmät ymmärtävät abstraktioiden avulla mistä on kyse (Bass, Clements ja Kazman 2013; Clements 2011; Diaz-Pace ym. 2015). Näin sidosryhmien tarpeet ja niiden vaikutukset tulevat paremmin huomioiduksi.

Ohjelmistoarkkitehtuurin määrityksessä arkkitehtuurin ja suunnittelun ero jää tulkinnan varaiseksi (Solms 2012; Martin 2018). Suunnittelulla tarkoitetaan lähdekoodin tasolla olevia teknisiä ratkaisuja toiminnallisten vaatimuksien kautta (Jaiswal 2019). Joidenkin näkemyksien mukaan suunnittelu on osa ohjelmistoarkkitehtuuria (Brown 2015; Martin 2018). Hohpe ym. (2016) mukaan koodi on arkkitehtuuria. Garlan (1995) mukaan termiä arkkitehtuuri ei saisi laimentaa soveltamalla sitä aivan kaikkeen, mutta se on hankalaa, koska arkkitehtuuri kattaa kuitenkin suuren osan suunnittelutyöstä riippuen arkkitehdistä, lähestymistavasta ja mallista.

Kuvassa 1 on esimerkki yksinkertaistetusta verkkokaupan ohjelmistoarkkitehtuurista. Arkkitehtuurityyli on mikropalveluarkkitehtuuri.



Kuvio 1. Esimerkki verkkokaupan toteuttavasta mikropalveluarkkitehtuurista

Yhteenvetona ohjelmistoarkkitehtuuri on siis abstraktio järjestelmän rakenteesta, josta voidaan arvioida miten järjestelmä toteuttaa sille asetetut vaatimukset ja tavoitteet. Arkkitehtuurin abstraktion taso riippuu (arkkitehdin) tulkinnasta ja sidosryhmän näkökulmasta.

3 Avoin lähdekoodi ja arkkitehtuuri

Tässä luvussa käsitellään avointa lähdekoodia ja koodin uudelleenkäyttämistä sekä niihin liittyvää arkkitehtonista yhteensopimattomuutta ja teknistä velkaa.

3.1 Avoin lähdekoodi ja koodin uudelleenkäyttäminen

Ohjelmistokehityksestä on tullut yhteisöllistä. Kehittäjät etsivät toisiltaan vastauksia ohjelmointiin liittyviin ongelmiin. Ongelmista voidaan keskustella ja lähdekoodia jakaa esimerkiksi Stack Overflow- ja GitHub-verkkopalveluissa.

Ohjelmakoodia voidaan uudelleenkäyttää kahdella tapaa (Heinemann ym. 2011):

1. whitebox uudelleenkäyttö, jossa koodia voidaan ensin muokata tai lisätä suoraan osaksi muuta lähdekoodia tai
2. blackbox uudelleenkäyttö, jossa koodi lisätään erillisenä komponenttina tai käyttäen rajapintaa siten, että lähdekoodia ei tarvitse itse muokata.

Ohjelmoijat uudelleenkäyttävät koodien osia kirjastoista, aikaisemmista projekteista ja netistä ohjelmien kehittämiseen ja ylläpitoon. Valtaosa uudelleenkäytöstä tapahtuu kopioimalla ja liittämällä lähdekoodin osia ohjelmasta toiseen (Constantinou ja Stamelos 2017). Opportunistisesti uudelleenkäytetty koodi ei automaattisesti ole keskenään yhteensopivaa ja se saattaa vaatia arkkitehtonisia muutoksia, jotta toiminnalliset ja laadulliset vaatimukset voidaan saavuttaa (Shaw 1995). Lähdekoodi tulisi auditoida ja yhteensovittaa harkitusti, jotta uudelleenkäyttö ei aiheuta yhteensopimattomuutta (Constantinou ja Stamelos 2017). Koodin yhteensopivuuden systemaattinen analysointi sekä avoimen lähdekoodin komponenttien ominaisuuksiin tutustuminen on haastavaa (Mikkonen ja Taivalsaari 2019).

Komponenttien, kirjastojen ja alustojen yhdistämisessä on tunnistettu kuusi ongelmaa (Garlan, Allen ja Ockerbloom 1995):

1. koodin laajuus,
2. huono suorituskyky,

3. muutostyö, joka aiheutuu yhteensopimattomasta koodista,
4. olemassa olevan toiminnallisuuden tekeminen uudelleen, jotta se soveltuisi paremmin käyttökohteeseen,
5. tarpeeton monimutkaisuus ja
6. monimutkainen, virheille altis rakennusprosessi.

Vaikka uudelleenkäytettävän koodin muokkaaminen tai korjaaminen tarvittaessa, jotta se saadaan sopivaksi vaatii kolminkertaisen määrän resursseja verrattuna itsekirjoitettuun koodiin, on se silti helpommin laajennettavissa ja ylläpidettävissä (Feitosa ym. 2020).

3.2 Arkkitehtoninen yhteensopimattomuus

Arkkitehtoninen yhteensopimattomuus syntyy komponenttien yhdistämisestä ja lähdekoodin uudelleenkäyttämisestä. Vaikka komponentit olisi tehty samalla ohjelmointikielellä, niitä ajettaisiin samalla alustalla, ja ne olisivat tarkoitettu uudelleenkäytettäviksi, niiden yhteiskäytössä voi tulla ongelmia. Kehittäjät voivat joutua toteuttamaan uudestaan olemassaolevaa toiminnallisuutta, kirjoittamaan lisäkoodia komponenttien yhdistämiseksi tai muokkaamaan komponenttien toteutusta, jotta komponentit saadaan toimimaan yhdessä. Tällaisesta arkkitehtonisesta yhteensopimattomuudesta johtuen lopulliset järjestelmät voivat olla suuria ja hitaita (Garlan, Allen ja Ockerbloom 1995).

Arkkitehtoniset yhteensopimattomuudet voidaan jakaa neljään eri luokkaan (Garlan, Allen ja Ockerbloom 1995):

1. olettamukset komponenttien rakenteesta, tiedon käsittelystä ja niiden toiminnasta,
2. olettamukset tietomalleista sekä protokollista, eli miten interaktiot tapahtuvat,
3. olettamukset arkkitehtuurin rakenteesta, eli topologia sekä tiettyjen komponenttien puuttuminen tai olemassaolo ja
4. olettamukset latausjärjestyksestä ja prosessista.

Arkkitehtonisen yhteensopimattomuuden voi välttää estämällä, havaitsemalla, tunnistamalla tai jälkeenpäin korjaamalla (Garlan, Allen ja Ockerbloom 2009). Kehittäessä ja käytettäessä avointa lähdekoodia arkkitehtonisen yhteensopimattomuuden voi välttää standardoimalla

tiettyjen sovelluskehyksien käytön ja arkkitehturityylin sekä tuottamalla esimerkkejä, jotka selkeyttävät mitkä arkkitehtoniset yleistykset ja sovellutukset käyvät minkäkin ohjelmiston kanssa (Garlan, Allen ja Ockerbloom 2009).

Pienemmät sovellukset tai järjestelmät voidaan rakentaa valmiiden sovelluskehyksien avulla, jotka rajaavat kompleksisuutta. Vaihtuvien vaatimuksien hallintaan voidaan käyttää sopivaa kehystä kompleksisuuden pienentämiseksi (Waterman 2018b).

3.3 Arkkitehtoninen tekninen velka

Tekninen velka on kielikuva, jolla viitataan huonolaatuisesta ohjelmistosuunnittelusta ja -kehityksestä aiheutuviin seurauksiin (Cunningham 1992). Teknistä velkaa syntyy myös itsestään järjestelmän elinkaaren aikana (Cunningham 1992). Teknistä velkaa muodostuu kahdessa eri kehityksen vaiheessa; toteutuksessa sekä jatkokehityksessä (Kruchten, Nord ja Ozkaya 2012). Tekninen velka on kierre (Fowler ja Beck 2018), joka vaikuttaa heikentävästi ohjelmistokehitykseen (Power 2013), ohjelman laatuun (Tom, Aurum ja Vidgen 2013) ja kehittäjän motivaatioon (Peters 2014). Tekninen velka muodostuu järjestelmän nykytilan ja tavoitetilan erotuksesta (Cunningham 1992).

Tekninen velka voi olla tarkoituksenmukaista ja strategista tai tahatonta ja vahingossa tehtyä (Fowler ja Beck 2018). Tarkoituksella otetulla teknisellä velalla voidaan nopeuttaa markkinoille vientiä, alentaa tilapäisesti kustannuksia tai poiketa olemassa olevasta arkkitehtuurista ja prosessista. Tekninen velka on kuitenkin aina laina, joka erääntyy maksettavaksi myöhemmin (Cunningham 1992).

Teknistä velkaa on viittä eri tyyppiä (Li, Avgeriou ja Liang 2015; Tom, Aurum ja Vidgen 2013):

- koodivelka,
- suunnittelu- ja arkkitehtoninen velka,
- testausvelka,
- tiedonjako- tai dokumentaatiovelka ja
- ympäristö- tai infrastruktuurivelka.

Arkkitehtuuri ja siihen liittyvät suunnittelupäätökset ovat yleisin teknisen velan syy (Ampatzoglou ym. 2016; Martini, Stray ja Moe 2019; Ernst ym. 2015).

Arkkitehtoninen tekninen velka aiheutuu huonoista ratkaisuista. Tiedossa olevat virheet, koodin rappeutuminen, toteuttamattomat ominaisuudet ja vanhentunut dokumentaatio ovat arkkitehtonisen velan esiintymismuotoja (Brown ym. 2010; Tom, Aurum ja Vidgen 2013). Mainittuja esiintymismuotoja voi aiheuttaa esimerkiksi monoliittiarkkitehtuuri useilla riippuvaisuuksilla moduulien välillä. Tämä saattaa johtaa koodimuutoksiin useissa eri paikoissa, kun jokin asia muuttuu. Muutoksiin tarvitaan enemmän aikaa ja niiden ohessa voi esiintyä virheitä. Hankalasti luettava arkkitehtuuri ja lähdekoodi aiheuttaa lisää teknistä velkaa.

Martini ja Bosch (2016) painottavat säännöllisen ja jatkuvan arkkitehtuurin huomioinnin tärkeyttä teknisen velan poistajana ja arkkitehtuurin rappeutumisen estäjänä, koska kertynyt tekninen velka ja refaktoroinnin tarve ovat suoraan verrannollisia todellisen arkkitehtuurin harhaanjohtavuudelle (Holmes ja Nicolaescu 2017). Tämä on haastavaa, koska arkkitehtonista teknistä velkaa on hankala arvioida tai mitata (Fernández-Sánchez ym. 2015).

4 Arkkitehtuuri ja ketterä ohjelmistokehitys

Tässä luvussa käsitellään ketterän kehityksen vaikutusta arkkitehtuurin syntymiseen, dokumentointiin sekä arkkitehdin roolin kehittymiseen.

4.1 Ketterän ohjelmistokehityksen julistus

Ketteriä lähestymistapoja yhdistää ketterän ohjelmistokehityksen julistus. Ketterä ohjelmistokehitys on filosofia, joka perustuu neljälle arvolle ("Agile Manifesto" 2001):

1. yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja,
2. toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota,
3. asiakasyhteistyötä enemmän kuin sopimusneuvotteluja ja
4. vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa.

Useat ketterät menetelmät hajoittavat ohjelmistokehityksen pienempiin inkrementteihin, eli askellisäyksiin. Jokainen askellisäys lisää järjestelmään toiminnallisuutta ja laajentaa sitä. Askellisäyksiä tarvitaan yksi tai useampia, jotta saadaan valmis järjestelmä. Yksittäinen askellisäys koostuu yhdestä tai useammasta iteraatiosta, eli kehitysjaksosta. Yksittäinen kehitysjakso kestää yleensä yhdestä neljään viikkoa. Kehitysjakson aikana järjestelmän parissa työskentelee tiimi, jossa on monenlaista osaamista. Tiimin työtehtäviin kuuluu esimerkiksi kehitysjakson suunnittelu, analysointi, käyttöliittymäsuunnittelu, ohjelmointi ja testaaminen. Jokaisen askellisäyksen ja kehitysjakson päätteeksi on muodostunut toimiva järjestelmä. Tämän tyyppinen iteratiivinen ja inkrementaalinen toimintatapa vähentää riskejä ja mahdollistaa nopean sopeutumisen muutoksiin. (Moran 2014; Beck 1999)

Ketterien menetelmien kirjo on laaja. Jokainen erillinen menetelmä koostuu yksilöllisestä käytännöstä ja terminologiasta, joita sovelletaan kehittäjän päivittäisessä työssä. Tutkimukset ovat osoittaneet, että ketterät menetelmät ovat hallitseva lähestymistapa ohjelmistokehitykseen (Ali Babar, Brown ja Mistrík 2014). Suosituin ketterän ohjelmistokehityksen menetelmä on Scrum (Allisy-Roberts ym. 2017). Tulevaisuudessa monet tiimit siirtynevät DevOpsiin tai täydentävät sillä olemassa olevia menetelmiä (Hoda, Salleh ja Grundy 2018).

4.2 Arkkitehtuuri ja ketterä kehitys

Ketterän ohjelmistokehityksen julistuksen periaatteiden mukaan ”parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä” (“Agile Manifesto” 2001). Kruchten (2010) mukaan ketterän arkkitehtuurin tulee ilmestyä tai syntyä asteittain, kehitysjaksojen edetessä, pienimuotoisen refaktoroinnin tuloksena. Ketterä arkkitehtuuri on mikä tahansa arkkitehtuuri, joka on suunniteltu ketterällä prosessilla ja on muokattavissa sekä kestää hyvin muutoksia (Waterman 2018a; Waterman, Noble ja Allan 2015; Miyachi 2011; Mancl ym. 2009; Booch 2006; Schade ja Plewnia 2018). Arkkitehtuuri kehittyy ja mukautuu järjestelmän mukana samanaikaisesti, kun tiimi käsittelee arkkitehtonisia suunnittelupäätöksiä. Arkkitehtuuri voi siis olla jollain tavalla suunniteltua tai syntyä asteittain itsestään.

Monien ketterien menetelmien harjoittajien mukaan perinteiset ohjelmistoarkkitehtuurin käytänteet ovat osa suunnitelmalähtöisiä malleja, joihin kuuluu raskas etukäteissuunnittelu (R. Nord ja J. Tomayko 2006; Babar 2009; R. L. Nord ja J. E. Tomayko 2006). Harjoittajat kokevat, että raskas korkean tason etukäteissuunnittelu ja arkkitehtuurin ylläpito vaatii paljon työtä tuottamatta kuitenkaan riittävästi lisäarvoa asiakkaalle (Babar 2009). Näistä syistä etukäteissuunnittelu ei ole ketterän kehityksen periaatteiden mukaista (Schade ja Plewnia 2018; Babar 2009).

