Oskari Kokki

TAPAUSTUTKIMUS MIKROPALVELUIDEN TEORIAN JA KÄYTÄNNÖN KOHTAAMISESTA SOVELLUSKEHITYKSESSÄ

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2024

TIIVISTELMÄ

Kokki, Oskari
Tapaustutkimus mikropalveluiden teorian ja käytännön kohtaamisesta sovelluskehityksessä
Jyväskylä: Jyväskylän yliopisto, 2024, 60 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Halttunen, Veikko

Sovelluskehityksessä sovellusjärjestelmät noudattavat arkkitehtuuria, mihin perustuen kokonaisuus rakentuu. Järjestelmäarkkitehtuuri kuvaa eri kokonaisuudet ja niiden väliset suhteet. Perinteisin ja vanhin järjestelmäarkkitehtuuri on monoliittiarkkitehtuuri, jossa kaikki komponentit eli järjestelmän osat ovat yhdessä suuressa koodipohjassa, mistä tulee nimitys monoliitti. Kaikilla järjestelmäarkkitehtuureilla on erilaisia hyötyjä sekä haasteita, joita tulee punnita, kun tehdään päätös järjestelmän tai sovelluksen arkkitehtuurista. Nämä haasteet ovat johtaneet arkkitehtuurien kehitykseen, kun erilaiset tekijät ja muutokset ovat puskeneet järjestelmäarkkitehtuurien vaatimuksia eteenpäin. Kasvaneiden vaatimusten lisäksi teknologian kehitys on mahdollistanut uusien järjestelmäarkkitehtuurien synnyn. Mikropalveluarkkitehtuuri on kehittynyt tarpeesta paikata aiempien järjestelmäarkkitehtuurien puutteita, kuten esimerkiksi monoliitin tehottomuus ja sen hankala jatkokehittäminen, kun järjestelmä kasvaa liian suureksi ja monimutkaiseksi. Tutkimuksessa selvitetään, kohtaavatko tieteellisen kirjallisuuden ja käytännön ammattilaisten näkemykset mikropalveluista sekä niiden hyödyistä ja haasteista. Käytännön ammattilaiset ovat ohjelmistoyritys Eduixin järjestelmäarkkitehteja. Tutkimus on laadullinen tutkimus, jossa ammattilaiset on haastateltu puolistrukturoitujen haastatteluiden avulla, ja näiden haastatteluiden tuloksia on verrattu kirjallisuuskatsauksessa kerättyyn materiaaliin, mikä muodostaa tieteellisen kirjallisuuden näkemyksen aiheesta. Tutkimuksen tarkoituksena on antaa Eduixille informaatiota koskien mikropalveluita ja heidän valmiuksiaan käyttää mikropalveluita. Tätä informaatiota he voivat hyödyntää tehdessään päätöksiä liittyen mikropalveluiden käyttämisestä. Tutkimuksen tarkoituksena on myös tarjota näkökulmaa kummaltakin osapuolelta, ja informoida tieteellistä kirjallisuutta ammattilaisten mielipiteestä koskien mikropalveluita: onko esimerkiksi tieteellinen kirjallisuus etääntynyt liikaa käytännöstä? Tämän tutkimuksen myötä voi myös paljastua tärkeitä jatkotutkimusaiheita tai keskeisiä kehityskohteita mikropalveluihin liittyen.

Asiasanat: mikropalvelut, komponentointi, modulaarisuus, järjestelmäarkkitehtuuri

ABSTRACT

Kokki, Oskari
A case study on the intersection of theory and practice of microservices in software development
Jyväskylä: University of Jyväskylä, 2024, 60 pp.
Information Systems Science, Master's Thesis
Supervisor: Halttunen, Veikko

In application development, application systems follow an architecture, which forms the basis for the whole application system. The system architecture describes the different components and the relationships between them. The most traditional and the oldest system architecture is the monolithic architecture, where all components are contained in one large code base, hence the name monolith. All system architectures have different benefits and challenges that need to be weighed when deciding on the architecture of a system or an application. These challenges have led to the evolution of architectures as various factors and changes have pushed the requirements for system architectures forward. In addition to increased requirements, technological advances have enabled the development of new system architectures. The microservice architecture has evolved from the need to address the shortcomings of previous system architectures, such as the inefficiency of the monolith when it comes to further developing it, when the system grows too large and complex. This study aims to find out if the views of scientific literature and the views of the professionals meet regarding microservices and their benefits and challenges. The professionals chosen for this study are system architects from a software company Eduix. The study is a qualitative study and the professionals have been interviewed in the form of semi-structured interviews, and the results of these interviews have been compared with the literature review, which forms the scientific literature view about microservices. The aim of the study is to provide Eduix with information about microservices and their capacity to use microservices, which they can utilize when they make decisions regarding the use of microservices. The study also aims to provide a perspective from both sides, and to inform the scientific literature about the opinion of the professionals concerning microservices. For example, if the scientific literature has become too distant from practice, or if this research reveals important areas for future research or key developments in the field of microservices.

Keywords: microservices, componentization, modularity, system architecture

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Mikropalveluarkkitehtuuri on järjestelmäarkkitehtuuri, jonka avulla järjestelmä jaetaan selkeästi määriteltyihin ja rajattuihin komponentteihin eli mikropalveluihin. Tämä tuo mukanaan hyötyjä, erityisesti jos järjestelmä on kasvanut liian suureksi, ja sen ylläpitäminen ja jatkokehittäminen ovat muuttuneet haastavaksi sen seurauksena. Lisäksi mikropalvelut tehostavat skaalaamista, mikä on hyödyllistä käyttäjämäärän kasvaessa tarpeeksi suureksi tai vaihdellessa merkittävästi. Mikropalvelut ovat suhteellisen nuori teknologia, ja sen ongelmia ovat sen monimutkaisuuteen ja rakentamiseen liittyvät puuttuvat standardit. Vaikka mikropalvelut tuovat paljon hyötyjä, on yritysten punnittava tarkkaan, ovatko mikropalveluiden hyödyt tarpeeksi suuria, jotta yrityksen kannattaa kohdata sen tuomat haasteet.

Eduix on suomalainen sovelluskehitysyritys, joka oli näiden kysymysten äärellä, ja halusi selvittää heidän valmiuksiaan liittyen mikropalveluiden hyödyntämiseen. Tutkimuksen avulla Eduix saa vastauksia näihin kysymyksiin, sekä lisää informaatiota tieteellisestä kirjallisuudesta koskien mikropalveluita.

Tieteellinen motivaatio tutkimukselle muodostuu mielenkiinnosta selvittää, onko mikropalveluiden tutkimus tieteellisessä kirjallisuudessa mennyt samaan suuntaan kuin käytännön ammattilaisten käsitys, vai ovatko heidän ajatuksensa asiaan liittyen menneet eri suuntiin vuosien varrella. Onko esimerkiksi käytännön ammattilaisilla opittavaa tieteellisen kirjallisuuden löydöistä mikropalveluihin liittyen? Ilmentyvien erojen perusteella käytännön ammattilaiset voivat oppia uusia asioita liittyen mikropalveluihin, jotka voivat mahdollistaa uusia käyttötapauksia mikropalveluille, mistä he eivät olleet aiemmin tietoisia. Heille voi myös ilmetä uusia haasteita, joihin varautua, tai jopa torjua mikropalvelut käytettävänä arkkitehtuurina kokonaan, jos haasteet ovat liian suuria verrattuna höytyihin. Tieteellinen kirjallisuus taas voi erojen perusteella muuttaa suuntaa liittyen mikropalveluiden tieteelliseen tutkimukseen. Jos tutkimuksessa paljastuu, että alan ammattilaiset tuovat esille parannusehdotuksia tai puutteita liittyen mikropalveluihin, on tieteellisellä kirjallisuudella mahdollisuus kohdistaa tutkimusta näihin kohteisiin. Toisaalta jos tutkimuksen seurauksena tulee esille, että jokin ominaisuus ei ole käytännössä hyödyllinen, vaan tuo mukanaan

ylimääräistä monimutkaisuutta tavoittamatta odotettuja höytyjä, voi kirjallisuudella olla syy muistuttaa itseään siitä, mikä on realistista toteuttaa käytännössä.

Kohdeyrityksen ja tieteellisen motivaation pohjalta muodostetaan kaksi tutkimuskysymystä:

Tutkimuskysymys 1: Miten tieteellinen kirjallisuus ja alan ammattilaiset käsittävät termin ”mikropalvelu”, ja mitä eroja näiden kahden näkökulman välillä esiintyy?

Tutkimuskysymys 2: Mitä hyötyjä ja haasteita tieteellinen kirjallisuus ja alan ammattilaiset näkevät liittyen mikropalveluiden käyttämiseen sovellus- ja järjestelmäkehityksessä, ja mitä eroja näiden kahden näkökulman välillä esiintyy?

Tutkimuskysymyksien avulla termi ”mikropalvelu”, sekä mikropalveluiden hyödyt ja haasteet nostetaan esille, koska teknologian valintaa päätettäessä on yhtä tärkeää tiedostaa arkkitehtuuri yleisesti, että omistaa käsitys arkkitehtuurin keskeisimmistä hyödystä sekä haasteista, jotta pystytään punnitsemaan, onko se sopiva arkkitehtuuri kyseessä olevalle järjestelmälle tai sovellukselle.

Tieteellisen kirjallisuuden näkökulma tuodaan esille toisessa ja kolmannessa luvussa kirjallisuuskatsauksen avulla, kun taas alan ammattilaisten näkökulma tuodaan esille viidennessä luvussa. Kumpaankin tutkimuskysymykseen annetaan lopulliset vastaukset kuudennessa luvussa, kun näitä kahta näkökulmaa verrataan toisiinsa.

Tutkimuksen rajauksena toimii kohdeyritys, missä haastateltavat järjestelmäarkkitehdit työskentelevät, mikä rajoittaa heidän mielipiteitään ja käyttökohteitaan suppeampaan käyttäjämäärään, verrattuna yritykseen, jonka käyttäjämäärät olisivat suuremman väestömäärän omaavassa valtiossa, tai jolla olisi kansainvälisempi käyttäjäkunta. Tästä huolimatta kaikki haastateltavat järjestelmäarkkitehdit ovat olleet myös mukana kansainvälisissä projekteissa, joten he omistivat myös näkökulmaa siltäkin kantilta.

Keskeisinä käsitteinä tutkimuksessa ovat järjestelmäarkkitehtuuri, mikropalvelut, komponentit ja moduulit. Järjestelmäarkkitehtuuri kuvastaa järjestelmän rakennetta, mistä osista kokonaisuus koostuu ja mitkä niiden suhteet ovat toisiinsa. Mikropalvelut ovat teknologia, jonka avulla nämä osat voidaan eristää omiksi itsenäisiksi kokonaisuuksiksi. Komponentit ja moduulit ovat vanhempia termejä, joiden ideologioiden pohjalta mikropalvelut ovat kehittyneet. Komponentit ja moduulit kuvastavat järjestelmän osien rajaamista omiin kokonaisuuksiin, ja niiden ideaalisena tarkoituksena olisi, että niitä voitaisiin liittää mahdollisimman pienellä vaivalla osaksi erilaisia kokonaisuuksia.

Tutkielman toinen luku koskee järjestelmäarkkitehtuuria, ja siinä pohjustetaan, mitä järjestelmäarkkitehtuuri on ja mikä merkitys sillä on sovelluskehityksessä. Lisäksi siinä tuodaan esille mitkä arkkitehtuurit ovat edeltäneet mikropalveluarkkitehtuuria, ja samalla tuodaan esille ideologioita ja tekijöitä sille, miksi kehitys on johtanut mikropalveluihin. Kolmannessa luvussa käsitellään itse mikropalvelut, ja tuodaan esille kirjallisuudessa käytetty määritelmä, siihen liittyviä

teknologioita, sekä kirjallisuudessa esitetyt hyödyt ja haasteet koskien mikropalveluita. Keskeisimpinä lähteinä mikropalveluihin liittyvässä tieteellisessä kirjallisuudessa ovat Fowler ja Lewisin 2014 kirjoittama artikkeli koskien mikropalveluiden määritelmää ja keskeisiä piirteitä, sekä Fowlerin 2015 kirjoittama artikkeli, joka käsittelee mikropalveluiden haasteita ja hyötyjä. Neljäs luku esittelee tutkimuksen menetelmän ja aineiston keruun toteuttamiset. Viides luku esittelee haastatteluiden tulokset, ja kuudes luku tulkitsee ja analysoi tuloksia vertaamalla niitä kirjallisuuteen sekä pohtii, mitä näiden vertailujen tulokset tarkoittavat sekä tieteellisissä että käytännön konteksteissa. Viimeisessä luvussa käydään yhteenvetona tutkimus läpi.

2 JÄRJESTELMÄARKKITEHTUURI

Tässä ensimmäisessä teorialuvussa käsitellään kirjallisuuskatsauksen muodossa järjestelmäarkkitehtuuria. Luku on jaettu alalukuihin, joista ensimmäisessä käsitellään järjestelmäarkkitehtuurin käsitettä yleisemmin, mikä antaa kuvan järjestelmäarkkitehtuurien roolista järjestelmäkehityksessä. Toisessa alaluvussa käsitellään järjestelmäarkkitehtuurien kehitystä erityisesti mikropalveluiden näkökulmasta. Toisessa alaluvussa käydään läpi kehitykseen vaikuttaneita tekijöitä ja aiempia arkkitehtuureja.

Termejä järjestelmä- ja sovellusarkkitehtuuri käytetään tieteellisessä kirjallisuudessa kuvaamaan melkein samaa asiaa: kaikki järjestelmäarkkitehtuurit eivät välttämättä ole sovellusarkkitehtuureja, mutta sovellukset ovat järjestelmien pääasiallisin käyttökohde. Tutkimuksen kontekstissa näiden kahden termin välinen eri ei ole merkittävä, eikä se tullut esille tämän tutkimuksen aikana. Tässä tutkimuksessa käsiteltävät asiat koskevat sekä järjestelmä- että sovellusarkkitehtuureita.

2.1 Järjestelmäarkkitehtuuri

Sovelluskehittäjät kohtaavat jatkuvasti uusia haasteita, kun liiketoiminta ja teknologia kehittyvät ja sovellukset monimutkaistuvat. Ei riitä enää, että järjestelmä täyttää sidosryhmien sille antamat vaatimukset, vaan järjestelmän pitää myös olla valmis evolutiivisesti mukautumaan tuleviin muutoksiin (Breivold, Crnkovic & Larsson, 2012). Sovelluskehityksessä arkkitehtuuri toimii siltana, joka yhdistää järjestelmän toiminnallisuuden ja laatuominaisuuksia koskevat vaatimukset, jotka järjestelmän on täytettävä (Dragoni, Giallorenzo, Lafuente, Mazzara, Montesi, Mustafin & Safina, 2017).

Teknologian kehitys on mahdollistanut uusien ja parempien työkalujen käytön myötä monipuolisempia ja tehokkaampia tapoja lähestyä järjestelmien ja sovellusten kehittämistä. Järjestelmien kehittämiseen liittyvän teknologian kehittymiseen liittyy keskeisesti halu ratkaista ongelmia, mitä aiemmissa

arkkitehtuureissa on ilmentynyt, ja nämä ongelmat eivät liity välttämättä liiketoimintaan, vaan ne voivat liittyä myös puhtaasti teknisiin puoliin.

Liiketoiminnan kehittyminen taas vaikuttaa arkkitehtuurien evoluutioon asettamalla uusia vaatimuksia, joita järjestelmiltä ja sovelluksilta vaaditaan. Liiketoimintojen muutokset voivat koskea joko yksittäistä yritystä, joka haluaa muuttaa tai laajentaa toimintaansa, tai muutos voi kattaa koko toimialaa, mikä on huomattavasti hitaampi muutos. Esimerkkinä koko toimialaa kattavasta muutoksesta on elokuvien vuokraus, jonka suunnannäyttäjä Netflix antoi kilpailijoilleen vaihtoehdoiksi seurata heidän esimerkkiään, tai poistua hiljalleen toimialalta, koska vanhat liiketoimintamallit elokuvien vuokrausalalla eivät voineet kilpailla Netflixin kanssa.

Järjestelmäarkkitehtuuri kuvailee sovellusjärjestelmän rakennetta ja käyttäytymistä. Järjestelmä määritetään arkkitehtuurissa kokonaisuutena, joka koostuu komponenteista, niiden välisistä yhteyksistä ja niiden käyttäytymiseen liittyvistä vuorovaikutuksista. Järjestelmäarkkitehtuurin luominen antaa paremman ymmärryksen järjestelmästä kokonaisuutena, mikä nopeuttaa suunnittelua. Lisäksi hyvä arkkitehtuuri mahdollistaa järjestelmän huolellisen analyysin, jonka seurauksena voidaan havaita virheitä tai ongelmia jo suunnitteluvaiheessa, ja niiden ratkaiseminen johtaa sekä korkealaatuisempaan sovelluksen koodiin että tarkempaan vaatimusten täyttymiseen. (Mens, Demeyer, Barais, Le Meur, Duchien & Lawall, 2008)

Ohjelmistokehittäminen on jatkuvan mallintamisen ja hiomisen prosessi. Arkkitehtuurillisten ongelmien havaitseminen ajoissa mahdollistaa resurssien säästämisen, kun muutoksia tarvitsee tehdä mahdollisimman vähän olemassa olevaan koodiin. Kaikkia ongelmia ei pystytä havaitsemaan suunnitteluvaiheessa, ja puutteiden ilmentyminen ja niiden korjaaminen suunnitteluvaiheen jälkeen on luonnollinen ja välttämätön osa sovelluskehitystä, ja se mahdollistaa sovelluksen toiminnan parantamisen ja elinajan pidentämisen. (Del Rosso, 2006)

Mitä myöhemmin muutoksia tehdään, sitä hankalampaa ne ovat toteuttaa (Del Rosso, 2006). Osa ongelmista voi ilmentyä vasta myöhemmässä vaiheessa joko teknisempien ongelmien esiintyessä, tai vielä myöhemmässä vaiheessa, kun asiakas tuo esille jonkin puutteen tai tarkennuksen. Mitä aiemmin huolen aiheet havaitaan, sitä kustannustehokkaampaa muutoksien tekeminen on.

Muutokset eivät myöskään välttämättä liity liiketoiminnan muutoksiin tai teknisiin kehityksiin, vaan ne voivat liittyä arkkitehtuurin päivittämiseen, jotta vältetään arkkitehtuurin rappeutuminen sitä ylläpidettäessä. Arkkitehtuuria voi olla hankalaa ylläpitää, kun siihen lisätään uusia osia, joihin alkuperäinen arkkitehtuuri ei välttämättä sovi. Tämän seurauksena arkkitehtuuria muutetaan, jotta sitä olisi kevyempää laajentaa ja ylläpitää. On taloudellisesti epäkannattavaa muuttaa arkkitehtuuria, jos siitä ei ole suoraa hyötyä käyttäjälle. Se on kuitenkin välttämätöntä järjestelmän evoluution kannalta. (Del Rosso, 2006).

Erityisen monimutkaista arkkitehtuurin muutoksien tekeminen on, jos arkkitehtuuri koskee useita eri tuotteita, kuten tuoteperheitä, joissa uudet tuotteet rakennetaan saman arkkitehtuurin pohjalta. Saman arkkitehtuurin käyttäminen tuoteperheille mahdollistaa nopeampien tuotteiden kehittämisen ja

komponenttien kierrätyksen. Liiketoiminnan ja teknologian kehityksen myötä järjestelmältä vaaditaan jatkuvasti uusia asioita, joten on erityisen tärkeää ylläpitää järjestelmän evolutiivisia mahdollisuuksia, jotta yritys ei ajaudu tilanteeseen, missä se joutuu korvaamaan järjestelmän kokonaan, mikä on erityisen suuri ongelma, jos kyseessä on tuoteperhearkkitehtuuri. (Del Rosso, 2006)

Järjestelmän laadun ja sen jatkuvan olemassaolon varmistaminen ovat arkkitehdin tärkeimpiä tehtäviä. Arkkitehdin tehtävänä on neuvotella ja tasapainottaa sidosryhmien keskenään ristiriitaisia vaatimuksia. Lisäksi arkkitehdin tulee ennakoida sidosryhmien mahdollisia vaatimusten muutoksia, eli järjestelmän evoluutiota. (Mens ym., 2008)

Mens ja muut (2008) jakavat järjestelmän muutokset kahteen osaan: sisäiseen evoluutioon ja ulkoiseen evoluutioon. Sisäinen evoluutio kuvastaa komponenttien ja niiden välisten kanssakäymisten muutoksia reaktiona sisäisiin muutoksiin, kuten toisten komponenttien kehittämiseen tai topologian muutoksiin. Sisäinen muutos on olennainen osa järjestelmien dynaamisuutta, kun uusia osia luodaan ja vanhoja poistetaan. Ulkoinen evoluutio liittyy taas komponenttien ja niiden kanssakäymisten tarkempaan määrittelyyn reaktiona järjestelmän ulkopuolisiin tekijöihin, mikä kuvastaa arkkitehtuurin kykyä adaptoitua evoluution vaatimuksiin. Ulkoisen evoluution keskeisenä aiheena on järjestelmäarkkitehtuurin ongelmien eritteleminen. Ongelmien eritteleminen tapahtuu yleensä hyödyntämällä modulaarisuutta tai kapsulointia. (Mens ym., 2008)

Modulaarisuus tai modularisaatio tarkoittavat tapaa, millä pyritään kehittämään järjestelmän joustavuutta ja ymmärrettävyyttä sekä vähentämään järjestelmän kehittämiseen käytettävää aikaa. Modularisaatio tarkoittaa järjestelmän jakamista moduuleihin. Alun perin moduulit nähtiin vastuualueina, vaikka nykyään ne hahmotetaan pieniksi ohjelmistoiksi järjestelmän sisällä. Modulaarisuuden keskeisimpiä hyötyjä on mahdollisuus kirjoittaa järjestelmän osa ilman täyttä tietämystä toisen moduulin koodista. Toinen keskeinen kehitysaskel modulaarisuudessa on mahdollisuus muokata ja korvata moduuleita ilman koko järjestelmän uudelleen kokoamista. (Parnas, 1972)

2.2 Järjestelmäarkkitehtuurien kehitys

Alun perin järjestelmäarkkitehtuuri keskittyi ratkomaan ongelmia liittyen staattisiin järjestelmäarkkitehtuureihin. Vielä 1980-luvulla suurin osa sovelluksista oli monoliitteja. Monoliiteiksi kutsutaan järjestelmiä, jotka on toteutettu monoliittiarkkitehtuuria noudattaen. Näitä monoliittijärjestelmiä pyöritettiin yksittäisellä tietokoneella ja sovelluksia kehitettiin yksittäisiä tehtäviä toteuttavina ohjelmina (program). (Woods, 2016)

Monoliittiarkkitehtuuri tarkoittaa järjestelmäarkkitehtuuria, jossa sovellus tai järjestelmä koostuu yhdestä suuresta koodipohjasta tai tietovarastosta, joka tarjoaa useita eri palveluita käyttäen erilaisia rajapintoja, kuten HTML-sivuja, Web-palveluita ja/tai REST-palveluita (Villamizar ym., 2016).

