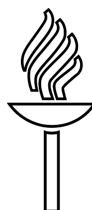
Tuomas Lehto

TURVALLISUUSKRIITTISTEN TIETOJÄRJESTELMIEN EPÄONNISTUMISET JA NIIDEN YHTEYS KETTERÄÄN OHJELMISTOKEHITYKSEEN



JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2020

TIIVISTELMÄ

Lehto, Tuomas
Turvallisuuskriittisten tietojärjestelmien epäonnistumiset ja niiden yhteys ketterään ohjelmistokehitykseen
Jyväskylä: Jyväskylän yliopisto, 2020, 57 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Halttunen, Veikko

Tietojärjestelmät muodostavat nykyaikaisen yhteiskunnan selkärangan. Organisaatioiden toiminta ei olisi nykyisellä laajuudella ja tehokkuudella mahdollista ilman, että tietojärjestelmien avulla ohjattaisiin niiden toimintaa. Hallinnan ja tehokkuuden kääntöpuolena on riippuvuus tietojärjestelmistä myös sellaisissa ympäristöissä, joissa tietojärjestelmien epäonnistuminen aiheuttaa vakavia seurauksia. Tällaisiin turvallisuuskriittisiin ympäristöihin suunnitellut tietojärjestelmät kehitetään tiukan sääntelyn puitteissa, mutta silti ne eivät ole immuuneja epäonnistumisille.

Järjestelmän tilaaja haluaa järjestelmän olevan paitsi suoristusvarma, myös mahdollisimman nopeasti käyttökelpoinen. Erityisesti nopeuden vaatimukseen vastattiin 2000-luvun taitteessa ketterän ohjelmistokehityksen manifestilla, josta sittemmin on muokattu malleja soveltuvaksi myös suuriin projekteihin ja turvallisuuskriittisille toimialoille.

Tämän tutkimuksen tarkoitus on tutkia, löytyykö turvallisuuskriittisten tietojärjestelmien epäonnistumisten ja ketterän ohjelmistokehityksen suuntauksen välille yhteyttä. Tutkimus jakautuu kahteen osioon, joista ensimmäinen on kirjallisuuskatsaus. Siinä tarkastellaan ohjelmistokehityksen suuntauksia erityisesti verraten ketterää ohjelmistokehitystä ja perinteiseksi miellettyä suunnittelupainotteista suuntausta. Kirjallisuus sidotaan tutkimuksen aihepiiriin tarkastelemalla ketterän ohjelmistokehityksen turvallisuuskriittistä kontekstia.

Tutkimuksen toinen osa on empiirinen tutkimus, jossa esitellään teoriaohjaavan sisällönanalyysin tulokset. Analyysin pohjana toimivat Helsingin Sanomien ja YLE:n uutisartikkelit. Tutkimuksella pyritään luomaan ymmärrystä ilmiöstä ja tarkastelemaan sitä, miten epäonnistuneet tietojärjestelmät ja ketterä ohjelmistokehitys mahdollisesti linkittyvät toisiinsa.

Tutkimuksen perusteella tietojärjestelmiä otetaan käyttöön keskeneräisenä myös turvallisuuskriittisissä tietojärjestelmissä. Tiukoista prosesseista lipsutaan, jotta järjestelmiä saadaan tuotantoon nopeammin ja kustannustehokkaammin. Järjestelmän toimittajan ja hankkijan välillä vaikuttaa usein olevan puutteellinen ymmärrys järjestelmän vaatimuksista, mikä johtaa suunnitteluvirheisiin. Myös testaukseen ei käytetä riittävästi aikaa.

Asiasanat: ketterä ohjelmistokehitys, tietojärjestelmien epäonnistuminen, turvallisuuskriittisyys

ABSTRACT

Lehto, Tuomas
The failures of safety critical information systems and their connection to agile software development
Jyväskylä: University of Jyväskylä, 2020, 57 p.
Information Systems, Master's Thesis
Supervisor: Halttunen, Veikko

Information systems form the backbone of modern society. Organizations could not operate as efficiently without information systems guiding their operations. Downside of efficiency is that organizations are dependent of information systems to large extent. Information systems that are designed for safety critical environments are under strict regulations but are nonetheless vulnerable and sometimes unsuccessful.

Customer of a given system requires the system to be stable but also to be quickly in use. Agile software development was the answer to this demand of rapid value production. It was originally intended for small it projects but at present there are numerous adaptations of agile methods that are designed for large scale projects as well as safety critical context.

The purpose of this research is to explore if there is a connection to be found between failing safety critical information systems and agile paradigm. The research consists of two parts. Firstly, literature review is made to examine agile software development and its' predecessor, plan-driven development. The literature is bound to the topic of this research by also examining agile in safety critical context.

The second part of this research is empirical. Theory guided content analysis is used to form results for this research. The analysis is based on news articles that are collected from Helsingin Sanomat and YLE. The aim of this research is to create understanding of the subject and find if there is a connection between failing safety critical information systems and agile software development.

This research suggests that information systems are often implemented even if they are not ready. Processes are ignored in search of quick releases and cost savings. The vendor of an information system and the customer sometimes fail to form a unified vision of system requirements which leads to design errors. Systems are also tested inadequately.

Keywords: agile software development, information system failure, safety criticality

KUVIOT

KUVIO 1 Vesiputousmalli……………………………………………………………...11
KUVIO 2 Ylläpidon luokittelu…………………………………………………………15
KUVIO 3 Ketterän ohjelmistokehityksen arvot 16
KUVIO 4 Ketterän ohjelmistokehityksen ongelma-alueet ja niiden vaikutussuhteet turvallisuuskriittisessä kontekstissa………………………………………...25

TAULUKOT

TAULUKKO 1 Ketterän ohjelmistokehityksen periaatteet………………………16
TAULUKKO 2 Ketterien ja suunnittelupainotteisten menetelmien erot 18
TAULUKKO 3 Päätöksenteon ominaispiirteet ketterässä kehityksessä 22
TAULUKKO 4 Sisällönanalyysin prosessi…………………………………………32
TAULUKKO 5 Suorat sanalliset viittaukset ohjelmistokehityksen tehtäviin tapauksittain……………………………………………………………………….35

SISÄLLYS

TIIVISTELMÄ 2

ABSTRACT 3

KUVIOT 4

TAULUKOT 4

SISÄLLYS 5

1 JOHDANTO 7

2 OHJELMISTOKEHITYS 10
- 2.1 Ohjelmistokehityksen perustehtävät 10
  - 2.1.1 Vaatimusmäärittely 11
  - 2.1.2 (Arkkitehtuuri)suunnittelu 12
  - 2.1.3 Toteutus 13
  - 2.1.4 Testaus 14
  - 2.1.5 Ylläpito 14
- 2.2 Ketterä ohjelmistokehitys 16
- 2.3 Ketterän ohjelmistokehityksen haasteet 21
- 2.4 Ketterän ohjelmistokehityksen haasteet turvallisuuskriittisissä tietojärjestelmissä 24

3 TUTKIMUKSEN TOTEUTUS 29
- 3.1 Tutkimusote ja tutkimusmenetelmä 29
- 3.2 Aineiston rajaus ja analysointi 30

4 TUTKIMUSTULOKSET 33
- 4.1 Tapausten esittely 33
- 4.2 Tietojärjestelmien havaitut ongelmat 34
- 4.3 Ongelmien linkittyminen ketterään ohjelmistokehitykseen turvallisuuskriittisessä kontekstissa 37

5 POHDINTA 40
- 5.1 Tulosten johtopäätökset 40
- 5.2 Tutkimuksen luotettavuus 43

6 YHTEENVETO 45

LÄHTEET 47

LIITE 1 UUTISARTIKKELIT 52

1 JOHDANTO

Tietojärjestelmät ovat nykyaikaisen yhteiskunnan ytimessä. Ihmiset syntyvät ja kuolevat sairaaloissa, joille tietojärjestelmien sisältämät potilastiedot ovat kriittinen toiminnan edellytys. Raha on suurimmaksi osaksi dataa pankkijärjestelmissä. Liikkuminen tapahtuu pitkälti tietojärjestelmien avustamana, esimerkiksi lentokoneet ovat hyvin pitkälti ohjelmistoja alumiinikuoressa.

Tietojärjestelmät mahdollistavat kaikenlaisen yhteiskunnallisen toiminnan ennennäkemättömän tehokkaasti ja koordinoidusti, minkä vuoksi niiden on annettukin uppoutua niin syvälle yhteiskunnan rakenteisiin. Kääntöpuolena tässä on se, että järjestelmien epäonnistuessa täyttämään niille annettuja tehtäviä, sairaanhoito, pankkitoiminta tai lentoliikenne voivat mennä solmuun tavalla, johon ihmisen mielikuvitus ei riitä.

Yhteiskunnan kannalta tärkeisiin tietojärjestelmiin panostetaan paljon ja niiden toimintavarmuus pyritään takaamaan aina lainsäädäntöä myöten. Silloin tällöin kuitenkin myös tällaiset turvallisuuskriittiset järjestelmät epäonnistuvat syystä tai toisesta, jolloin niistä uutisoidaan yleensä näyttävästi.

Ohjelmistokehitys on alana jatkuvassa muutoksessa ja järjestelmätoimittajat kohtaavat jatkuvaa painetta tuottaa ohjelmistoja aiempaa nopeammin, tehokkaammin ja edullisemmin. Tähän paineeseen sopeutuakseen ohjelmistoja on alettu kehittää ketterin menetelmin. Turvallisuuskriittisten tietojärjestelmien kohdalla kehittäjät kohtaavat ristipaineen, kun nopeuden, tehokkuuden ja edullisuuden lisäksi tietojärjestelmiä pitäisi kehittää myös niin, että niiden toimintavarmuus on taattu.

Tämän tutkimuksen motiivina toimii havainto siitä, että turvallisuuskriittisetkin tietojärjestelmät ovat joskus haavoittuvia, vaikka seuraukset niiden epäonnistumisesta ovat pahimmillaan todella synkkiä. Tutkimuksella halutaan selvittää, onko ketterän ohjelmistokehityksen suuntaus löytänyt tiensä myös turvallisuuskriittisille toimialoille. Täten tutkimuksessa vastataan kysymykseen:

Voidaanko ketterän ohjelmistokehityksen suuntaus yhdistää turvallisuuskriittisten tietojärjestelmien epäonnistumisiin?

Samalla tutkitaan aihetta seuraavalla apukysymyksellä:

Millaisia ohjelmistokehityksen ongelmia turvallisuuskriittisten tietojärjestelmien kohdalla esiintyy?

Tutkimuksen tarkoituksena on luoda kuvaa turvallisuuskriittisten tietojärjestelmien ja ketterän ohjelmistokehityksen yhteydestä. Ketterä kehitys ei itsessään ole hyvä tai paha ilmiö, mutta on varmistettava, ettei sen sinällään kannatettavat arvot ohjaa väärällä tavalla sellaisten tietojärjestelmien kehittämistä, joiden toimintavarmuus on äärimmillään elämän ja kuoleman kysymys. Aiheesta keskusteleminen on näin ollen äärimmäisen tärkeää ja tällaisen keskustelun avaajana tämän tutkimuksen on tarkoitus toimia.

Tutkimus rakentuu kahdesta osasta: kirjallisuuskatsauksesta ja empiirisestä osasta. Kirjallisuuskatsaus muodostaa tutkimuksen 2. luvun. Luvussa tarkastellaan ohjelmistokehitystä ensin yleisesti, sitten ketterän ohjelmistokehityksen kontekstissa ja lopulta turvallisuuskriittisessä kontekstissa. Luvussa 3 esitellään tutkimuksen toteutus. Siinä perustellaan valittu tutkimusote ja tutkimusmenetelmä, sekä aineistoon liittyvät valinnat. Luvussa 4 esitellään keskeiset tutkimustulokset. Luvussa 5 kytketään teoriapohja ja tulokset yhteen ja lopulta luvussa 6 esitetään johtopäätökset.

Seuraavassa luvussa tarkastellaan siis ohjelmistokehityksen suuntauksia alkaen yleisestä katsauksesta edeten kohti tutkimuksen kannalta oleellisimpia näkökulmia. Templierin Parén (2015) mukaan systemaattisesti toteutetussa kirjallisuuskatsauksessa on kuusi vaihetta: ongelman muodostaminen, kirjallisuuden etsiminen, sisällyttäminen, laadun arvioiminen, tiedon kerääminen ja kerätyn datan analysointi/yhteenveto. Nämä vaiheet esitetään soveltuvaksi itsenäisiin kirjallisuuskatsauksiin. Tämä tutkimus ei ole systemaattinen kirjallisuuskatsaus, vaan kirjallisuuskatsaus toimii ainoastaan teoreettisena pohjana varsinaiselle tutkimukselle. Kirjallisuuskatsauksessa on kuitenkin soveltaen noudatettu Templierin ja Parén (2015) kuusivaiheista mallia.

Ensimmäinen vaihe, ongelman muodostaminen, oikeuttaa kirjallisuuskatsauksen olemassaolon. Kirjallisuuskatsauksen tarkoitus ja avainkonseptit on määriteltävä, sekä tutkimusongelma(t), joihin kirjallisuuskatsaus vastaa, kirjattava (Templier & Paré, 2015). Tämän kirjallisuuskatsauksen osalta ongelman muodostaminen on tehty luvussa 1. Avainkonseptien määrittely on tehty luvussa 2.

Toinen vaihe eli kirjallisuuden etsiminen tähtää siihen, että löydetään kaikki tutkimusongelman kannalta oleellinen kirjallisuus ja tämä edellyttää tietolähteiden tunnistamista (Templier & Paré, 2015). Tätä kirjallisuuskatsausta varten on käytetty seuraavia hakusanoja: *software engineering, agile software engineering, agile software development, agile software challenges.* Haut tehtiin IEEE Xplore ja EBSCOhost -tietokannoissa. Täydentävästi käytettiin myös Google Scholaria.

Kolmas vaihe on määrittää, mitkä löydetyistä, potentiaalisista tutkimuksista ovat vastattavien tutkimuskysymysten kannalta relevantteja (Templier & Paré, 2015). Tutkimuksessa käytettyä teoriaa rajattiin niin, että tarkasti erilaisia

ohjelmistonkehitysmenetelmiä koskevat tutkimukset jätettiin pois. Niitä on käytetty esimerkinomaisesti joissain yhteyksissä, mutta niistä ei muodostettu kirjallisuuskatsauksen runkoa käytetyn aineiston vuoksi - aineisto ei salli syvälle meneviä johtopäätöksiä eri kehitysmenetelmien mahdollisista puutteista. Käytetyn aineiston perustelut on esitetty alaluvussa 3.3. Relevantit tutkimukset olivat näin ollen sellaisia, jotka tutkivat ketterää ohjelmistokehitystä ja sen haasteita ylätasolla, ketterän ohjelmistokehityksen arvoista käsin.

Neljäs vaihe, laadun arviointi, vaatii relevanteiksi määritettyjen lähteiden tutkimusasetelman ja niissä käytettyjen tutkimusmenetelmien arviointia (Templier & Paré, 2015). Tässä tutkimuksessa laatu pyrittiin varmistamaan ensisijaisesti etsimällä kirjallisuuslähteitä asemansa vakiinnuttaneista, vertaisarvioiduista julkaisuista. Jonkin verran käytettiin myös kirjallisuutta, sekä konferenssijulkaisuja.

Viides vaihe on tiedon kerääminen ja siinä valikoidaan kirjallisuuskatsaukseen sisällytetyistä lähteistä oleellinen tieto. Tiedon oleellisuus määräytyy pitkälti tutkimuskysymysten mukaan. (Templier & Paré, 2015.) Tieto on kerätty tämän tutkimuksen lukuun 2 ja tiedon oleellisuus perustellaan luvussa 5.

Kuudes vaihe, datan analysointi ja sen yhteenveto, tiivistää, järjestää ja vertailee löydettyä tietoa (Templier & Paré, 2015). Tässä tutkimuksessa kuudes vaihe toteutuu luvuissa 4 ja 5.

2 OHJELMISTOKEHITYS

Tässä luvussa käydään läpi tutkimuksen kannalta relevantti teoreettinen käsitteistö. Teoriaosuus on pyritty muodostamaan niin, että ensin luodaan konteksti, joka tässä tapauksessa on ohjelmistokehitys ja siitä tarkennetaan kohti tarkemmin rajattua tutkimuksen aihepiiriä eli turvallisuuskriittisiä tietojärjestelmiä ja niiden suhdetta ketterän kehityksen arvioihin.

Alaluvussa 3.1. esitellään lyhyesti ohjelmistokehityksen nykyistä edeltävä paradigma, suunnittelupainotteinen ohjelmistokehitys. Roycen (1970) vesiputousmalli toimii runkona ohjelmistokehityksen perustehtävien esittelylle. Nämä tehtävät eivät sinällään ole muuttuneet, vaikka ketterä ohjelmistokehitys onkin kasvattanut suosiotaan koko 2000-luvun (Douglass, 2015).

Alaluvussa 3.2. esitellään nykyinen ohjelmistokehityksen paradigma, ketterä ohjelmistokehitys. Aluksi käydään läpi Agile Manifestossa (Beck ym, 2001) julkaistut ketterän ohjelmistokehityksen arvot ja periaatteet, josta edetään käsitteen määrittelyyn. Ketterän ohjelmistokehityksen ominaispiirteitä verrataan suunnittelupainotteisen ohjelmistokehityksen piirteisiin.

Alaluvussa 3.3. esitellään ketterän ohjelmistokehityksen haasteita tutkimusten valossa. Näkökulma on ihmisten ja organisaatioiden toiminnassa. Haasteita peilataan Agile Manifestossa julkaistuihin ketterän ohjelmistokehityksen arvoihin.

Alaluvussa 3.4. tarkastellaan ketterän ohjelmistokehityksen haasteita turvallisuuskriittisten tietojärjestelmien kontekstissa. Siinä tarkastellaan niitä sääntelyn tuottamia ristiriitoja, joita ketterän ajattelutavan soveltaminen tuo kehitettäessä turvallisuuskriittisiä tietojärjestelmiä.

2.1 Ohjelmistokehityksen perustehtävät

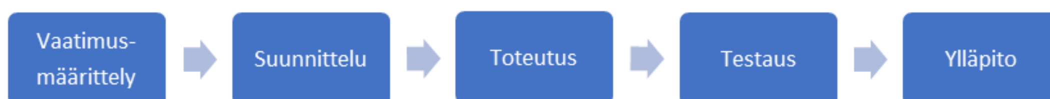
Ohjelmistokehitys on systemaattisen ja mitattavan menetelmän soveltamista ohjelmiston kehitykseen, operointiin ja ylläpitoon (IEEE, 1990).