Ohjelmistokehityksessä tulee miettiä kuinka paljon arkkitehtuurisuunnittelua ketterissä ympäristöissä kannattaa, ja voi tehdä, ennen varsinaisen ohjelmoinnin aloitusta (Ali Babar, Brown ja Mistrík 2014). Tätä ongelmaa ketterän kehityksen mallit ja metodologiat eivät lähtökohtaisesti huomioi (Waterman 2018a). Liiallinen suunnittelu ja suunnitelmallisuus hidastaa ohjelmointityön aloitusta sekä viivästyttää palautteen saamista käyttäjiltä, kun taas liian vähäinen arkkitehtuurisuunnittelu voi johtaa huonoihin ja hätäisesti tehtyihin päätöksiin, jotka eivät välttämättä tue järjestelmän vaatimuksia (Waterman 2018a).

Waterman (2018a) mukaan arkkitehtuurisuunnittelun optimaalinen taso on etukäteissuunnittelun ja ketterän toteutuksen välimaastossa. Optimaaliseen tasoon vaikuttaa toteuttavan järjestelmän vaatimuksien vaihtuvuus, teknisten riskien määrä, miten aikaisin asiakas haluaa julkistaa lopputuotteen, sekä tiimin ketteryys ja tekninen osaaminen Waterman (2018a). Kazman mukaan optimaalinen arkkitehtuurisuunnittelun taso riippuu projektin koosta (Ali

Babar, Brown ja Mistrík 2014). Arkkitehtuurisuunnittelun painottaminen alkuvaiheeseen on hyödyllistä, mikäli toteutettava järjestelmä on suuri ja monimutkainen. Pienissä järjestelmissä, joissa vaatimukset eivät ole tarkkoja tai selvillä, arkkitehtuurisuunnittelussa voidaan pyrkiä huomioimaan päätoiminnallisuudet. Etupainotteiseen suunnitteluun ei kannata käyttää paljoa aikaa. Kazmanin mukaan näitä kahta tapaa voidaan myös yhdistää toisiinsa oikeaaikaisesti parhaan tuloksen saavuttamiseksi (Ali Babar, Brown ja Mistrík 2014).

Poort ja van Vliet (2012) mukaan riski väärästä suunnittelupäätöksestä laskee ajan kuluessa, koska tietoa on enemmän saatavilla. Tästä syystä suunnittelupäätökset tulee tehdä mahdollisimman myöhään (Waterman, Noble ja Allan 2015). Tästä huolimatta arkkitehtuurin tulee olla mukautuva ja avoin muutoksille, koska muutoksia tapahtuu joka tapauksessa ennemmin tai myöhemmin (Waterman, Noble ja Allan 2015; Schade ja Plewnia 2018; Ali Babar, Brown ja Mistrík 2014).

Suunnitelmalähtöisissä menetelmissä arkkitehtuurin suunnittelu tehdään etukäteissuunnitteluna. Etukäteissuunnittelu tarkoittaa, että arkkitehtuuri suunnitellaan kokonaan ennen projektin aloitusta (Dragičević ja Bošnjak 2019). Suunnitelmalähtöisissä malleissa käytetty etukäteissuunnitteluna tehtävä arkkitehtuuri ei ole ketterien arvojen tai periaatteiden mukainen. Suunnitelmalähtöisen tiiviin ja laajan dokumentaation käytettävyys on yksi suunnitelmalähtöisen kehityksen rajoituksista (Larson ja Chang 2016) ja Woods (2015) mukaan näiden ongelmien vuoksi arkkitehtuuridokumentaatiota ei välttämättä lue kukaan. Näistä syistä ketterissä menetelmissä arkkitehtuuri tulee toteuttaa eri tavalla kuin suunnitelmalähtöisissä menetelmissä (Abrahamsson, Babar ja Kruchten 2010).

Arkkitehtuurisuunnitteluun ketterien menetelmien avulla on useita erilaisia lähestymistapoja, joita voidaan käyttää määrittelemään, kuinka paljon etukäteissuunnittelua tehdään. Näitä lähestymistapoja ovat *muutokseen vastaaminen* (engl. respond to change), *ilmestyvä arkkitehtuuri* (engl. emergent architecture) ja *kävelevä luuranko* (engl. walking skeleton) (Dragičević ja Bošnjak 2019; Cockburn 2004).

Muutokseen vastaaminen tarkoittaa, että arkkitehtuuria tulisi suunnitella ja dokumentoida tarpeeksi (Hoda, Noble ja Marshall 2012), mutta kuitenkin vain sen verran kuin on välttämätöntä seuraavaan vaiheeseen siirtymistä varten (Cockburn 2007; Dragičević ja Bošn-

jak 2019). Käytännössä tämä tarkoittaa, että arkkitehtuuridokumenttia ei luoda etukäteen ja suunnittelupäätöksiä tehdään vasta sitten, kun tietoa on riittävästi saatavilla. Tämä minimoi tarvittavien muutoksien määrän. Lähestymistapa voidaan toteuttaa esimerkiksi Scrumissa sprintti 0 -menettelyllä, jossa prosessin ensimmäinen iteraatio käytetään karkean arkkitehtuurin luomiseen, tai itse iteraatioiden aikana jokaisen sprintin suunnitteluvaiheessa (Rost ym. 2015).

Ilmestyvä arkkitehtuuri tarkoittaa, että tehdään ainoastaan ne päätökset, joiden avulla työ voidaan aloittaa (Dragičević ja Bošnjak 2019). Näitä päätöksiä ovat esimerkiksi teknologiavalinnat tai tiedossa olevat rajoitteet. Arkkitehtuuri on projektin alussa erittäin yksinkertainen. Tällä lähestymistavalla voidaan tuottaa toimiva järjestelmä jo hyvin aikaisessa vaiheessa, esimerkiksi pienin toimiva tuote (minimum viable product) (Dragičević ja Bošnjak 2019). Käytännössä tämä tarkoittaa, että arkkitehtuurisuunnittelua tehdään iteraatioiden aikana tai osana iteraatioiden suunnittelua ilman erillistä projektin alussa tapahtuvaa arkkitehtuurisuunnittelua (Rost ym. 2015).

Kävelevä luuranko ja ilmestyvä arkitehtuuri ovat lähestymistapoina hyvin samantyyppisiä. Kävelevä luuranko sisältää arkkitehtuurin näkökulmasta vain välttämättömät komponentit. Komponentit muodostavat pelkän luurangon, koska arkkitehtuuri ei sisällä toiminnallisuuteen liittyviä asioita. Luuranko on kävelevä, koska sillä voidaan tuottaa heti alussa toimiva järjestelmä. Arkkitehtuuria voidaan kuvailla lähdekoodilla (Cockburn 2004). Tällä tavalla arkkitehtuuria voidaan testata heti alussa ja jokaisella iteraatiolla saadaan toimiva järjestelmä. (Cockburn 2004)

Lähestymistavasta riippumatta arkkitehtuurisuunnittelua ja siihen liittyviä päätöksiä voi tehdä erillinen arkkitehti, tiimin sisäisesti nimetty arkkitehti, tiimi kokonaisuudessaan tai rinnakkainen nimetty arkkitehtuuritiimi (Rost ym. 2015).

Booch (2006) mukaan arkkitehtuuri voi olla myös suunnittelematonta. Arkkitehtuuri syntyy ja muodostuu (itsestään) suunnittelupäätöksistä, joita tehdään tietoisesti tai epätietoisesti kehityksen aikana. Joidenkin näkemyksien mukaan suunnittelematon arkkitehtuuri on pääsääntöisesti huono käytäntö (Waterman, Noble ja Allan 2015; Miyachi 2011).

4.3 Arkkitehtuurin dokumentointi ketterässä ohjelmistokehityksessä

Ketterän ohjelmistokehityksen julistuksessa ja periaatteissa kerrotaan dokumentaatiosta ja viestinnästä seuraavasti: arvostetaan ”toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota” sekä ”tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu” (“Agile Manifesto” 2001). Joillekin ketterien menetelmien harjoittajille nämä arvot ja periaatteet tarkoittavat, että yleisesti dokumentaatiota ei tarvita, koska dokumentaation ylläpitäminen vie paljon aikaa (Savaşcı ja Çetin 2018; Schade ja Plewnia 2018). Sherman ja Hadar (2012) mukaan arvoilla ja periaatteilla kuitenkin tarkoitetaan, että tarpeetonta dokumentaatiota ei kannata laatia tai ylläpitää. Toisin sanoen, ketterässä kehityksessä tarvittavat dokumentit ovat sellaisia, jotka palvelevat lopputuotetta ja tuovat lisäarvoa (Rodríguez ym. 2019; Larson ja Chang 2016). Babar (2009) toteuttaman tutkimuksen aineiston mukaan karsimalla lisäarvoa tuottamatonta eli tarpeetonta dokumentaatiota voitiin vähentää dokumentaatioon tarvittavia resursseja 30-40%.

Monet ketterien menetelmien harjoittajat eivät huomioi dokumentaation vaikutusta viestintään ja tiedon säilyttämiseen (Ali Babar, Brown ja Mistrík 2014; Cockburn 2007). Vähäinen arkkitehtuuridokumentaatio aiheuttaa tiedon rapautumista ja viestinnän heikentymistä (Gerdes ym. 2016). Dokumentaatio myös vaikuttaa yhdenmukaisen järjestelmän toteutukseen ja arkkitehtuurin arviointiin (Gerdes ym. 2016). Coplien ja Bjørnvig (2011) mukaan dokumentaatiota kirjoitetaan kahdesta syystä: i) asioiden muistamiseksi ja ii) niiden viestintään. Dokumentoidun tiedon haaste on kuitenkin käytännön soveltaminen ja dokumentaation ajantasaisuus, mikäli dokumentteja ei generoida automaattisesti (Mirakhorli ja Cleland-Huang 2013).

Arkkitehtuuridokumentaatiota on yleensä käytetty sidosryhmätyöskentelyn työvälineenä (Clements 2011; “ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description” 2011). Arkkitehtuuridokumentaatio toimii pohjana, jonka avulla voidaan keskustella sidosryhmien vaatimuksista sellaisella abstraktiotasolla, joka on ei-teknisille henkilöille selkeä (Diaz-Pace ym. 2015). Tästä syystä suunnitelma ei saa sisältää liikaa teknisiä yksityiskohtia (Selic 2009). Dragičević ja Bošnjak (2019) mukaan arkkitehtuurin esittäminen pelkästään erillisinä dokumentteina jää kuitenkin pian historiaan. Ohjelmistoarkkitehtuurin suunnittelu ja dokumentointi on siirtynyt perinteisistä tavoista moderneihin menetel-

miin ja käytäntöihin (Erder ja Pureur 2016). Arkkitehtuuri ei siis välttämättä ole dokumentti tai suunnitelma, jota sidosryhmät voivat hyödyntää.

Arkkitehtuuridokumentaation rakenne voi olla esimerkiksi Word-dokumentti, UML-kaavio tai erillinen verkkosivusto tai Wiki (Bachmann ja Merson 2005; Graaf 2015; Diaz-Pace ym. 2015; Rost ym. 2015). On myös mahdollista, että arkkitehtuuri perustuu kehittäjien tietämykseen, ilman erillistä dokumentaatiota (Rost ym. 2015; Ambler 2020; Booch 2006). Hohpe ym. (2016) mukaan arkkitehtuuri on koodia, jolla voidaan kuvata suunnittelupäätöksiä komponentteina ilman perinteistä dokumentaatiota. Toiminnallisten vaatimuksien dokumentointimenetelmiin kuuluvat esimerkiksi luonnollinen kieli eli tekstipohjaiset kuvaukset, tietovuokaaviot, käyttötapauskaaviot ja aktiviteettikaaviot (Pohl 2016). Näistä tehokkain ja ymmärrettävin loppukäyttäjälle ja kehittäjälle on aktiviteettikaavio (Ibriwesh, Ho ja Chai 2018). Ketterien arkkitehtuurien dokumentointiin ei enää käytetä arkkitehtuurin kuvauskieliä (ADL), koska näitä pidetään liian raskaina (Malavolta ym. 2013).

Huono dokumentaatio aiheuttaa tietojärjestelmäprojektien epäonnistumista (Adu 2014). Arviolta 60% kaikista virheitä tapahtuu vaatimusten määrittelyvaiheessa (Pohl 2016). Virheen sattuessa näin aikaisessa vaiheessa, projektin budjetti ja aikataulu todennäköisesti ylittyvät. Tästä syystä vaatimusten määrittelyä pidetään vaikeimpana ja tärkeimpänä vaiheena ohjelmiston kehittämisen elinkaaressa (Jabbar ym. 2007).

4.4 Arkkitehdin roolin kehittyminen

Ohjelmistoarkkitehdin rooli ja vastuut ovat kehittyneet ketterien menetelmien myötä. Arkkitehtuurisuunnittelun määrä ja tarve on vähentynyt. Pilvipohjaisten alustapalveluiden yleistyminen sekä modernit, yhteistyötä lisäävät kehitystyökalut vähentävät arkkitehtonisten suunnittelupäätöksien tarvetta (Hohpe ym. 2016). Ketterät menetelmät ovat vähentäneet tarvetta tehdä peruuttamattomia suunnittelupäätöksiä heti projektin alkaessa keskittyen yksinkertaiseen arkkitehtuuriin, jolla voidaan luoda lisäarvoa nopeasti (Abrahamsson, Babar ja Kruchten 2010). Viime vuosina osaaminen on suuntautunut full-stack-osaamiseen. Tämä muuttaa arkkitehdin roolia siten, että vastuu jakautuu tuotantotiimille erillisen arkkitehdin sijaan (Dragičević ja Bošnjak 2019).

Ohjelmistoarkkitehdin roolia kuvaa parhaiten ratkaisuarkkitehdin tai implementaatioarkkitehdin rooli (Erder ja Pureur 2016; Zimmermann 2016; Babar 2009). Arkkitehdilla tulee olla osaamista ja kykyä abstraktoida ja selkeyttää monimutkaisia asioita, sekä riittävästi valtaa tehdä tarvittavia päätöksiä (Poort, Pautasso ja Zimmermann 2016; Buschmann ja Henney 2013). Arkkitehti keskittyy arkkitehtuurin tuottamiseen, ohjelmointiin tarpeiden mukaisesti, teknisen osaamisen ja tiedon jakamiseen sekä tiimin avustamiseen arkkitehtuuria rikkovissa tilanteissa (Babar 2009). Arkkitehdin tulee tuottaa lisäarvoa asiakkaalle, tuotantotiimille ja muille sidosryhmille (Blair, Watt ja Cull 2010; Faber 2010). Arkkitehti on osa sidosryhmää laadun osalta (Blair, Watt ja Cull 2010). Arkkitehti keskittyy laadullisien vaatimuksien toteutumiseen ja tuotantotiimi toiminnallisien vaatimuksien toteutumiseen (Faber 2010).