Monoliittiarkkitehtuuri on yleinen ja luonnollinen tapa rakentaa sovellusta tai järjestelmää sen yksinkertaisuuden takia.

Monoliittiarkkitehtuuri on yleinen tapa aloittaa sovelluksen tai järjestelmän kehittäminen sen suoraviivaisuuden ansiosta. Monoliittijärjestelmä kehitetään ja julkaistaan yhtenä kokonaisuutena, joka sisältää kaikki siihen tarvittavat komponentit. Tyypillisesti monoliittijärjestelmä sisältää käyttöliittymäkerroksen, liiketoimintalogiikkakerroksen ja tiedonhakukerroksen, joka kommunikoi tietokannan kanssa. (Kalske, Mäkitalo & Mikkonen, 2018)

Monoliittinen arkkitehtuuri mahdollistaa yksinkertaisen tavan kehittää, julkaista, testata ja skaalata sovellusta, kun sen koodipohja on suhteellisen pieni (Richardson, 2017). Järjestelmää tai sovellusta kehitettäessä monoliitti voi riittää arkkitehtuuriksi, jos koodin määrä ei kasva liikaa. Jos liiketoiminnan tarpeet ovat yksinkertaiset ja asiakkaiden yhtäaikainen lukumäärä ei ole suuri, on monoliittiarkkitehtuuri looginen lähestymistapa.

Suuren skaalan ohjelmistokehittämisen ongelmat huomattiin jo 1960-luvulla (Brooks, 1995), ja viittaukset ohjelmistoarkkitehtuurien konsepteihin alkoivat ilmestyä 1980-luvulla (Dragoni ym., 2017). Sovellusten monimutkaisuuteen liittyvät ongelmat johtivat oliokeskeiseen ajattelutapaan, jossa järjestelmää hahmotettiin jakamalla se abstrakteihin komponentteihin, joita kutsuttiin objekteiksi. Oliokeskeisen suunnittelun keskeisenä tavoitteena oli paremmin mallintaa sovelluksia, jotta niiden osia voitaisiin käyttää uudestaan sen kehityksen aikana. Tämä ominaisuuksien periminen yksinkertaistaa ja nopeuttaa sovellusten ja järjestelmien kehittämistä. (Capretz, 2003)

Rentsch (1982) määritti oliokeskeisen ohjelmoinnin keskeisiksi ominaisuuksiksi perimisen, kapsuloinnin, metodit, sekä viestit. Oliot kommunikoivat käyttäen samaa mekanismia, ja periminen mahdollistaa ominaisuuksien jakamisen, kun oliot luokitellaan luokkiin, aliluokkiin (sub-class) ja yliluokkiin (super-class).

Vuonna 1992 julkaistiin ensimmäinen artikkeli ohjelmistoarkkitehtuurista, jossa Perry ja Wolf (1992) käsittelivät ohjelmistoarkkitehtuurin määrittämistä. 1990-luvulla hajautettujen järjestelmien suosio nousi, kun prosessointi kehittyi eräprosessoinnista verkkoprosessointiin, ja kolmitasoisesta asiakas-palvelinarkkitehtuurista tuli yritysjärjestelmien standardi. Kun internet 1990-luvulla otti paikkansa valtavirran teknologiana, organisaatiot saivat uuden työkalun, jolla laajentaa liiketoimintaansa. Järjestelmät alkoivat yksinkertaisina nettisivuina, mutta kehittyivät myöhemmin ottamalla käyttöön käyttöliittymät kuluttajamarkkinoinnin (B2C) ja yritysmarkkinoinnin (B2B) kanssakäymisten prosessointiin. (Woods, 2016)

Ongelmien erittely osiksi johti ajattelutapaan, jota kutsutaan komponenttipohjaiseksi ohjelmistotekniikaksi (Component-based software engineering, CBSE) (Szyperski, Gruntz & Murer, 2002). Komponenttipohjaisen ohjelmistotekniikan ideaalisena tavoitteena on ohjelmiston muodostuminen kolmannen osapuolen komponenteista. Komponentit eli ohjelmiston erilliset osat pystyisivät ideaalisessa tapauksessa liittymään yhteen siten, että niiden väliset konfliktit ratkaistaisiin ennen ohjelmiston käyttämistä. Jotta tämä saavutettaisiin, täytyisi

komponenttien tarjota sekä kokoonpano-operaattorit, että keinot sopivien käyttöönottosopimusten määrittelyyn. (Heineman & Councill, 2001)

Crnkovic (2001) mainitsee, miten perinteinen tapa rakentaa yksittäinen järjestelmä ei huomioi järjestelmän evolutiivisia tarpeita. Komponenttipohjaisen kehittämisen tavoitteena olisi ratkaista ongelma, missä uusi järjestelmä tai sovellus rakennetaan tyhjältä pöydältä. Komponenttipohjaisen ajattelutavan ratkaisuna ohjelmistot koostuisivat valmiista komponenteista, jotka olisivat valmiiksi kehitettyjä ja valmiita integroitaviksi. Yhtenä keskeisenä tavoitteena komponenttipohjaisessa kehittämisessä oli niiden käytettävyyden helppous, mikä oli pinnalla 2000-luvun alussa, koska järjestelmien käyttäjät ja käyttöönottajat eivät olleet enää välttämättä asiantuntijoita. Komponenttipohjainen ajattelutapa sai tuulta alleen teknologian kehittymisen myötä 2000-luvun vaihteessa. Siinä oli kuitenkin teknologian uutuudesta ja siihen liittyvästä kokemattomuudesta johtuvia ongelmia, kuten komponenttien kehittämiseen käytettävä aika ja vaiva, epäselvät vaatimukset, käytettävyyden ja uudelleenkäytettävyyden konfliktit, ylläpidon kustannukset, sekä luotettavuus ja sensitiivisyys muutoksiin (Crnkovic, 2001).

Evans (2003) näki objekteja tärkeämpänä teknisenä tekijänä liiketoiminnan. Hänen mukaansa ilman hyvin toteutettua liiketoimintalogiikkaa ei muuten hyvin toteutetulla arkkitehtuurilla olisi väliä. Liiketoimintalähtöinen suunnittelu (domain-driven design, DDD) on mallipohjainen kehittämismenetelmä, jonka ohjaavia periaatteita ovat rajatut organisaatiokontekstit ja jatkuva ohjelmistojen integrointi. Domain tässä kontekstissa tarkoittaa sekä verkkotunnusta että liiketoiminta-aluetta, ja tämä kahden osa-alueen yhdistäminen ja yhdessä pitäminen on keskeinen arvo. Jatkuva integrointi on seurausta kohdealueen, liiketoiminnan ja järjestelmän vaatimusten paremmasta ymmärryksestä. Liiketoimintalähtöinen suunnittelu suosii myös modularisointia ja kapsulointia, mutta siinä painotetaan käyttöliittymän ongelmien erilaisuutta verrattuna muihin järjestelmään liittyviin ongelmiin, ja niitä ei voi välttämättä ratkaista samalla tavalla teknisesti, vaan ne vaativat erilaista osaamista. (Evans, 2003)

Oliokeskeinen suunnittelu ja komponenttipohjainen ohjelmistotekniikka toimivat molemmat pohjana palvelukeskeiselle tietotekniikalle (service-oriented computing, SOC), jonka tarkoituksena on auttaa hallitsemaan hajautettujen järjestelmien monimutkaisuutta ja eri ohjelmistosovelluksien integrointia (MacKenzie, Laskey, McCabe, Brown, Metz & Hamilton, 2006). Palvelukeskeistä arkkitehtuuria (service-orientated architecture, SOA) pidetään ensimmäisenä versiona tästä ajattelutavasta (Dragoni ym., 2017). SOA pohjautuu myös verkkotunnuslähteiseen suunnitteluun (Valipour, Amirzafari, Maleki & Daneshpour, 2009).

SOA jakaa järjestelmän koostumaan palveluista sekä erottaa palveluiden toteuttamisen käyttöliittymästä. Palvelut ovat modulaarisia ja itsenäisiä. Uutena ominaisuutena aiempiin lähestymistapoihin verrattuna SOA antaa joustavuutta tarjoamalla vaihtoehtoja, kun valitaan käytettäviä teknologioita palvelun tarjoajien ja käyttäjien suhteen. Teknologiset kehitykset rajapintojen vakaudessa olivat SOA:n evoluution taustalla. Tämän seurauksena palvelut voidaan eristää, mikä rajoittaa muutosten laajuutta ja kustannuksia. (Valipour ym., 2009)

SOA:n tarkoituksena oli myös hyödyntää paremmin kehittyneitä verkkoyhteyksiä ja sen avulla käsitellä tehokkaammin palvelimelle tulevia pyyntöjä, joiden lukumäärä kasvoi huomattavasti. SOA:n ongelmiin lukeutuivat sen riippuvuus vanhoista monoliittijärjestelmistä, minkä seurauksena asiakkaiden tai liiketoiminnan tavoitteet eivät täyttyneet. (Salah ym., 2016.) Muita ongelmia olivat löydettävyyden ja palvelusopimusten epämääräiset vaatimukset (Dragoni ym. 2017).

Mikropalvelut nähdään toisena versiona SOA:n ja SOC:n konseptista, ja sen tarkoituksena on karsia monimutkaisuutta ja keskittyä ohjelmoimaan yksittäisiä palveluita, jotka toteuttavat yksittäisiä toimintoja (Dragoni ym. 2017). Mikropalveluiden lähestymistapa ohjelmistojen ja systeemiarkkitehtuurien toteuttamiseen perustuu vakiintuneeseen modulaarisuuden konseptiin, mutta keskittyy teknisiin rajoihin (Jamshidi, Pahl, Mendonca, Lewis ja Tilkov, 2018).

Jo ennen mikropalveluiden nousemista suurempaan suosioon Amazon ja Netflix olivat kehitelleet omia arkkitehtuurejaan, joiden keskiössä olivat samanlaiset ideat kuin mikropalveluissa myöhemmin. Amazonin Werner Vogels kuvaili heidän arkkitehtuurinsa lähestymistapaa seuraavasti: ”tieto kapsuloidaan dataa käyttävään liiketoimintalogiikkaan, johon pääsee käsiksi ainoastaan julkaistun palvelurajapinnan kautta” (Gray, 2006). Adrian Cockroft (2016) viittasi Netflixin arkkitehtuurin olevan ”löyhästi kytketty palveluorientoitunut arkkitehtuuri rajatuilla konteksteilla”.

Mikropalvelut muistuttavat SOA:a siten, että molemmat käyttävät itsenäisiä palveluita selkeiden rajojen kanssa. Voidaan myös sanoa, että mikropalvelut ovat SOA:n alatyyppi, mutta eroavat SOA:sta siten, että mikropalvelut hyödyntävät kevyempiä teknologioita, kun taas SOA hyödyntää raskaampia teknologioita kuten yrityspalveluväyliä (enterprise service bus, ESB). Lisäksi SOA nähdään enemmän integraatioratkaisuna, kun taas mikropalveluita käytetään yksittäisten ohjelmistojen rakentamiseen. (Jamshidi ym., 2018)

Tämän luvun aikana on tuotu esille, miten järjestelmäarkkitehtuuri on eri tekijöiden vaikutuksesta saanut alkunsa, ja seuraavassa luvussa käsitellään mikropalveluita tarkemmin aloittaen määritelmästä ja käyden läpi mikropalveluiden keskeisiä ominaisuuksia, vahvuuksia ja haasteita.

3 MIKROPALVELUT

Toinen teorialuku käsittelee mikropalveluita, ja se on jaettu kolmeen alalukuun. Ensimmäisessä alaluvussa käsitellään mikropalveluiden määritelmää ja keskeisiä piirteitä. Toisessa alaluvussa käsitellään teknologioita, jotka liittyvät mikropalveluiden toteuttamiseen. Näihin kuuluvat teknologiat, jotka mahdollistivat mikropalveluiden toteuttamisen, sekä teknologiat, jotka ovat tulleet mikropalveluiden jälkeen tekevät mikropalveluiden käytöstä helpompaa ja tehokkaampaa. Kolmannessa alaluvussa käydään läpi erilaisia mikropalveluihin liittyviä hyötyjä ja haasteita, joita tulee punnita valittaessa mikropalvelua järjestelmän tai sovelluksen arkkitehtuuriksi.

3.1 Mikropalveluiden määritelmä

Mikropalveluarkkitehtuurityyli on lähestymistapa, jossa yksi sovellus kehitetään pienistä palveluista koostuvana kokonaisuutena. Jokainen palvelu toteuttaa omaa prosessiaan ja kommunikoi kevyiden mekanismien avulla. Yleensä tämä mekanismi on HTTP-resurssirajapinta (HTTP-API). (Fowler & Lewis, 2014)

Yrityssovellukset koostuvat yleensä kolmesta osasta: käyttöliittymästä, tietokannasta ja palvelimenpuoleisesta sovelluksesta. Käyttöliittymä koostuu HTML-sivuista ja javascriptistä käyttäjän koneen selaimella. Tietokanta koostuu useista taulukoista, jotka on lisätty yhteiseen relaatiotietokannan hallintajärjestelmään. Palvelimenpuoleinen sovellus käsittelee HTTP-pyyntöjä, suorittaa toimialueen logiikkaa, noutaa ja päivittää tietoa tietokannasta sekä valitsee ja täyttää HTML-näkymän lähetettäväksi selaimelle. (Fowler & Lewis, 2014)

Monoliittimallissa tämä palvelimen puoleinen sovellus on yksi ainoa looginen suoritettava tiedosto, mistä tulee nimitys monoliitti. Jokainen muutos systeemiin vaatii, että palvelimenpuoleisesta sovelluksesta rakennetaan ja julkaistaan uusi versio. (Fowler & Lewis, 2014)

Mikropalveluiden ymmärtäminen on helpointa vertaamalla sitä monoliittiseen lähestymistapaan. Monoliittinen tapa rakentaa palvelin on perinteinen ja

luonnollinen valinta, koska kaikki loogiset pyynnöt kulkevat yksittäisen prosessin läpi, mikä mahdollistaa sovelluksen jakamisen yksinkertaisesti luokkiin, funktioihin ja nimialueisiin. (Fowler & Lewis, 2014)

Monoliittisen arkkitehtuurin vahvuuksina pidetään sen kehittämisen, testaamisen ja käyttöönoton yksinkertaisuutta. Heikkoutena puolestaan on se, että suureksi kasvaneen monoliittijärjestelmän koodi voi olla liian monimutkainen ja hankala käsittää, mikä hidastaa ja hankaloittaa ongelmien ratkaisua ja ominaisuuksien lisäämistä. (Li ym., 2019)

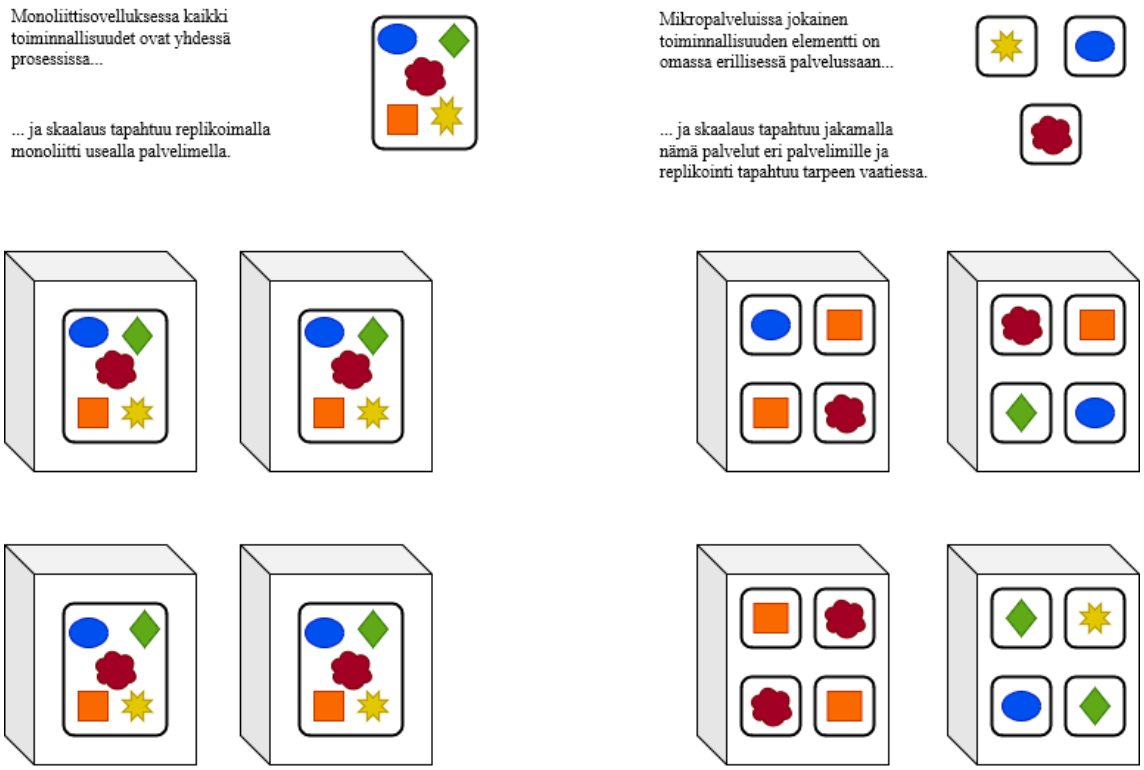
Monoliittijärjestelmissä esiintyy ongelmia myös niiden siirtyessä pilveen. Sen lisäksi, että koko systeemi pitää rakentaa ja julkaista uudestaan, on pidemmän päälle hankalaa ylläpitää hyvää modulaarista rakennetta, minkä seurauksena on vaikeaa pitää yhtä moduulia koskevat muutokset vain kyseisessä moduulissa. Lisäksi skaalaamiseksi ei riitä vain yksittäisten ja enemmän resursseja vaativien osien skaalaaminen, vaan koko sovellus on skaalattava. (Fowler & Lewis, 2014)

Kalske, Mäkitalo ja Mikkonen (2018) pitävät mikropalveluita hyvänä ratkaisuna hallita tietokannan monimutkaisuutta ja kokoa, kun ohjelmistoalan yritys kasvaa tarpeeksi isoksi ja tietokannan koko käy ongelmalliseksi. Mitä suurempi ja vanhempi tietokanta on, sitä monimutkaisemmaksi se muuttuu, minkä vuoksi sen ylläpitäminen ja päivittäminen on haastavaa. Ajan myötä yhä useampi henkilö on tehnyt tietokantaan muutoksia, ja koon myötä osien lukumäärä sekä osien välisten suhteiden lukumäärä kasvaa, minkä seurauksena pienetkin muutokset voivat vaatia paljon aikaa ja resursseja. Kalske, Mäkitalo ja Mikkonen (2018) näkevät myös, että muutoksen tuomat ongelmat voivat olla helpompia ratkaista verrattuna monoliittiarkkitehtuurin aiheuttamiin ongelmiin.

Mikropalveluarkkitehtuurin tavoitteena on ratkaista näitä ongelmia koostumalla palveluista, jotka ovat itsenäisesti käyttöönotettavia ja skaalattavia. Lisäksi jokainen palvelu tarjoaa moduulirajapinnan, mikä mahdollistaa eri palveluiden kirjoittamisen eri ohjelmointikielillä ja niiden hallitseminen eri tiimien toimesta. (Fowler & Lewis, 2014)

Yksittäistä mikropalvelua käsitellään mustana laatikkona, joka sisältää yhden liiketoiminta toiminnallisuuden ja yhden tai useamman verkkopäätepisteen. Mikropalvelun sisäiset tekniset ratkaisut ovat tuntemattomia ulkopuolisille, ja tämän seurauksena mikropalvelut omistavat omat tietokantansa. (Newman, 2021)

Kuvio 1 visualisoi monoliittiarkkitehtuurin ja mikropalveluarkkitehtuurin välisiä eroja.



Kuvio 1 Monoliitit ja mikropalvelut (Fowler & Lewis, 2014)

Fowler ja Lewis (2014) huomauttavat, miten mikropalveluiden juuret ovat Unixin kehitysperiaatteissa, eikä mikropalveluille ei ole yhtä formaalia määritelmää, mutta ne voidaan kuvailla yhdeksän mikropalveluille yhteisen piirteen avulla.

