Valtteri Harmainen

TIETEELLISEN LASKENTASOVELLUKSEN MODERNISOINTI PILVIPALVELUKSI: TAPAUSTUTKIMUS

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2014

TIIVISTELMÄ

Harmainen, Valtteri
Tieteellisen laskentasovelluksen modernisointi pilvipalveluksi: tapaustutkimus
Jyväskylä: Jyväskylän yliopisto, 2014, 85 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Leppänen, Mauri

Perinteisesti raskaaseen laskentaan vaadittavien IT-resurssien hankinta on edellyttänyt merkittäviä investointeja laitteistoihin ja ohjelmistoihin sekä niiden ylläpitoon. Pilvipalvelut vaikuttavat lupaavalta ratkaisulta IT-resurssien hankintaan. Erityisesti tutkijat hyötyisivät raskaaseen laskentaan vaadittavien IT-resurssien hankkimisesta palveluna, koska tällöin voitaisiin keskittyä vaadittavan laitteiston hankkimisen ja ylläpidon sijaan ydinongelman tutkimiseen. Miten olemassa olevia tieteellisiä laskentasovelluksia voidaan helpoiten muokata, jotta pilvipalveluiden hyödyntäminen olisi mahdollista?

Tämän tutkimuksen pyrkimyksenä on tunnistaa menetelmiä ja käytänteitä, jotka tukevat tieteellisten laskentasovellusten modernisointia pilvipalveluiksi yksinkertaisesti ja vaivattomasti. Tutkimuksella haetaan vastauksia kysymyksiin, mitä motiiveita on olemassa tieteellisen laskentasovelluksen modernisointiin pilvipalveluksi, mitkä modernisoinnin menetelmät ja käytänteet tukevat tutkijoiden toiveita ja miten tieteellisen laskentasovelluksen modernisointi voidaan toteuttaa käytännössä.

Tutkimuksessa tehdään aluksi kirjallisuuskatsaus tieteellisin laskentasovelluksiin, pilvipalveluihin sekä perinnesovellusten modernisointiin pilvipalveluiksi. Kirjallisuuskatsauksen pohjalta laaditaan tapaustutkimus, jossa eräs tieteellinen laskentasovellus modernisoidaan pilvipalveluksi. Tapaustutkimuksen ja kirjallisuuskatsauksen antaman laajemman kuvan perusteella pyritään päättelemään ja arvioimaan, millaiset modernisointimenetelmät ja -käytänteet olisivat hyödyllisiä tieteentekijöille, jotka haluavat hyötyä pilvipalveluiden tarjoamista mahdollisuuksista omien laskentasovellustensa tapauksessa.

Tutkimuksen tuloksena syntyy havaintoja pilvimodernisoinnin menetelmistä ja käytänteistä, jotka soveltuvat tieteellisen laskentasovelluksen tapaukseen. Tuloksia voidaan hyödyntää vastaavissa modernisointiprosesseissa ja pilvimodernisointimenetelmien tutkimuksessa.

Asiasanat: tieteellinen laskenta, modernisointi, pilvipalvelut, Amazon Web Services, tapaustutkimus

ABSTRACT

Harmainen, Valtteri
Modernization of a Scientific Application to a Cloud Service: a Case Study
Jyväskylä: University of Jyväskylä, 2014, 85 p.
Information Systems, Master's Thesis
Supervisor: Leppänen, Mauri

Huge computational power required by scientific applications has traditionally postulated acquiring expensive hardware, software and maintenance resources. Cloud computing seems to offer a promising solution to this problem. Researchers would benefit from acquiring the needed IT resources as a service instead of spending their limited budget on owning and maintaining those resources. How could existing scientific applications be easily modified to exploit the benefits of cloud computing?

The aim of this study is to identify methods and practices that support lightweight modernization of scientific applications into a cloud service. The study aims to find answers to the following questions: What are the motives of cloud modernization in the case of a scientific application? Which methods and practices support researchers' expectations and objectives? How can the modernization be done in practice?

The study starts with a literature review on scientific computing, cloud computing and modernization of legacy applications. Based on the review, a case study is conducted where a scientific application is modernized into a cloud service. Finally, the results combined from the literature review and the case study are used to derive conclusions on the support the observed methods and practices provide for the modernization process from the viewpoint of the researchers wanting to benefit from the advantages of cloud computing with their scientific applications.

The study provides a set of findings on the methods and practices that support the cloud modernization of scientific applications. The results can be used in similar modernization processes and in future research on cloud modernization.

Keywords: scientific computing, modernization, cloud computing, Amazon Web Services, case study

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Tässä luvussa johdatetaan aluksi lukija tämän tutkielman aihepiiriin ja esitellään siihen liittyviä aikaisemmin tehtyjä tutkimuksia. Toiseksi määritellään tämän tutkimuksen tutkimusongelma ja -menetelmät. Lopuksi esitellään tutkielman rakenne.

1.1 Aihepiirin esittely

Perinteisesti IT-resurssien hankinta on tapahtunut hankkimalla laitteistoa ja ohjelmistoja omaan tai organisaation omistukseen. Näiden resurssien käyttö on yleensä tarkoittanut merkittäviä investointeja laitteistoihin ja ohjelmistoihin sekä niiden ylläpitoon. Suurta laskenta- ja tallennuskapasiteettia vaativien sovellusten ajaminen on edellyttänyt järeän laitteiston omistamista, mikä ei yleensä ole pienempien toimijoiden saavutettavissa. Suuressa organisaatiossa kalliiden IT-resurssien käyttöönotto voi olla joissain tapauksissa hyvinkin byrokraattinen prosessi. Erityisesti tutkijat hyötyisivät raskaaseen laskentaan vaadittavien IT-resurssien hankkimisesta palveluna, jolloin usein niukat resurssit voitaisiin keskittää vaadittavan laitteiston hankkimisen ja ylläpidon sijaan ydinongelman tutkimiseen. Uusia tietotekniikkaresurssien käyttöönottomalleja onkin syntynyt viime vuosina, ja ne ovat monesti houkuttelevia joustavuutensa ja kustannustehokkuutensa vuoksi. Tutkijoilla on usein käytössään tieteellisiä laskentasovelluksia, joita ajetaan paikallisilla palvelimilla. Herää kysymys, kuinka tutkijat voisivat hyötyä uusista tietotekniikkaresurssien käyttöönottomalleista. Miten olemassa olevia tieteellisiä laskentasovelluksia tulisi muokata, jotta uusien mallien hyödyntäminen olisi mahdollista?

Pilvilaskenta (cloud computing) tarkoittaa mallia, jossa palveluntarjoaja tarjoaa käyttäjän tarpeiden mukaan verkon kautta saavutettavia ja konfiguroitavissa olevia resursseja, joiden käyttöönotto vaatii mahdollisimman vähän toimia käyttäjältä tai palveluntarjoajalta (Mell & Grance, 2009). Pilvipalvelut tarjoavat käytännössä lähes rajattomasti skaalautuvia resursseja raskaan laskennan suorittamiseen sekä laskennan tuloksena syntyvien datamassojen tallentamiseen. Pilvipalveluiden tarjoamat mahdollisuudet ja edulliset kustannusmallit houkuttelevatkin siirtämään raskasta laskentaa ja suurta tallennuskapasiteettia vaativat prosessit suoritettaviksi ja varastoitaviksi pilveen (Huang, Yu & Yu, 2013;

Srirama, Ivanistsev, Jakovits & Willmore, 2013). Käyttäjän ei tarvitse investoida ja ylläpitää omia laskenta- ja tallennusresursseja, mikäli niihin vaadittava infrastruktuuri voidaan ulkoistaa (Brian ym., 2012).

Amazon Web Services (AWS) on Amazonin pilvipalvelualusta, joka tarjoaa IT-resursseja käytön mukaisella hinnoittelulla (Krause, 2013). AWS koostuu kattavasta joukosta pilvilaskentaan ja pilvi-infrastruktuuriin liittyviä palveluita, kuten automaattisesti skaalautuvat EC2 virtuaalitietokoneet, S3 tietovarasto sekä SQS viestijonotuspalvelu ("AWS | What is AWS - Cloud Computing with Amazon Web Services", 2014).

Tieteellisillä laskentasovelluksilla tarkoitetaan sovelluksia, jotka ratkaisevat tieteellisiä, yhteiskunnallisia tai teknisiä ongelmia matemaattisten mallien ja numeeristen ratkaisutekniikoiden avulla (Vecchiola, Pandey & Buyya, 2009). Viime aikoina on havahduttu huomaamaan pilvipalveluiden edut myös tieteellisille sovelluksille. Tällaiset sovellukset tarvitsevat tyypillisesti erittäin paljon laskentatehoa, jotta laskenta pystytään suorittamaan kohtuullisessa ajassa. Esimerkkejä tällaisista sovelluksista ovat tähtitieteessä käytettävät kosmologisia malleja laskevat sovellukset sekä biologiassa käytetyt geenisekvensoijat. Tieteellisiä sovelluksia ajetaan tyypillisesti paikallisilla palvelimilla tai supertietokoneilla. Valitettavasti tutkijoilla harvoin on tietotaitoa, joka mahdollistaisi pilvipalveluiden tarjoamien mahdollisuuksien hyödyntämisen (Srirama ym., 2013).

Perinnesovellukselle (legacy application) ei ole olemassa täydellistä määritelmää, mutta eri määritelmistä on löydettävissä yhteneviä piirteitä. Perinnesovellus on usein vanhentunut, mutta edelleen käytössä oleva sovellus, jonka ylläpito ja kehittäminen on muodostunut vaikeaksi, koska sovelluksen tekninen viitekehys on vanhentunut (Fanqi & Yunqi, 2013). Modernisoinnin yhteydessä perinnesovelluksella tarkoitetaan sovellusta, joka on modernisoinnin kohteena (Saarelainen ym., 2006).

Modernisoinnilla (modernization) tarkoitetaan toimintaa, jolla muutetaan järjestelmän ylläpitoa laajemmassa mittakaavassa säilyttäen kuitenkin siitä merkittävä osa (Comella-Dorda, Wallnau, Seacord & Robert, 2000, s. 12). Modernisointi on sovelluksen uudistamisen menetelmä, jossa järjestelmään tehdään muutoksia arkkitehtuurin tasolla, esimerkiksi muuttamalla sovelluksen rakennetta tai uudistamalla lähdekoodia. Modernisaatiossa muutoksia tehdään laajemmin kuin järjestelmän ylläpidossa. (Kankaanpää, 2011.)

Modernisointi pilvipalveluksi tarkoittaa sitä, että modernisointiprosessin seurauksena käyttäjä voi käyttää sovellusta suoraan pilvessä internetselaimen avulla. Käytännössä tämä tekee sovelluksesta alustariippumattoman ja loppukäyttäjälle erittäin saavutettavan. Käyttäjä tarvitsee vain internetyhteyden käyttääkseen sovellusta pilvessä. (Mell & Grance, 2009)Perinnesovelluksen modernisointi pilvipalveluksi pitää sisällään monenlaisia haasteita (Bergmayr ym., 2013). Modernisointimenetelmiä on useita, ja oikeanlaisen menetelmän valitseminen on oleellista modernisoinnin onnistumisen kannalta (Comella-Dorda ym., 2000).

Pilvilaskentaa ja pilvipalveluita on tutkittu hyvin monipuolisesti aina vuodesta 2007 lähtien, jolloin pilvi-termiä alettiin käyttää (Zhang, Zhang, Chen & Huo, 2010). Tieteellisen (tietokoneella tapahtuvan) laskennan juuret ulottuvat nykyaikaisen tietokoneen keksimiseen 1940-luvulle (Nash, 1990, ss. 11–14), ja sitä on tutkittu siitä lähtien runsaasti. Tämän tutkimuksen kannalta oleellisia ovat ne tutkimukset, jotka käsittelevät niitä tieteellisen laskennan asettamia haasteita, joihin pilvipalvelut pystyvät vastaamaan.

Perinnesovellusten modernisointia on myös tutkittu paljon, ja useita erityyppisiä modernisointimenetelmiä ja -viitekehyksiä on esitetty (Comella-Dorda ym., 2000; Menychtas ym., 2013; Saarelainen ym., 2006). Perinnesovelluksen modernisointia pilvipalveluiksi ovat käsitelleet muun muassa Cretella ja Martino (2014) sekä Bergmayr ym. (2013). Sovellusten migraatiota pilveen on tutkittu viime aikoina runsaasti, mutta suurin osa tutkimuksista on keskittynyt kaupallisiin sovelluksiin ja prosessit ovat kokoluokaltaan sekä kompleksisuudeltaan pääosin liian monimutkaisia sovellettavaksi suoraan tämän tutkimuksen esittämään ongelmaan.

Tsaftaris (2014) ja Srirama ym. (2013) käsittelevät artikkeleissaan pilvimigraatiota tieteilijöiden näkökulmasta, joka on kohtalaisen lähellä tämänkin tutkimuksen tavoitetta. Näissä artikkeleissa käsitellään tieteellisen sovelluksen pilveen viemisen hyötyjä sekä esitellään käytännön sovelluksia siitä, miten migraatio voidaan suorittaa. Kummassakin artikkelissa korostuu näkemys siitä, että tutkijoiden ensisijaisen kiinnostuksen kohteena ei ole esimerkiksi laskentaklusterin konfigurointi, vaan pyrkimys raskasta tieteellistä laskentaa vaativien tutkimusten riippumattomaan, vaivattomaan ja kustannustehokkaaseen suorittamiseen pilvipalveluita hyödyntäen. Tsaftaris (2014) esittelee olemassa olevien työkalujen hyödyntämismahdollisuuksia empiirisin esimerkein. Srirama ym. (2013) päätyivät luomaan työkalun, jonka pyrkimyksenä on helpottaa tutkijoita ajamaan laskentasovelluksiaan pilvessä.

Yksi tämän tutkimuksen haasteista on tunnistaa suuresta joukosta olemassa olevaa aineistosta sellaiset menetelmät, jotka tukevat parhaiten niiden tutkijoiden tavoitteita, jotka haluavat vaivattoman, skaalautuvan ja helposti hallittavan ajoympäristön laskentasovelluksilleen.

1.2 Tutkimusongelma ja -menetelmät

Tutkimuksen tutkimusongelma on: *Miten sovitetaan yhteen tutkijoiden toiveet ja ohjelmistojen modernisointi pilvipalveluksi tieteellisen laskentasovelluksen tapauksessa?* Tutkimusongelma voidaan jäsentää kolmeksi tutkimuskysymykseksi:

- Mitkä ovat motiivit tieteellisen laskentasovelluksen modernisointiin pilvipalveluksi?
- Mitkä modernisoinnin menetelmät ja käytänteet tukevat tutkijoiden toiveita?
- Miten tieteellisen laskentasovelluksen modernisointi voidaan toteuttaa käytännössä?

Tutkimus toteutetaan tapaustutkimuksena, joka koostuu käsitteellisestä ja empiirisestä osuudesta. Käsitteellisessä osuudessa jäsennetään tieteellisten laskentasovellusten, pilvipalveluiden ja sovellusten modernisoinnin käsitteitä, yleisiä piirteitä ja periaatteita kirjallisuuskatsauksen avulla. Kirjallisuuskatsaus painotetaan ja rajataan siten, että tarkasteltavat asiat ovat relevantteja tutkimusongelman kannalta. Pilvipalveluita koskevasta kirjallisuudesta otetaan erityiseen tarkasteluun ne piirteet, jotka vastaavat tieteellisten laskentasovellusten asettamiin haasteisiin. Modernisaation kannalta tarkastellaan niitä kirjallisuudessa esitettyjä menetelmiä, jotka tukevat modernisointia pilveen ja vastaavat tutkijoiden toiveita.

Empiirinen osuus toteutetaan konstruktiivisena tapaustutkimuksena. Tapaustutkimuksen tarkoituksena on ymmärtää tutkittavaa ilmiötä syvällisemmin ja parantaa sekä kehittää tiettyjä piirteitä tutkittavasta ilmiöstä (Runeson & Höst, 2009). Tapaustutkimuksen kohteena on erään tieteellisen laskentasovelluksen modernisointi pilvipalveluksi. Tapaustutkimuksen aluksi haastatellaan sovelluksen tilaajaa ja käyttäjää edustavaa tahoa tarkoituksena selvittää modernisaation lähtökohtia ja motiiveja sekä erityisesti tutkijoiden toiveita käytettävistä modernisointimenetelmistä ja modernisoidun sovelluksen piirteistä. Haastattelut ovat lyhyitä, ja niistä saatuja tuloksia käytetään perusteluna modernisoinnin toteutuksessa tehtäville valinnoille. Konstruktiivisessa osuudessa valitaan, sovitetaan ja sovelletaan valittuja modernisointimenetelmiä tieteellisen laskentasovelluksen muokkaamiseksi pilvipalveluksi. Pilvipalvelusovelluksen toteutuksen jälkeen kysytään käyttäjien mielipiteitä sen laadusta. Modernisointiprosessista saatuihin kokemuksiin nojaten muodostetaan lista suositeltavista käytänteistä.

Tutkimuksen tuloksia voidaan hyödyntää vastaavissa tieteellisten laskentasovellusten modernisointiprosesseissa. Muodostettavaa listaa suositeltavista käytänteitä voidaan mahdollisesti soveltaa myös muihin samantyyppisiin perinnesovellusten modernisoinnin tapauksiin.

1.3 Tutkielman rakenne

Tutkielma koostuu kahdeksasta luvusta. Luvut 2-4 muodostavat kirjallisuuskatsauksen, ja luvut 5-7 käsittelevät kirjallisuuskatsauksen pohjalta suoritetun tapaustutkimuksen toteutusta, tuloksia ja pohdintaa.

Luvussa 2 kuvaillaan tieteellisten laskentasovellusten yleisiä piirteitä. Tarkoituksena on luoda pohja, jonka perusteella voidaan osoittaa pilvipalveluiden tarjoavan mahdollisia ratkaisuja raskaan tieteellisen laskennan asettamille haasteille sekä tutkijoiden toiveille. Luvussa 3 tarkastellaan ensin pilvipalveluita yleisesti ja sen jälkeen erityisesti AWS (Amazon Web Services) -pilvipalvelualustaa. Tarkoituksena on esitellä AWS-pilvipalveluiden tarjoamat mahdollisuudet raskaan laskennan suorittamiseen sekä sovelluksen tarjoamiseen palveluna (Software as a Service). Luvussa 4 kuvataan perinneohjelmistojen (legacy software) modernisointia, sen päätöstilanteita, menetelmiä ja käytänteitä. Tarkoituksena on myös selvittää, mitkä menetelmät ja käytänteet ovat sovellettavissa kevyisiin ja kustannustehokkaisiin (esim. ketterän kehityksen) menetelmiin, kun sovellusta modernisoidaan pilvipalveluksi. Pyrkimyksenä on edesauttaa menetelmien ja käytänteiden valintaa tapaustutkimusta varten.

Luvussa 5 esitellään ensin lyhyesti tapaustutkimuksen kohteena oleva tieteellinen laskentasovellus, sen käyttöympäristö ja modernisointitilanne. Luku sisältää myös kuvauksen tutkimusmenetelmästä ja tiedon keräämisestä. Luvussa 6 esitellään tapaustutkimuksen tulokset tutkimusprosessin mukaisesti. Luvussa osoitetaan myös, mitä modernisointimenetelmää ja käytänteitä missäkin vaiheessa on käytetty. Lopuksi esitetään arvioita modernisointiprosessista ja sen tuloksesta.

Luvussa 7 esitetään tapaustutkimuksen tulokset, verrataan niitä aiempiin tutkimuksiin ja esitetään johtopäätöksiä. Toiseksi luvussa tarkastellaan tutkimuksen reliabiliteettia ja validiteettia sekä kerrotaan tulosten hyödyntämismahdollisuuksista.

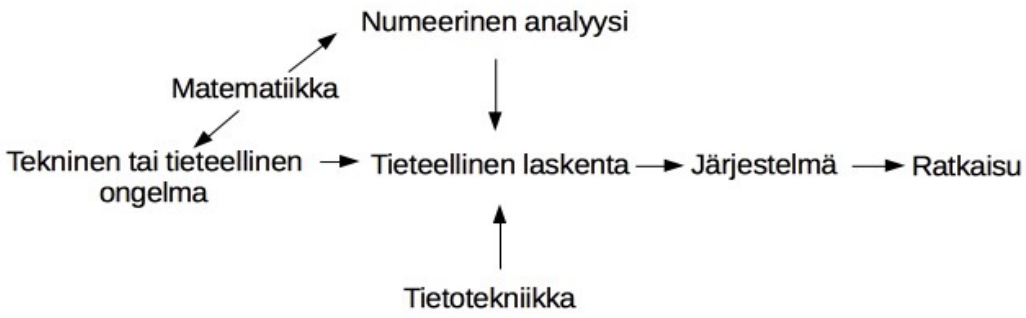
Luvussa 8 esitetään tiivistetysti tutkielman tulokset suhteutettuna tutkimusongelmaan ja -kysymyksiin sekä pyritään antamaan kokonaiskuva tehdystä tutkimuksesta ja sen merkityksestä. Lopuksi esitetään jatkotutkimusaiheita.

2 TIETEELLISET LASKENTASOVELLUKSET

Tässä luvussa määritellään tieteellisen laskentasovelluksen käsite ja esitetään kirjallisuuden avulla katsaus näiden sovellusten yleisistä piirteistä sekä esitellään muutamia yleisimpiä tieteellisen laskennan sovellustyyppejä.

2.1 Määritelmä ja taustaa

Tieteellisellä laskentasovelluksella tarkoitetaan tämän tutkielman yhteydessä tietokoneohjelmaa, joka suorittaa tieteellistä laskentaa yhdellä tai useammalla tietokoneella. *Tieteellisellä laskennalla* (scientific computing) tarkoitetaan tietokoneella suoritettavaa laskentaa, jossa pyritään matemaattisten mallien avulla ratkaisemaan tieteellisiä ongelmia (Golub, 1992, s. 1; Vecchiola ym., 2009). Tieteellisessä laskennassa pyritään tietotekniikan ja matematiikan avulla löytämään paras mahdollinen ratkaisu johonkin tieteelliseen ongelmaan. Kuvio 1 havainnollistaa tietotekniikan, matematiikan ja tieteellisen laskennan välisiä suhteita. Tieteistä ja tekniikasta kumpuavien ongelmien ratkaisemiseksi kehitetään matemaattisia malleja, joista laaditaan tieteellisiä laskentasovelluksia numeerisen analyysin ja tietotekniikan keinoin. Tuloksena syntyvä järjestelmä tuottaa mahdollisen ratkaisun tieteelliselle ongelmalle.



KUVIO 1 Tieteellinen laskenta suhteessa tietotekniikkaan ja matematiikkaan (Golub, 1992, s. 3)

Tieteellistä laskentaa on suoritettu tietokoneilla niin kauan kuin nykyaikaisia tietokoneita on ollut olemassa. Yleisesti on katsottu, että ensimmäiset nykyaikaiset tietokoneet kehitettiin toisen maailmansodan aikana laskemaan ammusten ballistisia ratoja, ja sodan jälkeen esim. ENIAC-tietokone valjastettiin satunnaislukujen tutkimiseen, tuulitunnelien suunnitteluun ja sään ennustamiseen (Ruttimann, 2006).

Olennaisin syy tietokoneiden käyttämiseen laskentatehtävissä on niiden suunnaton tehokkuus verrattuna ihmiseen. Tietokoneiden ylivoimaista laskentatehoa suhteutettuna ihmisen suorittamaan mekaaniseen laskentaan kuvastaa hyvin esimerkki piin likiarvon määrittämisen historiasta. 1500-luvulla saksalainen matemaatikko van Ceulen vietti suurimman osan elämästään laskien piin likiarvoa, ja saikin 4 611 686 018 427 387 904-sivuisen monikulmion avulla määritettyä piin 35 desimaalin tarkkuudella. Vuoteen 1945 mennessä piin likiarvo oli saatu kehittyneempien matemaattisten menetelmien avulla selvitettyä jo 620 desimaalin tarkkuudella. Vuoden 1947 jälkeen matemaatikot alkoivat käyttää tietokoneita piin desimaalien laskemiseen, ja uusien desimaalien löytymistahti kasvoikin räjähdysmäisesti. Esimerkiksi vuonna 1999 Hitachi SR8000 -supertietokone laski 206 158 430 000 desimaalia, mikä on lähes 6 miljardia kertaa enemmän kuin van Ceulen kykeni laskemaan mekaanisesti koko elämänsä aikana. Nykyisin kotitietokoneetkin pystyvät laskemaan biljoonia piin desimaaleja sekunneissa. (Bentley, 2008, ss. 146–149.)

Tieteellinen laskenta edellyttää usein hyvin suurta laskentatehoa tietokoneelta (Tsaftaris, 2014). Suuren mittakaavan tutkimukset vaativat monesti useiden tietokoneiden valjastamista laskennan rinnakkaista suorittamista varten (Vecchiola ym., 2009). Useat matemaattiset ratkaisumenetelmät fysikaalisen todellisuuden teorioiden tuottamiin ongelmiin ovat luonteeltaan sellaisia, että niiden ratkaiseminen edellyttää raskasta laskentaa (Golub, 1992, s. 9).

Suurta laskentakapasiteettia tarvitaan siihen, että laskentatehtävät pystytään suorittamaan järkevässä ajassa. Ratkaisun tehokkuuteen vaikuttavat myös ongelmanratkaisuun laaditut algoritmit. 1960-luvulta lähtien tutkijat ovat olleet tietoisia siitä, että pelkkä algoritmisen ratkaisun olemassaolo ei tarkoita sitä, että ongelma voitaisiin vain antaa tietokoneelle ja se tuottaisi ennen pitkää ratkaisun. Tietokoneessa on aina rajallinen määrä muistia ja laskentatehoa, mikä asettaa fysikaaliset rajoitteet algoritmin suorittamiselle (Hromkovi, 2004, s. 169).

Koska tieteelliset ongelmat vaativat runsaasti laskentatehoa, tulisi myös tietokoneen kyetä suorittamaan algoritmeja mahdollisimman tehokkaasti. Tähän voidaan vaikuttaa muun muassa optimoinnilla (optimization) sekä sovelluksen ohjelmointikielen valinnalla. Perinteisesti tieteellisten laskentasovellusten käytetyin kieli on ollut Fortran, mutta myös C- ja C++-kielillä on tehty paljon tieteellisiä laskentasovelluksia (Golub, 1992; Zachary, 1998). Matemaattisten ja tieteellisten ongelmien ratkaisua varten on laadittu erityisiä ohjelmointikieliä, kuten kaupallinen MATLAB ja avoimen lähdekoodin R. Nykyisin myös Python on suosittu kieli tieteellisten laskentasovellusten laatimiseen (Day, 2014).

Useat tieteen laskennalliset ongelmat ovat luonteeltaan sellaisia, että ne voidaan jakaa algoritmisesti pienempiin osiin, jolloin osaongelmat voidaan laskea samaan aikaan usealla eri prosessorilla joko yhdellä tai useammalla tietokoneella. Tätä kutsutaan *rinnakkaiseksi laskennaksi* (parallel computing). Rinnakkaisessa laskennassa prosessointikuorma jaetaan useammalle laskevalle instanssille, jolloin laskentatehoa voidaan kasvattaa lisäämällä instanssien määrää. Rinnakkaisesta laskennasta on tullut tärkeä osa tieteellisiä laskentasovelluksia, sillä sitä apuna käyttäen on pystytty muun muassa selvittämään ihmisen genomi,

luomaan uusia materiaaleja sekä käsittelemään avaruusteleskooppien tuottamia valtavia datamassoja. (Grama, 2003.)

2.2 Tieteellisiä laskentasovellusalueita

Tässä alaluvussa esitellään muutamia keskeisiä tieteellisten laskentasovellusten sovellusalueita. Erilaisia tieteellisiä laskentasovelluksia on valtava määrä, joten tässä käsitellään laskentasovelluksia kolmen sovellusalueen avulla. Nämä sovellusalueet ovat simulaatiot, data-analytiikka ja optimointi. Simulaatioita käsitellään yleisellä tasolla havainnollistaen, mistä simulaatioissa on kysymys, sekä selittäen miksi simulaatioita ylipäänsä laaditaan. Data-analytiikkaa lähestytään erityisesti tiedonlouhinnan näkökulmasta. Optimointi esitellään hyvin lyhyesti, koska tämän tutkielman puitteissa optimoinnin matemaattiseen perustaan ei ole syytä syventyä, mutta merkittävyytensä vuoksi sitä ei haluta jättää mainitsematta kaan.

2.2.1 Simulaatiot

Seuraavaksi esitellään tietokoneella tehtävien simulaatioiden yleisiä piirteitä Lawin ja Keltonin (2000) mukaan. Muita lähteitä käytetään täydentävästi.

Tietokoneella tehtävässä simulaatiossa pyritään jäljittelemään jotain todellisen maailman havaittua tai hypoteettista ilmiötä hyvin laadittua matemaattista mallia käyttäen (Heermann, 1990, s. 8; Law & Kelton, 2000). Winsberg (2014) mainitsee simulaatioiden käytännöllisiksi tarkoituksiksi sellaisen datan tuottamisen, jota ei ole saatavilla, sekä ymmärryksen lisäämisen siitä datasta, joka on jo olemassa.

Simulaatiossa tarkastelun kohteena on jonkin järjestelmän (system) oletettu toiminta. Oletettu toiminta ilmenee matemaattisina tai loogisina vuorovaikutussuhteina, jotka muodostavat mallin (model), jonka avulla pyritään kasvattamaan ymmärrystä järjestelmän toiminnasta. Mikäli vuorovaikutussuhteet ovat riittävän yksinkertaisia, voidaan algebran, differentiaali-, integraali- ja todennäköisyyslaskennan avulla johtaa analyyttisesti eksaktia tietoa kiinnostuksen kohteena olevasta ilmiöstä. Suurin osa todellista maailmaa kuvastavista malleista on kuitenkin liian monimutkaisia, jotta niiden analyyttinen tarkastelu olisi realistista. Tällöin mallia tutkitaan simulaatioiden avulla. Simulaatiossa mallille tuotetaan arvoja numeerisesti tietokoneen avulla ja kerätään simulaation tuottamaa dataa, jonka perusteella arvioidaan mallin todellisia ja odotettuja ominaisuuksia. (Law & Kelton, 2000.)

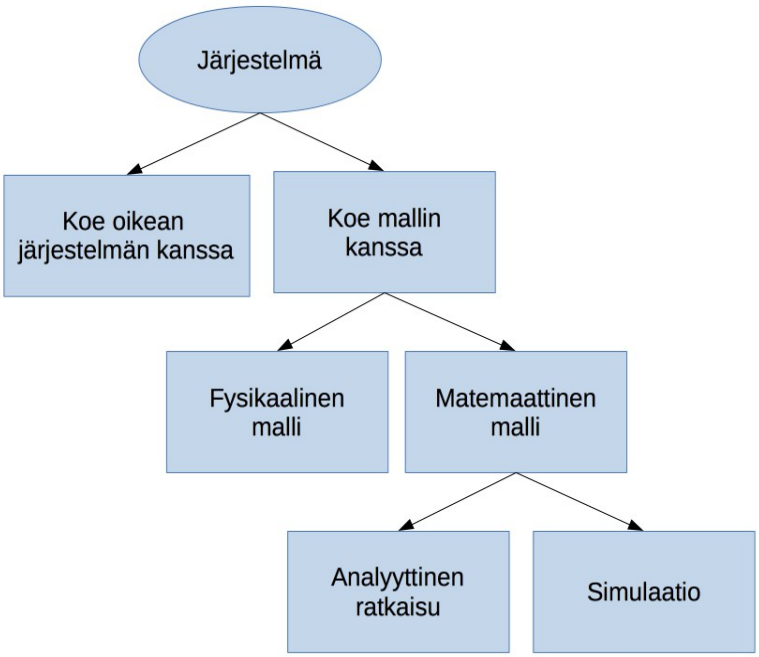
Järjestelmä (system) muodostuu joukosta entiteettejä, joiden toiminta ja vuorovaikutus johtavat johonkin loogiseen lopputulemaan (Schmidt & Taylor, 1970). Entiteetit (entity) ovat tarkastelun kannalta järkeviä yksiköitä kohteena olevasta ilmiöstä riippuen. Mikäli kiinnostuksen kohteena on esimerkiksi pankkijärjestelmä ja halutaan tarkastella erityisesti kysymystä ”paljonko konttorissa tarvitaan virkailijoita”, on mielekästä sisällyttää malliin ainakin entiteetit konttori, asiakas ja pankkivirkailija. Laajempaa kokonaisuutta tutkiva malli voi sisältää myös muita entiteettejä, kuten tili, raha, parkkipaikka jne. Järjestelmän tilalla (state) tarkoitetaan joukkoa muuttujia, jotka tarvitaan kuvaamaan järjestelmää tiettynä ajanhetkenä. Järjestelmä voidaan luokitella joko diskreetiksi tai jat-

kuvaksi. Diskreetissä järjestelmässä tilaa kuvaavat muuttujat voivat saada ainoastaan diskreettejä arvoja, eli toisin sanoen muuttujien tila muuttuu ajan edetessä välittömästi tilasta toiseen ilman välivaiheita. Esimerkiksi pankkijärjestelmässä tilille tehtävä talletus saa tilin saldon muuttumaan välittömästi esimerkiksi arvosta 0 arvoon 100 ilman, että käydään läpi arvoja 0,00...1-99,999... . Jatkuvassa järjestelmässä tilamuuttujien arvot voivat muuttua jatkuvasti ajan edetessä. Esimerkiksi ilmassa lentävä lentokone ei liiku hyppäyksittäin paikasta toiseen, vaan paikkamuuttujat voivat saada jatkuvasti muuttuvia arvoja ajan edetessä. Järjestelmä on harvoin kokonaisuutena joko täysin jatkuva tai diskreetti, mutta yleensä toinen näistä piirteistä esiintyy järjestelmässä selkeästi hallitsevana, minkä vuoksi järjestelmiä on mielekästä luokitella tätä jaottelua käyttäen. (Law & Kelton, 2000, s. 3.)

Tutkimuksen validiteetin kannalta paras ratkaisu olisi tutkia todellista järjestelmää, mutta aina se ei ole mahdollista tai järkevää. Järjestelmää tutkitaan simulaatioiden avulla, kun seuraavat ehdot toteutuvat osittain tai kokonaan:

- Todellisen järjestelmän tutkiminen on liian kallista, monimutkaista tai mahdotonta.
- Fysikaalisen mallin rakentaminen järjestelmästä on liian kallista, monimutkaista tai mahdotonta.
- Järjestelmästä laaditun matemaattisen mallin analyyttinen ratkaiseminen ei ole mahdollista (ainakaan järkevässä ajassa).

Simulaatiota voidaan pitää ”viimeisenä vaihtoehtona”, kun järjestelmän tarkastelu on muutoin liian monimutkaista tai mahdotonta (kuvio 2). Todellisuudessa simulaatioon joudutaan turvautumaan hyvin usein, koska reaalimaailman ilmiöiden pohjalta laaditut matemaattiset mallit ovat monesti hyvin monimutkaisia ratkaistavaksi. (Law & Kelton, 2000, ss. 3–5.)



KUVIO 2 Simulaatio "viimeisenä vaihtoehtona" (Law & Kelton, 2000, s. 4)

Simulaatio eroaa analyyttisesta tarkastelusta siten, että analyyttinen tarkastelu tuottaa eksakteja tuloksia, mutta simulaatiossa mallille syötetään muuttujien arvoja (input) ja tutkitaan näiden syötteiden arvojen tuottamia muutoksia mallin tuottamiin tuloksiin (output). (Law & Kelton, 2000, s. 5.)

Järjestelmän matemaattisen mallin pohjalta laaditaan simulaatiomalli. Näitä simulaatiomalleja voidaan tarkastella Lawin ja Keltonin (2000, ss. 5–6) mukaan kolmessa eri ulottuvuudessa:

1. staattiset vs. dynaamiset simulaatiomallit
2. deterministiset vs. stokastiset simulaatiomallit
3. jatkuvat vs. diskreetit simulaatiomallit.

