Joni Laari

LEGACYJÄRJESTELMIEN TIETOTURVA- JA LIIKETOIMINTARISKIT SEKÄ MODERNISOINTI

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2023

TIIVISTELMÄ

Laari, Joni
Legacyjärjestelmien tietoturva- ja liiketoimintariskit sekä modernisointi
Jyväskylä: Jyväskylän yliopisto, 2023, 25 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja(t): Kyppö, Jorma

Tietojärjestelmiä on käytetty laajamittaisesti niin esimerkiksi liiketoiminnan kuin yhteiskunnalle välttämättömien prosessien hallinnassa jo niin kauan, että monet yhä päivittäisessä käytössä olevat tärkeät tietojärjestelmät ovat teknisesti vanhentuneita. Ilmiön kokoluokkaa kuvaa, että näiden tuotantokäytössä olevien järjestelmien arvellaan sisältävän maailmanlaajuisesti jopa 200 miljardia riviä lähdekoodia. Järjestelmien modernisointi tai korvaaminen on perusteltua niin ydinliiketoiminnallisista, taloudellisista kuin tietoturvanäkökulmista. Monimutkaisten tietojärjestelmien migraatio tai uudistaminen ei ole aina lainkaan suoraviivaista.

Asiasanat: legacyjärjestelmät, refaktorointi, yritysjärjestelmät, kyberturvallisuus, operaatioturvallisuus

ABSTRACT

Laari, Joni
Security and business risks in legacy information systems, and their modernization
Jyväskylä: University of Jyväskylä, 2023, 25 pp.
Information Systems, Bachelor's Thesis
Supervisor(s): Kyppö, Jorma

Information systems have been used on a large scale, for example, in the management of processes necessary for business and society for so long that many important information systems still in daily use are technically outdated. The magnitude of the phenomenon is illustrated by the fact that these systems in production use are estimated to contain up to 200 billion lines of source code worldwide. The modernization or replacement of systems is justified from core business, financial and information security perspectives. The migration or renewal of complex information systems is not always straightforward.

Keywords: legacy systems, refactoring, enterprise information systems, cyber security, operational security

SISÄLLYS

1 JOHDANTO

Tietojärjestelmiä on laajamittaisesti käytetty organisaatioissa niin liiketoiminnan kuin yhteiskunnalle välttämättömienkin prosessien hallinnassa jo niin kauan, että monet yhä päivittäisessä käytössä olevat tärkeät tietojärjestelmät ovat teknisesti vanhentuneita. Kuitenkin niiden arvellaan sisältävän maailmanlaajuisesti jopa 100 miljardia riviä lähdekoodia, usein kirjoitettuna vuosikymmeniä vanhoilla teknologiastandardeilla kuten päivittämättömillä COBOL:illa tai Fortranilla. Kyseisten käyttöjärjestelmäalustojen tuki, esimerkiksi kriittisiä haavoittuvuuksia korjaavien tietoturvapäivitysten tarjoaminen, on voinut päättyä jo vuosia tai jopa vuosikymmeniä sitten.

Kuvatun kaltaisia järjestelmiä kutsutaan myös nimellä legacyjärjestelmä. Legacyjärjestelmä on määritelmällisesti sovellus, joka käyttää vanhentunutta laitteistoa tai ohjelmistoja. Ne on usein kehitetty ennen modernien ohjelmisto- ja järjestelmäkehitysmetodien yleistymistä ja niiden ylläpitokustannukset ovat usein erittäin suuret (Ransom, Sommerville & Warren, 1998). Esimerkiksi Yhdysvaltain liittovaltion vuosittaiset tietojärjestelmäkustannukset ylittivät 90 miljardia dollaria vuonna 2018, josta summasta suuri osa kuluu redundanttien, jopa 20 vuotta vanhojen järjestelmien ylläpitoon (McKinnon, 2018) Aikana ennen internetin läpimurtoa, tietoturvakysymyksiä ei luultavasti ole useinkaan pohdittu järjestelmäkehityksen aikana (Cox, 2019).

Kuitenkin legacyjärjestelmät ovat edelleen usein vastuussa organisaation toiminnan kannalta kriittisistä toiminnoista, niin sanotusta ydinliiketoimintalogiikasta. Suurten monoliittisten järjestelmien lisäksi organisaation toimintoihin on voinut kerrostua runsaasti pienempiä legacyjärjestelmiä, jotka hoitavat esimerkiksi yksittäisiä tehtäviä, tai jopa yksittäisen tehtävän osia. Tällaiset klusterit muodostavat internetiin epäsuorastikin yhteydessä ollessaan merkittävän tietoturvariskin (Weber, 2013).

Lisäksi tietojärjestelmiin on usein investoitu huomattavia määriä rahaa, joka toimii kannustimena refaktoroida eli tehostaa niiden sisäistä toimintaa puuttumatta varsinaiseen liiketoimintalogiikkaan, järjestelmän uudella korvaamisen sijaan.

Tämä niin kutsuttu ”järjestelmälaho” on taloudelliselta, kilpailukyvylliseltä ja valtionhallinnon järjestelmien kohdalla peräti kansallisen turvallisuuden kannalta merkittävä ongelma. Aiheen tieteellinen tutkimus on kuitenkin viime vuosikymmenen aikana jäänyt verrattain vähälle, vaikka kyseessä on laajuudeltaan ja vaikutuksiltaan alati kasvava ilmiö.

Katsauksen aikana kävi selväksi, että useimmat ongelmaan esitetyt ratkaisut ovat hyvin kohdealuespesifisiä, vaikkakin geneerisiä ja yleispäteviä ratkaisuja on koetettu tutkimuskirjallisuudessakin kehittää varsin ahkerasti (esim. Wong & Li, 2004; Ransom & Warren, 2002). Tietojärjestelmät ovat monimutkaisia kokonaisuuksia, ja sisältävät usein arvaamattomia syy-seuraus-suhteita.

1.1 Motivaatio ja tutkimuskysymys

1.1.1 Tutkimuskysymykset

Tutkimuksessa selvitetään

1) *mitä riskejä ja miksi legacyjärjestelmistä aiheutuu organisaatioille,*
2) *millaisissa tilanteissa järjestelmät kannattaa modernisoida ja milloin korvata ne tyystin sekä,*
3) *yleisiä metodologioita, joiden avulla legacyjärjestelmien modernisointi tai migraatio kannattaa toteuttaa.*

Tutkimuksen voidaan olettaa kiinnostavan ainakin sellaisia liike-elämän toimijoita, joiden organisaatiot ovat päivittäisissä toiminnoissaan riippuvaisia legacyteknologioista ja joilla on ajankohtainen tarve järjestelmämodernisoinnille. Aiheesta huolta kantavat tahot voivat työn avulla tutustua myös modernisointiprojektin käytännön vaatimuksiin. Ihanteellisena tavoitteena olisi, että aiheen tutkimus lisääntyisi tulevaisuudessa vastaamaan aiheen merkittävyyttä liike-elämälle, valtionhallinnolle ja yhteiskunnalle kokonaisuudessaan.