Ensimmäinen piirre on komponentointi palveluiden avulla. Komponentti määritellään ohjelmiston yksiköksi, joka voidaan korvata ja päivittää itsenäisesti. Sen lisäksi, että palvelut ovat prosessin ulkopuolisia komponentteja, palveluita ei kutsuta muistissa olevien funktiokutsujen avulla, kuten kirjastokomponentteja, vaan ne kommunikoivat esimerkiksi verkkopalvelupyynnön tai etäproseduurikutsun kaltaisen mekanismin avulla. (Fowler & Lewis, 2014)

Toinen piirre on organisoiminen liiketoimintamahdollisuuksien ympärille. Suurta ohjelmistoa kehitettäessä on yleistä jakaa tiimi osiin ohjelmiston eri osien perusteella, kuten esimerkiksi käyttöliittymätiimi, palvelimesta vastaava tiimi ja tietokantatiimi. Mikropalveluissa tiimien jakamista lähestytään eri tavalla: tiimit jaetaan siten, että tiimi kattaa kaikkien osa-alueiden osaamisen ja täten omistavat kaikki kehitystyössä tarvittavat taidot. Tämän tyylinen tiimien jakaminen on hankalampaa esimerkiksi monoliittiarkkitehtuurissa, koska komponenttien rajat ovat vähemmän selviä. (Fowler & Lewis, 2014)

Mikropalveluille ominaista on myös tuoteajattelu projektiajattelun sijaan, mikä on kolmas ominainen piirre mikropalveluille. Projektiajattelulle yleistä on kehityksen jälkeen projektitiimin hajottaminen ylläpidon siirtyessä toiselle organisaatiolle. Mikropalveluissa ajatellaan tiimin olevan vastuussa ohjelmistosta koko sen elinkaaren ajan. Tuoteajattelu liittyy aiemmin mainittuun

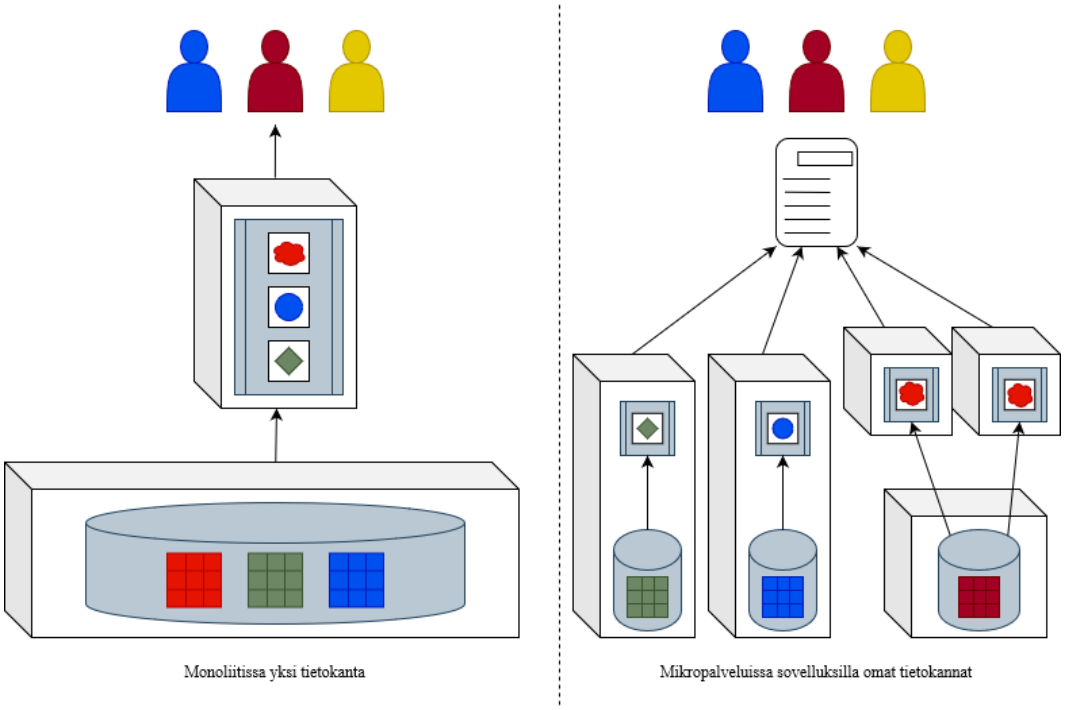
liiketoimintamahdollisuuksien ympärille organisoimiseen siten, että tuote nähdään jatkuvana suhteena, jossa on tavoitteena jatkuvasti parantaa ohjelmistoa liiketoimintamahdollisuuksien parantamiseksi. Tuoteajattelu on toteutettavissa myös esimerkiksi monoliittiarkkitehtuurissa, mutta mikropalveluiden pienempi rakeisuus, eli ohjelmiston jakaminen selkeisiin ja pienempiin osiin, helpottaa kehittäjien ja käyttäjien välisen persoonallisen suhteen muodostumista. (Fowler & Lewis, 2014)

Eri prosessien välistä kommunikaatiota suunnitellessa monet lähestymistavat panostavat erityisen paljon kommunikaatiomekanismiin itseensä. Esimerkkinä tästä on Enterprise Service Bus (ESB), joka sisältää kehittyneitä toimintoja viestien reitittämiseen, koreografiaan, muuntamiseen, sekä liiketoimintasääntöjen toteuttamiseen. Mikropalveluyhteisö suosii kommunikaation lähestymistapana älykkäitä päätepisteitä ja yksinkertaisia putkia (smart endpoints and dumb pipes), mikä on neljäs piirre. Mikropalveluista rakennetut sovellukset pyrkivät olemaan mahdollisimman erillisiä ja yhtenäisiä. Palvelut omistavat oman toimialueensa logiikan ja toimivat klassisessa Unix-mielessä enemmän suodattimina: ne vastaanottavat pyynnön, soveltavat logiikkaa tarpeen mukaan ja tuottavat vastauksen. Palveluiden organisoinnissa käytetään yksinkertaisia REST-protokollia eikä niinkään monimutkaisia protokollia kuten esimerkiksi WS-Choreographya tai BPEL:ää. Kaksi yleisimmin käytettyä protokollaa ovat http-pyynnöt ja vastaukset resurssi-API:n kanssa sekä kevyt viestinvälitys. Toinen yleinen lähestymistapa on viestinvälitys kevyen viestiväylän kautta. Käytettävä infrastruktuuri on yleensä yksinkertainen, eli toimii vain viestien reitittimenä, esimerkkeinä mainitaan RabbitMQ ja ZeroMQ. Älykkäät päätepisteet, eli palvelut, muodostavat ja vastaanottavat viestejä. Monoliittijärjestelmät kommunikoivat yleensä joko metodi- tai funktiokutsuilla, minkä seurauksena kommunikaation muuttaminen onkin suurimpia haasteita monoliittiarkkitehtuurista mikropalveluihin siirtyessä. (Fowler & Lewis, 2014)

Viides piirre on hajautettu hallinnointi. Keskitetyllä hallinnoinnilla on tapana standardisoitua yhden teknologian alustoille, mikä rajoittaa mahdollisuuksia hyödyntää erilaisia työkaluja. Mikropalveluita hyödynnettäessä pyritään käyttämään laajempaa kirjoa teknologioita ongelmien ratkaisemiseen, sen sijaan että seurattaisiin liian tiukasti standardien määrittelemiä rajoittavia vaihtoehtoja. Esimerkkinä Netflixin tapa jakaa koodia kirjastoina rohkaisee muita kehittäjiä ratkaisemaan samanlaisia ongelmia samoilla keinoilla, tai mahdollisesti keksimään oman soveltavan ratkaisun kirjastojen pohjalta. Githubin yleistyessä versionhallinta tarkoituksissa ovat myös avoimen lähdekoodin ratkaisut kasvaneet suosiossa. (Fowler & Lewis, 2014)

Tiedon hallitsemisen hajauttaminen on kuudes piirre, missä mikropalvelut suosivat datan säilömistä siten, että jokaisella palvelulla on oma tietokanta. Täten jokaisella palvelulla on vastuu omasta tietokannastaan, mikä mahdollistaa tietokantojen käyttävän joko samaa tietokantateknologiaa tai kokonaan eri tietokantasysteemiä. Hajautetuissa tietokannoissa tulee ottaa huomioon epäjohdonmukaisuus ongelmien ratkaisemisessa ja päivitysten käsittelyssä. Vaikka tästä voi seurata hankaluuksia, on vaihtokauppa sen arvoinen, jos virheiden

korjaamisesta aiheutuvat kustannukset ovat pienemmät kuin suuremman johdonmukaisuuden myötä menetetyn liiketoiminnan kustannukset. Kuvio 2 visualisoi hajautettua tiedon hallintaa. (Fowler & Lewis, 2014)



Kuvio 2 Tiedon hallinnan hajauttaminen (Fowler & Lewis, 2014)

Seitsemäs piirre on infrastruktuurin automaatio. Pilvipalveluiden, erityisesti Amazon Web Servicen (AWS), kehittymisen ansiosta mikropalveluiden rakentaminen, julkaiseminen ja operoiminen on yksinkertaisempaa, mikä on johtanut infrastruktuurin automaatiotekniikoiden kehittymiseen. Testien automatisoinnin lisäksi pyritään automatisoimaan ohjelmiston vieminen seuraavaan kehitysympäristöön julkaisuputkessa (build pipeline). Kun automatisointiin on panostettu tarpeeksi paljon, on useamman ohjelmiston käyttöönotto (deployment) merkittävästi yksinkertaisempaa. Lisäksi eri palveluiden käyttöönottoon liittyvät ongelmat ovat helpompia käsitellä automaation avulla. (Fowler & Lewis, 2014)

Ohjelmistojen rakentuessa palveluista pitää ottaa huomioon, kuinka ohjelmisto selviää yksittäisten palveluiden mahdollisista vioista. Suunnittelu mahdolliset viat huomioon ottaen on kahdeksas piirre. Mahdollisia vikoja ovat esimerkiksi palvelukutsun epäonnistuminen, kun palveluntarjoajaan ei saada yhteyttä, minkä asiakasohjelman (client) pitää käsitellä mahdollisimman huolellisesti. On tärkeää havaita viat mahdollisimman nopeasti ja automatisoida palvelun palauttaminen toimintakuntoon. Mikropalveluohjelmistoissa panostetaan sekä ohjelmiston arkkitehtuurin että liiketoiminnalle relevanttien mittareiden reaaliaikaiseen monitorointiin. Ongelman huomioiminen on tärkeää, koska mikropalvelut suosivat koreografiaa ja tapahtumayhteistyötä, mitkä johtavat helposti hitaasti esiintyviin käytöksen muutoksiin (emergent behaviour). On myös tärkeää

säilyttää läpinäkyvyys ja tieto kokonaistilanteessa erityisesti, kun ohjelmisto koostuu useasta eri osasta. (Fowler & Lewis, 2014)

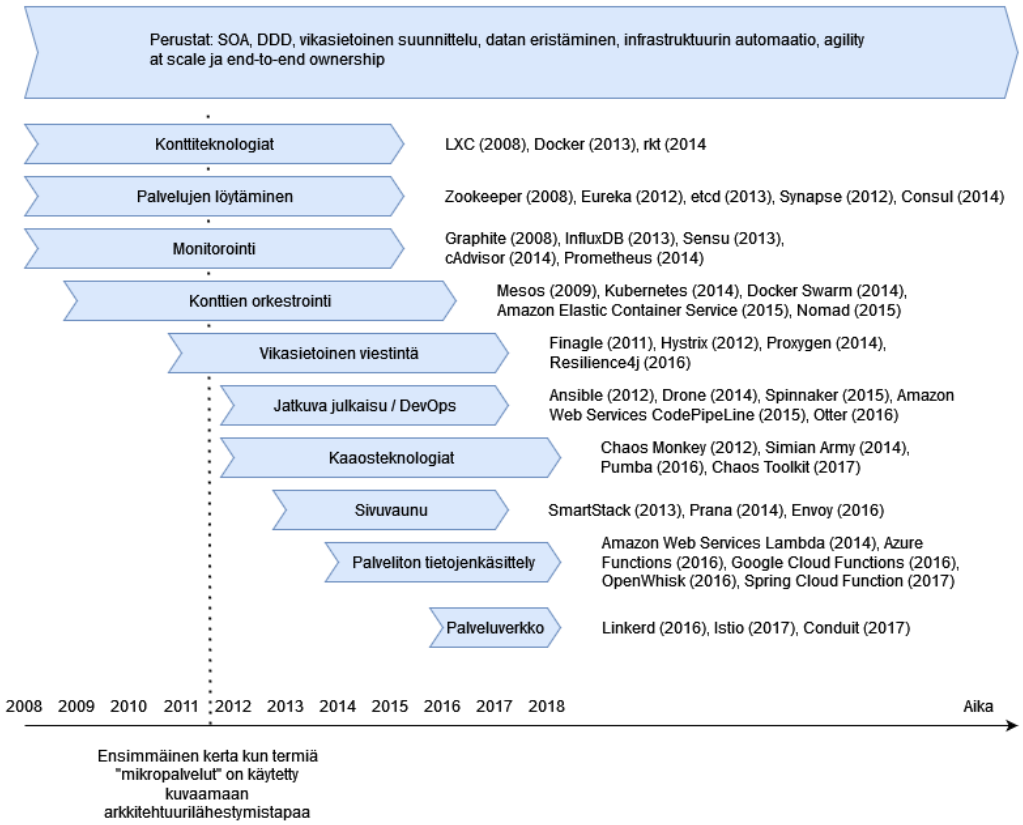
Viimeinen ja yhdeksäs piirre on evolutiivinen suunnittelu. Ohjelmistoa jaettaessa komponenteiksi kohdataan ongelma rajojen vetämisessä. Komponentin keskeisenä ominaisuutena nähdään riippumaton korvattavuus ja päivitettävyys. Missä kohtaa komponentti voidaan korvata ilman, että se koskee muita komponentteja? Esimerkiksi The Guardianin nettisivujen ytimenä on monoliittijärjestelmä, mutta siihen lisätään toimintoja mikropalveluina, jotka käyttävät monoliittijärjestelmän rajapintaa (API). Tämän ansiosta toiminnot voidaan poistaa niiden käytön jälkeen tai niitä voidaan kehittää itsenäisesti eteenpäin. Tämä korvattavuuteen keskittyminen on erikoistapaus modulaarisen suunnittelun periaatteesta, jonka tavoitteena on edistää modulaarisuutta muutoskuvion avulla (Beck, 2007). Samaan aikaan muutettavat asiat tulisi pitää samassa moduulissa, ja harvoin muutettavat asiat tulisi säilyttää erillään aktiivisesti muutoksen alla olevista moduuleista. (Fowler & Lewis, 2014)

Mikropalveluille keskeisten piirteiden lukumäärä on herättänyt kysymyksiä, erityisesti ei-toiminnallisten vaatimusten tarpeellisuutta on kyseenalaistettu (Li ym., 2019). Bushong ja muut (2021) näkevät ongelman olevan dynaaminen ja liittyvän mikropalveluiden jatkuvaan uudelleen modularisointiin, mikä monimutkaistuu vielä enemmän uusien teknologian kehitysten myötä, mistä esimerkkeinä esimerkiksi palvelittomuus ja mikroselainpuolet (microfrontends).

3.2 Mikropalveluihin liittyviä teknologioita

Teknologian kehitys on vaikuttanut keskeisesti siihen, että mikropalvelut ovat saaneet alkunsa. Esimerkiksi erot mikropalveluiden ja sitä edeltäneen arkkitehtuurin SAO:n välillä liittyvät keskeisesti niiden välisten teknologioiden eroihin. Teknologia on kehittynyt myös mikropalveluiden jälkeen, mikä on tuonut sovelluskehittäjille ja arkkitehdeille työkaluja, joiden avulla mikropalveluita voidaan toteuttaa tehokkaammin ja helpommin.

Jamshidi ja muut (2018) näyttävät Kuviossa 3 aikajanan mikropalveluiden kehityksestä. Kuvio sisältää kymmenen teknologia-aaltoa, jotka ovat vaikuttaneet mikropalvelusovellusten kehittämiseen, julkaisemiseen ja operoimiseen.



Kuvio 3 Mikropalveluiden teknologioiden aikajana (Jamshidi ym., 2018)

Ensimmäisten viiden aallon teknologiat olivat olemassa jo ennen mikropalveluita. Ensimmäinen aalto sisältää kevyitä konttiteknologioita (esimerkkinä Docker). Konttiteknologiat mahdollistavat palveluiden tehokkaamman pakkaamisen, käyttöönoton ja hallinnan sovelluksen ajonaikana. Toiseen aaltoon kuuluvat palveluiden havaintoteknologiat (service discovery), joiden avulla palvelut voivat kommunikoida keskenään viittamatta eksplisiittisesti verkkopaikkoihin. (Jamshidi ym., 2018)

Kolmanteen aaltoon kuuluvat monitorointiteknologiat, joiden avulla voidaan tehdä ajonaikaista monitorointia ja analyysia mikropalveluiden käyttäytymisestä. Neljänteen aaltoon kuuluvat konttiorkestrointiteknologiat, kuten Kubernetes, jotka automatisoivat konttien jako- ja hallintatehtävät, minkä ansiosta sovelluskehittäjien ei tarvitse puuttua taustalla olevaan fyysiseen tai virtuaaliseen infrastruktuuriin. Viides aalto koostuu vikasietoisen viestinnän kirjastoista, joiden avulla palvelut voivat kommunikoida tehokkaammin ja luotettavammin. (Jamshidi ym., 2018)

Viisi jälkimmäistä aaltoa syntyivät reaktiona mikropalveluiden suosioon, joista kuudes aalto koostuu jatkuvan julkaisun (continous delivery) teknologioista, jotka tarjoavat integraatioratkaisuja DevOps käytänteiden automatisointiin. Seitsemäs aalto koostuu kaaosteknologioista (chaos engineering), jotka automatisoivat kriittisten, koko järjestelmän kattavien, luotettavuus- ja tietoturvatestauksen tekniikoita. Kahdeksas aalto koostuu sivuvaunuteknologioista (sidecar),

jotka kapsuloivat kommunikaatioon liittyviä toimintoja siirtääkseen ne pois sovelluskehittäjien vastuulta. Yhdeksäs aalto koostuu palvelittomista tietojenkäsittelyteknologioista (serverless computing), jotka implementoivat pilvipalvelumallin toiminto-palveluna (funtion-as-a-service, FaaS). FaaS antaa pilvikäyttäjien kehittää, käyttöönottaa ja toimittaa tuotantoon hienojakoisempia palvelutoiminnallisuuksia ilman monimutkaisuuksia, jotka liittyvät niiden toteuttamiseen tarvittavien infrastruktuuriresurssien luomiseen ja hallintaan. Esimerkiksi sovelluskehittäjien ei tarvitse selvittää epäjohdonmukaisia liikennemalleja. Viimeinen aalto koostuu palveluverkkotekniikoista (Service Mesh), jotka hyödyntävät sivuteknologioita tarjotakseen integroidun palvelusta palveluun -viestinnän seuranta- ja hallintaympäristön. Melkein kaikki Kuviossa 3 luetellut teknologiat ovat teollisuuslähtöisiä, mutta siitä huolimatta suurin osa niistä on julkisesti avointa lähdemateriaalia. (Jamshidi ym., 2018)

Salah, Zemerly, Yeun, Al-Qutayri ja Al-Hammadi (2016) tuovat artikkelissaan esille neljä keskeistä asiaa mikropalveluiden hyödyntämisessä. Mikropalvelut sijaitsevat virtuaalikoneissa, mikä tekee niiden skaalaamisesta tehokasta. Virtuaalikoneiden ja niiden resurssien käytön hallintaa varten kehitettiin konttiteknologiat, joista Docker keskittyy kevyiden konttien rakentamiseen ja hallitsemiseen. Konttiteknologiat paketoivat yksittäisen palvelun yhdessä sen riippuvuuksien kanssa, mitä kutsutaan koodin siirrettävyydeksi (Kang, Le & Tao, 2016).

Konttien hallintaa varten taas on kehitetty Cluster Manager, joka aikatauluttaa konttien ja mikropalveluiden toiminnan siten, että resurssien käyttö on tehokasta, käyttäjän sijoitterurajoituksia noudatetaan, estetään odottavan tilan läpikulkemisen ja palvelut ovat aina saatavilla (Salah ym., 2016). Newman (2021) mainitsee Kubernetesin ja CoreOS:n avoimen lähdekoodin Cluster Managereina.

Virtuaalikoneiden ja konttiteknologian jälkeen kolmantena aiheena on palvelun löytäminen. Palvelun löytäminen pyrkii tietämään, mitä tietyssä ympäristössä on käynnissä, jotta jokainen palvelu, joka on riippuvainen toisesta palvelusta, on tietoinen toisen palvelun sijainnista (Salah ym., 2016). Newman (2021) mainitsee tähän tarkoitukseen seuraavat teknologiat: Zookeeper, Consul ja Eureka.

Neljäntenä ja viimeisenä Salah ja muut (2016) painottavat viestintä viitekehyksen tärkeyttä. SOA:ssa käytetty SOAP ei ole tehokas mikropalveluille, koska mikropalvelut ovat pieniä ja vaativat kevyitä työkaluja, kuten REST-rajapintaa. Useimmat sovelluskehittäjät valitsevat REST-rajapinnan kanssa käytettäväksi tiedostomuodoksi JSON:än, koska se on yksinkertainen ja kevyt. (Salah ym., 2016)

3.3 Mikropalveluiden hyötyjä ja haasteita

3.3.1 Mikropalveluiden hyötyjä

Mikropalveluiden keskeisimpiä hyötyjä on itsenäinen käyttöönotto. Vuosituhannen vaihteessa julkaiseminen tuotantoon (production release) oli harvinaisempi ja työläämpi prosessi, kun taas nykypäivänä taitavat tiimit voivat julkaista

tuotantoon usean kerran päivässä. Tämä muutos liittyy keskeiseen piirteeseen mikropalveluissa eli itsenäiseen käyttöönottoon, jonka palvelut komponentteina mahdollistavat. Testaus, julkaisu ja mahdolliset virheet koskevat vain yhtä osaa ohjelmistosta, eivätkä kaada koko järjestelmää. Nopea ja itsenäinen käyttöönotto antaa yritykselle mahdollisuuden reagoida muuttuviin markkinoihin nopeasti ja ennen kilpailijoita. Nopeasti toistuva julkaiseminen ei ole ainoa syy käyttää mikropalveluita, koska monoliittimalliset ohjelmistotkin voidaan julkaista nopeasti ja jatkuvasti, mistä Facebook toimii esimerkkinä, ja mikropalveluiden julkaiseminen voi epäonnistua huolellisen koordinaation puutteen seurauksena. (Fowler, 2015)

Newman (2019) pitää myös itsenäistä käyttöönottoa mikropalveluiden tärkeimpänä ominaisuutena ja suurimpana hyötynä. Lisäksi Newman (2019) pitää erittäin keskeisenä ja hyödyllisenä ominaisuutena mikropalveluiden joustavuutta, mikä on seurausta käytetyn teknologian valinnan vapaudesta.