Staattinen simulaatiomalli kuvaa järjestelmää jonain tiettynä ajanhetkenä tai edustaa kuvausta järjestelmästä, jossa ajalla ei ole merkitystä. Monte Carlo -simulaatiot ovat esimerkkejä staattisesta simulaatiomallista. Monte Carlo -simulaatioissa suoritetaan determinististä laskentaa suurelle määrälle syötteenä käytettäviä satunnaislukuja, jotka on generoitu ongelmakohteen syötemuuttujien todennäköisyysjakauman avulla (Binder & Heermann, 2010, ss. 7–8). Esimerkiksi pii voidaan määrittää Monte Carlo -simulaatiolla siten, että piirretään paperille neliö, jonka sisään piirretään ympyrä. Sen jälkeen heitetään suuri määrä (vaikkapa 30 000) hiekanjyviä paperin päälle ja oletetaan niiden jakautuvan siten, että $\pi/4$ hiekanjyvistä laskeutuu piirretyn ympyrän sisään; tällaista asetelmaa tutkivaa simulaatiota kutsuttaisiin nimellä ”Monte Carlo -simulaatio piin määrittämiseksi” (Winsberg, 2014). Dynaaminen simulaatiomalli puolestaan kuvastaa järjestelmää, jonka tila kehittyy ajan kuluessa, kuten esimerkiksi malli, joka kuvaa liukuhihnan etenemistä tehtaassa. (Law & Kelton, 2000, s. 6.)

Simulaatiomallia, jossa satunnaisuudella ei ole minkäänlaista roolia, kutsutaan deterministiseksi simulaatiomalliksi. Stokastisessa simulaatiomallissa ainakin osa syötteestä on satunnaistettua, minkä seurauksena myös simulaation tulos on aina jossain määrin satunnainen, ja siksi tulosta tulee käsitellä vain arviona mallin todellisesta luonteesta. Hyvin usein ainakin osaa syötemuuttujista täytyy käsitellä satunnaisina. Tämä on myös yksi simulaatioiden merkittävimmistä heikkouksista. (Law & Kelton, 2000, s. 6.)

Jatkuvan ja diskreetin mallin ero käsiteltiin edellä järjestelmien yhteydessä, mutta tässä yhteydessä on merkillepantavaa, että esimerkiksi luonteeltaan diskreetin järjestelmän kuvaamiseen ei välttämättä käytetä aina diskreettiä simulaatiomallia ja vastaavasti jatkuvalla simulaatiomallilla ei aina kuvata jatkuvaa järjestelmää. Valinta tehdään tutkimuksen tavoitteisiin perustuen. Esimerkiksi moottoritieliikenteen simulaatiomalli voi olla diskreetti, mikäli halutaan tarkastella yksittäisten autojen piirteitä ja liikkumista. Toisaalta moottoritien liikenteestä voidaan laatia myös jatkuva malli, jossa esimerkiksi liikenteen virtaa käsitellään differentiaaliyhtälöinä. (Law & Kelton, 2000, s. 6.)

Simulaation verifiointi on prosessi, jossa päätellään, ovatko simulaation tuottamat arviot todellisia ratkaisuja alkuperäisen mallin määrittämille differentiaaliyhtälöille. Simulaation validointi on puolestaan prosessi, jossa pohditaan, onko valittu malli riittävän hyvä representaatio todellisesta maailmasta tutkimuksen tarpeisiin nähden. (Winsberg, 2014.)

Simulaatiot ovat tärkeä osa tieteellisiä laskentasovelluksia. Tietokoneellista simulaatiota merkittävästi hyödyntäviä tieteenaloja ovat Winsbergin (2014) mukaan muun muassa astrofysiikka, hiukkasfysiikka, materiaalitieteet, tekniikka,

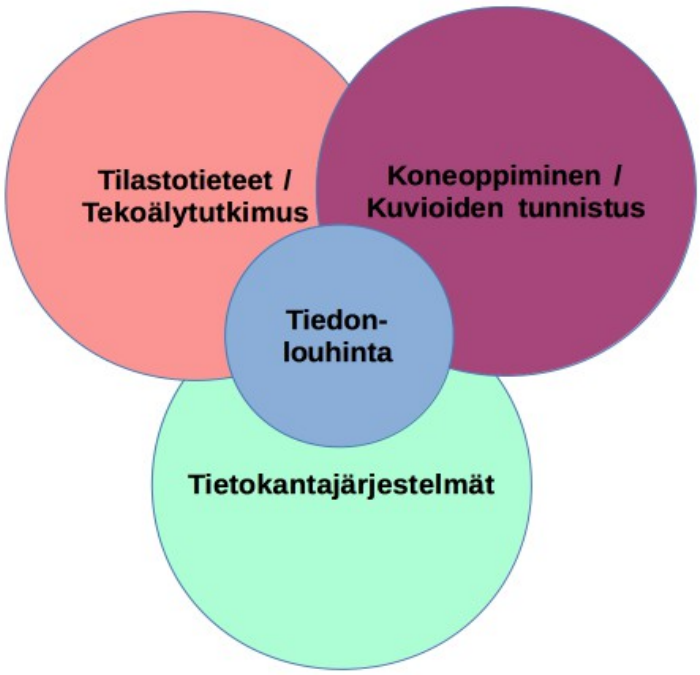
virtausmekaniikka, ilmastontutkimus, evoluutiobiologia, ekologia, taloustieteet, päätöksentekoteoria, lääketiede, sosiologia sekä epidemiologia.

2.2.2 Data-analytiikka

Tietokoneiden valtava laskentateho mahdollistaa datan analysoimisen monimutkaisten matemaattisten ja tilastollisten menetelmien avulla, jotka olisivat liian aikaavieviä ratkaistavaksi perinteisiä mekaanisia laskumenetelmiä käyttäen (Efron & Tibshirani, 1990). Tiedonlouhinta (data mining) pyrkii tuottamaan uutta tietoa etsimällä toistuvia ilmiöitä (patterns) olemassa olevasta suuresta datamassasta (Tan, Steinbach & Kumar, 2005, s. 2).

Dataa kerätään nykyisin valtavia määriä kaupallisista ja tieteellisistä syistä. Suuria datamassoja tieteellistä tutkimusta varten tuottavat muun muassa ilmakehää tarkkailevat satelliitit, avaruutta kartoittavat teleskoopit, geneettistä dataa tuottavat DNA-sirut sekä tietokoneella ajettavat simulaatiot. Suuret datamassat kätkevät usein sisäänsä tietoa, joka ei ilmene datasta suoraan tarkasteltuna. Suurten datamäärien analysointi voi viedä ihmiseltä viikkoja, ja suurta osaa datasta ei tutkitakaan koskaan. Tiedonlouhinnassa pyritään automaattisesti tai puoliautomaattisesti tunnistamaan suuresta datamäärästä merkityksellisiä kaavoja (pattern). (Tan ym., 2005, s. 2.)

Tiedonlouhinta yhdistää ajatuksia tilastotieteestä, tekoälytutkimuksesta, koneoppimisesta, mallien tunnistuksesta sekä tietokantajärjestelmistä (kuvio 3). Tiedonlouhintamenetelmät ovat muodostuneet, koska perinteisin menetelmin ei ole pystytty käsittelemään valtavia moniulotteisia ja heterogeenisia datamassoja riittävän tehokkaasti. Tiedonlouhinnan on mahdollistanut tietokoneiden laskentatehon ja tallennuskapasiteetin kasvu sekä kustannusten tippuminen. (Tan ym., 2005, s. 6.)



KUVIO 3 Tiedonlouhinnan perusta (Tan ym., 2005)

Tiedonlouhinnan menetelmät voidaan jakaa ennustaviin ja kuvaaviin menetelmiin. Ennustavissa menetelmissä pyritään joidenkin muuttujien avulla ennakoimaan muiden muuttujien arvoja (jotka ovat joko tunnettuja tai tuntemattomia) tulevaisuudessa. Kuvaavat menetelmät keskittyvät havaitsemaan datamassasta inhimillisesti tulkittavia kaavoja, jotka kuvaavat dataa. Seuraava taulukko käsittää tiedonlouhinnan keskeiset tehtävät näihin menetelmäluokkiin jaoteltuna (taulukko 1). (Tan ym., 2005, s. 7.)

TAULUKKO 1 Tiedonlouhinnan tehtävät (Tan ym., 2005, s. 7)

Luokka	**Tehtävä**
Ennustava	Luokittelu (classification)
Ennustava	Regressioanalyysi
Ennustava	Hajonnan tunnistus
Kuvaava	Ryhmittely (clustering)
Kuvaava	Assosiaatiosääntöjen havaitseminen
Kuvaava	Kaavantunnistus (pattern discovery)

Tiedonlouhinta yhdistää perinteisiä tilastomenetelmiä, koneoppimista ja tietokantajärjestelmiä kokonaisuudeksi, jonka avulla on mahdollista tunnistaa valtavassa datamassassa piileviä kuvioita, jotka antavat merkittävästi lisää tietoa datan luonteesta.

2.2.3 Optimointi

Optimoinnissa (optimization) pyritään määrittämään parhaat ratkaisut tietyille matemaattisesti määritellyille ongelmille, jotka ovat usein malleja fysikaalisesta todellisuudesta. Ennen nykyaikaisen tietokoneen keksimistä useita muuttujia sisältävien funktioiden numeerisia optimointimenetelmiä oli käsitelty hyvin vähän, mutta 1940-luvulla digitaalisen tietokoneen kehittäminen sysäsi optimoinnin tutkimuksen liikkeelle toden teolla, ja 1960-luvun alussa kyettiin jo ohjelmoimaan 100 muuttujan ongelmia lyhyessä ajassa ratkaisevia sovelluksia. (Fletcher, 2013, s. 3.)

Optimointimenetelmien sovellusalueet kattavat lähes kaiken toiminnan, jossa käsitellään numeerista informaatiota. Optimointia hyödynnetään muun muassa luonnontieteissä, tekniikassa, taloustieteissä ja mainonnassa. Optimointia sovelletaan lisäksi muihin numeerisen analyysin haaroihin, kuten datan sovittamiseen ei-lineaarisiin osittaisdifferentiaaliyhtälöihin. (Fletcher, 2013, s. 4.)

Optimoinnilla voidaan vaikuttaa ennen kaikkea siihen, kuinka tehokkaita algoritmeja pystytään käyttämään ongelman ratkaisemiseksi. Tehokkaammat algoritmit vaativat luonnollisesti vähemmän resursseja ohjelman suorittamiseen, mikä voi vaikuttaa merkittävästi myös tutkimuksen kustannuksiin.

2.3 Yhteenveto

Fysikaalisen maailman toimintaa selittävät teoriat nojaavat matemaattisiin malleihin, ja näiden mallien numeeriset ratkaisut edellyttävät usein raskasta laskentaa. Tieteelliset laskentasovellukset ovat tietokoneohjelmia, jotka laskevat tällaisia ratkaisuja. Jotta pystyttäisiin suoriutumaan yhä monimutkaisem-

mista laskentatehtävistä järkevässä ajassa, tarvitaan jatkuvasti suurempia määriä nopeampia tietokoneita ja tehokkaampia algoritmeja. Tällöin myös tehtävän tutkimuksen rahallinen resurssientarve ja energiankulutus kasvavat. Optimoinnin avulla pyritään kehittämään tehokkaampia algoritmeja numeeristen ongelmien ratkaisemiseksi. Rinnakkainen laskenta valjastaa useampia prosessoreita ratkaisemaan ositettavissa olevia ongelmia. Tieteellisessä laskennassa joudutaan myös usein käsittelemään valtavia datamassoja. Havaintolaitteet ja simulaatiot voivat tuottaa erittäin suuria määriä dataa, jonka käsittelyyn, analysointiin ja tulkitsemiseen tarvitaan yhä enemmän tallennuskapasiteettia ja prosessointitehoa.

3 PILVIPALVELUT

Tässä luvussa esitellään aluksi tiivistetysti pilvipalveluiden määritelmä ja pilvipalveluiden keskeisiä piirteitä. Sen jälkeen esitellään, millaisia yleisesti tunnistettuja pilvimalleja ja pilvipalvelumalleja on olemassa. Suurimmassa osassa pilvipalveluita koskevaa tutkimusta on keskitytty kuvaamaan pilvipalveluiden mahdollisuuksia ja liiketoimintamallia yritysten näkökulmasta, mutta tämän tutkielman yhteydessä keskitytään pilvipalveluihin erityisesti tieteentekijöiden näkökulmasta. Lopuksi pilvipalveluita lähestytään erityisesti AWS (Amazon Web Services) -pilvipalveluita esimerkkinä käyttäen.

3.1 Määritelmä ja yleisiä piirteitä

Pilvi-sanan käyttö pilvipalveluissa juontanee juurensa siihen, kun pilvisymbolia alettiin käyttää 1980-luvulla kuvaamaan asiakkaan ja puhelinoperaattorin vastuulla olevien laitteiden välistä rajapintaa. Jo vuonna 1961 John McCarthy oli esittänyt ajatuksen tietokonekapasiteetin hankkimisesta sähkön tai veden kaltaisesti, eli silloin kun niitä tarvitaan. (Heino, 2010, ss. 32–33.)

Pilvipalveluiden varsinainen nousu alkoi kuitenkin vasta 2000-luvun puolivälissä, kun suuret palveluntarjoajat alkoivat tarjota laskentakapasiteettiaan ja muuta IT-infrastruktuuriaan palveluina internetin välityksellä. Nousun taustalla oli pilvipalveluiden pohjana toimivien keskeisten teknologioiden, eli virtualisoinnin, tietoliikenteen, tallennuksen ja rinnakkaisen laskennan, kypsyminen yhdessä palveluntarjoajien tuottamien erilaisten sovellusten selainrajapintojen kanssa. (Brian ym., 2012, s. 4; Zhang ym., 2010.)

Pilvipalvelu on laaja käsite, eikä sen määrittely ole täysin yksiselitteistä (Zhang ym., 2010). Salon (2010) mukaan pilvi (cloud) on kielikuva, joka viittaa internetiin, ja pilvipalvelu on malli, jossa tietotekniikkaresursseja tarjotaan verkon välityksellä asiakkaan käyttöön. Nykyisin useimpia web-palveluita nimitetään pilvipalveluiksi, monesti puhtaasti markkinointisyistä (Brian ym., 2012, s. 6). Kattavan ja yleisesti käytetyn määritelmän pilvipalveluille on laatinut National Institute of Standards and Technology (NIST), jonka mukaan pilvipalvelu määrittyy viiden ominaispiirteen mukaan (taulukko 2). Ominaispiirteet ovat itsepalvelullisuus (on-demand self-service), pääsy palveluihin eri laitteilla, resurssien yhteiskäyttö, nopea joustavuus ja käytön tarkka mittaaminen.

TAULUKKO 2 Pilvipalveluiden ominaispiirteet (Brian ym., 2012, s. 7; Mell & Grance, 2009), suomennuksessa käytetty (Salo, 2010).

Ominaisuus	**Selite**
Itsepalvelullisuus	IT-resurssit ovat haluttaessa saatavissa palveluna ilman manuaalista interventiota.
Pääsy palveluihin eri laitteilla	Palvelu on saavutettavissa verkon välityksellä riippumatta käyttäjästä ja päätelaitteesta.
Resurssien yhteiskäyttö	Palveluntarjoaja jakaa tarvittavat resurssit useiden käyttäjien kesken esim. virtualisointia apuna käyttäen.
Nopea joustavuus	Tarvittavien resurssien hankkiminen ja vapauttaminen on mahdollista nopeasti ilman manuaalista interventiota.
Käytön tarkka mittaaminen	Palvelun käyttöön tarvittavien resurssien määrän on oltava tarkasti mitattavissa, jotta kulutuksen mukainen laskuttaminen on mahdollista.

Pilvipalveluille erityisiä piirteitä ja etuja ovat Brianin ym. (2012) mukaan kustannusten hillintä (cost containment), innovaationopeus (innovation speed), saavutettavuus (availability), skaalautuvuus (scalability) ja tehokkuus (efficiency). Zhang ym. (2010) lisäävät tähän joukkoon käsitteet virtualisointi (virtualization), luotettavuus (high reliability) sekä erittäin suuri mittakaava (ultra-large scale).

Pilvipalveluita käyttämällä voidaan välttyä oman IT-infrastruktuurin hankkimiselta ja ylläpidolta, kun tarvittava infrastruktuuri ja sovellukset ovat verkon kautta saavutettavissa ja niistä maksetaan käytön mukaan. Infrastruktuuriin ja sen ylläpitoon varatut resurssit voidaan tällöin kohdentaa ydintoimintaan. IT-resurssit eivät myöskään seiso tyhjän panttina silloin kun niitä ei tarvita. (Brian ym., 2012, s. 7)

Perinteisiin IT-projekteihin verrattuna pilvipalvelut voidaan ottaa käyttöön hyvin nopeasti. Palveluntarjoajan tehtävänä on taata, että resurssit ovat otettavissa käyttöön välittömästi verkon välityksellä. Perinteisellä mallilla hankittujen IT-resurssien käyttöönotto voi kestää viikkoja tai jopa kuukausia (Brian ym., 2012, s. 7). Zhang ym. (2010) korostavat virtualisoinnin merkitystä resurssien nopean käyttöönoton mahdollistavana tekijänä. He käyttävät pilvipalveluista vertausta virtuaalisena palvelinkeskuksena, jossa käyttäjä voi konfiguroida tarvitsemansa resurssit internetin välityksellä.

Suuret pilvipalveluntarjoajat pystyvät tarjoamaan korkean saavutettavuuden skaalautumiskykynsä ansiosta (Brian ym., 2012, s. 7). Saavutettavuus on tärkeä huolenaihe sekä palveluntarjoajan että palvelun käyttäjien kannalta. Forbes-lehden arvion mukaan vuonna 2013 Amazon menetti tuloja yli 66 000 Yhdysvaltain dollaria jokaista minuuttia kohden, kun sen AWS-pilvipalvelut olivat käyttäjien saavuttamattomissa (Clay, 2013). On kuitenkin eriteltävä, että suunniteltu saavuttamattomuus on sekä palveluntarjoajalle että asiakkaalle huomattavasti vähemmän haitallista kuin suunnittelematon saavuttamattomuus (Wee, 2011). Zhangin ym. (2010) mukaan pilvipalveluiden piirteisiin lukeutuvat varmistusmenetelmät tekevät pilvipalveluista paikallista tietokonetta luotettavamman.

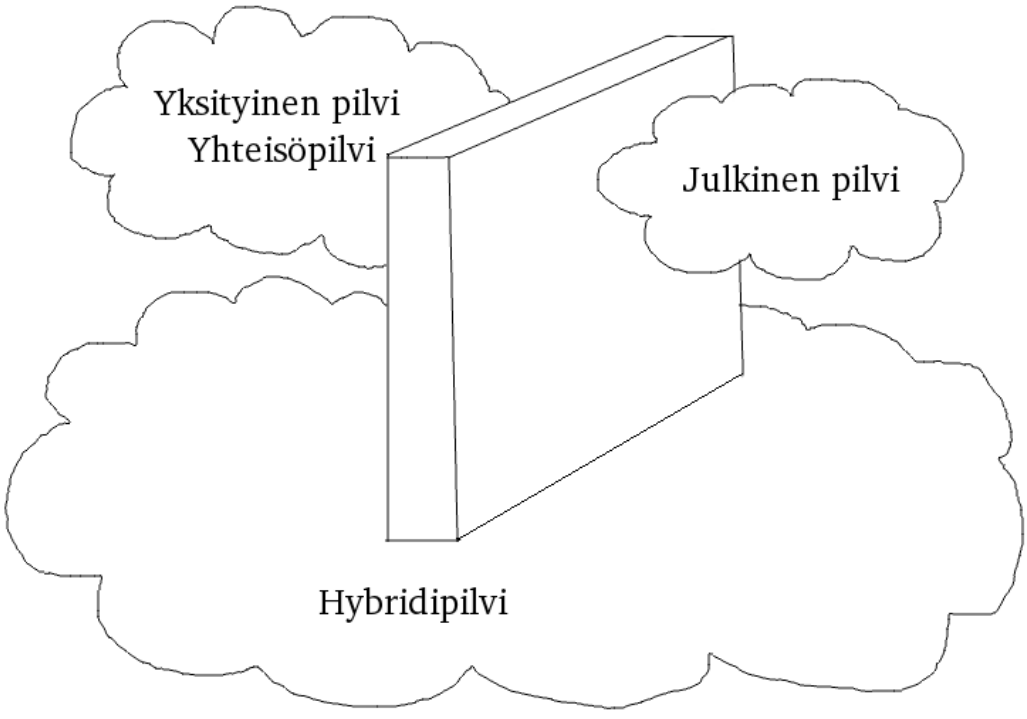
Pilvipalveluiden joustavuus ja skaalautuvuus mahdollistavat nopean ja joissain tapauksissa jopa automaattisen reagoinnin muuttuviin IT-resurssitarpeisiin (Brian ym., 2012, s. 7). On erittäin kustannustehokasta, että resursseja voidaan ottaa käyttöön silloin ja vain silloin kun niitä todella tarvitaan, koska tällöin resursseja ei mene hukkaan. Zhangin ym. (2010) mukaan pilvipalvelut ovat luonteeltaan ”äärimmäisen edullisia” verrattuna perinteisiin järjestelmiin, joissa laitehankinnat ja ylläpito syövät suuren osan resursseista.

Elastisuudella tarkoitetaan pilvipalveluiden kykyä adaptoitua korkeisiin kuormapiikkeihin sekä myös mahdollisuutta suojautua luonnonkatastrofeilta. Tämä on mahdollista siten, että palveluntarjoajalla on palvelinkeskuksia useissa eri maantieteellisissä sijainneissa, jolloin resursseja voidaan jakaa näiden palvelinkeskusten välillä ja kriittiset toiminnot voidaan peilata useaan maantieteelliseen sijaintiin. Tällöin yhden keskuksen tuhoutuessa peilatut resurssit voivat jatkaa toimintaansa toisessa fyysisessä sijainnissa. (Brian ym., 2012, s. 7.)

Pilvipalveluiden tarjoamat mahdollisuudet ja edulliset kustannusmallit houkuttelevatkin siirtämään raskasta laskentaa ja suurta tallennuskapasiteettia vaativat prosessit suoritettaviksi ja varastoitaviksi pilveen (Huang, Yu, & Yu, 2013; Srirama, Ivanistsev, Jakovits & Willmore, 2013). Toisaalta tekijänoikeuskysymykset ja tietoturva herättävät kysymyksiä pilvipalveluiden tapauksessa (Salo, 2010). Tässä tutkielmassa keskitytään pilvipalveluiden tarjoamiin mahdollisuuksiin tieteentekijöille, joten liiketoiminnan kannalta kriittisempiä tietoturva- ja tekijänoikeusseikkoja käsitellään vain pintapuolisesti.

3.2 Pilvimallit

Pilvipalveluita voidaan ottaa käyttöön usealla eri mallilla (Brian ym., 2012, s. 11). NIST-määritelmän mukaiset pilvimallit ovat yksityinen pilvi (private cloud), yhteisöpilvi (community cloud), julkinen pilvi (public cloud) sekä hybridipilvi (hybrid cloud). Hybridipilvi voidaan ajatella sekoituksena julkista ja yksityistä pilveä, ja yhteisöpilvi voidaan katsoa yksityisen pilven erikoistapaukseksi (kuvio 4).



KUVIO 4 Pilvimallit

Yksityinen pilvi (private cloud) tarkoittaa sitä, että pilvi-infrastruktuuri on organisaation LAN-lähiverkon tai muun luotetun verkon kautta käytettävissä ilman tarvetta erilliselle tietoliikenneyhteydelle (Heino, 2010, s. 55). Yksityinen pilvi voi olla organisaation itsensä omistama tai vuokrattu kokonaan sen omaa käyttöä varten kolmannelta osapuolelta (Brian ym., 2012, s. 11). Yksityisessä pilvessä kaikki kustannukset kohdentuvat pilveä käyttävälle organisaatiolle (Heino, 2010, s. 55).

Käytännön näkökulmasta yksityinen pilvi pitää sisällään LAN-verkon järjestelyitä, palvelimia, tallennuskapasiteettia, kuormantasauksen, klusteroinnin, VDI-järjestelmän virtuaalisten työasemien mahdollistamiseksi sekä sovellusohjelmia. Kaikki edellä mainitut toiminnot myös varmistetaan ja monistetaan. Näiden toimintojen paketointi palveluiksi sekä pilven autonominen tai lähes autonominen toiminta (hallinnan helppous) tekevät yksityisestä pilvestä houkuttelevamman vaihtoehdon kuin perinteinen konesaliratkaisu. (Heino, 2010, ss. 209–210.)

Pilvipalveluiden kaikkia etuja ei voida hyödyntää yksityisessä pilvessä, ja palveluiden muokattavuus voi olla rajoittunutta (Brian ym., 2012, s. 11). Yksityinen pilvi on myös suuri investointi, joten organisaation tulisi pystyä korvaamaan sillä lähes kaikki tietojenkäsittelytarpeensa, jotta yksityisen pilven hankkiminen olisi kannattavaa (Heino, 2010, s. 211).

Yhteisöpilvessä (community cloud) yksityinen pilvi avataan myös muille käyttäjille, jotka kuuluvat johonkin määriteltyyn ryhmään eli yhteisöön (Brian ym., 2012, s. 11). Useampi organisaatio omistaa yhteisöpilven, ja sen käytöstä koituvat kustannukset jaetaan näiden organisaatioiden kesken (Heino, 2010, s. 56). Palveluntarjoajia voi olla useita, ja tarvittava pilvi-infrastruktuuri voi sijaita yhteisöön kuuluvien organisaatioiden tiloissa tai sen ulkopuolella (Brian ym., 2012, s. 11; Salo, 2010, s. 19).

Julkisen pilven (public cloud) palvelut ovat kaikkien halukkaiden saatavilla yhden palveluntarjoajan toimesta (Brian ym., 2012, s. 11). Julkista pilvipalvelua käytetään internetyhteyden kautta, ja palveluntarjoaja tarjoaa asiakkaalle palveluiden lisäksi niiden käyttöön tarvittavat osoite- ja nimipalveluresurssit (Heino, 2010, ss. 54–55). Asiakas maksaa julkisen pilven käytöstä kuukausi-, tunti-, tai muun aikaan tai kapasiteettiin perustuvan hinnoittelun mukaisesti (Heino, 2010, s. 55). Julkisessa pilvessä skaalautuvuudesta ja resurssien yhteiskäytöstä voidaan saavuttaa suurin hyöty (Brian ym., 2012, s. 11).

Hybridipilvi yhdistää ominaisuuksia julkisesta ja yksityisestä (tai yhteisö-) pilvestä siten, että osa pilven toiminnoista on yksityisiä ja osa julkisia. (Salo, 2010, s. 19). Tämä mahdollistaa sen, että arkaluontoinen data voidaan säilyttää yksityisen pilven puolella, kun taas julkinen data ja sovellukset voidaan pitää julkisessa pilvessä (Brian ym., 2012, s. 11).

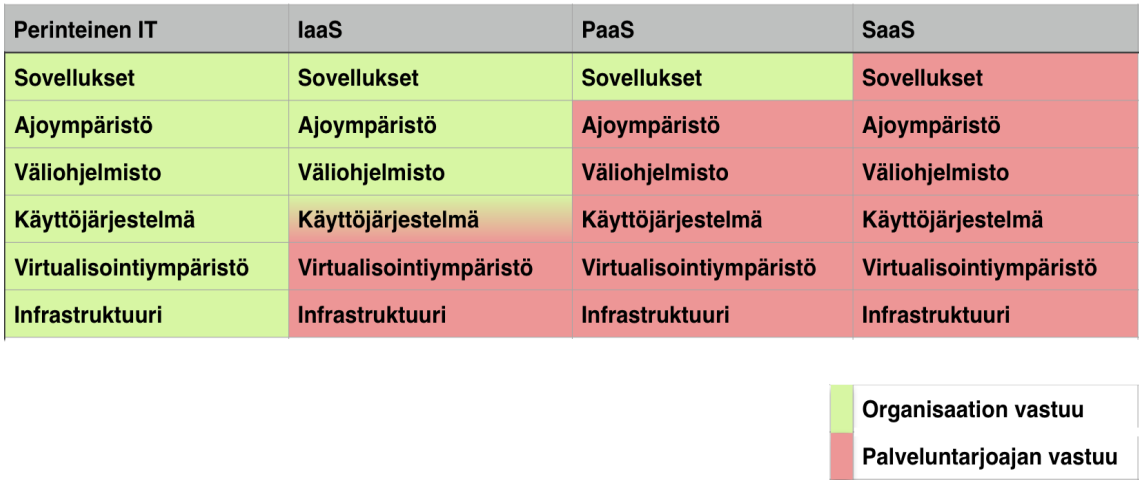
3.3 Palvelumallit

Pilvipalveluarkkitehtuuri voidaan jakaa NIST-määritelmän mukaisesti kolmeen eri kerrokseen (Mell & Grance, 2009):

1. IaaS – infrastruktuuri palveluna (Infrastructure as a Service)
2. PaaS – sovellusalusta palveluna (Platform as a Service)

3. SaaS – sovellukset palveluna (Software as a Service)

Infrastruktuurikerros (IaaS) toimii palvelualustan (PaaS) pohjana, jonka päälle rakennetaan sovelluksia (SaaS) (Salo, 2010, s. 22). Kuvio 5 havainnollistaa eri kerrosten sijoittumista toisiinsa nähden, sekä mitä vastuita ja toimintoja millä-kin kerroksella on. Kuviossa esiintyy vertailun vuoksi myös perinteinen IT-inf-rastruktuuri. Kuvassa vihreällä merkityt tasot ovat organisaation operoimia ja punaisella merkityt palveluntarjoajan tarjoamia palveluita.



KUVIO 5 Pilvipalvelumallit (Brian ym., 2012, s. 8; Mell & Grance, 2009)

Palvelumallien mahdolliset vastuut ja toiminnot ovat

- sovellukset (applications), jotka ovat organisaation johonkin erityiseen tarkoitukseen käyttämiä sovelluksia
- ajoympäristö (runtime), joka on ympäristö, jossa sovelluksia suoritetaan
- väliohjelmisto (middleware), joka hoitaa viestintää eri sovelluksien, tieto-kantojen ja käyttöjärjestelmän välillä
- käyttöjärjestelmä (OS), joka hallitsee ja tarjoaa virtualisointiympäristön tarjoamia järjestelmäresursseja käyttäjälle
- virtualisointiympäristö (hypervisor), joka on virtualisointitaso, joka tar-joaa virtualisoituja infrastruktuuriresursseja käyttöjärjestelmälle
- infrastruktuuri, joka koostuu fyysisistä laitteista, kuten palvelimista, pro-sessoreista, tiedontallennuslaitteista ja tietoliikenneverkosta. (Brian ym., 2012, ss. 8–9.)

Infrastruktuuri palveluna (Infrastructure as a Service) tarkoittaa sitä, että käyttä-jän on mahdollista ottaa käyttöön prosessointia, tiedon varastointia, tietoverk-koja sekä muita perustavanlaatuisia IT-resursseja, joihin käyttäjä voi asentaa ja ajaa mielivaltaisia ohjelmistoja, mukaan lukien käyttöjärjestelmiä ja sovelluksia (Mell & Grance, 2009, s. 8). IaaS:ssa resurssit ovat yhteiskäytössä, käyttöönotto,-ylläpito,- ja skaalautuvuustoiminnot ovat pitkälle automatisoituja ja käyttäjä maksaa infrastruktuurista käyttöperusteisesti (Salo, 2010, s. 25). Palvelun ylläpi-täjä lohkoo resurssit etukäteen määritellyiksi ja hinnoitelluiksi tuotteiksi, jotka

asiakas valjastaa käyttönsä ja asentaa näihin käyttöjärjestelmänsä ja sovelluksensa (Heino, 2010, ss. 52–53). Käyttäjä ei siis hallitse käyttöjärjestelmätason alapuolella olevia resursseja, kuten fyysisiä laitteita ja virtualisointiin vaadittavaa infrastruktuuria (Mell & Grance, 2009, s. 8), mutta käyttöjärjestelmätasosta ylöspäin käytännössä kaikki resurssit ovat käyttäjän hallittavissa.

Brian ym. (2012, s. 9) ovat listanneet IaaS:n etuja ja haittoja verrattuna siihen, että organisaatio omistaisi itse kaiken vaadittavan IT-infrastruktuurin (taulukko 3). Taulukosta voidaan todeta, että IaaS tarjoaa pilvipalveluiden edut asiakkaan käyttöön joustavasti, mutta juuri joustavuuden vuoksi asiakkaan vastuulle jää myös huomattava osuus turvallisuuteen liittyvistä seikoista.

TAULUKKO 3 IaaS:n edut ja haitat (Brian ym., 2012, s. 9)

Edut	**Haitat**
• Todellisiin tarpeisiin perustuva korkea skaalautuvuus. • Omaa erillistä tietovarastoa ei tarvita. • Datan käytön ja tallennuksen varastoinnin fyysinen erottelu. • Infrastruktuurin pystyttäminen ja ylläpito on palveluntarjoajan vastuulla. • Ei investointikuluja infrastruktuuriin. • Käyttöön perustuva laskutus.	• Datan sijainti ei ole aina identifioitavissa julkisissa ja yksityisissä pilvissä. • Riippuvuus infrastruktuurin saavutettavuudesta ja tietoliikenneyhteyksistä. • Puuttuva tai riittämätön datan prosessoinnin eristäminen eri käyttäjien välillä. • Väärinkonfigurointi voi mahdollistaa pääsyn dataan, jonka pitäisi olla suojattua. • Ei aukottomia takeita luottamuksellisuudesta, turvallisuudesta tai datan yhtenevyydestä. Kenelle vastuu kuuluu rikkeen tapahtuessa?

Sovellusalusta palveluna (Platform as a Service) tarjoaa alustan, jonka päälle käyttäjä voi rakentaa sovelluksia ja jossa voidaan tehdä sovellusten testaus-, ylläpito- ja kehitystoimia (Salo, 2010, s. 28). Palvelusta hyötyvät käyttäjät, jotka pystyvät kehittämään sovelluksensa itse, mutta haluavat valmiin ylläpidetyn ja tarpeen mukaan skaalautuvan ympäristön sovelluksilleen (Heino, 2010, s. 51). Riskinä PaaS-ratkaisussa on lukittautuminen johonkin tiettyyn palveluntarjoajaan, sillä standardoinnin puuttuessa eri palveluntarjoajilla voi olla keskenään yhteensopimattomia ratkaisuja (Brian ym., 2012, s. 11; Salo, 2010, s. 28).

Brian ym. (2012) ovat listanneet PaaS:n käytön etuja ja haittoja verrattuna tilanteeseen, jossa käyttäjä omistaa ja hallinnoi itse sovellusalustaansa (taulukko 4). Taulukosta voidaan havaita, että PaaS soveltuu tilanteisiin, joissa infrastruktuuria ei ole tarpeellista ylläpitää, mutta sovellukset halutaan kehittää itse. Toisaalta PaaS ei ole yhtä joustava vaihtoehto kuin IaaS, ja riski lukittautua palveluntarjoajaan sekä ohjelmistojen lisenssiehdot saattavat olla esteinä PaaS-palveluiden käytölle.

TAULUKKO 4 PaaS:n edut ja haitat (Brian ym., 2012, s. 10)

Edut	Haitat
• Vähemmän ylläpitoa, kun infrastruktuuria ei tarvitse itse ylläpitää. • Maantieteellisesti hajautettu kehittäminen on mahdollista. • Yhden alustan käyttäminen standardoi sovelluksia ja vähentää kuluja. • Sovellusalustan pystyttäminen ja ylläpito on palveluntarjoajan vastuulla. • Ei investointikuluja infrastruktuuriin. • Käyttöön perustuva laskutus.	• Palveluntarjoajaan lukittautuminen ◦ heikentää siirrettävyyttä ◦ heikentää yhteensopivuutta ◦ standardeja ei ole. • Riittämätön joustavuus. • Mahdolliset erityiset vaatimukset yksinoikeudella valmistettujen sovellusten ja kehitysympäristöjen tapauksessa.