Suunnitelmalähtöisissä menetelmissä arkkitehdilla on selkeä rooli (Martensson ym. 2019). Ketterissä projekteissa arkkitehdit saattavat jäädä jälkeen kehittäjistä, jolloin viestintään syntyvää kuilua on haastavaa korjata (Martensson ym. 2019). Waterman (2018a) mukaan paras lopputulos voidaan saavuttaa tuotantotiimien ja ohjelmistoarkkitehtien yhteistyöllä. Ketterät tiimit tuottavat nopeasti ja tehokkaasti lisäarvoa, mutta eivät välttämättä huomioi tietoturvallisuutta, monitorointia tai integraatioita, jotka taas kuuluvat arkkitehdin osaamisalueelle (Waterman 2018a).

Arkkitehdin pitää pystyä ajattelemaan rakennetta ja teknologiaa pidemmälle, huomioiden organisaatiorakenteen sekä infrastruktuurin (Nord, Ozkaya ja Kruchten 2014), keskittyen kuitenkin ihmisiin, joihin suunnittelupäätökset vaikuttavat (Buschmann 2012).

Arkkitehti tarvitsee viestintätaitojen (Faber 2010) lisäksi myös johto- ja hallinnollisia taitoja (Babar 2009; Buschmann 2012). Arkkitehti voi toimia roolissa (esimerkiksi Scrum Master), jossa arkkitehti poistaa esteitä, jotka haittavat tiimin ketterää toimintaa (Schwaber 2004; Buschmann ja Henney 2013) tai sidosryhmiä (Mirakhorli ja Cleland-Huang 2013).

Arkkitehdin rooli ketterissä projekteissa voidaan jakaa seuraavasti (Rost ym. 2015):

- arkkitehti on tiimiin nimetty henkilö,
- koko tiimi vastaa arkkitehdin tehtävistä ilman erillistä arkkitehtia,
- ketterän tiimin sisäinen arkkitehtuuritiimi tai
- ulkoinen erillinen arkkitehtuuritiimi.

5 Tutkimuksen toteutus

Tässä luvussa käydään läpi tutkimuksen tavoitteet, toteutus ja käytetyt tutkimusmenetelmät.

5.1 Tutkimuksen tavoite ja tutkimuskysymykset

Tutkimuksen tavoitteena oli kartoittaa arkkitehtuurisuunnittelun tilaa suomalaisissa yrityksissä ja selvittää, miten yrityksissä käytetään ohjelmistoarkkitehtuuria tuotannon tukena ja ohjauksessa. Koska aihe oli laaja, tehtävän tutkimuksen pohjana käytettiin eri tulokulmia: Avoimen lähdekoodin tai ketterien menetelmien käyttö. Avoimen lähdekoodin ja ketterien menetelmien vaikutusta arkkitehtuuriin selvitettiin dokumentaation laajuuden ja olemassaolon kautta.

Tutkimuksen kohderyhmä oli tietotekniikka-alalla ohjelmistokehityksen parissa suomalaisissa yrityksissä työskentelevät henkilöt. Yritykset voivat olla myös kansainvälisiä, kunhan yrityksellä oli toimipiste Suomessa, jossa vastaaja työskentelee. Kohderyhmään kuuluivat ohjelmistoarkkitehdit ja ohjelmistokehittäjät, alalla työskentelevät projektipäälliköt ja konsultit sekä johtajat.

Tutkimusaineistoa tarkasteltiin arkkitehtuurin näkökulmasta. Tutkimus ei ota kantaa ohjelmistokehityksen tai dokumentaation laatuun, vaikka tutkimuksessa tutkittiin dokumentaation vaikutusta arkkitehtuuriin.

5.2 Tutkimusote

Tutkielman tutkimusote on laadullinen kuvaileva tutkimus. Laadullisella analyysilla oli mahdollista saada tietoa ilmiön olemuksesta sekä löytää uusia näkökulmia. Laadullinen tutkimus voi olla joko teorialähtöinen, teoriasidonnainen tai aineistolähtöinen (Eskola 2019, s. 180). Tämä tutkimus oli aineistolähtöinen eli tutkimuksen pääpaino on aineistossa. Tämä tarkoittaa, että esimerkiksi analyysiyksiköt eivät ole ennalta määrättyjä (Saaranen-Kauppinen ja Puusniekka 2006).

Aineiston tulkinta voi olla joko induktiivinen (yksittäisestä yleiseen), deduktiivinen (yleisestä yksittäiseen) tai abduktiivinen (lopputuloksesta lähtevä) (Tuomi ja Sarajärvi 2019, s. 80). Tämän tutkimuksen sisällönanalyysin lähestymistapa oli induktiivinen. Induktiivisen lähestymistavan lähtökohtana ei ole teorian tai hypoteesien testaaminen (Hirsjärvi 2009, s. 155). Puhdas induktiivinen päättely ei ole mahdollista, koska se perustuu pelkkään havaintojen kuvaamiseen ilman ennakkokäsityksiä tutkittavasta ilmiöstä (Tuomi ja Sarajärvi 2019, s. 81). Tästä syystä aineistolähtöisyys ja induktiivinen päättely vaativat tutkijalta itsekuria aineistossa pysyttelemisessä, sekä ennakkokäsitysten ja teorioiden poissulkemisessa. Tutkijan tulee reflektoida tutkimuksen toteuttamistapaa ja monistettavuutta sekä arvioida tutkimuksen luotettavuutta.

5.3 Kyselytutkimus

Tutkimuksen aineiston kerääminen toteutettiin kyselytutkimuksena. Kyselytutkimus toteutettiin anonyymilla sähköisellä kyselylomakkeella eli internetkyselyllä. Kyselylomake on yksi perinteisimmistä keinoista kerätä tutkimusaineistoa. Kyselytutkimus valittiin siksi, että sitä pidettiin kohderyhmälle sopivampana esimerkiksi haastattelun sijaan. Internetkysely oli luonteva vaihtoehto kohderyhmälle, joka tekee näyttöpäätetyötä. Internetkysely oli vaivaton ja edullinen toteuttaa muihin vaihtoehtoihin verrattuna. Internetkyselystä voitiin toteuttaa täysin anonyymi. Anonyymi tieto ei ole luonteeltaan henkilötietoa, eikä sitä koske yksityiselämän suojaksi säädetyt salassapitovelvoitteet (Komulainen 2018, s. 1). Tässä tutkimuksessa ei sovellettu EU:n yleistä tietosuoja-asetusta, koska asetuksen siirtymäaika ei ollut vielä aineistoa kerätessä päättynyt.

Kysymykset rakennettiin tutkimuksen tavoitteiden ja tutkimusongelmien mukaisesti (Valli 2019, s. 82). Heikkilä (2014, s. 46) mukaan kyselylomakkeen laatimisen vaiheet ovat:

1. tutkittavien asioiden nimeäminen,
2. lomakkeen rakenteen suunnittelu,
3. kysymysten muotoilu,
4. lomakkeen testaus,
5. lomakkeen rakenteen ja kysymysten korjaaminen ja

6. lopullinen lomake.

Kysely laadittiin tätä tutkimusta varten eli tutkimuksessa ei käytetty valmista instrumenttia. Kysely löytyy liitteestä 1. Kysely koostui strukturoiduista eli suljetuista kysymyksistä sekä avoimista kysymyksistä. Lomake ei sisältänyt kontrollikysymyksiä. Kysymykset olivat luonteeltaan Eskola (1975, s. 178-181) luokittelun mukaisesti täsmällisiä tosiasiatietoja sekä asenteisiin, arvoihin ja mielipiteisiin liittyviä kysymyksiä. Täsmällisiin tosiakysymyksiin vastaaminen edellyttää faktatietoa, ei mielipiteitä tai käsityksiä. Asenteita, arvoja ja mielipiteitä mitattaessa on korostettava sitä, että vastaajat vastaavat kysymyksiin sen mukaan, mitä asiasta todella ajattelevat.

Kyselyn alussa kysyttiin vastaajaa ja vastaajan omaa yritystä tai työnantajayritystä profiloivia kysymyksiä. Seuraavaksi kysyttiin yrityksen sisäisistä toimintatavoista, avoimen lähdekoodin ja ketterän kehityksen käytöstä ja lopuksi kysymyksiä siitä, miten merkitykselliseksi vastaaja kokee arkkitehtuurisuunnittelun. Lomakkeeseen valitut ohjelmisto- ja arkkitehtuurisuunnittelun lähestymistavat valittin ajankohtaisuuden mukaan. Valikointi tapahtui selvittämällä mitä lähestymistapoja yleisesti on olemassa ja käytössä alan kirjallisuuden perusteella. Lähestymistapojen yleisyyttä arvioitiin selvittämällä yrityksien sisäisiä prosesseja esimerkiksi blogikirjoituksien, webinaarien, keskusteluiden ja YouTube-videoiden avulla.

Kysymysten määrä pidettiin maltillisena ja vastaustapaa rajattiin mahdollisimman vähän tarjoamalla myös avoimia tekstikenttiä vastaamista varten.

Otannan onnistuminen on keskeinen tekijä, mikäli pyritään yleistämään tutkimuksessa saatuja tuloksia perusjoukkoon eli populaatioon. Kyselyn kohderyhmä oli ohjelmointityötä tekevät sekä ohjelmoinnin rajapinnassa työskentelevät henkilöt.

5.4 Aineiston keruu

Tutkimuksen aineisto kerättiin kyselyllä, joka oli avoimesti vastattavana internetissä 8.2.2018 - 31.3.2018. Tänä aikana vastauksia saatiin 38 kappaletta. Yhden vastaajan vastauksia ei huomioitu osana aineistoa. Vastaaja työskenteli käyttöliittymäsuunnittelijana, joten vastaus poistettiin aineistosta analysointivaiheessa.

Vastaajista 11 henkilöä työskenteli ohjelmistosuunnittelun parissa. 15 henkilöä teki ohjelmistokehitystä. 4 henkilöä olivat konsultteja. 2 henkilöä tekivät projektinhallintaa ja 5 henkilöä olivat johtotehtävissä.

Kyselyn vastauksissa oli mahdollisuus saada useita vastauksia samoista yrityksistä, mutta vastauksissa olleissa yrityksen kokoluokissa, perustamisvuosissa ja toimialoissa ei ollut päällekkäisyyksiä. Tästä voidaan päätellä, että samoista yrityksistä ei tullut päällekkäisiä vastauksia.

Taulukossa 1 on eritelty yritysten henkilöstön lukumäärä ja vastaajamäärät. Vastaajista yksi ei halunnut ilmoittaa yrityksen kokoa. Yrityksien projektien keskimääräinen kesto oli 24 kuukautta. Projektien keston mediaani oli 9 kuukautta.

Yrityksen henkilöstö	**Lukumäärä**
11-50	13
yli 200	11
10 tai alle	6
101-200	5
51-100	1

Taulukko 1. Yrityksen koot ja vastaajamäärät

Kysely tavoitti arviolta 50 000 ihmistä useissa eri kanavissa, joista 1414 siirtyivät itse kyselyyn. Kyselyn linkkiä jaettiin saateteksteineen LinkedInissä, Slack-ryhmissä, sähköpostitse sekä keskustelupalstoilla.

Lomaketta katsottiin siis 1414 kertaa. Lomakkeen konversioista eli tavoitteista tarkasteltiin lomakkeen latauskertoja sekä lomakelähetyksiä. Lomakeen avauksien konversioprosentit on esitetty kanavakohtaisesti taulukossa 2. Lomakkeen lähetyksen konversio oli 2,7%. Konversioprosentti = (konvertoituneiden vierailijoiden määrä / kaikkien vierailijoiden määrä) x 100.

Lähde	Tavoittavuus	Klikkauskerrat	Klikkauksien konversio
Slack-kanavat[1]	1 637	677	41%
Twitter	2 357	43	1,8%
Facebook[2]	Ei tiedossa	292	-
Ohjelmointiputka.net	6550[3]	21	Minimi 0,3%
io-tech.fi	38 513[3]	27	Minimi 0,07%
Muut lähteet	Ei tiedossa	354	-

Taulukko 2. Lomakkeen latauskertojen konversioprosentit eri kanavissa

Io-tech.fi ja Ohjelmointiputka.net sivustoilta ei ole saatavilla tietoa, kuinka monta ihmistä kysely tavoitti, ainoastaan klikkauskerrat.

5.5 Aineiston analysointimenetelmä

Sisällönanalyysin avulla pyritään muodostamaan tiivistetty kuvaus tutkittavasta ilmiöstä, mikä kytkee tulokset ilmiön laajempaan kontekstiin ja aihetta koskeviin muihin tutkimustuloksiin (Tuomi ja Sarajärvi 2019, s. 85-88).

Aineiston analyysointiin ei ole muodostunut mitään tiettyjä strukturoituja vaiheita. Tässä tutkimuksessa noudatettiin seuraavaa vaiheistusta:

1. Redusointi
2. Klusterointi
3. Abstrahointi

Vaiheistus pohjaa soveltaen Tuomen ja Sarajärven (2019) esittämiin vaiheisiin.

Sisällönanalyysin ensimmäinen vaihe oli redusointi eli pelkistäminen. Aineistosta karsittiin pois tutkimukselle epäolennainen sisältö. Tässä tutkimuksessa tämä toteutettiin tiivistämällä,

1. Koodiklinikka ja Koodia Suomesta, kanava #random
2. Ohjelmointiryhmät
3. Luku perustuu jäsenmäärään, todellinen tavoittavuus ei tiedossa, ainoastaan klikkauksien määrät.

yksinkertaistamalla ja karsimalla dataa. Datasta karsittiin pois ne tiedot, jotka eivät vastanneet tai liittyneet tutkimuskysymyksiin. Dataa yksinkertaistettiin siten, että samantyyppiset vastaukset yhdistettiin, jotta niiden pohjalta voitiin piirtää kaavioita ja tehdä päätelmiä.

Seuraavassa vaiheessa aineisto ryhmiteltiin. Aineistosta etsittiin yhtäläisyyksiä ja eroavaisuuksia. Ryhmittely tehtiin sen mukaan mihin tutkimuskysymykseen mikäkin vastaus vastaa. Klusteroinnin pohjalta saatiin alustavat kuvaukset tutkittavasta ilmiöstä.

Klusterointia seurasi aineiston abstrahointi, eli käsitteellistäminen. Abstrahoinnissa edettiin alkuperäisdatan käyttämistä ilmaisuista teoreettisiin käsitteisiin ja johtopäätöksiin. Abstrahointi tehtiin siten, että alkuperäisdatan konteksti ei katoa.