Mikropalveluissa voidaan käyttää mitä tahansa kieliä, kirjastoja ja tietovarastoja. Tämän ansiosta tiimi voi valita ongelmanratkaisuun parhaiten sopivan työvälineen välittämättä yhteensopivuudesta muiden osien kanssa. Ongelmanratkaisun lisäksi teknologiavapaus tuo mukanaan myös paremman versionhallinnan. Monoliittimallissa kirjastosta voi olla olemassa vain yksi versio, mikä johtaa siihen, että yksittäistä osaa monoliittimallista ei voi päivittää ilman että toiset osat lakkaavat toimimasta. Versionhallinnan tuomat ongelmat kasvavat koodin määrän kasvaessa, mikä on suurimpia ongelmia monoliittimallisissa ohjelmistoissa. Teknologian valinnan vapaus voi johtaa ongelmiin liian monien vaihtoehtojen myötä, joten käytettäviä teknologioita on hyvä rajoittaa jossain määrin, jotta kokonaisuus pysyy yhtenäisenä. Tämä vapaus tuo kuitenkin loputtomasti mahdollisuuksia kokeilla eri vaihtoehtoja ja korvata vanhoja komponentteja uusilla ja paremmilla teknologioilla mahdollisuuksien ilmaantuessa. (Fowler, 2015)

Tätä teknologian vapauden valintaa koodaamisen kielessä sanotaan monikieliseksi ohjelmoinniksi (polyglot programming) ja teknologian valinnan vapautta tietokantojen valinnassa kutsutaan monikieliseksi pysyvyydeksi (polyglot persistence). Monikielinen pysyvyys mahdollistaa sen, että yksi tietokanta voi prosessoida yhtä osaa ohjelmistosta ja käyttää samaa dataa, jota toiset tietokannat käyttävät saman sovelluksen toisen osan käsittelyyn. (Khine & Wang, 2019)

Mikropalveluiden etuna ovat myös vahvat moduulien rajaukset. Kun ohjelma jaetaan osiin tai moduuleihin, on hyödyllistä pystyä muokkaamaan yksittäistä komponenttia tai osaa, ilman että muutos vaatii syvempää ymmärrystä koko ohjelmistosta. Tämä korostuu eksponentiaalisesti ohjelmiston ja sitä kehittävän tiimin kokojen kasvaessa. Moduulien välinen raja toimii muurina moduulien välisille viittauksille, mutta monoliittijärjestelmässä tämä muuri on helpompi kiertää oikoteiden tekemistä varten, mikä johtaa pidemmällä aikavälillä ja ohjelmiston koon kasvaessa arkkitehtuurin heikentymiseen ja kehitystyön hankaloitumiseen. Moduulien välisten rajojen vahvuus ei ole mikropalveluille uniikki ominaisuus, tai mahdotonta toteuttaa monoliittimallissa, mutta

mikropalvelut tekevät sen luontaisemmin esimerkiksi hajauttamalla tiedon hallinnan, mikä oli yksi mikropalveluiden pääpiirteistä. (Fowler, 2015)

Pahl ja Jamshidi (2016) sanovat artikkelissaan hyvin määritettyjen ja selkeiden rajapintojen olevan olennainen osa mikropalveluita. Koska kommunikaatio tapahtuu rajapintojen kautta, ei osien vaihtaminen vaikuta muihin osiin. Tämä johtaa löyhään kytkentään ja korkeaan yhteenkuuluvuuteen, eli usea palvelu yhdistyy toisiinsa muodostaakseen korkeamman tason palveluiden määrittämiseksi. Tämä toiminnallinen hajauttaminen mahdollistaa esimerkiksi ketteryyttä, joustavuutta ja skaalautuvuutta. (Paul & Jamshidi, 2016)

Pilvipalveluiden kehityksen ansiosta automaattinen resurssien skaalaaminen on helppoa ja kustannustehokasta, ja mikropalvelut hyötyvät tästä erityisen paljon (Kalske, Mäkitalo & Mikkonen, 2018). Skaalautuvuuden parantuminen koskee teknisen puolen lisäksi myös tiimien toimintaa. Sovelluskehitystiimit pystytään pitämään pienempinä ja ketterämpinä, mikä mahdollistaa ketterien (agile) tiimien toiminnan vieläkin itsenäisemmäksi, mikä taas mahdollistaa ketterien metodien skaalaamisen. Tiimien itsenäisyys kasvaa myös tiimien välisen kommunikaation vähentymisen seurauksena, mikä vie huomattavasti aikaa suuren skaalan ohjelmistokehityksessä. (Blinowski, Ojdowska, Przybylek, 2022)

Tämä vastuun ja itsenäisyyden lisääminen yksittäisille tiimeille sopii nykyaikana yhä useammalle yritykselle. Projekteja kehitetään globaalisti ja tiimit voivat sijaita eri puolilla maapalloa. Maantieteelliset etäisyydet voivat hidastaa kommunikaatiota, mutta mikropalveluiden avulla kommunikaation ei tarvitse olla yhtä nopeaa, koska tiimien tarvitsee huolehtia osien välisistä rajapinnoista. Lisäksi koodi on todennäköisemmin puhtaampaa ja ongelmien ratkaisu helpompaa. (Kalske, Mäkitalo & Mikkonen, 2018)

Pienempänä hyötynä mikropalveluilla on myös mahdollisuus parantaa ohjelmiston turvallisuutta jakamalla sensitiivistä tietoa sisältävät omiin osiinsa, jotka voidaan turvata huolellisemmin kuin ohjelmiston muut osat. Mikropalveluiden itsenäinen rakenne lisää turvallisuutta luonnollisesti, ja turvallisuuteen liittyvien huolien kasvaessa mikropalveluiden arvo voi kasvaa vielä enemmän. Hyöty koetaan pieneksi, koska monoliittimalleissa sensitiivisen datan käsittely voidaan myös hoitaa erillisinä palveluina. (Fowler, 2015)

3.3.2 Mikropalveluiden haasteita

Mikropalveluiden suurimpana haasteena pidetään monoliittiarkkitehtuurin jakamista sopivan kokoisiksi kokonaisuuksiksi mikropalveluita varten, koska osiin jakaminen tapahtuu yleensä manuaalisesti (Kecskemeti, Marosi & Kertesz, 2016). Mikropalveluiden suunnittelu tehdään yleensä intuitiivisesti ja perustuen kokemukseen, koska mikropalvelurajojen määrittämiseen ei ole systemaattista lähestymistapaa (Li ym., 2019). Mikropalveluita koskeva kirjallisuus on vielä uutta ja käytännössä testattuja menetelmiä on vähän, minkä seurauksena niistä ei ole vielä muodostunut yleisesti hyväksyttyä mallia.

Mikropalveluiden rajojen määrittämisen lisäksi Newman (2019) kiinnittää kirjassaan huomion ongelmiin, jotka liittyvät mikropalveluiden väliseen kommunikaatioon, mikä tarkoittaa verkkoyhteyksien tuomia haasteita.

Mikropalveluiden tarkoituksena on parantaa modulaarisuutta jakamalla järjestelmää osiin, mutta tämän mukanaan tuoma monimutkaisuus aiheuttaa ongelmia. Ensimmäinen ongelma on suorituskyky, mikä voi esiintyä etäkutsujen kasaantuessa viiveiksi, koska kutsuttavien osien lukumäärä voi olla erittäin suuri. Jos ongelmaa yrittää ratkaista vähentämällä kutsuja, törmää ongelmaan kutsujen yhdistämisessä, mikä ei siltikään poista viivettä, joka syntyy siitä, että jokaista osaa pitää kutsua erikseen. Asynkroninen eli aikariippumaton kommunikaatio tarkoittaa sitä, että kutsujen ei tarvitse odottaa toisiaan. Siten viive on kiinni vain hitaimmasta kutsusta eikä kutsujen viiveen summasta. Asynkroninen ohjelmointi on kuitenkin hankalaa ja vielä hankalampaa on sen debugaus eli vikojen määrittäminen. (Fowler, 2015)

Toinen ongelma systeemin osiin jakamisessa on luotettavuus, koska prosessin sisäinen kutsu on luotettavampi kuin etäkutsu, mikä voi epäonnistua milloin tahansa. Palveluiden määrän kasvaessa myös tämän riski kasvaa. Mikropalveluiden yhtenä keskeisenä piirteenä onkin tämä vikojen huomioon ottava ohjelmointi, mikä liittyy asynkroniseen ohjelmointiin. Tämä ei poista niihin liittyvä monimutkaisuutta ja niiden ohjelmoinnin hankaluutta tai sitä, että jokainen etäkutsu tulee ottaa huomioon. Näihin samoihin ongelmiin törmätään kuitenkin myös monoliittimalleissa, jotka kommunikoivat erillisten perintö (legacy) järjestelmien kanssa, mikä on johtanut mikropalveluiden puoleen kääntymiseen. (Fowler, 2015)

Mikropalveluiden keskeisenä ominaisuutena on sovelluskehittämisen ja suunnittelemisen hajauttaminen, mutta sen seurauksena arkkitehtuurillista yhtenäisyyttä on hankala säilyttää. Arkkitehtuurillinen yhtenäisyyden puute johtaa arkkitehtuurilliseen hajoamiseen, mikä tarkoittaa sitä, että koodin muutokset saavat arkkitehtuurin poikkeamaan alkuperäisestä suunnitelmasta. (Bushong ym., 2021)

Hajauttaminen johtaa myös lopulliseksi johdonmukaisuudeksi (eventual consistency) kutsuttuun ilmiöön. Ongelma ilmenee, kun yksi osa ohjelmistosta saa päivityksen ennen toista, mutta pyynnön käsittelee osa, joka ei ole vielä saanut päivitystä. Tämän seurauksena käyttäjä saa väärää informaatiota, mutta ongelmaa ratkaistaessa ongelma onkin jo korjaantunut itsestään toisen osan päivittyessä, mikä johtaa ongelmiin ongelman ratkaisussa. Ongelma kasvaa vakavammaksi, jos liiketoimintalogiikka tekee päätöksen väärän informaation perusteella, ja mikropalvelut kasvattavat todennäköisyyttä törmätä tähän ongelmaan hajauttamalla tiedon hallinnan, mikä johtaa useiden osien päivittymiseen yhden sijaan. Monoliittimallikin törmää samoihin ongelmiin, kun se käyttää välimuistia suorituskyvyn parantamiseksi, ja monoliittimallitkin joutuvat päivittämään ulkopuolisia järjestelmiä. Lisäksi liiketoimintalogiikka sietää yllättävän hyvin epäjohdonmukaisuuksia, koska ne priorisoivat eniten saatavuutta. (Fowler, 2015)

Ongelmat epäjohdonmukaisuuksissa esiintyvät myös ohjelmiston näkymän eroissa. Asiakas ja asiakastuki voivat nähdä eri näkymän ohjelmistosta, esimerkiksi jotkin asiat voivat puuttua kokonaan tai muuttujat voivat erota toisistaan tai tarkoittaa eri asiaa. Ongelma voi esiintyä sekä eri sovellusten välillä tai

sovelluksen sisällä, jos se on jaettu erillisiin komponentteihin. (Fowler & Lewis, 2014)

Mikropalveluiden osiin jakamisen seurauksena jokainen erillinen osa lisää riskiä yhdenmukaisuusongelmiin, jotka aiheuttavat haasteita sovelluksen kehittämiselle. Lisäksi ongelmia luo myös mikropalveluiden kyky skaalautua tarpeen mukaan lisäämällä tai poistamalla kopioita mikropalveluista. Tämän seurauksena mikropalveluiden elinaika ei ole välttämättä kovin pitkä, mikä lisää dynaamisuutta ja monimutkaisuutta sovelluksen kehittämiseen. (Sampaio ym. 2017)

Osiin jakaminen on myös raskasta ja monimutkaista sovelluskehittäjille, kun kymmenet sovellukset muuttuvat sadoiksi mikropalveluiksi. Automaation hyödyntäminen helpottaa tässä ongelmassa, mutta osiin jakamisessa piilee myös vaara virheiden määrittämiseen liittyen. Aiemmin komponenteissa ilmentyneet viat voivat siirtyä komponenttien välisiin suhteisiin, mitä vahvat palveluiden rajat voivat ehkäistä. Mikropalveluiden hyödyntäminen vaatii kehittäjiltä erilaisia ja uusia taitoja sekä DevOps kulttuuria, joka tarkoittaa pyrkimistä tiiviimpään yhteistyöhön kehittäjien, operaatioiden ja kaikkien muiden ohjelmistotoimitukseen osallistuvien osapuolien välillä. (Fowler, 2015) Salah ynnä muut (2016) ovat samaa mieltä Fowlerin (2015) kanssa siitä, miten mikropalvelut haastavat tiimien osaamista ja kommunikointikykyä. Tiimien välinen kommunikaatio kasvaa tärkeydessä mitä erilaisempia eri osat ovat, ja eri teknologioiden käyttäminen lisää monimutkaisuutta yhteistyössä tiimien välillä (Salah ym., 2016).

Mikropalveluita on kritisoitu myös testaamisen hankaloitumisesta verrattuna monoliittimallisiin sovelluksiin, mutta Fowler (2015) näkee mikropalveluiden testaamiseen käytettävät työkalut niin hyvinä, että testaamisen vakavasti ottaminen on merkittävämpää kuin erot monoliittijärjestelmän ja mikropalveluiden testaamisessa.

Blinowski, Ojdowska ja Przybylek (2022) arvioivat artikkelissaan monoliittimallin ja mikropalveluiden välisiä eroja, ja heidän tutkimuksessaan tuli esille, että yksittäisellä laitteella monoliittijärjestelmä toimii mikropalvelua tehokkaammin. Tutkimuksen mukaan monoliittijärjestelmä on parempi ratkaisu yksinkertaisille ja pienille järjestelmille, joiden ei tarvitse palvella suurta määrää yhtäaikaisia käyttäjiä.

Newman (2019) mainitsee mikropalveluiden olevan huono ratkaisu, jos järjestelmän toimialue on epäselvä, kyseessä on startup-yritys, tai tuote on asiakkaan hallinnassa. Toimialueen epäselvyys johtaa hankalasti määritettäviin rajoihin, mikä johtaa riskiin määrittää rajat väärin, mikä taas on huonompi lopputulos kuin monoliittiarkkitehtuurissa pysyminen. Startup-yrityksille mikropalveluita ei suositella, koska useat startup-yritykset kokeilevat erilaisia liiketoimintaratkaisuja alkuvaiheessa, koskien sekä tuotteita ja asiakasryhmää. Mikropalvelut sopivat paremmin yritykselle, joka haluaa skaalata olemassa olevaa järjestelmää. Esimerkiksi Netflix ja Airbnb ottivat mikropalvelut käyttöön vasta myöhemmässä vaiheessa kummankin yrityksen evoluutiota. Asiakkaan hallinnassa oleva ohjelmisto taas on ongelmallista muuttaa mikropalveluksi, koska mikropalvelut vaativat muutosta toteuttavalta tiimiltä korkean taitotason, sekä tiimin on

kyettävä sopeutumaan uusiin haasteisiin ja teknologioihin mikropalveluita käytettäessä, mikä on paljon pyydetty asiakkailta. (Newman, 2019)

Kun mietitään mikropalveluiden käyttämistä, tulee ottaa huomioon, että mikropalvelut aiheuttavat kustannuksia, jotka maksavat itsensä takaisin vain monimutkaisissa järjestelmissä. Jos järjestelmän monimutkaisuus on hallittavissa monoliittisessa mallissa, voivat mikropalveluiden tuomat ongelmat olla suurempia kuin sen hyödyt. Kehitystyötä tekevän tiimin taidot sekä kommunikaation ja yhteistyön laatu vaikuttavat loppujen lopuksi enemmän kuin päätös monoliittiarkkitehtuurin ja mikropalveluiden välillä. Teknisesti tärkeimpiä puolia ovat puhdas koodi, hyvä testaus ja evolutiivinen arkkitehtuuri. (Fowler, 2015)

4 MENETELMÄ

Tämän luvun ensimmäisessä alaluvussa käsitellään valittu tutkimusmenetelmä ja sen valinnan perustelut sekä kohdeorganisaatio, josta haastateltavat valittiin. Toisessa alaluvussa käsitellään haastatteluiden rakenne läpinäkyvyyden ja toistettavuuden saavuttamiseksi.

4.1 Tutkimusmenetelmän valinta ja kohdeorganisaatio

Tutkimus toteutettiin laadullisena tutkimuksena, koska tarkoituksena on ymmärtää mikropalveluiden laatua, ominaisuuksia ja merkitystä kokonaisvaltaisesti. Laadullinen tutkimus toteutetaan haastatteluiden muodossa, koska tutkimuksen tavoitteena on saada syvempää ymmärrystä aiheesta sekä siihen liittyvistä mielipiteistä, suhtautumisista ja kokemuksista (Rowley, 2012). Tutkimuksen teoriaosio antaa tutkimuskysymyksille näkökulman tieteellisen kirjallisuuden puolesta, ja empiirisen osion puolistrukturoidut haastattelut antavat materiaalia, jonka perusteella pystytään muodostamaan alan ammattilaisten näkökulma.

Puolistrukturoitu haastattelu sopii erittäin hyvin tapauksiin, jossa useampi kysymys on avoin ja vaatii jatkokysymyksiä. Puolistrukturoitua haastattelua suositellaan myös erityisen paljon, jos halutaan tietää jokaisen haastateltavan itsenäisiä ajatuksia aiheesta. Puolistrukturoitu haastattelu on myös hyvä valinta haastattelun muodoksi, jos tavoitteena on muodostaa arvioita jostain tietystä aiheesta, ja halutaan haastatella yhtä asiantuntevaa henkilöä kerrallaan. (Newcomer, Hatry ja Wholey, 2015)

Näiden kirjallisuudessa esiintyvien syiden lisäksi puolistrukturoidut haastattelut valittiin haastatteluiden muodoksi, koska niiden avulla haastattelut pystyttiin toteuttamaan kevyemmässä ilmapiirissä ja ennemminkin rentona keskusteluna aiheesta kuin pistokokeena. Haastattelut toteutettiin etänä Zoomissa, ja haastattelut tallennettiin analysointia varten. Haastateltavat valittiin kohdeorganisaatiosta eli Eduixista.

Eduix Oy on suomalainen IT-alan yritys, joka tarjoaa digitaalisia ratkaisuja korkeakoulujen ja kuntien tarpeisiin. Eduix suunnittelee, toteuttaa ja toimittaa digitaalisia työkaluja koulutuksen tarpeisiin ja kuntien sähköiseen asiointiin. Yli 50 000 opiskelijaa käyttää Eduixin kehittämiä tuotteita päivittäin. Yrityksellä on yli 40 työntekijää ympäri Suomea ja päätoimisto sijaitsee Tampereella. (Eduix Oy n.d.) Eduix ei ole käyttänyt mikropalveluita järjestelmissään, mutta haastateltaviksi valituilla kolmella järjestelmäarkkitehdilla on kokemusta mikropalveluista.

Haastateltavat valittiin yrityksen henkilöstöstä heidän mikropalvelutietämyksensä perusteella. Rowley (2012) tuo esille, miten haastateltavia valittaessa tulee kysyä ”kuka on sellaisessa asemassa, että hän pystyisi vastaamaan kysymyksiini tai antamaan syvempää ymmärrystä aiheeseen?”. Haastateltavaksi pohdittiin henkilöitä eri työtehtävistä, mutta lopulta tutkimuksessa päädyttiin haastattelemaan kolmea henkilöä, jotka kokivat tietävänsä tarpeeksi mikropalveluista, jotta haastatteluiden tuloksia pystyttäisiin vertaamaan järkevästi tieteelliseen kirjallisuuteen. Tutkimukseen olisi voitu haastatella henkilöitä muistakin työtehtävistä ja huonommalla ymmärryksellä mikropalveluista, mutta se ei antaisi realistista kuvaa siitä, ymmärtävätkö aiheen ammattilaiset mikropalvelut samalla tavalla kuin ne ymmärretään tieteellisessä kirjallisuudessa. Lisäksi se ei antaisi realistista kuvaa yrityksen mikropalveluosaamisesta, koska jokaisen yrityksen jäsenen ei tarvitse omistaa tietämystä mikropalveluista, jotta niiden käyttöä voidaan miettiä. Kaikki kolme haastateltavaa toimivat ohjelmisto- ja järjestelmäarkkitehdin työtehtävissä, ja arkkitehdit ovat keskeisin osapuoli yrityksessä tietämään mikropalveluarkkitehtuurista.

4.2 Haastatteluiden toteutus

Haastattelut toteutettiin puolistrukturoidun haastattelun ohjeen mukaisesti (Newcomer, Hatry ja Wholey, 2015). Haastateltaville ilmoitettiin etukäteen tutkimuksen aihe ja miten haastattelu toteutetaan. Lisäksi haastateltaville ilmoitettiin etukäteen, että haastattelu nauhoitetaan ja heidät anonymisoidaan haastattelun aineistoa esiteltäessä ja että tallenteet poistetaan tutkimuksen päätyttyä.

Haastattelun teemat valittiin tieteellisen kirjallisuuden ja teoriaosion perusteella.

1. Järjestelmäarkkitehtuuri ja niiden kehitys
2. Mikropalvelut
3. Mikropalveluiden toteuttamiseen liittyvät teknologiat
4. Mikropalveluiden hyödyt ja haasteet

Haastatteluun ja teemoihin ei ollut realistista sisällyttää kaikkea teoriaosiossa käsiteltyä sisältöä, joten niistä pyrittiin ottamaan aiheiksi keskeisimmät ja olennaisimmat asiat. Lisäksi joissakin haastatteluissa mentiin syvemmälle tiettyihin teemoihin ja aiheisiin riippuen haastattelun etenemisestä ja jäljellä olevasta ajasta.

