Sara Mäenpää

PERINNEJÄRJESTELMIEN MODERNISOIMISEN STRATEGIAT

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2021

TIIVISTELMÄ

Mäenpää, Sara
Perinnejärjestelmien modernisoimisen strategiat
Jyväskylä: Jyväskylän yliopisto, 2021, 37 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Marttiin, Pentti

Perinnejärjestelmät ovat vanhoja, joustamattomia sekä liiketoiminnan kannalta kriittisiä tietojärjestelmiä, jotka syövät yritysten kuluja kalliin ylläpidon muodossa. Näistä järjestelmistä on kuitenkin vaikea päästä eroon, sillä uudistaminen on kallis ja riskialtis projekti. Tämän kandidaatin tutkielman tarkoituksena on selvittää mitä perinnejärjestelmät ovat ja minkälaisilla eri strategioilla perinnejärjestelmiä voidaan uudistaa. Perinnejärjestelmät rasittavat yrityksiä ylläpitokulujen lisäksi kalliin työvoiman merkeissä. Sen lisäksi nykyajan teknologian integroiminen niihin on haastavaa. Modernisoiminen tähtää näiden ongelmien minimoimiseen ja jäykkien järjestelmien eheyttämiseen. Erilaisia strategioita on kirjallisuudessa esitetty lukuisia, mutta laadukasta ja yhtenäistä tutkimusta on vähän. Tässä tutkielmassa tuloksena esitellään neljä modernisoinnin strategiaa: uudelleenmallinnus, kääriminen, migraatio sekä korvaaminen, joiden hyviä ja huonoja puolia vertaillaan kattavan kokonaiskuvan saamiseksi. Tutkielma on toteutettu systemaattisena kirjallisuuskatsauksena.

Asiasanat: perinnejärjestelmät, modernisointi, evoluutio

ABSTRACT

Mäenpää, Sara
Modernization strategies of legacy systems
Jyväskylä: University of Jyväskylä, 2021, 37 pp.
Information Systems, Bachelor's Thesis
Supervisor: Marttiin, Pentti

Legacy systems are old, inflexible, and business critical information systems that trouble companies due to excessive maintenance costs. These information systems are hard to get rid of because modernization process is an expensive and risky project. This Bachelor's Thesis's purpose is to find out what characteristics legacy systems have and how those can be modernized. In addition to other maintenance costs legacy systems often require valuable high skilled system experts. Legacy systems are also not compatible with modern technology. Modernizing these systems aims to solve these problems. There are many different strategies to modernize legacy systems, but the research lacks papers that put together these strategies. In this systematic literature review different classifications for modernization are studied and four modernization strategies are evaluated: re-engineering, wrapping, migration and replacement.

Keywords: legacy system, modernization, software evolution

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Perinnejärjestelmiä on tutkittu jo ainakin 90-luvulta asti, mutta yhä edelleen yritysten käytössä on vuosikymmeniä vanhoja järjestelmiä. Niiden ylläpitämiseen kuluu suuria summia yrityksen resursseja, jotka ovat suoraan pois esimerkiksi innovaatiotoiminnan kehittämiseltä. Esimerkiksi Yhdysvalloissa valtion tietotekniikkaan osoittamasta budjetista arvioitiin noin 80 %:a kuluvan järjestelmien ylläpitoon ja operoimiseen (United States Government Accountability Office, 2019). Toki luku pitää sisällään myös muiden kuin perinnejärjestelmien tuomat kulut, mutta kehitys- ja innovaatiotoimintaan jäävä 20 % on kuitenkin huomattavan pieni. On siis merkittävää selvittää, miten perinnejärjestelmien tuomasta taakasta voi päästä eroon, jotta niiden ylläpitoon laitettuja resursseja voidaan kohdentaa paremmin. Modernisoimisen on tutkittu tuovan monia etuja yritykselle. Tällaisia hyötyjä ovat esimerkiksi kulujen pieneneminen, joustavuus, parempi ylläpidettävyys sekä suorituskyvyn paraneminen (Khadka ym., 2015).

Tutkielman tarkoituksena on esitellä eri strategioita, joiden avulla voidaan uudistaa järjestelmiä. Strategioilla tarkoitetaan konkreettisia keinoja, joilla perinnejärjestelmän toiminta muutetaan vanhalla teknologialla toimivasta uudella teknologialla toimivaan. Modernisoimisen tarkoituksena on muokata järjestelmistä paremmin modernin liiketoiminnan vaatimuksia vastaavia, eli usein halutaan implementoida ominaisuuksia, joita ei aikaisemmin ole vielä ollut olemassa. Eri vaihtoehdot tuntemalla on suurempi todennäköisyys löytää juuri oman perinnejärjestelmän uudistamiseen sopiva strategia, joka taas pienentää riskejä ja nostaa uudistamisprosessin onnistumisen todennäköisyyttä. Perinnejärjestelmien modernisoinnille on tutkimuksissa esitelty useita tapoja, mutta tutkijayhteisöä tuntuu jakavan mielipiteet siitä, missä menee järjestelmän ylläpidon, uudistamisen ja järjestelmän alas ajamisen rajat, ja voidaanko korvaamista ajatella modernisoimisen keinona.

Tämä tutkielma toteutettiin systemaattisena kirjallisuuskatsauksena, jonka tavoitteena on esitellä aiheen ympäriltä tehtyä ajankohtaista tutkimusta. Tutkimuskysymykseksi sekä apututkimuskysymyksiksi muodostuivat seuraavat:

- Minkälaisia strategioita perinnejärjestelmien modernisoimiseen käytetään?
    - Mitä ovat perinnejärjestelmät?
    - Mitä tarkoitetaan perinnejärjestelmien modernisoimisella?

Tutkielma keskittyy tutkimaan vain perinnejärjestelmien modernisoimisen eri strategioita, joten ulkopuolelle jäävät muun muassa perinnejärjestelmän ylläpidon ulkoistaminen sekä perinnejärjestelmän jäädyttäminen. Jäädyttämisellä tarkoitetaan ylläpidon lopettamista, jolloin järjestelmä jää oman onnensa nojaan (Bennett, Ramage & Munro, 1999). Perinnejärjestelmien alasajo, eli niiden poistaminen käytöstä kokonaan on myös oma prosessinsa, jota tässä tutkielmassa ei käsitellä. Tutkielmassa ei myöskään tutkita modernisointiin johtavia tekijöitä, vaan keskitytään eri strategioiden kuvailemiseen.

Lähteitä haettiin useasta eri tietokannasta mukaan lukien muun muassa seuraavat: IEEE Xplore, ACM, Elsevier ja Google Scholar. Rajausta tuloksille tehtiin pääosin seuraavia hakusanoja käyttämällä ja yhdistelemällä: 'legacy', 'modernization', 'system', 'evolution' ja 'software'. Lisäksi lähteitä löytyi artikkeleiden lähdeluetteloista. Päämääränä oli etsiä nimenomaan "legacy" -hakusanalla, sillä perinnejärjestelmien ominaisuudet poikkeavat tavallisista järjestelmistä. Lähteiksi valikoitiin mahdollisimman tuoreita vertaisarvioituja tieteellisiä artikkeleita sekä konferenssijulkaisuja. Aiheesta tehdyt tutkimukset olivat pääosin konferenssijulkaisuja, joka vaikuttaa lähteiden laatuun. Vaikka tarkoituksena oli löytää tuoreita tutkimuksia aiheesta, on merkittävimpiä oivalluksia tehty jo 90-luvulla, joten esimerkiksi määritelmien käytössä on ollut järkevää hyödyntää myös vanhempaa tutkimusta.

Ensimmäisessä luvussa kartoitetaan perinnejärjestelmien ominaisuuksia. Luvussa keskitytään vertailemaan näiden järjestelmien yrityksille tuomia hyötyjä ja haittoja ja antamaan lukijalle kokonaiskuva perinnejärjestelmien olemuksesta, jotta perinnejärjestelmien uniikkius ja modernisoimisen tarve voidaan ymmärtää.

Toisessa luvussa kerrotaan aluksi mitä järjestelmien modernisoimisella tarkoitetaan ja esitellään tapoja luokitella modernisoimisen strategioita. Sen jälkeen esitellään neljä eri strategiaa perinnejärjestelmän modernisoimisen toteuttamiselle. Nämä strategiat ovat uudelleenmallinnus, kääriminen, migraatio sekä korvaaminen.

Viimeisessä luvussa vedetään yhteen tutkielman tulokset ja esitetään pohdintaa sekä aiheen ympäriltä, että mahdollisista jatkotutkimusvaihtoehdoista. Tutkielman lopusta löytyy vielä lähdeluettelo.

2 PERINNEJÄRJESTELMÄT

Tässä kappaleessa selvitetään, mitä perinnejärjestelmällä tarkoitetaan sekä esitellään mitä haittoja perinnejärjestelmistä koituu liiketoiminnalle ja organisaatiolle. Vastapainoksi avataan myös järjestelmien hyötyjä ja syitä sille, miksi perinnejärjestelmiä on sitkeästi ylläpidetty parhaimmillaan jopa vuosikymmeniä.

2.1 Perinnejärjestelmät ja niiden ominaisuudet

Teknologian kehittyessä ja liiketoiminnan tarpeiden muuttuessa ajan mittaan tulevat monet tietojärjestelmät tiensä päähän. Vanhentuneet ja jäykät järjestelmät muuttuvat kalliin ylläpidon vuoksi taakaksi. Bennett (1995, s. 19) on jo 90-luvulla kuvaillut perinnejärjestelmiä seuraavasti: ”suuria järjestelmiä, joiden kanssa ei pärjätä, mutta jotka ovat elintärkeitä organisaatiollemme”. Tästä voidaan siis päätellä, että perinnejärjestelmälle tyypillisiä ominaisuuksia ovat järjestelmän koko, muokkaamattomuus sekä kriittisyys liiketoiminnalle. Useissa tutkimuksissa perinnejärjestelmän määrittelemiseen käytetäänkin edellä mainittua Bennettin luonnehdintaa. Tutkimuksissa tunnistetut yleisimmät ominaisuudet:

- Vanha tai vanhentunut
- Liiketoiminnalle kriittinen
- Vaikeasti muokattavissa
- Suuri koko

Khadkan, Batlajeryn, Saeidin, Jansenin, ja Hagen (2014) tutkimuksessa kysyttiin perinnejärjestelmien kanssa työskennelleiltä asiantuntijoilta, mikä heidän mielestään on perinnejärjestelmän määritelmä. Haastatteluista kävi ilmi, että erityisesti järjestelmän vanhuus sekä kriittisyys liiketoiminnalle olivat perinnejärjestelmälle tyypillisiä ominaisuuksia. Yksi haastatelluista asiantuntijoista kommentoi asiaa vapaasti suomennettuna seuraavasti: ”Järjestelmää, joka on iäkäs,

vanhentunut mutta ei liiketoiminnalle kriittinen, ei ikinä kutsuttaisi perinnejärjestelmäksi.” (Khadka ym. 2014, s. 38).

Vanhuus perinnejärjestelmän ominaisuutena toistuukin lukuisissa tutkimuksissa. Pelkästään iältään vanha järjestelmä ei kuitenkaan automaattisesti ole perinnejärjestelmä. Hyvin usein perinnejärjestelmä on toteutettu vanhalla ohjelmointikielellä, kuten COBOLilla tai Fortranilla. Monen mielikuvissa mielletään perinnejärjestelmät yksinomaan vanhoiksi, 80-luvun DOS-käyttöjärjestelmän päällä pyöriviksi järjestelmiksi. Todellisuudessa perinnejärjestelmän ei tarvitse olla kymmentäkään vuotta vanha tullakseen kutsutuksi ”legacyksi”. Langer (2016) argumentoi, että mikä tahansa järjestelmä voi olla perinnejärjestelmä. Hänen mukaansa perinnejärjestelmillä on tyypillisiä piirteitä, mutta järjestelmän ei tarvitse täyttää kaikkia ominaisuuksia ollakseen ”legacy”. Termiä perinnejärjestelmä käytetään siis monenlaisista järjestelmistä (M’Baya, Laval & Moalla, 2017; Matthiesen & Bjørn, 2015).

Yksi perinnejärjestelmän tunnusmerkki on myös usein järjestelmän suuri koko, sillä vuosien saatossa lähdekoodiin tehdyt lisäykset ovat paisuttaneet järjestelmän pahimmillaan miljoonien koodirivien kokoiseksi. Toisaalta suuri koko ei ole ehto perinnejärjestelmän määritelmälle, vaan se voi olla myös pieni ja yksinkertaisesti toteutettu. Tällöin myös sen uudistaminen on helpompaa ja nopeampaa. (M’Baya ym., 2017).

Perinnejärjestelmiä luonnehditaan usein myös joustamattomiksi sekä vaikeasti muokattaviksi, joka osittain johtuu myös niiden suuresta koosta. Esimerkiksi Bisbalin, Lawless:n, Wu:n ja Grimsonin (1999, s. 103) mukaan Brodie ja Stonebraker (1996) ovat kirjassaan tiivistäneet perinnejärjestelmät seuraavasti: ”—perinnejärjestelmä on tietojärjestelmä, joka merkittävästi vastustaa muutoksia ja muutosta”. Tähän johtaa muun muassa se, että perinnejärjestelmillä on monesti riippuvuuksia useisiin eri järjestelmiin, jolloin muokkaaminen voisi herkästi rikkoa näitä riippuvuuksia (Matthiesen & Bjørn, 2015).