Tutkielman alkuperäinen motivaatio syntyi mm. professori ja kirjailija Vernor Vingen esittämästä ajatuskokeesta, jossa kysyttiin, mikä on tietojärjestelmien käyttöiän ja niiden koodipohjan kasvavan kompleksisuuden kuviteltavissa oleva päätepiste (Rees, 2013).

1.2 Rakenne

Tutkimuksen tutkimusmenetelmänä toimii kuvailevan kirjallisuuskatsauksen alalaji, integroiva kirjallisuuskatsaus. Sitä käytetään, kun tutkittavaa ilmiötä halutaan tutkia laaja-alaisesti ja tuottaa siitä mahdollisesti myös uutta teoriaa.

Kirjallisuuskatsausten tyyppikentässä (Salminen, 2011) tämä katsaustyyppi sijoittuu kuvailevan ja systemaattisen kirjallisuuskatsauksen välimaastoon.

Tutkimusmenetelmä on valittu muun muassa siksi, että aiheesta tehdyn uuden tai uudehkon tutkimuksen määrä on rajallinen. Aihe on ollut laajemman tieteellisen kiinnostuksen kohteena esimerkiksi 1990-luvulla, jolloin niin kutsuttuun Y2K-ilmiöön valmistautuminen herätti paljon huolta ja keskustelua (Bisbal ym., 1997).

Lisäksi työn eräänä tutkimuskysymyksenä on löytää ja esitellä toimivia metodologioita, joten laaja-alainen kirjallisuuskatsaus tulee kyseeseen.

Tutkimuksen toisessa pääluvussa käsitellään legacyjärjestelmän määritelmää ja esitellään tärkeimmät yhä merkittävässä määrin käytössä olevat legacyteknologiat.

Kolmannessa pääluvussa esitellään konseptitasolla riskejä, joille legacyteknologioiden käyttö voi organisaation altistaa. Näitä ovat kustannukset, operatiiviset riskit niin ydinliiketoiminnan kuin tietoturvan kannalta sekä erikseen tietoturvaan kohdentuvat riskit.

Neljäs pääluku käsittelee tärkeimpiä löydettyjä ratkaisumalleja legacyjärjestelmien tuottamiin riskeihin niin teknisestä kuin organisatorisesta näkökulmasta.

Viides pääluku keskittyy tarkastelemaan tietojärjestelmämodernisoinnin metodologioita tarkemmin, sillä tämä näyttäytyy aiheen kentässä kenties moninaisimpana ja kiinnostavimpana osa-alueena.

Kuudennessa pääluvussa havainnoista pyritään muodostamaan synteesi ja erittelemään esille tulleita seikkoja kriittisesti.

1.3 Terminologia

Englannin kielen termille *legacy system* ei ole sovittu yhtä suomenkielistä vastinetta. Termistä on käytetty niin muotoa legacy-järjestelmä (Pulli, 2021, *i*), perinnejärjestelmä (Tieteen termipankki, 2023) sekä perinteinen järjestelmä (Euroopan komissio, 2010).

Tässä tutkimuksessa käytetään yhdyssanamuotoon kirjoitettua muotoa legacyjärjestelmä, johtuen termin yleisestä tunnettuudesta sekä vakiintumisesta suurtenkin organisaatioiden suomenkielisen viestinnän käyttöön.

2 LEGACYTEKNOLOGIAT

Tässä luvussa tutkitaan legacyjärjestelmän käsitteen moninaisuutta sekä määritellään tärkeimpiä tänä päivänä legacyksi katsottavia teknologioita, joista varhaisimpien historia ulottuu aina 1950-luvun alkuun asti.

Euroopan Unioni määrittelee nk. ”perinteisen järjestelmän” (Euroopan komissio, 2010) tarkoittavan

> --yleensä vanhahkoja järjestelmiä, jotka suorittavat edelleen olennaisia tehtäviä tai joilla säilytetään / joiden kautta käytetään olennaisia tietoja, mutta jotka käyttävät vanhaa tekniikkaa, jota on vaikea integroida uusiin järjestelmiin, ja tästä syystä järjestelmien uudelleen toteuttaminen on vaikeaa tai kallista. Perinteiseksi järjestelmäksi voidaan kuitenkin sen iästä riippumatta luokitella mikä tahansa tietojärjestelmä, jota ei ole suunniteltu uudelleenkäyttöä varten tai integroitavaksi muihin järjestelmiin, vaikka se olisikin toteutettu vasta äskettäin. (Euroopan komissio, 2010)

Informaatioteknologia-alan ammattilaisia haastatellessa legacyjärjestelmien määrittteleväksi piirteeksi on noussut niiden kronologinen ikä, jonka kriteerinä katsotaan yleensä olevan useita kymmeniä vuosia (Khadka ym., 2014). Toisaalta legacyjärjestelmä voi olla hyvinkin uusi, mutta todettu jo riittämättömäksi täyttämään sille osoitetut tehtävät (mm. Huijgens ym., 2016). Kuitenkin useimmat legacyjärjestelmän määritelmät sisältävät lisäkriteereitä, kuten liiketoimintakriittisyys (Khadka ym., 2014) sekä uusintamiseen kohdistuva huomattava tekninen vastarinta.

Yhä käytössä olevien legacytietojärjestelmien kehityshistoria voidaan laitteiston jakaa useisiin diskreetteihin kausiin (Weber, 2006). 1970- ja 80-luvuilla yritysjärjestelmien perusarkkitehtuurina toimi keskuskone (engl. *mainframe*), jossa tiedonkäsittely tapahtui keskitetysti, käyttäjien ohjatessa järjestelmää tiedonkäsittelykykyä sisältämättömien päätelaitteiden eli terminaalien avulla. 1990-luvun innovaatio olivat asiakas/palvelin -periaatteella toimivat järjestelmät.

Modernien operatiivisten liiketoimintaohjelmistojen kronologisena takarajana voidaan pitää 1950-lukua, jolloin COBOL- ja Fortran -kielten suunnittelu aloitettiin (Sammet, 1981; Backus, 1978). Tärkeimmät yhä käytössä olevat legacyteknologiat kuten C-kieli (Ritchie, 1996) oli määritelty 1970-luvun loppuun mennessä.