Ennen teemojen käsittelyä haastateltavilta kysyttiin lyhyesti taustakysymyksiä, joiden avulla pystytään havainnoimaan tiettyjä näkemys- ja mielipideeroja. Teemat käytiin läpi siten, että ensin kysyttiin aiheeseen liittyvä yksinkertainen kysymys, jonka vastauksien perusteella esitettiin jatkokysymyksiä. Kun teemaan liittyvät aiheet oli käsitelty tällä tavalla, esiteltiin haastateltaville tieteellisessä kirjallisuudessa esille tulleita asioita teemasta ja aiheesta. Näiden johdatteluiden seurauksena saatiin haastateltavilta mielipiteitä kirjallisuudessa esiintyneisiin väitteisiin ja mielipiteisiin. Tämä oli erityisen hyödyllistä tapauksissa, missä haastateltava ei välttämättä tiennyt jonkin tietyn aiheen teoriasta tai termistä erityisen paljoa, mutta osasi kommentoida tieteellisessä kirjallisuudessa esille tulleita asioita oman asiantuntijuuteensa perusteella. Johdattelut tehtiin varovaisesti siten, että vältettiin myöhempien teemojen ja aiheiden asioita, jotta myöhemmistä teemoista ja aiheista saataisiin autenttiset vastaukset.

Haastateltaville pyrittiin luomaan rento ja turvallinen keskusteluympäristö kysymällä ensin helppoja kysymyksiä, kuten heidän taustojaan. Haastatteluihin pyrittiin luomaan positiivinen ilmapiiri kysymällä aiheista positiiviset asiat ensin, sekä vältettiin painostamasta haastateltavaa vastaamaan kysymykseen. Haastateltavia tuettiin tuomalla esille esimerkkejä (kuitenkin välttämällä liikaa johdattelua), jos haastateltava tuntui jäävän jumiin johonkin kysymykseen esimerkiksi kysymyksen tai termien epäselvyyden takia. Näin haastateltava pystyi kertomaan mielipiteensä vertaamalla sitä esimerkkiin, ilman että koki painetta siitä, että hänelle ei välttämättä tullut sillä hetkellä aiheesta mieleen sen suurempia ajatuksia, mikä on täysin inhimillistä eikä kuvasta henkilön osaamista asiasta.

Haastattelut hyppivät myös paikoin teemojen välillä, kun keskustelu eteni johonkin aiheeseen, joka olisi tullut myöhemmin käsiteltäväksi, mutta keskustelun sulavana pitämiseksi keskustelu käytiin loppuun saakka aiheesta siitä huolimatta, mihin teemaan se olisi kuulunut, ja alkuperäiseen teemaan palattiin sen jälkeen.

Analysointi tapahtui yleisesti hyväksytyllä tavalla (Rowley, 2012), eli aineisto litteroitiin, koodattiin ja tulkittiin, joiden jälkeen se muotoiltiin esitettävään muotoon. Datan esittelyssä otettiin huomioon eettisiä ongelmia: esimerkiksi datan esille tuomisessa on tärkeää antaa lukijalle kontekstia haastateltavasta, mutta siten, ettei kontekstia anneta liikaa, jottei haastateltavaa pystytä identifioimaan sen perusteella. Lisäksi analyysissa otettiin huomioon vastauksia esitellessä se, että vastaukset muotoiltiin neutraalisti, mutta ilman että ne menettivät merkityksensä.

Haastateltavien henkilöiden määrä oli suhteellisen pieni, mutta tutkimukseen tuotiin lisää täsmällisyyttä ja laatua toteuttamalla toinen kierroshaastatteluita, missä käytiin läpi tarkentavia kysymyksiä ensimmäisten haastatteluiden pohjalta. Toisen kierroksen kysymykset muodostettiin muiden muassa puheenaiheista ja kysymyksistä, mitä ei esiintynyt kaikissa haastatteluissa, ja niihin haluttiin kysyä muiden haastateltavien mielipiteitä. Lisäksi kysymyksiä kysyttiin sen perusteella, että haastatteluissa käsiteltiin joitakin asioita, mihin ei haastatteluissa menty syvemmälle, mutta analyysivaiheessa huomattiin, että niihin olisi hyödyllistä saada tarkennusta. Kysymyksiä muodostettiin myös sen pohjalta,

että ensimmäisellä haastattelukierroksella ei ollut realistista käsitellä kaikkea teoriaosiossa esille tullutta informaatiota, ja haastatteluiden pohjalta opittiin, mihin asioihin haastateltavilta saatiin paremmin vastauksia, ja muodostettiin lisää kysymyksiä näiden ideoiden pohjalta.

5 TULOKSET

Viides sisältöluku sisältää haastatteluissa kerätyn datan analysoinnin. Luku on jaettu alalukuihin teemojen perusteella, ja data järjestetty niiden mukaan. Kolme haastateltavaa henkilöä on anonymisoitu ja heihin viitataan nimillä Arttu, Kari ja Tapio. Kaikki kolme henkilöä ovat työtehtäviltään ohjelmistoarkkitehtejä, mutta tekevät suunnittelun lisäksi aktiivisesti sovelluskehitystä. Kaikilla kolmella on sovelluskehitystaitojensa taustalla harrastuneisuutta ja korkeakouluopintoja, ja Arttu ja Tapio ovat käyneet korkeakouluopintonsa loppuun asti. Tapiolla on vähemmän työkokemusta suhteessa Arttuun ja Kariin. Viimeisenä taustatietona kaikilla kolmella on kokemusta mikropalveluiden ja komponenttipohjaisten arkkitehtuurien käytöstä.

5.1 Järjestelmäarkkitehtuuri yleisesti

Jokainen haastateltava kuvaili järjestelmäarkkitehtuuria suunnitelmana tai kuvauksena, jonka avulla jaetaan informaatiota järjestelmästä ja saavutetaan konsistenssi eli looginen ristiriidattomuus. Esimerkiksi Kari kuvaili arkkitehtuuria yhteisenä tapana kommunikoida ja tehdä asioita, ja arkkitehtuurin ansiosta jokainen osapuoli tietää, mitä komponentteja kokonaisuuteen liittyy sekä miten kokonaisuus toimii. Arttu kommentoi järjestelmäarkkitehtuurin hyödyiksi sen, että järjestelmäarkkitehtuurin lopputuloksena on ymmärrettävämpi ja ylläpidettävämpi kokonaisuus. Lisäksi haastateltavien yhteinen mielipide oli, että arkkitehtuurin avulla järjestelmä jaetaan osa-alueisiin, ja niiden avulla pystytään jakamaan vastuu- ja työalueita. Tapio toi esille, miten arkkitehtuurin tärkeys korostuu, mitä suurempi kokonaisuus on kyseessä ja mitä enemmän kehittäjiä sen toteuttamiseen osallistuu.

Jokainen haastateltava oli sitä mieltä, että arkkitehtuuria kuuluisi noudattaa mahdollisimman paljon, mutta on inhimillistä, että arkkitehtuurista joudutaan poikkeamaan jostain syystä. Tapion toi esille miten tällaisissa tapauksissa on tärkeää miettiä, miksi ollaan poikkeamassa arkkitehtuurista. Kari ja Arttu

mainitsivat, että arkkitehtuurista poikettaessa tulee poikkeukset ja perustelut dokumentoida selkeästi, jotta muut osapuolet ovat tietoisia asiasta.

Kysyttäessä haastateltavien mieltymyksistä joidenkin arkkitehtuurien käyttämiseen, Kari mainitsi palvelupohjaisten arkkitehtuurien olevan hyviä, koska ne ovat tehokkaita, niitä on helppo käyttää ja niiden rajapinnat ovat hyvät. Arttu ja Tapio eivät kommentoineet olevansa mieltyneitä yksittäisiin arkkitehtuureihin, ja Tapio piti arkkitehtuurin valintaa tapauskohtaisena. Hänen mukaansa on tärkeämpää valita arkkitehtuuri ohjelmiston perusteella, perustuen esimerkiksi siihen, kuinka laaja se on tai kuinka paljon käyttäjiä sillä on. Arttu kommentoi samaan tyyliin muistavansa, että arkkitehtuuria on valittu ratkaisemaan tietyn sovelluskentän ongelmia.

Kysyttäessä järjestelmäarkkitehtuurien kehityksestä liittyen mikropalveluihin, Arttu muisti kuulleensa aiemmin urallaan mikropalveluiden tyylisistä arkkitehtuureista, mutta niistä puhuttiin eri termeillä, ja että niiden käyttö on lisääntynyt ajan saatossa. Hän toi esille, miten ohjenuorana on ollut pyrkiä pilkkomaan toimintoja pieniin sopiviin lohkoihin, jotta työt ja vastuut saadaan jaettua ihmisten kesken. Hän mainitsi teknologian kehityksestä esimerkiksi Dockerin, Kubernetesin, pilvipalvelut ja erilaiset viitekehykset, mitkä ovat vaikuttaneet arkkitehtuurien kehitykseen. Kari mainitsi teknologian kehityksestä integraatioiden kasvaneen määrän, mikä tuo skaala- ja tietoturvahaasteita, jotka vaativat enemmän kuin monoliittiarkkitehtuurin. Hän mainitsi teknologioista Dockerin ja Kubernetesin lisäksi Go-ohjelmointikielen. Kari toi esille organisationaalisia tarpeita arkkitehtuurien kehitykselle, esimerkiksi miten saadaan suuret työntekijämäärät toteuttamaan sovelluskehitystä tehokkaasti. Hän pohti haastattelussa myös sitä, miten suuri osa kehityksestä on seurausta akatemiasta, ja kuinka suuri osa on peräisin esimerkiksi Yhdysvaltojen isoista yrityksistä. Erityismainintana hän toi esille Amazonin, joka on keskeinen osapuoli pilvipalvelualalla.

Haastateltaville esitettiin tieteellisessä kirjallisuudessa esiintyneitä arkkitehtuureja, jotka ovat relevantteja mikropalveluihin liittyen. Ensimmäisenä arkkitehtuurina käytiin läpi monoliittiarkkitehtuuri, josta kaikilla oli yhtenevä ymmärrys, eli että kaikki toiminnot ovat yhdessä kokonaisuudessa. Tapio toi esille monoliittiarkkitehtuurin hyvinä puolina sen, että se on yksinkertainen, ja että se voi sopia pienille ohjelmille, joilla ei ole riskiä kasvaa. Monoliittiarkkitehtuuri on myös Karin mielestä tehokas tapa tehdä pieniä järjestelmiä. Kari mainitsi, että monoliittijärjestelmä voi toimia paremmin, jos sitä pitää modulaarisena, mutta ongelmana siinä tulee se, että sitä on teknisesti haastavampaa pitää modulaarisena, koska ihmiset pystyvät helpommin oikomaan rajojen yli, mikä johtaa koodin laadun heikkenemiseen. Monoliittiarkkitehtuurin ongelmina Tapio toi esille skaalautuvuuden ja ylläpidettävyyden puutteet. Hän kommentoi myös, ettei lähtisi käyttämään monoliittiarkkitehtuuria enää tuoteohjelmistossa.

Oliokeskeinen suunnittelu, modulaarisuus ja komponenttipohjainen ohjelmistotekniikka olivat Artulle tuttuja termejä. Tapiolle oliokeskeinen suunnittelu ei herättänyt ajatuksia, mutta komponenttipohjainen ja moduulit olivat hänelle enemmän tuttuja termejä Front End -kehityksestä. Hän mainitsi modulaarisuudesta ideologian, missä eristetään osia ja toiminnallisuuksia, jotka voidaan

kapsuloida omiksi kokonaisuuksiksi. Karille oliokeskeinen ja komponenttipohjainen suunnittelu toivat mieleen ensimmäisenä pelit, jossa ne toimivat hänen mielestään paremmin. Kari piti niitä kätevinä, mutta ei nähnyt niitä yhtä hyödyllisinä tai käytettävinä pelien ulkopuolella.

SOC eli palvelukeskeinen tietotekniikka (service-oriented computing) ei terminä sanonut kenellekään haastateltavista mitään, mutta SOA eli palvelukeskeinen arkkitehtuuri (service-orientated architecture) oli kaikille tuttu. Kerrottaessa SOC:stä Kari kommentoi, että se kuulostaa samanlaiselta kuin EJB (Enterprise JavaBean). Kaikilla haastateltavista oli käytännön kokemusta SOA:sta, ja Tapio kuvaili SOA:ta siten, että jaetaan tarvittavat osat omiin moduuleihin, joita voidaan hyödyntää osana ohjelmistoa. Hänen mielestään SOA on monoliittiarkkitehtuuria parempi ja sen etuna on mahdollisuus kehittää palveluita itsenäisesti toisistaan riippumatta. Hän näki SOA:n puutteena verrattuna mikropalveluihin teknologiariippumattomuuden hankalamman hyödyntämisen.

Toisella haastattelukierroksella haluttiin saada tarkennusta Tapion kommenttiin SOA:n teknologiariippumattomuuden heikompaan hyödyntämiseen. Vastauksena kysymykseen Tapio täydensi vastaustaan seuraavasti: ”Omassa käytössä olleet alustat ja mallit ovat pitkälti estäneet useamman teknologian käyttämisen, koska erilliset moduulit ovat olleet toisistaan vahvasti riippuvaisia”.

Arttu kuvaili SOA:ta siten, että se tarjoaa orkestrointia palvelukomponenteille. Lisäksi Arttu vertasi SOA:ta mikropalveluihin siten, että SOA kokoaa eri komponentit yhteen virtuaalikoneeseen, eikä osaa jakaa eri virtuaalikoneiden kesken. Kirjallisuudessa esitetyt kommentit SOA:sta kuulostivat Artun ja Tapion mielestä järkeviltä ja vastasivat heidän omia kokemuksiansa asiasta.

Kari poikkesi SOA:sta puhuttaessa kahdesta muusta merkittävästi. Kari näki SOA:n ja mikropalveluiden rajan vähemmän selkeänä, eikä ollut samaa mieltä kirjallisuudessa esitetyn väitteen kanssa siitä, että SOA olisi riippuvainen vanhoista järjestelmistä. Hän toi esille esimerkin kokemuksestaan, että monoliittijärjestelmä on tehty uusiksi pilkkomalla se osiin, jonka jälkeen osat eivät kommunikoineet monoliitijärjestelmän kanssa. Lisäksi myös mikropalvelut joutuvat kommunikoimaan legacy-järjestelmien kanssa. Hän oli samaa mieltä siitä, että SOA käytti alkuun raskaita teknologioita, kuten SOAP:ta, viestintäkeinona, mutta hän ei näe syytä, miksi SOA ei voisi myös käyttää kevyempiä teknologioita, kuten JSON:ää. Hän kommentoi haastattelussa myös, miten SOA:ssa käytettiin 2000-luvun alussa ESB:tä (Enterprise Service Bus), ja lisäsi että se liittyi Enterprise Java Beansiin, joka on myös muita palveluita hyödyntävä arkkitehtuuri.

Karin mielipide SOA:n ja mikropalveluiden eroista erosi merkittävästi kirjallisuudesta, ja toisella haastattelukierroksella pyrittiin selvittämään, ovatko muut haastateltavat samaa mieltä asiasta. Tapio kommentoi, että työelämässä tämä pitää pitkälti paikkaansa, ja että aika harvoin päästään tekemään järjestelmä täysin nollasta, eli legacy-järjestelmien kanssa joudutaan kommunikoimaan, kuten The Guardianin esimerkissä, joka esiteltiin myös aiemmin teoria osiossa. Hänen mielestään jälkeenpäin ajateltuna tämä osoittaa hyvin, että koko ohjelmiston ei tarvitse olla mikropalveluarkkitehtuurilla rakennettu ja että mikropalveluissa käytettäviä teknologioita voidaan käyttää myös muissa malleissa. Arttu

kommentoi miten mikä tahansa palvelurajapinta voi piilottaa takanaan toimivan vanhan systeemin.

5.2 Mikropalveluarkkitehtuuri

Jokainen haastateltava kuvaili suhtautuvansa eri tavalla mikropalveluihin. Tapion mielestä mikropalveluissa on paljon hyviä asioita, ja hän lähtisi toteuttamaan laajempaa kokonaisuutta mikropalveluilla, koska hänellä on kokemusta muista arkkitehtuuriympäristöistä, joissa jatkokehitettävyys ja ylläpidettävyys ovat hankalampia toteuttaa. Näissä asioissa mikropalvelut auttaisivat hänen mukaansa merkittävästi. Arttu kuvaili suhtautuvansa enemmän neutraalisti ja kuvaili mikropalveluissa olevan hyviä puolia, mutta mikropalveluita käytettäessä tulisi miettiä tarkkaan kuinka pieniksi osiksi asioita pilkotaan, ja liian pieniksi paloiksi jaettaessa ovat haitat suuremmat kuin hyödyt.

Kari kommentoi suhtautuvansa maltillisesti ja skeptisesti uusiin asioihin, mukaan lukien mikropalveluihin, ja lisäsi näkevänsä useammin palvelupohjaista arkkitehtuuria kuin puhtaita mikropalveluita. Mikropalvelut ovat hänen mielestään hyvä ratkaisu, jos on puhdas tarve mikropalveluille, mutta toi esille kysymyksen, onko normaalissa teollisuudessa ja Suomen väestön mittakaavalla tarvetta käyttää mikropalveluita, koska käyttäjämäärät eivät nouse yhtä suuriksi kuin esimerkiksi Twitterillä ja Facebookilla.

Karin esille tuoma ajatus mikropalveluiden käyttäjien tarpeeksi suuresta määrästä oli mielenkiintoinen, ja toisella haastattelukierroksella pyrittiin selvittämään, ovatko muut haastateltavat samaa mieltä asiasta. Muut haastateltavat olivat samaa mieltä kommentin kanssa, että on tärkeää identifioida tarpeeksi suuri käyttäjämäärä mikropalveluiden käyttökohteeksi.

Haastateltavat ymmärsivät mikropalveluiden määritelmän samalla tavalla, ja Tapio kuvaili mikropalveluarkkitehtuuria seuraavasti: ”Suhteellisen pieniin palvelukokonaisuuksiin pilkottu ohjelmisto ja tavoitteena on, että palvelut ovat itsenäisiä ja niitä voidaan kehittää toisistaan riippumatta, ja niitä voidaan uudelleen käyttää”. Tapio mainitsi myös, miten yksi keskeisiä asioita on teknologian valinnanvapaus, minkä ansiosta voidaan tutkia tehokkaat ratkaisutavat, eikä tarvitse huolehtia yhteensopivuudesta aiemman ohjelmiston kanssa. Kari toi esille, että mikropalvelu toimii itsenäisesti, ja olennaista olisi, että koko järjestelmä ei kaadu, jos yksittäinen mikropalvelu on pois päältä. Lisäksi Kari mainitsi, että mikropalvelu on sitä tehokkaampi mitä yksinkertaisempi sen funktio on, koska mitä useampaa muuta palvelua mikropalvelu kutsuu, sitä enemmän herää kysymys, onko palvelu jaettu liian pieniin osiin. Arttu toi esille keskeisenä ominaisuutena mikropalveluiden ulospäin tarjoaman rajapinnan.

Monoliittiarkkitehtuurin ja mikropalveluarkkitehtuurin eroista haastateltavat olivat myös samaa mieltä. Artun mukaan monoliittijärjestelmässä on monta toimintoa ja rajapintaa, kun taas mikropalvelussa on vain muutama toiminto moduulin sisällä. Tapio ja Kari mainitsivat molemmat, että mikropalveluiden hyötynä monoliittiarkkitehtuuriin verrattuna on joustavampi ylläpidettävyys ja

jatkokehitettävyys, kun järjestelmää ei tarvitse päivittää kokonaan, vaan yksittäinen osa siitä.

Haastateltavat olivat yhtä mieltä myös tapauksista, joihin mikropalveluarkkitehtuuri sopii. Arttu toi esille sen, että organisaation pitää olla tarpeeksi suuri, ja jakaa työt ja vastuu komponenttien mukaan. Tapio mainitsi, miten mikropalvelut sopivat hyvin tapaukseen, jossa vaatimukset muuttuvat ja kehittyvät, koska mikropalveluiden avulla voidaan tehdä muutoksia nopeasti ja joustavasti. Lisäksi mikropalvelut sopivat hänen mukaansa hyvin tapauksiin, joissa käyttäjien määrä ei ole staattinen, koska mikropalveluarkkitehtuuri skaalautuu hyvin. Kari lisää myös, miten tärkeää on, että järjestelmään liitytään ulkopuolelta, koska sisäisessä käytössä esimerkiksi monoliittiarkkitehtuuri voi riittää. Hän lisäsi, että monoliittijärjestelmä toimii myös ulkopuolisessa käytössä tiettyyn käyttäjämäärään asti tehokkaana ratkaisuna. Esimerkiksi Suomessa käytettävä järjestelmä ei välttämättä tavoita käyttäjämäärää, mikä riittäisi ainoaksi syyksi valita mikropalvelut arkkitehtuuriksi.

Haastateltavien kanssa käytiin läpi kirjallisuudessa esitettyjä keskeisiä piirteitä kahdella eri johdattelun tasolla: ensin heille esitettiin piirre yleisesti, ja sen jälkeen heille kerrottiin tarkemmin, miten tieteellinen kirjallisuus ymmärtää nämä keskeiset piirteet.

Komponentoinnista kaikki olivat samaa mieltä kirjallisuuden kanssa. Esimerkiksi Tapio kuvaili komponentoinnin siten, että eristetään yhden mikropalvelun toimintalogiikka omaksi kokonaisuudeksi, joka tarjoaa rajapinnan. Tämän jälkeen muiden sovellusten ei tarvitse tietää siitä tarkemmin, eivätkä muut sovellukset ole alttiita yhden palvelun muutoksille.