Täysin syyttä perinnejärjestelmät eivät ole päätyneet yritysten käyttöön vuosiksi, vaan niiden käyttämiselle löytyy päteviä perusteluita ja niiden ylläpitämisestä koetaan myös monia hyötyjä. Harva tutkimus käsittelee perinnejärjestelmien hyviä puolia, sillä lähtökohtaisesti perinnejärjestelmät nähdään kuluerinä ja digitalisaation hidastajina organisaatioissa. Tämän kirjallisuuskatsauksen tarkentavana tutkimuskysymyksenä on kuitenkin selvittää mitä perinnejärjestelmät ovat, joten myös hyviä puolia on hyvä tuoda esiin monipuolisen kokonaiskuvan saamiseksi.

Khadka ym. (2014) ovat harvoja, jotka ovat ottaneet tutkimuksessaan myös hyvät puolet mukaan. Tutkimuksessa haastateltiin noin 200 perinnejärjestelmien kanssa tekemisissä ollutta asiantuntijaa. Heidän vastauksistaan löydettiin neljä perinnejärjestelmiin liitettyä hyvää puolta: suorituskyky, luotettavuus, hyväksi todettu teknologia sekä kriittisyys liiketoiminnalle.

Kriittisyys liiketoiminnalle oli tutkimuksessa yleisin perinnejärjestelmistä koettu hyöty. Tutkimuksesta käy kuitenkin ilmi, että ominaisuus ei ole niinkään koettu hyöty, vaan ennemmin selitys sille, miksi perinnejärjestelmistä ei ole hankkiuduttu eroon. Tämä tarkoittaa sitä, että perinnejärjestelmien koetaan

tukevan yrityksen liiketoimintaa niin paljon, että niistä ei ole haluttu luopua. Tutkimuksessa käy ilmi, että haastateltavat kokivat perinnejärjestelmän tärkeänä ja tarpeellisena. Nämä tekijät luultavasti ovat johtaneet siihen, että järjestelmästä on tullut liiketoiminnalle kriittinen. Täten liiketoiminnan kriittisyyden sijaan koettu hyöty onkin se, että perinnejärjestelmä vastaa hyvin liiketoiminnan tarpeisiin.

Kolmas hyvä ominaisuus, joka tutkimuksessa mainitaan, on hyväksi todettu teknologia. Koska järjestelmät ovat usein vanhoja, ovat ne käyneet jo läpi useat testaukset ja laadunvarmistukset. Toisin sanoen teknologia on siis osoittautunut toimivaksi, joten siitä luopuminen saatetaan kokea riskinä järjestelmän toimivuuden kannalta. Haastateltavat kokivat järjestelmät myös vakaina ja niitä pidettiin saatavuuden suhteen varsin luotettavina.

Luotettavuus mainittiin myös yhdeksi koetuksi hyödyksi sillä perinnejärjestelmät ovat toimineet jo useiden vuosien ajan. Toisaalta luotettavuutta perusteltiin haastateltavien toimesta samoilla tavoilla kuin hyväksi todetun teknologian ominaisuutta, sillä myös luotettavuus ilmeni vakaana järjestelmänä sekä vuosien kehitystyön tuloksena. Tutkimuksessa mainittiin myös käyttäjänäkökulma: perinnejärjestelmät koettiin tutkimuksen osallistujien keskuudessa luotettaviksi, sillä niiden käyttämisestä on tullut tuttua ja helppoa vuosien saatossa. Myös Fürnweger, Auer ja Biffl (2016) toteavat, että perinnejärjestelmän säilyttämisen puolesta puhuu sen vakaus. Tutkijoiden mukaan järjestelmään uppoavien kulujen suuruus on helpompi arvioida, kun järjestelmä on vakaa. Järjestelmän modernisoimisen lopullista hintaa taas on vaikea arvioida, joten pysyminen vanhassa järjestelmässä voi olla turvallinen vaihtoehto.

Neljäntenä ja viimeisenä hyvänä ominaisuutena mainittiin suorituskyky, joka kuitenkin jakoi mielipiteitä haastateltavien joukossa. Kokonaisuudessaan vain neljäsosa haastateltavista näki suorituskyvyn perinnejärjestelmän etuna. Myönteisesti suorituskyvyn kokevat kommentoivat näitä perinnejärjestelmiä muun muassa nopeiksi. Myös suorittimen pieni teho verrattuna järjestelmän suuriin käyttömääriin nähtiin tehokkaana suorituskykynä. Voidaan päätellä, että joissakin tilanteissa perinnejärjestelmä voi suorituskyvyssä pärjätä myös nykyajan huomattavasti tehokykyisemmälle teknologialle, mutta ei usein.

2.2 Perinnejärjestelmien haitat

Kuten jo luvun alussa käytiin läpi, monet perinnejärjestelmien ominaisuudet ovat yksinomaan negatiivisia. Perinnejärjestelmiä ei modernisoidakaan pelkästään trendien aallonharjalla pysymisen vuoksi, vaan niiden ylläpitämiseen ja käyttämiseen liittyy lukuisia ongelmia, jotka aiheuttavat usein yrityksille taloudellisia menetyksiä tai hidastavat niiden kehitystä.

Useimmiten tutkimuksissa korostuvat perinnejärjestelmistä koituvat taloudelliset kulut: kuten johdannossakin jo todettiin, perinnejärjestelmistä koituu suuria kuluja yrityksille esimerkiksi ylläpito- ja työntekijäkulujen muodossa. Työntekijäkuluja koituu tavallista järjestelmää enemmän, sillä ylläpitäminen voi

vaatia erityistä asiantuntijuutta, kuten vanhan ohjelmointikielen tuntemista tai kokemuksen myötä syntynyttä ymmärrystä koko perinnejärjestelmästä ja sen toiminnallisuuksista (Fürnweger ym., 2016). Toisaalta täytyy ottaa huomioon, että perinnejärjestelmien tuomia todellisia kokonaiskuluja voi olla hyvin vaikea laskea. Myös tutkimusta perinnejärjestelmien aiheuttamista kuluista on tehty hyvin niukasti. Erilaisia laskelmia ja laskentakaavoja löytyy toki eri konsultti- ja ohjelmistoyritysten sivuilta, mutta niiden luotettavuutta on vaikea arvioida. Yhdysvalloissa tutkittiin kymmentä perinnejärjestelmää ja niiden muodostamia kuluja ja huomattiin hinnaksi tulevan vuosittain 334 miljoonaa dollaria, eli noin 33 miljoonaa dollaria yhtä perinnejärjestelmää kohden (United States Government Accountability Office, 2019). Toisaalta Yhdysvalloissa valtion järjestelmiä myös käyttää huomattavan moni henkilö, joka osaltaan selittää suuria kuluja. Yhtä kaikki, perinnejärjestelmän modernisoimisella voidaan saavuttaa parhaimmillaan miljoonien eurojen vuosittaiset säästöt. Esimerkiksi Sneedin ja Verhoefin (2020) tutkimuksessa yksittäisen perinnejärjestelmän modernisoimisella saatiin vuosittaiset kulut laskemaan yhteensä miljoonalla eurolla. Kulujen ohjautuminen perinnejärjestelmien ylläpitoon ja järjestelmätukeen toimivat merkittävänä hidasteena ketteryydelle ja on pois uusien innovaatioiden kehittämiseen kohdennetuista resursseista (van Oosterhout, Waarts & van Hillegersberg, 2006).

Yksi perinnejärjestelmien yleisimmistä ominaisuuksista, vaikeasti muokattavuus, on myös yksi sen merkittävimmistä haitoista (M'Baya ym., 2017). Tällä tarkoitetaan sitä, että perinnejärjestelmät ovat monimutkaisesti toteutettuja, niillä on paljon sekä sisäisiä että ulkoisia riippuvuuksia ja jatkokehittäminen sekä järjestelmäintegraatiot ovat vaikeita tai mahdottomia toteuttaa. Perinnejärjestelmät eivät siis jousta liiketoiminnan tarpeisiin.

Nykyaikaisten järjestelmien tulisi myös olla helposti muokattavissa muuttuvien säännösten ja lakien myötä (van Oosterhout ym., 2006). Esimerkiksi yksityisyyttä pyritään varjelemaan lakien ja säännösten avulla ja monet yritykset ovat joutuneet muokkaamaan järjestelmiään esimerkiksi GDPR-säännöksen myötä. Perinnejärjestelmille on myös tyypillistä, että vuosien ylläpidon ohessa niihin on lisätty ominaisuuksia edellisten päälle, joka on tehnyt näistä järjestelmistä paisuneita ja hyvin monimutkaisia ylläpitää ja osaltaan myös siksi niistä on tullut myös erittäin vaikeasti muokattavia (Hussain, Bhatti, & Rasool, 2017).

Hainaut, Cleve, Henrard ja Hick (2008) mainitsevat tutkimuksessaan joustamattomuudesta konkreettiseksi esimerkiksi sen, että perinnejärjestelmät ovat usein ohjelmoitu vastaamaan tarkkoihin kysymyksiin ja hakemaan hakuja vastaavia tuloksia: nykyisin monilta järjestelmiltä halutaan, että ne kykenisivät ilman viivettä hakemaan vastauksia ennalta määrittämättömiin kysymyksiin tai hakuihin. Esimerkiksi nykyaikaiset data-analytiikkatyökalut perustuvat pitkälti siihen, että ne ymmärtävät monimutkaisia hakuja ja osaavat tehdä niiden pohjalta ennustuksia (SAS, n.d.).

Joustamattomuuteen vaikuttaa olennaisesti se, että perinnejärjestelmät pohjautuvat usein vanhalle ohjelmointikielelle. Vanhojen ohjelmointikielien ongelma on ensinnäkin se, että niillä ei yksinkertaisesti ole mahdollista toteut-

taa nykyajan järjestelmien toimintoja. Toisekseen jos järjestelmä on toteutettu konekielisenä, voi sen ylläpito olla vaikeaa. Tämä johtuu siitä, että konekieli on itsessään jo hyvin jäykkää, sillä se kääntää ohjelmat suoraan biteiksi ja ohjaa suorittimen toimintaa. Konekielellä toteutetut käskyt ovat erittäin yksinkertaisia ja jäykkiä. Vanhoissa tietokoneissa käytetyt konekielet voivat olla niin eksplisiittisiä, ettei niitä voi tulkita ilman pitkää kokemusta ja osaamista. Koska näitä vanhoja ohjelmointikieliä ei käytetä enää uusien järjestelmien luomiseen, ei niitä enää juuri opeteta kouluissa. Tämän takia vanhojen ohjelmointikielien osaajien määrä on varsin rajallinen. Esimerkkinä vanhasta ohjelmointikielestä käytetään usein COBOLia, joka on konekieltä korkeampitasoinen kieli, eli se kääntää kirjoitetun koodin alempitasoiselle konekielelle. COBOL onkin ollut yleisessä käytössä vanhoissa järjestelmissä, sillä koko ohjelmointikieli on kehitetty jo noin 60 vuotta sitten. Vuonna 2014 arvioitiin, että tänä päivänä käytössä on vielä 220 miljardia riviä COBOLilla kirjoitettua koodia ja noin 80 prosenttia maailman liiketoimista suoritettaisiin COBOLilla pyörivien järjestelmien kautta (Beach, 2014). COBOL on huomattavasti joustavampi kuin konekielet, mutta senkin ongelmaksi on muodostunut se, ettei osaajia tahdo riittää. Vaikka monet järjestelmät pyörivät yhä tällä vanhalla kielellä, nuorille opetetaan kouluissa modernimpia ohjelmointikieliä, kuten Javaa, Pythonia tai C#:ia. Sandbornin ja Prabhakarin (2015) mukaan nuorien kiinnostus vanhojen ohjelmointikielien opettelua kohtaan on vähäistä. Yrityksille jää vaihtoehdoksi joko modernisoida järjestelmänsä tai toivoa, että osaajia riittää vielä tulevaisuudessa.

Perinnejärjestelmien suurimpia ongelmia lienee myös se, että ne pohjautuvat usein hyvin vanhalle teknologialle ja tekniikoille muutenkin kuin pelkän ohjelmointikielen osalta. Järjestelmä saattaa käyttää vanhentuneita tietokantoja ilman nykyään suosittua relaatiotietokannan logiikkaa tai tiedon hakemisessa helpotusta tuovaa tietokannanhallintajärjestelmää (Rahgozar & Oroumchian, 2003). Teknologia on myös kehittynyt huimaa vauhtia 2000-luvun aikana, joten ei vaadita paljoa, että järjestelmä ei enää vastaa nykyaikaisen liiketoiminnan tarpeisiin.