Sovellukset palveluna (Software as a Service) tarkoittaa kokonaisia sovelluksia, joita käyttäjä voi käyttää internetselaimen kautta ja joista palveluntarjoaja laskuttaa käyttöperusteisesti (Brian ym., 2012, ss. 9–10). SaaS:ssa toteutuu monikäyttäjyys (multi tenancy), eli sama sovellus on laajemman käyttäjäkunnan käytössä, mutta asiakas saa silti sovelluksesta yksilöllisen käyttökokemuksen (Salo, 2010, s. 29). Palveluntarjoajat pystyvät tarjoamaan sovelluksia käyttäjille edullisesti, sillä virtualisoinnin, skaalautuvuuden ja jaettujen resurssien ansiosta palveluntarjoajalle ei koidu merkittäviä lisäkustannuksia sovelluksen käyttäjämäärän lisääntymisestä (Heino, 2010, ss. 53–54).

Brian ym. (2012) ovat listanneet SaaS:n käytön etuja ja haittoja verrattuna tilanteeseen, jossa käyttäjä omistaa ja hallinnoi itse sovelluksiaan (taulukko 5). Taulukosta voidaan todeta, että SaaS on asiakkaan näkökulmasta yksinkertaisin pilvipalveluiden käyttömuoto, mutta vastaavasti se on myös vähiten joustava.

TAULUKKO 5 SaaS:n edut ja haitat (Brian ym., 2012, s. 10)

Edut	Haitat
• Monikäyttäjyys. • Ylläpitoa ei tarvita, kun infrastruktuuria tai sovellusalustaa ei tarvitse itse ylläpitää. • Nopea sovellusten käyttöönotto. • Ei investointikuluja infrastruktuuriin. • Käyttöön perustuva laskutus. • Mobiilius ja sijainnista riippumattomuus.	• Oikean palveluntarjoajan valinta. • Siirrettävyyden puute. • Huonompi integroitavuus olemassa oleviin sovellusympäristöihin. • Vähemmän sovitusmahdollisuuksia. • Vasteajat voivat olla normaalia suuremmat. • Turvallisuus. • Käyttö vaatii internetyhteyden.

Yhteenvetona eri pilvimalleista voidaan todeta, että SaaS-palvelut ovat pilvipalveluita, joita käytetään, PaaS-palveluihin rakennetaan sovelluksia ja IaaS-palveluihin siirrytään. IaaS on palvelumalleista joustavin, mutta vastaavasti IaaS-palvelun käyttö vaatii käyttäjältä eniten vastuuta. SaaS-palvelut vaativat puolestaan vähiten käyttäjän aktiivisuutta, mutta ne sisältävät myös eniten rajoituksia. PaaS:ssa käyttäjän vastuut sijoittuvat IaaS:n ja SaaS:n välimaastoon. Kaikissa pilvipalvelumalleissa yhteistä on pyrkimys piilottaa infrastruktuurin monimutkaisuutta käyttäjän näkymättömiin, jotta käyttäjä voi keskittyä oleellisiin ongelmiin. Tätä IaaS, PaaS ja SaaS tekevät eri tasoilla. Se, mitä palvelumallia käyttä-

jän kannattaa kulloinkin käyttää, riippuu täysin käyttäjän tavoitteista ja tarpeista. (Salo, 2010, ss. 26, 36.)

3.4 Riskit, heikkoudet ja uhkakuvat

Pilvipalvelut eivät ole täydellinen ratkaisu kaikkiin tietohallinnon ongelmiin. Pilvipalvelumalli aiheuttaa myös useita erilaisia haittoja, jotka voivat joissain tapauksissa realisoitua hyötyjä suuremmiksi tekijöiksi. Pilvipalveluun siirtymisessä tulee puntaroida myös mallin mahdollisia heikkouksia. Tässä tutkielmassa keskitytään mahdollisiin heikkouksiin tieteellisen laskennan kannalta, eikä painoa aseteta juurikaan kaupallisiin seikkoihin.

Pilvipalveluiden *tietoturva* on käyttäjien mielikuvissa suurin huolenaihe ja este pilvipalveluihin siirtymiselle (Salo, 2010, s. 36). Todellisuudessa pilvipalveluiden tekninen ympäristö on usein suojattu monilla menetelmillä, joita käytetään myös perinteisissä konesaliratkaisuissa, kuten tietojen salauksen, palomuurien ja tunkeilijoiden havaitsemisjärjestelmien avulla (Heino, 2010, s. 93). Käyttäjillä on usein paljon kokemusta perinteisten konesalien tietoturvaratkaisuista, ja pilveen siirtymistä epäröidään, koska teknologia on suhteellisen nuorta ja asiantuntijoiden määrä on vähäisempi verrattuna perinteisiin ratkaisuihin (Salo, 2010, s. 103). Palveluntarjoajalla voi olla pilvipalvelumallista riippuen hyvinkin merkittävä rooli palveluiden tietoturvan teknisessä toteutuksessa, ja tämä edellyttää suurta luottamusta asiakkaan ja palveluntarjoajan välillä (Salo, 2010, s. 103).

Toinen realistinen huolenaihe on pilvipalveluiden *suorituskyky*. Pilvipalvelumallin mahdollistava virtualisointi aiheuttaa sen, että laitteiston suorituskyvystä osa joudutaan aina uhraamaan virtualisoinnin tarpeisiin, vaikkakin virtualisoinnin avulla muisti- ja prosessoriresurssit saadaan jaettua tehokkaasti (Armbrust ym., 2010). Pilvipalveluilla saavutettavien kustannusetujen suhdetta mahdolliseen suorituskyvyn menetykseen tulee arvioida, mikäli pilvessä suoritettavat tehtävät ovat luonteeltaan sellaisia, että suorituskyky nousee kriittiseksi tekijäksi (Vecchiola ym., 2009). Virtualisoinnin aiheuttamasta suhteellisesta suorituskyvyn menetyksestä oltiin huolestuneita varsinkin pilvipalveluiden alkuaikoina, mutta monet palveluntarjoajat ovat kuroneet virtualisoinnin aiheuttamaa eroa kiinni useilla teknisillä ratkaisuilla, ja nykyisin virtualisoinnin aiheuttama tehonmenetys saadaan pääasiassa hallittua optimoimalla (Xu, Liu, Jin & Vasilakos, 2014).

Merkittävä pullonkaula pilvipalveluissa on tiedonsiirto. Laskenta- ja tallennusresurssit sijaitsevat yleensä fyysisesti kaukana käyttäjästä, ja tietoa täytyy siirtää näiden välillä molempiin suuntiin. Mikäli IT-ratkaisu edellyttää valtavien datamäärien siirtämistä pilveen ja sieltä pois, muodostuu internetin rajallinen tiedonsiirtonopeus hyvin nopeasti pullonkaulaksi (Armbrust ym., 2010).

Myös tiedonsiirto pilven sisällä eri virtuaalitietokoneiden välillä voi muodostua pullonkaulaksi, erityisesti mikäli käsitellään erittäin suuria datamääriä (big data). Useilla palveluntarjoajilla on nykyisin saatavilla vaihtoehtoisia teknologioita nopeisiin tiedonsiirtomenetelmiin virtuaalikoneiden ja konesalien välillä. Big dataa käsittelevien sovellusten tietovuon tulisi olla hyvin suunniteltu ja mallinnettu, jotta saadaan tietoa sovelluksen tiedonsiirtovaatimuksista ja voidaan arvioida, pystyykö palveluntarjoaja tarjoamaan vaatimukset täyttävän infrastruktuurin. (Branch, Tjeerdsma, Wilson, Hurley & McConnell, 2014.)

Pilvipalveluiden toteutuksesta ei ole olemassa virallisia standardeja, joten eri palveluiden toteutustavat vaihtelevat palveluntarjoajien välillä. *Palveluntarjoajaan lukittautuminen* (vendor lock-in) on merkittävä huolenaihe pilvipalvelua harkitsevalle käyttäjälle (Armbrust ym., 2010; Brian ym., 2012, ss. 8–9, 14). Yhteen pilvipalveluun luotu sovellus ei välttämättä toimi sellaisenaan, kun se siirretään toisen palveluntarjoajan pilveen. Tämä voi olla erityisen kiusallista, mikäli pilvipalvelun toiminta päättyy jostain odottamattomasta syystä. Palveluiden jatkuvuus voi olla joskus kyseenalaista varsinkin pienempien toimijoiden tapauksessa. Open Grid Forumin kaltaiset tahot ovat kuitenkin alkaneet toimia lukittumisuhkaa vastaan laatimalla standardeja, joiden tarkoitus on yhdenmukaistaa palveluntarjoajien ohjelmointirajapinnat (Salo, 2010, s. 114).

Armbrust ym. (2010) nimittävät saavutettavuuden ja palveluiden jatkuvuuden merkittävimmäksi esteeksi pilvipalveluihin siirtymiselle, vaikkakin he toteavat, että vain harvoissa perinteisissä IT-ratkaisuissa saavutettavuus ja jatkuvuus on turvattu paremmin kuin pilvipalveluissa. Ratkaisuksi he ehdottavat palveluiden hankkimista useammalta palveluntarjoajalta. Palveluiden käyttökatkoilta voi suojautua parhaiten valitsemalla palveluntarjoajan, joka mahdollistaa toimintojen peilauksen useaan maantieteelliseen sijaintiin. Palvelun jatkuvuus on sitä taatumpaa, mitä suuremmalta palveluntarjoajalta palvelut on hankittu. Tämä tosiasia toisaalta myös vaikeuttaa pienten toimijoiden penetraatiota markkinoille.

3.5 Amazon Web Services

Tässä osiossa tarkastellaan erilaisia pilvipalveluiden tarjoamia mahdollisuuksia Amazon Web Servicesiä esimerkkinä käyttäen. Amazonin toimintaa esitellään yleisellä tasolla siten, että keskiössä ovat sellaiset palvelut ja piirteet, jotka ovat tieteellisen laskennan kannalta relevantteja.

3.5.1 Taustaa ja yleistä tietoa

Amazon Web Services (AWS) on kokoelma palveluntarjoaja Amazonin tarjoamia pilvipalveluita (Jinesh & Sajee, 2014; Krause, 2013). Amazon on elektroniseen kaupankäyntiin keskittyvä yritys, joka aloitti toimintansa vuonna 1994 kirjakauppana, mistä toiminta sittemmin laajentui kaikkien tavaroiden verkkokaupaksi (Heino, 2010, ss. 105–106). Amazon oli kehitellyt erilaisia keskitettyjä IT-ratkaisuja mm. massiivista verkkokauppaansa varten 1990-luvun puolivälistä lähtien, mikä johti lopulta siihen, että Amazon alkoi myydä kehittämäänsä infrastruktuuria palveluna muille käyttäjille, ja tähän tarkoitukseen lanseerattu AWS käynnistikin toimintansa vuonna 2006 (Jinesh & Sajee, 2014). Vuotta 2006 pidetään pilvitoiminnassa merkittävänä virstanpylväänä nimenomaan AWS:n toiminnan alkamisen johdosta (Heino, 2010, s. 34). Nykyisin AWS palvelee satojatuhansia asiakkaita ympäri maailman.

Amazonin pilvipalveluvalikoimaan kuuluu kattava joukko IaaS- ja PaaS-palveluita. Käytännössä IaaS mahdollistaa minkä tahansa sovellusalustan käyttämisen, mutta AWS:n yhteydessä PaaS-palveluilla tarkoitetaan valmiiksi konfiguroituja ja tuotteistettuja palveluja, jotka käyttäjä voi ottaa sellaisenaan käyttöönsä. AWS:n palveluvalikoimaan kuuluu tuotteita seuraavista kategorioista:

- tietokannat (database)
- laskenta ja verkot (compute and networking)
- sisällönvälitys (content distribution)
- tiedontallennus ja -jako (storage & content delivery)
- käyttöönotto (deployment)
- sovelluspalvelut (application services)
- tuki ja hallinta (support & management). (Jinesh & Sajee, 2014.)

Amazon pyrkii oman visionsa mukaan erottumaan kilpailijoistaan joustavuuden, kustannustehokkuuden, skaalautuvuuden ja elastisuuden sekä turvallisuuden ja kokemuksen avulla (Jinesh & Sajee, 2014). AWS:n palveluja, kuten muitakin IaaS-palveluja, voi käyttää hyvin monenlaisiin käyttötarkoituksiin. Kirjallisuudessa esiintyy monia tieteellistä laskentaa soveltavia tutkimuksia ja projekteja, joissa on käytetty hyväksi AWS:n palveluita. Tässä muutamia esimerkkejä:

- ”1000 genomia” -projekti, jossa 200 TB geenitutkimusdataa pystyttiin prosessoimaan ja pystytään edelleen jakamaan tutkijoille ympäri maailman käyttäen EC2-virtuaalikonepalvelua ja S3-tietovarastopalvelua (Z. Huang ym., 2013).
- ”Freesurfer.net”-projekti, jossa on EC2-virtuaalikoneita on valjastettu aivokuvausten analysointiin ja käsittelyyn (Tsaftaris, 2014).
- NASA:n Kepler satelliitin tuottaman kuvadatan analysointi (Juve, Rynge, Deelman, Vockler & Berriman, 2013).
- Astrologisen, seismologisen, sekä bioinformatiikan workflow-tyyppisten sovellusten ajaminen EC2-virtuaalikoneilla (Juve ym., 2010).
- Röntgen-spektroskopian mallinnus EC2-virtuaalikoneilla (Rehr, Vila, Gardner, Svec & Prange, 2011).

Amazonin itsensä mukaan AWS:ää on käytetty lisäksi muun muassa hiukkasfysiikan simulaatioihin, bioinformatiikkaan, molekyylien mallintamiseen, tekoälytutkimukseen, lääkkeiden kehittämiseen sekä tieteelliseen yhteystyöhön ja tieteellisen datan hallintaan (”Scientific Computing with EC2 Spot Instances”, 2011).

Nykyisin on olemassa myös pilvialustoja, joihin tutkijat voivat asentaa ja konfiguroida omat ohjelmistonsa ja työnkulkunsa sekä julkaista sen jälkeen sovelluksensa palveluna, mutta ainakin toistaiseksi nämä palvelut kärsivät monimutkaisuudesta ja keskeneräisyydestä (Cook, Milojicic, Kaufmann & Sevinsky, 2012; Mendez ym., 2013). Osa näistä palveluista käyttää hyväkseen AWS:n pilvi-infrastruktuuria. Esimerkki AWS:n resursseja hyödyntävästä tieteellisen laskennan palvelusta on Cycle Computing, jossa on mahdollista luoda edullisten EC2-spot-instanssien avulla jopa 30 000 prosessoriytimen laskentaklustereita (”Scientific Computing with EC2 Spot Instances”, 2011).

Eucalyptus on vapaa avoimen lähdekoodin ohjelmisto, jonka avulla voidaan rakentaa AWS:n kanssa yhteensopivia yksityisiä ja hybridipilviä. Hybridipilven tapauksessa Eucalyptus hallitsee pilven yksityistä puolta, joka kommunikoi pilven julkista osaa edustavan AWS:n kanssa. Vapaa ja avoin alusta vähentää huomattavasti palveluntarjoajaan lukittumisen riskiä. Toisaalta Eucalyptus ei tue kaikkia AWS:n palveluita, mutta keskeisimmät infrastruktuuripalve-

lut ovat täysin yhteensopivia, mukaan lukien EC2, S3 sekä automaattisen skaalauksen ja kuormantasauksen palvelut (”Eucalyptus and Amazon Web Services Compatibility”, 2014).

3.5.2 EC2

Amazon Elastic Compute Cloud (EC2) on verkkopalvelu, joka tarjoaa skaalautuvaa laskentakapasiteettia pilvessä. Käyttäjä luo EC2-instansseja valitsemalla esiasennetun virtuaalikoneen levykuvan (AMI, Amazon Machine Image) tai luomalla itse oman virtuaalikoneensa palveluun. EC2-koneita saa käyttöönsä joko hinnaston mukaisella tuntiveloituksella, joka perustuu instanssin kokoon ja valittuun käyttöjärjestelmään, tai vaihtoehtoisesti spottihinnoittelulla, jolla Amazon pyrkii myymään muuten käyttämättä jäävän kapasiteetin normaalia huomattavasti edullisemmalla hinnalla (Salo, 2010, ss. 119–120).

3.5.3 S3

Amazon Simple Storage (S3) on tarkoitettu tiedon pitkäaikaiseen tallennukseen. Palvelun kapasiteetti näkyy käyttäjälle yksinkertaistettuina hakemistoina, joista tietoja voidaan hakea suoraan HTTP:n avulla jokaiselle tiedostolle luotavan URL-osoitteen kautta. Palvelussa voidaan määrittää, kenellä on pääsy tiedostoihin. Amazon takaa talletetuille tiedoille yli 99,99 %:n säilyvyyden vuodeksi (Heino, 2010, ss. 107–108).

Eri maissa (erityisesti EU:n ja Yhdysvaltojen välillä) poikkeavien lainsäädäntöjen vuoksi Amazon takaa, että esimerkiksi Irlannissa sijaitseville AWS-palvelimille S3-palveluun talletettuja tietoja ei koskaan varastoida EU:n ulkopuolelle, mikäli käyttäjä niin haluaa (Salo, 2010, s. 121).

3.5.4 Elastic Map Reduce

Amazon Elastic Map Reduce (EMR) on palvelu valtavien datamäärien prosessointiin. EMR käyttää avoimen lähdekoodin HADOOP-viitekehystä, joka jakaa datan osiin, jotta halutun kokoinen EC2-klusteri voi prosessoida osat yhtäaikaisesti. Amazonin käyttäjät ajavat miljoonia EMR-klustareita vuosittain, ja sen käyttökohteiden sovellusalueita ovat muun muassa koneoppiminen, talousanalyysi, tieteelliset simulaatiot ja bioinformatiikka. (Jinesh & Sajee, 2014.)

3.6 Yhteenveto

Pilvipalvelut mahdollistavat monenlaisia kustannustehokkaita ja käyttäjäystävällisiä ratkaisuja infrastruktuurin, ohjelmistoalustojen ja sovellusten toimittamiseen asiakkaalle palveluna. Erityisesti runsaasti laskentatehoa vaativa tieteellinen laskenta voi hyötyä huomattavasti pilvipalveluiden mahdollisuuksista. Pilvipalveluita onkin hyödynnetty monissa tieteellisissä sovelluksissa ja kokeissa. Amazon Web Services on yksi suurimmista pilvipalveluntarjoajista, joka tarjoaa monipuolisia IaaS- ja PaaS-palveluita asiakkaiden käyttöön. Monissa tie-

teellisissä tutkimuksissa on käytetty erityisesti AWS:n EC2-virtuaalikoneita tieteellisten sovellusten suorittamiseen.

4 PERINNESOVELLUKSEN MODERNISOINTI PILVIPALVELUKSI

Tässä luvussa pyritään kirjallisuuskatsauksen avulla tunnistamaan niitä modernisoinnin menetelmiä, jotka ovat sovellettavissa tieteellisen laskentasovelluksen modernisointiin pilvipalveluksi. Pyrkimyksenä on myös selventää, mitä vaatimuksia ja rajoituksia pilvipalveluiden ja tieteellisen laskennan piirteet asettavat modernisoinnille. Aluksi käsitellään perinnesovelluksen modernisointia yleisellä tasolla. Toiseksi tarkastellaan modernisointia pilvipalveluksi yleisesti. Kolmanneksi käsitellään tieteellisen laskentasovelluksen modernisointia pilvipalveluksi. Luku päättyy yhteenvetoon.

4.1 Perinnesovelluksen modernisointi

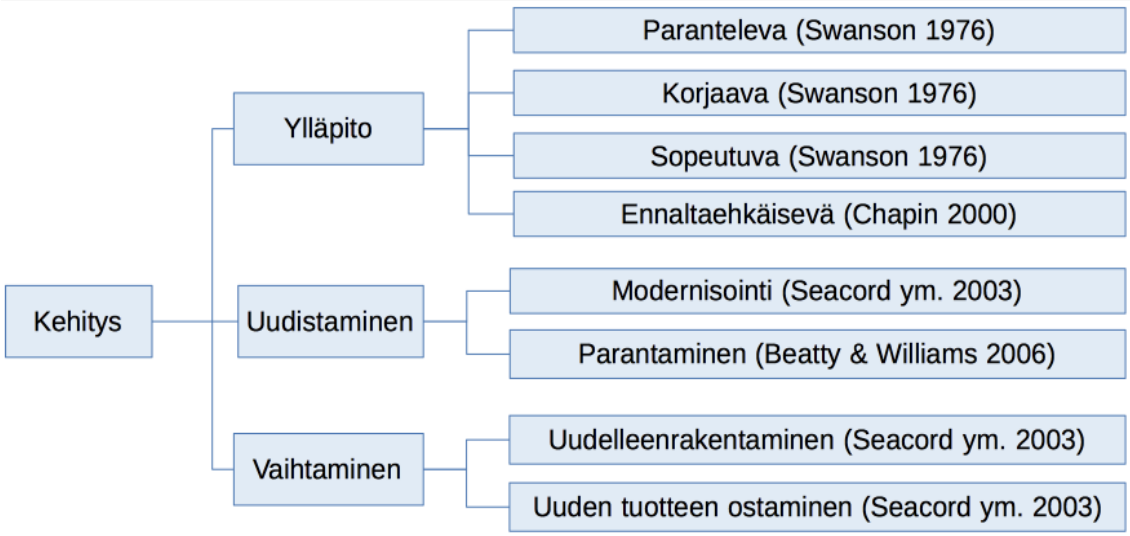
Tässä alaluvussa käsitellään perinnesovelluksen modernisointia Seacordin, Plakoshin ja Lewisin (2003) kirjan mukaisesti. Muita lähteitä käytetään täydentävästi. Alaluvussa keskitytään käsitteiden määrittelyyn sekä yleiskäyttöisten modernisointimenetelmien esittelyyn.

4.1.1 Määritelmä ja yleisiä piirteitä

Perinnesovellukselle (legacy application) ei ole olemassa yhtä yhtenäistä määritelmää, mutta eri määritelmistä (Alkazemi, Nour & Meelud, 2013; Ransom, Somerville & Warren, 1998; Seacord ym., 2003) on löydettävissä yhteneviä piirteitä. Näiden mukaisesti perinnesovellus on usein vanhentunut, mutta edelleen käytössä oleva sovellus, jonka ylläpito ja kehittäminen on muodostunut vaikeaksi, koska sovelluksen tekninen viitekehys on vanhentunut (Fanqi & Yunqi, 2013). Modernisoinnin yhteydessä perinnesovelluksella tarkoitetaan sovellusta, joka on modernisoinnin kohteena (Saarelainen ym., 2006).

Modernisoinnilla (modernization) tarkoitetaan toimintaa, jolla muutetaan sovellusta laajemmassa mittakaavassa säilyttäen kuitenkin siitä merkittävä osa (Seacord ym., 2003, s. 12). Modernisointi on sovelluksen uudistamista (renewal), jossa siihen tehdään muutoksia arkkitehtuurin tasolla esimerkiksi muuttamalla sovelluksen rakennetta tai uudistamalla lähdekoodia. Modernisaatiossa muu-

toksia tehdään laajemmin kuin järjestelmän ylläpidossa (maintenance), mutta suppeammin verrattuna koko järjestelmän vaihtamiseen (replacement) (Kankaanpää, 2011). Termi modernisaatio sijoittuu alan kirjallisuudessa IT-artefaktien evoluution alakäsitteiden joukkoon (kuvio 6).



KUVIO 6 Modernisoinnin sijoittuminen IT-artefaktien kehittymisen käsitteistöön kirjallisuudessa (Kankaanpää, 2011, s. 24)

Syitä modernisoinnille on monia. Tämän tutkielman yhteydessä jätetään liiketoimintaan liittyvien seikkojen painotus tarkoituksella hyvin vähäiseksi ja keskitytään enemmän muihin relevantteihin seikkoihin.

Sovellus alkaa vanheta heti julkistamisensa jälkeen (Seacord ym., 2003, s. 1). Sovelluksen vanhenemista vastaan taistellaan sovelluksen ylläpitovaiheessa. Sovelluksen ylläpidossa pyritään pienten muutosten avulla pitämään sovellus toimintakuntoisena ja ympäristön vaatimuksiin vastaavana (Kankaanpää, 2011, s. 24). Ylläpitotehtävät voidaan luokitella neljään kategoriaan (Kankaanpää, 2011, ss. 24–25; Seacord ym., 2003, s. 4):

1. *Parantelevassa ylläpidossa* (perfective maintenance) sovelluksen suorituskykyä ja käytettävyyttä kehitetään tai toteutetaan käyttäjälähtöisiä vaatimuksia.
2. *Korjaavassa ylläpidossa* (corrective maintenance) korjataan sovelluksessa ilmenneitä vikoja ja virheitä.
3. *Sopeutuvassa ylläpidossa* (adaptive maintenance) sovellusta muokataan vastaamaan muuttunutta ympäristöä.
4. *Ennaltaehkäisevässä ylläpidossa* (preventive maintenance) sovellukseen tehdään muutoksia, joiden avulla pyritään parantamaan sen luotettavuutta ja ylläpidettävyyttä, jotta voidaan mahdollisesti ennaltaehkäistä tulevaisuudessa ilmeneviä ongelmia.

Valtaosa ylläpidosta keskittyy parantelevaan ylläpitoon korjaavien, mukautuvien ja ennaltaehkäisevien toimien jäädessä selkeästi toissijaisiksi (Seacord ym., 2003, s. 4). Lehmanin (1980) toisen lain mukaisesti kumulatiiviset muutokset sovelluksen lähdekoodissa johtavat siihen, että koodista tulee vähemmän ylläpidettävää. Ylläpitotoimien kustannukset sekä lähdekoodin kasvava kompleksi-

suus ja huononeva ylläpidettävyys ovat pääsyitä sille, miksi sovelluksia päätetään modernisoida (Seacord ym., 2003, s. 5).

Modernisointiin liittyy monenlaisia haasteita. Seacordin ym. (2003, ss. 1–2) mukaan modernisoinnin monimutkaisuus on peräisin seuraavista seikoista:

- Erilaisia modernisointivaihtoehtoja (modernization options) on runsaasti, ja niistä jokainen voi sisältää merkittäviä kompromisseja (trade-offs).
- Perinnesovelluksen vaatimusten ja toteutuksen dokumentaatio voi olla hyvinkin puutteellista, jolloin näiden asioiden analysointiin tarvitaan paljon asiantuntemusta.
- Perinnesovelluksen täytäntöönpanossa (implementation) voi olla selvittämättömiä seikkoja, jotka voivat liittyä sovelluksen toiminnallisuuteen (functionality), eheyteen (integrity) ja laatumääreisiin (quality attributes).
- Modernisoinnin päätöksenteon tueksi on kerättävä kattava määrä kvantitatiivista ja kvalitatiivista dataa sovelluksesta.
- Modernisoinnin vaikutusten arviointi on haastavaa, koska vaikutuksia tulee arvioida kaikkien eri sidosryhmien (stakeholders) näkökulmasta.
- Modernisoinnin päätöksenteon pitää soveltua organisaation ja projektin päätöksentekomalliin ja noudattaa myös niiden muita mahdollisia rajoitteita.

4.1.2 Modernisointimenetelmiä

Modernisointi voidaan jakaa kahteen luokkaan sen mukaan, miten paljon tietoa ja ymmärrystä kohdesovelluksesta tarvitaan. *Lasilaatikkomodernisoinnissa* (White-Box modernization) tarvitaan tietämystä perinnesovelluksen sisäisestä toiminnasta. Mikäli sovelluksen lähdekoodi ei ole saatavilla, joudutaan sovelluksen sisäinen toiminta selvittämään *ohjelman ymmärtämisen* (program understanding) (Chikofsky & Cross II, 1990) menetelmin, jossa ympäristön mallintamisen ja muun ohjelmasta saatavan tiedon avulla pyritään päättelemään ohjelman sisäinen toiminta ja sovelluksen rakenne, mikä vaatii hyvin paljon työtä. Kun koodi on analysoitu ja ymmärretty, voidaan lähdekoodia ja sovelluksen rakennetta järjestellä uudelleen. *Uudelleenjärjestelyllä* (restructuring) tarkoitetaan sovelluksen muuttamista esitystavasta toiseen samalla suhteellisella abstraktiotasolla pysyen ja sovelluksen ulkoisen toiminnan säilyessä muuttumattomana (Chikofsky & Cross II, 1990, s. 15). Uudelleenjärjestelyllä pyritään lisäämään sovelluksen ylläpidettävyyttä ja suorituskykyä. (Seacord ym., 2003, s. 9.)

Musta laatikko -modernisoinnissa (Black-Box modernization) tarvitaan tietoa ainoastaan perinnesovelluksen ulkoisista rajapinnoista, mikä onnistuu sovelluksen syötteitä ja tulosteita tarkastelemalla. Musta laatikko -modernisointi ei ole yleensä niin haastavaa kuin lasilaatikkomodernisointi. Monesti musta laatikko -modernisointi perustuu *käärimiseen* (wrapping), jossa perinnesovelluksesta peräisin oleva koodinosa kääritään uudella koodilla siten, että uusi koodi hoitaa vanhan koodin kutsumisen ja syötteiden käsittelyn ja paljastaa itsestään ulospäin rajapinnan, joka on yhteensopiva uudistetun sovelluksen kanssa. Ideaalitilanteessa kehittäjän ei tarvitse tällöin tietää käärityn perinnesovelluksen osan sisäisestä toteutuksesta mitään, mutta käytännössä monesti käärimisenkin yhteydessä joudutaan turvautumaan lasilaatikkomodernisoinnin tekniikoihin. (Seacord ym., 2003, s. 9.)

Uudistustoimenpiteet (reengineering) ovat modernisoinnin muotoja, joissa perinnesovelluksen valmiuksia ja ylläpidettävyyttä parannetaan ottamalla käyttöön uusia teknologioita ja käytänteitä. Uudistustoimenpiteet ovat hallittuja lähestymistapoja, joissa perinnesovellus muutetaan systemaattisesti uuteen muotoon, jotta sen kehittyneet laatuvaatimukset täyttyisivät. Uudistustoimenpiteet eivät ole välttämättä kaikkein houkuttelevin vaihtoehto sovelluksen uudistamiselle, mutta usein ne ovat kaikkein käytännöllisimpiä. Uudistustoimenpiteillä jatketaan sovelluksen elinkaarta kustannustehokkaalla tavalla. Uudistustoimenpiteiden riskit ja kustannukset ovat pienemmät kuin *sovelluksen vaihtamisessa* (replacement), jossa sovellus vaihdetaan kokonaan uuteen, mutta kalliimmat kuin sovelluksen ylläpidossa. (Seacord ym., 2003.) Uudistustoimenpiteitä ovat seuraavat:

- uudelleenkohdistaminen (retargeting)
- parantelu (revamping)
- kaupallisten sovellusosien käyttö
- ohjelmointikielen vaihtaminen (source code translation)
- lähdekoodin vähentäminen (code reduction)
- toiminnallisuuden muuttaminen (functional transformation) (Seacord ym., 2003, ss. 10–11.)

Uudelleenkohdistamisella (retargeting) tarkoitetaan perinnesovelluksen migraatiota uudelle laitealustalle. Syynä migraatiolle voi olla esimerkiksi uuden tehokkaamman laitteiston houkuttelevuus tai vanhan laitteiston ylläpidon hankaluus. Uudelleenkohdistaminen yleensä vähentää ylläpitokuluja ja tarjoaa kehittyneemmän alustan tuleville modernisointitoimille. (Seacord ym., 2003, s. 11.)

Parantelu (revamping) on erityisesti sovelluksen pelkän käyttöliittymän vaihtamisesta usein käytetty termi. Eräs paljon käytetty parantelutekniikka on *screen scraping*, jossa yhden tai useamman perinnesovelluksen käyttöliittymän tulosteet kääritään uuteen käyttöliittymään. Käyttäjä näkee uudistetun käyttöliittymän, vaikka taustalla ajetaan perinnesovellusta. Sovelluksen ylläpitäjän toimia tällainen parantelu ei edesauta, vaan päinvastoin sovelluksen monimutkaisuus lisääntyy. Ylläpitäjän näkökulmasta screen scraping voi näyttää vertauskuvallisesti siltä, että ”raato on kuorrutettu kermavaahdolla”. (Seacord ym., 2003, ss. 11–12.)

Eräs mahdollinen uudistustoimenpide on *korvata* perinnesovelluksen osa *kaupallisella komponentilla*. Omistuksessa olevan toiminnallisuuden korvaaminen muualta ostetulla vaihtoehdolla voi tuntua järjenvastaiselta, mutta sillä voi olla myös selkeitä etuja. Ulkoisen tahon toimittaman komponentin käyttö vähentää ylläpidettävän koodin määrää. Joissain tilanteissa ulkoa hankitun kaupallisen komponentin käyttäminen on myös edullisempaa kuin perinnesovelluksen osan sovittaminen uuteen sovellukseen. Kaupalliset komponentit usein myös täyttävät alan standardit ja toteuttavat näiden standardien mukaiset rajapinnat, joten niitä on mahdollisesti helppo käyttää myös jatkossa. Toisaalta mikäli perustavanlaatuisia sovelluksen infrastruktuuriin liittyviä osia korvataan kaupallisilla komponenteilla, voi näiden komponenttien käytöstä luopuminen muodostua tulevaisuudessa haastavaksi. (Seacord ym., 2003, ss. 12–13.)

Ohjelmointikielen vaihtaminen kokonaan toiseen kieleen tulee kyseeseen esimerkiksi tilanteessa, jossa modernisoitavan sovelluksen alusta vaihdetaan uuteen eikä uusi alusta tue ohjelmointikieltä, jolla perinnesovellus on toteutettu. Usein perinnesovelluksen vanhentunut kieli halutaan vaihtaa modernimpaan.

Ohjelmointikielen vaihdoksessa käytetään apuna automatisoituja työkaluja, mutta niiden käyttö ei ole ongelmatonta, vaikka työkalut ovatkin kehittyneet vuosien aikana (Lidman, Quinlan, Liao & McKee, 2012; Noaje, Jaillet & Krajecki, 2011). Aina sovelluksen kieltä ei vaihdeta kokonaan toiseen kieleen vaan muutos saatetaan tehdä myös käytetyn kielen eri versioiden välillä. Lähdekoodin kielen vaihtamisen riski on sitä suurempi, mitä enemmän eroavaisuuksia alkuperäisen kielen ja kohdekielen välillä on. Vaihtamisen etuja myös monesti yliarvioidaan ja vastaavasti riskejä aliarvioidaan. (Seacord ym., 2003, ss. 13–14.)

Lähdekoodin vähentämisen (code reduction) konseptin voi rinnastaa asunnosta toiseen muuttamiseen: on helpompi luopua ylimääräisistä tavaroista ennen muuttoa kuin kantaa kaikki omaisuus uuteen paikkaan ja sitten vasta heittää tarpeettomat tavarat pois. Sovelluksen tapauksessa tarpeeton lähdekoodi pyritään poistamaan ennen kuin perinnesovellusta ryhdytään modernisoimaan uudelle alustalle. Tarpeettomat koodinosat voidaan tunnistaa selvittämällä niitä vastaavat sovelluksen toiminnallisuudet, jotka eivät ole enää tarpeellisia. Koodin osia poistettaessa täytyy varmistaa, ettei poistettava osa sisällä piiloriippuvuuksia (hidden dependencies) muihin sovelluksen osiin, mikä voisi aiheuttaa sovelluksen rikkoutumisen, kun tarpeettomaksi luultu osa poistetaan. Tarpeettomien koodin osien tunnistamiseen ja poistamiseen on olemassa monia automatisoituja työkaluja. Ohjelmoijat ja tuotepäälliköt kuitenkin jättävät sovelluksiin usein tarpeettomia koodin osia, koska he ajattelevat niitä mahdollisesti tarvittavan tulevaisuudessa, ja kerryttävät näin projektin painolastia tehden lähdekoodista kumuloituvasti kasvavaa sekä vähemmän hallittavaa (Bergman, 2012). (Seacord ym., 2003, ss. 14–15.)