Organisoiminen liiketoimintamahdollisuuksien ympärille ei myöskään jakanut mielipiteitä haastateltavien välillä. Haastateltavat näkivät vastuun ja omistamisen tiimeille tiettyjen mikropalveluiden kohdalla hyödyksi, mutta pitivät myös riskinä sitä, että tiimit siiloutuvat liikaa omaan osaansa, mikä voi hankaloittaa toimintaa myöhemmin. Arttu mainitsi, miten kansainvälisessä työskentelyssä tulee panostaa rajapintojen määrittelyssä siihen, millä palvelukutsulla toista palvelua voidaan simuloida, jos se ei ole vielä heti saatavilla. Kari toi esille kysymyksen siitä, minkä kokoinen organisaation olisi oltava, että sen on järkevää myötäillä palveluarkkitehtuurin rakennetta. Tarkemman johdattelun jälkeen Kari kommentoi haastattelussa, miten olisi ihanteellista, että tiimit olisivat täysin itsenäisiä, mutta käytännön ongelmat tulevat esille esimerkiksi silloin, kun kommunikoidaan muutoksista tiimien välillä. Esimerkkinä hän mainitsi, miten useampi palvelu voidaan joutua päivittämään samaan aikaan. Kari ja Arttu mainitsivat, miten taaksepäin yhteensopivat muutokset auttavat tällaisissa tapauksissa, kun palvelu tukee sekä uutta että vanhaa tapaa, minkä ansiosta vanhat palvelut toimivat siitä huolimatta, että uudemmat palvelut käyttävät uusia ja parempia tapoja.

Haastateltavat olivat samoilla linjoilla tuoteajattelusta. Kari kommentoi, miten henkilö, joka toteuttaa ja ylläpitää, oppii myös tehokkaammin omista virheistään, verrattuna siihen, että joku muu tekisi ylläpidon. Hänen mukaansa on myös tärkeää kuunnella asiakasta, koska asiakas on syy, miksi asioita tehdään. Artun

ajatukset haastoivat tieteellistä kirjallisuutta ylläpidon suhteen, kun hän kommentoi, että uusien henkilöiden on tärkeää päästä sisään järjestelmän toimintaan ja kyetä ylläpitämään sitä, koska yhden organisaation sisällä työntekijät vaihtuvat myös. Kirjallisuutta myötäillen hän kommentoi, miten mikropalveluiden keskeinen hyöty eli komponenttien korvattavuus ja kierrättäminen kärsii, jos ylläpito siirtyy eri tiimin käsiin, mutta kyseinen ongelma ei ole uniikki mikropalveluille.

Kevyestä kommunikaatiosta haastateltavat olivat myös samaa mieltä. Tapio kommentoi, miten SOA:ssa kommunikaatio on järeämpää, kun taas mikropalveluissa kommunikaatio on yksinkertaisempaa esimerkiksi viestijonojen avulla.

Hajautettu hallinnointi eli teknologian valinnan vapaus ei myöskään jakanut mielipiteitä, mutta Artulla tuli mieleen mielenkiintoinen huomio, että sen lisäksi, että teknologian käytössä tulee ottaa huomioon sen sopiminen sekä tapaukseen että käyttäjien osaamiseen, tulisi ottaa huomioon, onko teknologialla osaajia myös tulevaisuudessa. Kari mainitsi, miten siitä huolimatta, että teknologian valinnan vapaus on merkittävä hyöty, on tärkeää miettiä myös kokonaisuutta, ettei koodista tule ”Villi länsi ja kukin tekee millä lystää”. Eri teknologioita voidaan maltillisesti lisätä, ottaen huomioon esimerkiksi käyttöönotossa, että suuret erot teknologiassa voivat aiheuttaa ongelmia. Välillä ei hänen mukaansa voi valita parasta vaihtoehtoa, ja on tärkeämpää priorisoida kokonaisuuden toimivuutta, koska ylläpito kestää todennäköisesti pidempään kuin järjestelmän kehittäminen.

Tiedon hallinnan hajauttaminen ei herättänyt suurempaa keskustelua kenenkään haastateltavan kanssa, mutta kaikki olivat samaa mieltä kirjallisuuden kanssa.

Infrastruktuurin automaatio ei myöskään herättänyt suuremmin mielipiteitä, ja haastateltavat olivat samaa mieltä kirjallisuuden kanssa aiheesta. Arttu toi esille hyödyn kehitysympäristön automaattisesta pystyttämisestä, mikä veisi manuaalisesti useita päiviä. Lisäksi hän ja Tapio mainitsivat, miten CI (continous integration) putken ja testauksen automaatio ovat olennaisia asioita nykypäivänä. Kari mainitsi skaalaamisen automaatiosta sen, että hyödyllisyys on suhteessa käyttäjien määrän muutoksiin, koska käyttäjämäärän ollessa stabiili, ei skaalaaminen ole tärkeä aspekti.

Evolutiivinen suunnittelu oli haastateltavien mielestä myös tärkeä asia, ja Arttu kuvaili sitä prosessina, missä vanhaa rajapintaa ei rikota muutoksella, vaan rajapintaan tulee lisäyksiä ja vältytään erilaisilta ongelmilta. Kari kommentoi miten pienempiin osiin jakaminen helpottaa uusien osien tuomista ja vanhojen poistamista, minkä lisäksi rajapinnat helpottavat muutoksia. Haastateltavat pitivät kirjallisuudessa esille tuotua The Guardianin esimerkkiä järkevänä tapana hyödyntää mikropalveluita evolutiiviseen suunnitteluun.

5.3 Mikropalveluteknologia

Haastatteluissa käytiin läpi mikropalveluihin liittyvät teknologiat ensin puhumalla ilman johdattelua keskeisistä teknologioista, minkä jälkeen keskusteltiin tarkemmin kirjallisuudessa esiintyneistä keskeisistä teknologioista.

Arttu ja Tapio toivat esille konttiteknologiat ja konttien orkestrointiteknologiat, joiden avulla pystytään hallitsemaan suurempi määrä kontteja, esimerkkeinä konttien orkestrointiteknologioista Kubernetes, Docker Compose ja Docker Swarm. Lisäksi Artun mielestä koodin kielellä ei ole niin väliä, kunhan REST-rajapinnat ja niiden suhteet on määritelty hyvin. Kari toi esille, miten HTTP, JSON ja REST ovat helpompia käyttää kuin XML, ja lisäsi, että HTTP- pyynnöillä ja REST:n avulla on helpompaa tehdä myös integraatioita. Tapio mainitsi viestintäkeinoiksi RabbitMQ- ja ActiveMQ-tapahtumajonot, joilla pystytään hoitamaan palveluiden välistä kommunikaatiota.

Johdattelun jälkeen Tapio kommentoi, miten palvelunlöytämisteknologia mahdollistaa pyyntöjen välittämisen, jotta hajallaan olevat palvelut löytävät väylät, mitä pitkin pyynnöt välittyvät oikeisiin osoitteisiin. Hän myös mainitsi, että hän on tutkinut aiemmin palvelunlöytämisteknologioista Zookeeperiä, mutta hänen mielestään se oli kankea ja epäkäytännöllinen. Lisäksi hän kertoi käyttäneensä Consulia, mutta eniten hänellä oli kokemusta Ingressistä, mikä on Kubernetesin oma palvelunlöytämisteknologia. Arttu kertoi johdattelun jälkeen, ettei hän ole itse perehtynyt palvelunlöytämisteknologioihin erityisen tarkasti, mutta kuvaili niitä välittäjiksi (proxy), jotka ohjaavat palvelukutsuja oikeaan suuntaan.

REST-rajapinnan Tapio sanoi haastattelussa standardiksi, jota muutkin kuin mikropalvelut käyttävät, mutta hän lisäsi, että jos hän käyttäisi mikropalveluita, tekisi hän siten, että palvelu julkaisisi REST-rajapintoja, joiden kautta palvelua pystyy varsinaisesti käyttämään. Hän toi myös esille, miten jonot ovat enemmän palvelun sisäiseen kommunikaation toimivampia työkaluja. Arttu oli viestintäkeinoista samaa mieltä kirjallisuuden kanssa ja hänelle tuli mieleen JMS, joka tarjoaa asynkronista viestintää.

Kari kommentoi johdattelun jälkeen enemmän konttiteknologioita ja konttiorkestrointia, ja hänen mielestään konttiteknologiat ovat raskaita operoida. Hän kertoi esimerkin siitä, miten palvelinta ylläpitävän henkilön sijaan kaikkien kehittäjien tulee osata pilvipalveluiden rakentamiseen liittyviä ekosysteemeitä. Hänen mielestään olisi tärkeää miettiä, onko tarpeellista käyttää pilvipalveluita ja konttiteknologiaa, jos muita yksinkertaisempia vaihtoehtoja on olemassa. Konttiteknologioiden hyödyksi hän mainitsi sen, että kaikki tekevät asiat samalla tavalla, verrattuna siihen, että jokainen tekisi palvelimensa omalla tavallaan, minkä seurauksena ylläpito kärsisi. Hän toi myös esille sen, että Kubernetesiä käytettäessä ei ole pakko laittaa kontteja pilveen, vaan niitä voisi pyörittää omissa palvelimissa. Kontit tuovat hänen mukaansa kompleksisuutta, mutta hänen mielestään on hyvä, että on standardoitu tapa, jonka ansiosta useamman henkilön on helpompaa toteuttaa ylläpitoa. Nämä hyödyt kasvavat firman koon kasvaessa ja järjestelmän elinkaaren pituuden kasvaessa.

Ensimmäisellä haastattelukierroksella ei käsitelty teoriaosiossa läpikäytyjä kymmentä mikropalveluihin liittyvää teknologian aaltoa, koska ajateltiin, että sen läpikäyminen olisi vienyt liikaa aikaa haastatteluissa. Toisella haastattelukierroksella haastateltaville näytettiin kyseinen Kuvio, eli Kuvio 3, ja heitä pyydettiin kertomaan ovatko he kuulleet tai onko heillä mielipiteitä siinä esille tuoduista teknologioista.

Kari kommentoi, että mainitut teknologiat ovat hänelle tuttuja vähintään nimen tasolla. Hän kertoi, että kuviosta hän ei ole käyttänyt tai kokeillut vikasietoisen kommunikaation teknologioita tai kaaosohjelmointia. Oleellisina hän piti Ansiblea DevOps-teknologioista ja Dockeria konttiorkestroinnista. Muut teknologiat hän näki hyödyllisenä vasta, kun järjestelmä kasvaa huomattavasti isommaksi, koska ne tuovat merkittävän määrän monimutkaisuutta. Esimerkkeinä hän mainitsi, miten pilvipalveluilla julkaiseminen voi olla erittäin monimutkaista ja vaativat täysipäiväisiä ylläpitäjiä ja osaajia, eikä voida enää puhua DevOpsista vaan ”Opsista”, kuten aikanaan oli ”palvelinylläpitäjän” rooli. Arttu kommentoi, että hänelle tuttuja olivat Docker, Docker Swarm, Kubernetes ja Ansible. Hän lisäsi listan ulkopuolelta myös Apache Servicemixin ja ActiveMQ:n. Tapio ei kommentoinut asiaa.

5.4 Mikropalveluiden hyödyt ja haasteet

Tässä alaluvussa käsitellään haastatteluissa esiintyneitä ja läpikäytyjä hyötyjä ja haasteita, jotka käsiteltiin ilman johdattelua, jonka jälkeen aiheet käytiin tarkemmin läpi tuomalla esille kirjallisuudessa esiintyneitä asioita. Jos johdattelun jälkeisissä analyyseissa ei ole käyty läpi jokaisen henkilön mielipidettä tietystä kohdasta, on se joko käsitelty aiemmassa kohdassa tai hän on kirjallisuuden kanssa samaa mieltä, eikä hänellä ole asiasta poikkeavia mielipiteitä.

5.4.1 Hyödyt

Mikropalveluiden tuomista hyödyistä Artulle tuli mieleen se, että mikropalveluihin on helpompi tehdä muutoksia kuin monoliittijärjestelmään. Myös laajentaminen uusilla komponenteilla on helpompaa, eikä se riko vanhaa arkkitehtuuria yhtä helposti. Lisäksi versiohallinta voi olla selkeämpää, kun jokaisella komponentilla on omat versiot, ja arkkitehtuuri voidaan jakaa sopivan kokoisiin osiin selkeiden rajapintojen taakse.

Kari toi esille, miten pienemmän osan porukasta on helpompi kehittää mikropalveluarkkitehtuuria. Lisäksi hänen mukaansa ”tulee vähemmän sotkua siitä, että kaikki lukevat kaikkia tietokantoja, eikä kukaan tiedä, missä mitäkin tapahtuu, kun mikropalvelut omistavat omat alueensa”. Osiin hajauttamisen yhtenä hyötynä on myös high availability eli korkea saavutettavuus, joka tekee järjestelmästä vikasietoisemman, kun yksittäisen osan kaatuminen ei välttämättä estä koko järjestelmää toimimasta.

Tapion mielestä mikropalveluiden suurin hyöty on ylläpidon ja jatkokehitettävyyden helpottuminen. Lisäksi teknologiariippumattomuuden ansiosta voidaan valita teknologia tiettyyn tarkoitukseen ilman, että muut osat sidotaan käyttämään samaa teknologiaa. Skaalautuvuutta Tapio kuvaili haastattelussa seuraavasti: ”Pystytään replikoimaan tiettyä palvelua, jos esimerkiksi tiedetään, että login palvelu on todella ruuhkainen, ja se voidaan replikoida useaan kertaan tuomaan laskentatehoa ja voidaan näin ollen palvella isompia käyttäjämääriä. Samoin toiseen suuntaan, jos jokin palvelu on tiettyyn aikaan hiljaisempi, niin voidaan sammuttaa replikoita, joita ei tarvita.”

Kun haastattelua johdateltiin kirjallisuudessa esiintyneiden hyötyjen avulla, Arttu kommentoi, miten itsenäinen käyttöönotto on mahdollista, koska rajapinnat ovat riippumattomia muista komponenteista. Hän toi kuitenkin esille, miten hänen omasta kokemuksestaan oikeassa elämässä on harvinaista, että muutettava kokonaisuus koskisi vain yhtä komponenttia, vaan että yleensä vähintään puolet palveluista päivitetään joka tapauksessa. Hän oli kuitenkin samaa mieltä kirjallisuuden kanssa siitä, että tapauksissa, joissa komponentti ei ole kaikkein keskeisimmässä roolissa, ei välttämättä sen toiminnan lakkaaminen vaikuta koko järjestelmän toimintaan.

Artun kommentti itsenäisen käyttöönoton hankaluuksista ja harvinaisuudesta käytännössä erosi kirjallisuudesta ja muista haastatteluista merkittävästi, joten toisella haastattelukierroksella haluttiin ottaa selvää, ovatko muut haastateltavat samaa mieltä asiasta, tai heräsikö heille jotain uusia ajatuksia asiasta. Tapio kommentoi, että asia riippuu ohjelmiston ylläpitäjän tavasta päivittää ohjelmistoa. Hänen mielestään pieniä korjauksia ja yhden moduulin muutoksia olisi mukava ottaa pienellä vaivalla käyttöön, mutta isommissa päivityksissä pitää päivittää useampi moduuli, tosin tällöinkin osa toiminnoista voisi olla käytössä normaalisti. Kari kertoi, ettei ollut käytännössä törmännyt juurikaan itsenäiseen käyttöönottoon komponenteille. Hänen mukaansa järjestelmistä löytyy monesti riippuvuuksia Front Endistä Back Endiin ja tietokantaskeemoihin. Hän tarkensi, että monesti Front Endin päivitys vaatii jotain Back Endiin, joka sitten koskee montaa palvelua. Hän kertoi, miten käytännössä monesti on menty siihen, että koko järjestelmästä tehdään versio, johon kuuluu jokin tietty versio kaikista Front End- ja Back End -komponenteista, joka sitten tavallaan jäädytetään, kootaan ja käyttöönotetaan.

Johdattelun jälkeen myös Tapio oli samaa mieltä kirjallisuuden kanssa hyödyistä, joita rajapinnat ja itsenäinen käyttöönotto tuovat. Hän mainitsi, miten siitä huolimatta, että kaikki toimivat yhtenä kokonaisuutena, ei koko järjestelmä mene solmuun siksi että vähemmän keskeinen palvelu on kaatunut. Tapio, mainitsi miten tärkeää ja hyödyllistä tämä on, koska on normaalia, että yksittäinen osa tai palvelu välillä kaatuu, eikä ole realistista odottaa, että järjestelmän tai sovelluksen yksikään osa ei ikinä kaatuisi.

Kari lisäsi johdattelun jälkeen moduulien rajaukseen, miten muutoksien tekemisen rajaaminen tiettyjen rajoihin on etu. Hänen mukaansa esimerkiksi monoliittijärjestelmässä helposti lipsutaan rajoista ja oiotaan, vaikka siitä tekisi

modulaarisen. Mikropalveluissa ei voi oikoa samalla tavalla, mikä vähentää arkkitehtuurin rappeutumista.

Arttu ja Kari olivat molemmat samaa mieltä kirjallisuuden kanssa siitä, että virtuaalikoneiden ansiosta yksittäisiä osia voidaan skaalata tehokkaammin, ja että se on selkeä etu mikropalveluarkkitehtuurille. Kari lisäsi, että skaalautuvuus ei ole kuitenkaan aina yksinkertaista mikropalveluissakaan, koska tulee ottaa huomioon tietokannat ja muut palvelut, mitä ei muissa arkkitehtuureissa tarvitse huomioida samalla tavalla.

Karin kommentti tietokantojen tuomasta monimutkaisuudesta skaalautuvuuteen oli mielenkiintoinen ja poikkesi muista haastateltavista ja kirjallisuudesta, joten toisella haastattelukierroksella pyrittiin selvittämään, ovatko muut haastateltavat samaa mieltä asiasta, tai pitävätkö he aihetta tärkeänä asiana ottaa huomioon.

Arttu oli samaa mieltä asiasta, ja lisäsi, että tärkeys riippuu järjestelmän tulevaisuuden kasvunäkymästä. Tapion kokemuksesta tietokantojen kirjoitus ja mahdollinen klusterin synkronointi toki voivat aiheuttaa ongelmia. Hän kommentoi, että niitä voidaan yrittää kiertää, mutta asia ei ole suoraviivainen. Hän kuitenkin näkee, että tästä huolimatta mikropalvelut hoitavat skaalaamisen suurille käyttäjämäärille parhaiten.

Artun mielipide tiimien itsenäisen toiminnan hyödyistä myötäili kirjallisuudessa esiintyneitä ajatuksia. Vaikka haastattelussa tulikin esille, miten olisi tärkeää tietää kokonaisuudesta, oli hän sitä mieltä, että kehittämisen kannalta on myös hyödyllistä, että tiimien ei ole välttämätöntä tietää muista. Tapio kommentoi tiimien itsenäistä työskentelyä johdattelun jälkeen siten, että se on sitä hyödyllisempää, mitä suurempi järjestelmä tai sovellus on, mikä on myös mikropalveluarkkitehtuurin pääasiallinen käyttökohde. Kari otti itsenäisistä tiimeistä globaalin yhteistyön näkökulman ja mainitsi, miten hyödyllistä on, että tiimit eri puolilla maapalloa tekevät omia palveluitaan, eikä heidän tarvitse odotella toisia tiimejä eri aikavyöhykkeillä.

5.4.2 Haasteet

Mikropalveluiden haasteista Tapiolle tuli ensimmäisenä mieleen itse arkkitehtuurin rakentaminen. Hän kommentoi asiaa seuraavasti: ”Teknologia on suhteellisen nuori, eikä ole vakiintunutta tapaa, miten asioita tehdään. Eri yritykset tekevät asioita eri tavalla ja uusien tekijöiden on hankala päästä sisään, koska ei ole selkeitä raameja, miten asioita tehdään. Nämä ongelmat ovat olemassa ilman mikropalveluitakin, mutta mikropalvelut tuovat tähän aiheeseen yhden monimutkaisuuden tason lisää. Samaa ongelmaa pahentaa teknologiariippumattomuus, kun kehitetään useampaa palvelua, jotka on toteutettu eri teknologioilla. Teknologiat ja moduulien sisäiset arkkitehtuurit saattavat poiketa huomattavasti toisistaan, mikä vaikeuttaa kehittäjän työtä. Sovelluskehittäjien pitää tällaisissa tapauksissa oppia uusia tapoja kehittää järjestelmää tai sovellusta, kun he pomppivat eri palveluiden välillä. Kyseistä ongelmaa ei ilman teknologiariippumattomuutta olisi, ainakaan yhtä monimutkaisessa muodossa.”

Tapion esille tuoma ongelma teknologiariippumattomuuden tuomasta monimutkaisuudesta poikkesi kirjallisuudesta ja muista haastatteluista, joten pyrittiin selvittämään, ovatko muut haastateltavat samaa mieltä asiasta ja kuinka suurena ongelmana he mahdollisesti pitävät asiaa. Karin mielestä on järkevää käyttää tilanteeseen sopivaa teknologiaa, mutta tilanne, missä jokainen tekee mielensä ja uusimman muodin mukaan, ei ole toimiva. Hän pitää tärkeänä, että käytetään teknologian valinnan vapautta harkiten, esimerkiksi kirjastojen tai suorituskyvyn takia. Hän kuitenkin lisäsi, että eri kielissä ja teknologioissa voidaan käyttää samoja käytäntöjä ja saada ne mahdollisimman hyvin istumaan yleiseen arkkitehtuuriin. Arttu oli samaa mieltä, että useamman arkkitehtuurin kanssa työskentely vaatii ihmiseltä enemmän, ja että ongelman suuruus riippuu ihmisen kyvykkyydestä ja käytettyjen arkkitehtuurien eroista.

Kari näki haasteita mikropalveluiden tehokkuudessa. Mikropalvelut tarvitsevat tietoa muualta, mikä toteutetaan viestiketjuilla. Haasteeksi tämä aihe kasvaa, jos mikropalvelu kutsuu useampia muita palveluita, jotka taas kutsuvat vuorostaan useampaa palvelua, koska viestiketjut ovat hitaampia kuin prosessin sisäinen kutsuminen.