Osaajapula on ohjelmointikielien osaamisen lisäksi yksi perinnejärjestelmien myötä syntyvistä ongelmista (Sandborn & Prabhakar, 2015). Erityisesti siksi, että perinnejärjestelmille on myös tyypillistä, että niiden dokumentaatio on hoidettu huonosti, se ei ole ajan tasalla tai se on kokonaan hävinnyt vuosien saatossa (Matthiesen & Bjørn, 2015). Tämä tuo modernisoimisprosessiin haasteita, mutta myös vaikeuttaa perinnejärjestelmän ylläpitoa. Dokumentaation puuttuessa järjestelmän tunteminen on muutaman asiantuntijan varassa, jotka ovat vuosien saatossa oppineet perinnejärjestelmän toiminnallisuuksien ja arkkitehtuurin erityisosaajiksi. Sen lisäksi, että tällaisten osaajien palkkataso on korkea, ovat he myös tikittävä aikapommi. Kuten aikaisemminkin jo mainittu, monet perinnejärjestelmät ovat saaneet alkunsa 1970-, 1980- ja 1990-luvuilla, joka tarkoittaa sitä, että viimeisetkin osaajat eläköityvät seuraavan 20 vuoden kuluessa. Tämä saattaa johtaa siihen, että muuten toimiva perinnejärjestelmä saatetaan vastentahtoisesti ajaa alas, sillä työvoimapula on niin suurta (Hainaut ym., 2008).

Perinnejärjestelmiin liittyy myös usein teknovastarintaa, eli uuteen teknologiaan siirtymistä vastustetaan, pääosin järjestelmän kanssa työskentelevien toimesta. Teknovastarinta ei varsinaisesti ole perinnejärjestelmien huono ominaisuus, vaan enemmänkin seuraus sen käyttämisestä. Esimerkiksi Sneed ja Verhoef (2020) väittävät tutkimuksessaan, että mitä pidempään järjestelmä on käytössä ja mitä tottuneempia käyttäjät ovat vanhan järjestelmän käyttämiseen, sitä haluttomampia he ovat päästämään siitä irti. Ongelma hankaloittaa perinnejärjestelmästä poissiirtymistä, sillä teknovastarintaa saattaa esiintyä päättävissä tahoissa tai pelätään työntekijöiden reagoivan asiaan esimerkiksi irtisanoutumalla.

3 PERINNEJÄRJESTELMIEN MODERNISOIMINEN JA STRATEGIAT

Perinnejärjestelmien modernisoimiselle ei ole olemassa yhtä mallia, vaan näkemykset vaihtelevat runsaasti tutkijoiden välillä. Sekaisin menevät myös ylläpito, modernisointi sekä korvaaminen. Kokoavaa tutkimusta eri luokittelutavoista ja strategioista on hyvin vähän, ja ne ovat myös ristiriidassa toistensa kanssa. Päätös siitä, mitä perinnejärjestelmälle tehdään, on yritykselle usein haastava ja riskialtis. Tässä luvussa määritellään järjestelmän modernisoiminen sekä esitellään erilaisia konkreettisia tapoja uudistaa perinteisiä järjestelmä.

3.1 Perinnejärjestelmien modernisoiminen

Tutkimuksia, joissa vertailtaisiin erilaisia konkreettisia lähestymistapoja perinnejärjestelmien uudistamiseen, on vähän, sillä suurin osa tutkimuksesta keskittyy modernisoimiseen kehitetyn metodin tai jonkin tietyn strategian tekniseen kuvailuun. Suurin osa strategioita kuvaavista tutkimuksista on konferenssijulkaisuja ja niiden näkemykset poikkeavat toisistaan. Tämä kertoo tutkijoiden eriävistä näkemyksistä perinnejärjestelmien modernisoimien strategioista sekä tutkimuksen puutteesta. Myös käsitteitä on käytetty kuvaamaan eri asioita, joka osaltaan hankaloittaa kokonaiskuvan hahmottamista. Esimerkiksi perinnejärjestelmien modernisointi on tällainen käsite, jonka määritelmästä ei tunnu löytyvän yhtäläisiä näkemyksiä. Erityisesti tutkijayhteisöä jakaa kysymys siitä, lasketaanko järjestelmän osittainen tai kokonainen korvaaminen uudella järjestelmällä modernisoimiseksi. Käsitteiden päällekkäisyys ja hajanaisuus hankaloittavat esimerkiksi metatutkimuksen tekemistä, jonka myötä aiheen tutkimuksen luotettavuus kärsii.

Voidaan ajatella, että yrityksellä on muutamia vaihtoehtoja, joiden avulla kohdata perinnejärjestelmien tuomat ongelmat: jatkaa ylläpitoa, korvata järjestelmä uudella tai modernisoida se joko kokonaan tai osittain (Kankaanpää, Tiihonen, Ahonen, Koskinen, Tilus & Sivula, 2007; Hussain ym., 2017). Yrityksellä

on luonnollisesti vaihtoehtona myös luopua kokonaan järjestelmästä ajamalla se alas, jolloin tilalle ei hankita tai kehitetä korvaavaa järjestelmää. Perinnejärjestelmät ovat kuitenkin liiketoiminnan kannalta kriittisiä, joten voidaan olettaa, että yleensä järjestelmän täydellinen alasajo ei ole yrityksen kannalta järkevää.

Tässä tutkielmassa on tarkoituksena esitellä erilaisia modernisoimisen keinoja, joten on merkityksellistä rajata se, mitä modernisoimisella oikeastaan tarkoitetaan, jotta voidaan ymmärtää esimerkiksi ylläpitämisen ja järjestelmän uudistamisen ero. Harvassa tutkimuksessa käsitettä on erikseen määritelty, joka saattaa johtua siitä, ettei sille ehkä ole nähty tarvetta, sillä monesti tutkimuksissa käytetyt luokittelut ovat yksiselitteisiä. Bakar, Razali ja Jambari (2019, s. 2) määrittelevät perinnejärjestelmien modernisoimisen seuraavasti: ”--modernisoiminen päivittää ikääntyneet järjestelmät mahdollistaakseen kommunikoinnin uusimman teknologian kanssa”. Samassa tutkimuksessa todetaan modernisoimisen myös tähtäävän siihen, että järjestelmä pystyisi nopeammin vastaamaan alati muuttuvan liiketoiminnan tarpeisiin. Myös Agilar, Almeida ja Canedo (2016) ovat määritelleet modernisoimisen pyrkivän vastaamaan liiketoiminnan muuttuviin vaatimuksiin. Khadka ja kollegat (2015) määrittelevät järjestelmän modernisoimisen tapahtuvan silloin, kun perinteisillä ylläpidon toimilla ei enää voida päästä haluttuihin tuloksiin, eli järjestelmä ei enää pysty palvelemaan liiketoimintaa halutulla tavalla. Koskinen, Ahonen, Sivula, Tilus, Lintinen ja Kankaanpää (2005) taas näkevät modernisoimisen järjestelmään tehtävinä suurina muutoksina, jotka johtuvat joko teknisistä syistä tai liiketoiminnallisien prosessien muutoksista. Nämä tutkimukset siis näkevät modernisoimisen yhtenä järjestelmän evoluution vaiheena, joka on liiketoiminnan kannalta tärkeä. Tästä voidaan siis tehdä johtopäätös, että modernisoiminen on järjestelmän evoluution vaihe, joka pyrkii muokkaamaan järjestelmää paremmin liiketoiminnan vaatimuksiin vastaavaksi, kun tavalliset ylläpidon toimet eivät siihen enää riitä. Täytyy pitää myös mielessä, että modernisoiminen on aina prosessi, jossa tulee ottaa huomioon liiketoiminta kokonaisuutena sekä organisaation ja ympäristön näkökulma, jotta järjestelmän kriittiset ominaisuudet saadaan tehokkaasti tuotua myös uuteen järjestelmään tai teknologiseen ratkaisuun (Bakar ym., 2019).

Kuten luvussa kaksi todettiin, perinnejärjestelmät aiheuttavat lukuisia ongelmia ja esteitä liiketoiminnalle, joka usein herättää johdossa halua päästä niistä eroon. Modernisoimisen tuomista konkreettisista hyödyistä ei löydy myöskään juuri tutkimusta, vaikka hyötyjen tunnistaminen voisi toimia tärkeänä päätöksentekoa tukevana tekijänä. Khadka kollegoineen (2015, s. 478) tunnisti kymmenen modernisoimisen tuomaa hyötyä perinnejärjestelmien uudistamista koskevista tutkimuksista (taulukko 1). Huomattavasti suurimpana hyötynä havaittiin kulujen pieneneminen, joka selittyy pitkälti ylläpitoon ja henkilökuntaan uppoavilla resursseilla. Seuraavana listalla on lisääntynyt uudelleenkäytettävyys, jolla tarkoitetaan sitä, että vanhan järjestelmän osia pystytään kierrättämään. Tällöin esimerkiksi perinnejärjestelmän sisällään pitämät liiketoiminnalle arvokkaat tiedot voidaan säilyttää. Ketteryyden ja joustavuuden lisääntyminen käyvät myös yksiin van Oosterhoutin ja kumppaneiden (2006) havain-

non kanssa siitä, että perinnejärjestelmät toimivat yrityksen ketteryyden esteenä. Taulukko 1 esittää hyvin, että modernisoimisella voidaan saavuttaa hyvin monenlaisia etuja, mikäli se saadaan tehtyä onnistuneesti.

TAULUKKO 1 Modernisoimisesta koetut hyödyt tieteellisissä julkaisuissa (Khadka ym. 2015, s. 478)

Hyödyt	**Tieteellisissä julkaisuissa havaittu / määrä**
Kulujen pieneneminen	17
Lisääntynyt uudelleenkäytettävyys	8
Lisääntynyt ketteryys	8
Lisääntynyt joustavuus	7
Lisääntynyt suorituskyky	7
Lisääntynyt ylläpidettävyys	6
Kilpailukykyisenä pysyminen	6
Lisääntynyt saatavuus	6
Nopeampi pääsy markkinoille	5
Lisääntynyt yhteentoimivuus	5

Comella-Dorda, Wallnau, Seacord, ja Robert (2000) sekä Seacord, Plakosh ja Lewis (2003) käyttävät järjestelmän evoluution jaotteluun kolmea eri vaihetta: ylläpito, modernisoiminen sekä korvaaminen. Tutkimuksen mukaan modernisoiminen tarkoittaa järjestelmään tehtäviä merkittäviä muutoksia, joiden myötä kuitenkin merkittävä osa järjestelmästä säilytetään. Toisaalta ylläpidon ja modernisoimisen raja on häilyvä, sillä pienetkin muutokset järjestelmään voivat parantaa sen toimintaa huomattavasti. Esimerkiksi Fürnweger ja kumppanit (2016) esittävät, että ylläpitoa voisi olla kahta eri tyyppiä: yksinkertaista ylläpitoa sekä ylläpitoa, jossa on mukana uudelleenmallinnusta. Yksinkertainen ylläpito pitää sisällään vain välttämättömät toimet järjestelmän toiminnan takaamiseksi, kuten bugien korjaukset. Ylläpito, jossa on mukana uudelleenmallinnusta, taas tarkoittaa pieniä muutoksia, jotka tuovat pieniä parannuksia ohjelmiston toimintaan. Tällaisia muutoksia voivat olla esimerkiksi automaattisen testauksen lisääminen tai pienet esteettiset parannukset käyttöliittymään.

Erityisen paljon jakaa tutkijakuntaa kysymys siitä, lasketaanko järjestelmän korvaaminen kokonaan uudella järjestelmällä modernisoimisen keinoksi. Soliman ja Rinta-Kahila (2019) tutkimuksessaan muodostavat määritelmän tietojärjestelmän lakkauttamiselle (discontinuance), joka pitäisi heidän mukaansa sisällään myös uudella järjestelmällä korvaamisen. Agilarin ja kumppaneiden (2016) tutkimuksessa modernisointi nähtiin erillisenä prosessina järjestelmän korvaamisesta. Tutkimuksen mukaan korvaamista ei voida nähdä modernisoi-

misena, sillä järjestelmän toiminnallisuus muuttuu prosessissa, eikä kyse ole enää samasta järjestelmästä. Myös Koskisen ja kollegoiden (2005) käsitys modernisoimisesta käsitti vain erilaiset migraation, eli vanhojen tietojen siirtämisen uudelle alustalle, keinot jättäen ulkopuolelle uudella tuotteen korvaamisen sekä järjestelmän uudelleenkehityksen. Althani ja Khaddaj (2017) käsittelevät tutkimuksessaan modernisoimista migraation määritteen kautta. He jakavat migraation kuitenkin kolmeen luokkaan, josta yksi on täydellinen migraatio (complete migration), joka taas vastaa Sneedin ja Verhoefin (2020) sekä Ganesan & Chithralekhan (2016) näkemystä siitä, että järjestelmän korvaaminen on modernisoimisen strategia. Vaikka järjestelmän korvaamisesta uudella modernisoimisen keinona on ristiriitaisia mielipiteitä, käsitellään ne tässä tutkielmassa modernisoimisen strategiana, mikäli uuden järjestelmän toiminnallisuus on sama kuin vanhassa perinnejärjestelmässä. Näin ollen se on linjassa luvun alussa määritellyn modernisoimisen käsitteen kanssa. Seuraavassa luvussa käydään tarkemmin läpi sitä, mitä keinoja modernisoimiseen yleensä käytetään.