Tietojärjestelmien kehittäminen nähtiin aikaisemmin vaihe vaiheelta etenevänä työnä, jossa yksi vaihe suoritettiin loppuun ennen seuraavaan siirtymistä. Roycen (1970) vesiputousmalli on tästä tunnettu prosessimalli. Vaatimus-

määrittelyn jälkeen seuraa suunnittelu, josta edetään toteutukseen. Tämän jälkeen ohjelmistoa testataan ja ylläpidetään (kuvio 1).

Yleisesti vesiputousmallin mukainen kehittäminen nähdään soveltuvaksi järjestelmiin, joissa vaatimukset ja riskit ovat etukäteen tiedossa ja ymmärretty, eikä odotettavissa ole oleellisia muutoksia (Heeager & Nielsen, 2018). Vesiputousmallia on kritisoitu joustamattomuudesta ja hitaudesta (Sommerville, 2016). Vaihe vaiheelta etenevä tuotantoprosessi ei useinkaan vastaa modernin ohjelmistokehityksen toimintaympäristöä ja 2000-luvun alussa esitetyt ketterän ohjelmistokehityksen periaatteet muuttivatkin ohjelmistotuotantoa kohti iteratiivista ja inkrementaalista kehitystapaa.

Vesiputousmalli on kuitenkin selkeä tapa esitellä ohjelmistokehityksen perustehtävät, jotka käydään seuraavaksi läpi alla olevan kuvion 1 mukaisessa järjestyksessä.



KUVIO 1 Vesiputousmalli

2.1.1 Vaatimusmäärittely

Vaatimusmäärittely sisältää laajasti ajateltuna kaiken, mikä liittyy vaatimusten löytämiseen, dokumentointiin ja hallintaan (Kotonya & Sommerville, 1998). Pohl ja Rupp (2011) nimeävät neljä ydintehtävää vaatimusmäärittelylle, joita ovat edellä mainittujen lisäksi vielä sidosryhmien kanssa neuvottelu. Vaatimukset voidaan jakaa toiminnallisiin ja ei-toiminnallisiin vaatimuksiin.

Vaatimuksia voidaan löytää sidosryhmien, jo olemassa olevan dokumentoinnin ja jo olemassa olevien järjestelmien kautta. Dokumentoinnilla tarkoitetaan tässä yhteydessä esim. erilaisia standardeja tai lainsäädäntöä, jotka voivat luoda vaatimuksia. Jo olemassa olevat järjestelmät taas saattavat luoda vaatimuksia esim. sen kautta, että ne on saatava toimimaan uuden järjestelmän kanssa. Järjestelmät voivat myös tarjota sidosryhmille tarttumapintaa siihen, millainen uuden järjestelmän tulisi olla. (Pohl & Rupp, 2011.)

Sidosryhmiä ovat järjestelmän tilaajat ja sen käyttäjät, mutta myös järjestelmän kehittäjät, arkkitehdit ja testaajat. Glinzin ja Wieringan (2007) mukaan nimenomaan oleellisten sidosryhmien löytäminen on vaatimusmäärittelyn kriittisin osa. Jos oikeita sidosryhmiä ei tunnisteta tai niiden vaatimukset jätetään huomiotta, järjestelmään halutaan myöhemmin muutoksia ja se on yleensä

kallista. (Pohl & Rupp, 2011.) Vaatimusmäärittelyssä tarvitaan aina ohjelmiston kehittäjien ja asiakkaan tiivistä yhteistyötä (Wiegers & Beatty, 2013).

Vaatimuksia dokumentoidaan, koska suurissa järjestelmähankkeissa vaatimuksia saattaa olla tuhansia ja niillä voi olla monimutkaisia keskinäisiä vaikutussuhteita. Ilman dokumentointia ohjelmiston kehittäminen olisi tällöin hyvin hankalaa. Toisaalta dokumentointia tehdään jatkuvuuden varmistamiseksi. Järjestelmähankkeissa tapahtuu usein ennakoimattomia muutoksia, esim. henkilöstönvaihdoksia. Vaatimusten dokumentointi takaa sen, että hankkeeseen kesken kaiken tuleva saa nopeasti kuvan kehitteillä olevasta järjestelmästä. Kolmanneksi dokumenteilla on laillista merkitystä. Kirjatut vaatimukset ovat laillisesti sitovia ja mahdollisissa epäselvissä tilanteissa dokumentointi toimii asianosaisten oikeusturvana. (Pohl & Rupp, 2011.)

Vaatimusten hallinnalla viitataan mm. vaatimusten priorisointiin, versiointiin ja statuksen seurantaan (Pohl & Rupp, 2011). Dokumentoinnin tavoin hallinnan tarve on sitä suurempi, mitä isommasta järjestelmähankkeesta on kyse. Vaatimusten hallinta linkittyy läheisesti yleiseen projektinhallintaan. Priorisoinnin avulla osataan kohdistaa resursseja kaikista oleellisimpien vaatimusten kehittämiseen ja versioinnin ja statuksen seurannan avulla resurssit voidaan kohdentaa oikea-aikaisesti oikeille vaatimuksille. (Hull, Jackson & Dick (2011.)

Vaatimusmäärittelyä pidetään yhtenä keskeisimmistä tehtävistä ohjelmistojen kehittämisessä (Boehm, 1981) ja sen laiminlyönnillä on usein kehitettävän järjestelmän kannalta vakavat seuraukset joko niin, että järjestelmää ei koskaan edes oteta käyttöön tai niin, että valmiin järjestelmän toiminnassa on puutteita.

2.1.2 (Arkkitehtuuri)suunnittelu

Sommervillen (2016) mukaan suunnittelun perusta on arkkitehtuurisuunnittelu. Ohjelmistoarkkitehtuuri toimii linkkinä vaatimusmäärittelyyn, sillä siinä tunnistetaan kehitettävän järjestelmän pääkomponentit ja niiden väliset suhteet. Vaatimusmäärittely on suunnitteluun niin vahvasti sidoksissa, että käytännössä ne lomittuvat. Ideaalitilanteessa ne pidetään erillään. Tämä on kuitenkin mahdollista vain todella pienien järjestelmien kohdalla, koska arkkitehtuurin pääkomponentit heijastavat järjestelmän vaatimuksia.

Arkkitehtuurin suunnittelusta on kolmenlaista hyötyä. Sillä voidaan kommunikoida sidosryhmien kanssa, koska arkkitehtuuri edustaa järjestelmää ylätasolla. Toiseksi arkkitehtuurin suunnittelu vaatii järjestelmän kriittistä tarkastelua – voidaanko tietyllä ratkaisulla saavuttaa järjestelmän ydintehtävät eli suorituskyky, luotettavuus ja ylläpidettävyys. Kolmanneksi kerran suunniteltu arkkitehtuuri on uudelleenkäytettävissä järjestelmiin, joissa on samankaltaiset vaatimukset, kuin mitä varten arkkitehtuuri suunniteltiin. (Bass, Clements & Kazman, 2012.)

Tietojärjestelmällä on väistämättä arkkitehtuuri. Jos sitä ei suunnitella, se muotoutuu järjestelmän elinkaaren aikana tehtävien ratkaisujen myötä. Tällainen tilanne altistaa järjestelmän haavoittuvaksi muutostilanteissa eli esimerkik-

si, kun järjestelmää päivitetään tai tapahtuu henkilöstövaihdoksia. (Bass, Clements & Kazman, 2012.)

Arkkitehtuurin rooli on sitä merkittävämpi, mitä pidempi elinkaari suunniteltavalla järjestelmä on ja mitä monimutkaisempi se on. Suurissa järjestelmissä arkkitehtuurilla on ratkaiseva merkitys ohjelmiston laatuun ja ohjelmistokehityksen läpimenoaikaan. (Bosch, 2000.)

2.1.3 Toteutus

Ohjelmistotuotannossa oli aluksi tavallista, että kehitettävät ohjelmistot luotiin alusta alkaen koodaamalla. Hankkeiden koon kasvaessa ja aikataulujen kiristyessä tämä on kuitenkin nykyään käytännössä mahdotonta. Toteutuksen kulmakiveksi onkin noussut uudelleenkäytettävyys. Pyörää ei ole mielekästä keksiä aina uudestaan. (Sommerville, 2016.)

Uudelleenkäyttöä voi tapahtua abstraktio-, olio-, komponentti- ja järjestelmätasolla. Abstraktiotaso liittyy oikeastaan enemmän suunnitteluun, sillä siinä käytetään hyväksi havaittuja suunnittelu- ja arkkitehtuurikaavoja kehitettävän ohjelmiston mallinnukseen. Oliotasolla käytetään kirjastoja valmiin koodin tuomiseksi ohjelmistoon. Komponenttitasolla käytetään tiettyjen olioiden ja olioluokkien ryhmiä, joihin yleensä lisätään omaa koodia. Esimerkiksi käyttöliittymä tehdään usein uudelleenkäyttäen komponentteja. Järjestelmätasolla käytetään uudelleen kokonaisia järjestelmiä. Yleensä nekin vaativat jonkinlaista muuntamista juuri kyseessä olevan tapauksen vaatimukset täyttäviksi. Yksittäisen järjestelmän lisäksi voidaan uudelleenkäyttää myös useita järjestelmiä, jotka integroidaan kokonaisuudeksi. (Sommerville, 2016.)

Uudelleenkäyttöä tehdään, koska se nopeuttaa kehitystyötä, vähentää kehittämiseen liittyviä riskejä ja vie vähemmän resursseja, kuin alusta asti itse kehittäminen. Siihen liittyy kustannuksia, kuten uudelleenkäyttömahdollisuuksien löytämiseen käytetty aika, mahdolliset lisenssimaksut ja integroimiseen kuluvat resurssit. Silti uudelleenkäyttö on yleisesti ottaen halvempaa kuin tyhjästä aloittaminen. (Sommerville, 2016.)

Uudelleenkäytön lisäksi toteutuksessa tarvitaan konfiguraationhallintaa. Se käsittää versionhallinnan, järjestelmäintegraation, ongelmien jäljityksen ja julkaisujen hallinnan. Toteutus on järjestettävä niin, että eri kehittäjien työ välttää päällekkäisyydet. Ilman toimivaa versionhallintaa riskinä on muokata väärää versiota, toimittaa väärä versio asiakkaalle tai unohtaa, missä versiossa mikin muutos on tehty. Järjestelmäintegraatiolla hallitaan sitä, mitkä eri komponenttien versiot muodostavat tietyn version kehitettävästä järjestelmästä. Ongelmien jäljityksellä mahdollistetaan bugeista raportointi ja seurataan, kuka ongelmat korjaa ja milloin ne on korjattu. Julkaisujen hallinta liittyy ketterään kehitykseen, jossa julkaisuja on useita ohjelmistoprojektin aikana. Julkaisujen hallinnalla kontrolloidaan sitä, mitä toiminnallisuuksia mihinkin julkaisuun sisältyy. (Sommerville, 2016.)

2.1.4 Testaus

Testauksen tarkoituksena on varmistaa, että ohjelmisto tekee sen, mitä sen on tarkoitus tehdä. Tällöin puhutaan validointitestauksesta. Testaus osoittaa ohjelmiston kehittäjille ja asiakkaille, että ohjelmisto täyttää sille määritellyt vaatimukset. Siksi kutakin vaatimusta tulisi testata vähintään yhdellä testillä. (Sommerville, 2016.)

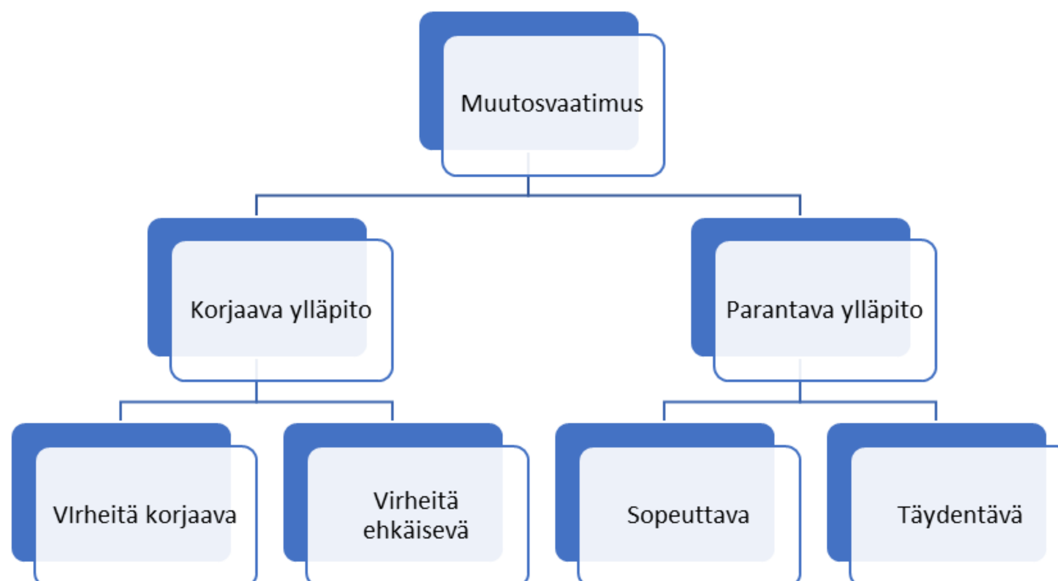
Lisäksi testauksella pyritään löytämään puutteita ja vikoja. Tietyillä syötteillä pyritään löytämään tilanteita, joissa ohjelmisto ei toimi oikein. (Sommerville, 2016.) Testaus, joka ei tuota virheilmoituksia, ei ole välttämättä onnistunut. Testauksella voidaan osoittaa virheiden läsnäolo, mutta niiden olemassaoloa testauksella ei voida yleispätevästi havaita. (Dijkstra, 1972.)

Testauksella haetaan varmuutta siitä, että ohjelmisto on tarkoitukseensa sopiva. Sen on oltava *riittävän* hyvä. Varmuuden ja laadun vaatimustaso riippuu ensinnäkin järjestelmän käyttötarkoituksesta. Mitä kriittisempi järjestelmä, sitä huolellisemmin se on testattava käyttövarmuuden takaamiseksi. Toisena tekijänä vaikuttaa käyttäjien odotukset. Yleisesti ottaen uudelta järjestelmältä siedetään enemmän viallisuutta siinä tapauksessa, että järjestelmän käytön hyödyt ovat suuremmat kuin siitä koituvat kustannukset. Järjestelmän vakiintuessa siltä aletaan kuitenkin odottaa vakaampaa toimintaa. Kolmantena vaikuttaa liiketoimintaympäristö. Kilpailevien järjestelmien laatu vaikuttaa odotuksiin kehitettävän järjestelmän laadusta. Jos toimiala on kovin kilpailtu, se voi aiheuttaa painetta julkaista järjestelmä ensimmäisenä. (Sommerville, 2016.)

Testaamisessa on tyypillisesti kolme vaihetta: kehitys-, julkaisu- ja käyttäjätestaus. Kehitystestaus on pääasiassa bugien löytämistä ja korjaamista. Sitä tehdään toteutuksen lomassa ja yleistä on myös testien kirjoittaminen ennen varsinaista koodia. Tällöin puhutaan testivetoisesta kehittämisestä (engl. *test-driven development*). Kehitystestaamisessa testaaja on yleensä koodari. Julkaisutestauksessa erillinen testiryhmä testaa kokonaista ohjelmiston versiota ja pyrkii selvittämään, vastaako se sidosryhmien vaatimuksia. Käyttäjätestauksessa ohjelmiston versio annetaan käyttäjien testattavaksi heidän omassa ympäristössään. Käyttäjä ei tässä yhteydessä välttämättä ole järjestelmän lopullinen asiakas, vaan esimerkiksi sisäinen markkinointiryhmä. Käyttäjätestauksen tarkoituksena on päättää, voidaanko versio julkaista. (Sommerville, 2016.)

2.1.5 Ylläpito

Järjestelmän ylläpidolla tarkoitetaan yleisesti ottaen sen muuttamista sen jälkeen, kun järjestelmä on toimitettu asiakkaalle (Sommerville, 2016). ISO/IEC 14764 -standardi jaottelee ylläpidon korjaavaan ja parantavaan ylläpitoon. Korjaava ylläpito on mahdollista erottaa virheitä korjaavaksi ja virheitä ehkäiseväksi ylläpidoksi. Parantava ylläpito voidaan jakaa sopeuttavaksi ja täydentäväksi ylläpidoksi (kuvio 2).



KUVIO 2 Ylläpidon luokittelu (ISO/IEC, 2006)

Ylläpidossa voidaan korjata bugeja ja haavoittuvuuksia koodin tasolla. Tämä on yleensä suhteellisen halpaa, mutta melko yleistä. Suunnitteluvirheet ovat kalliimpia korjata, koska ne vaativat järjestelmän laajempaa muokkaamista. Vaatimuksista nousevat virheet ovat kalleimpia, koska ne voivat vaatia jopa järjestelmän osittaista uudelleensuunnittelua. (Sommerville, 2016.) Davidsen ja Krogstie (2010) havaitsivat pitkittäistutkimuksessaan, että ylläpitokustannuksista noin 24 % muodostuu virheiden korjaamisesta.

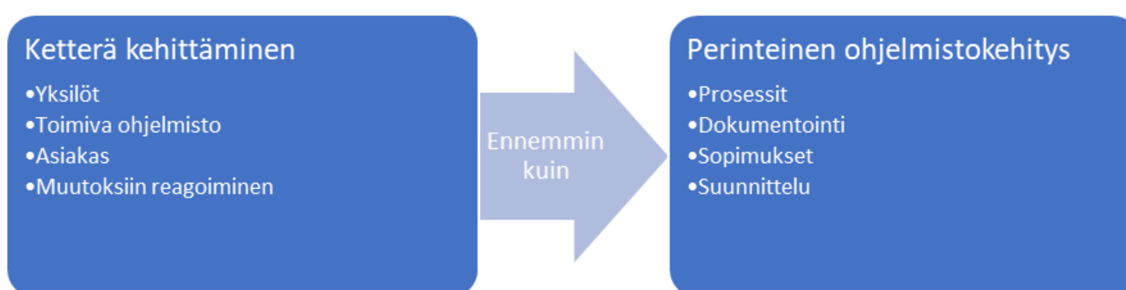
Virheiden korjaamisen lisäksi järjestelmää on tarvetta ylläpitää muuttuvan ympäristön vuoksi. Ympäristöllä tarkoitetaan tässä yhteydessä laitteiston, alustan tai tukiohjelmistojen muutoksia. (Sommerville, 2016.) ISO/IEC 14764 -standardissa ympäristön muutokset ovat sopeuttavaa ylläpitoa. Ne muodostavat noin 19 % ylläpitokustannuksista (Davidsen & Krogstie, 2010).