2.1 Tärkeimmät käytössä olevat legacyteknologiat

Jotkin legacyteknologiat ovat olleet käytössä lähes koko nykymuotoisten elektronisten tietokoneiden olemassaolon ajan. Teknisesti jo auttamattoman vanhentuneeksi katsottuja arkkitehtuureita ja teknologioita saattaa olla operatiivisessa

käytössä vuosikymmeniä myöhemmin, muun toimintaympäristön siirryttyä jo suurelta osin modernimpiin ratkaisuihin.

2.2 Ohjelmointikielet ja rajapinnat

2.2.1 FORTRAN

Ennen vuotta 1954 käytännössä kaikki tietokoneohjelmointi tapahtui suoraan joko binääristä tai symbolista konekieltä (engl. *assembly*) kirjoittamalla (Backus, 1978). Backuksen mukaan tämänkaltainen ohjelmointi on erittäin suurelta osin käsityötä, jota vaikeuttivat entisestään varhaisten tietokoneiden tekniset rajoitukset, kuten indeksirekistereiden ja liukulukulaskentakyvyn puute sekä vajavaiset suorittimien käskykannat. Esimerkiksi kyky ymmärtää Boolen operaattoreita kuten AND tai OR saattoi puuttua tyystin koko käskykannasta (Backus, 1978).
Suuri käsityön määrä johti tällä tavalla tuotettujen ohjelmien erittäin korkeisiin kehityskustannuksiin (Backus, 1978). Taloudelliset seikat olivatkin pääasiallinen motiivi Fortran-kielen (engl. lyh. *Formula Translator*) kehitystyön aloittamisessa aluksi IBM-yhtiön sisäisenä projektina vuonna 1953. Fortranin mukana keksittiin myös ensimmäinen kääntäjä, joka mahdollisti korkean tason ohjelmointikielten kehittämisen (Backus, 1978). Fortran on yhä laajalti yrityskäytössä ja yksi maailman tärkeimmistä ohjelmointikielistä (TIOBE, 2023).

2.2.2 COBOL

Siinä missä Fortran-kieli oli alkujaan IBM:n toteuttama yksityinen projekti sen omille keskustietokonemalleille (Backus, 1978), COBOL (eng. *Common Business Oriented Language*) alkoi yliopistomaailman, yritysten sekä Yhdysvaltain puolustushallinnon yhteisprojektina vuonna 1959 (Sammet, 1981). Projektille asetettuja vaatimuksia olivat esimerkiksi pohjautuminen luonnolliseen kieleen (englanti) ja helppokäyttöisyys tehokkuudenkin kustannuksella. Lisäksi Sammetin mukaan edellytettiin, että ”ongelmat tuolloin käytössä olleiden kääntäjien kanssa eivät saisi aiheuttaa vinoumia” kielen suunnittelussa. Toisaalta COBOL:in ennustettiin olevan käytössä hyvinkin pitkään erilaisissa käyttötapauksissa, toisaalta vuonna 1959 kokoontunut suunnittelutiimi itsekin piti sitä vain väliaikaisessa käytössä olevana ratkaisuna (Sammet, 1981).

2.2.3 C-kieli

Suuri osa nykyisestä IT-infrastruktuurista, käyttöjärjestelmien ytimistä jopa ohjelmistotasolle asti, on rakennettu C-kielen avulla (Interviewbits, 2022). C:n

ensimmäinen versio on kehitetty 1970-luvun alussa Bellin laboratorioissa Dennis Ritchien toimesta UNIX-käyttöjärjestelmää varten. Koska UNIX on toiminut joko suorana esi-isänä esimerkiksi Applen Macintoshille sekä eri BSD-käyttöjärjestelmille, tai GNU/Linuxin tapauksessa merkittävänä innoittajana ja teknisenä mallina, ja koska myös Microsoft Windows on alusta alkaen kehitetty C:llä, on se edelleen hyvin merkittävä ohjelmointikieli. C-kieltä on mahdollista kirjoittaa hyvin järjestelmäresurssitehokkaasti verrattuna moniin nykykieliin (joista esim. C++, C# ja Java toisaalta perustuvat siihen suoraan esitellen vain uusia ominaisuuksia) ja esimerkiksi mikrokontrollerit ja sitä kautta erilaiset esineiden internetiin kytketyt järjestelmät tuotetaan usein C:llä.

C-kielen synty tapahtui vuosien 1969 ja 1973 välillä, yhtäaikaisesti Unix-käyttöjärjestelmän kanssa; C-kieli oli tarkoitettu Unixin käyttöön (Ritchie, 1996). Ritchien mukaan kyseessä on yhä laajalti eri tietoteknisissä sovelluksissa käytössä oleva kieli (Ritchie, 1996). Viime vuosiin asti säilyneitä C:n tärkeitä käyttökohteita ovat esimerkiksi käyttöjärjestelmien ohjelmointi, sulautetut järjestelmät sekä käyttö välitason kielenä korkeamman tason ohjelmointikielten sekä symbolisen konekielen välillä (Interviewbits, 2022). C-kielen kehitys jatkuu yhä, ja viimeisin ISO-standardi (niin sanottu C17) on vuodelta 2018 (Dmitrović & Dmitrović, 2021).

2.3 Käyttöjärjestelmät

Ennen Microsoftin Windowsin saavuttamaa käytännön monopolia työpöytätietokoneiden käyttöjärjestelmämarkkinoilla 1990-luvulla (Gisser & Allen, 2001), laitteistotoimittajat varustivat tuotteensa usein omilla käyttöjärjestelmillään. Esimerkiksi Hewlett-Packardin työasemissa tarjottiin sen omaa kaupallista versiota UNIX-ytimestä, johon se itse lisäsi tarvittavia laajennuksia, kuten tuen esimerkiksi C- tai Fortran-kielille (Wang & Lindberg, 1984).

Nykyaikaiset työpöytäkäyttöjärjestelmät, kuten Windows sekä Linuxin työpöytädistribuutiot muodostavat merkittävän legacyvelan, sillä ne ovat teknisesti kahdessa suhteessa epäluotettavia ja turvatasoltaan heikkoja; niiden koodipohja on massiivinen ja virheiden rajaaminen on huonolla tasolla (Tanenbaum ym., 2006). Jopa Linux-ydin oli jo vuonna 2006 laajuudeltaan 2,5 miljoonaa C-koodiriviä, sisältäen laskennallisesti 15000 bugia (Tanenbaum, 2006), jonka jälkeenkin se on saanut merkittäviä lisäyksiä kuten KVM-virtualisointi-integraation (KernelNewbies, 2017).