3.2 Strategioiden luokittelu

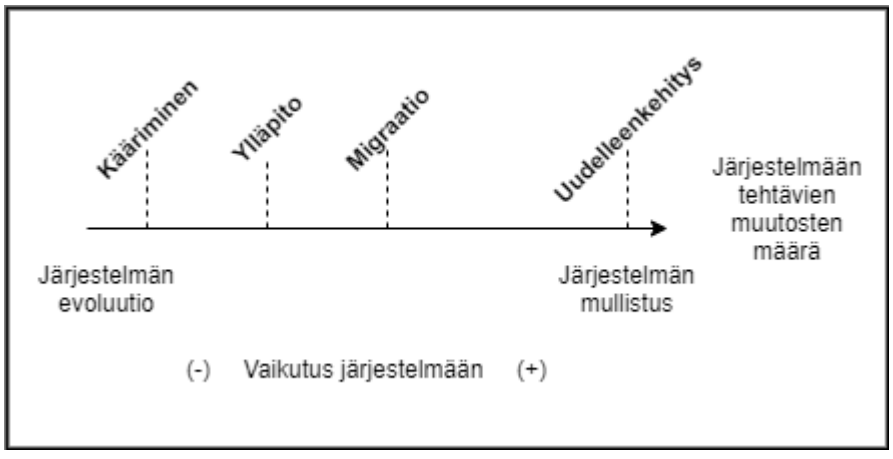
Modernisoimisen strategioihin löytyy useita eri luokittelutapoja, mutta alue kaipaisi vielä systemaattista kokoavaa tutkimusta niistä, sillä termejä käytetään sekaisin ja kokonaiskuvaa on vaikea hahmottaa. Tässä kappaleessa tarkoituksena on esitellä eri luokittelutapoja, jotta voidaan muodostaa mahdollisimman kattava kuva eri vaihtoehdoista. Lähteiksi on pyritty valitsemaan tuoreita tutkimuksia, sillä voidaan olettaa niiden ottavan paremmin huomioon myös nykyaikaisen teknologian. Aiheesta tehty tutkimus juontaa juurensa kuitenkin pitkälti 1990-luvulle, joten on mielekästä tarkastella asiaa myös vanhemman kirjallisuuden kautta. Täytyy myös ottaa huomioon, että perinnejärjestelmien modernisoiminen ei tapahdu aina vain yhtä strategiaa hyödyntäen, vaan monesti uudistaminen suoritetaan useampaa tapaa yhdistellen. Esimerkiksi Sneedin ja Verhoefin (2020) esittämä uudelleentoteutus yhdistelee automaattista kääntämistä sekä uudelleenkehittämistä. Weiderman, Northrop, Smith, Tilley ja Wallnau (1997) väittävät, että yhdistelmäratkaisu voisi olla jopa kaikista kustannustehokkain ratkaisu. Yhdistelmäratkaisu on todennäköisesti räätälöity juuri kohdejärjestelmälle ja sen uudistamiseen on voitu valita kohdennetut keinot, jolloin ylimääräistä työtä ei synny ja resursseja säästyy.

On myös tyypillistä, ettei modernisointia lähdetä tekemään kerralla kokonaan, vaan usein järjestelmää uudistetaan osa tai komponentti kerrallaan. Nykyajalle tyypillinen osittainen uudistus voi olla järjestelmän siirtäminen – eli migraatio – pilveen, jolloin välttämättä järjestelmän toiminnallisuus ei muutu ollenkaan, mutta järjestelmää päästään käyttämään selaimen kautta (Althani & Khaddaj, 2017). Näin ollen järjestelmä saadaan toimimaan uuden teknologian kanssa, vaikka varsinaiseen lähdekoodiin ei tehtäisi muutoksia.

Yksinkertaisimmillaan perinnejärjestelmien modernisoimisen keinot voi jakaa kahteen luokkaan: mustan laatikon (black-box) sekä valkoisen laatikon

(white-box) modernisoimiseen (Seacord ym., 2003). Mustan laatikon modernisoimisella tarkoitetaan uudistamista, joka ei vaadi järjestelmän syvällistä tuntemista. Tällä tarkoitetaan siis esimerkiksi käyttöliittymän parantamista, jolloin muutokset ovat ainoastaan pinnallisia. Valkoisen laatikon modernisoiminen sitä vastoin vaatii järjestelmän syvällistä tuntemista, jotta järjestelmän toiminnallisuus ei muutu alkuperäisestä uudistamisen myötä. Comella-Dorda ja kumppanit (2000) taas esittävät, että järjestelmän modernisoiminen voi tapahtua kolmella eri tasolla: toiminnallisella, datan ja/tai käyttöjärjestelmän tasolla. Luokittelu on hyvin samankaltainen Seacordin ja kumppaneiden (2003) tutkimuksen kanssa, mutta se ottaa huomioon myös järjestelmän sisältämään dataan tehtävät muutokset. Kummankin tutkimuksen luokittelu on hyvin yleisluontoinen ja tutkijoiden esittelemät luokittelutavat toimivatkin yläkäsitteinä konkreettisille modernisointitavoille.

Bisbal ja kollegat (1999) esittelevät tutkimuksessaan erilaisia tapoja vastata perinnejärjestelmien tuomiin ongelmiin. Kuvio 1 esittää nämä vaihtoehdot janalla, joka kuvaa järjestelmään tehtävien muutosten määrää. Eri vaihtoehdot asettuvat janalle sen mukaisesti, millainen vaikutus sillä on itse järjestelmään. Vähiten vaikutusta järjestelmään on ohjelman käärimisellä, jolloin ainoastaan järjestelmän käyttöliittymää eheytetään. Toisessa päässä taas on järjestelmän uudelleenkehittäminen, joka Bisbalin ja kumppaneiden (1999) mukaan voi tarkoittaa joko järjestelmän täydellistä uudistamista tai kokonaan uuden järjestelmän kehittämistä. Mielenkiintoista janassa on se, että tutkijoiden näkemyksen mukaan kääriminen aiheuttaa vähemmän muutoksia järjestelmään, kuin sen ylläpitäminen, sillä käyttöliittymän parantaminen voi muokata järjestelmää merkittävästikin käytettävyyden paranemisen muodossa. Kääriminen pienissä määrin voidaan nähdä myös osana normaalia järjestelmän ylläpitoa (Fürnweger ym., 2016). Tämä luokittelu perustuu siis yksinomaan järjestelmään tehtävien muutosten määrään ja kokoon.



KUVIO 1 Perinnejärjestelmän evoluutio Bisbalin ja kumppaneiden (1999) mukaisesti

Samaa ajatusta hyödyntävät myös Althani ja Khaddaj (2017), jotka esittävät, että uudistamiseen on neljä eri tapaa: kääriminen, täydellinen migraatio, osittainen migraatio sekä inkrementaalinen migraatio. Täydellinen migraatio tarkoittaa

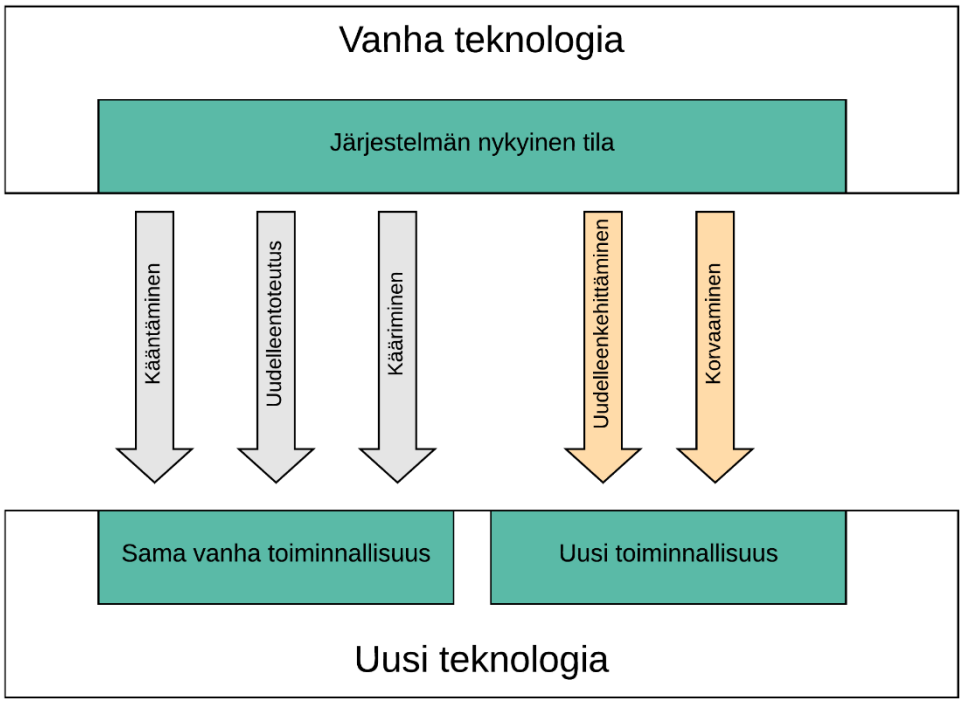
heidän mukaansa uuden järjestelmän kehittämistä alusta alkaen, eli vanhasta perinnejärjestelmästä ei siirretä uudelle alustalle mitään. Inkrementaalinen migraatio tarkoittaa vähitellen tapahtuvaa migraatiota, eli hiljalleen tapahtuvaa uudistamista. Osittaisella migraatiolla taas tarkoitetaan esimerkiksi yksittäisen komponentin modernisoimista. Modernisoiminen tapahtuu kuitenkin niin, että suurin osa järjestelmästä pysyy yhtä muuttamattomana. Luokittelu on selkeä, sillä se pohjautuu Bisbalin ja kumppaneiden (1999) tapaan sille, kuinka paljon muutoksia järjestelmään tehdään. Toisaalta inkrementaalisen ja osittaisen migraation raja ei ole selkeä: inkrementaalinen migraatio on myös osittain tapahtuvaa migraatiota, sillä se tapahtuu myös usein osa kerrallaan. Inkrementaalinen migraatio on kuitenkin ehkä hyvä erotella, sillä Althanin ja Khaddajin (2017) artikkelissa esitellään siihen pohjaavia metodeja. Näin ollen inkrementaalinen migraatio on siis oma modernisoimisen haaransa. Myös tämä tutkimus kuvailee kattotermejä, joiden alle mahtuu useampia eri tapoja modernisoida järjestelmä.

Fürnwegerin ja kollegoiden (2016) tutkimuksessa puhutaan järjestelmän evoluutiosta modernisoimisen sijaan, mutta näitä voidaan pitää synonyymeinä toisilleen. Tutkijoiden luokittelun mukaan järjestelmän evoluutio voidaan jakaa neljään: kahteen ylläpidon keinoon sekä kahteen migraation keinoon. Kappaleessa 3.1 pohdittiin jo kahta ylläpidon vaihetta, joista toinen – ylläpito takaisinmallinnuksella – voi olla modernisoimisen keino esimerkiksi käyttöliittymän eheyttämisen kautta. Oleellisimpia tämän tutkielman kannalta ovat kuitenkin tutkimuksessa esitellyt kaksi migraation keinoa: kova ja pehmeä migraatio. Pehmeä migraatio on kevyt migraatio, jonka tarkoituksena on hyödyntää mahdollisimman paljon vanhan järjestelmän osia. Käytännössä pehmeällä migraatiolla siis tarkoitetaan uudelleenmallinnusta, jolla pyritään tekemään korjaavia toimenpiteitä järjestelmään kuitenkaan muuttamatta sen toiminnallisuutta. Kovan migraation tarkoitus on sitä vastoin muokata järjestelmää radikaalimmin, esimerkiksi rakentamalla se kokonaan alusta. (Fürnweger ym., 2016). Myös Hussain kollegoineen (2017) käyttää modernisoimiseen Fürnwegerin ja kumppaneiden (2016) kaltaista jakoa: järjestelmää voidaan uudistaa vaiheittain, osa kerrallaan tai kokonaan alusta kehittäen.

Edellä mainitut lähteet kuvaavat yleisluontoisesti eri modernisoimisen strategioita, mutta myös yksityiskohtaisempia tutkimuksia löytyy. Esimerkiksi Jain ja Chana (2015) ovat jakaneet modernisoimisen keinot neljään: osittainen tai kokonaan korvaaminen, kääriminen, uudelleenmallinnus/-kehitys sekä migraatio. Tässäkin luokittelussa mukana on myös migraatio, jolla kuitenkin tarkoitetaan tässä tapauksessa järjestelmän siirtämistä uudelle alustalle, kuten esimerkiksi pilvipalveluun. Tutkijoiden mukaan siirtäminen vaatii sekä uudelleenmallintamisen keinoja että käärimistä. Kyseessä on siis oikeastaan jälleen yhdistelmä useampaa modernisoimisen keinoa.