Arttu piti haastavana sitä, että kehitettäessä toimintoa järjestelmään pitäisi joka tapauksessa jokainen mikropalveluarkkitehtuurin osa tuntea. Hän toi esille oman kokemuksen, missä vastuut ja työt jaettiin mikropalveluiden mukaan, mutta yksittäisen henkilön toteuttaessa ominaisuutta, on käytännössä kaikki komponentit oltava hallussa ja navigointi palveluiden välillä voi olla haastavaa. Lisäksi Artun kokemuksesta harva GUI (graafinen käyttöliittymä) kehitystyökaluista tukee käytännössä mikropalveluiden tuontia samaan projektiin. Avattaessa useita alihakemistoja GUI hukkaa riippuvuustarkastelun, joka kertoo, mistä komponentista löytyy mikäkin funktiokutsu, eikä GUI osaa kertoa riippuvuudesta komponenttien välillä. Arttu painotti haastattelussa, miten tärkeää on, että joku tai jokin osapuoli hallitsisi järjestelmästä tai sovelluksesta kokonaiskuvan. Hänen mielestään olisi jopa suositeltavaa, että kaikilla osallistuvilla osapuolilla olisi tämä kokonaiskuva, koska muuten siiloudutaan helposti ilman tietoa ympärillä tapahtuvista asioista. Arttu näki, että työkalu, joka avaisi kaikki mikropalvelut yhteen näkymään ja osaisi kertoa niiden kutsusuhteet, olisi hyödyllinen. Esimerkiksi puutteellisen dokumentaation tai monimutkaisen järjestelmän takia voi olla työlästä selvittää, mitä palvelua pitää kutsua, jotta saadaan kyseinen toiminto aikaiseksi.

Haasteeksi Arttu näki myös komponenttien rajaamisen, koska se on häilyvä käsite, pahimmillaan jokin komponentti saattaa muodostua monoliitiksi. Lisäksi komponentteja päivittäessä pitää olla normaalia huolellisempi, että kaikki tarvittavat komponentit ovat ajan tasalla, koska kaikkea ei tarvitse päivittää, mutta samaan aikaan pitää olla huolellinen, että ei päivitetä liian vähän.

Johdattelun jälkeen Arttu oli samaa mieltä siitä, että osiin jakaminen on työläs ja manuaalinen prosessi. Myös Karin mielestä osiin jakaminen on vaikeaa ja haastavaa, sekä etukäteen on hankala nähdä, miten osiin jakaminen olisi järkevää tehdä.

Karin mielestä olisi tärkeää, että olisi osapuolia, jotka tietäisivät kokonaisuuden, ja mielellään jokainen tietäisi mahdollisimman paljon realistisuuden rajoissa. Normaalisti tämä muodostuu haasteeksi kansainvälisemmissä projekteissa ja järjestelmissä, joissa käyttäjämäärä ja järjestelmän koko ovat suurempia, mutta tämä haaste esiintyy Suomenkin mittapuulla, koska hänen mukaansa julkisella puolella jotkin projektit alkavat olla erittäin suuria kooltaan.

Arttu oli samaa mieltä kirjallisuuden kanssa etäkutsujen ongelmista, mutta sanoi, että hänellä ei itsellään ole kokemusta responsiivisuuden priorisoinnista luotettavuuden edelle. Karin mielestä priorisointi riippuu käsiteltävästä tiedosta, ja hänen mielestään joskus voi olla parempi odottaa oikeaa vastausta kuin antaa virhe. Aiheeseen liittyen Tapio kertoi törmänneensä siihen, että data päivitetään ensin sen perusteella, että oletetaan mitä se tulee olemaan, mutta jälkimmäinen päivitys voi erota oletuksesta esimerkiksi johtuen jostain virheestä.

Jokainen haastateltavista toi esille asynkronisen ohjelmoinnin ratkaisuna viiveisiin. Tapio kommentoi, miten asynkronista ohjelmointia käytetään paljon, ja se on standardi monessa aiheessa. Vaikka asynkroninen ohjelmointi auttaa ongelman kanssa, on siinä silti edelleen samat riskit ja tapahtuman tieto voi hukkua. Hänen mukaansa asynkronisesti tehtäessä asiat eivät toimi välttämättä siinä järjestyksessä kuin mitä haluttaisiin, mikä aiheuttaa logiikkavirheitä ja datan korruptoitumista. Kari lisäsi, että etäkutsujen viiveille ja sen tuomille haasteille ei voi mitään muuta kuin tehdä parhaansa. Parhaansa tekeminen tässä tapauksessa lähtee siitä, että jos yhteys ei toimi, niin ainakaan se ei jää jumiin, vaan tekee aikakatkaisun ja antaa järkevän virheilmoituksen, jonka jälkeen toivotaan, että kaikki ei kaadu. Kari kommentoi, miten tämä ratkaisu toimii vaihtelevalla menestyksellä, ja kyseinen haaste on vaikea.

Kari toi haastattelussa esille itsenäisen työskentelyn vaaroja: ”Tietoisuus voi hämärtyä, kun tekee vain omaa osaa, voi käydä siten, että yksi ammattilainen tekee omaa asiaansa ja muut eivät uskalla koskea siihen tikullakaan, koska se on niin pelottavan näköinen eivätkä muut ymmärrä siitä. Olisi hyvä kierrättää työtehtäviä, ettei vuodesta toiseen tehdä vain yhtä palasta. Täten tietoisuus kokonaisuudesta lisääntyy. Oman osan tekeminen voi mennä putkinääöksi ja voi alkaa muuttumaan henkilökohtaiseksi asiaksi, kun joku tulee ehdottamaan muutoksia.”

Karin esille tuoma ongelma itsenäisestä työskentelystä erosi kirjallisuudesta ja muista haastatteluista, joten toisella haastattelukierroksella pyrittiin selvittämään, ovatko muut haastateltavat samaa mieltä. Arttu oli samaa mieltä ja kommentoi, että osa ihmisistä on kykenevämpiä ymmärtämään isompia kokonaisuuksia. Tapio oli myös samaa mieltä asiasta, ja toi esille, että esimerkiksi tilaisuudet missä eri kokonaisuuksia tekevät tiimit jakaisivat omaa tietoisuuttaan, voisivat auttaa asiaan. Lisäksi hän painotti dokumentoinnin tärkeyttä, ja että ongelma ei liity yksinomaan mikropalveluihin, vaan muissakin arkkitehtuureissa törmätään samaan ongelmaan, kun tehdään eri osa-alueita. Hän näkee tämän takia ongelman enemmän kehitysmallin kuin arkkitehtuurin ongelmana.

Sekä Arttu että Kari toivat esille saman haasteen itsenäisessä työssä, eli miten itsenäinen työskentely johtaa arkkitehtuurin rappeutumiseen, mutta Arttu toi esille, miten esimerkiksi arkkitehtuuritiimin vastuuna on pitää huolta siitä,

että näin ei pääsisi tapahtumaan. Tapio kommentoi arkkitehtuurin rappeutumista siten, että kehittäjällä on mikropalveluarkkitehtuurissa enemmän vastuuta arkkitehtuurin eheydestä ja kehittäjän laiskuus voi johtaa arkkitehtuurin rakoiluun.

Kari toi haastattelussa esille arkkitehtuurin rappeutumisesta toisen näkökulman: ”Jokainen henkilö tekee omalla tyylillään, vaikka kyseessä olisi sama ohjelmointikieli. Kun ei ole valmista muottia ja kokonaisuudet ovat pieniä, voi tulla mieleen kokeilla ja harjoitella jotain uutta ja hienoa tapaa. Itse tekijälle voi olla selkeää mitä hän tekee, mutta muut eivät pääse heti sisään.”

Artun mielestä mikropalveluiden osien lukumäärän liittyy omat haasteensa. Hänen mukaansa kutsusuhteiden selvittämien voi olla hankalaa ja lukumäärä lisää kompleksisuutta. Osia voidaan joutua myös hakemaan manuaalisesti, ja eri versionhallinnoista etsiminen lisää aikaviivettä ja selvitystyötä. Karin kokemuksesta osien suurempi lukumäärä ei ole häirinnyt tai lisännyt merkittävästi työmäärää. Tapiolle tuli mieleen osiin jakamisen työmäärästä se, että mikropalveluissa joudutaan kopioimaan CI-putkia ja rakentamisprosesseja (build process), koska samat asiat tehdään samalla tavalla siitä huolimatta, että ne ovat omia kokonaisuuksia. Monoliittijärjestelmässä on vain yksi rakentamisprosessi, mutta mikropalveluissa pitää katsoa, että se toimii useammassa paikassa. Tämä lisää mikropalveluihin CI-putken työmäärää tekemällä siitä monimutkaisempaa ja hankalampaa, mutta mikropalvelut vähentävät työmäärää muualta. Eri vaatimukset eri mikropalveluille lisäävät myös työmäärää.

Ongelmien siirtyminen prosessin sisältä osien välisiin suhteisiin toi Artun mukaan haasteita. Jos rajapinta ei ole täydellisesti dokumentoitu, niin käyttäjä voi ymmärtää sen eri tavalla kuin palvelun tarjoaja. Karin mielestä aihe vaatii keskitetyn lokituksen, johon laitetaan korrelaatio-ID mukaan, mutta tämä on hankalaa, koska pitää katsoa usean palvelun lokeja ja korreloida ID:n ja aikaleimojen mukaan. Tämä vaatii monitorointiteknologiaa, joka tuo mukaan monimutkaisuutta arkkitehtuuriin.

Arttu on samaa mieltä siitä, että mikropalveluiden arkkitehtuurin luominen ja rajapintojen määrittely vie paljon aikaa, ja että se on sopiva ratkaisu tarpeeksi monimutkaisissa järjestelmissä, eikä sovi ratkaisuksi tyhjältä pöydältä ideoita kokeileviin tilanteisiin. Hän oli samaa mieltä kirjallisuuden kanssa siitä, milloin mikropalvelut ovat järkevä ratkaisu käytettäväksi arkkitehtuuriksi. Myös Kari oli samaa mieltä, että käytetyt resurssit maksavat itsensä takaisin vain tietyissä puitteissa: Skaalan pitää olla oikea, ja onnistuvuus riippuu integraatiosta ja siitä, kuinka korkea saatavuus palvelulla pitää olla. Onko esimerkiksi hyväksyttävää, että palvelu on tiettyinä aikoina poissa käytöstä vai pitääkö sen olla koko ajan tavoitettavissa?

Myös Tapio oli kirjallisuuden kanssa samaa mieltä siitä, että mikropalveluiden käyttäminen on tasapainottelua sen välillä, milloin monimutkaisuus on sen arvoista, että uhrataan tiettyjä asioita ja milloin mikropalvelut maksavat itsensä takaisin. Tapion mielestä on hyvä muistaa, että pienemmissä esimerkiksi yrityksen sisäisissä sovelluksissa mikropalvelut luovat kompleksisuutta ilman merkittäviä hyötyjä. Tärkeimpänä Tapio piti käyttäjämäärää, jonka pitää olla tarpeeksi

suuri. Hänen mukaansa mikropalveluita voidaan käyttää alusta lähtien, jos tiedetään varmaksi, että tullaan tarvitsemaan mikropalveluita, mutta vesien kokeilussa mikropalvelut eivät ole vähäisten startup-resurssien järkevä sijoitus.

Arttu oli samaa mieltä kirjallisuuden kanssa siitä, että mikropalvelut eivät ratkaise puutteita tiimin osaamisessa. Kari oli eri mieltä kirjallisuuden kanssa arkkitehtuurin ja tiimin osaamisen tärkeydestä, koska isoissa yrityksissä kaikki eivät voi olla samantasoisia tekijöitä. Tasoerot selittyvät myös sillä, että jotkut ovat nuorempia ja vähemmän kokeneita.

Tapio ei myöskään ollut kirjallisuuden kanssa samaa mieltä siitä, että tiimin taidot ovat tärkeämpiä kuin arkkitehtuuri. Tapion kommentoi asiaa seuraavasti: ”Jos koetaan että tarvitaan mikropalveluita, niin kouluttaisin tiimin siihen. Mikropalveluiden käyttäminen ei ole vaikea homma, ja kehittäjät pystyvät oppimaan ja omaksumaan asian. Jos kehittäjä ei ole aikaisemmin käyttänyt mikropalveluarkkitehtuuria, on siinä paljon asioita ja se voi näyttää alkuun pelottavalta, mutta se on vaivan arvoinen. Lisäksi työntekijät vaihtuvat IT-alalla, jotenka ei ole kyse siitä, etteikö työntekijä olisi tarpeeksi taitava, vaan kyse on siitä, että eri organisaatiot ja eri tiimit tekevät asioita eri tavoilla, jonka seurauksena yhdessä firmassa erittäin kokenut työntekijä ei välttämättä ole toisessa organisaatiossa heti ensimmäisestä päivästä lähtien samalla tasolla kuin edellisessä organisaatiossa. Lisäksi organisaatiot ottavat uusia työntekijöitä, joilla on vähemmän kokemusta alasta, joten ei ole realistista olettaa, että organisaatio olisi siinä tilanteessa, että pystyisi luottamaan tiimin osaamiseen niin vahvasti, että voisi olla välittämättä arkkitehtuurista sen varjolla.”

5.5 Haastatteluiden yhteenveto

Tulosten esittelyssä haastateltavien vastaukset on esitelty aiheiden mukaisessa järjestyksessä, joten tässä alaluvussa selvennetään yhteenvedon avulla jokaisen haastateltavan vastauksia. Haastateltavat olivat samaa mieltä suurimmasta osasta käsitellyistä aiheista, mutta eroja löytyy silti heidän esille tuomista yksityiskohdista, joista voidaan nähdä, miten eri näkökulmista he lähestyvät aiheita ja mitkä puolet ovat heille tärkeitä tai tulevat ensimmäisenä mieleen. Haastateltavien keskeisimmät kommentit on koottu Taulukkoon 1.

Tapiolla oli kolmesta haastateltavasta järjestelmäarkkitehdista vähiten työkokemusta, mutta hän on suorittanut korkeakoulututkinnon. Järjestelmäarkkitehtuuriaiheessa Tapio tiesi aiempia arkkitehtuureja ja teknologioita, ja toi eniten esille niiden puutteita. Mikropalveluaiheeseen liittyen Tapiolla oli haastateltavista positiivisin näkökulma, ja hän piti erityisesti teknologian valinnan vapautta hyödyllisenä ominaisuutena. Hänellä oli tietämystä mikropalveluiden keskeisistä teknologioista erityisesti palvelunlöytämisteknologioista. Mikropalveluiden hyödyistä keskeisimpänä hän piti ylläpidon ja jatkokehityksen helpottumista. Hänen mielestään arkkitehtuurin rakentaminen ja monimutkaisuuden kasvu aiheuttavat eniten haasteita.

Artulla oli enemmän työkokemusta kuin Tapiolla, ja myös hänellä oli korkeakoulututkinto. Järjestelmäarkkitehtuuriaiheeseen liittyen vanhat teknologiat ja arkkitehtuurit olivat Artulle tutumpia kuin muille haasteltaville. Mikropalveluihin Arttu suhtautui neutraalimmin kuin muut haastateltavat, ja piti tärkeänä hyvin toteutettuja rajapintoja. Mikropalveluiden keskeisten teknologioiden suhteen Arttu ei nähnyt esimerkiksi kielen valintaa yhtä tärkeänä kuin hyvin toteutettua pohjaa rajapintojen avulla. Hyödyistä Arttu mainitsi, miten kokonaisuus kestää muutoksia paremmin, ja haasteista hän toi esille, miten jokaisen yksittäisen henkilön vastuu kokonaisuuden ymmärtämisestä kasvaa, ja pitää välttää, ettei putkinäön kanssa siiloudu vain omaan asiaan.

Myös Karilla oli enemmän työkokemusta, mutta hänellä ei ole korkeakoulututkintoa, vaikka hän onkin käynyt korkeakouluopintoja. Järjestelmäarkkitehtuuriaiheessa Kari mainitsi eniten arkkitehtuurien kehityksen tekijöitä. Hänellä haastateltavista maltillisin suhtautuminen mikropalveluihin. Kari toi keskeisissä teknologioissa esille vaihtoehtoisia ratkaisuja konttiteknologioille ja pilvipalveluille. Hyödyistä Kari mainitsi puhtaamman kokonaisuuden ja korkean saavutettavuuden. Haasteena Kari näki, että monimutkaiset palveluiden suhteet voivat johtaa tehokkuuden heikkenemiseen.

TAULUKKO 1 Haastateltujen keskeisimmät kommentit aihealueista.

	Tapio	Arttu	Kari
Taustat	Korkeakoulututkinto, suhteessa vähiten työkokemusta	Korkeakoulututkinto	Korkeakoulukokemusta, mutta ei tutkintoa
Järjestelmäarkkitehtuuri	Toi esille puutteita vanhoissa teknologioissa ja arkkitehtuureissa.	Vanhemmat teknologiat olivat hänelle eniten tuttuja.	Mainitsi eniten kehityksen tekijöitä ilman johdattelua.
Mikropalvelut	Positiivinen näkökulma mikropalveluiden käyttämisestä.	Neutraali suhtautuminen.	Maltillisempi suhtautuminen.
Teknologiat	Tietämystä erityisesti palvelunlöytämisteknologioista.	Piti rajapintojen toteuttamista tärkeimpänä.	Näki konttiteknologioissa haasteita ja puutteita.
Hyödyt	Ylläpidon ja jatkokehityksen helpottuminen.	Arkkitehtuuri kestää paremmin muutoksia.	Selkeämpi kokonaisuus ja korkea saavutettavuus.
Haasteet	Arkkitehtuurin rakentamisen monimutkaisuus.	Osiin jakaminen voi johtaa kokonaiskuvan hukkaamiseen.	Tehokkuus haasteena, jos mikropalveluiden suhteet liian ristissä.

6 TULOSTEN TULKINTA JA POHDINTA

Tutkimuksen tavoitteena oli ottaa selvää, kohtaavatko kirjallisuuden ja yritysmaailman ammattilaisten näkemykset mikropalveluista sekä mikropalveluiden hyödyistä ja haasteista. Tämä tavoite saavutetaan vastaamalla kahteen tutkimuskysymykseen haastatteluissa kerätyn tiedon ja teoriaosion kirjallisuuskatsauksen avulla, ja nämä tutkimuskysymykset ovat seuraavat:

Tutkimuskysymys 1: Miten tieteellinen kirjallisuus ja alan ammattilaiset käsittävät termin ”mikropalvelu”, ja mitä eroja näiden kahden näkökulman välillä esiintyy?

Tutkimuskysymys 2: Mitä hyötyjä ja haasteita tieteellinen kirjallisuus ja alan ammattilaiset näkevät liittyen mikropalveluiden käyttämiseen sovellus- ja järjestelmäkehityksessä, ja mitä eroja näiden kahden näkökulman välillä esiintyy?

Ensimmäisenä käydään läpi asiat, mistä haastatellut olivat samaa mieltä kirjallisuuden kanssa, ja miten hyvä tietämys heillä oli mikropalveluista etukäteen. Toisena käydään läpi tapaukset, missä haastatellut poikkesivat kirjallisuudesta, joko siten, että he toivat esille asioita, joita ei kirjallisuudessa mainittu, tai he eivät olleet samaa mieltä kirjallisuuden kanssa. Taulukko 2 visualisoi keskeisimmät yksimielisyydet ja eriävät mielipiteet haastatteluiden ja kirjallisuuden välillä.

TAULUKKO 2 Keskeisimmät erot ja yhtäläisyydet kirjallisuuden ja haastatteluiden välillä.

	Samaa mieltä kirjallisuuden kanssa	Eri mieltä kirjallisuuden kanssa
Järjestelmäarkkitehtuuri	Samaa mieltä melkein jokaisessa asiassa liittyen aiheeseen. Suhteellisen hyvä ymmärrys aiemmista teknologioista, ja kehityksen tekijöistä.	SOA ja mikropalveluiden väliset erot eivät yhtä selkeitä. Käytetyt teknologiat ja riippuvuus vanhoista järjestelmistä eivät erota niitä toisistaan merkittävästi.
Mikropalvelut	Samaa mieltä käytännössä koko aiheesta. Heillä oli hyvä tietämys mikropalveluarkkitehtuurista ja sen keskeisistä piirteistä.	Ei suoria erimielisyyksiä, mutta huomautuksia siitä, miten tulisi ottaa huomioon työntekijöiden vaihtuvuus ja teknologian osaajien määrä tulevaisuudessa.
Teknologiat	Samaa mieltä, ja heillä oli suhteellisen hyvä tietämys keskeisistä teknologioista.	Ei suoria erimielisyyksiä, mutta ajatuksia vaihtoehdoista konttiteknologialle.
Hyödyt	Suurimmaksi osaksi samaa mieltä kirjallisuuden kanssa, ja hyvä tietämys mikropalveluiden tuomista hyödyistä.	Itsenäisen käyttöönoton hyödyntäminen harvinaista.
Haasteet	Tässä aiheessa eniten kirjallisuudesta poikkeavia mielipiteitä tai kirjallisuuteen lisääviä kommentteja. Pääasiassa silti samaa mieltä kirjallisuuden kanssa, ja he tiesivät kirjallisuudessa esille tuoduista haasteista liittyen mikropalveluihin.	Lisäyksiä ja tarkennuksia useisiin haasteisiin. Eri mieltä työntekijöiden osaamisen ja arkkitehtuurin tärkeysjärjestyksestä.

6.1 Tulosten yhtenäisyys kirjallisuuden kanssa

Haastatellut järjestelmäarkkitehdit tiesivät suhteellisen hyvin aiemmista arkkitehtuureista ja teknologioista, jotka ovat johtaneet mikropalveluiden kehitykseen. Jotkin yksittäiset arkkitehtuurit kuten SOC eli palvelukeskeinen tietotekniikka eivät olleet tuttuja heille, mutta he olivat kaikki käyttäneet työelämässä paljon SOA:ta eli palvelukeskeisestä arkkitehtuuria, joten SOC:n ideat ovat sitä kautta heillä tiedossa. He olivat samaa mieltä järjestelmäarkkitehtuuriin liittyvistä asioista ja osasivat kertoa hyvin ilman johdattelua keskeisiä syitä arkkitehtuurin evoluutioon. Vaikka tässä aiheessa heillä ei ollut syvää tietämystä kaikista asioista, en näe, että sillä on merkittävästi väliä, koska he tiesivät yleisellä tasolla asioista, ja olivat hyvin perillä järjestelmäarkkitehtuurin merkityksestä. Lisäksi läpikäydyistä aiheista vanhat järjestelmäarkkitehtuurit ovat suhteessa vähemmän tärkeä aihe.