2.4 Keskustietokoneet ja muu laitteisto

Yhdysvaltalainen IBM on historiallisesti ollut maailman huomattavin keskustietokoneiden toimittaja (Britannica, 2022). Keskuskoneet ovat massiivisen

numeerisen datan käsittelyyn tarkoitettuja tietokoneita, jotka soveltuvat hyvin esimerkiksi suuryritysten palkanlaskentaan ja muihin toimintoihin. Niitä ohjataan esimerkiksi laskentatehoa itsessään sisältämättömillä päätelaitteilla. Keskuskoneet olivat yrityksille hyvin suuri kustannuserä, mutta digitalisaation ensiaskeleena merkittävä sijoitus. Kun PC-tietokoneet tulivat markkinoille noin vuonna 1980, ne syrjäyttivät joitain keskuskoneiden toimintoja ja toivat samalla tietotekniikan kuluttajien sekä pienempien organisaatioiden ulottuviin.

3 LEGACYJÄRJESTELMIEN TUOTTAMAT RISKIT

3.1 Kustannukset

Tässä alaluvussa esitetään tuloksia legacyjärjestelmiin tukeutumisen hinnoista verrattuna uusien järjestelmien hankintaan. Päätökseen jatkaa legacyjärjestelmän käyttöä vaikuttaa joukko sekä rationaalisia että toisaalta joitain loogisesti virheellisiä päätelmiä.

Legacyjärjestelmien suurten kustannusten on todettu olevan useiden teknisten ja järjestelmäarkkitehtonisten tekijöiden, kuten huollettavuuden, hajotettavuuden, laadullisen huonontumisen sekä vanhentuneisuuden summa (Crotty & Horrocks, 2017).

Kaikkien organisaatioiden, mutta varsinkin voittoa tuottamaan pyrkivien liikeyritysten tietojärjestelmien käyttöä ohjaa jatkuva kustannustehokkuuspaine sekä pyrkimys parempaan tuottavuuteen (Fujitsu Consulting, 2007). Legacytietojärjestelmien osalta etenkin päivittäisen toiminnan mutta myös ylläpitotoimien kustannusten hallinta on avainasemassa (Fujitsu Consulting, 2007). Nämä kustannukset tapaavat syödä valtaosan organisaatioiden IT-osastojen budjetista. Ohjelmistotalojen, eli itse ohjelmistotuotteita tuottavien yritysten, kohdalla myös ohjelmakoodin ylläpitokustannukset voivat olla huomattavat, varsinkin jos alkuperäiset suunnittelijat ovat syystä tai toisesta poistuneet organisaation käytettävissä olosta (Fujitsu Consulting, 2007).

3.2 Operatiiviset riskit

Tässä alaluvussa tutkitaan päivittäiselle liiketoiminnalle koituvia riskejä.

Mikäli organisaation ydinliiketoiminta luottaa teknisesti vanhentuneisiin, epävakaisiin tietojärjestelmiin, antaudutaan huomattaville palvelukatkoksille ajan myötä yhä pahenevien teknisten ongelmien myötä. Tietojärjestelmiä intensiivisesti hyödyntävä suuri organisaatio on tällaisten katkosten myötä, niiden ollessa toistuvia ja/tai laajoja, jopa eksistentiaalisen kriisin tilanteessa.

De Lucia ym. (2001) katsovat, että legacyjärjestelmien rappeutuminen on kahden niin kutsutun lainalaisuuden puutteellisuuden funktio; ohjelman on läpikäytävä muutosta jatkuvasti tai sen reaalimaailman rajahyöty alenee progressiivisesti. Lisäksi niin sanotun kasvavan kompleksisuuden säännön mukaan ohjelmistojen sisäinen rakenne rappeutuu ilman säännöllisiä korjaustoimenpiteitä (De Lucia ym., 2001).

3.3 Kyberturvallisuusriskit

Tässä luvussa eritellään vanhentuneesta, päivittämättömästä teknologiasta johtuvia tietoturvahaavoittuvuuksia ja niiden mahdollisia seurauksia.

Tietoturvapäivitykset ovat pysyvä kilpajuoksu haavoittuvuuksien korjaajien ja niitä hyväksikäyttävien välillä. Mikäli ulkoinen toimittaja päättää lopettaa tietoturvapäivitysten tarjoamisen tuotteilleen, altistuu niitä toiminnoissaan käyttävä organisaatio huomattaviin, ajan myötä vain todennäköisempiin ja laajempiin kyberturvallisuusriskeihin. Mikäli tämänkaltainen järjestelmä on kytketty niin sanottuun julkiseen internetiin, se voidaan paikantaa ja sen jopa vuosia vanhoja haavoittuvuuksia on erittäin helppoa hyödyntää tietomurrossa.

Esimerkkinä Suomen julkisesta keskustelusta on runsaasti huomiota, poikkeuksellisen henkilökohtaisten ja arkaluontoisten potilastietojen vuotamiseen johtanut terapeuttipalveluyritys Vastaamon tietomurto (Marttila, 2021; Lindroos-Hovinheimo, 2020), joka pohjimmiltaan johtui yksittäisen, internetiin suoraan yhteydessä avoimen portin kautta olleen tiedostopalvelimen huomattavasti vanhentuneesta ohjelmistosta. Kyseessä oli Python 2-virtuaalikoneen versio vuodelta 2005, josta löytyi vuosikymmenen ajan tiedossa ollut haavoittuvuus, jonka ansiosta yksin toiminut hyökkääjä (Kerkelä, 2022) pääsi käsiksi koko tiedostopalvelimen sisältöön ja vuoti asiakkaiden potilastiedot internetiin. Seurauksena kohdeorganisaation koko liiketoiminta romahti ja se ajautui hyvin pian konkurssiin. Yrityksen vastuuhenkilöiden kuten toimitusjohtajan sekä tietosuojavastaavan toiminnasta käynnistettiin tutkinta (Marttila, 2021; Lindroos-Hovinheimo, 2020).

4 MODERNISOINTITARPEEN TUNNISTAMINEN

4.1 Tuen jatkamisen kannattavuus

Tässä alaluvussa tutkitaan, millaisissa tilanteissa legacyjärjestelmän käytön jatkaminen tiettyjen muutosten jälkeen on edelleen kannattavaa.

Joissain tapauksissa vanhan järjestelmän ylläpitäminen jatkossakin osoittautuu kannattavimmaksi (tai vähiten tappiolliseksi) vaihtoehdoksi, joskaan se ei ole pysyvä ratkaisu. Strategiana on jatkaa olemassa olevien prosessien tukemista ja odottaa, kunnes saataville tulee realistisia vaihtoehtoja, tai kunnes modernisointiratkaisujen kannattavuus paranee halutulle tasolle teknisen kehityksen ja markkinoille saapumisen myötä.

Legacyjärjestelmien kanssa organisaatioissa työskentelevät henkilöt pitävät usein operationaalista vakautta legacyjärjestelmien käytön jatkamista puoltavana tekijänä (Khadka ym., 2014).