Analysoinnissa käytettiin tutkimuskysymyksien mukaisia tulokulmia, joita olivat i) avoin lähdekoodi ja ii) ketterät menetelmät. Tulokulmista muodostettiin seuraavat kategoriat:

1. avoimen lähdekoodin käyttämisen vaikutus arkkitehtuurisuunnitteluun ja dokumentaation määrään ja
2. ketterien menetelmien käyttämisen vaikutus arkkitehtuurisuunnitteluun ja dokumentaation määrään.

Kategorioita yhdisteltiin seuraavien muuttujien mukaisesti:

1. arkkitehtuurin tärkeys,
2. dokumentaation tärkeys,
3. laadittavien dokumenttien määrä,
4. arkkitehtuurin dokumentointi,
5. arkkitehtuurisuunnitteluun varattujen resurssien määrä ja
6. dokumenttien ajantasaisuus projektin päättyessä.

Vaikuttavia dokumentteja olivat projektisuunnitelma, vaatimusmäärittely ja arkkitehtuuri, joiden voidaan tulkita olevan tavanomaisia dokumentteja projektin tyypistä tai toteutustavasta huolimatta.

Jokaista muuttujaa ristiintaulukoitiin vastakategoriaan, eli niihin yrityksiin, joissa ei käytetä avointa lähdekoodia tai ketteriä menetelmiä. Tällä analyysilla saatiin selville miten tulokul-

mat vaikuttavat arkkitehtuuriin.

Yksinkertaistukset ja teemaluokat

Aineiston yksinkertaistukset ja teemaluokat toteutettiin siten, että aineistoon jäi tutkimuskysymykseen vastaava sisältö. Aineistossa oli samantyyppisiä vastauksia, jotka yksinkertaistettiin ja luokiteltiin kaavioita varten. Yksinkertaistukset tehtiin jokaisen tutkimuskysymyksen pohjalta. Luokka määrittää piirretyn kaavion tai taulukon. Yksinkertaistusta varten taulukoitiin kysymys, jolla aineistoa hankittiin kyselyssä, vastaus, yksinkertaistus sekä luokka. Taulukoissa 3, 4 ja 5 on eritelty eri yksinkertaistukset luokkineen tutkimuskysymyksittäin. Vastauksien avulla pyrittiin löytämään yhteisiä tekijöitä eri vastauksien välillä ja myös arvioimaan vastaajan näkemystä ja kokemusta tutkittavasta aihepiiristä.

Tutkimuskysymys 1: Miten arkkitehtuuria käytetään yrityksissä?			
Kysymys, jolla aineistoa hankittiin	**Esimerkki vastauksesta**	**Yksinkertaistus**	**Luokka**
Mitä työkaluja tai ohjelmia käytätte arkkitehtuurikuvauksien tekemiseen?	”Riippuu niin projektista, mutta ihan vapaamuotoisesti”	Muu	Käytetty työkalu
	”Tapauskohtaista, piirto- ja kirjoitusohjelmat”	Muu	Käytetty työkalu
	”Word”, ”Powerpoint” tai ”Visio”	Word, Powerpoint ja Visio	Käytetty työkalu
	”useita”	Muu	Käytetty työkalu
Mitä notaatiota käytätte arkkitehtuurikuvauksissa?	”Vapaamuotoiset diagrammit”	Yrityksen sisäisesti sovittu notaatio	Käytetty notaatio
	”Kontekstiin sopivaa. Riippuu kirjoittajasta ja lukijasta.”	Yrityksen sisäisesti sovittu notaatio	Käytetty notaatio
	”Maalaisjärki”	Muu	Käytetty notaatio

Taulukko 3. Tutkimuskysymykseen 1 liittyvät yksinkertaistukset ja teemaluokittelut (osa 1)

Tutkimuskysymys 1: Miten arkkitehtuuria käytetään yrityksissä?			
Kysymys, jolla aineistoa hankittiin	**Esimerkki vastauksesta**	**Yksinkertaistus**	**Luokka**
Miten koet arkkitehtuurin auttavan projektin eri vaiheissa?	”runko”, ”yhteinen suunta”, ”yhteinen linja”, ”projektin kulmakivi”	Runko, yhteinen linja tai suunta	Arkkitehtuurin tarkoitus
	”pitää olla joustava”, ”arkkitehtuuri ja linjanvedot helpottavat jatkokehitystä ja ylläpitoa”, ”arkkitehtuuri helpottaa ylläpitoa ja ongelmien selvitystä”, ”muutokset arkkitehtuuriin täytyy sallia”	Joustavuus, jatkokehitettävyys	Arkkitehtuurin ominaisuudet
Mitä ongelmia arkkitehtuurisuunnittelussa esiintyy?	”Arkkitehtuuri on koodia ja scriptiä”	Lähdekoodi	Arkkitehtuurin merkitys
	”ei voida jakaa omaksi vaiheekseen projektin alkuun, eikä selkeää rajaa vetää siihen missä arkkitehtuuri alkaa tai päättyy”	Suunnittelu ja toteutus	Arkkitehtuurin merkitys

Taulukko 4. Tutkimuskysymykseen 1 liittyvät yksinkertaistukset ja teemaluokittelut (osa 2)

Tutkimuskysymys 3: Miten arkkitehtuuri suunnitellaan ja dokumentoidaan, kun käytetään ketteriä malleja?			
Kysymys, jolla aineistoa hankittiin	**Esimerkki vastauksesta**	**Yksinkertaistus**	**Luokka**
Kerro lyhyesti prosessi/kehitysmallistanne	”Vesiputous/kanban malli johon vähän haetaan vivahteita scrumista.”	Yrityksessä käytetään ketteriä menetelmiä	Projektimalli

Taulukko 5. Tutkimuskysymykseen 3 liittyvät yksinkertaistukset ja teemaluokittelut

6 Tulokset

Tässä luvussa käydään läpi tutkimustulokset liittyen ohjelmistoarkkitehtuuriin suomalaisissa yrityksissä. Lisäksi käsitellään avoimen lähdekoodin ja ketterien menetelmien käytön vaikutuksia arkkitehtuuriin ja dokumentaation ajantasaisuuteen.

6.1 Arkkitehtuuri suomalaisissa yrityksissä

Pääsääntöisesti kyselyyn vastanneet kokivat arkkitehtuurin tärkeäksi osaksi projektia ja dokumentaatiota. Asteikolla 1-10 arkkitehtuurin dokumentaation laatimisen tärkeydeksi annettiin 8.2. Yleisesti dokumentaation tärkeys arvioitiin tasolle 6.9.

Notaationa käytettiin pääosin yrityksen sisäisesti sovittua notaatiota tai UML-mallinnuskieltä. Yrityksen sisäisesti sovittu notaatio oli käytössä 51%:ssa ja UML 31%:ssa yrityksistä.

Käytetyissä työkaluissa oli paljon hajontaa tussitaulusta Powerpointiin, kuten taulukosta 6 käy ilmi. Kaksi eniten käytettyä olivat Draw.io-verkkopalvelu sekä Microsoft Visio. Draw.io-verkkopalvelua käytetään yleensä Confluencen ja Jiran kanssa (draw.io 2020). Vastaajista noin 26% käytti jotain muuta työkalua erittelemättä työkalun nimeä. Vastaajista 42% oli integroinut arkkitehtuuridokumentaation automaattisen päivittymisen osaksi prosessia.

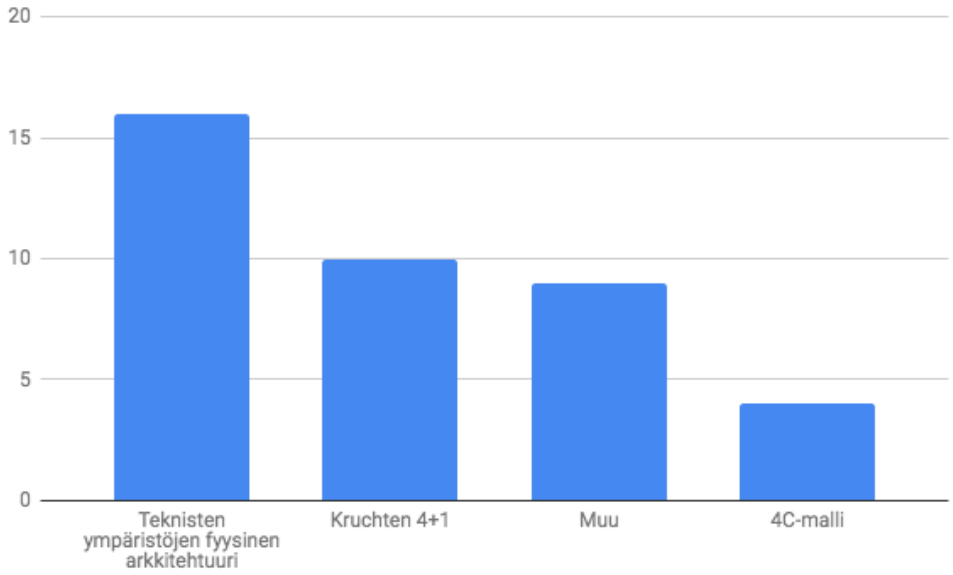
Työkalu tai ohjelma	**kpl (n=30)**
Word, PowerPoint ja Visio	10
Muu	9
Draw.io	6
ArchiMate	1
Gliffy	1
MagicDraw	1
Sparx Enterprise Architect	1
Valkotaulu + tussi	1

Taulukko 6. Käytetyt työkalut ja ohjelmat arkkitehtuurin mallintamisessa

Arkkitehtuurikuvauksia tehtiin eniten teknisen ympäristöjen fyysisen arkkitehtuurin mukaan, mutta myös Kruchten 4+1 sekä 4C-malli nousivat esille, kuten käy ilmi kuviosta 2. Vastaajista 23% käytti jotain muuta näkökulmaa erittelemättä näkökulmaa nimeltä. Näkökulmista oli myös hybridimalleja, joissa yhdistettiin infrastruktuurin arkkitehtuuri johonkin toiseen näkökulmaan tarpeen mukaan. Näkökulmien käyttöä ja arkkitehtuurin dokumentaation määrää kuvattiin seuraavasti:

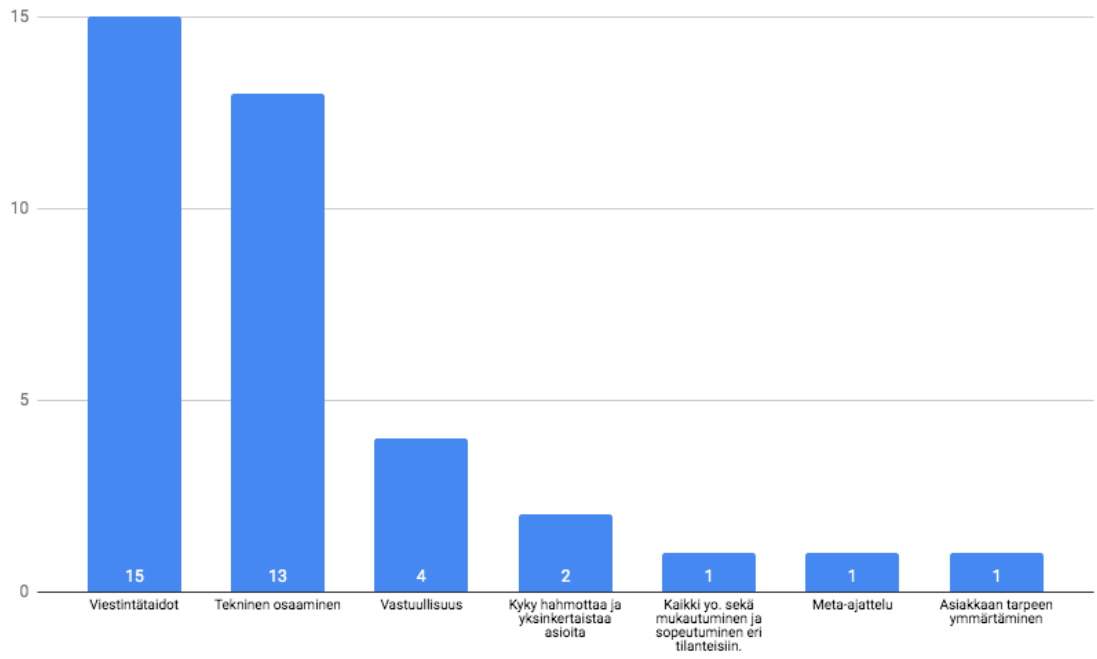
> *”Ei ole kovin muodollista lähestymistapaa. Dokumentoidaan asiat jotka nähdään tarpeelliseksi dokumentoida”* (vastaaja 7)

> *”arkkitehtuurissa kuvataan järjestelmän toiminnallisuus ja riippuvuudet muihin järjestelmiin”* (vastaaja 4)



Kuvio 2. Käytetyt näkökulmat arkkitehtuurin mallintamisessa

Arkkitehdin tärkeimmiksi ominaisuudeksi nousivat viestintätaidot sekä tekninen osaaminen, kuten käy ilmi kuviosta 3.



Kuvio 3. Arkkitehdin tärkein ominaisuus

Arkkitehtuurin koettiin kuvaavan kehitettävän järjestelmän runkoa, isoa kuvaa, yhteistä linjaa tai kulmakiveä. Hyvän arkkitehtuurin koettiin myös mahdollistavan monia laadullisia vaatimuksia, kuten tietoturvan tai suorituskyvyn.

Arkkitehtuuri koettiin hyödylliseksi kehittäjien näkökulmasta. Arkkitehtuurista kerrottiin, että sen avulla *”tietää joka vaiheessa millainen palikan pitää olla”* (vastaaja 1) ja *”koodaaminen on helpompaa kun suuri kuva on selvillä”* (vastaaja 2). Kehittäjät hahmottavat kokonaisuuden ja tarpeet paremmin arkkitehtuurin kuin liiketoiminnallisen tarpeen kautta.

Arkkitehtuurisuunnittelun ongelmaksi koettiin projektin alkuvaiheen ei-tiedossa olevat vaatimukset, tarpeet tai muutokset. Alussa tehdyt arkkitehtuuripäätökset saattavat muodostaa teknisiä rajoitteita toteutukselle projektin myöhemmässä vaiheessa. Tämän koettiin joissain tapauksissa johtavan laatuongelmiin tai arkkitehtuurin oikomiseen ja rikkoutumiseen. Vastauksissa korostui tarve minimiarkkitehtuurille, mutta kuitenkin on tärkeää, että arkkitehtuuri ei jää liian ylätasoiseksi tai epäselväksi. Arkkitehtuurilta vaaditaan joustavuutta ja tästä syystä arkkitehtuurin toivottiin kattavan lähinnä ne ratkaisut ja teknologiavalinnat, joita myöhemmin ei voida enää muuttaa.