Järjestelmää voidaan muuttaa myös toiminnallisuuksien osalta, kun uusia vaatimuksia ilmenee tai liiketoimintaympäristö muuttuu (täydentävä ylläpito). Nämä ylläpitotoimet ovat yleensä kokoluokaltaan paljon virheiden korjaamista tai ympäristöön sopeutumista suurempia. (Sommerville, 2016.) Ne muodostavatkin noin 58 % ylläpitokustannuksista (Davidsen & Krogstie, 2010).

Ylläpito muodostaa kokonaisuutena merkittävän osan koko järjestelmäkehitysprosessin kustannuksista, Glassin (2001) mukaan keskimäärin noin 60 %. Ylläpito ajoittuu järjestelmän elinkaaren loppupäähän ja silloin muutosten tekeminen on aina hankalampaa ja kalliimpaa, kuin vaatimusmäärittelyn tai suunnittelun aikana (Boehm, 1981).

2.2 Ketterä ohjelmistokehitys

Ketterä ohjelmistokehitys on monitahoinen termi. Se käsittää useita eri kehitysmenetelmiä ja laajimmillaan sen voi ajatella olevan kokoelma arvoja, joiden pohjalta tuottaa ohjelmistoja, kuten on Agile Manifeston (Beck ym., 2001) tapauksessa. Agile Manifeston perusarvoina ovat yksilöt ja yksilöiden välinen vuorovaikutus, toimiva ohjelmisto, asiakasyhteistyö ja muutoksiin reagoiminen. Näitä suositaan prosessien, dokumentoinnin, sopimusten tekemisen ja suunnitelmien sijaan (kuvio 3).



KUVIO 3 Ketterän ohjelmistokehityksen arvot

On huomioitava, että ensiksi mainitut ketterän kehityksen arvot eivät sulje jälkimmäisiä toimintoja pois, ne ovat vain preferenssi. (Beck ym., 2001.) Ketteryyden tarkoitus ei ole hylätä esimerkiksi suunnittelua tai dokumentointia, se vain muuttaa järjestystä ja sitä millä sekvenssillä eri ohjelmistotuotannon toimia tehdään (Douglass, 2015). Beck ym. (2001) johtavat arvoista periaatteet, jotka tarkentavat, miten edellä mainittuja arvoja tulisi soveltaa (taulukko 1).

TAULUKKO 1 Ketterän ohjelmistokehityksen periaatteet

Periaate
Tärkein painopiste on tuottaa asiakkaalle arvokasta ohjelmistoa aikaisessa vaiheessa ja jatkuvasti
Muutokset ovat tervetulleita missä vaiheessa tahansa, ne ovat mahdollisuuksia kilpailuedun saavuttamiseen
Toimivaa ohjelmistoa toimitetaan säännöllisesti mahdollisimman lyhyellä intervallilla
Liiketoiminnan edustajat ja ohjelmistokehittäjät työskentelevät päivittäin yhdessä koko ohjelmistoprojektin ajan
Projektit luodaan motivoituneiden yksilöiden ympärille
Tehokkain keino tiedon välittämiseen on kasvokkain kommunikointi
Toimiva ohjelmisto on ensisijainen edistymisen mittari
Kehitystyö on kestävää – työtahti on sellainen, että sidosryhmät kykenevät pitämään sen yllä
Huomion kiinnittäminen tekniseen laatuun ja hyvään suunnittelutyöhön
Yksinkertaisuus, maksimoidaan työ, jota ei tehdä
Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä
Tiimi tarkastelee omia työtapojaan säännöllisesti ja tekee tarvittavat muutokset tullakseen tehokkaammaksi

Drury-Grogan, Conboy ja Acton (2017) argumentoivat, että ketterää kehitystä on hedelmällistä tarkastella nimenomaan ilmiön taustalla olevien periaatteiden kannalta (Beck ym., 2001; taulukko 1) sen sijaan, että keskityttäisiin yksittäiseen kehitysmenetelmään, koska ketteryys on konseptina epämääräinen ja tietty menetelmä ei juuri koskaan esiinny puhtaassa muodossaan reaalimaailmassa.

Toisaalta ketteryyttä ei voida uskottavasti määritellä pelkästään Agile Manifeston kautta, koska se on luonteeltaan yleinen julistus ja näin ollen liian epämääräinen tieteelliseen tutkimukseen (Laanti, Similä & Abrahamsson, 2013). Kaikille Agile Manifestossa esitetyille periaatteille ei löydy suoraa teoreettista pohjaa (Batra, VanderMeer & Dutta, 2011).

Laanti ym. (2013) kyseenalaistavat ketteryydestä tai sen hyödyistä keskustelemisen yleisellä tasolla, koska termi on laaja ja lukuisten määrittelyjen avulla sen voidaan kuvitella olevan hopealuoti - ratkaisu kaikkiin ohjelmistokehityksen ongelmiin. Sen sijaan olisi kenties hyödyllistä tarkastella eri ketteriä käytänteitä ja niiden etuja. Conboy (2009) huomauttaa, että liian triviaalit mallit taas eivät ole yleistettäviä ja näin ollen hyödyllisiä koko ilmiön kuvaamiseksi.

Määrittelyjä on joka tapauksessa lukuisia, ne vaihtelevat laajuudeltaan ja niissä on erilaisia painotuksia (Laanti ym., 2013). Cockburn (2001) kuvaa ketteryyden olevan tehokkuutta ja toteutettavuutta. Andersonin (2003) mukaan ketteryys on kykyä nopeuttaa tekemistä ja Larman (2004) kuvailee ketteryyden olevan nopeaa ja joustavaa reagointia muutokseen. Schuh (2004) toteaa, että ketteryys on ohjelmistojen rakentamista ihmisiin luottaen, muutosten mahdollisuus tiedostaen ja palautetta jatkuvasti saaden. Subramaniamin (2005) mukaan ketteryys on jatkuvaa palautteen perusteella sopeutumista yhteistyötä korostavassa ympäristössä.

Nerurin ja Balijepallyn (2007) mukaan ketteryys on ihmiskeskeistä, yhteistyötä painottavaa ja sidosryhmien aktiivista osallistumista kehitystyöhön. Se on myös muutoksen luomista ja hyödyntämistä. Nerur ja Balijepally (2007) liittävät ketteryyteen myös laajemman, historiallisen näkökulman: ohjelmistokehityksen muuntuminen ketteräksi ei ole ainutlaatuista, vaan samansuuntaisia ajatusmalleja on kehittynyt jo aiemmin esimerkiksi strategisen johtamisen alalla. Ketterä kehitys on siis ohjelmistokehityksen älyllistä evoluutiota.

Määritelmän ollessa lyhyt, se ei voi ottaa huomioon kaikkia ketterään kehitykseen sisältyviä periaatteita (taulukko 1). Pituus ei sinänsä kerro määritelmän laadusta, mutta yleisesti ottaen pidempi määrittely voi ottaa ilmiön huomioon monitahoisemmin. (Laanti ym., 2013.) Yksi laajimmista ketterän kehityksen periaatteiden kiteytyksistä on Conboyn (2009, s. 340) määrittely:

> Kehittämismenetelmän jatkuva valmius nopeasti tai luontaisesti luoda muutosta, ennakoivasti tai reagoivasti tukea muutosta ja oppia muutoksesta samalla lisäten asiakkaan saamaa arvoa (tuottavuus, laatu ja yksinkertaisuus) yhteisten komponenttien ja ympäristösuhteiden avulla.

Conboy (2009) purkaa yllä olevan määrittelyn luokitteluksi, jonka ensimmäinen osatekijä on juuri suhtautuminen muutokseen. Ollakseen ketterä, kehitysmeto-

din tulee joko luoda muutosta, edistää sitä, reagoida siihen tai oppia siitä. Luonnollisesti metodi voi sisältää useampia kuin yhden näistä tunnuspiirteistä. Ketterien kehitysmetodien kykyä toimia muutoksessa käsitellään Conboyn (2009) mukaan yleisesti vaatimusten muuttumisen kautta, mutta muutosvalmius voidaan käsittää laajemminkin - esimerkiksi henkilöstö-, budjetti-, sopimus- tai laitteistomuutoksiin vastaaminen voi olla ketteryyttä.

Toisaalta ketteryys vaatii joko tuottavuuden, laadun tai yksinkertaisuuden edistämistä niin, ettei yksikään näistä kolmesta kärsi. Näiltä osin ketteryys on lähellä lean-ajattelua, jossa pyritään karsimaan kaikki sellainen, mikä ei tuota asiakkaalle lisäarvoa. Ketteryys ei kuitenkaan ole sama asia kuin lean, sillä jälkimmäinen soveltuu parhaiten standardoituun toimintaan, tunnetuimpana esimerkkinä Toyotan autonvalmistus. Ohjelmistokehitys on muutosalttiuden vuoksi melko kaukana liukuhihnatuotannosta. (Conboy, 2009.)

Kolmantena ketterän kehitysmetodin tunnuspiirteenä on jatkuva valmius. Ollakseen ketterä, metodia on pystyttävä käyttämään ilman mittavia valmistelukuluja tai valmisteluun käytettävää aikaa. (Conboy, 2009.)

Kun ketterää kehitystä tarkastellaan suhteessa vesiputousmallin mukaiseen, suunnittelupainotteiseen ohjelmistokehitykseen, ketteryyden ydin on raskaista prosesseista ja suunnittelupainotteisuudesta luopuminen, jotta muutoksiin on mahdollista reagoida joustavasti (Erickson, Lyytinen & Siau, 2005). Suunnittelupainotteisuudessa taustalla on oletus, että järjestelmät ovat etukäteen määriteltävissä ja siinä määrin ennustettavissa, että ne voidaan rakentaa etukäteen suunnitellen. Ketterä kehitys haastaa tämän käsityksen ja korostaa muutosta, siihen reagoimista ja palautteen tärkeyttä. (Nerur, Mahapatra & Mangalaraj, 2005.)

Ketterää kehitystä voidaan verrata perinteiseen ohjelmistokehitykseen tarkastelemalla kummankin ominaispiirteitä (taulukko 2).

TAULUKKO 2 Ketterien ja suunnittelupainotteisten menetelmien erot (Boehm 2002, s. 68 mukaan)

Tarkasteltava alue	**Ketterät menetelmät**	**Suunnittelupainotteiset menetelmät**
Kehittäjät	Osaavia ja yhteistyökykyisiä	Suunnitteluorientoituneita, riittävän taitavia, pääsy ulkoiseen tietoon
Asiakkaat	Omistautuneita, tietäviä, yhteistyökykyisiä, osallistuvia	Tarvittaessa yhteistyö asiakkaiden kanssa
Vaatimukset	Muodostuvat projektin aikana; nopea muutos	Aikaisin tiedossa; vakaus
Arkkitehtuuri	Suunniteltu nykyisille vaatimuksille	Suunniteltu nykyisille ja ennakoitaville vaatimuksille
Muokkaaminen	Edullista	Kallista
Projektin koko	Pienemmät ryhmät ja tuotteet	Suuremmat ryhmät ja tuotteet
Päätavoite	Nopea arvon tuottaminen	Korkea varmistamistaso

Ketteryyden tavoitteena on nopea arvon tuottaminen, suunnittelupainotteisesti toimiessa taas korkea varmistamistaso. Kuten ketterän kehityksen taustaperiaatteita tarkastellessa, on myös ajattelumallien ominaispiirteiden kohdalla huomioitava, etteivät piirteet aina sulje toisiaan pois – usein asiakas vaatii järjestelmätoimittajalta sekä nopeaa arvon tuottamista että varmuutta. (Boehm, 2002.)

Tarkoin määritetyt, toistettavat prosessit ovat vesiputousmallin kehityksessä avainasemassa ja tämä näkyy kehittäjien roolissa (taulukko 2, ”kehittäjät): he toteuttavat suunnitelmaa ja heidän johtamisensa on kontrollipainotteista. Ketterässä kehityksessä paino on kehitystiimin omassa aktiivisuudessa (Overhage, Schlauderer & Birkmeier, 2011). Kehittäjien oma-aloitteisuus vaatii sitä, että kehittäjät ovat osaavia ja yhteistyökykyisiä. Suunnittelupainotteisesti toimiessa taitovaatimukset ovat lähtökohtaisesti matalampia, koska suunnittelu ohjaa työskentelyä. (Boehm, 2002.) Toteutus on tällöin suunnitelman toteuttamista enemmän kuin yhteistyössä tapahtuvaa luovaa ongelmanratkaisua (Nerur ym., 2005).

Suunnittelupainotteisesti toteuttaessa asiakkaaseen ollaan yhteydessä silloin, kun on tarve eli pitkälti kehitystyön alkuvaiheessa vaatimusmäärittelyn ja suunnittelun aikana. Oletuksena on, että asiakkaat ovat tuolloin yhteistyökykyisiä ja osallistuvia (taulukko 2, ”asiakkaat”). Ketterässä kehityksessä asiakkaaltakin vaaditaan enemmän: he ovat mukana kehitystyössä omistautuneesti ja heidän palautteensa on oleellinen osa kehitystyön onnistumista. Tähän sisältyy oletuksena, että asiakas tietää mitä haluaa ja osaa viestiä siitä kehittäjille. (Boehm, 2002.) Kun suunnittelupainotteisessa kehitystyössä asiakkaan rooli on tärkeä, ketterässä kehittämisessä se on suorastaan kriittinen (Nerur ym., 2005).

Vaatimusmäärittely toteutuu suunnittelupainotteisessa kehittämisessä projektin aluksi ja vaatimukset pysyvät projektin ajan suhteellisen muuttumattomina. Ketterässä kehityksessä vaatimukset voivat muuttua nopeastikin ja ne täsmentyvät, kun kehitystyö etenee. (Boehm, 2002; taulukko 2, ”vaatimukset”.)

Ketterässä kehittämisessä vaatimusmäärittely elää koko kehitysprojektin ajan. Listattujen vaatimusten sijaan käytetään käyttäjätarinoita, joista vaatimuksen johdetaan. Tämä tekee vaatimusmäärittelystä toisaalta realistisempaa, koska varsinkin laajojen järjestelmien kohdalla kaikkien vaatimusten määrittäminen etukäteen on lähes mahdotonta. Toisaalta se tuo tiettyä epävakautta kehittämiseen ja aiheuttaa sen, että toteutettavan järjestelmän toimivuuden kannalta oleelliseksi seikaksi muodostuu ohjelmistokehittäjien ja asiakasorganisaation välinen viestintä. Koska yksityiskohtaista sopimusta vaatimuksista ei välttämättä ole, kehitystiimin tulee iteroinnin yhteydessä olla jatkuvasti tietoinen siitä, mikä muuttuu vaatimusten kannalta. (Ramesh, Cao & Baskerville, 2007.)

Suhtautuminen vaatimuksiin heijastuu myös järjestelmän arkkitehtuuriin (taulukko 2, ”arkkitehtuuri): ennalta lukkoon lyötyjen vaatimusten pohjalta luodaan arkkitehtuuri juuri näille vaatimuksille. Kun ketterästi oletetaan, että vaatimukset tulevat muuttumaan, arkkitehtuuri luodaan sillä hetkellä tiedossa oleville vaatimuksille. (Boehm, 2002.)

Jo kertaalleen suunniteltujen elementtien muokkaaminen on suunnittelupainotteisesti toimittaessa kallista, koska oletuksena on, että muutoksia ei tule. Ketterässä kehityksessä muutos on oletettavaa ja osa prosessia, joten muutokset ovat huomattavasti edullisempia toteuttaa. (Boehm, 2002; taulukko 2, "muokkaaminen".) Ketterä kehitys on vahvasti iteratiivista. Kokonaisuuden ennalta suunnittelun sijaan suunnittelua tehdään yksityiskohtaisella ja yleisellä tasolla. Yksityiskohtainen suunnittelu edeltää kutakin iteraatiota ja keskittyy iteraation eli esimerkiksi Scrum-viitekehyksen sprintin tavoitteisiin. Yleinen suunnittelu taas koskee ohjelmiston julkaisua. (Leffingwell, 2007.)

Boehm (2002) näkee suunnittelupainotteisuuden ja ketteryyden erona myös projektin koon (taulukko 2, "projektin koko"): suunnittelupainotteisuus soveltuu isoihin projekteihin ja ketteryys pieniin. Tämä ero on hämärtymässä. Ketterän ohjelmistokehityksen nähtiin aluksi soveltuvan pienten tiimien lähestymistavaksi ja 2000-luvun alun tutkimus aiheesta keskittyikin pitkälti ketterien menetelmien käyttöönottoon pienessä mittakaavassa (Dingsøyr ym., 2012). Menetelmien vakiintuessa kiinnostus ketterien menetelmien soveltamiseen yhä isompien it-projektien kohdalla on kasvanut (Dingsøyr & Moe, 2013).

Isojen, pitkäkestoisten ja globaalien it-projektien kohdalla ketteriä kehitysmenetelmiä on todennäköisesti muokattava niin, että niissä on myös ei-ketteriä elementtejä (Gill, Henderson-Sellers & Niazi, 2018). Kriittisiä tietojärjestelmiä kehittäessä ei voida sivuuttaa tiettyjä hallinnollisia vaatimuksia ja vastuullisuuden näkökulmaa. Tällöin kehitystyötä ei voida perustaa pelkästään löyhälle opille ketteryydestä, vaan tarvitaan jonkinlainen viitekehys kehitystoiminnalle. (Ebert & Paasivaara, 2017.)

Kuten todettua, päätavoite ketterän kehityksen omaksumisessa on se, että fokus pidetään koko ajan arvon tuottamisessa asiakkaalle (taulukko 2, "nopea arvon tuottaminen"). Jokainen iteraatio tuottaa ideaalisesti julkaisun, jossa on vastattu muuttuneeseen tilanteeseen. Toinen merkittävä tavoite on jatkokehityksen mahdollistaminen. Ketterässä ajattelutavassa ei oleteta, että järjestelmän toimintaympäristö tai sille asetetut vaatimukset pysyvät muuttumattomana. Täten on huolehdittava siitä, että järjestelmän arkkitehtuuri mahdollistaa ja kestää muutokset (taulukko 2, "arkkitehtuuri"). Koska järjestelmää on oletettavasti muokattava jälkikäteen, dokumentaatio on tärkeä seikka ketterässä kehityksessä vaikkei se olekaan ykkösprioriteetti. (Douglass, 2015.) Järjestelmän jatkokehittäjät eivät useinkaan ole samoja henkilöitä kuin sen alun perin kehittäneet (Sommerville, 2016), joten dokumentaatio on merkittävässä osassa järjestelmää ylläpidettäessä.