4.2 Uusien tietojärjestelmäprojektien riskit ja kustannukset

Tietojärjestelmäprojekti, jossa rakennetaan kokonaan uusi järjestelmä haluttujen vaatimusten mukaan alusta asti, useimmiten yhden tai useamman toimittajan yhteistyönä, on historiallisesti osoittautunut olevan äärimmäisen riskialtis toteuttaa. Jopa seitsemän kahdeksasta kehitysprojektista epäonnistuu täyttämään jonkin avainkriteereistään (McManus & Wood-Harper, 2007). Useimmiten projektin budjetti ylittyy tai sen ajallinen horisontti venyy huomattavasti, mikäli ne ylipäätään valmistuvat (Linberg, 1999).

Toisinaan tietojärjestelmän kehitysprojekti epäonnistuu jopa teknisellä tasolla, eikä kykene suorittamaan tehtäviä tai tukemaan niitä ydintoimintoja, joita varten projektiin on lähdetty (McManus & Wood-Harper, 2007), tai voi olla loppukäyttäjille jopa vaarallinen tehtäväkriittisten kohteiden kuten sairaaloiden tapauksessa (Linberg, 1999). Siksi monelle jo digitalisoituneelle, nyt legacyongelman kanssa kamppailevalle organisaatiolle täysin uuden järjestelmän hankinta ilman jatkuvuutta nähdään viimeisenä, lähes epätoivoisena vaihtoehtona.

5 MODERNISOINNIN METODOLOGIOITA

Legacyjärjestelmän modernisointiin liittyy useita, toisinaan kiistanalaisia ja vakiintumattomia käsitteitä (Laguna & Crespo, 2013) ja se tuo mukanaan monenlaisia haasteita johtuen tietojärjestelmien yhtäaikaisesta kompleksisuudesta ja hauraudesta (Seacord et al, 2001, *vii*).

Refaktorointi tarkoittaa [olio-ohjelmoidun] ohjelmiston muuttamista siten, että koodin toteuttama ulkoinen käytös ei muutu, vaikka koodi sisäisesti muuttuukin (Mens & Tourwé, 2004). Ohjelmiston uudelleensuunnittelu puolestaan kattaa myös ulospäin näkyvät muutokset.

Järjestelmän migraatiolla tarkoitetaan järjestelmän uudelleensijoittamista esimerkiksi uudelle ohjelmointikielelle, alustalle tai tietokantaratkaisuun (Laguna & Crespo, 2012).

Legacyjärjestelmien evoluutiolla tarkoitetaan uuden järjestelmän luomista käyttäen legacyjärjestelmää kattavasti suunnittelun pohjana, mutta hyödyntäen uudempia ratkaisuja suunnittelussa (Huijgens ym, 2016), esimerkiksi uudenaikaisempia standardeja.

Modernisaatio nähdään eduistaan huolimatta silti suurena haasteena, jossa on useita mahdollisia ja todennäköisesti limittyviä ongelmakohtia (Khadka ym., 2014). Näitä ovat esimerkiksi modernisaatiotyön aikapaineet, datan migraation haasteet, monoliittisesta tai muutoin huonosti suunnitellusta arkkitehtuurista koituvat ongelmat modernisoinnin osastoinnissa, puutteellinen dokumentaatio sekä sen johdannaisena vaikeus tunnistaa ja erotella järjestelmän ydinliiketoimintalogiikka (Khadka ym., 2014).

5.1 Liiketoimintalogiikan migraatio

Legacyjärjestelmän toteuttama liiketoimintalogiikka on mahdollista siirtää sellaisenaan tai lähes sellaisenaan uusille ohjelmisto- tai laitteistoalustoille. Tämä on perusteltua esimerkiksi tilanteessa, jossa laitteiston fyysinen käyttöikä on lopussa mutta ohjelmisto sinänsä on vakaa ja kykenee täyttämään sille asetetut tehtävät riittävällä tasolla, eikä ohjelmiston uudelleentoteutusta nähdä kustannustehokkaana.

5.1.1 Yleiset migraatiostrategiat

Aikana ennen nykyisten pilvialustojen yleistymistä katsottiin laajalti, että legacyjärjestelmän migraatio uuteen fyysiseen ympäristöön tulisi suorittaa mahdollisuuksien mukaan inkrementaalisesti (Brodie & Stonebaker, 1996; Seacord et al, 2001). Esimerkkinä Brodie ja Stonebaker (1996) käyttävät malliesimerkkinä tilannetta, jossa useita sovelluksia sisältävän järjestelmän migraatio toteutetaan sovellus kerrallaan, jonka jälkeen niiden tarvitsemat rajapinnat siirretään niin ikään

osina. Ajallisesti inkrementaalisen modernisaatioprojektin voidaan odottaa kestävän kauan, suurten järjestelmien kohdalla jopa useita vuosia (Seacord et al, 2001, s. 1)
Tietokantamigraation kohdalla Brodie ja Stonebaker (1996) pitävät yhdyskäytäväteknologiaa avainasemassa onnistumisen varmistamiseksi. Oraclen mukaan yhdyskäytävien tehtävä on yhdenmukaistaa heterogeenisessä muodossa saatua dataa uuden tietokannanhallintajärjestelmän käyttöön (Oracle, 2021).

5.1.2 Rajapintojen käyttö

Rajapinnat ovat ohjelmistoteknologioita, joiden avulla kaksi teknisesti erilaista, erilaisia sisäisiä kommunikaatiostandardeja käyttävää tietojärjestelmää, ohjelmistoa tai laitetta voi vaihtaa tietoa keskenään tarvitsematta tuntea toisen sisäistä toimintalogiikkaa (Red Hat, 2022).

REST-rajapinta-arkkitehtuurimalli kehitettiin alkujaan vuonna 2000 ratkaisuna internetiä uhanneeseen skaalautuvuuskriisiin, jossa datan määrä ylittäisi käytettävissä olevat keinot käsitellä ja jäsennellä sitä järkevästi. Se tarjoaa universaalin keinon vaihtaa dataa hyvinkin erilaisten tietojärjestelmien ja ohjelmistojen kesken (Masse, 2011).

Rajapinnat, etenkin REST-arkkitehtuurimallin mukaiset, ovat merkittävä keino pidentää legacyjärjestelmän hyödyllistä käyttöikää (Rodríguez-Echeverría ym., 2014). Niistä on tullut nykyisen integraatioliiketoiminnan merkittävä paradigma. Vanhoja järjestelmiä ei ajeta alas tai korvata lyhyellä aikavälillä, mutta niiden tarjoama liiketoimintalogiikka ja muut ominaisuudet saadaan tuotua uuden järjestelmän käytettäviin.