Arkkitehdiltä odotetaan projektin edetessä viestintää asiakkaan ja toteuttavan tiimin välillä, sekä kykyä tarkentaa arkkitehtuuria vaatimusten tai tarpeen tarkentuessa ja muuttuessa.

Viestintään liittyviä ongelmia tunnistettiin paljon. Nämä liittyivät pääosin projektin alkuvaiheen ei-tiedossa oleviin vaatimuksiin, päätöksiin ja rajoituksiin. Näiden lisäksi oli myös väärinymmärryksiä ja -käsityksiä asiakkaan tarpeista. Tarpeisiin liittyviä ongelmia kuvattiin seuraavasti:

> *"On ymmärretty väärin tai puutteellisesti mikä on asiakkaan lopullinen tarve, jolloin väärä arkkitehtuurivalinta voi tuottaa suuria ongelmia myöhemmin."* (vastaaja 3)

Arkkitehtuurin tulee olla niin helppo, että myös asiakas ymmärtää. Asiakasta ja sidosryhmää on hankala sitouttaa projektin suunnitteluun. Referenssiarkkitehtuureita voidaan testata MVP:n (Minimum Viable Product) tai PoC:n (Proof of Concept) kautta ennen toteutusta.

Vastauksissa kävi ilmi, että ohjelmiston elinkaari saattaa unohtua projektin toteutusvaiheessa. Kehityksessä ei huomioida, että ylläpitovaihe saattaa kestää vuosia. Perinteisesti elinkaariajattelu on kuulunut arkkitehdin tehtäviin. *Legacy*, eli perintökoodi, ja refaktorointi rajoittavat uuden kehittämistä. Elinkaareen liittyviä ongelmia kuvattiin esimerkiksi kertomalla:

> *"Keskitytään vain siihen että saadaan kehitysprojekteissa ja ensimmäinen versio syntymään. Unohdetaan se että edessä on kuitenkin myös vuosia kestävä ylläpitövaihe."* (vastaaja 9).

Arkkitehtuuri tunnistettiin tekijäksi, jolla voidaan helpottaa jatkokehitystä ja ylläpitoa. Yhteinen linja arkkitehtuurin kautta helpottaa myös uusien henkilöiden tuomista mukaan projektiin.

Ohjelmistostoarkkitehtuurin käsite oli monitulkintainen. Osa koki arkkitehtuurin olevan suunnittelupäätöksiä, osa lähdekoodia, ja osa taas koki, että ohjelmistoarkkitehtuurin rajanvetoa suunnittelun ja toteutuksen välillä on hankalaa tehdä.

Ohjelmistoarkkitehtuuri on kustannuskysymys. Lyhytkestoisissa projekteissa ei koettu olevan mielekästä kuluttaa aikaa arkkitehtuuriin. Arkkitehtuuria ei välttämättä lähdetty muutta-

maan projektin edetessä, vaikka sille olisi tunnistettu tarve, jotta projektin laajuus ei muuttuisi ja aiheuttaisi lisäkustannuksia. Tämä kävi ilmi vastauksesta:

> *”Toteutusvaiheessa arkkitehtuuria ei lähdetä oikein herkästi muuttamaan enää, koska tällöin projektin ”scope” (eli laajuus) muuttuu ja yleensä tulee vastaan taloudelliset seikat, mitkä estävät tämän.”* (vastaaja 4)

Laajuuden muuttuminen voi aiheuttaa scope creep -ilmiötä, jossa vaatimukset ja kustannukset muuttuvat merkittävästi ja hallitsemattomasti siitä, mitä tilaaja ja toimittaja ovat alunperin sopineet (Amoatey ja Anson 2017). Ohjelmistoarkkitehtuuri tunnistettiin myyntivaiheen työmääräarviointia ja tarjouslaskentaa tukevaksi tekijäksi. Arkkitehtuurin merkitystä myynnissä kuvattiin seuraavasti:

> *"osana myyntitarjouksia suunnitellaan miten projektia kannattaisi toteuttaa ja tästä seuraa korkean tason arkkitehtuurisuunnitelma, jossa valitaan teknologiat – ja lähestymistavat"* (vastaaja 10)

Arkkitehtuurin ja dokumentaation puuttuminen, keskeneräisyys ja ajantasaisuus oli haaste. Vastauksista kävi ilmi, että ilman automaatiota dokumentaatio vanhenee itsestään. Yksi vastaajista oli tunnistanut haasteen myös osaamisen osalta:

> *”Vain harva osaa tehdä sitä systemaattisesti ja mallintamisen taito on ketteryyden korostuessa unohtunut.”* (vastaaja 8)

6.2 Avoin lähdekoodi ja dokumentaatio

Vastaajista 43% työskenteli yrityksissä, joiden liikevaihto koostuu pääosin avoimen lähdekoodin päälle tehdyistä toteutuksista.

Arkkitehtuurin tärkeys asteikolla 1-10 arvioitiin tasolle 7.9 niissä yrityksissä, jotka hyödyntävät avointa lähdekoodia. Muissa yrityksissä taso oli 8.1. Yleisesti dokumentaation tärkeys arvioitiin tasolle 6.8 niissä yrityksissä, jotka hyödyntävät avointa lähdekoodia. Muissa yrityksissä taso oli 6.7. Yleisesti arkkitehtuurin ja dokumentaation tärkeys oli samaa tasoa molemmissa ryhmissä.

Taulukossa 7 on esitetty dokumentaation taso ja ajantasaisuus projektin päättyessä. 50%:ssa avointa lähdekoodia hyödyntävien yritysten työntekijöiden vastauksista laadittiin projektisuunnitelma, arkkitehtuuri ja vaatimusmäärittely. Muissa yrityksissä 43% laati vastaavat dokumentit. Arkkitehtuuri dokumentoitiin 81%:ssa, kun muissa yrityksissä niin tehtiin vain 62%:ssa. Arkkitehtuuri ja dokumentaatio on myös useammin ajan tasalla projektin päättyessä mikäli käytetään avointa lähdekoodia.

	Avointa lähdekoodia pääosin hyödyntävät yritykset	**Muut yritykset**
Dokumentoi vähintään arkkitehtuurin	81%	62%
Arkkitehtuuri on ajan tasalla projektin päättyessä	56%	52%
Dokumentoi projektisuunnitelman, vaatimusmäärittelyn ja arkkitehtuurin	50%	43%
Dokumentit ovat ajan tasalla projektin päättyessä	31%	24%

Taulukko 7. Dokumentaation taso ja ajantasaisuus

Avoimen lähdekoodin päälle toteutetuissa projekteissa varattiin enemmän resursseja arkkitehtuurin dokumentointiin, kuten käy ilmi taulukosta 8.

Projektin resurssien käyttö arkkitehtuuriin (%)	**Avointa lähdekoodia pääosin hyödyntävät yritykset**	**Muut yritykset**
0-20	8	17
21-40	3	1
41-60	1	1

Taulukko 8. Projektin resurssien käyttö arkkitehtuuriin (% projektin kokonaisresursseista)

On selkeästi havaittavissa, että dokumentaatio on kattavampi silloin, kun hyödynnetään avointa lähdekoodia. Tämä voi johtua siitä, että ilman kattavaa dokumentaatiota on hankalampi vä-

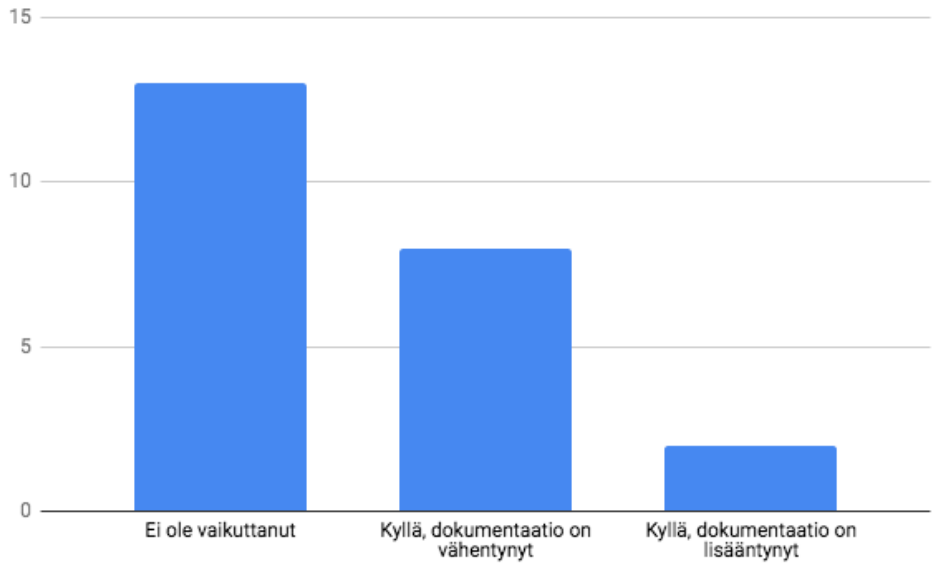
hentää teknistä velkaa tai saada jatkokehityksessä tulevia vaatimuksia arkkitehtonisesti yhteensopivaksi. Avoimen lähdekoodin käyttö ei kuitenkaan merkittävästi vaikuta siihen, koetaanko arkkitehtuuri tai dokumentointi tärkeäksi. Avointa lähdekoodia hyödyntävissä projekteissa arkkitehtuuriin varattavat resurssit ovat suuremmat kuin muissa yrityksissä.

6.3 Ketterät menetelmät ja dokumentaatio

Ketteriä menetelmiä hyödynsivät lähes kaikki vastanneista (91%). Pääsääntöisesti vastaajat kokivat, että ketterät menetelmät eivät joko ole vaikuttaneet dokumentaation määrään, tai ovat vaikuttaneet siihen vähentävästi, kuten käy ilmi kuviosta 4. Ketterien menetelmien vaikutusta dokumentaatioon kuvattiin seuraavasti:

> *”jotkut perustelevat dokumentoinnin puutetta ketterällä kehittämisellä”* (vastaaja 7)

> *”Dokumentaatiota ei enää osata tehdä kun kehitysmenetelmien syväosaaminen on hapattunut.”* (vastaaja 8).



Kuvio 4. Ketterien menetelmien vaikutus dokumentaation määrään

Arkkitehtuurin tärkeys asteikolla 1-10 arvioitiin tasolle 8.41 niissä yrityksissä, jotka käytti-

vät ketteriä menetelmiä. Muissa yrityksissä taso oli 6.81. Ketteriä menetelmiä hyödyntävissä yrityksissä arkkitehtuurin dokumentointi koettiin tärkeämmäksi kuin muu dokumentaatio. Dokumentaation tärkeys arvioitiin tasolle 6.94 niissä yrityksissä, joissa käytettiin ketteriä menetelmiä. Muissa yrityksissä taso oli 6.81. Dokumentaation tärkeydessä ei ollut merkittävää eroa.

Dokumentaation merkitys on erilainen eri vastaajille. Dokumentaation syntymistä suoraan lähdekoodin ja käytettyjen työkalujen avulla kuvattiin seuraavasti:

> ”*Menetelmät tuottavat itsessään dokumentaatioita*” (vastaaja 5)

> ”*laadukas koodi on itsessään dokumentaatio*” (vastaaja 6)

Osa vastaajista oli sulauttanut dokumentaation tuottamisen osaksi jokapäiväistä työtä, kuten käy ilmi vastaajan 7 vastauksesta:

> ”*Sen* (dokumentaation) *laatiminen kuuluu osana jokaisen päivittäiseen tekemiseen.*” (vastaaja 7)

Taulukossa 9 on esitetty dokumentaation taso ja ajantasaisuus projektin päättyessä. Taulukosta käy ilmi, että niissä yrityksissä, jotka eivät käytä ketteriä menetelmiä dokumentoidaan kattavammin ja dokumentaatio on myös useammin ajan tasalla projektin päättyessä. Tutkimuksen otanta oli hyvin pieni niiden yrityksien osalta, jotka eivät käytä ketteriä menetelmiä, joten tästä ei voida suoraan tehdä luotettavaa yleistystä. Ketteriä menetelmiä hyödyntävissä yrityksissä suosituin dokumentti oli projektisuunnitelma, jonka laati 80% yrityksistä.

	Ketteriä menetelmiä hyödyntävät yritykset	**Muut yritykset**
Dokumentoi arkkitehtuurin	66%	100%
Arkkitehtuuri on ajan tasalla projektin päättyessä	29%	67%
Dokumentoi projektisuunnitelman, vaatimusmäärittelyn ja arkkitehtuurin	40%	100%
Dokumentit ovat ajan tasalla projektin päättyessä	23%	67%

Taulukko 9. Dokumentaation taso ja ajantasaisuus ketteriä menetelmiä hyödynnettäessä

Arkkitehtuurin suunnitteluun varattujen resurssien määrä oli pääosin 0-20% projektin kokonaisresursseista. Myös muiden yritysten osalta resurssien käyttö oli samaa luokkaa, kuten käy ilmi taulukosta 10.

Projektin resurssien käyttö arkkitehtuuriin (%)	**Ketteriä menetelmiä hyödyntävät yritykset**	**Muut yritykset**
0-20	24	2
21-40	4	0
41-60	1	1

Taulukko 10. Projektin resurssien käyttö arkkitehtuuriin ketteriä menetelmiä hyödynnettäessä (% projektin kokonaisresursseista)

Arkkitehtuuriin käytettävien resurssien välillä ei ollut juurikaan eroa siltä osin käytettiinkö ketteriä menetelmiä vai ei. Dokumentaatio oli kattavampi ja ajantasaisempi niissä yrityksissä, joissa ei hyödynnetty ketteriä menetelmiä. Koska ketteriä menetelmiä käytettiin lähes kaikissa yrityksissä, otanta ja luotettavuus jää heikoksi niiden yritysten osalta, joissa ketterät menetelmät eivät ole käytössä.

Pääsääntöisesti vastaajat kokivat, että ketterät menetelmät eivät joko ole vaikuttaneet doku-

mentaation määrään, tai ovat vaikuttaneet siihen vähentävästi, mutta kuitenkin osa vastaajista tunnisti, että ketteriä menetelmiä käytetään perusteena dokumentoinnin puutteille. Ketterät menetelmät tuottavat itsessään dokumentaatiota, kuten tiketit, sprint-suunnittelmat ja käyttöliittymäsuunnitelmat.

Yleisesti voidaan sanoa, että ketteriä menetelmiä hyödyntävät yritykset pitävät arkkitehtuurin dokumentointia tärkeämpänä kuin muun dokumentaation ja se myös laaditaan huomattavasti useammin. Dokumenttien ajantasaisuus projektin edetessä pysyy kuitenkin melko matalana.