Toiminnallisuuden muuttaminen (functional transformation) voidaan jakaa kolmeen osa-alueeseen (Seacord ym., 2003, ss. 15–16):

- *Rakenteellisessa parantamisessa* (structure improvement) pyritään tunnistamaan ja korjaamaan sovelluksessa esiintyviä rakenteellisia puutteita ja virheitä. Korjaaminen tapahtuu käytännössä siten, että virheellinen rakenne korvataan paremmalla rakenteella joko automatisoidusti tai manuaalisesti.
- *Modularisoinnissa* (modularization) pyritään koostamaan toisiinsa liittyvistä sovelluksen osista loogisia ja uudelleenkäytettäviä kokonaisuuksia. Modularisointi helpottaa tarpeettoman koodin tunnistamista ja selkeyttää eri komponenttien välisiä rajapintoja. Monesti ohjelman modernisointi aloitetaan koodin modularisoinnilla, sillä hyvin määriteltyjen moduulien korvaaminen uusilla on suhteellisen helppoa.
- *Datan uudistustoimenpiteillä* (data reengineering) tarkoitetaan perinnesovelluksen prosessoiman tiedon varastoinnin, järjestyksen ja formaatin muokkaamista. Datan uudistamistoimenpiteet ovat usein tarpeen datan mahdollisen rapautumisen (degradation) vuoksi. Rapautuminen voi johtua muun muassa tiedon epäyhtenäisestä tallentamisesta tai datan sirpaloitumisesta useaan eri paikkaan.

4.1.3 Yhteenveto

Perinnesovelluksen modernisointiin on olemassa monenlaisia keinoja, jotka tosin sisältävät usein vähemmän yksiselitteisiä kompromisseja. Modernisoinnin päätöksenteossa tulee puntaroida näitä kompromisseja sekä teknisestä että or-

ganisaationaalisesta näkökulmasta. Modernisoinnin tulee olla suunniteltua ja hallittua, mutta modernisoinnin tulee myös olla tasapainossa siihen käytettävän työmäärän ja käytettävissä olevien resurssien kanssa.

4.2 Modernisointi pilvipalveluksi

Tässä alaluvussa käsitellään perinnesovelluksen modernisointia pilvipalveluksi. Aluksi määritellään modernisointi pilvipalveluksi käsitteenä ja selvitetään modernisoinnin motiiveja ja taustaa. Toiseksi esitellään kirjallisuudessa esitettyjä yleisiä modernisointimenetelmiä. Kolmanneksi tarkastellaan kolmea menetelmää perinnesovelluksen modernisoimiseksi pilvipalveluksi.

4.2.1 Määritelmä ja taustaa

Sovelluksen *modernisointi pilvipalveluksi* tarkoittaa sitä, että modernisointiprosessin seurauksena käyttäjä voi käyttää sovellusta suoraan pilvessä internetselaimen avulla. Käytännössä tämä tekee sovelluksesta alustariippumattoman ja loppukäyttäjälle erittäin helposti saavutettavan. Käyttäjä tarvitsee vain internetyhteyden käyttääkseen sovellusta pilvessä. (Mell & Grance, 2009.) Perinnesovelluksen modernisointi pilvipalveluksi pitää sisällään monenlaisia haasteita, joita syntyy, kun sovellus pyritään muuttamaan pilvipalvelumalliin sopivaksi (Bergmayr ym., 2013). Modernisointimenetelmiä on useita, ja oikeanlaisen menetelmän valitseminen on oleellista modernisoinnin onnistumisen kannalta (Comella-Dorda ym., 2000).

Motiiveja sovelluksen modernisoinnille pilvipalveluksi voi olla monia. Keskeisimpänä hyötynä sovelluksen ylläpidon kannalta pidetään yleensä mahdollisuutta päästä hyötymään IaaS-pilvipalveluiden eduista, kuten skaalautuvuudesta, elastisuudesta ja edullisesta käyttöön perustuvasta hinnoittelusta (Chauhan & Babar, 2012). Sovelluksen modernisointi SaaS-pilvipalveluksi avaa puolestaan uusia mahdollisuuksia sovelluksen hyödyntämiselle. Pilvipalveluna sovellus on saavutettavissa internetin välityksellä mistä tahansa pelkän internetselaimen avulla, joten sovelluksen potentiaalinen käyttäjäkunta kasvaa ja sovellus on mahdollista yhdistää muihin sovelluksiin ja palveluihin.

Perinnesovellukset ovat hyvin harvoin valmiita siirrettäviksi pilveen sellaisenaan, vaan niiden teknistä toteutusta ja liiketoimintamallia täytyy monesti muokata (Menychtas ym., 2013). Yleisin kirjallisuudessa esiintyvä lähestymistapa sovelluksen modernisoimiseksi pilvipalveluksi on muokata sovelluksen arkkitehtuuri palvelukeskeiseksi tavalla tai toisella (Chauhan & Babar, 2012).

Palvelukeskeinen arkkitehtuuri (service-oriented architecture) on ohjelmistotekniikan suunnittelumalli, joka perustuu löyhästi kytköksissä olevien (loosely coupled), joustavien, uudelleenkäytettävien ja yhteensopivien (interoperable) palveluiden (services) käyttöön (Waris, Khan & Fakhar, 2013). Pilvipalvelut perustuvat suurelta osin palvelukeskeisen arkkitehtuurin käyttöön (Chauhan & Babar, 2012).

Palvelut ovat yleensä verkon kautta käytettäviä itsenäisiä ohjelmistokomponentteja, jotka suorittavat usein liiketoimintaprosesseja abstrahoiden sisäänsä niiden sisäiseen toteutukseen vaaditun kompleksisuuden (vertaa musta laatikko) (Nurhasan, Dabarsyah & Fakhrurroja, 2013). Palveluiden kanssa kommuni-

koidaan viestien (messages) välityksellä, ja palvelun tulisi pystyä toimittamaan itsestään koneluettava (usein XML- tai JSON-muotoinen) metadata, joka antaa julkisen kuvauksen kyseisen palvelun semantiikasta ja toiminnasta (Nurhasan ym., 2013). Palveluita voidaan käyttää yksittäin tai niistä voidaan yhdistellä erilaisia *koostepalveluita* (mashup services), joissa kahden tai useamman sovelluksen tietoja ja toimintoja yhdistäen muodostetaan uusi sovellus (Su, Cheng, Wu & Li, 2011).

Erilaisia menetelmiä sovelluksen arkkitehtuurin muokkaamiseksi palvelukeskeiseksi ovat esittäneet muun muassa Razavian ja Lago (2010), Cetin, Ilker Altintas, Oguztuzun, Dogru, Tufekci ja Suloglu (2007) sekä Lewis, Morris ja Smith (2005). Razavian ja Lago (2010) esittelevät viitekehyksen, jossa perinnesovelluksesta laaditaan abstrakti malli, joka transformoidaan palvelukeskeiseksi abstraktioksi (service-oriented abstraction). Vaatimukset modernisoinnin tuloksena syntyvälle sovellukselle laaditaan tämän abstraktion perusteella, ja lopulta sovellus muokataan näihin vaatimuksiin sopivaksi. Kirjoittajat esittävät myös vaihtoehtoisen mallin, jossa koko sovellus kääritään palveluksi ilman sen hajottamista osiin.

Cetin ym. (2007) esittävät sovelluksen muokkaamiseksi palveluiksi kuusikohtaisen polun, josta he käyttävät nimeä MASHUP. Kuusi kohtaa ovat:

- vaatimusten mallintaminen
- perinnesovelluksen analysointi
- palveluiden tunnistaminen kartoittamalla vaatimuksia vastaavat komponentit
- vaatimukset täyttävien komponenttien laatiminen
- palveluiden määritys
- palveluiden toteutus ja täytäntöönpano.

Sovelluksen saattamiseksi palvelukeskeiseksi myös Lewis ym. (2005) ovat esittäneet oman lähestymistapansa, jonka mukaan migraation keskeiset toiminnot ovat:

- sovelluksen sidosryhmien tunnistaminen
- komponenttien tunnistaminen sen ja arviointi, miten hyvin ne soveltuvat palvelukeskeiseen arkkitehtuuriin
- mahdollisten migraatio-ongelmien tunnistaminen.

Edellä esitellyt menetelmät antavat ohjeita ja viitteitä siitä, miten perinnesovelluksesta modernisoidaan pilvipalvelumalliin soveltuva palvelusuuntautunut sovellus, mutta nämä menetelmät eivät ota kantaa pilvipalveluiden erityisiin piirteisiin modernisoinnin kohdealustana. Erityisesti palveluntarjoajan valinta on seikka, joka vaatii lisää selvittämistä pilvimigraatiota varten. (Chauhan & Babar, 2012.)

4.2.2 Chauhanin ja Babarin menetelmä

Chauhan ja Babar (2012) esittävät sovelluksen pilvimigraatioprosessille oman mallinsa, joka laajentaa edellä esiteltyjä palvelusuuntautuneita arkkitehtuureja

koskevia menetelmiä huomioiden pilvipalveluiden toimintamallin ja piirteet. Prosessi koostuu seitsemästä vaiheesta.

Ensimmäinen vaihe on *vaatimusten tunnistaminen* (requirements identification), jossa tunnistetaan vaatimukset, jotka laittavat migraatioprosessin alulle. Vaiheen tavoitteena on selkeyttää migraation motivaatiot ja tavoitteet, ja sen tuloksena syntyy joukko vaatimuksia (requirements).

Toisessa vaiheessa pyritään tunnistamaan vaatimusten pohjalta *potentiaaliset pilvipalveluntarjoajat*. Tunnistamisessa huomioitavia seikkoja ovat muun muassa projektin luonne, datan luottamuksellisuus, budjettirajoitteet sekä organisaation pitkän tähtäimen tavoitteet ja mahdolliset olemassa olevat asiakassuhteet eri palveluntarjoajiin. Toisen vaiheen tuloksena syntyy lista potentiaalisista palveluntarjoajista keskeisine ominaisuuksineen.

Kolmannessa vaiheessa *analysoidaan sovelluksen yhteensopivuutta* (compatibility) potentiaalisten palveluntarjoajien pilviympäristöjen kanssa. Pyrkimyksenä on selvittää, mitä muutoksia sovellukseen tulisi tehdä, jotta siitä saataisiin yhteensopiva palveluntarjoajan alustan kanssa. Kolmannen vaiheen tuloksena on selvitys, mitä kompromisseja (trade-offs) joudutaan mahdollisesti tekemään ja mitä etuja kunkin palveluntarjoajan valitseminen aiheuttaisi.

Neljännessä vaiheessa *tunnistetaan sovelluksen mahdolliset arkkitehtuurivaihtoehdot*. Eri arkkitehtuurivaihtoehdoista pyritään valitsemaan sellainen, joka täyttää kaikki toiminnalliset vaatimukset ja laatuvaatimukset. Mikäli tällaista vaihtoehtoa ei löydy, joudutaan tekemään kompromissi ja valitsemaan eniten tyydyttävä ja vähiten ongelmallinen arkkitehtuuri. Vaiheen tuloksena syntyy korkean tason suunnitteludokumentti järjestelmän arkkitehtuurista.

Viidennessä vaiheessa *arvioidaan pilviympäristöjä* kunkin ympäristön erityisten ominaisuuksien perusteella. Tarkasteltavia attribuutteja ovat muun muassa kyseisen ympäristön tukemat ohjelmointikielet, kehitysympäristöt sekä palvelun yhteensopivuus muiden palveluiden kanssa. Parhaiten laatuvaatimukset täyttävä palveluntarjoaja valitaan toteutusvaihetta varten. Vaiheen tuloksena syntyy artefakteja, joista ilmenee, kuinka hyvin kunkin potentiaalisen pilvivaihtoehdon erityiset ominaisuudet vastaavat määritellyn arkkitehtuurin laatuvaatimuksia.

Kuudennessa vaiheessa *arvioidaan komponenttitasolla, miten valittuun arkkitehtuuriin muokattu sovellus käytännössä pystytettäisiin valitun palveluntarjoajan pilveen*. Pyrkimyksenä on tunnistaa mahdollisia konflikteja valitun arkkitehtuurin ja pilviympäristön välillä. Mahdollisten ristiriitojen ilmetessä voidaan toistaa vaiheet 4 ja 5. Vaiheen tuloksena syntyy artefakteja, jotka kuvaavat kuinka hyvin yhteensopiva valittu ratkaisu on valitun pilviympäristön kanssa komponenttitasolla.

Viimeinen vaihe on *toteutus* (implementation), jossa tähdätään määritellyn ratkaisun toteuttamiseen joko luomisen tai olemassa olevan sovelluksen refaktoroinnin avulla. Vaiheen tuloksena on sovellus, joka on valmis pystytettäväksi valitulle pilvialustalle.

4.2.3 ARTIST

Alonso, Orue-Echevarria, Escalante, Gorronogoitia ja Presenza (2013) ovat myös esittäneet oman ARTIST-nimisen menetelmänsä perinnesovelluksen modernisoimiseksi pilvipalveluksi. Tekijät väittävät sen kattavan sekä teknologisen että liiketoiminnallisen näkökulman ja soveltuvan geneerisyytensä vuoksi monenlaisiin tapauksiin tarjoten tarvittavat menetelmät ja työkalut modernisoinnin ja

migraation viemiseen alusta loppuun saakka. Menetelmä perustuu *malliperustaiseen lähestymistapaan* (model-driven approach), jossa sovelluskehityksen pääasiallisena kehityksen ja ylläpidon kohteena ovat sovelluksesta määriteltävät formaalit mallit lähdekoodin jäädessä toissijaiseksi tuotokseksi, joka voidaan suurilta osin generoida hyvin määriteltyjen mallien ja metamallien pohjalta (Elleuch, Khalfallah & Ben Ahmed, 2007). ARTIST-menetelmä jakautuu kolmeen peräkkäin suoritettavaan vaiheeseen. (Alonso, Orue-Echevarria, Escalante, Gorronogoitia & Presenza, 2013; Bergmayr ym., 2013; Menychtas ym., 2013.)

Menetelmässä prosessi aloitetaan migraatiota *edeltävällä vaiheella* (pre-migration phase), jossa tutkitaan kattavasti perinnesovelluksen soveltumista modernisointiin pilvipalveluksi sekä teknologisesta että liiketoiminnan näkökulmasta. Vaiheessa pyritään määrittämään motivaatio migraation toteuttamiselle sekä tunnistamaan modernisointiin ja pilveen migraatioon vaikuttavat tekniset ja kaupalliset tekijät huomioiden myös erityisesti pilvipalveluiden ja -palveluntarjoajien erityispiirteet. Vaiheen tuloksena syntyy päätös siitä, toteutetaanko migraatio vai ei. (Bergmayr ym., 2013; Menychtas ym., 2013.)

Menetelmän toinen vaihe on *varsinainen migraatio* (migration). Migraatioprosessi räätälöidään ensimmäisessä vaiheessa selvitettyjen teknologisten ja liiketoiminnallisten seikkojen perusteella sopivaksi kyseiseen tapaukseen. Modernisointi toteutetaan määrittämällä takaisinmallinnuksen (reverse engineering) avulla mallit, jotka kuvaavat perinnesovelluksen toimintalogiikan. Kun mallit on alustavasti määritetty, suoritetaan niille *mallien ymmärrys* (model understanding) -prosessi, jossa malleista pyritään erilaisten transformaatioiden avulla suodattamaan epäoleellinen informaatio pois, jotta jäljelle jäisi ytimekäs kuvaus perinnesovelluksen taustalla olevista prosesseista. Sovelluksen kohdeympäristö määritetään tutkimalla määritettyjä malleja ja sovittamalla niiden mukaiset vaatimukset tarjolla oleviin IaaS-, PaaS- ja SaaS-vaihtoehtoihin. Seuraavaksi mallinnetut piirteet transformoidaan määritetylle kohdeympäristölle sopivaan muotoon, jonka jälkeen näitä malleja käyttäen voidaan generoida sovelluksen lähdekoodin runko, josta ohjelmoijat täydentävät toimivan sovelluksen. (Menychtas ym., 2013.)

Viimeisessä eli *migraation jälkeisessä vaiheessa* (post-migration phase) modernisoitu sovellus validoidaan ja sertifioidaan. Validoinnilla tarkoitetaan tässä yhteydessä sitä, että modernisoitu sovellus vastaa toiminnallisuudeltaan perinnesovellusta. Laajan sovelluksen tapauksessa tämän varmistaminen on tosin osoittautunut hyvin haastavaksi. Sertifioinnilla tarkoitetaan modernisoidun sovelluksen laadukkuuden varmistamista, mikä suoritetaan testaamalla. (Menychtas ym., 2013.)

4.2.4 REMICS

REMICS on myös mallivetoinen palveluarkkitehtuuriin perustuva menetelmä ja monilta osin hyvin saman kaltainen kuin ARTIST. REMICS koostuu kuudesta keskeisestä vaiheesta ja niitä tukevista työkaluista. (Mohagheghi, Berre, Henry, Barbier, & Sadovykh, 2010.)

Ensimmäisessä vaiheessa määritetään tavoitesovelluksen vaatimukset, pyritään tunnistamaan sovelluksen keskeiset komponentit ja laaditaan niille implementointistrategia. Vaatimusten pääpaino asetetaan niihin osa-alueisiin, jotka vaativat kehittämistä, jotta sovellus voitaisiin viedä pilveen. Perinnesovelluksesta pyritään tunnistamaan ne komponentit, jotka soveltuvat modernisoitaviksi tavoitesovellukseen. (Mohagheghi ym., 2010.)

Toisessa vaiheessa pyritään keräämään tietoa niistä perinnesovelluksen komponenteista, jotka on määritetty mahdollisesti modernisoitaviksi. Työkalujen avulla luodaan malli perinnesovelluksesta ja saadaan mahdollisesti lisätietoa vaatimuksista. (Mohagheghi ym., 2010.)

Kolmas vaihe on varsinainen migraatio, jossa uusi sovellus määritellään ja toteutetaan aiemmin tunnistettujen elementtien perusteella. Myös tarvittavat täysin uudet komponentit määritellään, jotta mahdolliset uudet vaatimukset ja palveluperusteinen arkkitehtuuri saadaan toteutettua. Laskutukseen, monitorointiin ja turvallisuuteen liittyvät komponentit määritellään REMICS-lähestymistavassa itsenäisiksi komponenteiksi, jotka integroidaan lopulliseen ratkaisuun. Tämän pyrkimyksenä on tuottaa enemmän uudelleenkäytettäviä komponentteja. Myös pilvienvälisen ohjelmointirajapinnan laatimista suositellaan yhteensopivuuden lisäämiseksi. (Mohagheghi ym., 2010.)

Neljäs vaihe on validointi, jossa määritellään sovelluksen testausstrategia, jonka tavoitteena on varmistaa, että toteutettu järjestelmä täyttää kaikki määritellyt vaatimukset ja kaikki toiminnot toimivat oikein. Toiminnallisten vaatimusten validoinnin lisäksi tässä vaiheessa suoritetaan ei-toiminnallisten vaatimusten, kuten tehokkuuden, luotettavuuden ja turvallisuuden, tarkistaminen. (Mohagheghi ym., 2010.)

Viidennessä vaiheessa tuotetaan työkalut, joilla voidaan hallita ja muokata sovelluksen suorituskykyä sen jälkeen kun sovellus on julkaistu palveluna. (Mohagheghi ym., 2010.) *Kuudennessa vaiheessa* tuotetaan työkaluja, jotka ratkaisevat mahdollisia yhteensopivuusongelmia muiden palveluntarjoajien, kolmansien osapuolien tuottamien komponenttien ja muiden palvelujen kanssa.

Viimeinen vaihe on *vetäytyminen* (withdrawal), jossa luodaan työkaluja palvelun pysäyttämiseen ja toiseen pilvipalveluntarjoajaan tai infrastruktuuriin siirtymiseen. (Mohagheghi ym., 2010.)

ARTIST-menetelmän laatijoiden mukaan REMICS-menetelmän merkittävimmät puutteet ovat SaaS-palvelumalliin liittyvien ei-toiminnallisten vaatimusten käsittelyssä ja niihin liittyvien sovellusarkkitehtonisten ratkaisujen esittämisessä sekä siinä, että laskutukseen ja turvallisuuteen liittyvät tekijät jätetään pilvipalveluntarjoajan vastuulle, eikä niitä käsitellä kiinteänä osana sovellusta. (Menychtas ym., 2013.)

4.3 Tieteellisen laskentasovelluksen modernisointi pilvipalveluksi

Motiivit tieteellisen laskentasovelluksen modernisointiin pilvipalveluksi ovat ilmeiset. Tieteelliset laskentasovellukset tarvitsevat monesti paljon laskentatehoa ja tallennuskapasiteettia. Pilvipalveluiden vaivaton käyttöönotto, skaalautuvat resurssit sekä palvelun käyttöön perustuva hinnoittelu tarjoavat ihanteelliselta vaikuttavan ratkaisun tieteellisen laskennan vaatimuksille (Vecchiola ym., 2009). Sekä tieteellisen laskennan suorittamisesta pilvessä (Bunch, Drawert & Norman, 2009; Satish Narayana Srirama, Batrashev, Jakovits & Vainikko, 2011; S.N. Srirama ym., 2013; Vecchiola ym., 2009) että sovellusten modernisoinnista pilveen (Alonso ym., 2013; Bergmayr ym., 2013; Chauhan & Babar, 2012; Cretella & Martino, 2014; Menychtas ym., 2013; Mohagheghi ym., 2010) on runsaasti kirjallisuutta, mutta tieteellisen laskennan erityispiirteitä sovellusten modernisoinnissa pilvipalveluksi on huomioitu kirjallisuudessa yllättävän vähän.

Merkittävä mutta vähäiselle huomiolle jäänyt tosiasia on se, että tieteentekijöillä ei aina ole määrättömästi resursseja tai osaamista vaativien pilvikonfiguraatioiden ja migraatioprosessien hallintaan (Mendez, Villamiazr & Castro, 2013; Srirama ym., 2013). Esimerkiksi yliopistoissa tutkijoilla on monesti käytettävissään raskaaseen laskentaan sopivia laskentaklustereita, mutta kilpailu niiden käyttöajasta sekä konfiguroinnin haasteet ovat saaneet tutkijat kiinnostumaan kaupallisista palveluntarjoajista (Tsaftaris, 2014). Valitettavasti tässäkin tutkielmassa esitetyt menetelmät perinnesovelluksen modernisoimiseksi pilvipalveluksi ovat niin raskaita ja monimutkaisia, että niiden käyttäminen edellyttäisi ammattitaitoa ja resursseja, joita tieteentekijöillä harvoin on käytettävissään.

Srirama ym. (2013) ovat kehittäneet D2CM-työkalun, jonka tehtävä on helpottaa tieteentekijöitä, joilla ei ole kattavaa osaamista pilvipalveluista ja migraatioprosesseista, suorittamaan tieteelliset kokeensa pilvipalveluita hyödyntäen. Työkalun avulla tutkijat pystyvät määrittämään paikallisilla palvelimilla ajamansa työnkulut virtuaalikoneille, jotka konfiguroidaan automaattisesti yhteensopiviksi Amazonin EC2-virtuaalikonekuvien kanssa. D2CM hallitsee myös näiden virtuaalikoneiden elinkaaren ja resurssien allokoinnin, joten tutkijoiden ei käytännössä tarvitse tietää migraatioiden tai pilvipalveluiden teknisestä toteutuksesta mitään. D2CM helpottaa tutkijoita suorittamaan tieteelliset kokeensa pilvipalveluita käyttäen, mutta koska palveluarkkitehtuuria ei toteuteta itse sovellukseen, pilvipalveluiden kaikkia etuja ei päästä hyödyntämään. D2CM:ää käyttämällä tieteellinen laskentasovellus ei ole muiden tutkijoiden saatavilla helposti, eikä sovellusta voi helposti yhdistää muihin sovelluksiin verkon kautta.

Biro, Bacu, Rodila, Barabas ja Gorgas (2012) ovat käsitelleet hajautettuun laskentaan (grid computing) perustuvan laskentasovelluksen modernisointia pilveen. He esittelevät lähestymistavan, joka muistuttaa vaiheiltaan hyvin paljon ARTIST- ja REMICS-menetelmien (Menychtas ym., 2013; Mohagheghi ym., 2010) vaiheita lisäten niihin ratkaisuehdotuksia hajautetun laskennan toteuttamiseksi eräässä pilviympäristössä. Itse modernisointiin artikkelissa otetaan hyvin vähän kantaa, eikä heidän ratkaisunsa myöskään toteuta SaaS-mallia.

Tsaftaris (2014) esittelee useita tieteentekijöille suunnattuja pilvipalveluita, kuten StarCluster, OpenCPU, CPUsage ja PiCloud, jotka tarjoavat laskentakapasiteettia ja infrastruktuuria raskaan laskennan suorittamiseen. Palvelut kärsivät ominaisuuksien rajoittuneisuudesta ja erityisesti siitä puutteesta, ettei niissä ole ainakaan toistaiseksi mahdollista julkaista omia laskentasovelluksia SaaS-palveluna. Grid-Cloud (P. Huang, Lin & Peng, 2010), VLAB-C (Yu & Dong, 2014), E-Clouds (Mendez, Villamiazr & Castro, 2013) ja Duckling (Yu, Dong & Nan, 2012) ovat tutkimuspapereissa esitettyjä ja lupaavalta vaikuttavia tieteellisille laskentasovelluksille suunnattuja palvelualustoja, joissa laskentasovelluksille voidaan toteuttaa myös täysi SaaS-malli. Niiden heikkous on siinä, että niitä ei ole ainakaan vielä toteutettu oikeasti. Tulevaisuutta ajatellen tämän kaltaiset ”Science as a Service” -palvelut vaikuttavat lupaavilta, mutta toistaiseksi ne eivät keskeneräisyytensä tai rajoittuneisuutensa vuoksi ole järin käytännöllisiä vaihtoehtoja (Cook, Milojicic, Kaufmann, & Sevinsky, 2012; Mendez ym., 2013).

4.4 Yhteenveto

Palvelukeskeisyys on pilvisovellusten ja -palveluiden keskeinen piirre. Mikäli perinnesovellus aiotaan modernisoida pilvipalveluksi, täytyy sen arkkitehtuuri muokata palvelukeskeiseksi. Palvelukeskeisyyttä on käsitelty kirjallisuudessa runsaasti, ja sovellusten arkkitehtuurin muokkaaminen palvelukeskeiseksi on laajalti käsitelty aihe. Pilvipalveluissa on kuitenkin omat erityiset piirteensä, jotka täytyy huomioida modernisoinnissa. Kirjallisuudessa on esitetty perinnesovelluksen modernisoimiseksi pilvipalveluksi menetelmiä, jotka pääasiassa laajentavat palvelukeskeisyyteen tähtääviä modernisointimenetelmiä lisäämällä niihin teknisiä ja liiketoiminnallisia seikkoja, jotka ovat ominaisia pilvipalveluille.

Tieteellisten laskentasovellusten tapauksessa modernisoinnista pilvipalveluksi on löydettävissä yllättävän vähän kirjallisuutta. Tieteentekijöillä on harvoin vaadittavaa IT-osaamista haastavan pilvimigraation suorittamiseen, eikä tutkimuksen resursseja haluttaisi käyttää suureen järjestelmäprojektiin, joita tässäkin tutkielmassa esitetyt mallit usein vaikuttavat edellyttävän. Tieteentekijöitä voisi mahdollisesti hyödyttää kevyt menetelmä, jonka avulla tutkijat pääsisivät mahdollisimman helposti hyötymään sekä pilvipalveluiden että IaaS- ja SaaS-mallien eduista.

5 TAPAUSTUTKIMUKSEN TOTEUTTAMINEN

Tutkimuksen empiirisen osuuden tarkoituksena on tunnistaa sellaiset sovellusten modernisointimenetelmät ja -käytänteet, jotka tukevat niiden tutkijoiden tavoitteita, jotka haluavat vaivattoman, skaalautuvan ja helposti hallittavan ajoympäristön tieteellisille laskentasovelluksilleen ilman valtavia IT-investointeja. Menetelmien ja käytäntöjen arviointi ja tunnistaminen tapahtuu käytännössä modernisoimalla eräs tieteellinen laskentasovellus pilvipalveluksi ja reflektoimalla tämän modernisointiprosessin haasteita kirjallisuudessa esitettyihin menetelmiin.

5.1 Tutkimuskohde

Tutkimuksen kohteena on eräs avaruusfysiikan alan tieteellinen laskentasovellus, jossa tutkitaan numeeristen simulaatioiden avulla plasman käyttäytymistä sen kohdatessa erilaisia fysikaalisia objekteja. Sovellusta käytetään erityisesti tutkimaan aurinkotuulen vaikutuksia aurinkokunnan objekteihin. Sovelluksen pääasiallisia käyttäjiä ovat kyseisen alan tutkijat. Simulaatioiden ajamisen lisäksi sovellus sisältää internetin kautta käytettäviä työkaluja simulaatioiden tuottaman datan lataamiseen ja visualisointiin kyselyjen perusteella. Osa sovelluksen kyselytoiminnallisuuksista on toteutettu palveluarkkitehtuuria käyttäen, jotta data olisi tutkijoiden yhteistyöverkoston saavutettavissa.

Sovelluksen ylläpitäjät ovat kokeneet sovelluksen ylläpidon monella tapaa ongelmalliseksi. Simulaatioiden ajaminen tapahtuu manuaalisesti sovelluksen ylläpitäjien toimesta, ja simulaatioiden tuottaman datan tallentaminen järjestelmään vaatii myös ylläpidollisia toimia. Sovelluksen arkkitehtuuri on koettu epämääräiseksi, ja lähdekoodia on kuvailtu ”spagettimaiseksi”, mikä on johtanut siihen, että sovellukseen on haastavaa luoda paljon kaivattuja ylläpitoa helpottavia toiminnallisuuksia. Simulaatiot tuottavat valtavia määriä dataa, mikä aiheuttaa omat ylläpito-ongelmansa paikallisilla palvelimilla, joissa tallennuskapasiteetti on rajallinen. Lisäksi simulaatioiden ajaminen edellyttää sopivan laitteiston resursoinnin ja sille sovellusympäristön konfiguroimisen ylläpidon toimesta. Kasvavat datamäärät ja monien toimintojen manuaalisuus aiheuttavat tarpeettomia henkilöstö- ja laitteistokustannuksia. Myös sovelluksen yhteensopivuutta erityisesti yhteistyöverkostoon kuuluvien sovellusten kanssa halutaan

kasvattaa tulevaisuuden visioista johtuen. Tarkemmin kohdesovelluksen modernisointipäätökseen vaikuttaneita tekijöitä on eritelty luvussa 6.

Tutkimuksen kohteena olevalla sovelluksella tehdään tieteellistä tutkimusta. Tutkimuksen kannalta olisi tärkeää, että sovelluksella saavutetut tulokset olisivat helposti myös muiden tutkijoiden verifioitavissa. Olemassa olevan sovelluksen tapauksessa tämä on periaatteessa mahdollista, mutta se teettäisi käytännössä valtavasti ylimääräistä työtä, joten tutkijat eivät ilman erityisen painavaa syytä toista kokeita kovin mielellään.

5.2 Tutkimusmenetelmä

Tutkimuksen empiirinen osuus toteutetaan konstruktiivisena tapaustutkimuksena, jossa tiedonkeruumenetelminä toimivat havainnointi ja haastattelut.

Tapaustutkimuksen tarkoituksena on ymmärtää tutkittavaa ilmiötä syvällisemmin ja parantaa sekä kehittää tiettyjä piirteitä tutkittavasta ilmiöstä (Runeson & Höst, 2009). Tapaustutkimus soveltuu tilanteisiin, joissa tutkitaan ajankohtaisia ilmiöitä ja pyritään saavuttamaan yksityiskohtaista tietoa ja tarkempaa ymmärrystä tutkimuskohteesta (Järvinen & Järvinen, 2011). Tapaustutkimuksessa keskeistä on ilmiön tapahtumaympäristö, ja aineisto kerätäänkin luonnollisissa tilanteissa tutkijan ollessa vuorovaikutuksessa tutkimuskohteen kanssa (Soininen, 1995). Tapaustutkimuksen heikkoudeksi voidaan katsoa kapea-alaisuus, koska sen tulokset eivät välttämättä ole yleistettäviä. Tapaustutkimuksen kohde valitaan usein sen kiinnostavuuden vuoksi, jolloin tapausta ei voida välttämättä pitää tyypillisenä koko populaatiota ajatellen, ja tutkijan subjektiivinen panos saattaa tällöin myös vaikuttaa tutkimustulosten arviointiin herkemmin (Anttila, 2000). Tapaustutkimukset voidaan jaotella Runesonin ja Höstin (2009) mukaan neljään luokkaan sen mukaan, mihin tutkimuksella pyritään:

- *Tutkivassa* (exploratory) tutkimustyypissä pyritään luomaan uusia ajatuksia ja hypoteeseja tulevaa tutkimusta varten.
- *Kuvaavassa* (descriptive) tutkimustyypissä pyritään kuvaamaan jokin ilmiö tai tilanne.
- *Selittävässä* (explanatory) tutkimustyypissä etsitään (usein kausaalisia) selityksiä tilanteelle tai ongelmalle.
- *Parantavassa* (improving) tutkimustyypissä pyritään parantamaan jotain tutkittavan ilmiön tai kohteen tiettyä aspektia. (Runeson & Höst, 2009.)

Konstruktiivinen tutkimusote on eräs tapa suorittaa tapaustutkimusta. Konstruktiivisen tutkimusotteen avulla pyritään ratkaisemaan reaalimaailman ongelmia. Tutkimusotteen ydinkäsite on *konstruktio*, joka on abstrakti käsite, jolla on loputon määrä mahdollisia toteutumia. Kaikki ihmisen luomat artefaktit ovat konstruktioita, jotka eivät ole sellaisenaan löydettyjä, vaan ne keksitään ja kehitetään. Mikäli kehitetään konstruktio, jollaista ei ole koskaan aiemmin tehty, luodaan samalla uutta todellisuutta. Konstruktiivinen tutkimus johtaa todelliseen toimintaan kohdeorganisaatiossa ja tämän toiminnan perusteelliseen analysointiin. Konstruktiivisen tutkimusotteen yleisiin piirteisiin lukeutuu etnografisten

metodien (mm. havainnointi ja haastattelut) soveltaminen tutkimuksen empiirisessä osassa. (Lukka, 2001.)

Tässä tutkimuksessa tapaustutkimuksen kohteena oleva tapaus on perinnesovelluksen modernisointiprosessi pilvipalveluksi. Modernisoinnin tuloksena syntyvä artefakti on tieteellinen laskentasovellus, jota voidaan käyttää pilvipalveluna. Runesonin ja Höstin (2009) jaottelun mukaan tässä tutkimuksessa on sekä tutkivan että parantavan tapaustutkimuksen piirteitä.

5.3 Tiedon kerääminen

Yinin (1994, s. 80) mukaan tietoja voidaan kerätä tapaustutkimuksessa dokumenteista, arkistoista, havainnoinnilla, haastatteluilla ja fyysisten artefaktien tutkimisella. *Havainnointi* voi olla systemaattista tai osallistuvaa. Tämän tutkimuksen yhteydessä käytetään *osallistuvaa havainnointia*, jossa tutkija osallistuu tutkittavan kohteen toimintaan ja tekee havaintoja tutkimuskohteesta luonnollisessa tai luonnolliseen toimintaan mukautuneessa ympäristössä (Hirsjärvi, Remes & Sajavaara, 2010).