Käytännössä rajapintaratkaisu toteutetaan takaisinmallintamalla (engl. reverse engineering) ja uudelleenrakentamalla kohdejärjestelmää siinä määrin, että se kykenee käsittelemään esimerkiksi REST-muotoisia kutsuja ja vastavuoroisesti antamaan sisältämäänsä dataa esimerkiksi uudelle tietojärjestelmälle (Rodríguez-Echeverría ym., 2014) mielekkäässä muodossa.

Rajapintaintegraation haittapuolena on vanhan järjestelmän jääminen yhä toimintaan, joskaan ei luultavasti aivan aiempaa vastaavan tasoiseen käyttöön. Tietoturva-, ohjelmisto- ja laitteistotekniset ongelmat pysyvät niin ikään.

Rajapinnat eivät ole myöskään turvassa tietoturvaongelmilta, vaan API-turvallisuus on kiinteä osa niiden suunnittelua (Siriwardena, 2014, s. 11).

5.1.3 Inkrementaalinen migraatio

Mikäli modernisoitava järjestelmä on jo valmiiksi hajautettu useampiin alijärjestelmiin, voidaan niiden migraatio tehdä osissa (Brodie & Stonebaker, 1996). Tällä voidaan esimerkiksi turvata järjestelmän toiminnan jatkuvuus ja minimoida migraatiosta aiheutuvia operatiivisia riskejä.

5.2 Ohjelmiston uudelleensuunnittelu ja refaktorointi

Lagunan ja Crespon (2012) mukaan Fowler (2002) määrittelee refaktoroinnilla tarkoitettavan ”ohjelman tai sovelluksen sisäisen rakenteen parantamista ja kehittämistä muuttamatta sen ulkoista käyttäytymistä”. Käsite syntyi varhaisen oliokielen Smalltalkin piirissä, mutta levisi nopeasti yleiseen käyttöön olio-ohjelmointiparadigman sisällä. Refaktoroinnista on olemassa runsaasti erilaisia koulukuntia käytännön toteutustapojen suhteen (Fowler 2002; Liu et al., 2006).

Sneedin ym. (2019) mukaan olemassaolevan legacykoodin reimplementaatio tulee kyseeseen, kun kohdejärjestelmän toiminta on erittäin kietoutunutta johonkin kriittiseen vanhaan teknologiaan kuten tietokannanhallintajärjestelmään, tai kun koodipohja on syystä tai toisesta laadultaan riittämätön muunnettavaksi uusille alustoille.

Etenkin ennen UML-mallinnuskielen laajamittaista käyttöönottoa kehitetyt laajat tietojärjestelmät voivat hyötyä systemaattisesta refaktorointityöstä (Griffith, ym., 2011). Toinen kohde refaktoroinnille ovat web-sovellukset, jotka on kehitetty ennen REST-rajapintamallin keksimistä (Rodríguez-Echeverría ym., 2014).

Ohjelmiston uudelleensuunnittelu on refaktoroinnin kiistelty lähikäsite, joka on ymmärrettävissä järjestelmän laaja-alaisempana uudelleensuunnitteluna, sisältäen jopa sen laitteistokomponentteihin tehtävät muutokset (Laguna & Crespo, 2012).

5.2.1 Geneeriset uudelleensuunnittelumetodit

Vaikka legacyjärjestelmät ovat usein keskenään erittäin heterogeenisiä, on niiden uudelleensuunnitteluun laadittu runsaasti niin sanottuja geneerisiä menetelmiä. Wongin ja Li’n (2003) yleinen käytännön muutos ohjelmakoodissa on muuttaa vanhemmilla paradigmoilla ja kielillä laadittu koodi vastaamaan oliosuuntautunutta paradigmaa. Oliosuuntautuneesti toteutettua koodia pidetään yleisesti helpommin uudelleenkäytettävänä sekä luotettavampana kuin varhaisempia paradigmoja (Wong & Li, 2003). Esimerkiksi C-kielellä proseduraalisesti kirjoitetut ohjelmat on mahdollista uusintaa hyvällä menestyksellä oliopohjaiseksi tiettyjä geneerisiä vaiheita noudattaen (Wong & Li, 2003).

5.2.2 Ohjelmistojen uudelleenkäyttö

Ohjelmistojen uudelleenkäyttö tarkoittaa organisaation olemassaolevan tuotteen, artefaktien tai tiedon muuntamista uuteen käyttöön. Uudelleenkäytettävän elementin laajuus ja laatu voi olla koodinpätkä tai metodi, vaatimus, suunnitteluspesifikaatio tai vaikkapa kohdealueen tuntemus organisaatiossa (Karlsson, 1995).

Ohjelmistojen uudelleenkäyttö tapahtuu kuitenkin usein spontaanisti, eikä sen kaikkia hyötyjä esimerkiksi tehtäväkriittisten ohjelmistojen kuten avaruushallintojen käytössä olevien luotettavuutta parantavaa uudelleenkäyttöä ole osattu hyödyntää suunnitelmallisesti (Orrego & Mundy, 2007).

5.3 Laitteiston uusintaminen

Komponenttien ikääntyminen on legacyjärjestelmien käyttöön väistämättä kuuluva tosiasia, vaikka esimerkiksi sotilaskäytössä olevat tieto- tai muut elektroniset järjestelmät pyritäänkin tietoisesti suunnittelemaan kuluttaja- tai yrityskäytössä olevia vastaavia pitkäikäisemmiksi (Zhang ym., 2018; Gee, 2009). Monenlaisissa operaatiokriittisissä järjestelmissä esimerkiksi pilvimigraatio ei syystä tai toisesta ole modernisaatiotoimissa huomioon otettava vaihtoehto. Esimerkiksi lainsäädännölliset vaatimukset tiedon fyysisen sijainnin suhteen tai tekninen vaatimus internet-riippumattomuudesta voivat estää näiden käytön. Etenkin hyvin ikääntyneen järjestelmän kohdalla oikeanlaisten, teknisesti jo sinänsä vanhanaikaisten komponenttien saatavuus voi olla huomattavan haastavaa (Zhang, ym., 2018).

Eräänä ratkaisuna on Zhangin ym. (2018) mukaan ehdotettu ohjelmoitavia FPGA-porttimatriiseja, tai varta vasten tuotettuja ASIC-piirejä (engl. application specific integrated circuit). Näiden hankintaan markkinatoimijoiden kautta voi kuitenkin sisältyä erittäin vakavia turvallisuusriskejä, kuten piilotettujen haittaohjelmien pääsy kohdejärjestelmään FPGA:n laiteohjelmistojen koodin kautta (Zhang ym., 2018).