7 Pohdinta

Tässä luvussa käsitellään tämän tutkimuksen tulokset ja tuodaan ilmi tutkijan omia pohdintoja tutkimuksesta sekä siinä esiintyvistä ilmiöistä. Lisäksi arvioidaan tutkimuksen luotettavuutta ja pohditaan tarvittavaa jatkotutkimusta.

7.1 Tulokset

Ohjelmistoarkkitehtuurin käyttö yrityksissä

Ohjelmistoarkkitehtuurin määrittely oli tämän tutkimuksen tuloksien mukaan vastaajille vaikeaa. Määrittelyn haasteet näkyivät myös ohjelmistoarkkitehtuuriin liittyvässä tutkimuskirjallisuudessa.

Ohjelmistoarkkitehtuuri on yhdistymässä ohjelmistosuunnitteluun — tämä on pääteltävissä tämän tutkimuksen tuloksista ja tutkimuskirjallisuudesta. Tämän tutkimuksen tulosten mukaan osa vastaajista koki arkkitehtuurin tarkoittavan lähdekoodia. Tutkimuskirjallisuuden ja tämän tutkimuksen pohjalta on havaittavissa, että ketterät menetelmät ovat muuttaneet arkkitehtuurisuunnittelua korkean tason suunnittelusta lähemmäksi, tai osaksi lähdekoodia. Tämän tutkimuksen avointen kysymyksien vastauksista käy ilmi, että arkkitehtuuria lähestytään niin ketterän arkkitehtuurin kuin etukäteissuunnittelun kautta. Tämän tutkimuksen mukaan arkkitehtuuri suomalaisissa yrityksissä on tällä hetkellä suunnitelmalähtöisyyden ja ketterien menetelmien välimaastossa. Suunnitelmalähtöisyys ei ole ketterien arvojen tai periaatteiden mukaista. Tähän voi omalta osaltaan vaikuttaa myös projektin hankintaprosessi tai hinta, joka voi johtaa siihen, että korkean tason arkkitehtuuri kuvataan jo tarjousvaiheessa, kuten käy ilmi tämän tutkimuksen tuloksista. Omalta osaltaan arkkitehtuurin muuttumiseen ovat tutkimuskirjallisuuden mukaan vaikuttaneet myös pilvipalvelut, jotka ovat vähentäneet arkkitehtuurisuunnittelun tarvetta, koska osa teknisistä päätöksistä voidaan ulkoistaa. Pilvipalvelut eivät nousseet esille vaikuttavana tekijänä tässä tutkimuksessa, eikä niiden vaikutuksesta arkkitehtuuriin myöskään kysytty suoraan.

Ketterän arkkitehtuurin suunnittelun haasteet ovat tutkimuskirjallisuuden ja tämän tutkimuk-

sen tulosten pohjalta alkuvaiheen tuntemattomat vaatimukset ja tarpeet. Alussa tehdyt suunnittelupäätökset muodostavat rajoituksia projektin elinkaaren aikana. Tämän tutkimuksen mukaan elinkaariajattelu on haastavaa. Tämä voi johtua siitä, että perinteisesti arkkitehti on huolehtinut elinkaariajattelusta (Erder ja Pureur 2016) ja tutkimuskirjallisuuden mukaan ketterissä menetelmissä arkkitehdille kuuluvia tehtäviä ei välttämättä hoida kukaan. Kysyttäessä arkkitehdin tärkeimmästä ominaisuudesta, yksikään vastaaja ei vastannut, että arkkitehti olisi tarpeeton.

Tutkimuskirjallisuuden mukaan ohjelmistoarkkitehtuurin tehtävä on varmistaa sidosryhmiltä tulevien laadullisten vaatimusten toteutuminen. Ohjelmistoarkkitehtuurin vaikutus sidosryhmiin, tai sen hyödyntäminen työvälineenä sidosryhmätyöskentelyssä, ei noussut esille tämän tutkimuksen pohjalta. Vain yksi vastaaja toi esille arkkitehtuurin vaikutuksen laatuun. On mahdollista, että tutkimuskirjallisuudesta nouseva arkkitehtuurin merkitys laatuattribuuttien toteutumisen varmistajana ei ole nykyaikainen.

Tutkimuskirjallisuuden mukaan arkkitehtuurisuunnittelun tarve tulisi skaalata projektin laajuuteen. Tähän tutkimukseen vastanneet henkilöt työskentelivät pääosin yrityksissä, joissa oli alle 200 työntekijää ja projektien mediaanikesto oli 9 kuukautta. Näillä tuloksilla ei vielä voida tehdä yleistystä, minkä kokoisissa projekteissa arkkitehtuurisuunnittelua tulisi tehdä. Yrityksen koko tai projektin kesto kuukausissa ei myöskään välttämättä korreloi projektin laajuuteen. Vastaajat 11 ja 12 käyttivät suurille organisaatioille tarkoitettua SAFe-viitekehystä, jossa arkkitehteja on kolme; kokonais-, ratkaisu- ja järjestelmäarkkitehti (“Agile Architecture in SAFe - Scaled Agile Framework” 2020). Tästä huolimatta vastaaja 11 kertoi arkkitehtuurin ongelmaksi, että ”Sitä ei tehdä tai se on projektissa työskenteleville epäselvä”.

Avoimen lähdekoodin vaikutus ohjelmistoarkkitehtuuriin

Vaikka arkkitehtuurin tärkeys arvioitiin matalammaksi, tämän tutkimuksen tulosten mukaan dokumentaatio oli kattavampi silloin, kun hyödynnettiin avointa lähdekoodia. Myös arkkitehtuuri dokumentoitiin ja se oli projektin päättyessä ajan tasalla useammin. Tämän tutkimuksen mukaan avointa lähdekoodia hyödyntävissä projekteissa arkkitehtuuriin varattavat resurssit olivat suuremmat kuin muissa yrityksissä.

Tämän tutkimuksen pohjalta ei vielä voi muodostaa yleistyksiä siitä, mistä tämä johtuu. Tutkimuskirjallisuuden mukaan teknisen velan suurin aiheuttaja on arkkitehtuuri. On mahdollista, että arkkitehtuuri tästä syystä dokumentoidaan useammin ja pidetään ajan tasalla. Tutkimuskirjallisuuden ja tämän tutkimuksen mukainen arkkitehtuurin roolin muuttuminen ketterien menetelmien myötä on tuonut arkkitehtuurin osaksi kehittäjän tehtäviä arkkitehtuurin suunnittelutavasta riippuen. Tästä syystä ketterien menetelmien yleistyminen on myös saattanut vaikuttaa siihen, miksi tämän tutkimuksen tulosten mukaan arkkitehtuurin dokumentaatio on projektin päättyessä useammin ajantasalla, kuin vaatimusmäärittely tai projektisuunnittelma.

Ohjelmistoarkkitehtuurin suunnittelu ja dokumentointi ketterissä malleissa

Tämän tutkimuksen tulosten mukaan arkkitehtuuriin käytettävien resurssien välillä ei ole eroa siltä osin käytetäänkö ketteriä menetelmiä vai ei. Tämän tutkimuksen mukaan dokumentaatio oli kattavampi ja ajantasaisempi niissä yrityksissä, joissa ei hyödynnetty ketteriä menetelmiä. Tuloksista kävi myös ilmi, että ketteriä menetelmiä hyödyntävät yritykset pitivät arkkitehtuurin dokumentointia tärkeämpänä kuin muuta dokumentaatiota. Arkkitehtuurin dokumentaatio myös laadittiin useammin. Näiden pohjalta ei kuitenkaan vielä voida tehdä yleistyksiä ketterien menetelmien käyttämisen ja käyttämättömyyden välille, koska pääosin kaikki vastaajat käyttivät ketteriä menetelmiä.

Tämän tutkimuksen tuloksista kävi ilmi, että ketterät menetelmät eivät joko ole vaikuttaneet dokumentaation määrään, tai ovat vaikuttaneet siihen vähentävästi. Jotkut vastaajat olivat tunnistaneet, että ketteriä menetelmiä käytetään perusteena dokumentoinnin puutteille. Tämän tutkimuksen mukaan ketterät menetelmät ovat vaikuttaneet dokumentaation tasoon heikentävästi, mutta samalla tutkimuskirjallisuuden mukaan dokumentoinnin tarve on myös muuttunut. Pilvipalvelut ovat tutkimuskirjallisuuden mukaan vähentäneet arkkitehtuurin dokumentoinnin tarvetta. Tutkimuskirjallisuudesta käy ilmi, että dokumentaatio on menossa automaation suuntaan, sekä prosesseihin, jotka itsessään tuottavat dokumentaatiota. Tämän tutkimuksen tulosten mukaan lähes puolet vastaajista olivat integroineet arkkitehtuurimallien päivittymisen automaattisesti osaksi ohjelmistokehitystä. Tutkimuskirjallisuuden pohjalta on myös yleistettävissä, että arkkitehtuurin muoto on muuttunut. Tutkimuskirjallisuuden mukaan arkkitehtuuria ei välttämättä tuoda näkyväksi (paperimuotoisella) dokumentaa-

tiolla, vaan jollain toisella tavalla; esimerkiksi arkkitehtuurin kuvauskieliä ei tutkimuskirjallisuuden mukaan enää käytetä, tai niiden käyttö on vähentynyt. Tämä voi johtua siitä, että tutkimuskirjallisuuden mukaan dokumentaatiota, joka ei päivity itsestään, on haastava pitää ajan tasalla. Tähän vaikuttaa myös arkkitehtuurin monet erilaiset merkitykset eri vastaajille sekä tutkimuskirjallisuuden mukainen arkkitehtuurin merkityksen muuttuminen korkean tason sidosryhmätyöskentelyvälineestä tämän tutkimuksen tuloksien mukaiseksi kehittäjän työn helpottajaksi.

7.2 Tutkimuksen luotettavuus ja kehitysaiheet

Kyselylomakkeeseen liittyi ongelmia. Vastaajajoukko jäi melko pieneksi, eikä selkeää eroa esimerkiksi ketteriä menetelmiä hyödyntävien yritysten ja muiden yritysten välillä pystytty muodostamaan. Ketteriä menetelmiä käytettiin lähes kaikissa yrityksissä, joten otanta ja luotettavuus jäi heikoksi niiden yritysten osalta, joissa ketterät menetelmät eivät ole käytössä. Ketterät menetelmät ovat johtava tapa toteuttaa ohjelmistoja, joten tämän tyyppinen vertailu ei muutoinkaan ole mielekästä.

Kyselyn laajasta jakelusta huolimatta lomakelähetyksiä tuli vähän. Kyselyllä saavutettiin 2,7% konversio, joka on yleiseen tasoon verrattuna matala. Vastausmäärän pienuuteen saattoivat vaikuttaa kysymysten asettelu ja saate. Kysymysten asettelu oli haaste. Heikkilä (2014) mukaan kysymysten muoto onkin yksi suurimmista aineiston virheiden aiheuttajista. Kyselylomake ei ollut tyhjentävä siltä osin mitä *ohjelmistoarkkitehtuurilla* tässä yhteydessä tarkoitetaan. Tästä syystä vastaajilla oli eriäviä näkemyksiä ohjelmistoarkkitehtuurista. Tätä voidaan selittää myös sillä, että ohjelmistoarkkitehtuuri käsitteenä on moniulotteinen ja laaja, sekä se on muuttunut ajan kuluessa. Näin jälkiviisaana tutkimuksessa olisi pitänyt keskittyä yleisesti ohjelmistosuunnitteluun, ei pelkkään arkkitehtuuriin, koska nämä tuntuvat menevät sekaisin joka tapauksessa. Arkkitehtuurin merkitys sidosryhmien laatuvaatimuksia toteuttavana työkaluna ei tullut tutkimuksessa esille.

Kyselyn pituus haluttiin pitää maltillisena, jotta vastaaminen olisi helpompaa. Tämä kuitenkin johti vastauksien jäämiseen hyvin yleistasoisiksi. Vastausten monimuotoisuus aiheutti haasteita vastauksien vertailuissa. Yleistyksiä voitiin tehdä yksittäisten avoimien kysy-

mysten vastausten tasolla, mutta ei vastaajamäärän mukaan, koska jokainen vastaaja vastasi ikäänkuin erilaiseen kysymykseen. Tarkkoja yleistyksiä oli mahdollista tehdä numeeristen vastauksien perusteella. Tästä johtuen tutkimusmenetelmänä toimisi paremmin haastattelu kyselyn sijaan. Vastauksien määrä olisi automaattisesti riittävä, eikä käsitteellistä ongelmaa pääsisi muodostumaan. Vastauksien analysointi ja yleistyksien teko olisi helpompaa.

Tutkijalla on näkemyksiä aihepiiriin, jotka ovat voineet vaikuttaa aineiston analysointiin. Aineiston analysointimenetelmät eivät ole tutkijalle tuttuja, tästä syystä kokeneempi tutkija olisi voinut saada kattavammat ja tarkemmat tutkimustulokset. Tutkimuksessa käytettiin paljon erilaisia lähteitä enimmäkseen sillä rajauksella, että ne liittyvät edes jollain tavalla tutkittavaan aiheeseen. Lähdekritiikkinä toimi lähinnä tutkimuksen jakokanavien luotettavuuden arviointi.

Tutkimuksen tuloksista kävi ilmi, että arkkitehtuuria ei suunnitella pienissä projekteissa. Aihe vaatisi lisää tutkimusta millä laajuudella, elinkaarella ja budjetilla arkkitehtuurin suunnittelu koetaan mielekkääksi. Kyselyssä kysyttiin projektin kestoa kuukausissa, mutta parempi olisi ollut kysyä projektin keskimääräistä kestoa henkilötyöviikkoina. Avoimen lähdekoodin ja koodin uudelleenkäytön osalta tutkimustulokset jäivät suppeaksi. Arkkitehtoninen yhteensopivuus ei suoraan näy tutkimustuloksissa siten, että voitaisiin päätellä sen aiheutuvan juuri avoimesta lähdekoodista tai koodin uudelleenkäytöstä. Jatkotutkimuksessa tulisi keskittyä koodin uudelleenkäytön vaikutukseen esimerkiksi refaktoroinnin, tietoturvan tai jonkin muun näkökulman kautta. Pelkästään ohjelmistoarkkitehtuuriin keskittyminen ei tämän tutkimuksen pohjalta ole mielekästä. Arkkitehtuuri on laaja asia, jonka salaisuuksia ei pelkällä pintaraapaisulla pysty selvittämään. Tutkimus jätti tutkija enemmän kysymyksiä kuin vastauksia.

Lähteet

Abrahamsson, P., M. Babar ja P. Kruchten. 2010. “Agility and Architecture: Can They Coexist?” *IEEE Software* (Los Alamitos, CA, USA) 27 (02): 16–22. doi:10.1109/MS.2010.36.