Ketterästä kehittämisestä on lukuisia hyötyjä. Koska palautetta saadaan jo aikaisessa vaiheessa ja jatkuvasti, puutteiden havaitseminen on helpompaa. Samasta syystä puutteiden korjaaminen on edullisempaa. Sijoitetuille resursseille saadaan nopeasti konkreettisia tuloksia, mikä nostaa sidosryhmien tyytyväisyyttä. Projektit ovat hallittavampia, koska saavutettuja tuloksia peilataan projektin tavoitteisiin, ei niinkään suunnitelman mukaiseen etenemiseen. Näin pysytään paremmin kosketuksissa siihen, kuinka lähellä tavoitteiden toteutu-

mista ollaan. Lisäksi muutoksiin reagoiminen on nopeampaa. Muutokset ovat lähtöoletus, eivät epätoivottuja yllätyksiä. (Douglass, 2015.)

Ketterien menetelmien suosio kertoo osaltaan siitä, että niistä todella on etua modernissa ohjelmistokehityksessä. 12. vuosittaiseen ketterän kehityksen tila –raporttiin (engl. *12th Annual State of Agile Report*) osallistuneista yrityksistä 97 % käytti ketteriä menetelmiä ainakin osittain (VersionOne, 2018). Täysin ongelmatonta niiden soveltaminen ei kuitenkaan ole.

2.3 Ketterän ohjelmistokehityksen haasteet

Ketterässä ohjelmistokehityksessä muutos on paitsi käyttövoima, myös haaste. Misra, Kumar & Kumar (2010) tunnistavat neljä kategoriaa, joissa ketteryys vaatii muutoksen: organisaation kulttuuri, johtamistapa, tiedonhallintastrategiat ja kehittämismenetelmät. Haasteet eivät koske vain ketteryyden omaksumista, vaan niitä esiintyy myös toimijoilla, jotka mieltävät jo toimivansa ketterästi (Conboy Coyle, Xiaofeng & Pikkarainen, 2011).

Iivari ja Huisman (2007) toteavat, että organisaatiokulttuurin voidaan katsoa sisältävän kaiken, mitä organisaatio on. Kulttuurilla on eri tasoja. On olemassa syviä, tiedostamattomia oletuksia, jotka kulttuurin omaksuneet ottavat itsestäänselvyytenä. Arvot ja uskomukset edustavat sanallistettua ideaalia siitä, mitä pitäisi tehdä ja pintatasolla ovat aistittavissa olevat kulttuurin luomat käyttäytymismallit. Kulttuurin sisällä voi olla ja todennäköisesti onkin keskenään ristiriidassa olevia elementtejä.

Ohjelmistokehityksessä panostettiin perinteisesti pysyvyyteen hiomalla prosesseja. Tämä pyrkimys vakauteen on toki läsnä monissa organisaatioissa laajemminkin. Prosessien noudattaminen on usein syvällä organisaation kulttuurissa ja on huomattava, että jonkin tietyn ketterän menetelmän käyttöönotto ei sinänsä muuta toimintaa ketteräksi, jos taustalla vaikuttavat edelleen vanhat prosessit ja tapa ajatella niiden kautta. (Nerur ym., 2005; Moe, Aurum & Dybå, 2012.)

Ketterää ohjelmistokehitystä tukeva organisaatiokulttuuri vaatii painottumista jatkuvaan yhteistyöhön ja asiakaskeskeisyyteen. Luottamusta tulisi vaalia ja ilmapiirin olla virheet salliva (Misra ym., 2010). Tämä johtaa siihen, että ohjelmistokehityksen johtamista ja päätöksentekoa tulee tarkastella uudella tavalla.

Koska suunnittelupainotteinen ohjelmistokehitys tukee kontrolloivaa johtamistapaa ja ketterä kehittäminen taas perustuu monilta osin yksilöiden ja tiimien itseohjautuvuuteen (Overhage ym., 2011), johtamiskulttuurin muuttaminen ketteryyttä tukevaksi on laaja-alaista ja vaatii sekin enemmän kuin pelkän uuden kehitysmenetelmän käyttöönottoa. Haaste on myös poliittinen: projektipäällikön on luovuttava osasta valtaansa, jotta tiimi voi aidosti toimia itseohjautuvasti. (Nerur ym., 2005.)

Agile Manifeston arvot eli yksilöt ja yksilöiden välinen vuorovaikutus, toimiva ohjelmisto, asiakasyhteistyö ja muutoksiin reagoiminen (Beck ym., 2001)

ovat sinällään positiivisia asioita ja niillä on usein katsottu olevan päätöksentekoa tehostava vaikutus. Ohjelmistojen kehittäminen ketterästi saattaa kuitenkin tietyiltä osin myös heikentää päätöksentekoa. (Drury-Grogan, Conboy, & Acton, 2017.) McAvoy ja Butler (2009) näkevät keskeisenä erityisesti yksilöiden korostamisen prosessien kustannuksella, sillä kehitystiimin jäsenet muodostavat päätöksenteon ytimen ketterässä ohjelmistokehityksessä.

Yksilön korostaminen prosessien sijaan synnyttää tilanteita, joissa edellisten iteraatioiden ongelmia ratkotaan uudelleen. Päätösten laatua ei useinkaan mitata johdonmukaisesti, vaan esimerkiksi asiakkaan ja kehitystiimin vapaamuotoisen palautteen perusteella. Tehtävien jaossa paino on helposti kokemuksella – yksilölle jaetaan ne tehtävät, joita hän on tehnyt aiemminkin. Tällä toivotaan saavutettavan tehokkuutta. Kokemuksen painottumisen vuoksi kokeneemmat kehitystiimin jäsenet saavat ylikorostuneen roolin päätöksenteossa. (taulukko 3; Drury-Grogan ym., 2017.) Moe ym. (2012) kuvaavat ilmiötä teknokratiaksi, jossa tekninen osaaminen on myös auktoriteetin lähde. Kun tiimin jäsen on tarpeeksi osaava, hänellä on mahdollisuus ajaa päätöksiä läpi itsenäisesti perustuen kokemukseensa ja oletuksiinsa.

Vaikka yksilön korostaminen on osaltaan ongelmallista (McAvoy & Butler, 2009), saavuttaakseen toivottuja tuloksia ketterät ohjelmistokehitysmenetelmät vaativat joka tapauksessa erittäin osaavia ohjelmistokehittäjiä (Boehm & Turner, 2004). Tämä asettaa haasteita projekteissa, kun kehittäjältä vaaditaan osaamista niin koodaamisesta, testaamisesta kuin arkkitehtuureistakin. Taitovaatimukset eivät rajoitu vain ohjelmistokehitykseen – ketterästi kehittävän ryhmän jäsenen on oltava myös hyvä kommunikoimaan niin kanssakehittäjien kuin asiakkaankin suuntaan. Lisäksi vaaditaan liiketoimintaosaamista tai vähintäänkin ymmärrystä liiketoiminnasta. Kehittäjiltä vaaditaan osaamista paitsi laajasti, myös syvällisesti. Kehittäjien yksilöllisen paineen lisäksi yritykset kohtaavat rekrytointiongelmia, kun tarpeeksi osaavia kehittäjiä ei löydy tarpeeksi. Yrityksessä jo työskentelevien kehittäjien kouluttaminenkin on aiempaa vaikeampaa. (Conboy ym., 2011.)

TAULUKKO 3 Päätöksenteon ominaispiirteet ketterässä ohjelmistokehityksessä (Drury-Grogan ym., 2017, s. 255 mukaan)

Ketterän kehityksen arvot	Yksilö ennen prosesseja	Toimiva ohjelmisto ennen dokumentaatiota	Asiakasyhteistyö ennen sopimusten tekemistä	Muutoksiin reagoiminen ennen suunnittelua
Arvon ominaispiirteet	Päätökset toistavat samoja ongelmia; kokeneet kehittäjät hallitsevat päätöksentekoa	Päätöksistä ei viestitä hyvin eikä niitä dokumentoida	Asiakkaan tietämys ajaa päätöksentekoa; yhteydenpito vain tiettyyn asiakkaan edustajaan rajoittaa päätöksenteon näkökulmia	Päätökset tehdään lyhytkestoista hyötyä silmällä pitäen; reagointinopeus rajoittaa keskustelua monimutkaisista ongelmista

Toimivan ohjelmiston tuottaminen dokumentaation sijaan aiheuttaa päätöksiä, jotka tehdään huonosti viestien ja päätöksistä ei välttämättä jää dokumentaatiota (Drury-Grogan ym., 2017). Liiaksi ryhmän oma-aloitteiseen, verbaaliseen kommunikaatioon luottaminen saattaa johtaa ryhmäajatteluun (engl. *groupthink*). Tällöin yksilöt tekevät päätöksiä kuviteltua konsensusta myötäillen eikä käsillä olevaa ongelmaa ratkaisten. (McAvoy & Butler, 2009.) Lisäksi dokumentaation karsiminen saattaa aiheuttaa sen, että päätöksiä tehdään ilman täyttä käsitystä siitä, miten päätös vaikuttaa kehitteillä olevaan toiminnallisuuteen. Dataa voi hävitä ja päätöksiä unohtua, jos niistä ei jää dokumentaatiota. Puutteellisen seurannan vuoksi on mahdollista, että kehitystiimin jäsenillä on eri käsitykset siitä, mitä on jo tehty ja mitä tulee tehdä seuraavaksi. (taulukko 3; Drury-Grogan ym., 2017.)

Asiakasyhteistyön korostaminen sopimusten teon sijaan aiheuttaa sen, että päätöksiä tehdään asiakkaan tai asiakkaan edustajien osaamistason mukaan (Drury-Grogan ym., 2017). Asiakas on keskeinen toimija, koska heiltä saatava palaute ajaa kehitystyötä eteenpäin. (Moe ym., 2012). Asiakas osaa usein ilmaista toiveensa liiketoiminnan näkökulmasta, mutta käsitys teknisestä toteutuksesta voi olla puutteellinen, jolloin yhteistyö on tekniseltä osin liikaa järjestelmän toteuttajan varassa. Kehitystiimi saattaa olla yhteydessä vain yhteen asiakkaan edustajaan ja usein asiakkaan osallistuminen ja palautteenanto on pitkälti asiakkaan itsensä vastuulla. Tästä seuraa, että jos asiakas on passiivinen tai osaamaton, yhteistyö ei toimi kunnolla. (Drury-Grogan ym., 2017.)

Muutoksiin reagoiminen suunnittelun sijaan saa päätöksenteon keskittymään lyhyen aikavälin etuun. Päätöksiä tehdään iteraation sisällä niin, että juuri käsillä olevassa iteraatiossa päästään eteenpäin. (Drury-Grogan ym., 2017.) Päätöksiä on tehtävä yhteistyössä, mutta tiukka julkaisuaikataulu aiheuttaa sen, että samalla päätöksiä on tehtävä nopeasti (Moe ym., 2012). Päätöksien nopeusvaatimus rajoittaa keskustelua monimutkaisten ongelmien kohdalla ja niitä lykätään, koska asiaan ei juuri sillä hetkellä ehditä syventymään, onhan asiakkaalle tuotettava arvoa jatkuvasti (Drury-Grogan ym., 2017). Moen ym. (2012) mukaan ongelmana tällöin on se, että strateginen, taktinen ja operatiivinen päätöksenteko eivät ole keskenään linjassa. Dikertin, Paasivaaran ja Lasseniuksen (2016) sanoin ylempi johto saattaa toimia ”vesiputousmoodissa”, kun operatiivisella tasolla, kehitystiimissä, yritetään olla ketteriä iteraatio kerrallaan.

Ketterässä kehityksessä on yllä kuvatun laisia haasteita ja toisaalta perinteinen, vesiputousmallin mukainen ohjelmistokehitys ei puhtaassa muodossaan ole monestikaan mielekäs lähestymistapa. Molemmilla suuntauksilla on vankat kannattajansa, mutta vastakkainasettelun sijaan on suotavaa, että kukin organisaatio löytää itselleen optimaalisen tavan soveltaa ketteriä tai perinteisiä menetelmiä, sekä näiden menetelmien yhdistelmiä. (Nerur ym., 2005.)

2.4 Ketterän ohjelmistokehityksen haasteet turvallisuuskriittisissä tietojärjestelmissä

Turvallisuuskriittisillä tietojärjestelmillä tarkoitetaan sellaisia järjestelmiä, joiden epäonnistumisella on vakavia seurauksia. Niitä voivat olla ihmisten loukkaantuminen, ympäristön vahingoittuminen tai suuren mittaluokan taloudelliset vahingot. (Sommerville, 2016.) Turvallisuuskriittiset järjestelmät ovat nykyaikaisen yhteiskunnan ytimessä: niitä käytetään niin terveydenhuollossa, energiantuotannossa kuin lentoliikenteessäkin. (Heeager & Nielsen, 2018.)

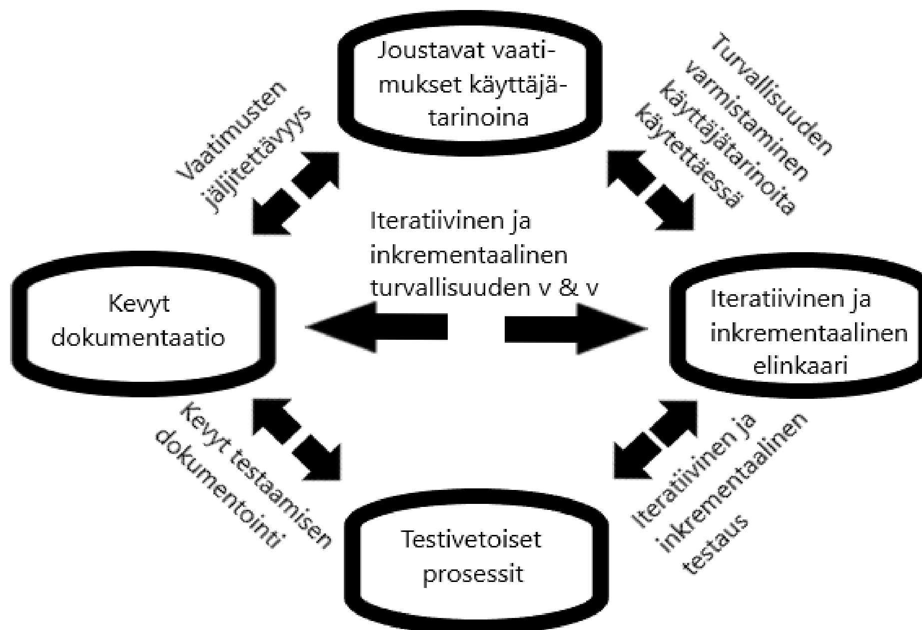
Turvallisuuskriittisiä tietojärjestelmiä on usein kehitetty vesiputousmallin mukaisesti ketterän kehityksen voittokulusta huolimatta (Heeager & Nielsen, 2018). Tämä johtuu siitä, että niiden kohdalla on erityisen tärkeää turvata korkea varmistamistaso, jonka Boehm (2002) näkee yhtenä vesiputousmallin vahvuuksista. Suurelta osin kyse ei edes ole valinnasta, sillä turvallisuuskriittisten tietojärjestelmien kehittämistä säätelevät useat standardit ja prosessimallit (Chrissis, Konrad & Shrum, 2003). Näiden lisäksi lainsäädäntö, yleinen kiinnostus ja mahdollinen ennakoiva huoli potilaiden, kuluttajien tai kansalaisten taholta luo omanlaisensa rajoitteen kehitystyölle (Heeager & Nielsen, 2018.)

Jotta turvallisuuskriittinen järjestelmä hyväksytään, se on verifioitava ja validoitava. Verifioinnilla tarkoitetaan sen varmistamista, että järjestelmä kehitetään oikealla tavalla, jotta se on turvallinen. Validointi on sitä, että järjestelmä vastaa sitä tarkoitusta, jota varten se on luotu. (Beznosov, 2003.)

Edellä mainitut seikat johtavat siihen, että ketterää ajattelutapaa on huomattavasti haastavampi toteuttaa turvallisuuskriittisissä hankkeissa kuin pienissä, epämuodollisemmissa kehitysprojekteissa (Heeager & Nielsen, 2018). Alun perinhän ketterä ajatusmalli ilmensi ideaalia juuri pienien kehitystiimien työskentelyä (Dingsøyr ym., 2012). Tämä ei silti ole estänyt ketteryyden leviämistä paitsi kooltaan yhä isompiin hankkeisiin, myös turvallisuuskriittisten tietojärjestelmien kehittämiseen (Heeager & Nielsen, 2018).

Heeagerin (2012) mukaan nämä kaksi on mahdollista sovittaa yhteen ja ketteryydestä on saavutettavissa merkittäviä hyötyjä myös turvallisuuskriittisillä alueilla. On kuitenkin selvää, että turvallisuuskriittisyys vaatii tietyssä määrin prosesseja ja sääntelyä eivätkä ketterän kehityksen arvot voi sellaisenaan toteutua tällaisissa kehityshankkeissa (Cockburn, 2006).

Heeager ja Nielsen (2018) esittävät neljä ongelma-aluetta, jotka liittyvät ketterien kehitysmenetelmien käyttöön turvallisuuskriittisissä tietojärjestelmähankkeissa: kevyt dokumentaatio, joustavat vaatimukset kirjoitettuna käyttäjätarinoiksi, iteratiivinen ja inkrementaalinen elinkaari, sekä testivetoiset prosessit. Nämä ongelma-alueet kytkeytyvät toisiinsa viidellä vaikutussuhteella, jotka ovat vaatimuksien jäljitettävyys, turvallisuuden varmistaminen, kun käytetään joustavia vaatimuksia, iteratiivinen ja inkrementaalinen turvallisuuden verifiointi ja validointi, iteratiivinen ja inkrementaalinen testaus, sekä kevyt testaamisen dokumentointi (kuvio 4).



KUVIO 4 Ketterän ohjelmistokehityksen ongelma-alueet ja niiden vaikutussuhteet turvallisuuskriittisessä kontekstissa (Heeagerin ja Nielsenin, 2018, s. 28 mukaan)

Agile Manifestossa todetaan: ”toimiva ohjelmisto ennen dokumentaatiota” (Beck ym, 2001). Ketterä arvomaailma ja siitä johdetut kehitysmenetelmät ovat taipuvaisia dokumentaation minimalisointiin. Niiden tarkoitus, kuten kaiken muunkin toiminnan, on arvon tuottaminen asiakkaalle. (Cockburn, 2006.) Turvallisuuskriittisiä tietojärjestelmiä ei kuitenkaan ole lain, standardien tai pakollisten prosessien vuoksi mahdollista kehittää ilman kattavaa dokumentointia (Heeager & Nielsen, 2018), koska sillä on keskinen rooli laadunvarmistuksessa (Conboy, 2009) ja sisäinen valvonta ja säätelytoimet luovat paineen todistaa dokumentoinnilla, että kehitystoiminnassa on menetelty oikein (Cockburn, 2006).