Mikropalveluarkkitehtuurista he olivat myös kaikki hyvin tietoisia, ja heidän vastauksissaan näkyi se, että heillä kaikilla on kokemusta mikropalveluiden käyttämisestä työelämässä. He tiesivät hyvin mikropalveluiden idean ja keskeiset käsitteet, ja olivat samaa mieltä kirjallisuuden kanssa siitä, mitä mikropalveluarkkitehtuuri on. Läpikäydyistä aiheista mikropalveluarkkitehtuuri oli aihe, mistä haastatellut järjestelmäarkkitehdit tiesivät suhteessa parhaiten, ja olivat myös kirjallisuuden kanssa samaa mieltä käytännössä kaikesta, mitä heidän kanssaan keskusteltiin. Tämä on erittäin tärkeä asia, koska se tarkoittaa sitä, että sekä kirjallisuudella että kyseisen yrityksen järjestelmäarkkitehdeilla on sama pohjakäsitys siitä, mitä mikropalvelut ovat.

Mikropalveluihin liittyvien teknologioiden suhteen haastatelluilla oli hyvin tietoa keskeisistä teknologioista, tarkemmin sanottuna konttiteknologioista ja konttien orkestrointiteknologioista. Palvelunlöytämisteknologiat eivät olleet haastatelluille yhtä tuttuja. Heillä oli myös jonkin verran tietämystä kymmenestä teknologian aallosta, jotka eivät kuitenkaan olleet heille käytännössä kovin tuttuja. Tämä on ymmärrettävää, sillä kuten Kari haastattelussa mainitsi, ne ovat monimutkaisia eivätkä Suomen mittakaavalla tarpeeksi hyödyllisiä.

Mikropalveluiden hyödyistä haastatellut olivat myös hyvin perillä, ja olivat muutamaa poikkeusta lukuun ottamatta samaa mieltä kirjallisuudessa esille tuoduista ajatuksista niihin liittyen. Hyödyt osattiin tuoda esille hyvin käytännön esimerkkien kautta, mikä osoittaa heidän ymmärtävän hyvin, mitkä mikropalveluiden vahvuudet ovat. Haastatellut osasivat myös huomioida Suomen mittakaavan ja käyttäjämäärät hyvin ajatellessaan käyttötapauksia ja yksittäisiä mikropalveluiden tuomia hyötyjä, eivätkä keskittyneet vain parhaaseen mahdolliseen käyttötapaukseen, vaan toivat realistisia tilanteita esille.

Mikropalveluiden haasteista haastatellut olivat myös hyvin tietoisia, mutta kaikista käsitellyistä aiheista heidän mielipiteensä erosivat eniten koskien mikropalveluiden haittoja. Monesta haasteesta he olivat samaa mieltä kirjallisuuden kanssa, ja heiltä tuli hyvin lisäyksiä kirjallisuudessa mainittuihin haasteisiin,

mikä tuo tässäkin kohtaa esille sen, että he ovat ajatelleet syvemmin mikropalveluita ja heillä on kokemuksen lisäksi syvempää tietämystä aiheesta.

Haastatteluiden tulokset koskien yhteneviä mielipiteitä kirjallisuuden kanssa ovat merkittäviä siten, että yrityksellä on ainakin heidän järjestelmäarkkitehtiensa suhteen hyvät valmiudet käyttää mikropalveluita. Lisäksi näiden tulosten perusteella voidaan päätellä, että kyseisen yrityksen ammattilaisilla on pääasiassa kirjallisuuden kanssa yhtenevä ymmärrys mikropalveluista. Tulokset ovat myös luotettavia ja käytettäviä, kun otetaan huomioon konteksti: haastatellut ovat suomalaisen sovelluskehitysyrityksen järjestelmäarkkitehteja, mikä vaikuttaa heidän näkemyksiinsä käytännön käyttötarkoituksista. Käytännössä nämä tulokset antavat arvoa sekä tapauksen yritykselle että muille samanlaisessa tilanteessa oleville yrityksille, tarjoamalla materiaalia liittyen mikropalveluihin. Tulosten avulla yritykset pystyvät punnitsemaan, ovatko he valmiita käyttämään mikropalveluita, tai onko heidän tapauksessansa järkevää käyttää mikropalveluita. Tieteellisesti tulokset tuovat arvoa vahvistamalla, että kirjallisuuden käsitys ja kyseisen yrityksen kohdalla ammattilaisten käsitys mikropalveluista kohtaavat. Jatkotutkimusaiheena voisi olla mielenkiintoista selvittää vaikuttaako esimerkiksi yrityksen koko tai yrityksen kansainvälisyys tuloksiin. Olisiko esimerkiksi suurempi yritys tai kansainvälisempi yritys samaa mieltä kirjallisuuden kanssa?

6.2 Tulosten eroavaisuus kirjallisuudesta

Järjestelmäarkkitehtuurin kohdalla haastateltujen näkemykset erosivat kirjallisuudesta koskien SOA:n ja mikropalveluiden eroja. Haastatellut toivat esille, miten heidän kokemuksensa mukaan on harvinaista, että järjestelmää päästäisiin rakentamaan ilman yhteyksiä vanhempiin järjestelmiin. Lisäksi haastatteluissa tuli esille, miten SOA voisi myös hyödyntää mikropalveluissa käytettyjä kevyempiä teknologioita. Tästä seurasi mielenkiintoinen huomio siitä, miten höydyllisiä mikropalvelut voivat olla myös siten, että niitä hyödynnetään osana arkkitehtuuria.

Mikropalveluarkkitehtuurista haastatellut toivat esille kirjallisuudesta poiketen sen, että tuoteajattelussa tulee ottaa huomioon, että kokonaisuus ei ole liian hankala päästä sisälle, koska työntekijät vaihtuvat myös organisaation sisällä. Lisäksi haastatteluissa nostettiin teknologian valinnan vapauteen liittyen huomio, että tulisi valinnat tehdä miettien myös tulevaisuutta, eikä valita vanhaa teknologiaa, jonka osaajia on vähän ja jota ei enää opeteta uusille ihmisille. Lisäksi teknologian valinnan vapautta pitäisi käyttää maltillisesti, ja mahdollisesti tehdä valinta priorisoiden kokonaisuuden toimivuus yksittäisen osan parhaan toimivuuden edelle, koska ylläpito kestää todennäköisesti järjestelmän kehittämistä pidempään. Nämä huomiot koskien mikropalveluiden hyödyntämistä ovat hyviä huomioita perustuen käytännön kokemukseen, ja ovat tieteellisesti hyödyllisiä, jottei tieteellisessä teoriassa ajatella liian paljon pelkästään täydellistä mahdollista kehitystiimiä ja ympäristöä.

Mikropalveluteknologioissa haastatelluilla esiintyi jonkin verran tuntemattomampia käsitteitä: esimerkiksi kymmenen teknologian aallosta moni oli heille käytännössä vieraampi. Tämä on ymmärrettävää, koska moni näistä teknologioista tuo hyötyjen mukana monimutkaisuutta, mikä maksaa itsensä takaisin todennäköisesti vasta suuremmissa käyttäjämäärissä ja suuremmissa järjestelmissä kuin mitä kyseinen yritys on kehittänyt. Lisäksi palvelunlöytämisteknologiat eivät olleet kaikille haastatelluista yhtä tuttuja kuin konttiteknologiat ja konttiorkestrointiteknologiat. Asia voi selittyä sillä, että konttiorkestrointiteknologiat tarjoavat myös palvelunlöytämisteknologian, joten niitä ei välttämättä mietitä erikseen ja ne liitetään osaksi konttiorkestrointia. Lisäksi haastatteluissa mietittiin esimerkiksi vaihtoehtoja konttiteknologioille, esimerkiksi olisiko välimalleja, missä hyödynnettäisiin mikropalveluita, muttei pilvipalveluita tai konttiteknologiaa. Nämä tapaukset olisivat todennäköisesti erittäin harvinaisia, mutta ajatus on silti mielenkiintoinen.

Mikropalveluiden hyödyistä haastatellut olivat eri mieltä koskien itsenäisen käyttöönoton realistisuutta. Heidän kokemuksestaan on erittäin harvinaista, että muutos koskisi vain yksittäistä komponenttia. Haastatellut olivat kuitenkin samaa mieltä siitä, että on hyödyllistä, että on vaihtoehto tällaisiin tapauksiin, vaikka ne olisivatkin harvinaisia. Itsenäisen käyttöönoton hyödyntämisen realistisuus erikokoisissa yrityksissä, erikokoisissa järjestelmissä ja erikokoisissa käyttäjämäärissä voisi olla mielenkiintoinen jatkotutkimusaihe. Minkä kokoinen ja kuinka huolellisesti toteutettu arkkitehtuurin pitää olla, että itsenäisestä käyttöönotosta saataisiin mahdollisimman paljon hyötyä irti?

Haasteet liittyen mikropalveluihin herätti eniten poikkeamia kirjallisuudesta. Arkkitehtuurin rakentaminen ja osiin jakaminen herättivät huomioita, joita ei kirjallisuudessa tullut esille, koskien yritysten erilaisia käytänteitä järjestelmien rakentamista, mikä muuttuu vielä suuremmaksi ongelmaksi, kun mikropalveluiden osiin jakamiselle ei ole vakiintunutta tapaa. Lisäksi teknologiariippumattomuus näihin lisättynä luo suuret paineet sovelluskehittäjille oppia monta uutta asiaa kerralla. Sovelluskehittäjille lisää haasteita luo myös arkkitehtuurin kokonaisuuden hallinta, sillä kokonaisuuden ja osien suhteiden tiedostaminen korostuu, kun kokonaisuus on rajattu selkeämpiin osiin. Ratkaisuna tähän ongelmaan haastatteluissa tuli esille esimerkiksi työkalu, mikä avaisi näkymän, mistä näkisi kaikki mikropalvelut ja niiden kutsusuhteet.

Erimielisyyttä kirjallisuuden kanssa herätti myös väite siitä, että mikropalvelut eivät ratkaise puutteita tiimin osaamisessa. Haastatellut pitivät tätä epärealistisena ajatuksena, koska työntekijöiden osaaminen vaihtelee suuresti luonnollisestikin, lisäksi osa on vähemmän kokeneita ja työntekijät vaihtavat työpaikkoja ja projekteja, jolloin heidän täytyy opetella uuden työpaikan tavat tehdä asioita. Haastatellut kokivat, että ei ole realistista ajatella tilannetta, missä organisaatio voisi luottaa tiimin osaamiseen niin vahvasti, ettei arkkitehtuurista tarvitsi välittää.

Haastatteluiden tulokset koskien erimielisyyksiä nostivat esille huomiota mikropalvelujen realistisesta toteuttamisesta ja asioiden realistisista hyödyistä. Huomioon tulee ottaa silti, että kyseessä olevan yrityksen käyttäjämäärät ja

järjestelmät eivät ole yhtä suuret kuin suuremmilla ja kansainvälisemmillä yrityksillä, joten olisi mielenkiintoista tietää, jakavatko erikokoiset yritykset esille tulleita mielipiteitä. Tulokset luovat arvoa käytännössä sekä kyseiselle yritykselle että muille samanlaisessa tilanteessa oleville yrityksille tuomalla esille asioita, joita tulee punnita valitessa mikropalveluiden käyttämistä. Tulokset tuovat tieteellistä arvoa siten, että jatkotutkimukset aiheesta voivat ottaa huomioon esille tuotuja huomioita mikropalveluiden realistisemmasta toteuttamisesta.

7 YHTEENVETO

Tutkimuksen tuloksena saatiin vastaukset tutkimuskysymyksiin siitä, kohtaavatko kirjallisuuden ja käytännön ammattilaisten näkemykset mikropalveluista sekä mikropalveluiden hyödyistä ja haasteista. Vastaus näihin kahteen kysymykseen lyhyesti: näkemykset kohtasivat pääasiassa, mutta useassa kohdassa ammattilaisilla oli lisättävää ja täydentäviä huomioita perustuen heidän käytännön kokemukseensa, ja muutamasta asiasta he olivat jopa eri mieltä kirjallisuuden kanssa. Tutkimuskysymyksien lisäksi saatiin vastattua Eduixin kysymykseen siitä, kuinka hyvät valmiudet heillä olisi hyödyntää mikropalveluita, ja tutkimuksessa tuotiin esille tietoa, jota he voivat hyödyntää päättäessään, haluavatko he käyttää mikropalveluita tulevaisuudessa.

Tutkimus toteutettiin laadullisena tutkimuksena ja tutkimusmenetelmänä käytettiin puolistrukturoituja haastatteluita. Rajoittavina tekijöinä tutkimuksen toteuttamiselle oli yrityksen pieni koko ja haastateltavien järjestelmäarkkitehtien määrä, mutta määrä kompensoitiin toteuttamalla kaksi haastattelukierrosta, joista toisella esitettiin tarkentavia ja täydentäviä kysymyksiä. Vaikka yrityksen muiltakin osapuolilta olisi ollut mielenkiintoista kysyä mikropalveluista, ovat järjestelmäarkkitehdit tärkeimmät henkilöt koskien järjestelmäarkkitehtuuria ja he jakavat muille työntekijöille osaamistaan sen perusteella kuin näkevät tärkeäksi.

Tutkimuksen teoriaosio perustuu aiempaan kirjallisuuteen, ja kirjoittajan oma kontribuutio on haastatteluiden toteuttaminen ja tulosten analysointi sekä johtopäätösten muodostaminen. Tutkimuksen tuloksia on hankala verrata aiemmin saavutettuihin tuloksiin, koska vastaavaa tutkimusta ei ole tehty koskien mikropalveluita, mutta tieteellinen tutkimus yleisesti mikropalveluihin liittyen on ollut pääasiassa samoilla linjoilla kuin tutkimuksen tulokset. Kuten tässäkin tutkimuksessa, on mikropalveluita kohtaan esitetty kritiikkiä muissakin tieteellisissä tutkimuksissa vedoten mikropalveluiden toteuttamisen haasteisiin (Ghofrani & Lübke, 2018).

Haastatelluilla ammattilaisilla eli Eduixin järjestelmäarkkitehdeilla on erittäin hyvin tietoa järjestelmäarkkitehtuureista ja mikropalveluista. Heidän tietämyksensä ei ollut täydellinen koskien kaikkia vanhempia arkkitehtuureja, tai

kaikkia mikropalveluissa mahdollisesti käytettäviä teknologioita. Vanhempien arkkitehtuurien osaamisella ei välttämättä ole yrityksen tulevaisuudelle suurta merkitystä, mutta heidän voisi olla hyödyllistä tutustua mikropalveluihin liittyviin uusien teknologioihin, jos yrityksellä on mielenkiintoa käyttää mikropalveluita. Tietenkin teknologioiden tutustumisen jälkeen täytyy vielä tehdä päätös, ovatko ne tarpeeksi höydyllisiä, koska ne ovat omalla tavallaan monimutkaisia ja niiden opettelu vie aikaa.

Kaikista tärkeimpänä yrityksen järjestelmäarkkitehdit tiesivät erittäin kattavasti mikropalveluarkkitehtuurista itsestään sekä mikropalveluiden hyödyistä ja niihin liittyvistä haasteista, jotka ovat kaikista tärkeimmät aihealueet, kun tutkimuksen aiheena on yrityksen valmius hyödyntää mikropalveluita. Täten yrityksellä on erittäin hyvät valmiudet heidän osaamisensa avulla muodostaa päätös mikropalveluiden käyttämisestä, ja mahdollisesti hyödyntää mikropalveluita tulevaisuudessa. Käytännön arvoa tutkimuksen tulokset antavat sekä Eduixille sekä muille samassa tilanteessa oleville yrityksille, ja mahdollisesti tarjoaa lisää tietoa ja ajateltavaa koskien mikropalveluita.

Tutkimuksen tulokset tuovat tieteellistä arvoa vahvistamalla, että tieteellinen näkemys mikropalveluista on pääasiassa saman suuntainen kuin käytännön ammattilaiset näkemys, mutta tutkimuksessa heräsi huomioita erityisesti tieteellisissä tutkimuksissa esitetyistä ihanteellisista näkemyksistä kehitystyötä tekevistä tiimeistä ja ympäristöistä. Tieteellisen kirjallisuuden olisi ehkä tulevaisuudessa hyvä myös keskittyä realistisiin tilanteisiin, missä kaikki työntekijät eivät ole yhtä kokeneita, sekä huomioida se, että työntekijöiden vaihtuvuus on suhteellisen yleistä.

Merkittävimpiä kehityskohteita mikropalveluihin liittyen ovat standardien asettaminen mikropalveluarkkitehtuurin osiin jakamiselle ja rakentamiselle sekä kokonaisuuden hahmottamiselle. Nämä kaksi asiaa voisivat selventää mikropalveluiden kehittämistä ja olisivat suuremmista konsepteista tärkeimmät kehityksen kohteet sekä jatkotutkimusaiheet. Muita jatkotutkimusaiheita voisivat olla tämän vertailevan tutkimuksen toteuttaminen eri kokoisissa ja kansainvälisemmissä yrityksissä. Kuinka yleistä esimerkiksi itsenäisen käyttöönoton hyödyntäminen on? Riittääkö esimerkiksi, että kyseessä olisi suurempi suomalainen yritys vai pitäisikö sen olla kansainvälisempi yritys?

Tuloksien yleistettävyydessä on huomioitava, että tutkimus on toteutettu haastattelemalla yksittäisen yrityksen järjestelmäarkkitehteja, jotka toteuttavat järjestelmiä, joiden käyttäjät ovat suomalaisia, joten käyttäjämäärät ovat erittäin kaukana esimerkiksi Facebookin käyttäjistä.

LÄHTEET

Beck, K. (2007). Implementation patterns. Pearson Education.

Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. IEEE Access, 10, 20357-20374.

Breivold, H. P., Crnkovic, I., & Larsson, M. (2012). A systematic review of software architecture evolution research. Information and Software Technology, 54(1), 16-40.

Brooks Jr, F. P. (1995). The mythical man-month: essays on software engineering. Pearson Education.

Bushong, V., Abdelfattah, A. S., Maruf, A. A., Das, D., Lehman, A., Jaroszewski, E., ... & Bures, M. (2021). On microservice analysis and architecture evolution: A systematic mapping study. Applied Sciences, 11(17), 7856.

Capretz, L. F. (2003). A brief history of the object-oriented approach. ACM SIGSOFT Software Engineering Notes, 28(2), 6.

Cockcroft, A. "The Evolution of Microservices," presentation at 2016

Crnkovic, I. (2001). Component-based software engineering—new challenges in software development. Software focus, 2(4), 127-133.

Del Rosso, C. (2006). Continuous evolution through software architecture evaluation: a case study. Journal of Software Maintenance and Evolution: Research and Practice, 18(5), 351-383.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering, 195-216.

Eduix Oy n.d. Perustietoa Eduixista. Verkkosivu. Viitattu 11.07.2023. https://eduix.fi/

Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

Fowler, M. & Lewis, J., Microservices – a Definition of this new Architectural Term, March 25, 2014. http://martinfowler.com/articles/microservices.html

Fowler, M., Microservice Trade-Offs, July 01, 2015. https://martinfowler.com/articles/microservice-trade-offs.html

Ghofrani, J., & Lübke, D. (2018). Challenges of Microservices Architecture: A Survey on the State of the Practice. ZEUS, 2018, 1-8.

Gray, J. "A Conversation with Werner Vogels," ACM Queue, vol. 4, no. 4, 2006, pp. 14–22.

Heineman, G. T., & Councill, W. T. (2001). Component-based software engineering. Putting the pieces together, addison-westley, 5, 1.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), 24-35.

Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture. In Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17 (pp. 32-47). Springer International Publishing.

Kang, H., Le, M., & Tao, S. (2016, April). Container and microservice driven design for cloud infrastructure devops. In 2016 IEEE International Conference on Cloud Engineering (IC2E) (pp. 202-211). IEEE.

Kecskemeti, G., Marosi, A. C., & Kertesz, A. (2016, July). The ENTICE approach to decompose monolithic services into microservices. In 2016 International Conference on High Performance Computing & Simulation (HPCS) (pp. 591-596). IEEE.

Khine, P. P., & Wang, Z. (2019). A review of polyglot persistence in the big data world. Information, 10(4), 141.

Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., ... & Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 157, 110380.

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., & Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. OASIS standard, 12(S18), 1-31.

Mens, T., Demeyer, S., Barais, O., Le Meur, A. F., Duchien, L., & Lawall, J. (2008). Software architecture evolution. Software Evolution, 233-262.

Newcomer, K. E., Hatry, H. P., & Wholey, J. S. (Eds.). (2015). Handbook of practical program evaluation. San Francisco, CA: Jossey-Bass & Pfeiffer Imprints, Wiley.

Newman, S. (2019). Monolith to microservices: evolutionary patterns to transform your monolith. O'Reilly Media.

Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. CLOSER (1), 137-146.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053-1058.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. ACM SIGSOFT Software engineering notes, 17(4), 40-52.

Rentsch, T. (1982). Object oriented programming. ACM Sigplan Notices, 17(9), 51-57.

Richardson, C.: Microservices—Pattern: Monolithic Architecture, March 2017. http://microservices.io/patterns/monolithic.html

Roberts, M. "Serverless Architec- tures," 4 Aug. 2016; martinfowler.com/articles/serverless.html.

Rowley, J. (2012). Conducting research interviews. Management research review, 35(3/4), 260-271.

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2016, December). The evolution of distributed systems towards microservices architecture. In 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST) (pp. 318-325). IEEE.

Sampaio, A. R., Kadiyala, H., Hu, B., Steinbacher, J., Erwin, T., Rosa, N., ... & Rubin, J. (2017, September). Supporting microservice evolution. In 2017 IEEE international conference on software maintenance and evolution (ICSME) (pp. 539-543). IEEE.

Szyperski, C., Gruntz, D., & Murer, S. (2002). Component software: beyond object-oriented programming. Pearson Education.

Valipour, M. H., Amirzafari, B., Maleki, K. N., & Daneshpour, N. (2009, August). A brief survey of software architecture concepts and service oriented architecture. In 2009 2nd IEEE International Conference on Computer Science and Information Technology (pp. 34-38). IEEE.

Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., ... & Lang, M. (2016, May). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 179-182). IEEE.

Woods, E. (2016). Software architecture in a changing world. IEEE Software, 33(6), 94-97.