Adu, Michael. 2014. “Inadequate Requirements Engineering Process: A Key Factor for Poor Software Development in Developing Nations: A Case Study.” *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 8 (tammikuu): 1572.

“Agile Architecture in SAFe - Scaled Agile Framework”. 2020. 24. helmikuuta. Viitattu 22. lokakuuta 2020. https://www.scaledagileframework.com/agile-architecture/.

“Agile Manifesto”. 2001. Viitattu 20. heinäkuuta 2020. https://agilemanifesto.org/iso/fi/manifesto.html.

Ali Babar, Muhammad, Alan W. Brown ja Ivan Mistrík, toimittaneet. 2014. *Agile software architecture: aligning agile processes and software architectures.* Amsterdam: Morgan Kaufmann. ISBN: 978-0-12-407772-0. doi:10.1016/C2012-0-01208-2.

Allisy-Roberts, P, P Ambrosi, DT Bartlett, BM Coursey, LA DeWerd, E Fantuzzi ja JC McDonald. 2017. “The 11th Annual State of Agile Report”. *Journal of the ICRU* 6 (2): 7–8.

Ambler, S. W. 2020. “Agile Architecture: Strategies for Scaling Agile Development”. Viitattu 8. lokakuuta. http://www.agilemodeling.com/essays/agileArchitecture.htm.

Amoatey, Charles, ja Betty Anson. 2017. “Investigating the major causes of scope creep in real estate construction projects in Ghana”. *Journal of Facilities Management* 15 (heinäkuu). doi:10.1108/JFM-11-2016-0052.

Ampatzoglou, A., A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun ja K. Systa. 2016. “The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study”. Teoksessa *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD),* 9–16.

Babar, M. A. 2009. “An exploratory study of architectural practices and challenges in using agile software development approaches”. Teoksessa *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture,* 81–90. doi:10.1109/WICSA.2009.5290794.

Bachmann, Felix, ja Paulo Merson. 2005. “Experience Using the Web-Based Tool Wiki for Architecture Documentation”.

Baragry, J., ja K. Reed. 1998. “Why is it so hard to define software architecture?” Teoksessa *Proceedings 1998 Asia Pacific Software Engineering Conference (Cat. No.98EX240),* 28–36.

Bass, Len, Paul Clements ja Rick Kazman. 2013. *Software architecture in practice.* 3rd ed. SEI series in software engineering. Upper Saddle River, NJ: Addison-Wesley. ISBN: 978-0-321-81573-6.

Beck, K. 1999. “Embracing change with extreme programming”. *Computer* 32 (10): 70–77. doi:10.1109/2.796139.

Blair, S., R. Watt ja T. Cull. 2010. “Responsibility-Driven Architecture”. *IEEE Software* 27 (2): 26–32.

Booch, G. 2006. “The Accidental Architecture”. *IEEE Software* (Los Alamitos, CA, USA) 23 (03): 9–11. ISSN: 1937-4194. doi:10.1109/MS.2006.86.

Brown, Nanette, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim ym. 2010. “Managing Technical Debt in Software-Reliant Systems”. Teoksessa *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research,* 47–52. FoSER ’10. Santa Fe, New Mexico, USA: Association for Computing Machinery. doi:10.1145/1882362.1882373.

Brown, Simon. 2015. *Software Architecture for Developers.* Nide 2015. Leanpub.

Buschmann, F. 2012. “A Week in the Life of an Architect”. *IEEE Software* 29 (3): 94–96.

Buschmann, F., ja K. Henney. 2013. “Architecture and Agility: Married, Divorced, or Just Good Friends?” *IEEE Software* 30 (2): 80–82.

Clements, Paul, toimittanut. 2011. *Documenting software architectures: views and beyond.* 2nd ed. SEI series in software engineering. OCLC: ocn640079469. Upper Saddle River, NJ: Addison-Wesley. ISBN: 978-0-321-55268-6.

Cockburn, Alistair. 2004. “Crystal clear a human-powered methodology for small teams” (tammikuu).

———. 2007. *Agile software development : the cooperative game.* Upper Saddle River, NJ: Addison-Wesley. ISBN: 0321482751.

Constantinou, Eleni, ja Ioannis Stamelos. 2017. “Identifying evolution patterns: a metrics-based approach for external library reuse: Identifying evolution patterns: a metrics-based approach for external library reuse”. *Software: Practice and Experience* 47, numero 7 (heinäkuu): 1027–1039. Viitattu 13. syyskuuta 2020. doi:10.1002/spe.2484.

Coplien, James O, ja Gertrud Bjørnvig. 2011. *Lean architecture: for agile software development.* John Wiley & Sons.

Cunningham, Ward. 1992. “The WyCash Portfolio Management System”. Teoksessa *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum),* 29–30. OOPSLA ’92. Vancouver, British Columbia, Canada: Association for Computing Machinery. doi:10.1145/157709.157715.

Diaz-Pace, Andres, Christian Villavicencio, Silvia Schiaffino, Matías Nicoletti ja Hernan Vazquez. 2015. “Producing Just Enough Documentation: An Optimization Approach Applied to the Software Architecture Domain”. *Journal on Data Semantics* 5 (lokakuu). doi:10.1007/s13740-015-0053-0.

Dijkstra, Edsger W. 1968. “The structure of the “THE”-multiprogramming system”. *Communications of the ACM* 11, numero 5 (1. toukokuuta): 341–346. Viitattu 6. kesäkuuta 2020. doi:10.1145/363095.363143.

Dragičević, Zoran, ja Saša Bošnjak. 2019. “Agile architecture in the digital era: Trends and practices”. *Strategic Management* 24 (2): 12–33. doi:10.5937/straman1902011d.

draw.io. 2020. “draw.io - Diagrams for Confluence and Jira”. Draw.io. Viitattu 19. lokakuuta. https://drawio-app.com/.

Erder, M., ja P. Pureur. 2016. “What’s the Architect’s Role in an Agile, Cloud-Centric World?” *IEEE Software* 33 (5): 30–33.

Ernst, Neil A., Stephany Bellomo, Ipek Ozkaya, Robert L. Nord ja Ian Gorton. 2015. “Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt”. Teoksessa *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,* 50–60. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery. doi:10.1145/2786805.2786848.

Eskola, Jari. 2019. “Laadullisen tutkimuksen juhannustaiat: laadullisen aineiston analyysi vaihe vaiheelta”. Teoksessa *Ikkunoita tutkimusmetodeihin 2: Näkökulmia aloittelevalle tutkijalle tutkimuksen teoreettisiin lähtökohtiin ja analyysimenetelmiin,* 183. ISBN: 978-952-451-516-0.

Faber, R. 2010. “Architects as Service Providers”. *IEEE Software* 27 (2): 33–40.

Fairbanks, George. 2010. *Just enough software architecture: a risk-driven approach.* OCLC: 837823299. Boulder, Colo: Marshall & Brainerd. ISBN: 978-0-9846181-0-1.

Feitosa, Daniel, Apostolos Ampatzoglou, Antonios Gkortzis, Stamatia Bibi ja Alexander Chatzigeorgiou. 2020. “CODE reuse in practice: Benefiting or harming technical debt”. *Journal of Systems and Software* 167:110618. doi:10.1016/j.jss.2020.110618.

Fernández-Sánchez, C., J. Garbajosa, C. Vidal ja A. Yagüe. 2015. “An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture Perspective”. Teoksessa *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics,* 22–28.

Fowler, M. 2003. “Design - Who needs an architect?” *IEEE Software* 20 (5): 11–13. doi:10.1109/MS.2003.1231144.

Fowler, Martin, ja Kent Beck. 2018. *Refactoring: Improving the Design of Existing Code, Second Edition.* OCLC: 1159628261. Viitattu 13. syyskuuta 2020. https://www.safaribooksonline.com/library/view/-/9780134757681/?ar.

Garlan, David. 1995. "Research directions in software architecture". *ACM Computing Surveys (CSUR)* 27 (2): 257–261.

Garlan, David, Robert Allen ja John Ockerbloom. 1995. "Architectural mismatch or why it is so hard to build systems out of existing parts", nide 1995. Huhtikuu.

———. 2009. "Architectural Mismatch: Why Reuse is Still So Hard". 2009 (heinäkuu). https://repository.upenn.edu/library_papers/68/.

Gerdes, Sebastian, Stefanie Jasser, Matthias Riebisch, Sandra Schröder, Mohamed Soliman ja Tilmann Stehle. 2016. "Towards the Essentials of Architecture Documentation for Avoiding Architecture Erosion". Teoksessa *Proccedings of the 10th European Conference on Software Architecture Workshops.* ECSAW '16. Copenhagen, Denmark: Association for Computing Machinery. ISBN: 9781450347815. doi:10.1145/2993412.3004844.

"Google Trends". 2020. Viitattu 22. lokakuuta. https://trends.google.com/trends/explore?date=all&geo=US&q=software%20architecture.

Graaf, Klaas. 2015. "Ontology-based Software Architecture Documentation". Tohtorinväitöskirja. doi:10.13140/RG.2.1.4034.7603.

Heikkilä, Tarja. 2014. *Tilastollinen tutkimus.* Edita Publishing Oy.

Heinemann, Lars, Florian Deissenboeck, Mario Gleirscher, Benjamin Hummel ja Maximilian Irlbeck. 2011. "On the Extent and Nature of Software Reuse in Open Source Java Projects". Teoksessa *Top Productivity through Software Reuse,* toimittanut Klaus Schmid, 207–222. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-21347-2.

Hirsjärvi, 1940- kirjoittaja, Sirkka. 2009. *Tutki ja kirjoita.* 15. uud. p. Toimittanut kuvittaja Sinivuori Eila. Helsinki: Tammi.

Hoda, R., N. Salleh ja J. Grundy. 2018. "The Rise and Evolution of Agile Software Development". *IEEE Software* 35 (5): 58–63. doi:10.1109/MS.2018.290111318.

Hoda, Rashina, James Noble ja Stuart Marshall. 2012. “Documentation strategies on agile software development projects”. *International Journal of Agile and Extreme Software Development* 1 (1): 23–37.

Hohpe, G., I. Ozkaya, U. Zdun ja O. Zimmermann. 2016. “The Software Architect’s Role in the Digital Age”. *IEEE Software* 33 (6): 30–39.

Holmes, Benedikt, ja Ana Nicolaescu. 2017. “Continuous architecting: Just another buzzword”. *Full-scale software engineering/the art of software testing:* 1–6.

Ibriwesh, I., Sin-Ban Ho ja Ian Chai. 2018. “Identify the effective documentation method for representing the functional software architecture”. *International Journal of Engineering and Technology(UAE)* 7 (maaliskuu): 46–49. doi:10.14419/ijet.v7i2.3.9966.

“ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description”. 2011 (joulukuu).

Jabbar, HS, T Gopal, A Ofuoku, B Isife, G Emah, M Fezari, KM Sulieman, X Huang, J Liu ja M Tang. 2007. “Qualitative analysis model for software requirements driven by interviews”. *J. Eng. Applied Sci* 2 (1): 1–9.

Jaiswal, Manishaben. 2019. “Software Architecture and Software Design”. *International Research Journal of Engineering and Technology (IRJET) e-ISSN:* 2395–0056.

Jansen, A., ja J. Bosch. 2005. “Software Architecture as a Set of Architectural Design Decisions”. Teoksessa *5th Working IEEE/IFIP Conference on Software Architecture (WICSA’05),* 109–120.

Keuler, Thorsten, Stefan Wagner ja Bernhard Winkler. 2012. “Architecture-aware Programming in Agile Environments”. Teoksessa *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture,* 229–233. IEEE.

Komulainen, Joni. 2018. *Tietojen anonymisointi ja sen vaikutus tietojen luovutukseen tietoluvan tai tietopyynnön perusteella,* 5. joulukuuta. Viitattu 22. kesäkuuta 2020. https://www.eduskunta.fi/FI/vaski/JulkaisuMetatieto/Documents/EDK-2018-AK-229673.pdf.

Korhonen, S. 2017. “Suomen it-alan kasvulle ei näy loppua”. Tivi. 16. toukokuuta. Viitattu 7. kesäkuuta 2017. https://www.tivi.fi/Kaikki_uutiset/suomen-it-alan-kasvulle-ei-nay-loppua-6691521.

Korpimies, A. 2017. “It-alan kasvu ohitti muut alat – osaajapula jarruttaa tahtia”. Tivi. 16. toukokuuta. Viitattu 7. kesäkuuta 2017. https://www.tivi.fi/Kaikki_uutiset/it-alan-kasvu-ohitti-muut-alat-osaajapula-jarruttaa-tahtia-6649362.

Koskimies, Kai, ja Tommi Mikkonen. 2005. *Ohjelmistoarkkitehtuurit.* OCLC: 76846495. Helsinki: Talentum. ISBN: 978-952-14-0862-5.

Kruchten, P., R. L. Nord ja I. Ozkaya. 2012. “Technical Debt: From Metaphor to Theory and Practice”. *IEEE Software* 29 (6): 18–21.

Kruchten, Philippe. 2004. *The rational unified process: an introduction.* Addison-Wesley Professional.

———. 2010. “Software architecture and agile software development: a clash of two cultures?” Teoksessa *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2,* 497–498.

Larson, Deanne, ja Victor Chang. 2016. “A review and future direction of agile, business intelligence, analytics and data science”. *International Journal of Information Management* 36 (5): 700–710. doi:10.1016/j.ijinfomgt.2016.04.013.

Li, Zengyang, Paris Avgeriou ja Peng Liang. 2015. “A systematic mapping study on technical debt and its management”. *Journal of Systems and Software* 101:193–220. ISSN: 0164-1212. doi:10.1016/j.jss.2014.12.027.

Malavolta, Ivano, Patricia Lago, Henry Muccini, Patrizio Pelliccione ja Antony Tang. 2013. “What Industry Needs from Architectural Languages: A Survey”. *IEEE Transactions on Software Engineering* 39, numero 6 (kesäkuu): 869–891. doi:10.1109/tse.2012.74.

Mancl, Dennis, Steven Fraser, Bill Opdyke, Ethan Hadar ja Irit Hadar. 2009. “Architecture in an agile world”. Teoksessa *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications,* 719–720.

Martensson, Torvald, Daniel Stahl, Antonio Martini ja Jan Bosch. 2019. “Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles”. Teoksessa *2019 IEEE International Conference on Software Architecture (ICSA).* IEEE, maaliskuu. doi:10.1109/icsa.2019.00022.

Martin, Robert C. 2018. *Clean architecture: a craftsman’s guide to software structure and design.* Robert C. Martin series. Boston, MA: Prentice Hall.