Sneed ja Verhoef (2020) ovat avanneet kenties kattavimmin modernisoimisen strategioita. Kuvio 2 havainnollistaa uudistamisstrategioiden jakoa Sneedin ja Verhoefin (2020, s. 3) mukaan Sneedin, Wolfin ja Heilmannin (2010) näkemyksen mukaisesti. Malli kuvaa eri strategioiden aiheuttamia muutoksia vanhaan perinnejärjestelmään. Kuvan mukaisesti modernisoimisella tähdätään

siihen, että järjestelmä saadaan toimimaan uudella teknologialla ja näin vastaamaan paremmin alati muuttuviin liiketoiminnan tarpeisiin. Kuvion kolme harmaata nuolta kuvaavat eri migraatiovaihtoehtoja. Migraatiolla tarkoitetaan tässä yhteydessä ohjelmiston datan ja toiminnallisuuksien siirtämistä uuteen alustaan tai ympäristöön. Eli järjestelmän toiminnallisuus ei muutu, mutta se toteutetaan uudella teknologialla. Oranssit nuolet taas viittaavat järjestelmän korvaamiseen, jolloin koko järjestelmän toiminnallisuus muuttuu uudistusprosessin myötä.



KUVIO 2 Perinnejärjestelmän modernisointistrategiat (Sneed, Wolf & Heilmann, 2010)

Yhteenvetona voidaan siis todeta, että eri tapoja modernisoimiseen on lukuisia, eikä yhtenäistä eri tapoja kokoavaa ja vertailevaa tutkimusta ole juuri tehty. Taulukko 2 kokoaa tässä luvussa käsitellyt tutkimukset ja niissä esitetyt strategiat yhteen, josta voidaan huomata, että strategioita on lukuisia. Täytyy kuitenkin pitää mielessä, että eri termeillä saatetaan tarkoittaa täysin samoja asioita, kuten aikaisemmin huomattiin. Esimerkiksi migraation käsite vaihtelee useamman keinon kattavasta yläkäsitteestä (Bisbal ym., 1999; Althani & Khaddaj, 2017) yksittäisen keinon kuvailuun (Jain & Chana, 2015).

TAULUKKO 2 Yhteenveto modernisoimisen strategioista

Lähde	Esitetyt modernisoimisen strategiat
Bisbal, Lawless, Wu & Munro (1999)	• Kääriminen • Migraatio • Uudelleenkehitys
Join & Chana (2015)	• Kääriminen • Migraatio • Uudelleenmallinnus / -kehitys • Korvaaminen
Fürnweger, Auer & Biffl (2016)	• Pehmeä migraatio • Kova migraatio
Hussain, Bhatti & Rasool (2017)	• Uudistaa osia järjestelmästä • Uudelleenkehitys / korvaaminen
Althani & Khaddaj (2017)	• Kääriminen • Täydellinen migraatio • Inkrementaalinen migraatio • Osittainen migraatio
Sneed ja Verhoef (2020)	• Kääriminen • Uudelleenkehitys • Korvaaminen • Uudelleentoteutus • Kääntäminen

Taulukosta voidaan kuitenkin huomata, että kääriminen, migraatio sekä uudelleenkehitys ovat toistuvia teemoja tutkimuksissa. Näiden strategioiden pohjalta voidaan muodostaa seuraavanlainen luokittelutapa, joka kattaa tutkimusten mainitsemat keinot:

- Uudelleenmallinnus (re-engineering)
- Kääriminen (wrapping)
- Migraatio (migration)
- Korvaaminen (replacement)

Tähän luokitteluun päädyttiin, sillä useat keinoista olivat joko yleisluontoisia, kuten esimerkiksi pehmeä ja kova migraatio, kun taas osa keinoista oli yhdistelmä kahta tai useampaa yllä mainituista strategioista, kuten esimerkiksi uudelleentoteutus. Vaikka eri tapoja yhdistelevät modernisoimisen keinot ovatkin varmasti hyviä vaihtoehtoja, on tässä tutkielmassa haluttu kuitenkin rajata vaihtoehdot mahdollisimman tarkasti, jotta saadaan kattava yleiskuva erilaisten strategioiden luonteesta. Tutkielmassa käytettyjen artikkeleiden mainitsemat eri strategiat voidaan kukin jakaa jonkun ylläesitetyn strategian alle. Esimerkiksi Sneedin ja Verhoefin (2020) esittämä kääntäminen voidaan nähdä yhtenä uudelleenmallinnuksen keinona.

Oikean strategian valitseminen on hyvin yritys- ja järjestelmäkohtaista ja riippuu monista seikoista. Tässä tutkielmassa keskitytään kuitenkin vain esittelemään eri vaihtoehtoja, joten strategioiden vertailua keskenään ei varsinaisesti tehdä. Seuraavat alaluvut esittelevät tiiviisti kunkin keinon kirjallisuudessa esitettyjä hyviä ja huonoja puolia tarkoituksena antaa yleiskuva modernisoimiseen käytetyistä strategioista.

3.2.1 Uudelleenmallinnus

Usein perinnejärjestelmien toiminnallisuuksissa ja lähdekoodissa on sellaisia osia, joita halutaan siirtää myös uudistettuun järjestelmään, jolloin on järkevää lähteä luomaan uudistuksia vanhaan järjestelmään pohjautuen. Tällaista modernisoimisen lähestymistapaa kutsutaan uudelleenmallinnukseksi. Chikofsky ja Cross (1990) määrittelevät uudelleenmallinnuksen kohdejärjestelmän tutkimiseksi ja muokkaamiseksi, jonka tarkoituksena on rakentaa järjestelmä uudessa muodossa. Käytännössä siis uudelleenmallinnus tarkoittaa parantavien muutosten toteuttamista moderneja teknologioita ja ohjelmointikieliä hyödyntäen järjestelmään niin, että sen perimmäinen toiminnallisuus ei muutu. Uudelleenmallinnus voi tapahtua pienissä osissa tai koko järjestelmä kerrallaan. (Chikofsky & Cross, 1990; Majthoub, Qutqut & Odeh, 2018).

Uudelleenmallinnusta tehtäessä on tärkeää tuntea hyvin järjestelmä, jota lähdetään uudistamaan. Järjestelmän tuntemisella tarkoitetaan sen olemassa olevan dokumentaation, lähdekoodin, rakenteiden ja arkkitehtuurin tarkkaa analysointia, jotta uudelleenmallinnus on ylipäätään mahdollista suorittaa vanhat toiminnallisuudet säilyttäen (Chikofsky & Cross, 1990; Majthoub ym., 2018). Tätä vaihetta kutsutaan usein termillä takaisinmallinnus. Takaisinmallinnuksen avulla määritellään myös uuden järjestelmän vaatimukset. Perinnejärjestelmille on tyypillistä, että dokumentaatio on puutteellista, tai sitä ei ole laisinkaan, jolloin takaisinmallinnuksen merkitys uudelleenmallintamisen onnistumiselle korostuu (Majthoub ym., 2018). Chikofskyn ja Crossin (1990) mukaan uudelleenmallinnus pitää aina sisällään myös jonkinasteista takaisinmallinnusta.

Nykyään uudelleenmallinnusta voidaan tehdä myös automaattisesti, jolloin käännösohjelma kääntää suoraan järjestelmän lähdekoodin vanhasta ohjelmointikielestä uuteen. Sneed ja Verhoef (2020) ovat tutkineet tätä ja käyttävät prosessista nimitystä kääntäminen (translating). Prosessi on automaattinen ja kääntäminen tapahtuu erillisen työkalun avulla koodirivi kerrallaan. Ratkaisu on helppo ja luultavasti myös halvin keino toteuttaa järjestelmän uudistaminen, mutta usein monimutkaisille perinnejärjestelmille menetelmä ei ole kovin käytännöllinen. Ensinnäkin järjestelmä voi olla kirjoitettuna niin vanhalla tai alhaisen tason kielellä, että sille ei ole olemassa työkalua joka kieltä osaisi tulkita. Tällöin luonnollisesti menetelmää ei voi käyttää ilman lisäkustannuksia aiheuttavaa kääntäjän kehitystyötä. Toisinaan myös moderni ohjelmointikieli ei välttämättä tue osaa vanhan ohjelmiston toiminnoista, jolloin virheelliset lausekkeet täytyy käydä manuaalisesti läpi aiheuttaen ylimääräistä työtä. Toisekseen koodi voi myös olla laadultaan huonoa, eli alkuperäisessä kehitystyössä ei ole

esimerkiksi osattu tehdä ohjelmasta mahdollisimman suorituskykyistä tai tehokasta. Mikäli tällaista koodia sisältävä ohjelma käännetään suoraan, siirtyvät suorituskyvyn ongelmat myös uudistettuun järjestelmään. Modernit ohjelmointikielet pitävät sisällään myös ominaisuuksia, joita monet vanhat ohjelmointikielet eivät kykenisi toteuttamaan. Tämä tarkoittaa siis sitä, että kaikkia uuden teknologian tuomia mahdollisuuksia ei ketterästi voitaisi implementoida uudistettuun järjestelmään. (Sneed ja Verhoef, 2020).

Uudelleenmallinnus toimii ikään kuin kattoterminä useille eri modernisoimisen strategioille, sillä kaikki tässä tutkielmassa mainitut keinot vaativat jonkin verran uudelleen- ja takaisinmallinnusta, ainoastaan järjestelmään koituvan vaikutuksen määrä vaihtelee. Koska perinnejärjestelmästä halutaan usein säästää vanha toiminnallisuus, on järjestelmässä jo olevien ominaisuuksien tunteminen tärkeää. Tämän takia takaisinmallinnus on useissa uudistamisstrategioissa merkittävässä roolissa; järjestelmää on vaikea lähteä uudistamaan, jos sen toimintalogiikkaa ei ymmärretä. Toisaalta dokumentaation ja automatisoinnin puute tekee takaisinmallinnuksesta usein kallista ja vaikeaa toteuttaa (Bennett & Rajlich, 2000). Uudelleenmallinnus onkin hyvä vaihtoehto, kun järjestelmästä halutaan uudistaa vain osia. Majthoub ja kollegat (2018) esittävät, että uudelleenmallinnusta kannattaa harkita, kun järjestelmän rakenne ja/tai dokumentaatio on heikkolaatuista, järjestelmää ei voi enää ylläpitää, järjestelmän ylläpitämiseen ei löydy enää asiantuntijoita tai jos koko järjestelmän korvaaminen on liian kallista tai riskialtista. Toisaalta Hussain kollegoineen (2017) esittävät, että järjestelmän osien uudelleenkäyttö – eli tässä tapauksessa uudelleenmallinnus – vähentää järjestelmässä esiintyviä virheitä merkittävästi. Modernisointikeinona uudelleenmallinnus voi siis olla myös tehokas.

Koska uudelleenmallinnus kattaa hyvin laaja-alaisesti erilaisia tapoja uudistaa järjestelmää, vaihtelee siitä koituvat kulutkin sen mukaisesti. Sneedin (2005) mukaan uudelleenmallinnusprojekti eroaa kuitenkin tavanomaisesta ohjelmistoprojektista, sillä projektiin uppoavat kulut ovat helpompi arvioida etukäteen. Tämä johtuu siitä, että projektin laajuus on helpompi arvioida, kun kehitystyö pohjautuu olemassa olevaan järjestelmään. Järjestelmän vaatimukset ovat myös tiedossa olemassa olevaan järjestelmään pohjautuen, joten resursseja säästyy myös vaatimusmäärittelyn osalta. Järjestelmän osittainen uudistaminen voi myös olla riskialtista, mikäli järjestelmässä on paljon riippuvuuksia. Tällöin muutos järjestelmän yhteen komponenttiin voi häiritä koko järjestelmän toimintaa ja pahimmillaan aiheuttaa katkoksia järjestelmän saatavuuteen (Althani & Khaddaj, 2017).

Yksi uudelleenmallinnuksen keinoista on uudelleenkehitys, eli samanlaisen järjestelmän kehittäminen kokonaan uusiksi uusia teknologioita ja ohjelmointikieliä hyödyntäen. Saman järjestelmän kehittämien uudelleen ei yleensä ole yritykselle kannattavaa. Sneed ja Verhoef (2019) listaavat viisi tilannetta, jolloin uudelleenmallinnus ei ole yleensä paras ratkaisu. Ensinnäkin järjestelmä voi olla sellainen, jolle ei yksinkertaisesti nähdä enää käyttöä tai sen merkitys liiketoiminnalle on kadonnut. Toisaalta ensimmäisessä kappaleessa todettiin jo, että perinnejärjestelmät ovat liiketoiminnan kannalta kriittisiä, joten tällaisen

järjestelmän kohdalla modernisoimisen strategioita ei tarvitsisi edes miettiä. Toisekseen vanhan järjestelmän toiminnallisuudet voivat olla jo riittäviä, ettei uuden ohjelmiston kehittämiselle ole oikeasti tarvetta. Kolmantena on mainittu uudelleenkehittämisestä koituva hinta, jolloin jonkin toisen strategian valitsemalla voitaisiin samaan lopputulokseen päästä halvemmalla. Neljäntenä mainitaan ohjelmistokehitysprosessiin kuluva aika. Koska järjestelmän tekeminen aloitetaan alusta, muistuttaa prosessi normaalia ohjelmistokehitysprojektia, joka on työläs ja pitkäkestoinen. Pitkä uudelleenkehitysprosessi voi viedä myös niin paljon aikaa, että sen valmistuessa se on jo vanhentunutta teknologiaa (Bisbal ym., 1999). Viidentenä mainitaankin, että ilman kunnollisia perusteluita uudelleenkehitykselle koko projektilla on suuri riski epäonnistua, kuten suurimmalle osalle isoista ohjelmistoprojekteista käykin (Bisbal ym., 1999; Sneed ja Verhoef, 2019).