Haastattelu on tehokas ja tärkeä tiedonkeruumenetelmä, joka perustuu vuorovaikutustilanteeseen, jossa tutkijan tehtävä on edistää keskustelua omalla toiminnallaan (Järvinen & Järvinen, 2011). Haastattelu on tiedonkeruutapana hyvin joustava, koska haastattelutilannetta voidaan säädellä tilanteen mukaan (Järvinen & Järvinen, 2011), mutta toisaalta haastateltavien taipumus tuottaa toivottuja vastauksia ja pidättäytyä puhumasta epämiellyttävistä asioista voi heikentää haastattelun luotettavuutta (Hirsjärvi ym., 2010).

Tässä tutkimuksessa käytetty haastattelun muoto on *teemahaastattelu*, joka sijoittuu strukturoidun ja avoimen haastattelun välimaastoon. *Strukturoidussa haastattelussa* kysymysten muoto ja esittämisjärjestys on ennalta määrätty, kun taas *avoimessa haastattelussa* ei ole lainkaan kiinteää runkoa ja tutkija pyrkii selvittämään haastateltavien ajatuksia, mielipiteitä ja käsityksiä keskustelun avulla. *Teemahaastattelussa* haastattelun aihepiiri (teema) on etukäteen tutkijan ja haastateltavien tiedossa, mutta kysymyksiä ei ole muotoiltu tarkasti, eikä kysymysten esittämisjärjestys ole ennalta määritelty. (Hirsjärvi ym., 2010.)

Tässä tutkimuksessa pyritään konstruktion luomisen kautta lisäämään tietämystä tarkastelun kohteena olevasta ilmiöstä ja tämän tiedon avulla tuottamaan vastauksia määriteltyihin tutkimuskysymyksiin, jotka ovat peräisin alkuperäisestä tutkimusongelmasta. Konstruktion luomisen yhteydessä tutkija kirjaa havaintoja, joita reflektoidaan aihetta käsittelevään kirjallisuuteen. Reflektoinnin tarkoituksena on tunnistaa, miten tutkijan konstruktiota tehdessä havainnot asettuvat suhteessa tutkimuskysymyksiin ja kirjallisuuteen.

Empiirisen osuuden pääasiallisena tiedonkeruumenetelmänä käytetään siis havainnointia, jonka lisäksi tutkimuksen alussa ja lopussa suoritetaan teemahaastattelut, joiden avulla voidaan arvioida tutkimuksen lähtö- ja lopputilanteita sekä niiden kautta konstruktion onnistumista. Tieteen kannalta valitun tutkimusmenetelmän merkittävin heikkous on se, että tutkija toimii itse sekä konstruktion laatijana että myös konstruktioprosessista kumpuavien havaintojen tekijänä. Toinen tutkija saattaisi havainnoida ja nostaa esiin erilaisia seikkoja. Myös tutkijan ammattiosaamisen kohdentuminen vaikuttaa käytettävien menetelmien, käytänteiden ja teknologioiden valintaan. Tämä pyritään huomioimaan lopullisia tutkimustuloksia arvioitaessa (vrt. luku 7).

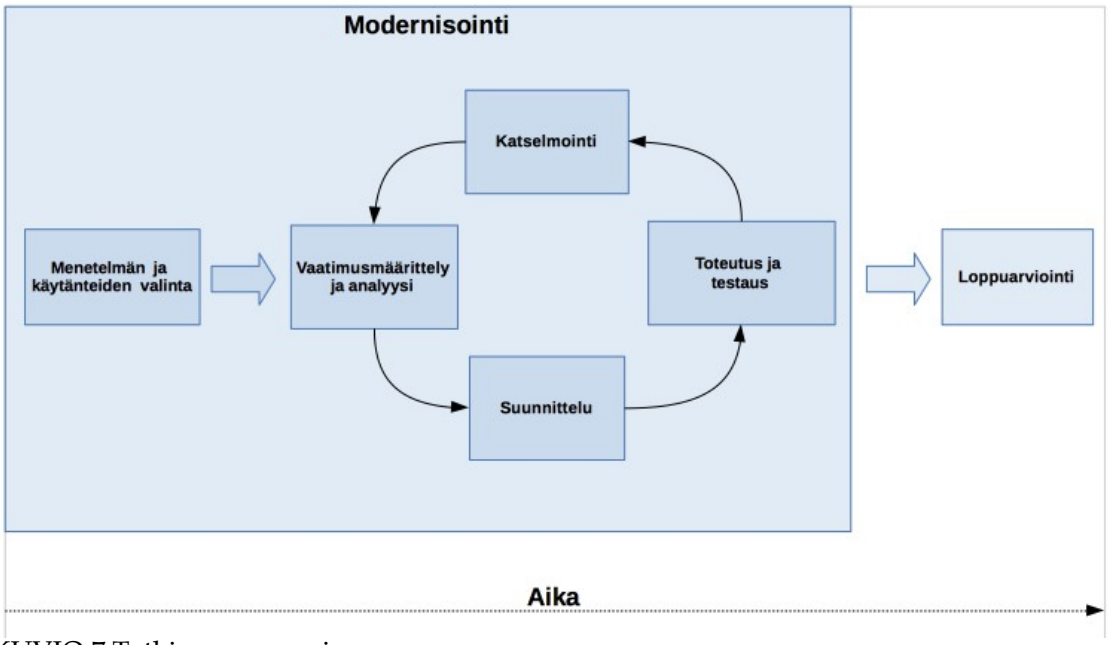
5.4 Tutkimusprosessi

Tapaustutkimusta varten suunniteltiin tutkimusprosessi. Prosessi on esitetty kuviossa 7. Tutkimus alkoi menetelmän ja käytänteiden valinnalla. Tiedon kerääminen tehtävien valintojen tueksi toteutettiin haastattelemalla sovelluksen ylläpitäjää ja johtavaa tutkijaa. Haastattelu oli tyypiltään teemahaastattelu, ja siinä käsiteltiin modernisoinnin motiiveja sekä määriteltiin, millainen haluttu modernisoinnin lopputulos olisi. Haastattelutilanne oli teemahaastattelun hengessä vapaamuotoinen, eikä siinä noudatettu mitään tunnistettavaa kaavaa. Tutkija laati haastattelutilanteessa muistiinpanoja käsitellyistä asioista. Haastattelu kesti neljä tuntia, ja sen aikana käsiteltiin laajasti modernisoitavaan sovellukseen ja sen toimintaympäristöön vaikuttavia seikkoja sekä modernisoinnin tuloksena syntyvän uudistetun sovelluksen piirteitä. Haastattelun jälkeen tutkija analysoi haastattelutilanteessa laaditut muistiinpanot ja litteroi niistä ydinasiat dokumentiksi (liite 1), joka toimi pohjana modernisointiprosessin menetelmävalinnalle ja alustaville vaatimuksille. Modernisointiprosessin suoritti tutkija itse kolmen kuukauden aikana kesä-elokuussa 2014.

Menetelmän ja käytänteiden alustavan valinnan jälkeen modernisointia toteutettiin iteratiivisesti. Yksi iteraatio koostui vaatimusmäärittelystä ja analyysista, suunnittelusta, toteutuksesta ja testauksesta sekä katselmoinnista. *Katselmoinnit* suoritettiin sopivin aikavälein perinnesovelluksen ylläpitäjän ja johtavan tutkijan kanssa, ja niissä arvioitiin jo toteutettuja toiminnallisuuksia sekä tarvittaessa tehtiin päätös muokata vaatimuksia seuraavassa iteraatiossa. Myös uusia hyödylliseksi osoittautuneita käytänteitä otettiin käyttöön tutkimuksen edetessä.

Tutkija teki modernisaation aikana jatkuvasti havaintoja siitä, mitkä käytännön asiat aiheuttivat eniten ongelmia, sekä vastaavasti siitä, mitkä asiat saatiin toteutettua vaivattomasti valitulla toimintatavalla. Tutkija kirjasi havaintonsa, jotta niitä voitaisiin myöhemmin arvioida suhteessa modernisoinnin lopputulokseen ja erityisesti aihetta käsittelevään kirjallisuuteen. Modernisaatiovaihe päättyi, kun ylläpitäjä ja tutkija olivat tyytyväisiä modernisoituihin ja täysin uusiin toiminnallisuuksiin. Modernisoitua sovellusta ei otettu välittömästi käyttöön, koska sitä haluttiin käyttää ”toiminnallisuuden validointi” -tyyppisenä (proof of concept) artefaktina tulevaisuuden valintoja ajatellen. Modernisoinnin tuotos korvaa mahdollisesti tulevaisuudessa vanhan sovelluksen. Tutkija sai myös palkkaa tekemästään työstä.

Modernisoinnin päätyttyä johtavalle tutkijalle ja sovelluksen ylläpitäjälle tehtiin toinen teemahaastattelu, jossa heidän tuli arvioida modernisointiprosessin onnistumista. Tutkimuksen suorittanut tutkija litteroi haastattelun tulokset (liite 2) ja arvioi niitä osana tutkimuksen tuloksia (luku 6).



KUVIO 7 Tutkimusprosessi

Tutkimuksen tuloksissa pyritään kirjallisuuskatsauksen reflektoinnin ja tutkijan tekemien havaintojen pohjalta vastaamaan tutkimuskysymyksiin (alaluku 1.4) tämän tapauksen kautta alakysymysten avulla:

- Miksi ja miten modernisointiin päädyttiin?
- Miksi sovellusalustaksi ja -malliksi päätettiin valita nimenomaan pilvipalvelut?
- Miten ja miksi päädyttiin valittuun pilvipalveluntarjoajaan?
- Miksi olemassa olevia pilvimodernisointimenetelmiä ei hyödynnetty sellaisenaan?
- Miten modernisointi toteutettiin käytännössä?
- Mitkä toimintatavat, menetelmät ja käytänteet havaittiin hyviksi ja mitkä huonoiksi?
- Millaiseen sovellusarkkitehtuuriin päädyttiin ja mitkä olivat syyt kyseisen arkkitehtuurin valinnalle?
- Mitä yhtäläisyyksiä ja eroja kirjallisuudessa esitettyihin menetelmiin prosessissa oli havaittavissa?
- Sopisivatko jotkin osat kirjallisuudessa esitetyistä menetelmistä sellaisenaan tämän tapauksen kaltaisiin modernisointiprosesseihin?
- Mitä erityisiä vaatimuksia tieteellisen laskentasovelluksen modernisointi aiheuttaa, ja kuinka nämä vaatimukset toteutettiin tässä tapauksessa?
- Nousiko esiin hyviä käytänteitä, toimintatapoja tai jopa menetelmiä, jotka olisivat sovellettavissa vastaavanlaisiin modernisointiprosesseihin?
- Mitä olisi voitu tehdä paremmin? Nousiko esiin huonoja käytänteitä, joita vastaavissa tapauksissa tulisi välttää?

6 TULOKSET

Tässä luvussa esitellään suoritetun tapaustutkimuksen tulokset. Tulokset esitetään siinä järjestyksessä, jossa modernisaatioprosessi eteni. Tutkijan havaitsemia ja esille nostamia seikkoja käsitellään alan kirjallisuuteen ja tutkimusprosessiin reflektoiden.

6.1 Modernisointimenetelmän valinta

Tässä alaluvussa kerrotaan pohdinnoista ja päätöksistä koskien tieteellisen laskentasovelluksen modernisoinnissa noudatettavaa menetelmää käsillä olevassa tapauksessa.

Modernisointimenetelmän valintaan vaikuttaa monta seikkaa, joten on mahdotonta asettaa menetelmiä paremmuusjärjestykseen yksiselitteisesti. Menetelmiä tulee aina arvioida suhteessa kyseessä olevaan sovellukseen, kehittämisprosessiin ja sen ympäristöön. Tässä tutkimuksessa keskiössä on tieteellisen laskentasovelluksen tapaus, jolle on tyypillistä se, että liiketoiminnallisia seikkoja ei tarvitse selvittää ja arvioida kovin tarkkaan. Tieteen piirissä pätevät yleensä muutenkin erilaiset arvot kuin liiketoiminnassa, mikä vaikuttaa useisiin sovelluksen ulkoisiin reunaehtoihin ja jopa sovelluksen sisäiseen toteutukseen. Esimerkiksi asiat, jotka haluttaisiin mahdollisesti pitää liiketoiminnassa tiiviisti yrityssalaisuuksina, voivat tieteen piirissä olla seikkoja, jotka halutaan tuoda tieteen läpinäkyvyyden vuoksi näkyvästi tutkijayhteisön tietoisuuteen.

Mikäli halutaan valita menetelmä, jonka avulla voidaan modernisoida tieteellinen laskentasovellus pilvipalveluksi, on valittava jokin kolmesta valmiista kirjallisuudessa esiintyvästä pilvimodernisointiratkaisusta (Chauhan & Babar, 2012; Menychtas ym., 2013; Mohagheghi ym., 2010) tai sovellettava muihin tarkoituksiin tarkoitettuja tai geneerisiä menetelmiä. Joka tapauksessa menetelmää valittaessa täytyy puntaroida, miten hyvin se soveltuu käsillä olevaan tapaukseen, mikä edellyttää jossain määrin perehtymistä menetelmän yksityiskohtiin. Joissain tilanteissa menetelmistä on kuitenkin yksinkertaista havaita, mikäli ne eivät sovellu kyseiseen tapaukseen.

Tieteellisen laskentasovelluksen tapauksessa voidaan tehdä oletus, että ne menetelmät, jotka painottavat modernisoinnin liiketoimintapuolta, tuovat modernisointiin ylimääräistä painolastia. Mikäli liiketoiminnallinen puoli on sovel-

luksen tai sen ympäristön kannalta oleellista, on ehdottomasti kannattavaa valita sellainen malli, joka huomioi myös liiketoimintapuolen, sillä pilvipalvelumalli on merkittävä muutos myös sovelluksen mahdolliseen ansaintalogiikkaan. Monesti tämä on tieteellisten sovellusten tapauksessa toissijaista.

Seuraavaksi pohditaan soveltuvuutta ensin niiden menetelmien osalta, joita on kirjallisuudessa esitetty pilvipalveluja käyttävään modernisointiin. Näitä menetelmiä ovat ARTIST (Menychtas ym., 2013), REMICS (Mohagheghi ym., 2010) ja Chauhanin ja Babarin (2012) menetelmä. Sen jälkeen tarkastellaan palvelukeskeistä arkkitehtuuria (Service-Oriented Architecture) koskevien menetelmien soveltuvuutta yleisellä tasolla.

ARTIST- ja REMICS-menetelmät perustuvat *mallintamiseen*, jossa perinnesovelluksesta laaditaan formaali malli, joka transformaatioiden ja mahdollisten metamallien avulla muunnetaan modernisoitavaan muotoon. Mallien käytön etu on se, että lähdekoodin runko saadaan usein generoitua automaattisesti mallin perusteella. Toinen merkittävä etu on se, että mallit ovat usein uudelleenkäytettäviä. Mikäli modernisointi tapahtuu organisaatiossa, jossa tehdään useita vastaavanlaisia projekteja ja saatavilla on mallivetoisen kehittämisen ammattilaisia ja riittävästi resursseja, on näiden menetelmien käyttö ehdottomasti suotavaa, koska mallintamisen hyödyt nousevat esiin parhaiten juuri uudelleenkäytön kautta. Suhteellisen pienessä ja kertaluontoisessa projektissa mallintamisen käyttäminen voi vaikuttaa ylilyönniltä. Tämä tosin riippuu pitkälti myös siitä, miten harjaantuneita kehittäjät ovat mallintamisessa ja mitkä ovat organisaation tavoitteet modernisaation suhteen.

Chauhanin ja Babarin (2012) esittämä menetelmä painottaa erityisesti sopivan palveluntarjoajan valintaprosessia tunnistettujen vaatimusten perusteella. Menetelmä ei edellytä mallintamista ja vaikuttaa muutenkin vähemmän formaalilta kuin ARTIST ja REMICS. Menetelmässä keskiössä on sopivimman palveluntarjoajan arviointi erinäisten vaatimuksista kumpuavien attribuuttien perusteella. Neljä menetelmän seitsemästä vaiheesta muodostuu erilaisten arviointiartefaktien laatimisesta. Menetelmä on kirjallisuudessa esiintyvistä menetelmistä sellaisenaan käyttökelpoisin tieteellisten laskentasovellusten tapauksessa. Toisaalta muuttuvien vaatimusten tapauksessa menetelmä on kuitenkin huono. Arviointiartefaktit laaditaan vaatimusten perusteella, joten ne kaikki täytyy laatia uudestaan aina, kun vaatimukset muuttuvat, ja arviointi täytyy tämän jälkeen tehdä uudelleen.

Iteratiivisuuden puute on keskeinen ongelma kirjallisuudessa esiintyvissä pilvimodernisaatiomenetelmissä. REMICS-menetelmän dokumentaatiossa menetelmän väitetään olevan iteratiivinen (Mohagheghi ym., 2010), mutta iteratiivisuus ei vaikuttaisi mallin spesifikaation perusteella toteutuvan siinä kuin pieniltä osin. ARTIST-menetelmä on selvästi vesiputousmallin kaltainen, toisin sanoen vaiheet seuraavat toisiaan (Menychtas ym., 2013). Chauhanin ja Babarin (2012) esittämässä menetelmässä iteratiivisuutta esiintyy ainoastaan arkkitehtuurin valinnan ja pilviympäristöjen erojen arvioinnin yhteydessä. Mikään esitetyistä malleista ei huomioi vaatimusten mahdollista muuttumista modernisointiprosessin aikana. Tämä vaikuttaa merkittäviltä osin siihen, että menetelmät tuntuvat kovin raskailta. Inkrementaalisuuden puute ja siitä johtuva heikko tai olematon kyky reagoida muuttuviin vaatimuksiin tekee näistä menetelmistä käytännössä hyvin haastavia ellei mahdottomia sovittaa ketterän kehityksen periaatteisiin (Beck ym., 2001) sellaisenaan.

Edellä mainituista syistä johtuen tutkija päätyi ratkaisuun, jossa yhdisteltiin tutkijan ammattiosaamista järjestelmäkehityksestä valikoituihin piirteisiin

tutkijalle ennestään tutuista menetelmistä ja käytänteistä, kuten testivetoisesta kehittämisestä (test-driven development) (Beck, 2003), ketteristä periaatteista (Beck ym., 2001) sekä Lean-ajattelusta sovellettuna ketterään ohjelmistokehitykseen (Poppendieck & Poppendieck, 2003). Kehittämisprosessille ominaisia piirteitä ovat iteratiivisuus (Beck ym., 2001), turhan toiminnan välttäminen (Poppendieck & Poppendieck, 2003, ss. 1–9), nopea toimivan artefaktin toimittaminen ja arviointi (Poppendieck & Poppendieck, 2003, ss. 69–91), päivittäisten toimien automatisointi (Dzhurov, Krasteva & Ilieva, 2009), sekä yksikkötestien laatiminen olennaisena osana kehitysprosessia (Beck, 2003, s. 15). Menetelmässä tunnistettavia vaiheita ovat XP (Extreme Programming) -menetelmän (Agarwal & Umphress, 2008) mukainen vaatimusmäärittely, suunnittelu, toteutus ja testaus sekä katselmoinnit (yhteistyössä asiakkaan kanssa). Kehittämisprosessista kerrotaan yksityiskohtaisemmin alaluvussa 6.4.

Mikäli pilvimodernisoinnin yhteydessä ei haluta käyttää mitään edellä mainituista valmiista menetelmistä, mutta halutaan soveltaa jotain olemassa olevaa menetelmää, on hyödyllistä lähestyä palvelukeskeisiin arkkitehtuureihin tähtääviä menetelmiä. Pilvipalvelut perustuvat suurilta osin palvelukeskeisen arkkitehtuurin hyödyntämiseen (Chauhan & Babar, 2012), minkä vuoksi nämä kaksi ovat teknisesti hyvin lähellä toisiaan. Pilvipalvelumalli on laajennus palvelukeskeiseen arkkitehtuuriin, joten menetelmiä, joissa sovellus modernisoidaan palvelukeskeiseen arkkitehtuuriin, voidaan yleensä käyttää myös pilvimodernisointimenetelmän pohjana, kuten Chauhan ja Babar (2012) ovat tehneet. Esimerkkejä menetelmistä, joissa sovellus modernisoidaan palvelukeskeiseksi, ovat esittäneet muun muassa Razavian ja Lago (2010), Cetin ym. (2007) sekä Lewis ym. (2005).

Yhteenvetona menetelmän valinnasta voidaan todeta, että olemassa olevat pilvimodernisointimenetelmät ovat sellaisenaan liian raskaita tai muuten sopimattomia ketterään iteratiiviseen kehitykseen. Mikäli organisaatiossa on mallinnusosaamista ja tulevaisuudennäkymissä on useita modernisointiprojekteja, kannattaa hyödyntää mallivetoisia REMICS- (Mohagheghi ym., 2010) ja ARTIST-menetelmiä (Menychtas ym., 2013). Mikäli modernisoinnissa suuri huolenaihe on oikean palveluntarjoajan valinta, kannattaa hyödyntää Chauhanin ja Babarin (2012) laatimia arviointimenetelmiä. Mikäli pilvimodernisaatio halutaan toteuttaa ketterästi, on järkevintä ketterien periaatteiden mukaisesti antaa modernisaation toteuttavien ammattilaisten valita itse menetelmänsä ja käytänteensä oman harkintansa mukaan, kuten toimittiin myös tämän tutkimuksen yhteydessä.

6.2 Vaatimusmäärittely ja analyysi

Aloite tutkimuksen kohteena olevan sovelluksen uudistamiseksi tuli sovelluksen parissa työskentelevältä tutkijalta sekä sovelluksen ylläpitäjältä. Teemahaastattelussa tutkija pyrki selvittämään ne seikat, jotka olivat johtaneet nykyisen sovelluksen kanssa siihen tilanteeseen, jossa modernisointi vaikutti järkevältä vaihtoehdolta. Haastattelussa (vrt. liite 1) ilmenneet keskeisimmät syyt olivat:

1. Nykyinen sovellus on hankala ylläpitää ja siihen on vaikea tuoda uusia ominaisuuksia.

2. Nykyisen sovelluksen prosesseja tulisi automatisoida, esimerkiksi simulaatioiden ajo ja tallennus järjestelmään.
3. Datan määrä on suuri ja kasvaa jatkuvasti, mikä aiheuttaa ylläpito-ongelmia.
4. Simulaatioiden konfigurointi ja ajaminen kaipaisi yhtenäisen alustan.
5. Sovellus halutaan integroida muihin palveluihin.
6. Sovellus halutaan tehdä helposti muiden tutkijoiden saavutettavaksi ja käytettäväksi.
7. Myös sovelluksella tehtävän tutkimuksen kannalta helpompi käytettävyys ja parempi saavutettavuus olisivat positiivisia asioita, sillä kokeiden tulokset olisivat helpommin muiden saatavilla ja jopa muiden verifioitavissa, mikäli sovelluksen käyttäjät voivat suorittaa ja toistaa kokeita itsenäisesti.
8. Käyttäjiä halutaan mahdollisesti pystyä laskuttamaan jossain vaiheessa käyttämästään prosessoriajasta ja tallennustilasta.
9. Tutkimusprojektien rahoitus on aina rajallinen, joten mitä vähemmän resursseja kuluu infrastruktuuriin ja automatisoitavissa olevien prosessien suorittamiseen, sitä parempi.
10. Sovelluksesta halutaan tehdä nykyistä vakaampi. Esimerkiksi yksittäisen tutkimuslaitoksen tiloissa toimiva sovellus on altis muun muassa sähkökatkoille ja luonnonkatastrofeille.

Kaksi ensimmäistä ongelmaa voitaisiin korjata siten, että paikallisia palvelimia voitaisiin edelleen käyttää sovelluksen alustana. Kyseiset muutokset tarvitsisivat vain sovelluksen arkkitehtuurin selkeyttämistä sekä lähdekoodin refaktorointia niiltä osin kuin koodi on toteutettu heikosti. Seikat 3-10 ovat sitä vastoin ratkaistavissa tehokkaimmin hyödyntämällä pilvipalveluiden IaaS- ja SaaS-palvelumalleja.

Ensimmäiseen seikkaan vaikutti sovelluksen web-komponenttien huono suunnittelu, heikkolaatuinen toteutus sekä se, että sovelluksen ylläpitotoimet olivat keskittyneet Seacordin (2003, s. 4) oletuksen ja sitä tukevien havaintojen mukaisesti parantelevaan ylläpitoon, eikä korjaavia, mukautuvia ja ennaltaehkäiseviä toimia ollut juuri pantu toimeen. Nämä seikat olivat johtaneet lähdekoodin huomattavaan rapautumiseen ja Lehmanin (1980) toisen lain mukaiseen ylläpidettävyyden heikkenemiseen. Tutkijan suorittama perinnesovelluksen lähdekoodin tarkastelu tuki näitä väittämiä yhdessä sovelluksen ylläpitäjän näkemysten ja kokemusten kanssa.

Kaikkia haluttuja toiminnallisuuksia ei ollut toteutettu, koska uusien ominaisuuksien lisääminen sovellukseen oli vaikeaa. Web-komponenttien lähdekoodin ja arkkitehtuurin epäselvyys johti siihen, että sovellukseen oli haastavaa toteuttaa uusia toiminnallisuuksia. Uusien ominaisuuksien lisäämisen ja vanhojen toiminnallisuuksien refaktoroinnin hankaluutta lisäsi se tosiasia, että sovellukseen ei ollut laadittu minkäänlaisia automatisoituja testejä, joiden avulla lähdekoodista olisi voitu havaita virheitä. Sovelluksen vähäinen dokumentaatio vaikeutti asiaa entisestään.

Ongelma suuren ja jatkuvasti kasvavan datamäärän kanssa on ratkaistavissa pilvipalveluiden ominaispiirteisiin kuuluvan *joustavuuden* (Brian ym., 2012, s. 7; Mell & Grance, 2009) avulla, jonka mukaan tarvittavien resurssien vapauttaminen ja hankkiminen on mahdollista nopeasti ja vaivattomasti. Käytännössä tämä tarkoittaa sitä, että pilvessä palveluntarjoajalta on nopeasti ja vaivattomasti saatavissa lisää tallennuskapasiteettia aina, kun sitä tarvitaan, ja vastaa-

vasti ylimääräisistä resursseista on yhtä vaivatonta luopua. Tieteellisille laskentasovelluksille on hyvin tyypillistä, että ne tuottavat valtavia määriä dataa ja vaativat paljon prosessointitehoa (Tsaftaris, 2014; Vecchiola ym., 2009). Pilvipalveluja on hyödynnetty useissa tieteellisten laskentasovellusten tapauksissa (Z. Huang ym., 2013; Juve ym., 2010, 2013; Rehr ym., 2011; "Scientific Computing with EC2 Spot Instances", 2011; Tsaftaris, 2014) juuri niiden tarjoamien skaalautuvien resurssien ja edullisten hintojen vuoksi.

Neljäs ongelma juontui siitä, että vanhassa ratkaisussa simulaatiot ajetaan käytännössä manuaalisesti erillään muusta sovelluksesta. Tutkijat saavat sovelluksen web-komponentin kautta käyttäjiltä pyyntöjä ajaa simulaatio, jonka jälkeen tutkija kääntää, asentaa ja konfiguroi simulaatiokirjaston jollekin saatavissa olevalle laitteistoalustalle (usein kannettava tietokone) ja ajaa simulaation käyttäjän toivomilla parametreilla. Simulaation ajaminen on hyvin prosessori- ja muisti-intensiivinen prosessi, ja simulaation ajaminen voi käyttäjän haluamista parametreistä ja käytetystä laitteistosta riippuen kestää jopa viikkoja, minkä ajan kyseiset resurssit ovat varattuja. Kun simulaatio on suoritettu, kerää tutkija simulaatiokirjaston tuottamat data- ja lokitiedostot laitteiston kovalevyltä, tuo ne palvelimelle, jossa sovelluksen web-komponenttia ajetaan ja skriptien avulla lisää uudet tiedostot muiden käyttäjien kyseltäviksi web-komponentin kautta. Prosessin automatisoinnin kannalta ongelmaksi paikallisia palvelimia käyttäen muodostuisi se, mistä varataan prosessointi- ja muistikapasiteetti simulaation suorittamista varten. Paikallisilla palvelimilla fyysisten ja mahdollisesti virtuaalisten resurssien jatkuva varaaminen koituisi kalliiksi tarvittavien investointien vuoksi ja resurssit olisivat mitä luultavimmin suurimman osan ajasta käyttämättöminä. IaaS-pilvipalvelua hyödyntämällä voidaan valjastaa kokonainen virtuaalikone tai virtuaalikonerypäs simulaation tarvitsemaksi ajaksi ja maksaa siitä vain käytetystä ajasta riippuva korvaus (Heino, 2010, ss. 52–53; Mell & Grance, 2009, s. 8; Salo, 2010, s. 25). Simulaation suorittamisen kannalta optimaalinen virtuaalilaitteisto ja ohjelmisto voidaan konfiguroida pilvipalveluun ylläpidon toimesta, ja tämän virtuaalikoneen instansseja voidaan luoda aina kun niitä tarvitaan palveluntarjoajan huolehtiessa infrastruktuurista. IaaS-pilvipalveluun perustuva ratkaisu tekee mahdolliseksi automatisoidusti valjastaa simulaatioille parhaiten soveltuvan ja ennen kaikkea yhtenäisen (virtuaalisen) laitteiston aina silloin kun sitä tarvitaan.

Kohdat 5, 6 ja 7 ovat ratkaistavissa toteuttamalla sovellukseen *palvelukeskeinen arkkitehtuuri*, jossa sovelluksen keskeiset toiminnot paljastetaan verkon kautta kutsuttaviksi palveluiksi (Nurhasan ym., 2013). Palveluiden merkittävä etu on myös se, että palveluita yhdistämällä voidaan laatia koostepalveluita (Su ym., 2011), mikä laajentaa sovelluksen käyttömahdollisuuksia merkittävästi. Tämä seikka on erityisen mielenkiintoinen kyseisen aihealueen tutkijayhteisön kannalta, sillä palveluiden avulla ensinnäkin tiedon hankkiminen verkon kautta helpottuu ja toiseksi se mahdollistaa periaatteessa tulevaisuudessa myös useampien tieteellisten sovellusten yhteiskäytön. Palvelukeskeisyyttä oli toteutettu vanhaan sovellukseen jo valmiiksi datan kyselyominaisuuksien osalta. Toteutettujen palveluiden pääasiallisia käyttäjiä ovat tutkimusprojektin yhteistyötahot. Tutkijoiden toiveissa on mahdollistaa tulevaisuudessa useamman simulaatiosovelluksen yhdistäminen toisiinsa siten, että eri simulaatiot olisivat ajon aikana vuorovaikutuksessa keskenään ja voisivat vaikuttaa toistensa syötteisiin. Yksi tapa saavuttaa tämä tavoite on tarjota simulaatioiden ajamisen ja ajonaikaisen muokkaamisen mahdollistavat toiminnot verkon kautta palveluna muille sovelluksille.

SaaS-pilvipalvelumallin toteuttamalla käyttäjät voivat käyttää sovellusta pelkän internetselaimen avulla (Brian ym., 2012, ss. 9–10). Tämän sovelluksen tapauksessa tämä tarkoittaisi sitä, että käyttäjät voisivat määrittää, ajaa, tallettaa, julkaista ja kysellä simulaatioita suoraan omalla internetselaimellaan. Tällöin kaikki toiminnot olisivat automatisoituja, eikä tutkijoiden tai sovelluksen ylläpitäjän väliintuloa tarvita muuten kuin mahdollisissa konsultointi- ja virhetilanteissa. Verrattuna tämänhetkiseen tilanteeseen SaaS-malli olisi merkittävä parannus sovelluksen ylläpidon, tutkijoiden ja muiden käyttäjien sekä tehtävän tutkimuksen kannalta. SaaS-mallissa käyttäjää voitaisiin myös laskuttaa tämän kuluttamien resurssien mukaisesti, mikä täyttäisi tutkijoiden ja ylläpitäjän toiveen kohdassa 8.

Pilvipalvelumallilla pystyttäisiin saavuttamaan myös huomattavia säästöjä verrattuna paikallisten palvelinten käyttöön (kohta 9). Tarkkoja laskelmia nykytilanteen ja pilvipalvelumallin vertailuksi ei päätetty suorittaa, koska pelkästään muutaman pilvipalveluntarjoajan hinnastoja tarkastelemalla selvisi, että IaaS-palvelut ovat yleisesti ottaen hyvin edullisia, mikä yhdistettynä toimintojen mahdollisen automatisoinnin tuomiin henkilöstöresurssien säästöihin olisi merkittävä parannus nykytilanteeseen. Väitettä pilvipalveluiden edullisuudesta verrattuna infrastruktuurin ylläpitämiseen itse tukevat mm. Zhang (2010), Srirama ym. (2013) sekä Huang ym. (2013).

Kohta 10 on ratkaistavissa hankkimalla IaaS-palvelut sellaiselta palveluntarjoajalta, joka mahdollistaa toimintojen peilaamisen useampaan maantieteelliseen sijaintiin. Esimerkiksi Amazon on tällainen palveluntarjoaja (Jinesh & Sajee, 2014).

Yhteenvetona modernisoinnin motiiveista voidaan todeta, että päätös modernisoinnista ja valinta modernisoida sovellus pilvipalveluksi olivat hyvin selkeitä tässä tapauksessa. Perinnesovelluksen merkittäväksi rapautumisen syyksi jäljitettiin huono suunnittelu, mikä viittaa siihen, että korjauksia tulisi tehdä arkkitehtuurin tasolla, mikä sopii modernisoinnin määritelmään. IaaS- ja SaaS-pilvipalvelumallien hyödyntämiseen päädyttiin, koska se ratkaisisi periaatteessa kaikki loput perinnesovelluksen ylläpidossa ja käytössä havaitut ongelmat sekä toteuttaisi tutkijoiden toiveet sovelluksen paremmasta saavutettavuudesta ja tulevaisuuden mahdollisuuksista. Toisaalta on huomioitava, että tässä kyseisessä tapauksessa pilvipalveluiden edut ovat hyvin ilmeiset verrattuna paikallisten palvelimien käyttämiseen. Myös modernisoinnin syyt ovat hyvin selkeät perinnesovelluksen huomattavan rapautumisen vuoksi. Tämän tapauksen yhteydessä prosessin hyödyllisyyttä kasvattaa se, että modernisoimalla sovellus pilvipalveluksi tehdään lukuisia parannuksia, kun sovelluksen modernisoinnin tuloksena sen arkkitehtuuri ja lähdekoodi kokevat kasvojenkohotuksen, minkä lisäksi otetaan käyttöön IaaS-pilvipalveluiden hyödyt ja sovellukseen toteutetaan SaaS-palvelumalli. Tieteellisen laskentasovelluksen pääasiallinen tarkoitus on hyvin harvoin liiketoiminnan harjoittaminen. Tämän vuoksi motiivien tarkastelu keskittyi pääasiassa teknisiin ja käytännöllisiin seikkoihin, ja modernisaation taloudellista puolta on käsitelty lähinnä kiinteiden kustannusten minimoimisen näkökulmasta.

Vaatimusmäärittely on lähes aina sovelluskehitysprojektien ensimmäisenä tehtävien asioiden joukossa. Myös kirjallisuudessa esiintyvissä pilvimodernisointimenetelmissä suoritetaan aluksi vaatimusmäärittely, vaikkakin menetelmissä samasta asiasta puhutaan nimillä vaatimusten tunnistaminen (Chauhan & Babar, 2012) ja vaatimusten mallintaminen (Menychtas ym., 2013; Mohagheghi ym., 2010). ARTIST ja REMICS menetelmissä vaatimukset mallinnetaan pe-

rinnesovelluksesta takaisinmallinnusta (reverse engineering) käyttäen. Vaatimusten saattaminen formaalin mallin muotoon vaatii kuitenkin huomattavaa osaamista ja ymmärrystä mallintamisen käytöstä ohjelmistokehityksessä.