Turvallisuuskriittisille tietojärjestelmille on ominaista, että niiden kehittäminen kestää useita vuosia ja niiden käyttöikä voidaan mitata jopa vuosikymmenissä (Drobka, Noftz & Raghu, 2004). Tällöin on todennäköistä, että kehitysprojektin kuluessa tapahtuu henkilöstövaihdoksia ja että järjestelmää jatkokehittää ja ylläpitää kolmas osapuoli (Sommerville, 2016). Dokumentoinnilla osaltaan varmistetaan järjestelmän turvallisuus. Näistä seikoista johtuen kevyt dokumentointi on yksi keskeisistä ongelma-alueista ketterien kehitysmenetelmien soveltamisessa turvallisuuskriittiseen kontekstiin. (Heeager & Nielsen, 2018.)

Toinen ongelma-alue on joustavat vaatimukset, jotka ilmaistaan käyttäjätarinoina (kuvio 4). Käyttäjätarinat ovat yleinen, löyhästi jäsennelty tapa esittää vaatimuksia ja sitä suositaan ketterissä kehitysmenetelmissä, koska ne antavat joustavuutta ja niitä on helppo muokata vaatimusten muuttuessa (Ramesh ym.,

2007). Vaatimusten muuttuminen on turvallisuuskriittisten tietojärjestelmien kohdalla ongelma, koska muutoksilla saattaa olla vakavia seurauksia järjestelmän arkkitehtuuriin (Boehm & Turner, 2004).

Turvallisuuskriittiset tietojärjestelmät eivät kuitenkaan ole immuuneja muutoksille siinä missä muutkaan järjestelmät ja varsinkin toiminnalliset vaatimukset usein muuttuvat kehityshankkeen aikana (Heeager & Nielsen, 2018). Sääntelyn vuoksi pelkät luonnollisella kielellä muodostetut käyttäjätarinat eivät turvallisuuskriittisessä kontekstissa kuitenkaan riitä, koska niillä ei voida todeta, että järjestelmä vastaa tarkoitustaan. Järjestelmää ei siis voida validoida käyttäjätarinoilla. (Beznosov, 2003.)

Kolmantena ongelma-alueena Heeager ja Nielsen (2018) näkevät iteratiivisen ja inkrementaalisen elinkaaren, joka on ketterille kehitysprojekteille tyypillinen. Tällaisella elinkaarella tarkoitetaan sitä, että jokainen iteraatio, esimerkiksi Scrum-menetelmän sprintti, tuottaa testatun ja toimintakykyisen inkrementin, joka on suoraan asiakkaan käytettävissä (Cockburn, 2006). Turvallisuuskriittisiä tietojärjestelmiä ei perinteisesti ole kehitetty iteratiivisella ja inkrementaalisella elinkaarella (Heeager & Nielsen, 2018), vaan esimerkiksi v-mallin mukaisesti, joka on eräs vesiputousmallin sovelluksista. Se soveltuu turvallisuuskriittisten tietojärjestelmien kehittämiseen, koska siinä keskitytään laadunvarmistukseen monipuolisen testauksen avulla. (de Sousa Santos, de Castro Andrade, Rocha, Matalonga, de Oliveira & Travassos, 2017.)

Iteratiivisuus ja inkrementaalisuus on kuitenkin ketterän kehittämisen keskeisiä piirteitä, joten niitä ei voida kokonaan hylätä, jos turvallisuuskriittisiä tietojärjestelmiä halutaan kehittää ketterästi (Heeager & Nielsen, 2018). Suunnitteluorientoituneita kehittämismenetelmiä on tällöin muokattava iteratiiviseen ja inkrementaaliseen suuntaan, mutta tämä on usein haastavaa (Beck, 2000). Heeager ja Nielsen (2018) suosittelevat iteraatioiden pidentämistä, jolloin kehittäjillä on aikaa keskittyä turvallisuuden kannalta olennaisiin seikkoihin, kuten riittävään dokumentaatioon.

Neljäs ongelma-alue on testivetoiset prosessit. Ketterässä ohjelmistokehityksessä kehitystyö tapahtuu usein testivetoisesti käyttäen hyödyksi automaattista testausta. Esimerkiksi Extreme Programming -menetelmässä testit kirjoitetaan ennen toteutusta ja inkrementti katsotaan valmiiksi, kun se läpäisee testit. (Beck, 2000.) Jos turvallisuuskriittisiä tietojärjestelmiä kehitetään vaikkapa v-mallin mukaisesti, testaus tapahtuu kunkin vaiheen lopussa. (de Sousa Santos ym., 2017).

Edellä mainitun ristiriidan yhteensovittaminen on haastavaa, koska turvallisuuskriittisten tietojärjestelmien kehittämiseen liittyvä sääntely ulottuu myös testaukseen. Esimerkiksi testin kirjoittajan ja varsinaisen koodaajan tulee olla eri henkilö. Tietyt prosessivaatimukset edellyttävät ylimääräistä testaamista. Testauksen sääntely tuottaa paljon sellaista työtä, joka vähentää mahdollisuuksia ketterien kehitysmenetelmien käyttöön. (Heeager & Nielsen, 2018.)

Edellä mainittujen neljän ongelma-alueen lisäksi Heeager ja Nielsen (2018) tunnistavat niiden välillä viisi vaikutussuhdetta (kuvio 4), joista yksi on vaatimusten jäljitettävyys silloin, kun dokumentoidaan vain kevyesti. Ketterissä ke-

hitysmenetelmissä vaatimusten muuttuminen on tavallista ja muuttuminen tekee jäljitettävyydestä hankalampaa. Jäljitettävyyttä on mahdollista hallita dokumentoinnilla, mutta vaatimusten muuttuessa myös dokumentteja on muokattava. Niiden muokkaaminen tapahtuu turvallisuuskriittisessä kontekstissa säänneltyjen menettelytapojen kautta (Conboy, 2009). Koska ketterän kehityksen arvot korostavat toimivaa ohjelmistoa ennen dokumentaatiota, arvoista johdetut kehitysmenetelmät eivät usein ota dokumentaatiota huomioon siinä määrin, että vaatimusten jäljitettävyys voitaisiin turvata (Heeager & Nielsen, 2018).

Toinen vaikutussuhde on joustavien ja muuttuvien vaatimusten suhde iteratiiviseen ja inkrementaaliseen elinkaareen (Heeager & Nielsen, 2018; kuvio 4). Turvallisuuskriittisessä kehitystyössä myös vaatimuksiin liittyy sääntelyä ja useat prosessistandardit edellyttävät, että vaatimukset on määritelty ennen suunnittelua ja käyttöönottoa (Conboy, 2009). Iteratiivisuus vaatimusten hallinnassa saattaa rajoittaa niiden näkemistä riittävän laaja-alaisesti. Lisäksi vaatimuksia ei tiheään iteroidessa välttämättä tarkastella tarpeeksi perusteellisesti turvallisuuden näkökulmasta. (Cawley, Wang & Richardson, 2010.)

Kolmas vaikutussuhde on iteratiivisen ja inkrementaalisen työtavan vaikutus testaukseen (Heeager & Nielsen, 2018; kuvio 4). Turvallisuuskriittisten tietojärjestelmien kohdalla jokaisen inkrementin täytyy olla täysin toimiva, testattu ja validoitu ennen julkaisua (Fitzgerald, Stol, O'Sullivan & O'Brien, 2013). Tämän vuoksi iteratiivinen ja inkrementaalinen elinkaari vaatii testauksen sisällyttämistä jo aikaiseen vaiheeseen kehitystyötä. Aikainen testaaminen ja testauksen kattavuus tuo lisätyötä iteraatioon, jolloin todennäköisemmin tarvitaan pidempiä iteraatioita, kuin mitä ketterissä kehitysmenetelmissä yleisesti suositaan. Turvallisuuskriittiset tietojärjestelmät ovat myös usein sulautettuja johonkin laitteeseen. Laitetta ei voida rakentaa inkrementaalisesti kuten ohjelmistoa, joten sulautetut järjestelmät ovat haastavia iteratiivisen ja inkrementaalisen elinkaaren kannalta. (Heeager & Nielsen, 2018.) Elinkaari saattaa vaatia myös kehittäjiltä omaksumista: aiemmin v-mallin mukaisesti toimineet kehittäjät eivät välttämättä sisäistä sitä, että testausta on tehtävä etupainotteisesti ja sitä ei voi lykätä iteraation loppuun (Conboy, 2009).

Neljänneksi testaus ja kevyt dokumentointi kytkeytyvät toisiinsa. Testauksella määritetään, milloin tietty ratkaisu on riittävän laadukas ja turvallinen (Heeager & Nielsen, 2018). Perusteellinen testaus turvallisuuskriittisessä kontekstissa tuottaa suuren määrän testidokumentaatiota (Cockburn 2006). Testausta joudutaan dokumentoimaan paljon perusteellisemmin, mihin ketterät kehitysmenetelmät yleisesti ottaen ohjaavat (Heeager & Nielsen, 2018).

Viides vaikutussuhde on iteratiivinen ja inkrementaalinen turvallisuuden verifiointi ja validointi, jonka Heeager ja Nielsen (2018) näkevät haastavimpana edellä mainituista vaikutussuhteista. Turvallisuuden verifioinnin ja validoinnin todisteena toimii dokumentaatio ja kun se yhdistetään iteratiiviseen ja inkrementaaliseen elinkaariajatteluun, tuloksena on se, että dokumentointia pitää ylläpitää ja päivittää jokaisen iteraation yhteydessä (Heeager & Nielsen, 2018). Tämä on haastavaa muutoksenhallinnan kannalta. Useimpaan iteraatioon sisäl-

tyy muutoksia ja ne saattavat vaarantaa aiemmissa iteraatioissa jo validoituja järjestelmän osia. Iteratiivisella ja inkrementaalisella toiminnalla pyritään nopeaan arvon tuottamiseen asiakkaalle. Tällainen elinkaariajattelu saattaa kuitenkin olla kallista turvallisuuskriittisessä kontekstissa, kun jokainen iteraatio ja sen tuomat muutokset tulee dokumentoinnin avulla verifioida ja validoida. (Cordeiro, Barreto, Barcelos, Oliveira, Lucena, & Maciel, 2007.)

3 TUTKIMUKSEN TOTEUTUS

Tässä luvussa esitellään tutkimuksen kulku. Alaluvussa 3.1. esitellään tutkimusote ja tutkimusmenetelmä ja niiden valikoitumisen perustelut. Alaluvussa 3.2. esitellään aineiston valintaan ja sen rajaukseen liittyvät perustelut, jonka lisäksi esitellään analyysin kulku.

3.1 Tutkimusote ja tutkimusmenetelmä

Tämän tutkimuksen tavoite on luoda ymmärrystä ilmiöön, josta aiempaa tietoa on vähän eli turvallisuuskriittisten tietojärjestelmien ja ketterän ohjelmistokehityksen välistä suhdetta. Tämän vuoksi tutkimus toteutettiin laadullisena. Ketterä ohjelmistokehitys itsessään ei ole aiheena uusi ja siitä on kasvavissa määrin tutkimusta myös isoja kehitysprojekteja koskien (Dingsøyr & Moe, 2013), mutta turvallisuuskriittisessä kontekstissa sitä on tutkittu melko vähän ainakaan siinä määrin, että ratkaisuja tutkimuksissa nousseisiin ongelmiin olisi esitetty (Heeager & Nielsen, 2018). Laadullinen tutkimusote soveltuu sellaisten aiheiden tutkimiseen, joista ei vielä ole syvällistä ymmärrystä (Hirsjärvi, Remes & Sajavaara 2004).

Tutkimusmenetelmänä käytettiin teoriaohjaavaa sisällönanalyysiä. Aineiston analyysin annettiin tietoisesti perustua ketterän ohjelmistokehityksen arvoihin ja periaatteisiin, jotka on esitelty tutkimuksen luvussa 2. Aineiston jäsentelyssä hyödynnettiin ohjelmistokehityksessä vakiintuneita käsitteitä, erityisesti Roycen (1970) vesiputousmallissa esittämiä ohjelmistokehityksen vaiheita. Toisaalta aineisto ohjasi sitä, minkälaista teoriaa ketterän ohjelmistokehityksen haasteista valikoitui tarkasteltavaksi. Tutkimuksen analyysi on tuotettu abduktiivisella päättelyllä. Näin voidaan toimia, kun tutkimusta ohjaavat sekä aineisto että valmiit mallit (Sarajärvi & Tuomi, 2018).

Sisällönanalyysi voi Sarajärven ja Tuomen (2018) mukaan sisältää sisällön erittelyn ja varsinaiseen laadullisen sisällönanalyysiin. Tässä tutkimuksessa käytettiin molempia, koska sisällön erittelyllä voidaan kvantifioida aineistoa

(Sarajärvi & Tuomi, 2018) ja haluttiin selvittää, vaikuttaako jokin tietty ohjelmistokehityksen vaihe esiintyvän muita yleisemmin aineistossa.

Tutkimusmenetelmän valinnan perusteena on tutkimuskysymys: "voidaanko ketterän ohjelmistokehityksen suuntaus yhdistää turvallisuuskriittisten tietojärjestelmien epäonnistumisiin?" Koska jo tutkimuksen asettelu ohjaa ohjelmistokehityksen nykyparadigman tarkasteluun, ajateltiin, ettei aineistoa ole järkevää analysoida puhtaan aineistolähtöisesti. Näin siksi, että aineistona käytettiin turvallisuuskriittisten tietojärjestelmien epäonnistumisista kertovaa uutisointia ja niistä ei ollut mahdollista erotella esimerkiksi yhteyttä yksittäiseen ohjelmistokehitysmenetelmään. Analysoimalla aineistoa niin, että analyysia ohjasi ketterän ohjelmistokehityksen arvojen ja periaatteiden muodostama löyhä viitekehys, pystyttiin paremmin vastaamaan tutkimuskysymykseen.

Koska aineisto ei tarjonnut yksityiskohtaista tietoa epäonnistumisten syistä, tutkimuksen näkökulmaksi valikoitui tarkastella ylätasolla koko ohjelmistokehityksen nykysuuntausta ja sen ilmenemistä tietojärjestelmien epäonnistumisissa. Tutkimuksen tarkoituksena ei ole osoittaa, että jokin tietty kehitysmenetelmä olisi johtanut tiettyyn epäonnistumiseen eikä se tällä tutkimusasetelmalla olisi edes mahdollista. Sen sijaan haluttiin tutkia, onko ketterän ohjelmistokehityksen arvomaailma – muutoksiin reagoiminen ja nopea arvon tuottaminen – löytänyt tiensä myös turvallisuuskriittisten tietojärjestelmien kehittämiseen.

3.2 Aineiston rajaus ja analysointi

Koska tutkimuksessa halutaan ymmärtää tietojärjestelmien epäonnistumista yhteiskunnan kannalta kriittisten tietojärjestelmien kontekstissa, valittiin aineistoksi yhteiskunnallisesti merkittävien uutisten tuottajien (Helsingin Sanomat ja YLE) uutisartikkelit. Sommervillen (2016) mukaan tietojärjestelmä on turvallisuuskriittinen silloin, kun sen epäonnistuminen tuottaa henkilövahinkoja, ympäristön vahingoittumista tai mittavia taloudellisia vahinkoja. Aineiston valinnassa tehtiin oletus, että tämän kaltaiset epäonnistumiset päätyvät uutisiin.

Haku rajattiin Helsingin sanomien ja YLE:n uutisarkistoihin. Tällä rajauksella saatiin riittävän kattava aineisto, koska aineistoa kerätessä havaittiin, että tietyn tietojärjestelmän epäonnistumisesta kerrottiin Helsingin Sanomissa ja YLE:llä hyvin saman tapaisesti. Sen vuoksi oletettiin, että pienempien uutisoijien artikkelit eivät olisi enää tuoneet lisäarvoa tutkimukselle. Aineistoa haettiin vain Suomen kielellä resurssisyistä. Haun kasvattaminen myös kansainvälisiin uutisartikkeleihin olisi kasvattanut aineiston määrää huomattavasti ja tuonut lisäksi haasteita aineiston analysointiin eri kielien takia.

Aineistoa kerättiin lähes koko tutkimuksen ajan. Uutisartikkeleiden haku on toteutettu aikaväliltä 1.1.2017 – 10.4.2020. Tässä tutkimuksessa keskitytään ohjelmistokehityksen nykytilaan, joten aineistoa ei ollut mielekästä kerätä kovin kaukaa menneisyydestä. Valitulla aikajänteellä aineisto pysyi ajallisesti relevanttina ja aineiston koko hallittavana. Uutisartikkeleita kertyi aineistoon yhteensä 172 kappaletta.

Aineiston haussa käytettiin hakusanoja ”tietojärjestelmä” ja ”ohjelmisto”. Hakusanat olivat tarkoituksella laajoja, jotta tutkimuksen kannalta relevantteja artikkeleita ei jäisi pois aineistosta. Uutiset ovat Helsingin Sanomissa ja YLE:llä suunnattu suurelle yleisölle, joten tarkoilla ammattitermeillä hakemisen ei katsottu olevan mielekästä.

Saadut hakutulokset suodatettiin ensin otsikoiden perusteella. Otsikon liittyessä tutkimuksen aihepiiriin, uutisartikkeli silmäiltiin läpi. Jos tällöin paljastui, että tietojärjestelmän epäonnistumien on tapahtunut tutkimuksen aihepiiriin liittymättömien seikkojen vuoksi (esimerkiksi sähköverkkovika tai tietoturvan puutteet), artikkeli ei päätynyt aineistoon. Myöskään lyhyet, luonteeltaan toteavat uutiset karsittiin pois, koska niistä ei ollut analysoitavissa järjestelmän epäonnistumiseen johtaneita syitä.

Aineistoon päätyneet uutisartikkelit jaoteltiin ensin tapauksen mukaan. On huomattava, että tämä tutkimus ei ole tapaustutkimus eli siinä ei yritetä ymmärtää juuri tietyn tapauksen taustalla olevia tekijöitä, vaan ilmiötä yleisesti. Tästä huolimatta aineiston jaottelu ensin tapauksiin oli mielekäs aineiston hahmottamisen kannalta. Artikkelit järjestettiin tapauksittain kronologisesti julkaisupäivän mukaan. Artikkeleiden määrän perusteella eriteltiin viisi tapausta ja lisäksi kuudes muut-kategoria, johon sisältyvät artikkelit tapahtumista, joista löytyi vain vähän uutisointia (alle 10 artikkelia).