Toinen esimerkki uudelleenmallinnuksen keinosta on uudelleentoteutus. Uudelleentoteutus modernisoimisen strategiana sijoittuu uudelleenkehittämisen ja kääntämisen välimaastoon. Sneedin ja Verhoefin (2019) mukaan uudelleentoteutuksella tarkoitetaan vanhan ohjelmointikielen automaattista kääntämistä modernille ohjelmointikielelle perinnejärjestelmän logiikkaan ja dokumentaatioon perustuen. Siinä missä uudelleenkehittämisessä sama järjestelmä rakennetaan alusta manuaalisella työllä ja konvertoiminen suoritetaan täysin automaattisesti, uudelleentoteutus hyödyntää molempia keinoja. Toisin sanoen uudelleentoteutus on siis osittain automatisoitu prosessi, jonka rinnalla tehdään myös manuaalista työtä. Tämän strategian ehdoton etu on se, että vanhan järjestelmän toiminnallisuudet säilyvät, mahdolliset vialliset tai heikkolaatuiset koodipätkät saadaan korjattua ja samalla käyttöön saadaan uusin teknologia ja järjestelmä ilman perinnejärjestelmien heikkouksia.

Luonnollisesti huonona puolena uudelleentoteutuksella on sen hinta ja sen vaatima lisätyö. Hintaa nostaa järjestelmän koko, dokumentaation puute tai heikkous sekä osaavat asiantuntijat. Tässä tavassa korostuu erityisesti toisessa luvussa käsitelty osaajapula; järjestelmän ymmärtäminen vaatii sen kielen osaamista. Uudelleentoteutuksen kanssa työskentelevän tulee vanhan kielen lisäksi olla myös kohdekielen asiantuntija. Tällaiset asiantuntijat ovat harvassa, sillä modernit ja vanhat ohjelmointikielet voivat poiketa syntaksiltaan, logiikaltaan ja algoritmien osalta hyvin paljon toisistaan. Osaajapula luokin riskin strategian onnistumiselle, sillä koko projekti voi mennä pieleen, jos kehitystyötä tekevät ihmiset eivät ole tarpeeksi hyviä. (Sneed & Verhoef, 2019).

3.2.2 Kääriminen

Yksi tapa tuoda järjestelmää nykyaikaisemmaksi, on olla koskematta varsinaiseen lähdekoodiin ja uudistaa järjestelmää pienin ehostavin parannuksin. Tällöin vanha perinnejärjestelmä pysyy yhä toiminnassa, mutta käytettävyys paranee ja järjestelmään saadaan lisättyä pieniä teknologisia parannuksia (Bisbal ym., 1999). Esimerkiksi komentopohjainen käyttöliittymä saatetaan korvata graafisella käyttöliittymällä, jolla on usein positiivisia vaikutuksia käytettävyy-

teen. Tällöin järjestelmän toiminnallisuus pysyy samana, mutta järjestelmästä saadaan astetta modernimpi. Tätä tapaa kutsutaan yleisesti käärimiseksi ja se on yksi yleisimmistä modernisoimisen keinoista (Bakar ym., 2019). Aiemmin tässä luvussa listattiin modernisoimisen luokittelukeinoja, joista myös suurin osa mainitsi käärimisen yhtenä strategiana, joka osaltaan kertonee strategian suosiosta.

Vaikka tavallisin tapa kääriä perinnejärjestelmä lienee käyttöliittymän modernisoiminen, voi käärimistä tehdä myös järjestelmän muille osille. Rahgozar ja Oroumchian (2003) esittävät, että käärimistä tapahtuu kolmella eri tasolla: käyttöliittymätasolla, datan tasolla sekä toiminnallisuuden tasolla. Myös Comella-Dorda ja kumppanit (2000) näkevät, että sekä data että käyttöliittymä voidaan kääriä. Käytännössä tämä tarkoittaa erilaisten rajapintojen hyödyntämistä, jolloin perinnejärjestelmän ominaisuuksia voidaan käyttää uusilla alustoilla tai niihin voidaan integroida uusia ominaisuuksia. Esimerkiksi datan käärimisen tarkoitus on yhdistää perinnejärjestelmän käyttämiä tietokantoja erillisen rajapinnan kautta; näin saadaan helpotettua dataan käsiksi pääsyä. Sen lisäksi käärimisen avulla voidaan järjestelmän ja datan väliin asettaa rajapintoja esimerkiksi statistiikan keräämiseen tai tietokannan toiminnan valvomiseen. Näin perinnejärjestelmä saadaan kytkettyä moderniin teknologiaan. (Thiran, Hainaut, Houben & Benslimane, 2006; Comella-Dorda ym., 2000).

Käyttöliittymän ja datan lisäksi voidaan siis Rahgozarin ja Oroumchianin (2003) mukaan myös perinnejärjestelmän toiminnallisuutta kääriä. Tällä tarkoitetaan sitä, että myös järjestelmän toiminnallisuuteen voi päästä käsiksi esimerkiksi toisen, uudemman, järjestelmän kautta. Käytännössä siis luodaan rajapinta perinnejärjestelmän ja ulkopuolisen järjestelmän välille, jolloin ne saadaan kommunikoimaan keskenään. Canfora, Fasolino, Frattolillo ja Tramontana (2008) tutkimuksessaan hyödyntävät tätä ajatusta koko perinnejärjestelmän käärimiseksi, joka tutkimuksen mukaan on kustannustehokas tapa modernisoida järjestelmä.

Strategia on muihin keinoihin verrattuna varsinkin riskitön, sillä järjestelmän toiminnallisuus ja data säilyy koskemattomana ja samalla saadaan parannettua käytettävyyttä (Comella-Dorda ym., 2000). Täten se on myös luultavasti helpoin tapa uudistaa järjestelmä, sillä se ei vaadi esimerkiksi muinaisten ohjelmointikielien, kadotettujen dokumentaatioiden tai paisuneiden ja monimutkaisten koodinpätkien läpikäymistä. Kääriminen voi myös monissa tapauksissa olla halvin vaihtoehto, sillä ohjelmointityötä on varsin vähän verrattuna koko järjestelmän modernisoimiseen. Käärimisestä koituvat kuluvat ovat usein pienemmät kuin muissa strategioissa, sillä työmäärä on pienempi (Bisbal ym., 1999). Kääriminen voidaan myös suorittaa monesti ainakin osittain automaattisesti, jolloin asiantuntijatyöstä ei synny kuluja (Althani & Khaddaj, 2017; Sneed & Verhoef, 2020). Tämä strategia ei kuitenkaan poista varsinaista ongelmaa, sillä periaatteessa käytössä on edelleen sama vanha perinnejärjestelmä, joka vaatii kallista ylläpitoa ja on todennäköisesti suorituskyvyltään heikko (Comella-Dorda ym., 2000; Jain & Chana, 2015; Rahgozar & Oroumchian, 2003). Perinnejärjestelmä on myös käärimisen jälkeen edelleen riippuvainen vanhan ohjel-

mointikielen ja järjestelmän lähdekoodin osaajista (Sneed & Verhoef, 2020). Vaikka siis strategian kulut ovat pienet, kokonaiskustannukset voivat olla suuret vaativan ylläpidon tarpeen takia (Bisbal ym., 1999). Käärityt komponentit vaativat myös paljon testausta, jotta järjestelmä toimii kokonaisuudessaan toivotulla tavalla, joka osaltaan myös lisää kustannuksia (Sneed & Verhoef, 2020).

Sen lisäksi kääriminen voi tehdä ennestään joustamattomasta perinnejärjestelmästä vielä joustamattomamman, kun vanhan lähdekoodin päälle lisätään uusia kerroksia (M'Baya ym., 2017). Käärimistä suositellaankin vain väliaikaiseksi ratkaisuksi esimerkiksi siksi aikaa, kun pohditaan, miten järjestelmä lopulta halutaan modernisoida vai ajetaanko se lopullisesti alas (Bisbal ym., 1999; Sneed & Verhoef, 2020).

On myös syytä miettiä, voiko tapaa edes kutsua uudistamisen keinoksi, sillä perinnejärjestelmään tehdään niin vähän muutoksia. Pienet esteettiset korjaukset voidaan ajatella tavallisena järjestelmän ylläpitona, kuten esimeriksi Seacordin ja kumppaneiden (2003) sekä Fürnwegerin ja kumppaneiden (2016) tutkimuksissa todettiin. Huono käytettävyys tai huono käyttöliittymä yksistään, ei tee järjestelmästä perinnejärjestelmää, kuten tutkielman toisessa kappaleessa todettiin, joten mikäli kääriminen tähtää vain käytettävyyden parantamiseen, voidaan se ajatella normaalina järjestelmän toimintaa tukevana ylläpitona.

3.2.3 Migraatio

Yksi tapa uudistaa perinnejärjestelmää, on siirtää se uudelle alustalle, jolloin vanha järjestelmä saadaan ainakin osittain toimimaan uuden teknologian kanssa. Migraatio on käsitteenä laaja-alainen ja monesti sitä käytetään kuvaamaan esimerkiksi uudelleenmallinnuksen keinoja (Bisbal ym., 1999). Esimerkiksi Althani ja Khaddaj (2017) jakavat modernisoimisen keinot täydelliseen, osittaiseen ja inkrementaaliseen migraatioon, joista yksikään ei kuitenkaan pidä sisällään perinnejärjestelmän siirtämistä uudelle alustalle. Myös Sneed ja Verhoef (2020) näkevät, että migraatio käsittäisi useampia keinoja. Heidän mukaansa esimerkiksi kääntäminen, uudelleentoteutus ja kääriminen ovat migraation keinoja. Bisbal kollegoineen (1999) argumentoi kuitenkin, että migraatiolla tarkoitetaan perinnejärjestelmän tai sen komponenttien siirtämistä uuteen ympäristöön ilman, että järjestelmän toiminnallisuus muuttuu. Tutkijoiden mukaan migraation pyrkimyksenä on säästää mahdollisimman paljon vanhasta järjestelmästä ja näin välttää uudelleenkehittämisen tuomat haasteet ja kulut. Tätä määritelmää käytetään myös tässä tutkielmassa. Kenties yksi yleisimmistä tavoista hyödyntää migraatiota nykypäivänä on siirtää perinnejärjestelmä pilvipalveluun. Käytännössä tämä siis tarkoittaa esimerkiksi perinteisen tietokoneelle asennettavan ohjelmiston tuomista selaimen kautta käytettäväksi.

Migraation suurin etu on se, että sen avulla perinnejärjestelmään saadaan upotettua uusia teknologisia ratkaisuja (Gholami, Daneshgar, Beydoun & Rabhi, 2017). Uuteen alustaan siirryttäessä voidaan myös parantaa järjestelmän käyttäjäystävällisyyttä esimerkiksi paremman saatavuuden kautta (Fürnweger ym., 2016). Bisbal ja kumppanit (1999) väittävät myös, että migraation avulla järjes-

telmän tunteminen paranee, ylläpito helpottuu ja kulut laskevat, mutta väitöksille ei ole kuitenkaan artikkelissa annettu perusteita. Migraation vaatima takaisin- ja uudelleenmallinnustyö viittaa ennemmin suuriin kuluihin.

Kuten uudelleenmallinnuksessa ja käärimisessäkin, migraatio voidaan tehdä koko järjestelmälle tai vain jollekin järjestelmän komponentille (M'Baya ym., 2017). Myös migraatio vaatii järjestelmän takaisin- ja uudelleenmallinnusta, jotta perinnejärjestelmä saadaan sovitettua uuteen alustaan (Bisbal ym., 1999; Jain & Chana, 2015). Järjestelmän takaisinmallinnuksella saavutetaan ymmärrys järjestelmän toiminnasta ja riippuvuuksista, joka toimii pohjana migraation onnistumiselle (Bisbal ym., 1999). Koska migraatio vaatii järjestelmän tuntemista ja ohjelmointityötä, voi se viedä aikaa sekä resursseja yllättävänkin paljon (M'Baya ym., 2017). Migraatioprosessin ajallinen kesto on myös riski yrityksen toiminnalle, sillä pitkä katkos toiminnassa liiketoiminnalle kriittisessä järjestelmässä tarkoittaa häiriöitä yrityksen toimintaan. Tätä riskiä voidaan minimoida rakentamalla väliaikainen varajärjestelmä tai -ratkaisu (Gholami ym., 2017). Varajärjestelmän tai -ratkaisun rakentaminen tarkoittaa kuitenkin lisätyötä ja näin ollen myös lisääntyviä kuluja. Migraatiota on myös hankala automatisoida, sillä sekä perinnejärjestelmät että kohdealustat ovat teknologioiltaan monimuotoisia (Jain & Chana, 2015). Automatisoinnin puute puolestaan lisää työn ja kulujen määrää.