Martini, A., ja J. Bosch. 2016. “A Multiple Case Study of Continuous Architecting in Large Agile Companies: Current Gaps and the CAFFEA Framework”. Teoksessa *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA),* 1–10.

Martini, Antonio, Viktoria Stray ja Nils Brede Moe. 2019. “Technical-, Social- and Process Debt in Large-Scale Agile: An Exploratory Case-Study”. Teoksessa *Agile Processes in Software Engineering and Extreme Programming – Workshops,* toimittanut Rashina Hoda, 112–119. Cham: Springer International Publishing. ISBN: 978-3-030-30126-2.

Mikkonen, T., ja A. Taivalsaari. 2019. “Software Reuse in the Era of Opportunistic Design”. *IEEE Software* 36 (3): 105–111.

Mirakhorli, M., ja J. Cleland-Huang. 2013. “Traversing the Twin Peaks”. *IEEE Software* 30 (2): 30–36.

Miyachi, Christine. 2011. “Agile software architecture”. *ACM SIGSOFT Software Engineering Notes* 36 (2): 1–3.

Moran, Alan. 2014. *Agile risk management.* Springer briefs in computer science. OCLC: ocn869267001. New York: Springer. ISBN: 978-3-319-05007-2.

Nord, R. L., ja J. E. Tomayko. 2006. “Software architecture-centric methods and agile development”. *IEEE Software* 23 (2): 47–53.

Nord, Robert L., Ipek Ozkaya ja Philippe Kruchten. 2014. “Agile in Distress: Architecture to the Rescue”. Teoksessa *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation,* toimittanut Torgeir Dingsøyr, Nils Brede Moe, Roberto Tonelli, Steve Counsell, Cigdem Gencel ja Kai Petersen, 43–57. Cham: Springer International Publishing. ISBN: 978-3-319-14358-3.

Nord, Robert, ja J.E. Tomayko. 2006. “Software architecture-centric methods and agile development”. *Software, IEEE* 23 (huhtikuu): 47–53. doi:10.1109/MS.2006.54.

Obbink, H, P Kruchten, W Kozaczynski, H Postema, A Ran, A Dominick, R Kazman, R Hilliard, W Tracz ja E Kahane. 2020. *Software Architecture Review and Assessment (SARA) Report.* 6. helmikuuta. Viitattu 24. syyskuuta 2020. https://pkruchten.files.wordpress.com/2011/09/saravl.pdf.

Perry, Dewayne E., ja Alexander L. Wolf. 1992. “Foundations for the Study of Software Architecture”. *SIGSOFT Softw. Eng. Notes* (New York, NY, USA) 17, numero 4 (lokakuu): 40–52. doi:10.1145/141874.141884.

Peters, Lawrence. 2014. “Technical Debt: The Ultimate Antipattern - The Biggest Costs May Be hidden, Widespread, and Long Term”. *Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014* (joulukuu): 8–10. doi:10.1109/MTD.2014.7.

Pohl, Klaus. 2016. *Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant.* Rocky Nook, Inc.

Poort, E., C. Pautasso ja O. Zimmermann. 2016. “Just Enough Anticipation: Architect Your Time Dimension”. *IEEE Software* 33 (6): 11–15.

Poort, Eltjo R., ja Hans van Vliet. 2012. “RCDA: Architecting as a risk- and cost management discipline”. Selected papers from the 2011 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2011), *Journal of Systems and Software* 85 (9): 1995–2013. doi:10.1016/j.jss.2012.03.071.

Rajala, Ari. 2019. “Digitalisaatio vauhdittaa it-alan kasvua, jota jarruttaa vain pula osaajista | Kauppalehti”. Kauppalehti. 17. syyskuuta. Viitattu 22. lokakuuta 2020. https://www.kauppalehti.fi/uutiset/digitalisaatio-vauhdittaa-it-alan-kasvua-jota-jarruttaa-vain-pula-osaajista/d54ebd97-b9af-4ca3-b9e6-25b7b456e46b.

Rechtin, Eberhardt. 1994. “THE SYSTEMS ARCHITECT: SPECIALTY, ROLE AND RESPONSIBILITY”. *INCOSE International Symposium* 4, numero 1 (elokuu): 521–524. ISSN: 23345837, viitattu 23. syyskuuta 2020. doi:10.1002/j.2334-5837.1994.tb01752.x.

Rodríguez, Pilar, Mika Mäntylä, Markku Oivo, Lucy Ellen Lwakatare, Pertti Seppänen ja Pasi Kuvaja. 2019. “Chapter Four - Advances in Using Agile and Lean Processes for Software Development”, toimittanut Atif M. Memon, 113:135–224. Advances in Computers. Elsevier. doi:10.1016/bs.adcom.2018.03.014.

Rost, Dominik, Balthasar Weitzel, Matthias Naab, Torsten Lenhart ja Hartmut Schmitt. 2015. “Distilling Best Practices for Agile Development from Architecture Methodology: Experiences from Industrial Application”. Syyskuu. doi:10.1007/978-3-319-23727-5_21.

Saaranen-Kauppinen, Anita, ja Anna Puusniekka. 2006. “KvaliMOTV - Menetelmäopetuksen tietovaranto”. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 15. lokakuuta 2020. https://www.fsd.tuni.fi/menetelmaopetus/kvali/L2_3_2_3.html.

Savaşcı, Mustafa, ja Fatih Çetin. 2018. “Software Architecture Documentation in Agile”. Toukokuu.

Schade, Lukas, ja Christian Plewnia. 2018. “Does Agile Software Development need Architecture?” *Continuous Software Engineering & Full-scale Software Engineering:* 49.

Schwaber, Ken. 2004. *Agile project management with Scrum.* Redmond, Wash: Microsoft Press. ISBN: 978-0-7356-1993-7.

Selic, Bran. 2009. “Agile Documentation, Anyone?” *IEEE Software* 26, numero 6 (marraskuu): 11–12. doi:10.1109/ms.2009.167.

Shaw, Mary. 1995. “Architectural Issues in Software Reuse: It’s Not Just the Functionality, It’s the Packaging”. Teoksessa *Proceedings of the 1995 Symposium on Software Reusability,* 3–6. SSR ’95. Seattle, Washington, USA: Association for Computing Machinery. doi:10.1145/211782.211783.

Sherman, Sofia, ja Irit Hadar. 2012. “Identifying the need for a sustainable architecture maintenance process”. *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering, CHASE 2012 - Proceedings* (kesäkuu). doi:10.1109/CHASE.2012.6223010.

Solms, Fritz. 2012. “What is Software Architecture?” Teoksessa *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference,* 363–373. SAICSIT ’12. Pretoria, South Africa: Association for Computing Machinery. doi:10.1145/2389836.2389879.

Tom, Edith, Aybüke Aurum ja Richard Vidgen. 2013. “An exploration of technical debt”. *Journal of Systems and Software* 86 (6): 1498–1516. doi:j.jss.2012.12.052.

Tuomi, Jouni, ja Anneli Sarajärvi. 2019. *Laadullinen tutkimus ja sisällönanalyysi.* ISBN: 978-952-04-0011-8.

Valli, Raine. 2019. *Ikkunoita tutkimusmetodeihin 1: Metodin valinta ja aineistonkeruu: virikkeitä aloittelevalle tutkijalle.* ISBN: 978-952-451-516-0.

Waterman, M. 2018a. “Agility, Risk, and Uncertainty, Part 1: Designing an Agile Architecture”. *IEEE Software* 35 (2): 99–101.

———. 2018b. “Agility, Risk, and Uncertainty, Part 2: How Risk Impacts Agile Architecture”. *IEEE Software* 35 (3): 18–19.

Waterman, Michael, James Noble ja George Allan. 2015. “How much up-front? A grounded theory of agile architecture”. Teoksessa *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering,* 1:347–357. IEEE.

“What is your definition of software architecture?” 2010. Joulukuu. Viitattu 24. syyskuuta 2020. https://resources.sei.cmu.edu/asset_files/FactSheet/2010_010_001_513810.pdf.

Woods, E. 2015. “Aligning Architecture Work with Agile Teams”. *IEEE Software* (Los Alamitos, CA, USA) 32 (05): 24–26. ISSN: 1937-4194. doi:10.1109/MS.2015.119.

Zimmermann, Olaf. 2016. “Designed and Delivered Today, Eroded Tomorrow? Towards an Open and Lean Architecting Framework Balancing Agility and Sustainability”. Teoksessa *Proccedings of the 10th European Conference on Software Architecture Workshops*. ECSAW ’16. Copenhagen, Denmark: Association for Computing Machinery. ISBN: 9781450347815. doi:10.1145/2993412.3014339.

Liitteet

A Kyselylomake

ARKKITEHTUURITUTKIMUS

Kysely on osa Jyväskylän Yliopistossa tehtävää pro gradu-tutkimusta. Tutkimuksen tavoitteena on kartoittaa arkkitehtuurisuunnittelun tilaa suomalaisissa yrityksissä.

Kaikki kysymykset ovat vapaaehtoisia ja tulokset kerätään nimettömästi. Lomakevastaukset poistetaan tutkimuksen valmistumisen jälkeen ja tämän jälkeen vastaukset ovat luettavissa vain koosteena lopullisesta pro gradu -tutkielmasta, joka tallennetaan Jyväskylän Yliopiston julkaisuarkisto Jyxiin.

Tutkimusta suorittaa Tiia Rantanen. Lisätietoa vois kysyä sähköpostiosoitteella tiia.k.rantanen@student.jyu.fi

Rooli yrityksessä

- ○ Sovellus/ohjelmistokehitys (esim. ohjelmointi)
- ○ Ohjelmistosuunnittelu (esim. arkkitehti)
- ○ Projektinhallinta (esim. BA)
- ○ Konsultointi
- ○ Muu

Titteli

Koulutus

Monta vuotta olet työskennellyt alalla?

Yrityksen koko

- ○ 10 tai alle
- ○ 11-50
- ○ 51-100
- ○ 101-200
- ○ yli 200

Minä vuonna yritys on perustettu?

Yrityksen toimiala

Ei valintaa

Mitä ohjelmointi- tai skriptikieliä yrityksessä käytetään?

- ☐ Java
- ☐ C#
- ☐ Python
- ☐ Node.js / Lambda
  (tai jokin muu backend JavaScript)
- ☐ PHP
- ☐ JavaScript
- ☐ Muu

Mitä tuotteita yritys tuottaa eniten?

- ○ Pelit
- ○ Markkinointisivustot ja muut nettisivut

- Web-sovellukset
- Työpöytäohjelmia tai -ohjelmistoja
- Mobiilisovelluksia
- IoT ja sulautetut järjestelmät
- Muu

Kuinka pitkä on keskimääräisen projektin elinkaari kuukausissa?

Elinkaarella tarkoitetaan tässä aikaväliä aloituksesta siihen, että toteutus on valmis ja se uudistetaan (esim. refaktorointi), tehdään kokonaan uudestaan tai poistetaan käytöstä. Vastaa kokonaisina kuukausina.

Koostuuko suurin osa yrityksen liikevaihdosta avoimen lähdekoodin päälle rakennetuista toteutuksista?

- Kyllä
- Ei

Käytättekö ketteriä menetelmiä?

- Kyllä
- Ei

Mitä prosessi/kehitysmallia käytätte?

- [ ] Scrum
- [ ] Kanban
- [ ] DevOps
- [ ] Talon sisäinen (inhouse) malli
- [ ] Vesiputousmalli

- [ ] Muu

Mitä seuraavista hyödynnätte yrityksessänne?

- [ ] IaaS (ulkoistettu infrastruktuuri)
- [ ] PaaS (ulkoistettu alusta)
- [ ] BaaS (ulkoistettu backend)
- [ ] FaaS (kaiken laskentatehon ulkoistaminen)
- [ ] Tuotamme palvelut itse

Mitä dokumentaatiota laaditte kehitysprojektista ennen varsinaisen kehitystyön alkamista?

- [ ] Projektisuunnitelma
- [ ] Vaatimusmäärittely
- [ ] Arkkitehtuuri
- [ ] Testaussuunnitelma
- [ ] Backlog
- [ ] Muu

Mikä osa dokumentaatiosta on ajan tasalla projektin päättyessä?

- [ ] Kaikki tuotettu dokumentaatio
- [ ] Projektisuunnitelma
- [ ] Vaatimusmäärittely
- [ ] Arkkitehtuuri
- [ ] Testaussuunnitelma
- [ ] Backlog
- [ ] Muu

Onko arkkitehtuurimallien päivittyminen integroitu osaksi sovelluskehitystä?

- ○ Kyllä
- ○ Ei

Mitä lähestymistapaa käytätte arkkitehtuurin suunnittelussa?

- ○ Domain Driven Design
- ○ Attribute Driven Design
- ○ Object-Oriented Design
- ○ Component-Based Development
- ○ Model-Driven Architecture
- ○ Muu

Mistä näkökulmista laaditte arkkitehtuurin dokumentaatiota?

- ☐ Kruchten 4+1 (kehitysnäkymä, looginen näkymä, fyysinen näkymä, prosessinäkymä, skenaariot)
- ☐ 4C-malli (context, component, container, code)
- ☐ Kehitys- tai tuotantoympäristöjen fyysinen arkkitehtuuri
- ☐ Muu

Mitä notaatiota käytätte arkkitehtuurikuvauksissa?

- ○ UML
- ○ Yrityksen sisäisesti sovittu notaatio
- ○ Muu

Mitä työkaluja tai ohjelmia käytätte arkkitehtuurikuvauksien tekemiseen?

- IBM Rational Rose Modeler
- Micro Focus Together
- LucidChart
- Visio
- draw.io
- ArchiMate
- Muu

Mikä mielestäsi on arkkitehdin tärkein ominaisuus?

- Viestintätaidot
- Tekninen osaaminen
- Johtamistaidot
- Stressin sietokyky
- Vastuullisuus
- Muu

Asteikolla 1-10, miten tärkeänä pidät dokumentaation laatimista?

1 = ei ollenkään tärkeä, 10 = erittäin tärkeä

Asteikolla 1-10, miten tärkeänä pidät arkkitehtuurin laatimista?

1 = ei ollenkään tärkeä, 10 = erittäin tärkeä

Kuinka monta prosenttia keskimääräisen projektin resursseista on varattu dokumentaation laatimiseen?

- 0-20
- 21-40
- 41-60
- Muu

Kuinka monta prosenttia keskimääräisen projektin resursseista on varattu arkkitehtuurin laatimiseen?

- 0-20
- 21-40
- 41-60
- Muu

Miten koet arkkitehtuurin auttavan projektin eri vaiheissa?

0 / 2000 merkkien enimmäismäärästä

Mitä ongelmia arkkitehtuurisuunnittelussa esiintyy?

0 / 2000 merkkien enimmäismäärästä

Lähetä