Tapauksista muodostettiin erilliset tiedostot, johon tapaukseen liittyvät artikkelit siirrettiin sellaisenaan. Aineistosta eriteltiin sisältöä kvantifioimalla ne kerrat, jolloin aineistossa toistuu ohjelmistokehityksen vaiheisiin viittaavat sanat *vaatimusmäärittely, suunnittelu, toteutus, testaus* ja *ylläpito*. Tarkkojen sanamuotojen lisäksi käytettiin katkaistuja sanoja taivutusmuotojen vuoksi. Lisäksi käytettiin vaihtoehtoisia sanoja, koska uutisoinnissa ei välttämättä käytetä ohjelmistokehityksen ammattisanastoa (Tamai, 2009). *Vaatimusmäärittelyn* lisäksi käytettiin sanaa *vaatimus. Toteutuksen* kohdalla käytettiin myös sanaa *ohjelmistovirhe*. Ylläpidon kohdalla haettiin myös sanalla *päivitys*. Tulokset ristiintaulukoitiin ja ne esitellään tarkemmin luvussa 4.

On huomattava, ettei tutkimuksessa tehty kvantifiointi ole verrattavissa määrällisen tutkimuksen tilastollisiin menetelmiin, koska jo kvantifioinnissa aineistoa analysoitiin yhdistämällä samaa asiaa tarkoittavia hakusanoja ja tulkitsemalla sanan kontekstia. Esimerkiksi sana *vaatimus* ei aina viitannut aineistossa nimenomaan vaatimusmäärittelyyn. *Suunnittelu* taas ei aina viitannut suoraan järjestelmän suunnitteluun, vaan esimerkiksi suunnitelmaan yleisesti tehdä jotakin. Kvantifiointi on laadullisessa tutkimuksessa pikemminkin keino havainnollistaa aineistoa (Sarajärvi & Tuomi, 2018).

Kvantifioinnin jälkeen tehtiin varsinainen, laadullinen sisällönanalyysi. Tapauskohtaisesta tarkastelusta luovuttiin ja aineisto pelkistettiin niin, että jäljelle jäivät tutkimusongelman kannalta oleelliset osat. Tämän jälkeen aineisto ryhmiteltiin luomalla siitä alaluokkia ja edelleen abstrahoitiin muodostamalla alaluokista yläkäsitteitä niin, että aineisto voidaan johtaa esitettyihin tutkimuskysymyksiin (taulukko 4).

TAULUKKO 4 Sisällönanalyysin prosessi

Alaluokka	**Yläkäsite**	**Pääluokka**	**Yhdistävä luokka**
Järjestelmän vaatimusmäärittely/suunnittelu	Ohjelmistokehityksen perustehtävät	Ohjelmistokehityksen perustehtävät	Turvallisuuskriittisten tietojärjestelmien havaitut haasteet ja niidenyhteys ketterään ohjelmistokehitykseen
Toteutus/ohjelmistovirheet			
Testaus			
Ylläpito/päivitys			
Käyttäjien palaute			
Päätöksenteko	Ketterän ohjelmistokehityksen haasteet	Ketterän ohjelmistokehityksen yhteys turvallisuuskriittisiin tietojärjestelmiin	
Keskeneräisyys			
Lainsäädäntö/viranomaiset	Turvallisuuskriittinen konteksti		
Organisaatioiden ulostulot julkisuuteen			

Prosessi noudattaa Sarajärven ja Tuomen (2018) näkemystä aineistolähtöisestä sisällönanalyysistä. Alaluokat osittain ja niistä johdetut yläkäsitteet kokonaisuudessaan on johdettu luvussa 2 esitetystä teoriasta, jolloin analyysi ei ole puhtaan aineistolähtöinen, vaan teoriaohjaava (Sarajärvi & Tuomi, 2018).

4 TUTKIMUSTULOKSET

Tässä luvussa käydään läpi tutkimuksen kannalta tärkeimmät aineistosta nousseet asiat. Ensin esitellään lyhyesti tapaukset, joita ensimmäisellä analyysikierroksella nousi esiin. Kuten todettua, tämä tutkimus ei ole tapaustutkimus, mutta tapaukset auttavat jäsentämään tuloksia. Alaluvussa 4.1. esitetään tulokset eli tietojärjestelmien ongelmat ohjelmistokehityksen perustehtävien näkökulmasta. Alaluvussa 4.2. esitellään tulokset niiltä osin, kuin niistä on nähtävissä yhteys ketterän ohjelmistokehityksen suuntaukseen ja turvallisuuskriittiseen kontekstiin.

Erillisiä tapauksia oli aineistosta eroteltavissa kaikkiaan viisi: Apotti, Boeing, HUS, Lifecare ja Oriola. Tapausten ulkopuolelle jäävä aineisto muodosti muut-kategorian. Tapausta ei eroteltu omaksi tapauksekseen, jos aineistosta nousi sitä koskien alle 10 artikkelia. Viisi tapausta esitellään seuraavaksi. Viittaustekniikasta luvuissa 4 ja 5 todettakoon, että tiettyyn uutisartikkeliin viitataan hakasuluilla niin, että niiden sisässä on numero, jonka mukaiseen artikkeliin viitataan. Tutkimuksessa käytetyt uutisartikkelit löytyvät numeroituna liitteestä 1.

4.1 Tapausten esittely

Apotti on sairaalaympäristöön suunniteltu tietojärjestelmä, joka otettiin asteittaan käyttöön eri Uudenmaan sairaaloissa. Järjestelmään budjetoitiin noin 600 miljoonaa euroa [6]. Ensimmäisenä se otettiin käyttöön Peijaksen sairaalassa vuonna 2018 [1,2]. Järjestelmän toimittaja oli yhdysvaltalainen Epic, mutta Apottia muokattiin Suomen oloihin sopivaksi [4]. Apotti aiheutti laajaa tyytymättömyyttä hoitajien ja lääkärien tahoilta [3,4] muun muassa siksi, että sitä oli kankea käyttää ja raportteihin tulostui paljon turhaa tietoa. Järjestelmän puutteet aiheuttivat myös yhden kuolemantapauksen [16,27] ja alensivat sairaaloiden työtehoa merkittävästi [4].

Boeingin toimittamille 737 Max -koneille sattui kaksi lentoturmaa, jotka olivat keskenään niin samanlaisia, että syyksi epäiltiin järjestelmävirhettä. Tämä epäily osoittautui oikeaksi [34,37]. Lentoturmien syynä oli sakkauksenestojärjestelmän virheellinen toiminta. Järjestelmä pakotti koneen keulaa alas lentäjien toiminnasta huolimatta [40]. Yhdysvaltain ilmailuviranomainen FAA käynnisti asiasta tutkinnan, jonka edetessä ilmeni useita puutteita järjestelmän testauksessa ja jopa suoranaista testausprosessin välttelyä [73]. Kahdessa lentoturmassa kuoli yhteensä 346 ihmistä.

Yli 20 Helsingin ja Uudenmaan sairaanhoitopiirin (HUS) järjestelmää eivät toimineet kahteen päivään marraskuussa 2017. Esimerkiksi potilastietojärjestelmien ja automatisoitu lääkkeiden jakaminen eivät toimineet [105]. Tänä aikana tietoja jouduttiin kirjaamaan paperille, josta aiheutui huomattavia viiveitä sairaanhoitopiirin sairaaloiden toiminnalle. Esimerkiksi leikkauksia jouduttiin siirtämään [93]. Ongelman syynä oli linjakortin rikkoutuminen ja samaan aikaan tapahtunut odottamaton häiriö laitteen ohjelmistossa [92]. Häiriön syytä aineistosta ei käynyt ilmi.

Lifecare on Tieto Oyj:n toimittama potilastietojärjestelmä, jossa on ollut lukuisia ongelmia keväästä 2018 lähtien erityisesti Päijät-Hämeen hyvinvointiyhtymässä [138]. Järjestelmässä on havaittu suunnitteluvirheitä, jotka ilmenevät käytettävyyden ongelmina [133] ja ovat esimerkiksi johtaneet tilanteisiin, joissa potilaalle on määrätty väärää lääkettä [128]. Tieto on julkaissut Lifecaren uusia versioita, mutta järjestelmien käyttäjien näkökulmasta hitaalla aikataululla. Lisäksi päivityksissä ei ole saatu kaikkia havaittuja puutteita korjattua [141]. Lifecare on ollut käytössä myös Helsingin kaupungin hammashoidossa, jossa se on aiheuttanut kasvavia jonoja ajanvaraukseen ja ongelmia laskutuksessa ja palkanmaksussa [137].

Oriolan tapauksessa kyse oli uudesta toiminnanohjausjärjestelmästä, jonka käyttöönotossa oli haasteita. Syyskuussa 2017 tapahtunut käyttöönotto epäonnistui siinä määrin, että vain 72 prosenttia lääkkeistä pystyttiin toimittamaan apteekkeihin normaalisti [158]. Tilanne oli uhka potilasturvallisuudelle, koska tietyt lääkkeet uhkasivat loppua apteekkien varastoista [152]. Ongelma kesti usean viikon ajan ja Oriolaa kritisoitiin paitsi puutteellisesta lääkejakelusta, myös huonosta viestinnästä ongelmatilanteessa. Oriola halusi myös salata tarkastusraportin liikesalaisuuteen vedoten [158].

Näiden viiden tapauksen lisäksi aineistosta muodostettiin muut-kategoria, joka käsittää ne tapaukset, joista löytyi alle 10 uutisartikkelia. Vaikka ne eivät muodosta omia tapauksiaan, ne käsiteltiin samoin analysoiden kuin eritellyt viisi tapaustakin ja niihin viitataan seuraavissa alaluvuissa 4.2 ja 4.3, sekä pohdintaosiossa luvussa 5.

4.2 Tietojärjestelmien havaitut ongelmat

Tutkimuksessa kävi ilmi, että ohjelmistokehityksen eri tehtävät ilmenivät aineistossa suhteellisen tiheään sanamuodossa. Maininnat kuitenkin jakautuivat

epätasaisesti niin, että perinteisesti alkuvaiheen tehtäviksi mielletyt vaatimusmäärittely ja suunnittelu ilmenivät huomattavasti harvemmin, kuin testaus ja ylläpito (taulukko 5). On huomattava, että sanojen esiintymistiheyden tarkastelusta ei tutkimuksessa pyritty eikä niistä voitukaan tehdä johtopäätöksiä aineistosta valikoituneiden tietojärjestelmien todellisista ongelmista.

Aineistoa sisällöllisesti analysoitaessa kuitenkin havaittiin, että suoria mainintoja järjestelmien epäonnistumisten syistä löytyi selkeämmin testauksesta ja ylläpidosta. Vaatimusmäärittelyn tai suunnittelun kohdalla tulokset vaativat enemmän tulkintaa. Varsinkin vaatimusmäärittelystä suoria ilmauksia oli hyvin vähän, jos ollenkaan. Vaatimusmäärittely ilmeni aineistossa lähinnä epäsuorina viittauksina lainsäädännön tai säädöksien asettamiin järjestelmän reunaehtoihin. Suunnittelun virheistä aineistossa oli enemmän viittauksia, mutta ne olivat harvoin suoria. Toki on huomioitava, että suunnittelun virheet todennäköisesti nivoutuvat vaatimusmäärittelyyn (Sommerville, 2016).

TAULUKKO 5 Suorat sanalliset viittaukset ohjelmistokehityksen tehtäviin tapauksittain

Tapaukset	**Artikkelit kpl**	**Vaatimusmäärittely**	**Suunnittelu**	**Toteutus (ohjelmistovirhe)**	**Testaus**	**Ylläpito (päivitys)**
Apotti	30	4	4	2	6	3
Boeing	45	2	4	8	29	69
HUS	19	-	2	5	5	10
Lifecare	21	1	4	5	22	19
Oriola	12	3	3	1	1	-
Muut	45	-	2	6	10	24
Yhteensä	**172**	**10**	**19**	**27**	**73**	**125**

Apotin tapauksessa mainittiin, että järjestelmä toimi suhteellisen häiriöttömästi, mutta ohjelmiston logiikka ja käytettävyys olivat huonolla tasolla [14]. Jos sama käyttäjä käytti järjestelmää eri rooleissa, näkymä saattoi olla kullakin kerralla täysin erilainen. Joissain tapauksissa potilasta siirrettäessä siirtoraportti antoi paljon turhaa tietoa, joka hidasti oleellisten asioiden löytämistä [14,16]. Käyttöoikeuksia oli joillain käyttäjillä tietoihin, joilla heillä ei työnsä puolesta ollut oikeutta [9,10]. Pohlin ja Ruppin (2011) mukaan käyttäjien ja heidän vaatimustensa huomiotta jättäminen aiheuttaa usein sen, että järjestelmään halutaan muutoksia. Näin oli Apotin tapauksessa, sillä aineistosta nousi runsaasti käyttäjien tyytymättömyyttä ilmaisevia katkelmia. Esimerkiksi lääkärit kokivat, että työ hidastui ja tiedon etsiminen hankaloitui [18,21].

Myös Lifecaren tapauksessa oli viitteitä siitä, ettei käyttäjien vaatimuksia oltu loppuun asti kuultu. Rullahiiren käyttö saattoi vaihtaa aiemmin valittujen kenttien tietoja, jolloin potilaalle valikoitui väärä laboratoriolähete, ellei järjestelmää käyttänyt lääkäri huomannut rullahiiren vaikutusta valikon käyttäytymiseen. Lisäksi järjestelmässä oli näytön skaalautuvuusongelma, jossa kaksi

samanaikaista näkymää saattoi piilottaa osan potilastiedoista [133]. Boeingin lentoturmissa lentäjät toimivat manuaalin mukaisesti, mutta eivät silti saaneet konetta hallintaan [47].

Apotin suunnittelussa oli aineiston mukaan viitteitä siitä, että järjestelmän arkkitehtuuri ei ollut täysin toimiva. Potilastietojen siirrossa Apotista Kanta-palveluun oli vaikeuksia ja joissain tapauksissa lääkärit joutuivat siirtämään tietoja käsin vanhasta Uranus-järjestelmästä Apottiin [8]. Joissain muissa aineiston järjestelmissä (esim. Lifecare ja Jobiili) nousi esiin se, ettei niiden suunnittelussa oltu otettu huomioon suurta käyttäjämäärää [109,139].

Eniten mainintoja ohjelmistovirheistä oli Boeingin tapauksessa. Lentoturmia tapahtui kaksi ja niissä oli paljon samankaltaisuuksia. Turmakoneissa käytetyt sakkauksenestojärjestelmät aktivoituivat väärällä hetkellä ja pakottivat koneiden keulaa alas [47]. Myöhemmin löytyi vielä muitakin puutteita liittyen lentokoneiden hallintajärjestelmiin. Boeing pyrki korjaamaan puutteita koodaamalla ohjelmistoja uudelleen [50]. Boeing joutui myös palauttamaan avaruuskapselin maahan ohjelmistovikojen takia. Aineiston mukaan "Boeing käy läpi miljoona riviä koodia löytääkseen ohjelmiston virheet" [74]. Avaruuskapselilla ei ollut yhteyttä lentoturmiin.

Lifecaren tapauksessa potilaan lääkkeet saattoivat vaihtua toisiin, kun reseptiä uusittiin. Järjestelmäntoimittajan mukaan taustalla on ollut ohjelmistovirhe [128]. Muussa aineistossa ohjelmistovirhe mainittiin, kun verottaja oli lähettänyt asiakkaille verotuspäätöksiä, joissa oli osittain toisten ihmisten tietoja. Taustalla oli ohjelmiston bugi [172]. Itä-Suomen yliopistoon hyväksyttiin aiottuun nähden kaksinkertainen määrä opiskelijoita, koska järjestelmä sallinut tehdä ylimääräisiä varauksia [109].

Apotin testauksesta mainittiin, että sen harjoitusympäristö tuli käyttöön vain kuukautta ennen käyttöönottoa [4]. Tämä vaikuttaa lyhyeltä ajalta suuren tietojärjestelmän käyttäjätestaukselle. Ennen päätymistä juuri Apottiin, sitä testattiin käytettävyystesteillä, joissa "käytiin läpi pieni määrä tyypillisiä ammattilaisten työtehtäviä" [28]. Vantaan kaupunki lykkäsi Apotin käyttöönottoa usealla kuukaudella, koska järjestelmän testausta jatkettiin [6].

Boeingin testauksen osalta aineistosta nousi se, että testauksen päätösvaltaa uusien lentokoneiden hyväksymisestä on siirretty viime vuosina valmistajille. Osa lentoturmien koneissa olevista ohjelmistoista oli Boeingin itsensä hyväksymiä. Aineiston mukaan Boeingin koneita on ennen Boeing 737 Max 8 -konetyyppiä testattu kielteisessä ilmapiirissä, jossa viranomaiset "eivät uskaltaneet kertoa havainnoistaan koston pelossa" [39]. Lentoturmien jälkeen Boeing teki satoja testilentoja, jotta lentokieltoon asetettu konetyyppi saataisiin taas käyttöön [72].

Lifecaren käyttöönotossa ilmeni ongelmia, joita testauksessa ei havaittu. Testauksissa oli havaittu vain pikkuvikoja, mutta todellisessa tilanteessa suuri käyttäjämäärä vaikutti järjestelmän toimintaan ennakoimattomalla tavalla [131]. Oriolan tapauksessa järjestelmää testattiin useita kuukausia, mutta se ei silti toiminut odotetulla tavalla [150]. Vanha totuus siitä, että onnistunut testaus ei kerro onnistuneesta järjestelmästä (Dijkstra, 1972) pitänee yhä paikkansa.

Ylläpidosta tai erilaisista päivityksistä oli kaikkein eniten mainintoja aineistossa. Jos ylläpito mainittiin suoraan, se liittyi usein ylläpitokustannuksiin. Esimerkiksi Apotin tapauksessa järjestelmän tarpeellisuutta perusteltiin säästöillä ylläpitokustannuksissa [3]. Maininta päivityksistä oli usein reagoimista havaittuun ongelmaan.

Aineistossa oli kuitenkin myös tilanteita, joissa päivitys johti ongelmaan. Hätäkeskuksissa käytössä oleva Erica-järjestelmä lakkasi ajoittain toimimasta eri päivitysten yhteydessä. ”Ongelmat johtuivat päivityksistä valtion turvallisuusverkossa, jonka päällä Erica toimii” [82]. Lifecarea päivitettiin, jonka yhteydessä hävisi 270 hoitolähetettä [145]. Jyväskylän yliopiston sähköposti lakkasi toimimasta viallisen tietoturvapäivityksen takia [118]. Kanta-palvelussa oli ongelmia huoltotöiden vuoksi ja siksi, että oli siirrytty pilvipalveluihin [121].