Järjestelmän tuntemista voi vaikeuttaa myös perinnejärjestelmän asiantuntijoiden puute, jolloin työ myös hidastuu merkittävästi, kun järjestelmän toimintaa täytyy lähteä takaisinmallintamaan perinpohjaisesti (Fürnweger ym., 2016). Migraatio on keinona myös siksi vaikea, että perinnejärjestelmille luontainen joustamattomuus hankaloittaa uudelle alustalle siirtymistä. Toisinaan tämän joustamattomuuden takia migraatio vaatii uusien työkalujen kehittämistä, jotta järjestelmä saadaan toimimaan uuden alustan teknologian kanssa (Bisbal ym., 1999). Yritys tai organisaatio, joka migraation kuitenkin haluaa suorittaa, joutuu tekemään paljon taustatyötä onnistumisen eteen, sillä yhtä kaikille sopivaa tapaa suorittaa migraatio ei ole (Bisbal ym., 1999). Tämä tarkoittaa siis sitä, että valmiita malleja ei joko ole, tai ne eivät ole tapaukseen soveltuvia.

Toisinaan koko järjestelmän migraatio ei ole mahdollista, joka tuo osaltaan haasteita prosessiin. Tällaisia syitä voivat olla esimerkiksi tietoturvallisuus, yksityisyydensuojaa koskeva lainsäädäntö sekä pilvipalveluiden tietoverkkoongelmiin liittyvät syyt (Gholami ym., 2017). Järjestelmän jakautuminen kahdelle alustalle tarkoittaa sitä, että myös ylläpidon tarve jakautuu, joka taas tietää lisääntyviä kuluja (Fürnweger ym., 2016). Tällaisessa tapauksessa perinnejärjestelmän modernisoiminen jää vajaaksi, ja varsinaisia perinnejärjestelmän aiheuttamia ongelmia ei kuitenkaan ole ratkaistu.

Yksi pilvipalveluun siirtymisen eduista on sen skaalautuvuus. Pilvipalvelun skaalautumisella tarkoitetaan sitä, että järjestelmän käyttöön vapautuu resursseja, kuten muistikapasiteettia, tarpeen mukaan. Skaalautumisella saadaan siis optimoitua resursseja sinne, missä niitä tarvitaan. Perinnejärjestelmien arkkitehtuurin joustamattomuus toimii kuitenkin skaalautumisen esteenä (Gholami ym., 2017). Toisaalta joustamattomuus on molemminpuolista; myös pilvi-

alustat ovat usein teknologialtaan uniikkeja ja siksi taipuvat huonosti perinnejärjestelmien tarpeisiin (Gholami ym., 2017, Jain & Chana, 2015). Skaalautuvuuden saavuttamiseksi voidaan siis joutua tekemään runsaasti ohjelmointityötä kummassakin päässä.

Pilvipalveluun siirtymiseen liittyy myös muita haasteita. Kun järjestelmä siirretään palveluntarjoajan pilvipalveluun, on järjestelmä riippuvainen tästä kolmannesta osapuolesta, jolloin järjestelmän toiminnallisuus saattaa kärsiä epäyhteensopivuuksien takia (Gholami ym., 2017). Palveluntarjoajan valitseminen sisältää myös riskejä, sillä palveluntarjoajien määrä on rajallinen (Jain & Chana, 2015) ja niiden tarjoamien pilviratkaisujen sisältämä teknologia vaihtelee paljon. Palveluntarjoajien joukosta on vaikea valita sopiva ehdokas, joka luo riskin siitä, että esimerkiksi valitaan markkinoinnin paremmin taitava yritys teknologisesti yhteensopivamman yrityksen sijaan (Fürnweger ym., 2016). Ennalta on vaikea arvioida sitä, mihin teknologiseen ratkaisuun perinnejärjestelmä on helpointa sovittaa (Gholami ym., 2017).

3.2.4 Korvaaminen

Korvaaminen on erityisesti strategia, joka usein mielletään järjestelmän alasajona modernisoimisen sijaan. Kokonaan korvaamisella tarkoitetaan perinnejärjestelmän korvaamista uudella ja modernilla järjestelmäkokonaisuudella (Sneed & Verhoef, 2020). Korvaamisella tarkoitetaan tässä tutkielmassa nimenomaan korvaamista ostetulla valmiilla järjestelmällä. Korvaamista itsekehitetyllä järjestelmällä kutsutaan termillä uudelleenkehittäminen, ja sitä käsitellään kappaleessa 3.2.1. Voidaan ajatella, että järjestelmän korvaamista on ainakin kahdenlaista. Ensimmäisessä tavassa järjestelmä korvataan toisella, samat toiminnallisuudet sisältävällä järjestelmällä. Esimerkiksi vanhan Internet Explorerin vaihto modernimpaan Google Chromeen on konkreettinen esimerkki tällaisesta korvaamisesta. Toisessa tavassa uusi järjestelmä eroaa vanhasta järjestelmästä merkittävästi esimerkiksi toiminnallisuuksiltaan. Tällainen esimerkki voisi vaikka olla vanhan CRM-järjestelmän korvaaminen isolla ohjelmistokokonaisuudella, johon CRM on integroituna. Tässä osiossa käsitellään vain ensimmäisen tavan kaltaisia, eli alkuperäisen järjestelmän kaltaiseen järjestelmään siirtymistä, sillä useimmat lähteet käsittelevät vain sitä.

Yksi suurimmista ongelmista järjestelmän korvaamisessa uudella on siinä, että vanhan järjestelmän sisältämä data menetetään samalla. Järjestelmä voi sisältää paljon liiketoiminnan kannalta arvokasta tietoa, jota tullaan myöhemmin kaipaamaan. Järjestelmän sisältämän liiketoimintakriittisen tiedon menettäminen voi tarkoittaa sitä, että järjestelmä ei enää vastaa yhtä tehokkaasti liiketoiminnan tarpeisiin (Jain & Chana, 2015). Perinnejärjestelmät ovat myös jo pitkän elinikänsä aikana testattu useaan kertaan ja ovat sen myötä hyvin vakaita. Tämä vakaus kuitenkin menetetään, kun järjestelmä korvataan täysin uudella tuotteella.

Uusi järjestelmä vaatii siis paljon testaamista vakauden takaamiseksi, eikä se ole läheskään yhtä luotettava kuin vanha perinnejärjestelmä (Seacord ym.,

2003). Tämä tarkoittaa sitä, että kun halutaan korvata järjestelmä kokonaan, tulee varautua siihen, että uuden järjestelmän ylläpitokulut voivat ollakin korkeammat. Huolellinen käyttöönotto ja testaus vaativat myös asiantuntijaresursseja, joka on pois organisaatioiden muilta IT-projekteilta (van Oostehout ym., 2006).

Uuden järjestelmän toiminnallisuus ei välttämättä vastaa vanhaa, joka taas tarkoittaa sitä, ettei järjestelmä vastaa enää sen liiketoiminnan asettamiin odotuksiin (Seacord ym., 2003). Usein korvaaminen on kannattava vaihtoehto vasta sitten, kun muut modernisoimisen keinot ovat joko liian hankalia tai kalliita toteuttaa (M'Baya ym., 2017). Korvaaminen voi olla myös silloin vaihtoehto, kun perinnejärjestelmän käyttäjät eivät ole liian kiintyneitä joihinkin vanhan järjestelmän ominaisuuksiin. Usein käyttäjät ovat kuitenkin tottuneet perinnejärjestelmiin ja niiden ominaisuuksiin ja vaativat myös uudelta järjestelmältä tiettyjä samoja ominaisuuksia (Sneed & Verhoef, 2019). Joissakin tapauksissa ohjelmiston pienimuotoinen räätälöinti voi auttaa tähän ongelmaan, mutta ei läheskään aina. Korvaamista harkittaessa tulisi siis ottaa aina myös käyttäjänäkökulma huomioon.

Korvaaminen voi silti monesti olla yritykselle houkuttelevin vaihtoehto. Siirtyminen valmiiseen järjestelmään voi käytännön tasolla sujua kohtuulliseen hintaan ja pienellä vaivalla verrattuna esimerkiksi uudelleenkehittämiseen, johon liittyy aina ohjelmistoprojektin epäonnistumisen riski (Sneed & Verhoef, 2020). Toisaalta toisten tutkimusten mukaan korvaaminen voi tulla erittäin kalliiksi ja siksi kustannustehokkuutta on hyvä arvioida ennen strategian valitsemista (Jain & Chana, 2015; Pérez-Castillo, de Guzmán, & Piattini, 2011).

4 YHTEENVETO

Perinnejärjestelmät ovat monelle yritykselle kallis ongelma, joista luopuminen ei ole itsestäänselvyys. Modernisoiminen on riskialtis prosessi, joka voi vaarantaa liiketoiminnalle kriittisen järjestelmän toiminnan. Tässä tutkielmassa tarkoituksena oli selvittää perinnejärjestelmien ominaisuuksia sekä kartoittaa eri keinoja niiden modernisoimiseksi. Kandidaatintutkielman tutkimuskysymykset olivat seuraavanlaiset:

- Minkälaisia strategioita perinnejärjestelmien modernisoimiseen käytetään?
    - Mitä ovat perinnejärjestelmät?
    - Mitä tarkoitetaan perinnejärjestelmien modernisoimisella?

Perinnejärjestelmän käsiteen kiteyttää parhaiten Bennett (1995): ”suuria järjestelmiä, joiden kanssa ei pärjätä, mutta jotka ovat elintärkeitä organisaatiollemme”. Muut tutkimukset vahvistivat tätä käsitystä; vanha, joustamaton ja liiketoiminnalle kriittinen järjestelmä on perinnejärjestelmä. Perinnejärjestelmien hyviä puolia on dokumentoitu todella vähän, vaikka on selvää, ettei hankalia järjestelmiä ylläpidettäisi ilman järkevää perustetta. Toisaalta huonojen puolien määrä puhuu modernisoimisen kannattavuuden puolesta. Kriittisyys liiketoiminnalle tekee modernisoimisesta riskialtista koko liiketoiminnalle, joka osaltaan voi selittää miksi joissakin organisaatioissa käytetään pahimmillaan 50 vuoden ikäisiä tietojärjestelmiä (United States Government Accountability Office, 2019). Monet hyvistä puolista olivat mainittuna myös perinnejärjestelmien huonoissa puolissa, joten eri ominaisuuksia on vaikea mustavalkoisesti lajitella hyviin ja huonoihin. Haittoja oli mainittu suuressa osassa käyttämistäni lähteistä ja usein toistuivat samat haitat: kallista ja vaikeaa ylläpitää, dokumentaatio on heikkoa tai sitä ei ole ja osaajista on pulaa. Monet järjestelmät pohjautuvat vanhalle ohjelmointikielelle, jolle voi olla vaikeaa tai mahdotonta enää löytää osaajia. Esimerkiksi konekieli voi olla eksklusiivisesti tietylle laiteelle kehitetty, jolloin ainoa vaihtoehto on yrittää tulkita koodia. Myös COBOL, yksi tunne-

tuimmista vanhoista ohjelmointikielistä, on yli 60 vuotta vanhaa ja sen osaajia on yhä vähemmän.

Tärkeänä havaintona kirjallisuuskatsausta koottaessa huomattiin, että aiheesta tehty tutkimus pohjautuu pitkälti 1980- ja 1990-luvun tutkimuksille. Teknologia on ehtinyt kehittyä huimasti tuolta ajalta, joten tutkimusten sopiminen nykyajan järjestelmien vaatimuksiin on huomionarvoinen seikka. Toisekseen 2010-luvulla tehty tutkimus on pitkälti konferenssijulkaisuja ja vertaisarvioituja tieteellisiä artikkeleita on hyvin harvasti. Tutkimusten sisältö poikkesi toisistaan myös jossain määrin, eikä kokoavaa metatutkimusta juuri löytynyt. Sen lisäksi, termejä käytettiin kuvaamaan eri asioita, joka hankaloittaa edellä mainitun metatutkimuksen tekemistä.

Modernisoimisen käsite on hyvin jakautunut tutkijayhteisössä. Erityisesti ylläpidon termiä käytettiin usein päällekkäin modernisoimisen termin kanssa. Esimerkiksi kääriminen modernisoimisen keinona nähtiin toisinaan normaalina ylläpidon tehtävänä (Fürnweger ym., 2016; Bisbal ym., 1999). Modernisoiminen tähtää kuitenkin siihen, että perinnejärjestelmä saataisiin kommunikoimaan uuden teknologian kanssa (Bakar ym., 2019). Näin ollen kääriminen voidaan hyvin nähdä modernisoimisen keinona. Toisaalta voitiin myös todeta, että pienet esteettiset muutokset käyttöliittymään voidaan nähdä normaalina ylläpidon toimenpiteenä. Raja ylläpidon ja modernisoimisen välillä ei siis aina ole selkeä.