5.4 Pilvimigraatio

Pilvipalvelut tarjoavat asiakkailleen mahdollisuuden tallettaa käyttämänsä tiedot palveluntarjoajan internet-palvelimille itse hallinnoitujen tallennusvälineiden sijaan (Rashid & Chaturvedi, 2019).
Pilvipalveluista on tullut useiden tekijöiden yhteisvaikutuksena merkittävästi alhaisempien transaktiokustannusten (Rashid & Chaturvedi, 2019) vuoksi keskeinen osa organisaatioiden IT-infrastruktuuria. Pilviratkaisut ovat suurelta osin korvanneet vanhat, omissa tiloissa ja omilla serverikoneillaan sijaitsevat ”on-

premise”-ratkaisut. IaaS- eli infrastruktuuri palveluna -tyyppiset (engl. lyh. *Infrastructure as a Service*) pilvipalvelut tarjoavat reaaliaikaista skaalautuvuutta ja siten mahdollisuuden välttää suuret pääomainvestoinnit (Rashid & Chaturvedi, 2019). Lisäksi monet turvallisuuskysymykset voidaan ulkoistaa näin palveluntarjoajalle (Rashid & Chaturvedi, 2019).

Legacysovellukset on useimmiten kehitetty aikana ennen pilvipalveluiden läpilyöntiä ja ne on siten kehitetty metodeilla, jotka eivät usein ole yhteensopivia pilviympäristöjen olosuhteiden kanssa (Gholami et al, 2016). Tämä on kuitenkin yleensä ratkaistavissa suhteellisen helposti esimerkiksi rajapintoja hyödyntämällä (Rodríguez-Echeverría ym., 2014).

6 YHTEENVETO JA POHDINTA

Suurimmassa osassa legacyjärjestelmäskenaarioita liiketoimintalogiikan ja tietovarastojen pilvimigraatio lienee esitetyistä ratkaisuista mielekkäin. Suurimmassa osassa tapauksia koodipohjan automaattinen migraatio uuteen teknologiaan voi toimia ilman suuria liiketoimintalogiikan menetyksiä. Parhaimmillaan pilvimigraatio voi ollakin vaihtoehdoista turvallisin, edullisin ja nopein, säästäen etenkin pieniltä organisaatioilta merkittävästi työtunteja.

Yhtä geneeristä tapaa ei kuitenkaan ole saatu kehitettyä, yrityksistä huolimatta. Tämä on odotettavissa oleva lopputulos, sillä tietojärjestelmissä on aivan liian suuri määrä huomioonotettavia muuttujia, jotta geneeriset metodit toimisivat edes jossain määrin luotettavasti, saati aukottomasti.

Joka tapauksessa ongelman skaala on valtava. Varsinkin pk-sektorin yrityksillä ja muilla pienillä organisaatioilla ei välttämättä ole kapasiteettia tai riittävää kannustinta lähteä työläisiin ja mahdollisesti kalliisiin järjestelmämuutosprojekteihin.

Samaan aikaan ohjelmisto- ja tietojärjestelmäprojektien kustannukset ja epäonnistumisen riski ovat niin suuria, että teknisesti vakaata, internetistä tietoturvallisesti eristettyä legacyjärjestelmää ei itsearvoisesti ole syytä lähteä korvaamaan uudella järjestelmällä. Tämä korostuu, mikäli vanha järjestelmä kykenee selviämään menestyksekkäästi liiketoiminnan sille asettamista tehtävistä. Usein merkittävin haaste näiden järjestelmien ylläpidossa löytyykin henkilöresurssien puolelta, mikäli legacyteknologioihin perehtynyttä työvoimaa edes rutiiniluontoisiin ylläpitotehtäviin ei yksinkertaisesti löydy.

On silti huomioitava, että minkä tahansa järjestelmäriskin, oli kyseessä joko operatiivinen, kustannus- tai tietoturvariski, kasvaessa ohittamattomaksi on migraatio- tai modernisaatioprosessi syytä käynnistää välittömästi kustannuksiin katsomatta. Riskien realisoituessa syntyneen kriisin seuraukset voivat olla organisaatiolle ja sen toiminnan edellytyksille katastrofaaliset.

Legacyjärjestelmien tarkastelun ei tarvitse kuitenkaan olla puhdasta taaksepäin katsomista. Ohjelmistoja tuottavien tahojen on syytä tiedostaa, että ”mutkien oikominen” esimerkiksi järjestelmäkehityksen arkkitehtuuri- ja toteutusvaiheissa tuottaa suurella todennäköisyydellä ohjelmistoja, jotka voidaan nähdä legacykoodina jo tuotantoon asetettaessa (Feathers, 2002).

LÄHTEET

Backus, J. (1978). The history of Fortran I, II, and III. *ACM Sigplan Notices, 13(8), 165-180.*

Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., & O'Sullivan, D. (1997). A survey of research into legacy system migration. *Technique report.*

Brodie, M. L., Stonebracker, M. (1995). Migrating legacy systems: gateways, interfaces & the incremental approach. San Francisco. Morgan Kaufmann Publishers, Inc.

Cox, S. & Levinson, H. (2019). Cybersecurity Engineering for Legacy Systems: 6 Recommendations [blogikirjoitus]. Haettu osoitteesta https://insights.sei.cmu.edu/sei_blog/2019/08/cybersecurity-engineering-for-legacy-systems-6-recommendations.html

Crotty, J., & Horrocks, I. (2017). Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company. *Applied computing and informatics, 13(2), 175-183.*

De Lucia, A., Fasolino, A. R., & Pompelle, E. (2001, November). A decisional framework for legacy system management. *In Proceedings IEEE International Conference on Software Maintenance. ICSM 2001* (pp. 642-651). IEEE.

Dmitrović, S., & Dmitrović, S. (2021). The C17 Standard. Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language, 273-273.

Euroopan komissio (2010). Commission Communication 'Towards interoperability for European public services' COM(2010) 744 final. Euroopan komissio: Bryssel. https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:52010DC0744&from=EN

Fujitsu Consulting (2007). Modernize Your Legacy Systems AND Cut Costs?. Haettu osoitteesta https://www.fujitsu.com/downloads/SVC/fc/wp/lm-status.pdf. Haettu 22.3.2023.

Feathers, M. (2002). *Working Effectively with Legacy Code.* Upper Saddle River. Prentice Hall.

Gee, W. A. (2009, March). Systems Engineering requirements for legacy DoD hardware upgrade and sustainment requirements definition, analysis, and validation. *In 2009 3rd Annual IEEE Systems Conference (pp. 431-436). IEEE.*

Gholami, M. F., Daneshgar, F., Low, G., & Beydoun, G. (2016). Cloud migration process—A survey, evaluation framework, and open challenges. Journal of Systems and Software, 120, 31-69.