HUS:n tapaus ei suoraan liittynyt päivitykseen, mutta kylläkin ylläpitovaiheessa olevaan järjestelmään: ”keskeisessä osassa järjestelmää meni rikki linjakortti, ja samaan aikaan tapahtui odottamaton häiriö laitteen ohjelmistossa.” [92] Tilanteessa myös varajärjestelmä petti. Myöhemmin kahdella tietokantapalvelimella oli yhteysongelma, jonka seurauksena potilastietojärjestelmä kaatui. Tilanteella ei ollut yhteyttä linjakortin rikkoutumiseen, mutta myös tässä tilanteessa varajärjestelmäkin toimi epätäydellisesti.

4.3 Ongelmien linkittyminen ketterään ohjelmistokehitykseen turvallisuuskriittisessä kontekstissa

Tiedetään, että ketterä ohjelmistokehitys on iteratiivista ja inkrementaalista. (Beck, 2000). Kerätyssä aineistossa on useita viitteitä siitä, että järjestelmiä on otettu käyttöön keskeneräisinä.

Apotin tapauksessa kyse vaikuttaa olleen osin tietoisesta valinnasta. Järjestelmä otettiin käyttöön vaiheittain, jotta ensimmäisiltä käyttäjiltä saatu palaute voitiin ottaa huomioon myöhemmissä käyttöönotoissa [3]. Ketterässä ohjelmistokehityksessä kehitystyö tapahtuu tiiviissä yhteistyössä asiakkaan kanssa ja saatu palaute on tärkeässä roolissa jatkokehityksessä (Boehm, 2002; Nerur ym., 2005). Aineistossa mainitaan mm. näin: ”—saamme vielä lisää palautetta sieltä ja pystymme tietämään, miten seuraavaksi pitää edetä kehitystyössä”. ”Emme odottaneetkaan, että se (Apotti) heti käyttöönoton jälkeen saisi 5/5 arvion.” [20] ”- - parannuksia järjestelmään tehdään jatkuvasti. Hänen mukaansa on ollut tiedossa, ettei järjestelmä ole kerralla valmis” [20].

Boeingin tapaukseen liittyi jopa aktiivista yritystä vältellä järjestelmän simulaattorikoulutusta, jota voidaan pitää järjestelmän testauksena. Boeing julkaisi viestejä liittyen ohjelmiston kehittämiseen, joissa todettiin mm.: ”Haluan painottaa, kuinka tärkeää on pysyä tiukkana sen kanssa, ettei tulossa ole minkäänlaista vaatimusta simulaattorikoulutuksesta - -” [73]. Lisäksi ilmailuviranomaiset ovat jo ennen lentoturmia tienneet konetyypin yllättävästä käyttäytymisestä: ”- - sertifikaatissa Boeing 737 Max todetaan turvalliseksi, kunhan pilo-

teille tehdään koulutuksessa selväksi epätavalliset tilanteet - -" [44]. Yhtiön silloinen toimitusjohtaja tiesi ohjelmiston puutteista ennen toista lentoturmaa ja "vuonna 2016 yhtiölle työskennellyt lentäjä oli lähettänyt kollegalleen viestin, jossa hän kertoi olevansa huolissaan 737 Max -koneen MCAS-automaatiojärjestelmästä" [72].

Lifecaren tapauksessa Valvirassa oli epäilyjä siitä, että järjestelmä on otettu käyttöön keskeneräisenä: "Varmasti on testattu, mutta jokin asia on jäänyt huomiotta. Kyllähän se siltä vaikuttaa, että vähän keskeneräisenä tämä on päässyt tuotantoon - -" [132]. "Tässä tapauksessa näkyy mielestäni potilastietojärjestelmän käytettävyystestauksen niukkuus." [133]

Hätäkeskuksen Erica-järjestelmästä todettiin, ettei se ollut kaikilta osin valmis käyttöön otettavaksi. Järjestelmä yliarvioi hälytystehtävien riskit, jolloin pelastustehtävään lähetettiin ylimitoitetusti henkilökuntaa ja kalustoa. "Järjestelmä on rakennettu yhteistyössä viranomaisten kanssa, eikä se ole vielä täydellinen." [79] Lisäksi aineistossa korostetaan käyttäjien palautetta: "Ensihoitopäällikkö Hagström korostaa, ettei järjestelmään kannata tehdä suuria muutoksia, ennen kuin saadaan kerättyä kokemuksia." [81]

Asiakkaalla on ketterässä kehittämisessä kriittinen rooli, koska asiakkaalta saatu palaute ohjaa kehitystyötä (Nerur ym., 2005) ja asiakkaan oma osallistuminen ja osaamistaso vaikuttaa siihen, kuinka tarkasti vaatimuksia vastaava ohjelmisto tulee olemaan (Drury-Grogan ym., 2017). Aineistosta nousi esiin joitakin viitteitä siitä, ettei asiakasyhteistyö ole ollut täysin onnistunutta.

Apotin tapauksessa "Apottia kohtaan oli paljon odotuksia ja toiveita, mutta nyt päällimmäisenä tunteena on pettymys" [4]. Apottia verrattiin Tanskassa käyttöön otettuun, hyvin samankaltaiseen järjestelmään, jonka kohdalla "hankinta perustui naiiveihin ja ylioptimistisiin arvauksiin" [14]. Apottia hankkiessa "arvioitiin, mikä kuudesta kisassa mukana olevasta järjestelmästä olisi paras. Ei, onko mikään tarpeeksi hyvä" [28].

Muusta aineistosta nousi kaksi järjestelmää, Espoon kaupungin Tierasap ja Kajaanin kaupungin talousjärjestelmä, joiden käyttöönotto ei edes toteutunut puutteellisen valmistelun vuoksi. Espoo ehti käyttää järjestelmän suunnitteluun 6-7 miljoonaa euroa, mutta "järjestelmä on todettu Espoon kaupungin tarpeisiin sopimattomaksi." [84] Kajaanin kaupungin saama arvio järjestelmän kustannuksista oli reilusti alakanttiin, "eikä kaupunki osannut kyseenalaistaa tarjousta". [120] "Melkoinen asiantuntija täytyisi poliitikon olla, että tietää kaikki kustannukset mitä seuraa" [120].

Tietojärjestelmien turvallisuuskriittisyys näkyy siinä, että aineistossa on runsaasti mainintoja lainsäädäntöön ja erilaisiin viranomaistahoihin. Ne luovat reunaehdot turvallisuuskriittisten tietojärjestelmien kehitykselle Chrissis, Konrad & Shrum, 2003; Heeager & Nielsen, 2018).

Apotin tapaus kytkeytyi tuolloin eduskunnassa käsittelyssä olleeseen valinnanvapauslakiin, jonka mahdollisuutena oli, että Apotin käyttäjäryhmä muuttuisi. Alun perin oli suunniteltu, että käyttäjiä ovat perusterveydenhuolto, erikoissairaanhoito ja sosiaalitoimi. "Apottia ei käytetä perusterveydenhuollossa, jos valinnanvapauslaki menee eduskunnassa läpi nykyisessä muodossaan."

[1] ”Nyt sitten yksi kolmasosa jää tämän ulkopuolelle, niin onhan se melko suuri toiminnallinen ongelma.” [1] Liian laajoihin käyttöoikeuksiin liittyi siihenkin lainsäädännöllisiä seikkoja. ”- - päätös Apotin käyttöönotosta tehtiin, vaikka oli tiedossa, ettei järjestelmä ole lakien mukainen.” [9]

Boeingin tapauksessa Yhdysvaltojen ilmailuhallinto FAA:lla oli keskeinen rooli sen valvonnassa, että Boeing toteuttaa vaadittavat korjaustoimenpiteet Boeing 737 Max -konetyypin ohjelmistoon. FAA:n rooli vaikuttaa olleen kuitenkin melko reaktiivinen eli lentoturmien jälkeen asetettiin vaatimuksia ja valvottiin toimintaa. Konetyyppiä hyväksyttäessä ”ilmailuhallinto tukeutui turvallisuusarviossaan pitkälti lentokonevalmistaja Boeingilta saamiinsa varhaisiin arvioihin.” [61] Toisaalta ”Boeing ei myöskään tuonut esille, että sakkauksenestojärjestelmän vialla saattaisi olla tuhoisat seuraukset.” [61]

Kuten aiemmin tuotiin esiin, Boeing on itse ollut testaamassa ohjelmistoa ja ilmailuhallinnon rooli on ollut valvoa toimintaa. Syynä ovat resurssikysymykset: ”järjestelyn tarkoituksena on ollut vapauttaa FAA:n voimavaroja kaikkein tärkeimpien ja monimutkaisimpina pidettyjen turvallisuusuhkien käsittelyyn.” [61]

Viranomaisten rooli on ollut näkyvää myös HUS:n tapauksessa, jossa onnettomuustutkintakeskus päätyi antamaan suosituksia jatkotoimenpiteistä: ”terveydenhuollon toimijoilla pitäisi olla selkeä huolto- ja päivitysohjelma tärkeimmille tietojärjestelmille.” [105] Viranomaiset ottivat kantaa tapahtumiin myös Lifecaren (Valvira) ja Oriolan (Fimea) tapauksissa.

Valviran toimien rajallisuus oli edustettuna: ”- - sitten on tietysti järjestelmän sulkeminen, mutta se on vihonviimeinen keino. Siinä pitäisi silloin harkita, kumpi on suurempi riski: kieltää järjestelmän käyttö vai jatkaa käyttöä.” [132] Toisaalta haluttiin tiukempaakin puuttumista: ”jos Valvira painostaisi, se vaikuttaisi potilastietojärjestelmiin ja toisaalta myös terveydenhuollon organisaatioihin niin, että osattaisiin vaatia järjestelmätoimittajalta enemmän.” [135]

Fimea ja Oriola kiistelivät siitä, voiko tarkastuskertomuksen julkaista, sillä Oriola halusi salata sen liikesalaisuuteen vedoten. Fimean mukaan ”ei ole käytännössä mahdollista, että koko tarkastuskertomus olisi kokonaan liikesalaisuutta”. [155] Tarkastuskertomus päätyi lopulta julkaistavaksi.

5 POHDINTA

Tässä luvussa esitellään luvussa 4 todettujen tutkimustulosten johtopäätökset ja liitetään ne luvussa 2 esitettyyn teoriapohjaan. Alaluvussa 5.2 tarkastellaan tutkimuksen luotettavuutta.

5.1 Tulosten johtopäätökset

Tulokset koottiin kahta tutkimuskysymystä silmällä pitäen: voidaanko ketterän ohjelmistokehityksen suuntaus yhdistää turvallisuuskriittisten tietojärjestelmien epäonnistumisiin ja millaisia ohjelmistokehityksen ongelmia tällaisten tietojärjestelmien kohdalla on? Ongelmien jaottelu perustui Roycen (1970) vesiputousmallissa esitettyihin ohjelmistokehityksen vaiheisiin.

Tämä jaottelu oli hieman keinotekoinen, sillä tuloksista oli jo vähäisellä analyysillä nähtävissä, että ongelmat nivoutuvat lähes aina useampaan ohjelmistokehityksen tehtävään. Tutkimuksen tarkoitus ei kuitenkaan ollut jaotella ongelmia ohjelmistokehityksen vaiheisiin, vaan vesiputousmalli toimi aineiston jäsentäjänä ja keinona hahmottaa aineistoa.

Kun tarkasteltiin suunnittelun ongelmia, järjestelmän loppukäyttäjien kommenteista oli tulkittavissa, että myös vaatimusmäärittelyssä oltiin joiltain osin epäonnistuttu. Kuten Sommerville (2016) toteaa, vaatimusmäärittely ja suunnittelu sekoittuvat käytännössä aina toisiinsa. Tutkimus ei kiistä tätä näkemystä.

Aineistossa oli yleisesti ottaen melko vähän suoria tulkinnan mahdollisuuksia suunnitteluvirheisiin. Tässä on huomioitava käytetyn aineiston olemus. Uutisissa ei välttämättä voida tai haluta paneutua asioihin niin syvällisesti, kuin tutkimuksen kannalta olisi toivottavaa ja osaltaan kyse voi olla toimittajien osaamisen puutteesta tai järjestelmätoimittajan haluttomuudesta avata epäonnistumisen perimmäisiä syitä (Tamai, 2009).

Tulosten kvantifioinnin ja sisällönanalyysin perusteella varsinaisia syitä tietojärjestelmien epäonnistumisille oli runsaammin tarjolla testauksesta ja yllä-

pidosta. Tämä ei kuitenkaan suoraan tarkoita, että vain näissä asioissa olisi epäonnistuttu.

Kehitystestauksen epäonnistumisesta oli pääteltävissä, että myös toteutuksessa oli tapahtunut virhe. Käyttäjätestauksien epäonnistumisissa virhe oli voinut tapahtua jo vaatimusmäärittelyssä. Boehmin (1981) mukaan merkittävä osa epäonnistumisista voidaankin johtaa vaatimusmäärittelyyn. Ohjelmistopäivityksen yhteydessä tapahtuva virhe voi kertoa siitä, ettei järjestelmän arkkitehtuuria ole suunniteltu muuttuvia vaatimuksia silmällä pitäen. Tosin Boehmin (2002) mukaan ketterässä ohjelmistokehityksessä arkkitehtuuri pyritäänkin luomaan vain sillä hetkellä olemassa oleville vaatimuksille.

Ketterän ohjelmistokehityksen suuntaus näkyi tuloksissa erilaisin, epäsuorin tavoin. Koska ketterästä kehitystavasta on lukuisia hyötyjä (Douglass, 2015) ja sitä sovelletaan kasvavissa määrin myös suuriin tietojärjestelmähankkeisiin (Dingsøyr & Moe, 2013), voidaan ajatella sen olevan vallalla oleva ohjelmistokehityksen paradigma. Vaikka ohjelmistokehityksen menetelmiä ei suoraan käytettäisikään, on oletettavissa, että sen arvot, kuten nopea arvon tuottaminen asiakkaille, ovat houkuttelevia.

Nopean arvon tuottamisen kääntöpuolena on, että useita tutkimuksessa tarkasteltuja tietojärjestelmiä on ilmeisesti otettu käyttöön keskeneräisinä. Vaikkapa Apotin tapauksessa on ajateltu porrastaa käyttöönottoa niin, että järjestelmää testataan ensin pienemmällä käyttäjäryhmällä (tässä tapauksessa Peijaksen sairaala) [3] ja heiltä saadun palautteen perusteella parantaa järjestelmää, jotta se saadaan käyttöön useammassa sairaalassa.

Tällainen toiminta on ketterien arvojen mukaista: saadaan pilottiversio nopeammin kokeiltavaksi tiiviisti asiakkaan kanssa kehitystyötä tehden. Voidaan kuitenkin kyseenalaistaa, onko tällainen toiminta eettisesti kestävää sairaalaympäristössä. Apotin tapauksessa Peijaksen sairaalan työskentely vaikeutui merkittävästi ja liikaa turhaa tietoa sisältänyt siirtoraportti johti yhden potilaan kuolemaan [14]. Heeager ja Nielsen (2018) näkevät turvallisuuskriittisessä kontekstissa ketterän kehittämisen ongelmallisena, koska järjestelmää on vaikea validoida ja verifioida, kun työskennellään iteratiivisesti ja inkrementaalisesti. Apotin kohdalla verifioinnin voidaan tulkita epäonnistuneen.

Nopealla arvon tuottamisella voi olla synkkiä seurauksia, jos se ajaa toimintaa liiallisesti. Boeingin tapauksessa oli viitteitä siitä, että vaadituissa testausprosesseissa oli oiottu jopa siinä määrin, että simulaattoritestejä oli kielletty. Julkaistuissa viesteissä todettiin: ”Haluan painottaa, kuinka tärkeää on pysyä tiukkana sen kanssa, ettei tulossa ole minkäänlaista vaatimusta simulaattorikoulutuksesta - -”. [73] Tässä on viitteitä siitä, että yksittäinen ihminen on saanut liian suuren roolin päätöksenteossa. Agile Manifestossa (Beck ym., 2001) todetaan: ”yksilö ennen prosesseja”. Boeingin tapauksessa yksilö vaikuttaa todellakin ajaneen prosessien ohi. Kyseessä on saattanut olla Moen ym. (2012) kuvailema teknokratia, jossa tietty, osaavaksi mielletty henkilö on sanellut toiminnan kulun. Kyse on myös voinut olla siitä, että henkilön status on ollut muulla tavoin tarpeeksi korkea. Joka tapauksessa kehittäjät eivät ole tilanteessa

puuttuneet asiaan. Kun prosessi oli tapauksessa kyseenalaistettu, yksilöiden väliset valtasuhteet ajoivat niiden yli.

Päätöksentekoa ajetaan ketterän kehityksen mukaisesti paitsi yksilöiden, myös asiakasyhteistyön kautta. Asiakkaan osaaminen määrittää päätöksenteossa käydyn keskustelun tason (Drury-Grogan ym., 2017). Tämä vaikuttaa tutkimuksen perusteella olevan ongelma erityisesti tilanteissa, joissa poliittiset päättäjät päättävät tietojärjestelmistä, joiden loppukäyttäjänä on joku muu taho. Esimerkiksi Kajaanin kaupungin tapauksessa päättäjät eivät osanneet kyseenalaistaa reilusti alakanttiin laskettua kustannusarviota [120]. Espoon kaupungin päättäjät käyttivät 6-7 miljoonaa euroa tietojärjestelmään, jonka todettiin myöhemmin olevan suurilta osin vanhentunut [85]. Kummassakaan tapauksessa järjestelmää ei edes otettu käyttöön. Näissä tapauksissa on tulkittavissa, että on luotettu liikaa asiakkaan osaamiseen päättää kehityshankkeen teknisistä osatekijöistä. Ei todennäköisesti ole ymmärretty, minkälainen järjestelmän tulisi todellisuudessa olla ja mitä tällaisen järjestelmän kehittäminen vaatii. Tällaisissa tapauksissa kehitystyö on liikaa järjestelmäntoimittajan vastuulla (Drury-Grogan ym., 2017).