Kenties tärkeimpänä tutkielman tuloksena vertailtiin eri luokittelutapoja ja löydettiin neljä yhdistävää modernisoimisen strategiaa: uudelleenmallinnus, kääriminen, migraatio ja korvaaminen. Koska eri strategiavaihtoehtoja oli paljon ja niistä käytettiin erilaisia termejä, ei tässä tutkielmassa esitetty luokittelu välttämättä kata kaikkia keinoja, joita on olemassa. Tässä tutkielmassa käytettyjen lähteiden mainitsemat keinot voidaan kuitenkin jakaa näiden kategorioiden alle.

Strategioiden vertailun myötä selvisi, että strategioiden välillä on suuria eroja ja ne on selkeästi suunniteltu erilaisille ja erikokoisille järjestelmille, sekä erilaisiin tarpeisiin. Esimerkiksi käärimistä pidettiin ehdottomasti vain väliaikaisena ratkaisuna pienien ongelmien korjaamiseksi, kun taas korvaaminen nähtiin isona projektina, joka toimii vain harvoille yrityksille. Tämä korostaa strategioiden tutkimisen tärkeyttä: väärän strategian valinta voi koitua yritykselle kalliiksi. Esimerkiksi uudelleenkehittäminen on käytännössä täysi ohjelmistoprojekti ja useassa tutkimuksessa on todettu, että suurin osa ohjelmistoprojekteista epäonnistuu. Perinnejärjestelmät ovat liiketoiminnalle kriittisiä, joten vanhaa järjestelmää joudutaan hyvin todennäköisesti ylläpitämään samaan aikaan pahimmillaan vuosia kestävän uudelleenkehityksen kanssa. Se, millä perusteella modernisoimisen strategia valitaan, on monimutkainen prosessi ja vaatii usean tekijän huomioonottamista (Sneed & Verhoef, 2020). Oikean strategian ratkaisu on haastavaa, sillä modernisoimisesta koituvat kustannukset ja siihen uppoava aika vaihtelevat paljon järjestelmästä ja yrityksestä riippuen. Ongelmaan on kehitetty erilaisia laskukaavoja sekä metodeja, mutta laajuutensa vuoksi se on jonkin toisen tutkielman aihe.

Kirjallisuushaun aikana tuli huomattua, että perinnejärjestelmien tutkimusta on tehty erittäin paljon 1990-luvun ja 2000-luvun taitteessa. Myös monet viimeisen viiden vuoden aikana tehdyt tutkimukset tuntuivat käyttävän lähteinään näitä artikkeleita ja teoksia tuolta ajalta. Tämä voisi tarkoittaa sitä, että esimerkiksi tuoreiden case-tutkimusten avulla voitaisiin selvittää nykyajan perinnejärjestelmien piirteitä ja ovatko ne muuttuneet 1990-luvun järjestelmistä. Mielenkiintoista voisi olla myös pohtia eri aikakausien perinnejärjestelmien välisiä eroja: mikä on muuttunut ja mikä on pysynyt samana? Tällaisesta tutkimuksesta voisi olla hyötyä esimerkiksi yrityksille, jotka eivät tahdo ajautua perinnejärjestelmän ylläpitämiskierteeseen. Samalla voitaisiin ennakoida tietojärjestelmien kehitystä tulevaisuudessa.

Mielestäni tutkimusta voisi myös keskittää todellisten kulujen vertailemiseen eri strategioiden välillä. Tällä hetkellä kustannuksia on vertailtu modernisoimisen ja järjestelmän alasajon välillä, mutta mielenkiintoista olisi tehdä useampia case-tutkimuksia ja vertailla eri strategioiden suhteutettuja kokonaiskuluja. Tähän tutkielmaan oli vaikea löytää perinnejärjestelmien aiheuttamia kuluja, vaan usein kalleutta perusteltiin yrityksen ylläpitokuluilla, jotka pitivät sisällään myös muiden järjestelmien, kuin perinnejärjestelmien normaalit ylläpitokulut. Toisaalta kuluarvioita voisi löytyä laajentamalla hakusanojen käyttöä. Esimerkiksi ”decommission cost” ja ”migration cost” -hakutermit tuovat molemmat tuloksia, jotka pitävät luultavasti sisällään myös perinnejärjestelmien kululaskelmia.

Tutkimuksissa keskitytään monesti myös erilaisten strategioiden yksityiskohtaiseen läpikäymiseen, johon hyödyllisenä lisänä voisi olla tarkastella modernisoimista kokonaisuutena, eli millainen prosessi perinnejärjestelmän uudistaminen on ja millaisia vaikutuksia sillä on liiketoimintaan sekä organisaatioon.

Perinnejärjestelmien tutkimus voisi myös hyötyä kvalitatiivisesta tutkimuksesta: kuinka paljon mitäkin modernisointivaihtoehtoa käytetään ja millä todennäköisyydellä projekteissa onnistutaan. Mielenkiintoista voisi olla myös kuulla, kuinka tietoisia yritysmaailmassa eri modernisointistrategioista ollaan ja vastaavatko ne akateemisen maailman näkemyksiä.

LÄHTEET

Agilar, E., Almeida, R., & Canedo, E. (2016). A Systematic Mapping Study on Legacy System Modernization. Teoksessa *Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering*. San Fransisco, California. https://doi.org/10.18293/seke2016-059

Althani, B., & Khaddaj, S. (2017). Systematic Review of Legacy System Migration. *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)* (154-157). Anyang, China. https://doi.org/10.1109/dcabes.2017.41

Bakar, H. K. A., Razali, R., & Jambari, D. I. (2019). Implementation Phases in Modernisation of Legacy Systems. Teoksessa *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS)*. Johor Bahru, Malaysia. doi: 10.1109/ICRIIS48246.2019.9073628

Beach, G. (2.10.2014). Cobol Is Dead. Long Live Cobol! The Wall Street Journal. Haettu osoitteesta https://blogs.wsj.com/cio/2014/10/02/cobol-is-dead-long-live-cobol/

Bennett, K. (1995). Legacy systems: Coping with success. *IEEE Software*, *12*(1), 19-23. doi:10.1109/52.363157

Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. Teoksessa *Proceedings of the Conference on The Future of Software Engineering - ICSE '00* (73-87). Limerick, Ireland. doi:10.1145/336512.336534

Bennett, K. H., Ramage, M., & Munro, M. (1999). Decision model for legacy systems. *IEE Proceedings - Software*, *146*(3), 153. https://doi.org/10.1049/ip-sen:19990617

Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: issues and directions. *IEEE Software*, *16*(5), 103–111. https://doi.org/10.1109/52.795108

Brodie, M., & Stonebraker, M. (1996). *Migrating legacy systems: Gateways, Interfaces & The Incremental approach*. Morgan Kaufmann.

Canfora, G., Fasolino, A. R., Frattolillo, G., & Tramontana, P. (2008). A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. *Journal of Systems and Software*, *81*(4), 463–480. https://doi.org/10.1016/j.jss.2007.06.006

Chikofsky, E. J., & Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE software*, *7*(1), 13-17. doi: 10.1109/52.43044

Comella-Dorda, S., Wallnau, Seacord, & Robert. (2000). A survey of black-box modernization approaches for information systems. Teoksessa *Proceedings International Conference on Software Maintenance* (ICSM-94). San Jose, California. https://doi.org/10.1109/icsm.2000.883039

Fürnweger, A., Auer, M., & Biffl, S. (2016). Software Evolution of Legacy Systems - A Case Study of Soft-migration. Teoksessa *Proceedings of the 18th International Conference on Enterprise Information Systems*. ICEIS. https://doi.org/10.5220/0005771104130424

Ganesan, A. S., & Chithralekha, T. (2016). A Survey on Survey of Migration of Legacy Systems. Teoksessa *Proceedings of the International Conference on Informatics and Analytics (ICIA-16)* . https://doi.org/10.1145/2980258.2980409

Gholami, M. F., Daneshgar, F., Beydoun, G. & Rabhi, F. (2017). Challenges in migrating legacy software systems to the cloud — an empirical study. *Information systems*, *67*, 100-113. doi:10.1016/j.is.2017.03.008

Hainaut, J.-L., Cleve, A., Henrard, J., & Hick, J.-M. (2008). Migration of Legacy Information Systems. *Software Evolution*, 105–138. Berlin: Springer. https://doi.org/10.1007/978-3-540-76440-3_6

Hussain, S. M., Bhatti, S. N., & Ur Rasool, M. F. (2017). Legacy system and ways of its evolution. *2017 International Conference on Communication Technologies (ComTech)*. doi: 10.1109/comtech.2017.8065750

Jain, S. & Chana, I. (2015). Modernization of Legacy Systems: A Generalised Roadmap. *ICCCT '15: Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*. http://dx.doi.org/10.1145/2818567.2818579

Kankaanpää, I., Tiihonen, P., Ahonen, J., Koskinen, J., Tilus, T., & Sivula, H. (2007). Legacy system evolution - A comparative study of modernisation and replacement initiation factors. *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007).* (280-287). INSTICC Press. doi:10.5220/0002378102800287

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., & Hage, J. (2014). How do professionals perceive legacy systems and software modernization? *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. https://doi.org/10.1145/2568225.2568318

Khadka, R., Shrestha, P., Klein, B., Saeidi, A., Hage, J., Jansen, S., … Bruntink, M. (2015). Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies. *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. doi: 10.1109/icsm.2015.7332499

Koskinen, J., Ahonen, J., Sivula, H., Tilus, T., Lintinen, H., & Kankaanpaa, I. (2005). Software Modernization Decision Criteria: An Empirical Study. *Ninth European Conference on Software Maintenance and Reengineering*. https://doi.org/10.1109/csmr.2005.50

Langer, A. M. (2016). Legacy Systems and Integration. *Guide to Software Development*, 179–213. London: Springer. https://doi.org/10.1007/978-1-4471-6799-0_10

Majthoub, M., Qutqut, M. H., & Odeh, Y. (2018). Software Re-engineering: An Overview. *2018 8th International Conference on Computer Science and Information Technology (CSIT)*. doi:10.1109/csit.2018.8486173

Matthiesen, S., & Bjørn, P. (2015). Why Replacing Legacy Systems Is So Hard in Global Software Development. *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15*. https://doi.org/10.1145/2675133.2675232

M'baya, A., Laval, J., & Moalla, N. (2017). An assessment conceptual framework for the modernization of legacy systems. *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*. https://doi.org/10.1109/skima.2017.8294120

Pérez-Castillo, R., de Guzmán, I. G. & Piattini, M. (2011). Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer standards and interfaces*, *33*(6), 519-532. doi:10.1016/j.csi.2011.02.007

Rahgozar, M. & Oroumchian, F. (2003). An effective strategy for legacy systems evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, *15*(5), 325-344. doi:10.1002/smr.278

Sandborn, P. A., & Prabhakar, V. J. (2015). The Forecasting and Impact of the Loss of Critical Human Skills Necessary for Supporting Legacy Systems. *IEEE Transactions on Engineering Management*, *62*(3), 361–371. https://doi.org/10.1109/tem.2015.2438820

SAS (n.d.). Big Data Analytics - What it is and why it matters. Haettu 15.5.2020 osoitteesta https://www.sas.com/en_us/insights/analytics/big-data-analytics.html

Seacord, C. R., Plakosh, D., & Lewis, G. A. (2003). The Legacy Crisis. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. 1-17. Boston: Addison-Wesley.

Sneed, H. M. (2005). Estimating the costs of a reengineering project. Teoksessa *12th Working Conference on Reverse Engineering (WCRE'05)* (111-119). IEEE.

Sneed, H. M., Wolf, E., & Heilmann, H. (2010). *Softwaremigration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung*. Heidelberg: Dpunkt-Verl.

Sneed, H. M., & Verhoef, C. (2019). Re-implementing a legacy system. *Journal of Systems and Software*, *155*, 162–184. doi: 10.1016/j.jss.2019.05.012

Sneed, H. M., & Verhoef, C. (2020). Cost-driven software migration: An experience report. *Journal of Software: Evolution and Process*. *32*(7), 1-26. https://doi.org/10.1002/smr.2236

Thiran, P., Hainaut, J., Houben, G. & Benslimane, D. (2006). Wrapper-based evolution of legacy information systems. *ACM Transactions on Software Engineering and Methodology* (TOSEM), 15(4), 329-359. doi:10.1145/1178625.1178626

United States Government Accountability Office. (2019). Agencies Need to Develop Modernization Plans for Critical Legacy Systems (Raportti nro. GAO-19-471). Haettu osoitteesta https://www.gao.gov/assets/700/699616.pdf

Soliman, W., & Rinta-Kahila, T. (2020). Toward a refined conceptualization of IS discontinuance: Reflection on the past and a way forward. *Information & Management*, *57*(2), 103-167. https://doi.org/10.1016/j.im.2019.05.002

van Oosterhout, M., Waarts, E. & van Hillegersberg, J. (2006). Change factors requiring agility and implications for IT. *European journal of information systems*, *15*(2), 132-145. doi:10.1057/palgrave.ejis.3000601

Weiderman, N., Northrop, L., Smith, D., Tilley, S., & Wallnau, K. (1997). *Implications of Distributed Object Technology for Reengineering* (Tech. No. CMU/SEI-97-TR-005). Carnegie Mellon University: Software Engineering Institute. doi:10.21236/ada326945