Yksinkertaisissa tapauksissa vaatimusten mallintaminen voi vaikuttaa liioittelulta, minkä takia tässäkään tutkimuksessa vaatimuksia ei päätetty mallintaa mihinkään formaaliin muotoon, vaan tutkija laati modernisoitavan sovelluksen alustavat vaatimukset johtavan tutkijan ja sovelluksen ylläpitäjän haastattelun perusteella perinteiseksi luonnollisella kielellä kuvatuksi dokumentiksi. Vaatimuksia ei pyritty laatimaan lopulliseen muotoonsa kerralla, koska ketterän lähestymistavan periaatteiden (Beck ym., 2001) mukaisesti oletettiin, että vaatimukset voivat muuttua modernisointiprosessin edetessä.

Modernisoinnin yhteydessä vaatimusmäärittelyssä voi olla hyödyllistä korostaa niitä asioita, jotka ovat johtaneet modernisoinnin tarpeeseen. Perinnesovelluksessa ilmenevistä ongelmista ja erityisesti näiden ongelmien syiden jäljittämisestä saadaan tietoa, jonka avulla voidaan pyrkiä ennaltaehkäisemään kyseisen ilmiön toistuminen modernisoidussa sovelluksessa. Joskus tämä voi tosin olla haastavaa, sillä perinnesovelluksen dokumentaatiossa ja täytäntöönpanossa voi esiintyä seikkoja, jotka eivät ole aina ilmeisiä (Seacord ym., 2003, ss. 1–2). Tämän tapauksen yhteydessä perinnesovelluksen alkuperäisiin vaatimuksiin lisättiin korkean tason laatuvaatimuksia kuten ylläpidettävyys, modulaarisuus ja siirrettävyys, koska kyseisten vaatimusten toteutumisen puute olivat keskeisiä havaittuja ongelmia perinnesovelluksessa.

Tieteellisen laskentasovelluksen vaatimusmäärittelyn havaittiin muistuttavan hyvin pitkälti kaupallisen sovelluksen vaatimusmäärittelyä. Huomattavin käytännön ero näiden kahden välillä lienee liiketoiminnallisten vaatimusten vähyys tieteellisten sovellusten tapauksessa, mikä juontuu liiketoiminnan ja tieteen motiivien eroista. Vaatimusmäärittelyn menetelmiä ja hyviä käytänteitä voidaan hyvin todennäköisesti käyttää sellaisenaan myös tieteellisten laskentasovellusten tapauksessa.

Pilvipalveluiden kasvaneen suosion yhteydessä lienee tarpeellista korostaa, että pilvipalveluiden käyttäminen ei yleensä ole järkevä vaatimus sovellukselle. Pilvipalvelumallit ovat ratkaisuja sovelluksen vaatimusten asettamille ongelmille. Sovellusten vaatimusten tulisi muodostua reaalimaailman tarpeista, eikä menetelmistä, joita halutaan käyttää. Monesti muodissa olevia teknologioita halutaan käyttää ilman sen järkevämpiä perusteluita ja tämä asia on hyvä tunnistaa myös tässä asiayhteydessä.

Yhteenvetona vaatimusmäärittelyn toteuttamisesta tieteellisen laskentasovelluksen pilvimodernisoinnin tapauksessa voidaan todeta, että vaatimusmäärittely ei poikkea tässä yhteydessä merkittävästi muusta vaatimusmäärittelystä. Modernisoinnin yhteydessä on kuitenkin hyödyllistä havaita ja oppia perinnesovelluksessa tehdyistä virheistä ja pyrkiä varmistamaan vaatimusten yhteydessä, että kyseisiä virheitä ei tehdä enää koskaan uudestaan. Pilvipalveluiden suuren suosion vallitessa kannattaa muistaa, että pilvi ei (yleensä) ole vaatimus, vaan mahdollinen ratkaisu vaatimusten asettamiin ongelmiin.

6.3 Suunnittelu

Tässä alaluvussa käsitellään tutkimusprosessissa määriteltyä suunnitteluvaihetta. Suunnittelussa keskitytään ensin pilvipalveluntarjoajan valintaan ja toiseksi sovelluksen arkkitehtuurin määrittämiseen.

Modernisoinnissa muutoksia sovellukseen tehdään arkkitehtuurin tasolla (Seacord ym., 2003, s. 12), ja modernisoitaessa pilveen sovelluksen arkkitehtuurissa on usein huomioitava eri pilvipalvelumallien sekä palveluntarjoajien määrittämät vaihtoehdot ja rajoitteet (Chauhan & Babar, 2012). Arkkitehtuurin määrityksen ja palveluntarjoajan valinnan voidaan katsoa liittyvän toisiinsa, minkä vuoksi niitä tarkastellaan myös tässä tutkielmassa yhdessä. Pilvipalveluntarjoajan ei kuitenkaan tarvitse olla sovelluksen arkkitehtuurin määrittävä tekijä kaikissa tapauksissa, kuten alla tullaan havainnollistamaan. Seacord (2003, s. 15) käyttää perinnesovelluksen virheellisen arkkitehtuurin rakenteellisten puutteiden ja virheiden korjaamisesta termiä *rakenteellinen parantaminen*.

Kirjallisuudessa esiintyvissä ARTIST- (Menychtas ym., 2013) ja REMICS-menetelmissä (Mohagheghi ym., 2010) mallinnetut vaatimukset transformoidaan haluttua arkkitehtuuria vastaavaan muotoon. Tämä takaa monien eri arkkitehtuurien kokeilumahdollisuuden, mutta edellyttää sitä, että vaatimusten mallintaminen perinnesovelluksesta on onnistunut hyvin. Chauhanin ja Babarin (2012) menetelmässä tunnistetaan vaatimusten ja määritetyn palveluntarjoajajoukon perusteella kaikki potentiaaliset arkkitehtuurivaihtoehdot ja valitaan niistä sopivin tai ainakin vähiten ongelmallinen arkkitehtuuri toteutusta varten.

Tämän tutkimuksen tapauksessa alustava arkkitehtuuri määritettiin myös vaatimusten pohjalta. Arkkitehtuurin määrityksessä tosin poikettiin kirjallisuudessa esiintyvistä pilvimodernisointimenetelmistä siten, että arkkitehtuurista pyrittiin tekemään palveluntarjoajasta riippumaton. Yksi sovelluksen laatuvaatimuksista oli sovelluksen siirrettävyys, millä tarkoitetaan tässä yhteydessä sitä, että sovellus on mahdollista siirtää toisen palveluntarjoajan pilveen. Ainoa oletus, joka palveluntarjoajasta tehtiin, oli se, että palveluntarjoaja tarjoaa IaaS-palveluna virtuaalikoneita, jotka ovat verkossa ja joihin voi itse asentaa ohjelmistoja, mikä kuuluu myös IaaS-palveluiden yleisesti hyväksyttyyn määritelmään (Brian ym., 2012, s. 9). Lopullisessa ratkaisussa käytetään tosin yksinomaan AWS:n tarjoamia palveluita, kuten SQS ja S3, mutta näiden vaikutus arkkitehtuuriin pystyttiin minimoimaan sovelluskehityksessä tehdyillä ratkaisuilla, joista lisää alaluvussa 6.4.

Modernisointiprosessin aluksi määriteltyä arkkitehtuuria kehitettiin iteraatioiden aikana vastaamaan muuttuvia ja täsmentyneitä vaatimuksia. Lopullisen arkkitehtuurin määrittämistä helpotti myös kehityksen edetessä lisääntynyt tieto pilvipalveluista, valitusta palveluntarjoajasta ja modernisointiprosesseista ylipäänsä. Iteratiivinen prosessimalli mahdollisti arkkitehtuurin muokkaamisen kesken modernisointiprosessin.

Palveluntarjoajan valinnassa käytettiin tämän tutkimuksen tapauksessa oikotietä. Amazon Web Services (AWS) valittiin perustuen tutkijan aiempaan kokemukseen Amazonin palveluista sekä useisiin tässäkin tutkielmassa esitettyihin raportoituihin tapauksiin (Z. Huang ym., 2013; Juve ym., 2010, 2013; Rehr ym., 2011; "Scientific Computing with EC2 Spot Instances", 2011; Tsaftaris, 2014), joissa AWS:ää on käytetty tieteellisten laskentasovellusten alustana. AWS:n valintaa puolsi myös palveluntarjoajan suuruus ja merkittävä asema palveluntar-

joajien keskuudessa (Jinesh & Sajee, 2014). Palveluntarjoajaan lukittautumista pidetään yhtenä ongelmana pilvipalvelumallissa (Armbrust ym., 2010; Brian ym., 2012, ss. 8–9, 14), mutta AWS:n tapauksessa ongelma ei ole niin suuri, koska avoin Eucalyptus-alusta on yhteensopiva AWS:n keskeisimpien palveluiden kanssa (”Eucalyptus and Amazon Web Services Compatibility”, 2014), joten hätätapauksessa oman AWS-yhteensopivan yksityisen pilven voi perustaa kuka tahansa. Myös alaluvussa 6.4 on kerrottu, millaisilla sovelluskehityksen ratkaisuilla palveluntarjoajaan lukittautumista pystytään ehkäisemään. Amazonin houkuttelevuutta ketterän kehityksen yhteydessä kasvattaa myös palveluiden mahdollinen ilmaiskäyttö tiettyyn rajaan saakka (”AWS Free Tier”, 2014), mikä mahdollistaa sovelluksen prototypoinnin AWS-palveluiden kanssa ilman lisäkustannuksia.

Kun perinnesovellus modernisoidaan SaaS-pilvipalveluksi, sen arkkitehtuuri muokataan hyvin usein palvelukeskeiseksi. Palvelukeskeisyyttä voitaneen pitää eräänlaisena edellytyksenä pilviympäristössä, koska pilvipalveluiden tarjoaminen ja käyttö perustuu nimenomaan pitkälti palvelukeskeiseen arkkitehtuuriin (Chauhan & Babar, 2012). Palvelukeskeisessä arkkitehtuurissa sovelluksen vaatimuksista tunnistetaan usein komponenteiksi miellettäviä kokonaisuuksia, jotka toteutetaan palveluiksi, jotka kätkevät sisäänsä kaiken teknisen kompleksisuuden, mitä palvelun toteuttamiseen tarvitaan, ja paljastavat itsestään ulospäin yksinkertaisen verkon kautta kutsuttavan rajapinnan (Nurhasan ym., 2013).

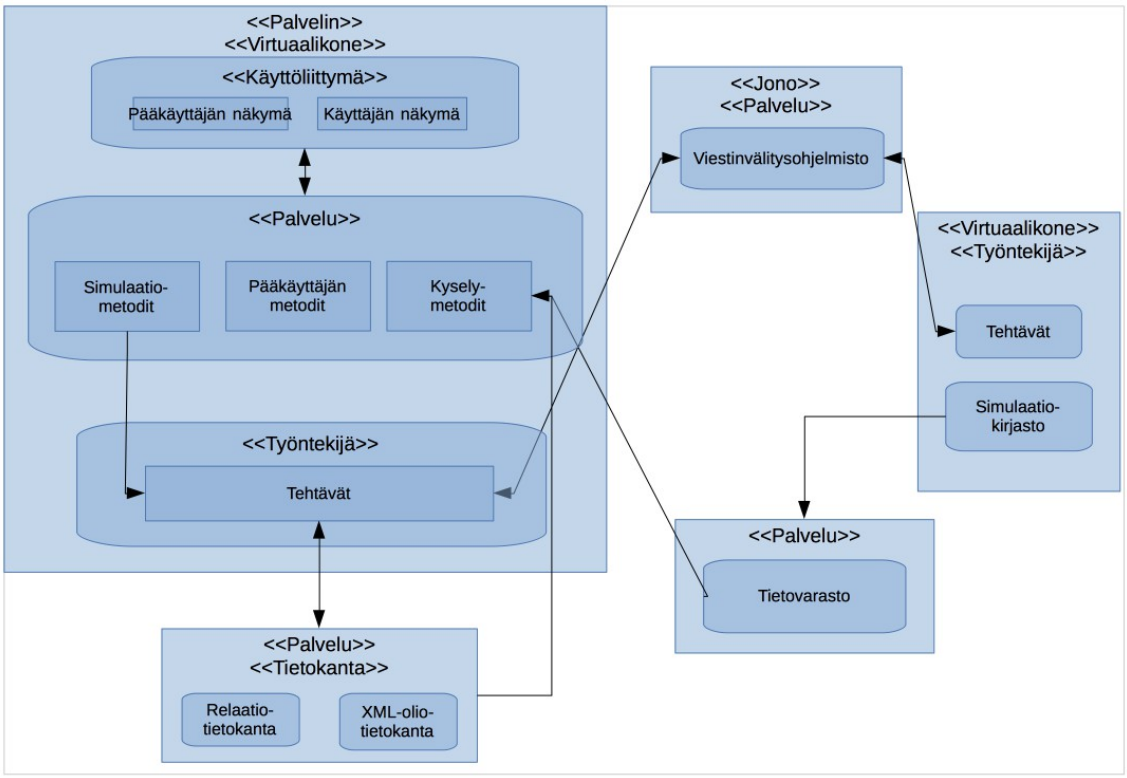
Palvelukeskeisen arkkitehtuurin toteuttaminen sovellukseen tuo mukanaan useita hyötyjä. Palvelukeskeisyys ohjaa suunnittelijan palastelemaan sovelluksen toiminnot loogisiksi ja itsenäisiksi komponenteiksi. Komponentit kasvattavat sovelluksen modulaarisuutta ja toteuttavat *haasteiden eriyttämisen* (separation of concerns), jolla tarkoitetaan sitä, että komponenteilla on selkeästi määritellyt omat vastuualueensa. Laadukkaasti toteutettuja komponentteja voi monesti käyttää uudelleen sellaisenaan myös muissa sovelluksissa. Koska komponentit ovat usein hyvin itsenäisiä kokonaisuuksia, on sovelluksen kehityskin helppo jakaa komponenteittain eri kehittäjille. Komponentin itsenäisyyden vuoksi myös komponentin korvaaminen toisella komponentilla on yleensä hyvin yksinkertaista. Seacord (2003, s. 15) käyttää modernisoinnin yhteydessä termiä *modularisointi* tällaisesta toiminnasta, jossa sovelluksen osista pyritään koostamaan loogisia ja uudelleenkäytettäviä komponentteja.

Kun sovelluksen toiminnot julkaistaan verkon kautta saavutettavina palveluina, on sovelluksen palveluita hyödyntäen yhdessä muiden palveluiden kanssa mahdollista koostaa koostepalveluita (Su ym., 2011), mikä laajentaa sovelluksen potentiaalisia käyttömahdollisuuksia valtavasti. Erityisen houkuttelevaa tämä on tieteentekijöiden kannalta. Tieteen tekemiseen käytetyt sovellukset voidaan julkaista palveluna, jolloin kuka tahansa pystyy verifioimaan työkalulla saavutetut tieteelliset tulokset. Lisäksi juuri yhdistelemisen avulla voidaan sovelluksia yhdistämällä luoda periaatteessa uusia sovelluksia, jotka ratkovat aina laajempia ja laajempia ongelmakokonaisuuksia.

Myös tässä tapauksessa perinnesovelluksen arkkitehtuuri päätettiin muokata palvelukeskeiseksi. Perinnesovelluksessa oli jo olemassa muutamia itsenäisiä palveluita, joiden kautta voitiin kysellä dataa, mutta modernisoinnin yhteydessä päätettiin toteuttaa kaikki sovelluksen keskeiset toiminnot palveluina, mukaan lukien simulaatioiden määrittely ja ajaminen. Palveluita määrittäessä pidettiin myös mielessä, mitkä kaikki toiminnot on mahdollista saada valmiiksi toteutettuna palveluna pilvipalveluntarjoajalta. Mitä enemmän suunnitellun

arkkitehtuurin komponentteja pystyttäisiin korvaamaan palveluntarjoajan tarjoamilla palveluilla, sitä vähemmän jäisi ohjelmoijan toteutettavaksi.

Sovelluksen korkean tason arkkitehtuurikaavio on esitetty kuviossa 8. On kuitenkin huomioitava, että tämä esitetty arkkitehtuuri ei pyri olemaan minkäänlainen referenssiarkkitehtuuri tieteellisten laskentasovellusten pilvimodernisointiprosesseja varten, vaan tässä tutkielmassa esitettyjä havaintoja havainnollistava esimerkki siitä, miten palvelukeskeinen arkkitehtuuri voidaan mallintaa yksinkertaisesti ja pilvipalveluntarjoajasta riippumattomasti.



KUVIO 8 Modernisoidun sovelluksen korkean tason arkkitehtuurikaavio

Kuviosta 8 nähdään, että laadittu arkkitehtuuri koostuu komponenteista, jotka tarjoavat tai kuluttavat palveluita. Osa palveluista viestii keskenään suoraan käyttäen palvelurajapintoja, mutta jotkin toiminnot on toteutettu viestinvälityspalvelua apuna käyttäen. Viestinvälitykseen päädyttiin siksi, että siihen oli olemassa valmiita ratkaisuja, joten sen käyttöönotto oli nopeaa ja vaivatonta.

Arkkitehtuuri ei syntynyt yhdellä kertaa lopulliseen muotoonsa, vaan sitä muokattiin prototypoinnin avulla saatujen havaintojen myötä. IaaS-pilvipalvelumallin edut havainnollistuvat hyvin viimeistään tässä vaiheessa, kun voidaan visuaalisesti nähdä, mitkä komponentit pystytään hankkimaan suoraan palveluntarjoajalta. Tässä tapauksessa viestinvälitys-, tietokanta-, ja tietovarastopalveluissa päätettiin hyödyntää AWS:n tarjoamia palveluita. Myös kuviossa esiintyvät virtuaalitietokoneet otetaan käyttöön luonnollisesti AWS:n EC2 IaaS -palvelun kautta. Mikäli palveluntarjoajalla olisi suppeampi valikoima valmiita palveluita, jouduttaisiin puuttuvat komponentit toteuttamaan itse. Perinnesovellukseen jo toteutetut palvelut voidaan integroida sovellukseen periaatteessa sellaisenaan palveluiden itsenäisyyden vuoksi. Jäljelle jäävät toiminnallisuudet pyritään tuomaan perinnesovelluksesta siinä määrin kuin se on järkevää. Yksityiskohtaisemmin aiheesta kerrotaan alaluvussa 6.4.

Yhteenvetona arkkitehtuurin määrittämisestä ja palveluntarjoajan valinnasta tieteellisen laskentasovelluksen modernisoinnin tapauksessa voidaan todeta, että kumpaakin tulee harkita aina tapauskohtaisesti, mutta tutkimusta tehtäessä havaittiin useita hyviä käytänteitä, jotka voivat osoittautua hyödyllisiksi vastaavanlaisissa tapauksissa. Mikäli modernisointi on päätetty toteuttaa jotain kirjallisuudessa esitettyä menetelmää käyttäen, on järkevää määrittää sovelluksen arkkitehtuuri kyseisen menetelmän keinoja noudattaen. Mikäli taas halutaan valita inkrementaalinen kehitysmalli, voidaan toimia kuten tässä tapauksessa ja laatia arkkitehtuuri itse käyttäen alustavana pohjana palvelukeskeistä arkkitehtuuria, jota kehitetään ja täsmennetään modernisoinnin edetessä. Tässä tapauksessa palveluntarjoajan valintaan keskityttiin kenties liian vähäisesti, koska sovelluksesta pyrittiin tekemään mahdollisimman siirrettävä. Mikäli palveluntarjoajan valintaa halutaan painottaa enemmän, voidaan valintaprosessissa soveltaa Chauhanin ja Babarin (2012) menetelmässä esitettyä vertailuviitekehystä.

6.4 Toteutus ja testaus

Tässä alaluvussa esitetään tutkijan tekemiä havaintoja tapaustutkimuksessa tehdyn modernisointiprosessin toteutusvaiheesta. Kirjallisuudessa esitetyt pilvimodernisointimenetelmät eivät juuri ota kantaa käytännön toteutukseen, joten havaintoja reflektoidaan pääasiassa modernisointiin yleisesti. Modernisointia pidetään yleensä ottaen haasteellisena prosessina (Menychtas ym., 2013; Mohagheghi ym., 2010; Seacord ym., 2003, ss. 1–2). Tässä alaluvussa nostetaan esiin havaintoja siitä, mitkä yleiset modernisointimenetelmät soveltuvat tämän tutkimuksen modernisointitapaukseen.

Tämän tapauksen yhteydessä oli mahdollista käyttää *lasilaatikkomodernisointia*, jossa modernisoinnin tekijällä on pääsy perinnesovelluksen sisäiseen toteutukseen (lähdekoodi mukaan lukien) (Seacord ym., 2003, s. 9). Mikäli pääsy sisäisen toteutuksen tarkasteluun olisi evätty ja perinnesovellusta voitaisiin tutkia vain syötteiden ja tulosteiden avulla, olisi kyse *musta laatikko -modernisoinnista* (Seacord ym., 2003, s. 9), joka todennäköisesti lisäisi huomattavasti lähdekoodin analysoinnin kompleksisuutta ja vähentäisi perinnesovelluksen lähdekoodin käyttömahdollisuuksia modernisoidussa sovelluksessa.

Perinnesovelluksesta ei haluttu tuoda läheskään kaikkea lähdekoodia modernisoituun sovellukseen, eli tuotavan lähdekoodin määrää pyrittiin vähentämään (Seacord ym., 2003, ss. 14–15) huomattavasti, jotta vuosien aikana kumuloituneesta ylimääräisestä painolastista (Bergman, 2012) päästäisiin eroon. Perinnesovelluksesta käyttökelpoisen koodin tunnistamiseksi suoritettiin lasilaatikkotarkastelua ja lähdekoodin analysointia. Tarkastelua ei suoritettu systemaattisesti missään tietyssä prosessin vaiheessa, vaan arviointi tehtiin aina tapauskohtaisesti, kun jotain komponenttia alettiin toteuttamaan. Tehtyä analysointia voitaisiin luonnehtia pintapuoliseksi, koska varsinaista dataa analysointivaiheesta ei kerätty. Tutkija käytti omaa harkintaansa siinä, pyrittäisiinkö käyttämään perinnesovelluksesta saatavilla olevaa lähdekoodia vai kirjoitettaisiinko kyseinen ohjelman osa käytännössä uudelleen.

Modernisoinnin alkuvaiheessa tehtiin päätös siitä, että sovelluksen webkomponentit (palvelinpää ja käyttöliittymä) tultaisiin uudistamaan kokonaan (lukuun ottamatta perinnesovellukseen toteutettuja palveluita), koska perinne-

sovelluksen huonon suunnittelun vuoksi nämä osiot sisälsivät hyvin vähän uudelleenkäytettäviä komponentteja ja erityisesti näiden osioiden koodi oli hyvin ”spagettimaista”. Simulaatiokirjasto taas oli täysin itsenäinen komponentti, ja se pystyttiin sen vuoksi tuomaan perinnesovelluksesta sellaisenaan.

Kun jonkin toiminnallisuuden tai komponentin kehityksessä tuli esiin monimutkaisia tai muuten vaikeita asioita, katsottiin perinnesovelluksen lähdekoodista, kuinka asia oli ratkaistu. Mikäli perinnesovelluksen ratkaisu todettiin käyttökelpoiseksi ja se sisälsi paljon lähdekoodia, jonka kirjoittaminen uudelleen olisi vienyt huomattavasti aikaa, siirrettiin kyseiset koodin osat käärintää (Seacord ym., 2003, s. 9) hyödyntämällä modernisoituun sovellukseen. Käärintä osoittautui erityisen käytännölliseksi sellaisten toimintojen yhteydessä, jotka oli toteutettu komponenteiksi tai jotka olivat suhteellisen helposti muokattavissa sellaisiksi. Komponentteja, jotka ottavat syötteitä sisäänsä ja tuottavat tulosteita, voidaan käsitellä musta laatikko -modernisoinnin avulla siten, että komponentin sisäistä toteutusta ei tarvitse selvittää eksplisiittisesti. Tässä tapauksessa esimerkiksi monimutkaisten parserointioperaatioiden yhteydessä oli huomattavasti mielekkäämpää tuoda kyseiset komponentit perinnesovelluksesta kuin toteuttaa niitä itse uudelleen.

SaaS-palvelumalli toteutettiin sovellukseen siten, että sovelluksen keskeisille toiminnoille määriteltiin ja laadittiin palvelurajapinnat ja toteutettiin näitä palveluita kutsuva käyttöliittymä. Ihmiskäyttäjä näkee sovelluksesta vain selainkäyttöliittymän, ja mahdolliset konekäyttäjät näkevät sovelluksen palvelurajapinnan. Käyttöliittymä ja rajapinnat kätkevät taakseen sovelluksen sisäisen arkkitehtuurin ja sen toteutuksen kompleksisuuden. Sovelluksen sisäiseen toteutukseen päätettiin soveltaa myös paljon käytettyä viestinvälityssuunnittelumallia, joka tukee palveluiden itsenäistä ja asynkronista periaatetta.

Pyrkimys tehdä sovelluksesta palveluntarjoajasta riippumaton otettiin huomioon erityisesti modernisoidun sovelluksen lähdekoodin kirjoittamisessa ja teknologioiden valinnassa. Ohjelmoinnissa käytettiin sellaisia tekniikoita, joiden avulla palveluntarjoajan palvelulle yksinomaiset piirteet saatiin rajattua mahdollisimman pieniin ja itsenäisiin komponentteihin. Näiden komponenttien suoran käytön sijasta laadittiin rajapinnat, jotka kyseiset komponentit toteuttavat. Näin palveluntarjoajasta riippuvaiset komponentit pystytään tulevaisuudessa korvaamaan uusilla komponenteilla, joiden tarvitsee vain toteuttaa määritellyt rajapinnat.

Erityisen hyödylliseksi havaittu tekniikka oli toteuttaa kaikki komponentit siten, että sovelluksen konfiguraatiotiedostosta riippuen sovelluksen kaikki komponentit voitiin ajaa joko paikallisella palvelimella, jolla kehitystä suoritettiin, tai vaihtoehtoisesti autenttisessa pilviympäristössä. Tämä tekniikka ohjasi kehitysprosessia siten, että sovelluksesta muodostui siirrettävä, minimaalisesti riippuvainen palveluntarjoajasta ja ennen kaikkea helposti testattava.

Sovelluksen modernisoinnin yhteydessä käytettiin *testivetoista kehitystapaa* (TDD) (Beck, 2003), jossa jokaiselle ohjelmistokomponentille laaditaan ennen ohjelmointia *yksikkötestit*, joiden avulla voidaan varmistua siitä, että toteutettava komponentti käyttäytyy halutulla tavalla. Testejä laatiessa havaittiin, että yksikkötestien toteuttaminen pilvipalveluympäristössä ei aina ole kovin yksiselitteistä. Mikäli yksikkötestit käyttävät palveluntarjoajan resursseja, voi testaamisesta koitua rahallisia kuluja. Testejä laadittaessa hyväksi käytännöksi havaittiin eritellä palveluntarjoajan resursseja käyttävät testit muista testeistä, jotta testien ajajalla olisi mahdollisuus olla ajamatta niitä tarpeen vaatiessa.

Seacord (2003, ss. 12–13) mainitsee kaupallisten komponenttien hyödyntämisen yhtenä mahdollisena sovelluksen uudistustoimenpiteenä. Tämän tapauksen yhteydessä tätä menetelmää sovellettiin siten, että hyödyntämisen kohteena oli avoimen lähdekoodin komponentteja. Esimerkiksi asynkronisten tehtävien suorittamiseen hyödynnettiin tässä tapauksessa olemassa olevaa ja kypsää avoimen lähdekoodin komponenttia. Komponenttien valinnassa kannattaa kuitenkin kiinnittää huomiota siihen, kuinka joustavia ne ovat muutosten suhteen, ja arvioida, kärsivätkö sovelluksen laatuattribuutit, kuten siirrettävyys, mahdollisesti kyseisten komponenttien käytöstä.

Sovelluksen käyttöliittymä päätettiin toteuttaa sovelluksen palveluiden kuluttajana. Tällöin käyttöliittymän ja sovelluksen välille ei pääse muodostumaan epäselviä riippuvuuksia ja käyttöliittymän kehittäminen edellyttää palveluiden kehittämistä, jolloin prosessit tukevat toisiaan, vaikka ovatkin erillisiä.

Sovellukselle tehtiin modernisoinnin yhteydessä myös *datan uudistustoimenpiteitä*, joilla tarkoitetaan tiedon varastoinnin, järjestyksen ja formaatin muokkaamista (Seacord ym., 2003, s. 16). Sovelluksen relaatiotietokantamallia paranneltiin ja tietokantaan talletettavien tietojen määrää kasvatettiin, jotta tulevaisuudessa saadaan laadittua monipuolisempia ja tehokkaampia kyselyitä dataan ja erityisesti datan ja palveluiden metatietoihin liittyen.

Taulukossa 6 on esitetty modernisoidun sovelluksen keskeisten komponenttien (vrt. kuvio 8) toteutustapa ja kommentoitu kunkin komponentin testattavuutta paikallista palvelinta käyttäen. Taulukosta voidaan nähdä, että neljä keskeistä komponenttia (viestinvälityspalvelu, relaatiotietokantapalvelu, tiedostovarasto ja virtuaalikonepalvelu) hankittiin palveluna palveluntarjoajalta. Tämä vähensi huomattavasti itse kirjoitettavan ja ylläpidettävän ohjelmakoodin määrää. Myöskään näiden palveluiden infrastruktuurista tai muista resursseista ei tarvitse huolehtia. Palveluiden käyttöönotto toteutettiin ohjelmakoodissa siten, että riippuvuus nimenomaiseen palveluun olisi rajoitettu mahdollisimman pieneen ja korvattavaan osaan sovellusta. Tämä mahdollistaa sen, että sovellusta voidaan testata kokonaisuudessaan paikallista palvelinta käyttäen. Testattaessa palvelut korvataan käyttämällä paikallisesti ajettavia toteutuksia palveluista.

TAULUKKO 6 Modernisoidun sovelluksen komponenttien toteutus ja testattavuus

Komponentti	**Toteutustapa tai -tavat**	**Testattavissa paikallisesti**
Käyttöliittymä	Uuden luonti (JavaScript)	Kyllä
Sovelluksen palvelut	Modernisointi nykyisestä sovelluksesta ja uuden luonti (Python)	Kyllä
Tehtävien (tasks) suorittajat (worker)	Avoimen lähdekoodin Python-kirjaston hyödyntäminen (Celery)	Kyllä
Viestinvälityspalvelu	Hankittu palveluna (Amazon SQS)	Korvattavissa esim. Celeryn tukemalla Sqlite-pohjaisella viestinvälityksellä
Simulaatiokirjasto	Modernisointi nykyisestä sovelluksesta (C++, Python)	Kyllä
Relaatiotietokanta	Hankittu palveluna (Amazon RDBMS), tietorakenne modernisoitu nykyisestä järjestelmästä	Kyllä, esim. käyttämällä Sqlite:ä
XML-tietokanta	Ei vielä toteutettu	-
Tiedostovarasto	Hankittu palveluna (Amazon S3)	Kyllä, käyttämällä paikallista tiedostojärjestelmää
Virtuaalikoneet	Hankittu palveluna (Amazon EC2)	Ei yleensä tarvetta. Mahdollista esim. VirtualBox-virtualisoinnin avulla

Yhteenvetona modernisoinnin toteutuksen yhteydessä esiin nousseista havainnoista voidaan todeta, että monet Seacordin (2003) mainitsemista modernisointimenetelmistä, kuten käärintä, modularisointi ja lähdekoodin vähentäminen, soveltuvat hyvin myös tässä tutkimuksessa käytettyyn kevyempään ja iteratiiviseen pilvimodernisoinnin toteutustapaan. Tämän lisäksi tutkija teki joukon omia havaintoja siitä, millaiset menettelytavat koettiin erityisen hyödyllisiksi. Jokainen pilvimodernisointiprojekti on kuitenkin yksilöllinen, joten tarvittaisiin havaintoja myös muista tapauksista, jotta käytetyn menetelmän ja käytänteiden menestyksekkyydestä voitaisiin saada vakuuttavampaa näyttöä.

6.5 Katselmointi

Modernisointiprosessin etenemistä arvioitiin katselmoinneissa, joita järjestettiin sopivin aikavälein modernisoinnin toteuttajan, johtavan tutkijan sekä nykyisen sovelluksen ylläpitäjän välillä. Katselmointien yhteydessä voitiin tarvittaessa muokata modernisoitavan sovelluksen vaatimuksia, jotka huomioitaisiin seuraavassa iteraatiossa.

Usein ketterissä menetelmissä, kuten XP:ssä (Agarwal & Umphress, 2008), korostetaan tiivistä kehitystiimin ja asiakkaan välistä kommunikaatiota. Monesti on suositeltavaa, että sovelluksen tilaajan taholta joku henkilö on fyysisesti

läsnä kehittäjäorganisaation tiloissa. Tämän tutkimuksen tapauksessa suuri maantieteellinen etäisyys tahojen välillä teki tämän tavoitteen toteutumisen käytännössä mahdottomaksi, mutta kommunikaatiota pidettiin yllä tiiviisti sähköpostin ja Skypen välityksellä.

Katselmoinneissa käytettiin apuna Skype-sovellusta, jossa tutkija pystyi demonstroimaan toteutettuja toiminnallisuuksia ääni- ja videoyhteyden avulla. Tutkija jakoi tietokoneensa työpöydän muiden asianomaisten nähtäville ja esitteli sovelluksen toimintaa ajamalla autenttista sovellusta tai sen prototyyppiä. Nykyisen sovelluksen ylläpitäjä sekä johtava tutkija antoivat huomioita ja kommentteja toteutuksesta. Näiden huomioiden ja kommenttien pohjalta käytyjen keskustelujen avulla päästiin usein yksimielisyyteen vaatimusten priorisoinnista ja mahdollisista muutoksista.

Modernisoinnin jälkeen tehdyssä teemahaastattelussa (vrt. liite 2) todettiin, että Skype-palaverit olivat tehokkaita,, mutta informaation vaihto olisi voinut nopeutua, mikäli työntekijä ja työnantaja olisivat voineet tavata päivittäin. Yleisesti ottaen Skype havaittiin hyväksi ratkaisuksi katselmointien toteuttamiseen, kun maantieteellinen etäisyys on liian suuri päivittäisten tapaamisten järjestämiseksi.

6.6 Loppuarviointi

Modernisointiprosessin loppuarvioinnissa arviointikriteereinä käytetään funktionaalisten vaatimusten ja laatuvaatimusten toteutumista sekä teemahaastatteluiden pohjalta johtavan tutkijan ja nykyisen sovelluksen ylläpitäjän tyytyväisyyttä modernisointiprosessin ja sen lopputulokseen. Aluksi modernisointiprosessia arvioidaan funktionaalisten vaatimusten ja laatuvaatimusten toteutumisen perusteella. Laatuvaatimuksissa tarkastellaan erityisesti modernisoidun sovelluksen ylläpidettävyyttä, koska nykyisen sovelluksen suurimmat ongelmat liittyvät siihen. Lopuksi arvioidaan modernisoinnin onnistumista kokonaisuutena.

Perinnesovelluksen tapauksessa huono suunnittelu oli johtanut sovelluksen merkittävään rapautumiseen. Vaikka modernisoinnin yhteydessä sovellukselle pyrittiin luomaan ehyempi, modulaarisempi ja selkeämpi arkkitehtuuri, on silti tärkeää, että myös sovelluksen ylläpitoon kiinnitetään jatkossa huomiota. Ylläpidettävyys on yksi modernisoidun sovelluksen vaatimuksista, ja tässä alaluvussa käsitellään sitä, mitä sillä oikein tarkoitetaan.