Gisser, M., & Allen, M. S. (2001). One monopoly is better than two: Antitrust policy and microsoft. *Review of Industrial Organization, 19(2), 211-225.7*

Griffith, I., Wahl, S., & Izurieta, C. (2011, November). Evolution of legacy system comprehensibility through automated refactoring. *In Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering (pp. 35-42).*

Huijgens, H., Van Deursen, A., & Van Solingen, R. (2016). *Success factors in managing legacy system evolution: a case study. In Proceedings of the International Conference on Software and Systems Process (pp. 96-105).*

Interviewbits. (31.1.2022). *Most useful applications of C programming language.* https://www.interviewbit.com/blog/applications-of-c-programming-language/. Haettu 28.3.2023.

Karlsson, E. (1995). Software reuse: a holistic approach. Chichester. John Wiley & Sons.

Kerkelä, L. (2022). Tällainen on epäilty hakkeri: Tuomittu lukuisista tieto-murroista ja perättömästä pommi-uhkauksesta. Helsingin Sanomat. https://www.hs.fi/kotimaa/art-2000009164561.html. Haettu 22.02.2023.

KernelNewbies (2017). Linux_2_6_20 [muutosluettelo]. https://kernelnewbies.org/Linux_2_6_20#head-bca4fe7ffe454321118a470387c2be543ee51754 . Haettu 4.6.2023.

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., & Hage, J. (2014, May). How do professionals perceive legacy systems and software modernization?. In Proceedings of the 36th International Conference on Software Engineering (pp. 36-47).

Laguna, M. A., & Crespo, Y. (2013). A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming, 78(8), 1010-1034.*

Linberg, K. R. (1999). Software developer perceptions about software project failure: a case study. *Journal of Systems and Software, Volume 49, Issues 2–3, s. 177-192*

Marttila, H. (2021). Major security breach in Finnish psychotherapy center: legal issues in liability and personal ID factors. *European Journal of Privacy Law & Technologies.*

Masse, M. (2011). REST API design rulebook: designing consistent RESTful web service interfaces. " O'Reilly Media, Inc.".

McKinnon, J. D. (2018, May 15). Trump signs order to overhaul government's computer systems; money is wasted now on maintaining costly and duplicative legacy systems often badly out of date, critics say. *Wall Street Journal (Online)* Haettu osoitteesta https://www-proquest-com.ezproxy.jyu.fi/newspapers/trump-signs-order-overhaul-governments-computer/docview/2038941213/se-2?accountid=11774

McManus, J., & Wood-Harper, T. (2007). Understanding the sources of information systems project failure. Haettu osoitteesta https://www.researchgate.net/profile/John-Mcmanus-11/publication/329539985_A_study_in_project_failure/links/5c1138aea6fdcc494fef184b/A-study-in-project-failure.pdf

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. IEEE Transactions on software engineering, 30(2), 126-139.

Oracle. (2021). *Overview of Oracle Database Gateways.* https://docs.oracle.com/en/database/oracle/oracle-database/21/gmswn/database-gateway-sqlserver-overview.html

Orrego, A. S., & Mundy, G. E. (2007). A study of software reuse in NASA legacy systems. *Innovations in Systems and Software Engineering*, 3(3), 167-180.

Pulli, N. (2021). *Parhaat käytännöt legacy-järjestelmän uudistamisessa* [pro gradu -tutkielma, Tampereen yliopisto]. Trepo. https://urn.fi/URN:NBN:fi:tuni-202106216010

Ransom, J., Sommerville, I. & Warren, I. (1998). A Method for Assessing Legacy Systems for Evolution. Haettu osoitteesta doi=10.1.1.128.9889

Ransom, J., Warren, I. (2002). Renaissance: A Method to Support Software System Evolution. Haettu osoitteesta http://www.cse.dmu.ac.uk/COMPSAC/wimpe/secretpath/authors/author.93/paper/paper.93.pdf

Rashid, A., & Chaturvedi, A. (2019). Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering*, *7*(2), 421-426.

Red Hat, Inc. (2.6.2022). *What is an API?*. https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces. Haettu 15.3.2023.

Rees, G. (12.6.2013). Software archaeology and technical debt. *Garethrees.org (blogi)*. https://garethrees.org/2013/06/12/archaeology/

Ritchie, D. M. (1996). The development of the C programming language. In History of Programming languages---II (pp. 671-698).

Rodríguez-Echeverría, R., Maclas, F., Pavón, V. M., Conejero, J. M., & Sánchez-Figueroa, F. (2014). Generating a rest service layer from a legacy system. In *Information System Development: Improving Enterprise Communication* (pp. 433-444). Springer International Publishing.

Lindroos-Hovinheimo, S. (2020). Serious Cyberattack Raises Questions About GDPR Application in Finland. Verfassungsblog.

Salminen, A. (2011). *Mikä kirjallisuuskatsaus? Johdatus kirjallisuuskatsauksen tyyppeihin ja hallintotieteellisiin sovelluksiin. Vaasan Yliopiston julkaisuja 62.* Vaasa: Vaasan yliopisto.

Sammet, J. (1981) The early history of COBOL. *History of programming languages,* 199-243. Association for Computing Machinery, Inc.

Seacord, R. C., Comella-Dorda, S., Lewis, G., Place, P., & Plakosh, D. (2001). Legacy system modernization strategies. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Siriwardena, P. (2014). Advanced API Security. Apress: New York, NY, USA.

Sneed, H., & Verhoef, C. (2019). Re-implementing a legacy system. Journal of Systems and Software, 155, 162-184.

Tanenbaum, A. S., Herder, J. N., & Bos, H. (2006). Can we make operating systems reliable and secure?. Computer, 39(5), 44-51.

Tieteen termipankki 30.3.2023: Tietojenkäsittelytiede:perinnejärjestelmä. (Tarkka osoite: https://tieteentermipankki.fi/wiki/Tietojenkäsittelytiede:perinnejärjestelmä.)

TIOBE (2023). TIOBE Index fpr March 2023. Haettu osoitteesta https://www.tiobe.com/tiobe-index/. Haettu 22.3.2023.

Wang, S. W. Y., Lindberg, J. B. (1984). HP-UX: Implementation of UNIX on the HP 9000 Series 500 Computer Systems. Hewlett-Packard Journal Vol. 35 No. 3 (1984-03), s. 7-15

Weber, C. (2013). Software Security in Legacy Systems. Haettu osoitteesta https://web.archive.org/web/20161210200830/https://www.us-cert.gov/bsi/articles/best-practices/legacy-systems/assessing-security-risk-in-legacy-systems

Wong, W. E., & Li, J. (2004). Redesigning legacy systems into the object-oriented paradigm. International Journal of Software Engineering and Knowledge Engineering, 14(03), 255-276.