Grubb ja Takang (2003) määrittelevät ylläpidettävyyden ylläpitotehtävien suorittamisen helppoudeksi. Ylläpidettävyyden tärkeimmät tekijät ovat sovelluksen tarkoituksenmukaisuus, oikeellisuus, siirrettävyys, testattavuus, käytettävyys, tehokkuus, yhtenäisyys, uudelleenkäytettävyys ja yhteentoimivuus. Tässä arvioinnissa näistä attribuuteista korostuvat erityisesti testattavuus, uudelleenkäytettävyys, siirrettävyys ja oikeellisuus, koska näihin seikkoihin liittyvät ongelmat nousivat esiin ensimmäisessä teemahaastattelussa (vrt. liite 1).

Seacordin (2003, s. 4) mukaan ylläpitotoimet keskittyvät yleensä *parantelevaan ylläpitoon*, jossa sovellukseen lisätään uusia toiminnallisuuksia ja parannetaan suorituskykyä sekä käytettävyyttä. Myös tämän tutkimuksen yhteydessä ilmenneet havainnot perinnesovelluksesta tukevat tätä väitettä. Parantelevaan ylläpitoon keskittynyt ylläpito tarkoittaa käytännössä sitä, että korjaavalle, mukautuvalle ja ennaltaehkäisevälle ylläpidolle jää paljon vähemmän huomiota.

Tämä johtaa helposti sovelluksen rapautumiseen, kun sovellukseen kehitetään lisää uusia toiminnallisuuksia, eikä aikaa sovellusta paranteleville ja ongelmia ennaltaehkäiseville toimille jää paljoa.

Ylläpidettävyyden arviointi on toistaiseksi mahdotonta, koska sovelluksen ylläpidosta ei ole vielä ehtinyt kertymään kokemuksia. Ylläpidettävyyteen vaikuttavia seikkoja on kuitenkin pyritty huomioimaan koko modernisointiprosessin ajan. Sovelluksen ylläpidettävyyttä on pyritty lisäämään perinnesovellukseen verrattuna käyttämällä selkeää ja dokumentoitua arkkitehtuuria, dokumentoimalla lähdekoodia, tekemällä sovelluksesta helposti testattava ja käyttämällä versionhallintajärjestelmää. Pyrkimyksenä on, että sovellukseen olisi luontevaa tuoda uusia toiminnallisuuksia ja vanhoja ominaisuuksia voitaisiin tarpeen tullen muokata ilman pelkoa sovelluksen rikkoutumisesta. Erityisesti testattavuuden havaittiin lisäävän kehittäjän luottamusta siihen, että sovellus toimii vielä toteutettujen muutosten jälkeenkin. Palvelukeskeisen arkkitehtuurin pitäisi tehdä sovelluksesta helposti liitettävä muihin palveluihin ja sovelluksiin.

Yhteenvetona ylläpidettävyydestä tehdyistä havainnoista voidaan todeta, että sovelluksen testattavuus on tärkeää myös pilvipalveluiden tapauksessa. Testien laatiminen ei välttämättä ole täysin yksiselitteistä pilviympäristössä, mutta se on mahdollista ja koettiin hyödylliseksi ainakin tämän tutkimuksen tapauksessa. Testattavuus lisää kehittäjän luottamusta suorittaa ylläpitotoimia. Ylläpitotoimissa tulisi keskittyä parantelevan ylläpidon lisäksi myös korjaavaan, mukautuvaan ja ennaltaehkäisevään ylläpitoon, jotta sovelluksen rapautumista voitaisiin ehkäistä.

Tapaustutkimuksessa tehdyn tieteellisen laskentasovelluksen pilvimodernisoinnin onnistumisen arvioimiseksi perinnesovelluksen ylläpitäjälle ja johtavalle tutkijalle tehtiin toinen teemahaastattelu, jossa pyydettiin arvioimaan modernisoinnin onnistumista. Haastattelun arvoa vähentää jonkin verran se, että modernisoitua sovellusta ei ollut vielä otettu käyttöön todellisessa ympäristössä, eikä vanhan sovelluksen kaikkea dataa ollut tuotu uudistettuun sovellukseen. Modernisoidun sovelluksen käyttökokemukset rajoittuivat sillä hetkellä haastateltujen henkilöiden osalta sovelluksen koekäyttöön. Haastateltavat olivat koekäyttäneet sovellusta pilvessä ja tarkastelleet sen dokumentaatiota sekä lähdekoodia.

Sovelluksen modernisointiprosessiin ja -tulokseen oltiin haastattelun (vrt. liite 2) perusteella pääasiassa tyytyväisiä. Kaikki yhdessä määritellyt toiminnalliset vaatimukset saatiin toteutettua kolmessa kuukaudessa tutkijan toimesta, joka oli siis ainoa modernisointiin osallistunut suunnittelija ja ohjelmoija. Johtava tutkija oli positiivisesti yllättynyt toteutettujen toiminnallisuuksien lukumäärästä. Laatuvaatimusten osalta modernisoinnin onnistumista voidaan arvioida vasta, kun sovellus on otettu käyttöön ja siitä on saatu realistisia käyttökokemuksia käyttäjien ja ylläpidon osalta. Laatuvaatimusten toteutumisen todentamiseksi tässä vaiheessa täytyisi suorittaa empiirisiä kokeita, mutta tämä vaatisi liikaa resursseja. Laatuvaatimusten toteutumiseen on kuitenkin pyritty kiinnittämään erityistä huomiota koko modernisointiprosessin ajan, mikä lisää luottamusta siihen, että laatuvaatimukset toteutuvat.

6.7 Havaintoja modernisointikäytänteistä

Tässä alaluvussa esitetään tutkimusprosessissa tehtyjen havaintojen pohjalta laadittu ehdotus parhaiksi käytänteiksi tapauksissa, joissa tieteellinen laskentasovellus modernisoidaan pilvipalveluksi käyttäen kevyttä ja inkrementaalista toimintatapaa:

- Puntaroi modernisoinnin motiiveja ennen päätöksentekoa.
- Selvitä, mitkä seikat ovat johtaneet modernisoinnin tarpeellisuuteen ja pohdi, miten ne voitaisiin välttää tulevaisuudessa.
- Punnitse pilvipalveluiden mahdollisuuksia ja heikkouksia kyseisen sovelluksen kannalta, ja ota huomioon myös muut mahdolliset toteutusvaihtoehdot. Älä suosi pilvipalveluita vain siksi, että kaikki muutkin käyttävät niitä.
- Mikäli pilvipalvelumalli(t) tarjoaa merkittäviä etuja sovelluksen nykyiseen toteutukseen verrattuna, arvioi, miten suuria muutoksia perinnesovellukseen tulee tehdä, jotta näistä eduista päästään hyötymään. Edellyttävätkö muutokset suurten kompromissien tekemistä?
- Modernisoinnissa, kuten muussakin ohjelmistokehityksessä, vaatimukset voivat muuttua. Sovella siis modernisointiin iteratiivisia menetelmiä, jotka ovat valmiita reagoimaan muutoksiin.
- Opi perinnesovelluksessa tehdyistä virheistä. Varmista vaatimusmäärittelyssä, että perinnesovelluksessa havaitut ongelmat korjataan siten, että ne eivät koidu enää koskaan tulevaisuudessa ongelmaksi.
- Pyri tekemään sovelluksen arkkitehtuurista riippumaton tietystä palveluntarjoajasta.
- Toteuta riippuvuus palveluntarjoajaan ennemmin ohjelmisto- kuin arkkitehtuuritasolla. Ohjelmistokomponentteja voi vaihdella konfiguraation mukaan, arkkitehtuuria yleensä ei.
- Mikäli modernisoinnin toivottuna tuloksena on SaaS-sovellus, luo sovelluksen vaatimusten pohjalta sovellukselle palvelukeskeinen arkkitehtuuri ja seuraa palvelukeskeisten arkkitehtuurien suunnittelumalleja ja hyviä käytänteitä.
- Käytä yleisesti käytettyjä suunnittelumalleja (patterns) aina, kun se on mahdollista. Tällöin löydät todennäköisesti myös valmiita ratkaisuja kohtaamiisi ongelmiin.
- Pyri toteuttamaan sovellus siten, että pystyt ajamaan sovelluksen sekä paikallisella palvelimella että pilviympäristössä sovelluksen konfiguraatiota muuttamalla. Näin sovelluksesta tulee siirrettävä ja helposti testattava.
- Ennen kuin alat toteuttaa jotain loogista kokonaisuutta, tarkista, olisiko se mahdollisesti korvattavissa jollain olemassa olevalla kaupallisella tai avoimen lähdekoodin komponentilla tai palvelulla.
- Valitse palveluntarjoaja, joka mahdollistaa palveluiden kokeilun hyvin edullisesti, mieluiten ilmaiseksi.
- Laadi yksikkötestit myös niille komponenteille, jotka käyttävät palveluntarjoajan palveluita, mutta käytä jotain mekanismia näiden testien erottamiseksi muista testeistä.

Yllä mainitut käytänteet ovat mahdollisesti sovellettavissa myös muihin tapauksiin, mutta tämän väitteen tueksi tarvitaan jatkotutkimuksia.

6.8 Yhteenveto

Tässä luvussa esiteltiin tapaustutkimuksen tulokset, jotka muodostuivat pääasiassa tutkijan tekemistä havainnoista tutkimusprosessin aikana. Näiden lisäksi kerrottiin ennen ja jälkeen modernisoinnin tehdyistä teemahaastatteluista. Tulokset käsiteltiin tutkimusprosessin mukaisessa järjestyksessä (vrt. kuvio 7), ja lopuksi esitettiin tutkijan tekemiä havaintoja modernisointikäytänteistä.

Modernisointimenetelmän valinnasta, vaatimusmäärittelystä ja analyysista, suunnittelusta sekä toteutuksesta ja testauksesta saatiin paljon havaintoja tutkimusprosessin edetessä. Katselmoinnin osalta havainnot jäivät niukemmiksi. Keskeisimmät havainnot koottiin tiivistetysti listaukseksi, joka toimii ehdotuksena parhaista käytänteistä tapauksissa, joissa tieteellinen laskentasovellus modernisoidaan pilvipalveluksi käyttäen kevyttä ja inkrementaalista toimintatapaa.

Modernisointiprosessin arviointi osoittautui vielä tässä vaiheessa osittain haastavaksi. Toteutettujen toiminnallisuuksien osalta arviointi on suoraviivaista, koska toimintojen toteutuminen ja oikeellisuus voidaan varmentaa laadittujen yksikkötestien avulla, mutta laatuvaatimusten osalta arviointi on haastavaa, koska käytännön kokemuksia sovelluksesta on kertynyt toistaiseksi hyvin vähän. Jälkimmäisen haastattelun perusteella modernisointiprosessin tuloksiin oltiin pääasiassa erittäin tyytyväisiä.

7 POHDINTA

Tässä luvussa tarkastellaan tapaustutkimuksen suorittamista ja tuloksia. Aluksi tarkastellaan tutkimustuloksia ja niiden pohjalta tehtäviä johtopäätöksiä. Toiseksi arvioidaan tapaustutkimuksen reliabiliteettia ja validiteettia. Lopuksi pohditaan tulosten hyödyntämismahdollisuuksia ja esitetään jatkotutkimusaiheita.

7.1 Tutkimustulokset ja johtopäätökset

Tässä alaluvussa käydään läpi tutkimuksen tuloksia aluksi esittelemällä hyvin lyhyesti tapaustutkimuksen kohde ja sen jälkeen vastaamalla luvun 5.4 lopussa esitettyihin tapaustutkimusta koskeviin kysymyksiin. Vastausten pohjalta pyritään esittämään johtopäätöksiä.

Tämän tutkimuksen tapauksessa modernisoitiin eräs tieteellinen laskentasovellus pilvipalveluksi. Tutkimus toteutettiin konstruktiivisena tapaustutkimuksena, jossa tiedonkeruumenetelminä toimivat havainnointi ja haastattelut. Modernisoinnin kohteena olevan sovelluksen käyttö ja ylläpito koettiin sitä käyttävien tutkijoiden sekä sovelluksen ylläpitäjien toimesta haasteelliseksi, ja he kaipasivat asiaan parannusta.

Modernisointipäätös tehtiin ensimmäisessä teemahaastattelussa (vrt. liite 1) esiin nousseiden, nykyisen sovelluksen puutteiden ja virheiden vuoksi. Nykyisen sovelluksen lähdekoodin tarkastelu vahvisti näiden ongelmien olemassaolon. Perinnesovelluksen modernisoinnin (Seacord ym., 2003) menetelmiä ja tekniikoita päätettiin hyödyntää, koska kaikkea nykyisen sovelluksen toiminnallisuutta ei olisi järkevää rakentaa alusta asti uudelleen.

Päätös modernisoida sovellus pilvipalveluksi oli tässä tapauksessa hyvin selkeä. IaaS-palveluiden hyödyntämisestä ja SaaS-pilvipalvelumallin toteuttamisesta koituisi merkittäviä etuja, ja niiden avulla saataisiin ratkaistua useita nykyisessä sovelluksessa olevia ongelmia sekä laajennettua sovelluksen käyttömahdollisuuksia, kuten on todettu alaluvussa 6.2. Osa nykyisen sovelluksen ongelmista johtui tieteellisen laskennan edellyttämästä suuresta resurssitarpeesta ja osa sovelluksen toteutukseen liittyvistä virheistä. Keskeisinä etuina pilvipalveluiden hyödyntämisessä nähtiin infrastruktuurin ulkoistamisesta seuraava ylläpidon helpottuminen (Brian ym., 2012), resurssien nopea joustavuus (Mell & Grance, 2009), käyttöön perustuva hinnoittelu (Brian ym., 2012) sekä palvelu-

keskeisen arkkitehtuurin mahdollistama yhteentoimivuus muiden palveluiden ja sovellusten kanssa (Su ym., 2011). Pilvipalveluiden korkea saavutettavuus koettiin myös positiiviseksi asiaksi sovelluksella tehtävän tieteellisen tutkimuksen läpinäkyvyyden kannalta.

Johtopäätöksenä modernisoinnin motiiveista voidaan todeta, että motiivit tieteellisen laskentasovelluksen modernisoimiseksi pilvipalveluksi ovat usein – kuten tässäkin tapauksessa - hyvin selkeät. Tieteellisen laskennan perusta on numeerisissa menetelmissä, joiden ratkaisut edellyttävät monesti hyvin raskasta laskentaa (Huang ym., 2010; Srirama ym., 2013; Vecchiola ym., 2009), jonka suorittamiseen tarvitaan runsaasti resursseja, joita pilvipalvelut tarjoavat hyvin edullisilla hinnoilla (Zhang ym., 2010). Tämän tutkimuksen tapauksessa perinnesovellus oli tämän lisäksi rapautumisen vuoksi vaikeasti ylläpidettävä, mikä lisäsi modernisoinnin motivaatiota.

Amazon Web Services ("AWS | What is AWS - Cloud Computing with Amazon Web Services", 2014) valittiin palveluntarjoajaksi tähän tapaukseen siitä syystä, että vaatimuksissa ei esitetty rajoitteita palveluntarjoajan suhteen ja tutkijalla oli aiempaa kokemusta AWS:n pilvipalveluista. Johtopäätöksenä palveluntarjoajan valinnasta voidaan todeta, että palveluntarjoajan valintaa voidaan painottaa enemmän tai vähemmän riippuen siitä, miten kriittiseksi valinta koetaan. Tapaustutkimuksessa kehitetyllä toimintatavalla pyrittiin toteutukseen, joka on mahdollisimman vähän riippuvainen palveluntarjoajasta, ja siinä onnistuttiin tutkijan näkemyksen mukaan erittäin hyvin. Tällainen toimintatapa edistää sovelluksen siirrettävyyttä. Mikäli palveluntarjoajan valinta koetaan kriittiseksi, voidaan valinnassa soveltaa Chauhanin ja Babarin (2012) laatimaa vertailuviitekehystä.

Kirjallisuudessa on esitelty menetelmiä, kuten mallivetoiset REMICS- (Mohagheghi ym., 2010) ja ARTIST-menetelmä (Menychtas ym., 2013) sekä Chauhanin ja Babarin (2012) menetelmä, perinnesovelluksen modernisoimiseksi pilveen. Nämä menetelmät havaittiin liian raskaiksi tämän tutkimuksen kaltaiseen tapaukseen. Iteratiivisuuden puuttuminen on kirjallisuudessa esitettyjen pilvimodernisointimenetelmien suurin heikkous, koska siitä johtuen kyseiset menetelmät ovat hyvin vaikeasti sovitettavissa mm. ketterän kehityksen periaatteisiin (Beck ym., 2001).

Modernisointi päätettiin tehdä käyttäen tutkijan itsensä soveltamaa toimintatapaa, joka yhdistelee ketteriä periaatteita (Beck ym., 2001), testivetoisuutta (Beck, 2003) ja Lean-periaatteita sovellettuna ohjelmistokehitykseen (Poppendieck & Poppendieck, 2003). Toimintatapaa on kuvattu tarkemmin tutkimusprosessin yhteydessä alaluvussa 5.4 ja tutkimuksen tuloksissa luvussa 6. Kaikki sovelluksen modernisoinnin osapuolet kokivat valitun toimintatavan toimivaksi ja tehokkaaksi, ja toimintatavalla saavutettuihin tuloksiin oltiin pääasiassa erittäin tyytyväisiä (vrt. liite 2). Käytetyn toimintatavan menestyksekkyydestä muissa tapauksissa ei ole varmuutta ennen kuin on tehty lisää tutkimusta aiheesta.

Johtopäätöksenä menetelmän valinnasta ja pilvimodernisoinnin käytännön toteutuksesta voidaan todeta, että mikäli prosessi halutaan toteuttaa ketterästi, voi kirjallisuudessa esitettyjen menetelmien (Chauhan & Babar, 2012; Menychtas ym., 2013; Mohagheghi ym., 2010) hyödyntäminen osoittautua hyvin hankalaksi. Tämä tapaustutkimus antaa näyttöä siitä, että ketterää ja inkrementaalista toimintatapaa voi soveltaa perinnesovelluksen modernisointiin pilvipalveluksi tieteellisen laskentasovelluksen tapauksessa. Tutkimus antaa myös näyttöä siitä, että yleiset modernisoinnin menetelmät (Seacord ym., 2003) sovel-

tuvat myös kevyeen ja inkrementaaliseen toimintatapaan. Näiden väitteiden varmistamiseksi tarvitaan kuitenkin lisää tutkimuksia.

Tutkimusprosessin aikana havaittiin useita tässä tapauksessa hyviksi osoittautuneita käytänteitä. Käytänteet pyrittiin kirjaamaan yleistettävämmässä muodossa ehdotetuksi listaukseksi parhaista käytänteistä perinnesovelluksen modernisoimiseksi pilvipalveluksi käyttäen kevyttä ja inkrementaalista toimintatapaa. Listaus on esitetty alaluvussa 6.7. Esitettyjen käytänteiden toimivuudesta muissa tapauksissa ei ole toistaiseksi näyttöä. Käytänteiden soveltuvuudesta muihin tapauksiin tarvitaan lisää tutkimuksia, mutta laadittua listaa voidaan käyttää esimerkiksi alustavana pohjana tulevissa tutkimuksissa.

Pilvipalveluissa sovellusten arkkitehtuuri perustuu lähes poikkeuksetta palvelukeskeiseen arkkitehtuuriin (Chauhan & Babar, 2012). Tästä syystä myös tässä tapauksessa sovellukseen toteutettiin palvelukeskeinen arkkitehtuuri. Itsenäisiin komponentteihin perustuvan palvelukeskeisen arkkitehtuurin (Bloomberg, 2013) havaittiin selkeyttävän sovelluksen rakennetta tekemällä siitä modulaarinen. Modulaarisuuden puolestaan havaittiin tekevän sovelluksesta helpommin siirrettävä ja testattava. Sovelluksen lopullinen arkkitehtuuri ei syntynyt kerralla, vaan sitä kehitettiin iteraatioissa tehtyjen havaintojen ja lisääntyneen tietämyksen avulla. Arkkitehtuuriin sovellettiin myös paljon käytettyä viestinvälitysmallia. Tässä tapauksessa sovelluksen arkkitehtuurista pyrittiin tietoisesti tekemään riippumaton yksittäisen palveluntarjoajan palveluista.

Johtopäätöksenä arkkitehtuurin määrittämisen osalta voidaan todeta, että sovelluksen arkkitehtuuri tulee suunnitella aina tapauskohtaisesti. Toisaalta myös hyviksi havaittuja käytänteitä nousi esille tapaustutkimuksen yhteydessä. Pilvipalveluiden tapauksessa havaittiin, että sovelluksen arkkitehtuuri kannattaa suunnitella siten, että se on riippumaton yksittäisen palveluntarjoajan palveluista. Riippuvuus palveluntarjoajaan on järkevää toteuttaa yksittäisen komponentin toteutuksen tasolle. Näin arkkitehtuuri säilyy riippumattomana palveluntarjoajasta ja sovellus on paremmin siirrettävissä toisen palveluntarjoajan palveluun. Kyseisen toimintatavan havaittiin myös helpottavan sovelluksen testaamista.

Tässä tapauksessa modernisointiprosessissa käytetyssä toimintatavassa on huomattavia eroja verrattuna ARTIST- (Menychtas ym., 2013), REMICS- (Mohagheghi ym., 2010) ja Chauhanin ja Babarin (2012) menetelmiin. Merkittävin ero muihin menetelmiin nähden on valitun toimintatavan inkrementaalisuus, mikä tekee mahdolliseksi toimintatavan sovittamisen ketterän kehityksen periaatteisiin (Beck ym., 2001). Yhteistä kirjallisuudessa esitetyissä menetelmissä ja tämän tapauksen toimintatavassa on ainoastaan vaatimusmäärittely prosessivaiheena sekä palvelukeskeisen arkkitehtuurin toteuttaminen sovellukseen. Chauhanin ja Babarin (2012) esittämää palveluntarjoajien vertailun viitekehystä voitaisiin mahdollisesti soveltaa myös kevyempään ja inkrementaaliseen toimintatapaan, mikäli palveluntarjoajan valinta koetaan erityisen kriittiseksi tekijäksi pilvimodernisoinnissa. REMICS- (Mohagheghi ym., 2010) ja ARTIST-menetelmissä (Menychtas ym., 2013) käytettyä mallivetoista lähestymistapaa modernisointiin voidaan myös mahdollisesti soveltaa inkrementaaliseen toimintatapaan, mutta yksittäistapauksissa se ei ole raskautensa vuoksi järkevää.

Johtopäätöksenä kirjallisuudessa esitetyistä menetelmistä suhteessa tässä tutkimuksessa käytettyyn toimintatapaan voidaan todeta, että kirjallisuudessa esitettyjen menetelmien (Chauhan & Babar, 2012; Menychtas ym., 2013; Mohagheghi ym., 2010) havaittiin soveltuvan huonosti ketterän kehityksen periaatteisiin (Beck ym., 2001). Kyseisistä menetelmistä voidaan kuitenkin mahdollisesti

hyödyntää osia myös tämän tutkimuksen kaltaisessa kevyessä ja inkrementaalisessa toimintatavassa.

Tutkimuksessa oletettiin ja myöhemmin myös havaittiin, että tieteellisen laskentasovelluksen tapauksessa modernisoinnin ja pilvipalvelumallin liiketoiminnalliset seikat ovat toissijaisia. Tästä syystä tässä tutkimuksessa tehtiin rajaus siten, että liiketoiminnallisia seikkoja käsiteltiin hyvin niukasti. Tutkijat haluavat usein myös välttyä turhalta byrokratialta, johon he monesti suurissa tutkimuslaitoksissa törmäävät (Tsaftaris, 2014). Tutkijat toimivat usein niukoilla resursseilla, eikä heillä yleensä ole tarvittavaa osaamista monimutkaisten migraatioprosessien hallitsemiseksi (Srirama ym., 2013). Näistä syistä johtuen tehtiin johtopäätös, että tieteellisen laskentasovelluksen modernisoimiseksi pilvipalveluksi olisi tarvetta helposti lähestyttävälle ja kevyelle menetelmälle, jonka avulla tieteentekijät pääsisivät mahdollisimman helposti hyötymään pilvipalvelumallien eduista.

Tieteellisen laskentasovelluksen vaatimusmäärittelyn ei havaittu poikkeavan merkittävästi muusta vaatimusmäärittelystä. Tieteellisille laskentasovelluksille on tyypillistä suuri laskenta- ja tallennusresurssien tarve (Tsaftaris, 2014; Vecchiola ym., 2009), mikä yhdessä aiemmin esitettyjen oletusten tutkijoiden resurssien niukkuudesta ja kevyen ja yksinkertaisen menetelmän toiveesta muodosti tässä tutkimuksessa tieteellisen laskentasovelluksen erityiset vaatimukset.

Nykyisen sovelluksen johtava tutkija ja ylläpitäjä olivat jälkimmäisen haastattelun perusteella (vrt. liite 2) erittäin tyytyväisiä modernisointiprosessiin ja sen tuloksiin. Ainoa esille noussut kehitettävä seikka oli päivittäiset tapaamiset sovelluksen kehittäjän ja tilaajan välillä tässä tutkimuksessa tarpeen mukaan käytyjen Skype-palavereiden ja -katselmointien sijasta.

7.2 Reliabiliteetti ja validiteetti

Runesonin ja Höstin (2009) mukaan tapaustutkimus soveltuu tutkimusmenetelmänä erinomaisesti tietojärjestelmätieteen ympäristöihin, joissa teoriat ovat vasta muotoutumassa. Tapaustutkimuksessa reliabiliteetin ja validiteetin vaatimukset eivät välttämättä päde samalla tavalla kuin kvantitatiivisessa tutkimuksessa (Hirsjärvi ym., 2010), mutta niitä pyritään silti arvioimaan.

Reliabiliteetti mittaa tutkimuksen tulosten toistettavuutta. Tekemällä tutkimus uudelleen samoja mittareita käyttämällä tulisi saada samat tulokset (Järvinen & Järvinen, 2011, ss. 161–162). Runesonin ja Höstin (2009, s. 154) mukaan tutkimuksen tulosten ei pitäisi riippua tutkijasta, vaan muidenkin tulisi voida toistaa tutkimus ja päätyä samoihin tuloksiin. Tämä edellyttää tutkimusprosessin läpinäkyvyyttä ja tarkkaa dokumentointia (Yin, 1994). Yksittäistä tapaustutkimusta on mahdotonta toistaa, mutta toistettavuutta on pyritty helpottamaan dokumentoimalla tutkimusprosessi ja johtopäätösten perustelut mahdollisimman tarkasti. Tämän tutkimuksen reliabiliteettia heikentää se tosiasia, että havainnot pohjautuvat yksittäisen tutkijan tekemiin huomioihin käsitellyistä aiheista. Havaintojen relevanttiutta ja todenperäisyyttä pyrittiin vahvistamaan reflektoimalla näitä havaintoja kyseistä aihetta käsittelevään tieteelliseen kirjallisuuteen, sekä tutkimuksen aluksi ja lopuksi tehdyillä teemahaastatteluilla. Haastattelujen avulla hankittiin perusteluja modernisointiprosessin aloittamiselle ja menetelmän valinnalle sekä arvioitiin prosessin lopputulosta. Haastateltaviksi valittiin henkilöt, joilla on paras asiantuntemus modernisoinnin kohtee-

na olleesta sovelluksesta. Haastattelut kärsivät kuitenkin reliabiliteettiongelmista, koska haastattelut olivat tutkijan itsensä laatimia, toteuttamia ja analysoimia. Tällöin on vaarana se, että haastateltavat antavat ns. ”oikeita” vastauksia. Tässä tutkimuksessa pyrittiin kuitenkin ensisijaisesti synnyttämään uutta tietoa konstruktion laatimisessa syntyvien havaintojen kautta, eikä täydelliseen reliabiliteettiin ollut pyrkimystä. Havaintojen vahvistamiseksi tarvitaan joka tapauksessa lisää tutkimuksia. Nämä seikat on pyritty huomioimaan tutkimuksen tuloksissa ja erityisesti tulosten pohjalta tehtävissä johtopäätöksissä.

Validiteetti mittaa tutkimuksen pätevyyttä. Käytännössä validiteetti ilmaisee, miten hyvin tutkimus mittaa sitä, mitä sen avulla on tarkoitus selvittää (Hirsjärvi ym., 2010). Kvalitatiivisessa tutkimuksessa ollaan validiteetin kannalta kiinnostuneita siitä, ovatko havaitut ilmiöt yhteensopivia ilmiöille esitettyjen selitysten kanssa. Validiteettia voidaan arvioida rakennevaliditeetin, sisäisen validiteetin ja ulkoisen validiteetin avulla (Yin, 1994).

Rakennevaliditeetin tarkastelu on tapaustutkimuksessa ongelmallista, koska tapaustutkimuksessa tutkimusaineiston keruu ja tutkimuksen toteuttaminen perustuvat tutkijan subjektiivisiin päätelmiin, eikä yksinkertaisia mittareita yleensä ole käytettävissä (Yin, 1994). Rakennevaliditeetin lisäämiseksi on olemassa kolme keinoa:

- usean tietolähteen käyttäminen, eli triangulaatio
- tutkimusvaiheiden raportointi
- tutkimusraportin luetuttaminen asiantuntijoilla (Yin, 1994).

Tässä tutkimuksessa on hyödynnetty kahta ensimmäistä rakennevaliditeetin kasvattamisen keinoa. Tutkimuksen teoreettiseksi pohjaksi laadittiin kattava kirjallisuuskatsaus käsiteltyihin aihepiireihin. Havaittuja ilmiöitä reflektoitiin kirjallisuuteen ja verrattiin kirjallisuudessa esitettyihin malleihin. Myös haastatteluita käytettiin tietolähteenä. Eräänä haasteena rakennevaliditeetin suhteen on se, jos haastateltavat ja haastattelija tulkitsevat asioita eri tavalla johtuen taustasta ja koulutuksesta (Runeson & Höst, 2009). Tässä tapauksessa tutkijan ja muiden osallisten taustat olivat melko yhtenäiset. Kaikilla osapuolilla oli akateeminen tausta ja runsaasti kokemusta erilaisista sovellusprojekteista. Tutkimusvaiheet raportoitiin perusteellisesti, eli tutkimusprosessilla on korkea läpinäkyvyyden aste.

Sisäinen validiteetti tarkastelee tutkimuksessa esiintyneiden kausaalisuhteiden validiutta (Runeson & Höst, 2009). Tutkimuksen sisäistä validiteettia voidaan kasvattaa tarkastelemalla tutkittavia ilmiöitä teoreettisten perusteiden pohjalta. Tässä tutkimuksessa tehtyjä havaintoja reflektoitiin aiheesta esitettyyn kirjallisuuteen, mikä auttoi asettamaan tutkittavat ilmiöt teoreettiseen viitekehykseen ja vertailemaan havaittuja ilmiöitä kirjallisuudessa esitettyihin ilmiöihin. Tässä tutkimuksessa ei pyritty kuitenkaan paljastamaan kausaalisuhteita, joten tätä validiteetin lajia ei ole tarkoituksenmukaista soveltaa tarkemmin tässä yhteydessä.

Ulkoinen validiteetti mittaa tulosten yleistettävyyttä (Runeson & Höst, 2009). Tapaustutkimuksen yleistettävyys on hyvin usein kyseenalaista (Yin, 1994). Ulkoista validiteettia voidaan parantaa testaamalla tutkittavaa ilmiötä useammissa tapauksissa. Tämän tutkimuksen yhteydessä aihetta käsiteltiin vain yhdessä, hyvin rajatussa tapauksessa, mikä laskee tutkimuksen yleistettävyyttä huomattavasti. Tämä on huomioitu tutkimuksen tuloksissa ja niiden pohjalta tehdyissä

päätelmissä. Tutkimustulosten ulkoisen validiteetin parantamiseksi tarvitaan lisää jatkotutkimuksia aiheesta.

7.3 Tulosten hyödyntäminen

Tässä alaluvussa kerrotaan, millä tavalla tutkimuksen tuloksia voidaan hyödyntää. Aluksi kerrotaan, miten tuloksia voidaan hyödyntää käytännössä. Toiseksi kerrotaan, miten tuloksia voidaan hyödyntää muissa tutkimuksissa.

Tässä tutkimuksessa esitetyn kaltaisen kevyen ja inkrementaalisen toimintatavan ja havaittujen hyvien käytänteiden avulla modernisoitu sovellus voidaan ottaa vaiheittain käyttöön. Kuten aiemmissa luvuissa on todettu, palvelukeskeisen arkkitehtuurin toteuttaminen hyviä käytänteitä noudattaen tekee sovelluksesta huomattavasti helpommin ylläpidettävän aiempaan sovellukseen verrattuna.

Tässä tutkimuksessa käytetty ja dokumentoitu modernisointiprosessi havaintoineen tarjoaa hyvän lähtökohdan vastaavien modernisointien suorittamiseen. Tutkimuksessa on tuotu esiin myös erilaisia malleja pilvimodernisoinnin menetelmän valintaan, prosessien jäsentämiseen ja toteuttamiseen. Tutkimuksessa esiin tulleisiin haasteisiin on laadittu käytänteitä, joilla näitä haasteita voidaan yrittää ratkaista.

Perinnesovellusten modernisoinnista on kirjoitettu varsin paljon, ja myös empiiristä tutkimusta on tehty (Bergmayr ym., 2013; Cetin ym., 2007; Mohagheghi ym., 2010; Ransom ym., 1998; Seacord ym., 2003). Tässä tapauksessa kysymyksessä on erityistapaus, jossa modernisoinnin kohteena oli tieteellinen laskentasovellus ja teknologiana pilvipalvelut. Modernisoinnista, jossa tavoitteena on toteuttaa olemassa olevaan tieteelliseen laskentasovellukseen SaaS-pilvipalvelumalli käyttäen ketterän kehityksen periaatteisiin (Beck ym., 2001) soveltuvaa iteratiivista toimintatapaa, ei ole tiettävästi aiemmin tehty tutkimusta.

Tulevassa tutkimuksessa olisi mahdollista pyrkiä toistamaan modernisointiprosessi vastaavan kaltaisen sovelluksen yhteydessä tehtyjen havaintojen varmistamiseksi tai hylkäämiseksi. Prosessin skaalautuvuutta voitaisiin tutkia toistamalla prosessi huomattavasti laajemman ja monimutkaisemman sovelluksen yhteydessä. Prosessia voisi myös kehittää eteenpäin tai tarkentaa sitä erilaisia tilanteita varten.

Kevyelle pilvimodernisointiprosessimallille vaikuttaisi olevan tarvetta, koska kirjallisuudessa esitetyt menetelmät (Chauhan & Babar, 2012; Menychtas ym., 2013; Mohagheghi ym., 2010) eivät vaikuta sopivan ketterän kehityksen periaatteisiin (Beck ym., 2001). Kevyen modernisointiprosessin kehittämistä voisi lähestyä esimerkiksi ketterään ylläpitoon esitettyjen prosessien (Choudhari & Suman, 2010; Kajko-Mattsson & Nyfjord, 2009) suunnasta.

8 YHTEENVETO

Tämän tutkimuksen tutkimusongelma on "*Miten sovitetaan yhteen tutkijoiden toiveet ja ohjelmistojen modernisointi pilvipalveluksi tieteellisen laskentasovelluksen tapauksessa?*". Tutkimusongelma on syntynyt oletuksesta, että tieteentekijät haluaisivat päästä nauttimaan pilvipalveluiden hyödyistä, mutta heillä on harvoin resursseja ja tarvittavaa osaamista vaativia pilvimigraatioprosesseja varten. Tämän vuoksi tutkimuksessa lähdettiin selvittämään mahdollisia keinoja tieteellisen laskentasovelluksen modernisoimiseksi pilvipalveluksi. Tutkimusongelman lähestymisen avuksi muodostettiin seuraavat tutkimuskysymykset:

- Mitkä ovat motiivit tieteellisen laskentasovelluksen modernisointiin pilvipalveluksi?
- Mitkä modernisoinnin menetelmät tukevat tutkijoiden toiveita?
- Miten tieteellisen laskentasovelluksen modernisointi voidaan toteuttaa käytännössä?

Tutkimuskysymyksiin lähdettiin hakemaan vastauksia tekemällä aluksi kirjallisuuskatsaus tieteellisiin laskentasovelluksiin, pilvipalveluihin ja perinnesovellusten modernisointiin. Tämän jälkeen laadittiin konstruktiivinen tapaustutkimus, jossa eräs tieteellinen laskentasovellus modernisoitiin pilvipalveluksi.

Tieteellisillä laskentasovelluksilla on pitkä historia, joka ulottuu aivan digitaalisen tietokoneen alkuaikoihin saakka. Monet numeeriset ongelmat ovat luonteeltaan sellaisia, että niiden ratkaisut vaativat erittäin suurta laskentatehoa. Tietokoneiden laskentateho on kehittynyt ja kehittyy jatkuvasti valtavalla vauhdilla, mutta tutkijoilla vaikuttaa aina olevan tarvetta tehokkaammille ratkaisuille. Siinä missä tutkimukset tarvitsevat paljon laskentatehoa, tarvitaan myös runsaasti tallennustilaa tutkimusten tuottamalle datalle. Valtavat tietomassat asettavat oman haasteensa tiedon varastoinnille ja jakelulle.

Pilvipalvelut vaikuttavat erittäin lupaavalta ratkaisulta tieteellisen laskennan vaatimuksille valtavasta laskentatehosta ja erittäin suuresta tallennuskapasiteetista. Monissa tieteellisissä tutkimuksissa ja sovelluksissa onkin hyödynnetty pilvipalveluita viime vuosien aikana. Amazon Web Services on yksi suurimmista pilvipalveluntarjoajista, jonka valikoimaan kuuluu kattava joukko IaaS- ja PaaS-palveluita. Kirjallisuudesta löytyy runsaasti esimerkkejä, joissa AWS-pilvipalveluita on hyödynnetty tieteellisten laskentasovellusten tapauksessa. Infra-

struktuurin ulkoistamisessa on myös omat riskinsä, joita tulee puntaroida, mikäli pilveen aiotaan siirtyä.

Perinnesovelluksen modernisointi on eräs keino, jolla olemassa olevan sovelluksen arkkitehtuuriin voidaan tehdä muutoksia. Erilaisia modernisointimenetelmiä on useita, mutta niissä kaikissa on pyrkimyksenä hyödyntää perinnesovellukseen toteutettuja toimintoja modernisoidussa sovelluksessa. Modernisoinnissa ei luoda kokonaan uutta sovellusta, vaan perinnesovelluksen toimintoja siirretään nykyaikaisempaan tekniseen viitekehykseen.

Perinnesovelluksen modernisoinnilla pilvipalveluksi tarkoitetaan sitä, että sovelluksen arkkitehtuuriin tehdään tarvittavat muutokset, jotta siihen voidaan toteuttaa SaaS-pilvipalvelumalli. Toisin sanoen modernisoinnin tuloksena syntyy sovellus, jonka käyttämiseen käyttäjä tarvitsee vain internetselaimen. Pilvimodernisoinnin tapauksessa sovellukseen toteutetaan hyvin usein palvelukeskeinen arkkitehtuuri. Palvelukeskeisen arkkitehtuurin toteuttaminen mahdollistaa usein myös IaaS-pilvipalveluiden etujen hyödyntämisen sovelluksessa.

Tämän tutkimuksen yhteydessä tehtiin tapaustutkimus, jonka kohteena oli erään tieteellisen laskentasovelluksen modernisointi pilvipalveluksi. Kirjallisuudessa esitetyt pilvimodernisointimenetelmät vaikuttivat liian raskailta pienehkön tieteellisen laskentasovelluksen tapaukseen, joten tutkija suoritti modernisoinnin soveltaen kevyttä ja iteratiivista toimintatapaa, jossa yhdisteltiin ketterän kehityksen periaatteita, Lean-ajattelua sovellettuna ohjelmistokehitykseen sekä testivetoista kehittämistä. Valittua toimintatapaa arvioitiin reflektoimalla tutkijan tekemiä havaintoja modernisointiprosessista tieteelliseen kirjallisuuteen sekä erityisesti vertailemalla käytettyä toimintatapaa kirjallisuudessa esitettyihin pilvimodernisointimenetelmiin.

Modernisointi saatiin toteutettua valitulla menetelmällä, ja tuloksiin oltiin pääasiassa erittäin tyytyväisiä. Tutkija laati tutkimusprosessin yhteydessä keskeisimmistä käytännön havainnoista ehdotuksen parhaat käytänteet -listauksesta tieteellisen laskentasovelluksen modernisoimiseksi pilvipalveluksi. Tulosten reliabiliteetin ja validiteetin kasvattamiseksi olisi kuitenkin tehtävä jatkotutkimuksia, joissa esitettyjä käytänteitä testattaisiin muissa tapauksissa.

Ensimmäiseen tutkimuskysymykseen saatiin kirjallisuuskatsauksen avulla selkeä näkemys, jota myös tapaustutkimuksessa tehdyt havainnot tukevat. Tieteellisille laskentasovelluksille todettiin olevan hyvin tyypillistä se, että ne edellyttävät käytetyltä laitteistolta usein hyvin suurta laskentakapasiteettia sekä runsaasti tallennustilaa. Pilvipalvelut mahdollistavat käytännössä rajattoman laskenta- ja tallennuskapasiteetin hankkimisen käytön mukaan laskutettavana palveluna. Pilvipalveluita käyttämällä voidaan siis välttyä oman IT-infrastruktuurin hankkimisesta ja ylläpidosta koituvilta investoinneilta ja muilta ongelmilta, minkä todettiin olevan hyvä asia myös tieteentekijöiden kannalta, koska näihin asioihin aiemmin tarvitut resurssit voidaan käyttää ydinasian tutkimiseen. Tutkimuksessa nousi esiin myös tutkijoiden houkutus käyttää kaupallisia pilvipalveluita voidakseen välttää tutkimuslaitoksissa esiintyvää laskentaresurssien jakamiseen liittyvää byrokratiaa ja kilpailua. Tieteellisen laskentasovelluksen modernisointi pilvipalveluksi voi avata myös uusia mahdollisuuksia tehtävälle tutkimukselle.

Tutkimuksessa havaittiin, että sovelluksen modernisointi pilvipalveluksi edellyttää palvelukeskeisen arkkitehtuurin toteuttamista sovellukseen. Palvelukeskeisen arkkitehtuurin toteuttaminen sovellukseen tekee sovelluksesta erittäin helposti saavutettavan sekä helposti integroitavan muihin palveluihin, mikä avaa uusia mahdollisuuksia tieteellisinä instrumentteina käytettävien so-

vellusten yhdistelemiselle. Sovelluksen parempi saavutettavuus tarkoittaa sitä, että kyseinen tieteen tekemiseen käytetty sovellus on yhä useampien tutkijoiden käytettävissä, mikä tekee helpommaksi sovelluksen avulla saavutettujen tulosten varmistamisen, mikä puolestaan lisää tutkimuksen läpinäkyvyyttä. SaaS-pilvipalvelumalli edellyttää, että sovellusta voidaan käyttää pelkän internetselaimen avulla, joten sovellusta ei tarvitse asentaa tai konfiguroida, jotta sen saa käyttöönsä.

Pilvipalveluissa on myös ongelmia, joita tulee puntaroida ennen päätöstä siirtyä niiden käyttöön. Yleisiä huolenaiheita pilvipalveluissa ovat tietoturvaan ja suorituskykyyn liittyvät seikat. Suorituskyvyn suhteen pilvipalvelut ovat kehittyneet viime vuosina huomattavasti, ja kyseinen huolenaihe liittynee enemmän ihmisten mielikuviin pilvipalveluista kuin todellisiin ongelmiin. Tämän tutkimuksen yhteydessä tehtiin myös oletus, että tieteellisissä sovelluksissa liiketoiminnalliset seikat ovat toissijaisia, ja näiden seikkojen sivuuttamisen havaittiin vähentävän pilvipalveluiden ongelmien merkitystä ja siten yksinkertaistavan päätöksentekoprosesseja.

Toiseen tutkimuskysymykseen liittyen havaittiin, että tieteellisessä kirjallisuudessa esiintyvät menetelmät perinnesovelluksen modernisoimiseksi pilvipalveluksi vaikuttavat turhan raskailta yksinkertaisissa tapauksissa. Vaikutelmaa menetelmien raskaudesta havaittiin lisäävän erityisesti sen, että kirjallisuudessa esitetyt menetelmät kärsivät iteratiivisuuden puutteesta, minkä vuoksi niiden sovittaminen esimerkiksi ketteriin menetelmiin voi olla haastavaa. Tutkimuksessa tehtiin oletus, että tutkijoilla on harvemmin riittäviä resursseja ja taitoja laajojen ja monimutkaisten pilvimigraatioprosessien hallitsemiseksi, minkä vuoksi tapaustutkimuksessa päätettiin soveltaa kevyttä ja iteratiivista toimintatapaa ja selvittää sen avulla, mitkä modernisoinnin menetelmät sopivat myös kevyempään toimintatapaan.

Eräs merkittävä tapaustutkimuksen yhteydessä tehty havainto oli, että tieteellisen laskentasovelluksen modernisointi ei teknisiltä osin poikkea muiden sovellusten modernisoinnista, mutta liiketoiminnalliselta näkökulmalta tieteellisen laskentasovelluksen kohdealueen oletettiin ja havaittiin olevan huomattavasti yksinkertaisempi kuin kaupallisen sovelluksen. Pilvipalvelumalli on modernisoinnin yhteydessä usein merkittävä muutos myös sovelluksen liiketoimintamalliin. Tämän tutkimuksen yhteydessä tehtiin päätös olla tieteen kontekstissa painottamatta pilvipalveluiden ja modernisaation liiketoiminnallista puolta. Näkökulman poisjättäminen on toisaalta merkittävä heikkous tutkimuksen tulosten yleistettävyyden kannalta.

Kolmanteen tutkimuskysymykseen havaittiin olevan useita mahdollisia lähestymistapoja. Mikäli organisaatiossa on riittävästi resursseja ja vaadittavaa osaamista olemassa olevien pilvimodernisointimenetelmien käyttöön, voidaan käyttää kyseisiä menetelmiä. Tapaustutkimuksessa tehdyt havainnot antavat näyttöä siitä, että ainakin tieteellisen laskentasovelluksen tapauksessa sovelluksen pilvimodernisointiin voidaan soveltaa myös kevyempää iteratiivista lähestymistapaa. Väitteen tueksi toimintatapaa tulisi kokeilla useammassa tapauksessa ja suuremmassa mittakaavassa. Tämä tutkimus toteutettiin yhden henkilön toimesta. Usein sovellusprojekteihin liittyy useampia sovellussuunnittelijoita sekä ohjelmoijia, ja modernisoitava sovellus voi olla merkittävästi laajempi. Toimintatavan skaalautuvuudesta ei ole tietoa, koska näyttöä toimintatavan menestyksellisyydestä tuotettiin ainoastaan pienessä mittakaavassa.

Tämän tapaustutkimuksen osalta modernisoinnin onnistumista edesauttoi myös perinnesovelluksen vakavat ongelmat, joihin verrattuna mikä tahansa

edistys olisi voinut tuottaa positiivisia havaintoja. Tapaustutkimuksen onnistumisesta ei myöskään ole toistaiseksi tarjota kattavaa näyttöä, koska modernisoitua sovellusta ei ole vielä otettu käyttöön, joten varsinaista todistusaineistoa laatuvaatimusten toteutumisesta ei ole. Toiminnallisten vaatimusten osalta sovellus saatiin kuitenkin todistettavasti modernisoitua, ja modernisoinnin tuloksiin oltiin tyytyväisiä johtavan tutkijan ja perinnesovelluksen ylläpitäjän taholta. Myös tutkimuksen laatija on itse tyytyväinen valittuun toimintatapaan ja koki sen erittäin lupaavaksi varsinkin ketteriin menetelmiin sovittamista ajatellen.

Tutkimuksen tuloksia ei voida vielä tässä vaiheessa yleistää, koska kyseessä ovat yksittäisen erityistapauksen pohjalta tehdyt havainnot ja johtopäätökset. Tutkimuksessa tehdyt havainnot olivat yksittäisen tutkijan tekemiä ja analysoimia. Tutkija laati, toteutti ja analysoi itse myös havaintojen vahvistamiseen käytetyt teemahaastattelut, jolloin on vaara, että haastateltavat ovat antaneet ns. ”oikeita” vastauksia. Tutkimuksen reliabiliteetin ja validiteetin kasvattamiseksi tarvitaan lisää jatkotutkimuksia aiheesta.

Tässä tutkimuksessa käytetty ja dokumentoitu modernisointiprosessi havaintoineen tarjoaa hyvän lähtökohdan vastaavien modernisointien suorittamiseen ja tutkimiseen. Käytännössä tässä tutkimuksessa esitetty iteratiivinen toimintatapa mahdollistaa modernisoidun sovelluksen vaiheittaisen käyttöönoton. Tutkimuksessa esiin tulleet ongelmat ja hyviksi havaitut käytänteet tarjoavat lähestymistapoja ja ratkaisuja pilvimodernisoinnissa vastaan tuleviin tilanteisiin. Mahdollinen jatkotutkimusaihe on esimerkiksi tämän tutkimuksen toistaminen vastaavan kaltaisessa tapauksessa. Havaittujen käytänteiden yleistettävyyttä ja skaalautuvuutta voidaan tutkia soveltamalla niitä myös erilaisiin tapauksiin ja huomattavasti laajempiin sovelluksiin. Kirjallisuudessa esitetyt pilvimodernisointimenetelmät eivät vaikuta soveltuvan kevyeen ja iteratiiviseen toimintatapaan, joten ketteriä menetelmiä hyödyntävän pilvimodernisointimenetelmän kehittäminen olisi myös erittäin houkutteleva jatkotutkimuksen aihe.

LÄHTEET

Agarwal, R. & Umphress, D. (2008). Extreme Programming for a Single Person Team. Teoksessa *Proceedings of the 46th Annual Southeast Regional Conference on XX* (ss. 82–87). New York, NY, USA: ACM.

Alkazemi, B. Y., Nour, M. K. & Meelud, A. Q. (2013). Towards a Framework to Assess Legacy Systems. Teoksessa *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (ss. 924–928).

Alonso, J., Orue-Echevarria, L., Escalante, M., Gorronogoitia, J. & Presenza, D. (2013). Cloud modernization assessment framework: Analyzing the impact of a potential migration to Cloud. Teoksessa *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)* (ss. 64–73).

Anttila, P. (2000). Tutkimuksen taito ja tiedon hankinta. 3. painos. *Jyväskylä, Gummerus*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. & Zaharia, M. (2010). A View of Cloud Computing. *Commun. ACM*, *53*, 50–58.

AWS | What is AWS - Cloud Computing with Amazon Web Services. (2014). *Amazon Web Services, Inc.* Noudettu 14.9.2014 osoitteesta http://aws.amazon.com/what-is-aws/

AWS Free Tier. (2014). *Amazon Web Services, Inc.* Noudettu 4.11.2014 osoitteesta http://aws.amazon.com/free/

Beatty, R. C. & Williams, C. D. (2006). ERP II: Best Practices for Successfully Implementing an ERP Upgrade. *Commun. ACM*, *49*, 105–109.

Beck, K. (2003). *Test-driven Development: By Example*. Addison-Wesley Professional.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. & others. (2001). Principles behind the agile manifesto. *Agile Alliance*, 1–2.

Bentley, P. (2008). *The Book of Numbers: The Secret of Numbers and How They Changed the World*. Richmond Hill, Ont: Firefly Books.

Bergman, R. (2012). Embracing Nihilism as a Software Development Philosophy and the Birth of the Big Book of Dead Code. Teoksessa *Agile Conference (AGILE), 2012* (ss. 86–91).

Bergmayr, A., Bruneliere, H., Canovas Izquierdo, J. L., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarriam L., Pezuela, C. & Wimmer, M. (2013). Migrating Legacy Software to the Cloud with ARTIST. Teoksessa *2013 17th European Conference on Software Maintenance and Reengineering (CSMR)* (ss. 465–468).

Binder, K., & Heermann, D. (2010). *Monte Carlo Simulation in Statistical Physics: An Introduction*. Springer Science & Business Media.

Biro, L., Bacu, V., Rodila, D., Barabas, L. & Gorgan, D. (2012). Grid to cloud migration of scientific applications, using dynamically created cloud clusters. Teoksessa *2012 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)* (ss. 335–340).

Bloomberg, J. (2013). *The Agile Architecture Revolution: How Cloud Computing, REST-Based SOA, and Mobile Computing Are Changing Enterprise IT*. John Wiley & Sons.

Branch, R., Tjeerdsma, H., Wilson, C., Hurley, R. & McConnell, S. (2014). Cloud Computing and Big Data: A Review of Current Service Models and Hardware Perspectives. *Journal of Software Engineering and Applications*, *07*, 686–693.

Brian, O., Brunschwiler, T., Dill, H., Christ, H., Falsafi, B., Fischer, M., Grivas, S. G., Giovanoli, C., Gisi, R. E., Gutman, R. & others (2012). Cloud Computing. *White Paper SATW*. Noudettu 21.10.2014 osoitteesta https://www.satw.ethz.ch/organisation/tpf/tpf_ict/box_feeder/2012-11-06_2_SATW_White_Paper_Cloud_Computing_EN.pdf

Bunch, C., Drawert, B. & Norman, M. (2009). Mapscale: a cloud environment for scientific computing. *University of California, Computer Science Department, Tech. Rep.* Noudettu 14.10.2014 osoitteesta http://cs.ucsb.edu/~cgb/papers/mapscale.pdf

Cetin, S., Ilker Altintas, N., Oguztuzun, H., Dogru, A. H., Tufekci, O. & Suloglu, S. (2007). Legacy Migration to Service-Oriented Computing with Mashups. Teoksessa *International Conference on Software Engineering Advances, 2007. ICSEA 2007* (ss. 21–21).

Chapin, N. (2000). Software maintenance types - a fresh view. Teoksessa *International Conference on Software Maintenance, 2000. Proceedings* (ss. 247–252).

Chauhan, M. & Babar, M. (2012). Towards Process Support for Migrating Applications to Cloud Computing. Teoksessa *2012 International Conference on Cloud and Service Computing (CSC)* (ss. 80–87).

Chikofsky, E. J. & Cross II, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Softw.*, *7*, 13–17.

Choudhari, J. & Suman, U. (2010). Iterative Maintenance Life Cycle Using eXtreme Programming. Teoksessa *2010 International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom)* (ss. 401–403).

Clay, K. (2013.) *Amazon.com Goes Down, Loses $66,240 Per Minute*. *Forbes*. Noudettu 21.10.2014 osoitteesta http://www.forbes.com/sites/kellyclay/2013/08/19/amazon-com-goes-down-loses-66240-per-minute/

Comella-Dorda, S., Wallnau, K., Seacord, R. C. & Robert, J. (2000). *A survey of legacy system modernization approaches*. DTIC Document. Noudettu 21.10.2014 osoitteesta http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA377453

Cook, N., Milojicic, D., Kaufmann, R. & Sevinsky, J. (2012). N3phele: Open Science-as-a-Service Workbench for Cloud-based Scientific Computing. Teoksessa *Open Cirrus Summit (OCS), 2012 Seventh* (ss. 1–5).

Cretella, G. & Martino, B. D. (2014). An Overview of Approaches for the Migration of Applications to the Cloud. Teoksessa L. Caporarello, B. D. Martino, & M. Martinez (Toim.), *Smart Organizations and Smart Artifacts* (ss. 67–75). Springer International Publishing.

Day, C. (2014). Python Power. *Computing in Science Engineering*, *16*, 88–88.

Dzhurov, Y., Krasteva, I. & Ilieva, S. (2009). *Personal Extreme Programming – An Agile Process for Autonomous Developers*. Demetra EOOD. Noudettu

14.10.2014 osoitteesta http://research.uni-sofia.bg:8080/handle/10506/251

Efron, B. & Tibshirani, R. (1990). *Statistical data analysis in the computer age*. University of Toronto, Department of Statistics. Noudettu 21.10.2014 osoitteesta https://statistics.stanford.edu/sites/default/files/EFS%20NSF%20379.pdf

Elleuch, N., Khalfallah, A. & Ben Ahmed, S. (2007). Software Architecture in Model Driven Architecture. Teoksessa *International Symposium on Computational Intelligence and Intelligent Informatics, 2007. ISCIII '07* (ss. 219–223).

Eucalyptus and Amazon Web Services Compatibility. (2014). *Eucalyptus*. Noudettu 24.10.2014 osoitteesta https://www.eucalyptus.com/aws-compatibility

Fanqi, M. & Yunqi, K. (2013). Research on the Model of Legacy Software Reuse Based on Code Clone Detection. Teoksessa *2013 5th International Conference on Computational Intelligence and Communication Networks (CICN)* (ss. 491–495).

Fletcher, R. (2013). *Practical Methods of Optimization*. John Wiley & Sons.

Golub, G. H. (1992). *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. Academic Press.

Grama, A. (2003). *Introduction to Parallel Computing*. Pearson Education.

Grubb, P. & Takang, A. A. (2003). *Software Maintenance: Concepts and Practice* (2 edition.). River Edge, N.J: World Scientific Publishing Company.

Heermann, P. D. D. W. (1990). Computer-Simulation Methods. Teoksessa *Computer Simulation Methods in Theoretical Physics* (ss. 8–12). Springer Berlin Heidelberg.

Heino, P. (2010). Pilvipalvelut. *Helsinki: Talentum*.

Hirsjärvi, S., Remes, P. & Sajavaara, P. (2010). *Tutki ja kirjoita*. Tammi.

Hromkovi, J. (2004). *Theoretical Computer Science: Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography*. Springer Science & Business Media.

Huang, P., Lin, P. & Peng, H. (2010). Grid-Cloud: IT platform for Service Science. Teoksessa *2010 2nd International Conference on Future Computer and Communication (ICFCC)* (Vsk. 3, ss. V3–143–V3–147).

Huang, Z., Yu, J. & Yu, F. (2013). Cloud processing of 1000 genomes sequencing data using Amazon Web Service. Teoksessa *2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (ss. 49–52).

Jinesh, V. & Sajee, M. (2014). Overview of Amazon Web Services. Amazon Web Services. Noudettu 21.10.2014 osoitteesta https://media.amazonwebservices.com/AWS_Overview.pdf

Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B. P. & Maechling, P. (2010). Data Sharing Options for Scientific Workflows on Amazon EC2. Teoksessa *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for* (ss. 1–9).

Juve, G., Rynge, M., Deelman, E., Vockler, J.-S. & Berriman, G. B. (2013). Comparing FutureGrid, Amazon EC2, and Open Science Grid for Scientific Workflows. *Computing in Science Engineering*, *15*, 20–29.

Järvinen, P. & Järvinen, A. (2011). *Tutkimustyön metodeista*. Opinpajan kirja.

Kajko-Mattsson, M. & Nyfjord, J. (2009). A Model of Agile Evolution and Maintenance Process. Teoksessa *42nd Hawaii International Conference on System Sciences, 2009. HICSS '09* (ss. 1–10).

Kankaanpää, I. (2011). *IT artefact renewal : triggers, timing and benefits*. University of Jyväskylä. Noudettu 14.9.2014 osoitteesta https://jyx.jyu.fi/dspace/handle/123456789/37192

Krause, S. (2013). Tutorial: Hands on Introduction to Amazon Web Services. Teoksessa *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC)*.

Law, A. M. & Kelton, W. D. (2000). *Simulation Modelling and Analysis* (3rd Edition). Boston: McGraw Hill Higher Education.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*, 1060–1076.

Lewis, G., Morris, E. & Smith, D. (2005). Service-Oriented Migration and Reuse Technique (SMART). Teoksessa *13th IEEE International Workshop on Software Technology and Engineering Practice, 2005* (ss. 222–229).

Lidman, J., Quinlan, D. J., Liao, C. & McKee, S. A. (2012). ROSE::FTTransform - A source-to-source translation framework for exascale fault-tolerance research. Teoksessa *2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W)* (ss. 1–6).

Lukka, K. (2001). Konstruktiivinen tutkimusote. Noudettu 21.10.2014 osoitteesta http://www.metodix.com/fi/sisallys/01_menetelmat/02_metodiartikkelit/lukka_const_research_app/kooste

Mell, P. & Grance, T. (2009). The NIST definition of cloud computing. *National Institute of Standards and Technology*, *53*, 50.

Mendez, D., Villamiazr, M. & Castro, H. (2013). e-Clouds: Scientific Computing as a Service. Teoksessa *2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)* (ss. 481–486).

Menychtas, A., Santzaridou, C., Kousiouris, G., Varvarigou, T., Orue-Echevarria, L., Alonso, J., Gorronogoitia, J., Bruneliere, H., Strauss, O., Senkova, T., Pellens, B. & Stuer, P. (2013). ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud. Teoksessa *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)* (ss. 424–431).

Mohagheghi, P., Berre, A. J., Henry, A., Barbier, F. & Sadovykh, A. (2010). REMICS- REuse and Migration of Legacy Applications to Interoperable Cloud Services. Teoksessa E. D. Nitto & R. Yahyapour (toim.), *Towards a Service-Based Internet* (ss. 195–196). Springer Berlin Heidelberg.

Nash, S. G. (toim.). (1990). *A History of Scientific Computing*. New York, NY, USA: ACM.

Noaje, G., Jaillet, C. & Krajecki, M. (2011). Source-to-Source Code Translator: OpenMP C to CUDA. Teoksessa *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)* (ss. 512–519).

Nurhasan, Y. I., Dabarsyah, B. & Fakhrurroja, H. (2013). Information model design as model-driven for Service Oriented Architecture (SOA) implementation in PK-BLU institution using SOA Ontology: Case study: Financial Administration Bureau Padjadjaran University. Teoksessa *2013 International Conference on ICT for Smart Society (ICISS)* (ss. 1–9).

Poppendieck, M. & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.

Ransom, J., Somerville, I. & Warren, I. (1998). A method for assessing legacy systems for evolution. Teoksessa *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 1998* (ss. 128–134).

Razavian, M. & Lago, P. (2010). A Frame of Reference for SOA Migration. Teoksessa E. D. Nitto & R. Yahyapour (toim.), *Towards a Service-Based Internet* (ss. 150–162). Springer Berlin Heidelberg.

Rehr, J., Vila, F., Gardner, J., Svec, L. & Prange, M. (2011). Scientific Computing in the Cloud. *Computing in Science Engineering, Early Access Online*.

Runeson, P. & Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Softw. Engg.*, *14*, 131–164.

Ruttimann, J. (2006). 2020 computing: Milestones in scientific computing. *Nature*, *440*, 399–405.

Saarelainen, M.-M., Ahonen, J. J., Lintinen, H., Koskinen, J., Kankaanpää, I., Sivula, H., Juutilainen, P. & Tilus, T. (2006). Software modernization and replacement decision making in industry: A qualitative study. Teoksessa *Proceedings of the 10th international conference on Evaluation and Assessment in Software Engineering* (ss. 12–21). British Computer Society.

Salo, I. (2010). Cloud computing - palvelut verkossa. *Jyväskylä: WSOYpro Oy*.

Schmidt, J. W. & Taylor, R. E. (1970). *Simulation and Analysis of Industrial Systems*. Richard D. Irwin.

Scientific Computing with EC2 Spot Instances. (2011). Noudettu 24.10.2014, osoitteesta http://aws.amazon.com/blogs/aws/scientific-computing-with-ec2-spot-instances

Seacord, R. C., Plakosh, D. & Lewis, G. A. (2003). *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Soininen, M. (1995). *Tieteellisen tutkimuksen perusteet*. Turun yliopisto.

Srirama, S. N., Batrashev, O., Jakovits, P. & Vainikko, E. (2011). Scalability of Parallel Scientific Applications on the Cloud. *Sci. Program.*, *19*, 91–105.

Srirama, S. N., Ivanistsev, V., Jakovits, P. & Willmore, C. (2013). Direct migration of scientific computing experiments to the cloud. Teoksessa *2013 International Conference on High Performance Computing and Simulation (HPCS)* (ss. 27–34).

Su, H., Cheng, B., Wu, T. & Li, X. (2011). Mashup service release based on SOAP and REST. Teoksessa *2011 International Conference on Computer Science and Network Technology (ICCSNT)* (Vsk. 2, ss. 1091–1095).

Swanson, E. B. (1976). The Dimensions of Maintenance. Teoksessa *Proceedings of the 2nd International Conference on Software Engineering* (ss. 492–497). Los Alamitos, CA, USA: IEEE Computer Society Press.

Tan, P.-N., Steinbach, M. & Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Tsaftaris, S. (2014). A Scientist's Guide to Cloud Computing. *Computing in Science Engineering*, *16*, 70–76.

Vecchiola, C., Pandey, S. & Buyya, R. (2009). High-Performance Cloud Computing: A View of Scientific Applications. Teoksessa *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)* (ss. 4–16).

Waris, M., Khan, S. A. & Fakhar, M. Z. (2013). Factors effecting service oriented architecture implementation. Teoksessa *Science and Information Conference (SAI), 2013* (ss. 1–8).

Wee, S. (2011). Debunking Real-Time Pricing in Cloud Computing. Teoksessa *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (ss. 585–590).

Winsberg, E. (2014). Computer Simulations in Science. Teoksessa E. N. Zalta (Toim.), *The Stanford Encyclopedia of Philosophy* (Fall 2014.). Noudettu 24.10.2014 osoitteesta http://plato.stanford.edu/archives/fall2014/entries/simulations-science/

Xu, F., Liu, F., Jin, H. & Vasilakos, A. V. (2014). Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions. *Proceedings of the IEEE*, *102*, 11–31.

Yin, R. K. (1994). *Case study research: design and methods*. Sage Publications.

Yu, J. & Dong, K. (2014). VLAB-C: A Cloud Service Platform for Collaborative Virtual Laboratory. Teoksessa *2014 IEEE International Conference on Services Computing (SCC)* (ss. 829–835).

Yu, J., Dong, K. & Nan, K. (2012). Duckling: Towards Cloud Service for Scientific Collaboration System. Teoksessa *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion* (ss. 259–262). New York, NY, USA: ACM.

Zachary, J. L. (1998). An introduction to scientific programming. *IEEE Computational Science Engineering*, *5*, 6–10.

Zhang, S., Zhang, S., Chen, X. & Huo, X. (2010). Cloud Computing Research and Development Trend. Teoksessa *Second International Conference on Future Networks, 2010. ICFN '10* (ss. 93–97).

LIITE 1 HAASTATTELU ENNEN MODERNISOINTIPROSESSIA

Tämä on anonymisoitu versio alkuperäisen haastattelun muistiinpanoista. Kaikki mahdolliset viittaukset henkilöihin, organisaatioihin ja tuotteisiin on poistettu tutkijan toimesta.

Miksi nykyinen sovellus tulisi modernisoida? Mitä mahdollisia ongelmia ja puutteita nykyisessä sovelluksessa on, joihin kaivataan ratkaisua?

Vastaus:

"Nykyinen sovellus on hankala ylläpitää ja siihen on vaikea tuoda uusia ominaisuuksia."

"Nykyisen sovelluksen prosesseja tulisi automatisoida, esimerkiksi simulaatioiden ajo ja tallennus järjestelmään."

"Datan määrä on suuri ja kasvaa jatkuvasti, mikä aiheuttaa ylläpito-ongelmia."

"Simulaatioiden konfigurointi ja ajaminen kaipaisi yhtenäisen alustan."

"Sovellus halutaan integroida muihin palveluihin."

"Sovellus halutaan tehdä helposti muiden tutkijoiden saavutettavaksi ja käytettäväksi."

"Myös sovelluksella tehtävän tutkimuksen kannalta helpompi käytettävyys ja parempi saavutettavuus olisivat positiivisia asioita, sillä kokeiden tulokset olisivat helpommin muiden saatavilla ja jopa muiden verifioitavissa, mikäli sovelluksen käyttäjät voivat suorittaa ja toistaa kokeita itsenäisesti."

"Käyttäjiä halutaan mahdollisesti pystyä laskuttamaan jossain vaiheessa käyttämästään prosessoriajasta ja tallennustilasta."

"Tutkimusprojektien rahoitus on aina rajallinen, joten mitä vähemmän resursseja kuluu infrastruktuuriin ja automatisoitavissa olevien prosessien suorittamiseen, sitä parempi."

"Sovelluksesta halutaan tehdä nykyistä vakaampi. Esimerkiksi yksittäisessä tutkimuslaitoksessa toimiva sovellus on altis sähkökatkoille."

LIITE 2 HAASTATTELU MODERNISOINTIPROSESSIN PÄÄTYTTYÄ

Tämä on anonymisoitu versio alkuperäisen haastattelun muistiinpanoista. Kaikki mahdolliset viittaukset henkilöihin, organisaatioihin ja tuotteisiin on poistettu tutkijan toimesta.

Miten arvioisit(te) modernisoinnin onnistumista alla esitettyihin lähtökohtiin nähden?

-Nykyinen sovellus on hankala ylläpitää ja siihen on vaikea tuoda uusia ominaisuuksia.

-Nykyisen sovelluksen prosesseja tulisi automatisoida, esimerkiksi simulaatioiden ajo ja tallennus järjestelmään.

-Datan määrä on suuri ja kasvaa jatkuvasti, mikä aiheuttaa ylläpito-ongelmia.

-Simulaatioiden konfigurointi ja ajaminen kaipaisi yhtenäisen alustan.

-Sovellus halutaan integroida muihin palveluihin.

-Sovellus halutaan tehdä helposti muiden tutkijoiden saavutettavaksi ja käytettäväksi.

-Myös sovelluksella tehtävän tutkimuksen kannalta helpompi käytettävyys ja parempi saavutettavuus olisivat positiivisia asioita, sillä kokeiden tulokset olisivat helpommin muiden saatavilla ja jopa muiden verifioitavissa, mikäli sovelluksen käyttäjät voivat suorittaa ja toistaa kokeita itsenäisesti.

-Käyttäjiä halutaan mahdollisesti pystyä laskuttamaan jossain vaiheessa käyttämästään prosessoriajasta ja tallennustilasta.

-Tutkimusprojektien rahoitus on aina rajallinen, joten mitä vähemmän resursseja kuluu infrastruktuuriin ja automatisoitavissa olevien prosessien suorittamiseen, sitä parempi.

-Sovelluksesta halutaan tehdä nykyistä vakaampi. Esimerkiksi yksittäisessä tutkimuslaitoksessa toimiva sovellus on altis sähkökatkoille.

Vastaus:

”Modernisointi ylitti alkuperäiset odotukset, sillä uusi ohjelmisto sisältää jo tässä vaiheessa lähes kaikki oleelliset lopulliselle järjestelmälle asetetut vaatimukset. Lisäksi työn aikana on tullut esille uusia mahdollisuuksia, joista osa on toteutettu jo nyt. Työ osoitti, että pilvipalvelu tarjoaa mahdollisuuden rakentaa tieteellinen työväline, joka tarjoaa käyttökelpoisen ja laajennettavissa olevan palvelualustan pitkälle tulevaisuuteen.”

Miten arvioisit(te) modernisointiprosessin hinta/laatusuhdetta?

Vastaus:

"Erittäin hyvä. Työ osoitti selkeästi, kuinka tieteellinen tutkimus hyötyy uusista ICT-mahdollisuuksista."

Olisiko prosessin voinut hoitaa jotenkin paremmin? Jos kyllä, niin miten?

Vastaus:

"Jos vanhasta systeemistä olisi ollut hyvät dokumentit, niin uuden systeemin rakentaminen olisi voinut olla yhä nopeampaa. Skype-palaverit olivat tehokkaita, mutta informaation vaihto olisi voinut nopeutua, jos työntekijä ja työnantaja olisivat voineet tavata päivittäin (esim. olemalla samassa rakennuksessa)."

Tuleeko mieleen muita mahdollisia huomioita?

Vastaus:

"Modernisointi oli niin onnistunut, että se olisi kannattanut tehdä jo paljon aikaisemmin."