Viranomaisilla vaikuttaa tutkimuksen mukaan olevan vahva rooli turvallisuuskriittisten tietojärjestelmien kehittämisessä. Juuri lainsäädäntö ja korostunut viranomaisvalvonta erottaakin turvallisuuskriittiset tietojärjestelmät muiden järjestelmien kehittämisestä (Heeager & Nielsen, 2018). Viranomaiset vaikuttavat kuitenkin ottavan korostetun roolin siinä vaiheessa, kun epäonnistuminen on jo tapahtunut. Kyse saattaa olla vain siitä, että viranomaistyö järjestelmiä kehitettäessä ei ole uutisoinnin arvoista ja pääosa työn vaikuttavuudesta tapahtunee lainsäädännön ja säädösten kautta, joita kehityshankkeissa noudatetaan. Voidaan kuitenkin kysyä, luotetaanko järjestelmäntoimittajien ja heidän asiakkaidensa sisäiseen valvontaan liikaa, kun näyttäviä epäonnistumisia tapahtuu myös sellaisten tietojärjestelmien kohdalla, joiden toimintavarmuus tulisi olla huippuluokkaa.

Ainakin Boeingin tapauksessa viranomaisen toiminta vaikuttaa olleen lepsua. FAA oli resurssisyistä antanut Boeingille valtuuksia päättää koneidensa testaajista itse [61]. Tämä lienee osaltaan vaikuttanut siihen, että Boeingilla on voitu oikaista prosesseista, jotta kone on saatu tuotantoon nopeammin ja edullisemmin.

Ketterän kehityksen arvot näkyivät epäsuorasti tutkimukseen valikoituneessa aineistossa. Yksilöt ottivat korostuneen roolin päätöksenteossa, asiakas oli keskeinen toimija, toimivaa ohjelmistoa haluttiin tuotantoon lähes hinnalla millä hyvänsä ja prosesseista tingittiin. Mielenkiintoista on, että arvojen mukaisen toiminnan tulos oli joskus päinvastainen, kuin mitä arvoista johdettuihin ketterän kehityksen periaatteisiin on kirjattu.

Tutkimuksen perusteella ainakin viranomaisen roolia turvallisuuskriittisten tietojärjestelmien kehitystyössä voisi kehittää proaktiivisempaan suuntaan. Yleisesti ottaen toiminnan läpinäkyvyyttä tietojärjestelmähankkeissa tulisi lisätä. Tämä tosin on haastavaa, jos epäonnistumisen yhteydessä jatketaan yleiseltä vaikuttavaa tapaa salata tutkimusraportteja liikesalaisuuksiin vedoten. Tar-

kemmin eri tapauksia voisi tutkia tapaustutkimuksilla, joissa hyödynnettäisiin mahdollisuuksien mukaan tutkimusraportteja ja syvähaastatteluja.

5.2 Tutkimuksen luotettavuus

Tutkimuksen luotettavuuden kannalta keskiössä on valittu aineisto, uutisartikkelit. Tutkimuksessa etsittiin yhteyttä ketterän ohjelmistokehityksen ja uutisoitujen tietojärjestelmien epäonnistumisen välillä. Kerätyn aineiston pohjalta ei voida kuitenkaan suoraan osoittaa, että tarkastelluissa tietojärjestelmissä olisi käytetty ketterän ohjelmistokehityksen menetelmiä.

Tämä luotettavuuden ongelma pyrittiin ratkomaan tutkimuksen perspektiivillä. Tutkimuksessa ei haettu vastausta juuri tietyn tapauksen ongelmiin. Ei myöskään pyritty osoittamaan minkäänlaista korrelaatiota tai kausaliteettia ongelmien yhteydestä ketterään ohjelmistokehitykseen. Korrelaatiota tai kausaliteettia olisi muutenkin vaikea todentaa laadullisella tutkimuksella (Sarajärvi & Tuomi, 2018). Sen sijaan pyrittiin luomaan yleisen tason näkemystä siitä, miten ketterän kehityksen arvot heijastuvat modernissa ohjelmistokehityksessä.

Perspektiivistä huolimatta ketterän ohjelmistokehityksen ja uutisartikkeleista tulkittujen ongelmien välinen yhteys on auttamatta löyhä. Uutisissa ei paljastettu muutamaa poikkeusta lukuun ottamatta epäonnistumisen syitä kovinkaan syvällisesti. Tämän vuoksi aineisto on tietynlainen uhka tutkimuksen luotettavuudelle. Toisaalta voidaan argumentoida, että uutiset ovat autenttinen näkemys tietojärjestelmistä siitä näkökulmasta, josta yhteiskunta on kiinnostunut.

Luotettavuuden ongelmana on myös tutkijan kokemattomuus sisällönanalyysin soveltamisessa ja tutkimustyössä ylipäätään. Kokeneempi tutkija olisi saattanut jäsentää ja tulkita aineistoa tehokkaammin tai eri tavalla, jolloin tutkimuksen tuloskin voisi olla erilainen. Kokemattomuuden ongelmaa pyrittiin minimoimaan noudattamalla Sarajärven ja Tuomen (2018) esittämää sisällönanalyysin prosessia. Prosessi on kuvattu alaluvussa 3.2. Sen tarkoitus on tehdä tutkimuksen toteutus läpinäkyväksi ja näin parantaa tutkimuksen toistettavuutta.

Myös valikoituneen teoriapohjan osalta tutkijan kokemattomuus voidaan nähdä luotettavuusriskinä. Kirjallisuuden arvioimisessa käytettiin metodina lähinnä lähdekritiikkiä. Tutkimusartikkeleiden menetelmäosalle ei tehty erillistä arviointia, vaan luotettiin siihen, että artikkelin ollessa yleisesti arvostetusta lähteestä, myös artikkeli itsessään olisi luotettava. Toisaalta kirjallisuutta haettiin systemaattisesti, kuten tutkimuksen johdannossa on esitetty. Myös aineiston kokoaminen oli systemaattista ja näiden tekijöiden voidaan katsoa nostavan tutkimuksen luotettavuutta.

Tutkimuksen taustalla oli havainto siitä, että kriittisten tietojärjestelmien uutisoinnissa on nähtävissä ketterän ohjelmistokehityksen piirteitä. Tämä lähtöoletus tunnistettiin ja tunnustettiin tutkimusta tehdessä ja sen annettiin ohjata analyysiä siinä missä ohjelmistokehityksen vakiintuneiden teorioidenkin. Sara-

järvi ja Tuomi (2018) toteavat, että laadullinen tutkimus on aina eräänlaista abduktiivista päättelyä ja objektiivinen tieto on tuolloin melko kyseenalainen käsite. Tutkimus ei vastaa ehdottomalla varmuudella siihen, onko ketterä ohjelmistokehitys uutisoitujen epäonnistumisten taustalla. Asetettuihin tutkimuskysymyksiin se sen sijaan vastaa, jolloin voidaan argumentoida tutkimuksen olevan validi.

6 YHTEENVETO

Ohjelmistokehityksen paradigma muuttui merkittävästi tultaessa 2000-luvulle, josta merkkipaaluna on yleisesti käytetty Agile Manifestoa. Vaiheittain etenevän, prosessivetoisen kehitystavan rinnalle tulivat ketterät ohjelmistokehitysmenetelmät, jotka pystyivät vastaamaan siihen paineeseen, jota rytmiltään alati kiihtyvän liike-elämän vaatimukset loivat. Ketterät kehitysmenetelmät antoivat joustavuutta kehitysprojekteihin ja mahdollistivat nopean reagoinnin, jos ja kun projektin olosuhteissa tapahtui muutos.

Ketterät ohjelmistokehitysmenetelmät ovat kuitenkin haastavia, sillä ne vaativat toimiakseen monialaista osaamista kaikilta kehitysryhmän jäseniltä. Tämä luo paitsi yksilöllistä painetta, myös haasteita organisaatioille, joiden on vaikeaa löytää tarpeeksi osaavia kehittäjiä. Toisaalta organisaatiot eivät aina osaa mukautua ketteriin kehitysmenetelmiin, sillä pintapuolinen uuden menetelmän käyttöönotto ei vielä tee ohjelmistotuotannosta ketterää. Jos organisaation johtamiskulttuuri jää kontrolloivaksi ja prosessivetoiseksi, käytännön kehitystyökin noudattaa loppujen lopuksi vallitsevia prosesseja. Johtamisen haasteet näkyvät erityisesti päätöksenteossa. Pahimmillaan ketterän kehityksen periaatteet voivat laskea ohjelmiston laatua välillisesti sen vuoksi, että kyky päätöksentekoon on heikentynyt.

Ketterät ohjelmistokehitysmenetelmät eivät usein sovellu sellaisenaan isokokoisiin projekteihin tai turvallisuuskriittiseen kontekstiin, jossa resurssipaineiden lisäksi reunaehtoja asettavat vielä lainsäädäntö, erilaiset standardit ja julkinen kiinnostus kehitystyötä kohtaan. Julkinen kiinnostus ilmenee esimerkiksi uutisointina, joka turvallisuuskriittisten järjestelmien epäonnistumiseen liittyy.

Turvallisuuskriittisessä kontekstissa prosessit ovat monilta osin välttämättömyys. Niiden tarkoitus on mm. taata järjestelmien riittävä testaus. On monilta osin ongelmallista, jos tällaisissa prosesseissa pyritään oikomaan. Lyhyen tähtäimen etuna saattaa olla nopeampi julkaisutahti, alemmat testauskustannukset ja näiden kautta tyytyväisempi asiakas. Lyhyet pikavoitot muuttuvat kuitenkin nopeasti tappioksi, jos turvallisuuskriittinen tietojärjestelmä pettää.

Nerur ja Balijepally (2007) toteavat ketterän ohjelmistokehityksen olevan alan älyllistä evoluutiota, joka ei ilmiönä poikkea muista aloista. Jokainen tieteenala luo uutta tietoa ja kun reaalimaailma ja uusi tieto yhtenevät riittävästi, saattaa uudesta tiedosta tulla vallitseva suuntaus. Ketterä kehittäminen on tällainen suuntaus ja se ei ole millään muotoa irrallaan reaalimaailmasta. Ketterän kehityksen arvot, kuten yksilöllisyys ja asiakaskeskeisyys vaikuttavat olevan heijastumia yleisestä individualismista ja markkinaehtoisuudesta.

Yhteiskunnan kannalta olisi edullista, että turvallisuuskriittisiä tietojärjestelmiä olisi mahdollista kehittää läpinäkyvästi ja huolellisesti suunnitellen. Tuloksena saattaisi olla vähemmän epäonnistumisia ja vakaampia tietojärjestelmiä, jotka osaltaan luovat vakaan yhteiskunnan.

LÄHTEET

Anderson, D. J. (2003). *Agile management for software engineering: Applying the theory of constraints for business results*. Prentice Hall Professional.

Bass, L., Clements, P. & Kazman, R. (2012). *Software Architecture in Practice*. (3. painos). Pittsburgh: Addison Wesley.

Batra, D., VanderMeer, D., & Dutta, K. (2011). Extending agile principles to larger, dynamic software projects: A theoretical assessment. *Journal of Database Management (JDM)*, *22*(4), 73-92.

Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-wesley professional.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). Manifesto for agile software development.

Beznosov, K. (2003). Extreme security engineering: On employing XP practices to achieve'good enough security'without defining it. In *First ACM Workshop on Business Driven Security Engineering (BizSec)*. Fairfax, VA.

Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

Boehm, B. (2002). Get Ready for Agile Methods, with Care. IEEE Computer, 35(1), 64–69.

Boehm, B., & Turner, R. (2004). Balancing agility and discipline: A guide for the perplexed. Addison-Wesley Professional.

Bosch, J. (2000). *Design and use of Software Architectures: Adopting and Evolving a Product-line Approach*. Great-Britain: Pearson Education.

Cawley, O., Wang, X., & Richardson, I. (2010). Lean/agile software development methodologies in regulated environments–state of the art. In *International Conference on Lean Enterprise Software and Systems* (pp. 31-36). Springer, Berlin, Heidelberg.

Chrissis, M. B., Konrad, M., & Shrum, S. (2003). *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc..

Cockburn, A. (2006). *Agile software development: the cooperative game*. Pearson Education.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. Computer, 34(11), 131-133.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. Information Systems Research, 20(3), 329-354.

Conboy, K., Coyle, S., Xiaofeng, W. & Pikkarainen, M. (2011). People over Process: Key Challenges in Agile Development. Software IEEE, 28(4), 48–57. Los Alamitos: IEEE Computer Society.

Cordeiro, L., Barreto, R., Barcelos, R., Oliveira, M., Lucena, V., & Maciel, P. (2007). TXM: an agile HW/SW development methodology for building medical devices. *ACM SIGSOFT Software Engineering Notes*, *32*(6), 4-es.

Davidsen, M. K., & Krogstie, J. (2010). A longitudinal study of development and maintenance. Information and Software Technology, 52(7), 707-719.

de Sousa Santos, I., de Castro Andrade, R. M., Rocha, L. S., Matalonga, S., de Oliveira, K. M., & Travassos, G. H. (2017). Test case design for context-aware applications: Are we there yet?. *Information and Software Technology*, *88*, 1-16.

Dijkstra, E. W. (1972). The humble programmer. Communications of the ACM, 15(10), 859-866.

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, *119*, 87-108.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. Journal of Systems and Software. Vol. 85, (6). 1213-1221.

Dingsøyr, T., & Moe, N. B. (2013). Research challenges in large-scale agile software development. ACM SIGSOFT Software Engineering Notes, 38(5), 38-39.

Douglass, B. P. (2015). *Agile systems engineering*. Morgan Kaufmann.

Drobka, J., Noftz, D., & Raghu, R. (2004). Piloting XP on four mission-critical projects. *IEEE software*, *21*(6), 70-75.

Drury-Grogan, M. L., Conboy, K., & Acton, T. (2017). Examining decision characteristics & challenges for agile software development. Journal of Systems and Software, 131, 248-265.

Ebert, C., & Paasivaara, M. (2017). Scaling agile. IEEE Software, 34(6), 98-103.

Erickson, J., Lyytinen, K. & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: the state of research. Journal of database Management, 16(4), 88–100.

Fitzgerald, B., Stol, K. J., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 863-872). IEEE.

Gill, A. Q., Henderson-Sellers, B., & Niazi, M. (2018). Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, *20*(2), 315-341.

Glass, R.L. 2001. Frequently Forgotten Fundamental Facts about Software Engineering. IEEE Softw. 18, 3 (May 2001), 112-111.

Glinz, M., & Wieringa, R. J. (2007). Stakeholders in requirements engineering. *IEEE software*, *28*(1), 18-20.

Heeager, L. T. (2012). Introducing agile practices in a documentation-driven software development practice: a case study. *Journal of Information Technology Case and Application Research*, *14*(1), 3-24.

Heeager, L. T., & Nielsen, P. A. (2018). A conceptual model of agile software development in a safety-critical context: A systematic literature review. *Information and Software Technology*, *103*, 22-39.

Hirsjärvi, S., Remes, P. & Sajavaara, P. (2004). *Tutki ja kirjoita*. (10. Osin uudistettu painos). Helsinki: Kustannusosakeyhtiö Tammi.

Hull, E., Jackson, K. & Dick, J. (2011). Requirements Engineering. 3. painos. London: Springer.

IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

Iivari, J., & Huisman, M. (2007). The relationship between organizational culture and the deployment of systems development methodologies. *Mis Quarterly*, 35-58.

ISO/IEC. (2006). Software Engineering – Software Life Cycle Processess – Maintenance. Haettu 8.2.2020 osoitteesta https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1703974&tag=1

Kotonya, G. & Sommerville, I. (1998). Requirements Engineering Processes and Techniques. John Wiley & Sons. Great Britain.

Laanti, M., Similä, J. & Abrahamsson, P. (2013). Definitions of agile software development and agility. Teoksessa F. McCaffery, R. V. O'Connor, R. Messnarz (toim.), Systems, Software and Services Process Improvement, EuroSPI 2013 (s. 247–258). Berliini: Springer-Verlag.

Larman, C. (2004). *Agile and iterative development: a manager's guide*. Addison-Wesley Professional.

Leffingwell, D. (2007). Scaling Software Agility – Best Practises for Large Enterprises. Addison-Wesley.

McAvoy, J., & Butler, T. (2009). The role of project management in ineffective decision making within Agile software development projects. *European Journal of Information Systems*, *18*(4), 372-383.

Misra, S., Kumar, V., & Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management*, 27(4), 451-474.

Moe, N. B., Aurum, A., & Dybå, T. (2012). Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology*, *54*(8), 853-865.

Nelson, R. R. (2007). IT project management: Infamous failures, classic mistakes, and best practices. MIS Quarterly executive, 6(2).

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. Communications of the ACM, 48(5), 72-78.

Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies. Communications of the ACM, 50(3), 79-83.

Overhage, S., Schlauderer, S. & Birkmeier, D. (2011). What Makes IT Personel Adopt Scrum? A Framework of Drivers and Inhibitors to Developer Acceptance. Teoksessa R., H. Sprague, Jr. (toim.), Proceedings of the 44th Annual Hawaii International Conference on System Sciences. Kolua,

Kauai, Hawaii, 4–7 January. (s. 1–10). Los Alamitos: IEEE Computer Society.

Pohl, K. & Rupp, C. (2011). *Requirements engineering fundamentals: A study guide for the Certified Professional for Requirements Engineering exam : foundation level, IREB compliant*. Santa Barbara, CA: Rocky Nook.

Royce, W.W. (1970). Managing the development of large software systems: concepts and techniques. 1970 WESCON technical papers. Vol. 14. 723.

Ramesh, B., Cao, L. & Baskerville, R. (2007). Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal, 20(5), 449–480. Blackwell Publishing ltd.

Sarajärvi, A., & Tuomi, J. (2018). *Laadullinen tutkimus ja sisällönanalyysi: Uudistettu laitos*. Tammi.

Schuh, P. (2004). *Integrating agile development in the real world*. Charles River Media, Inc..

Sommerville I. (2016). Software Engineering 10th Edition (International Computer Science). Pearson Education Limited. Essex, England.

Subramaniam, V., & Hunt, A. (2005). *Practices of an agile developer: Working in the real world*. Pragmatic Bookshelf.

Tamai, T. (2009). Social impact of information system failures. Computer, 42(6).

Templier, M., & Paré, G. (2015). A Framework for Guiding and Evaluating Literature Reviews. Communications of the Association for Information Systems, 37.

VersionOne. (2018). 12th annual state of agile development survey. http://stateofagile.versionone.com. Viitattu: 20.4.2018.

Wiegers, K. & Beatty, J. (2013). Software Requirements. (3. painos). Redmond, Washington: Microsoft Press.

LIITE 1 UUTISARTIKKELIT

Apotti:

[1] https://yle.fi/uutiset/3-10162480 (17.4.2018)
[2] https://yle.fi/uutiset/3-10503807 (12.11.2018)
[3] https://www.hs.fi/kaupunki/art-2000006042847.html (21.3.2019)
[4] https://yle.fi/uutiset/3-10700107 (27.3.2019)
[5] https://yle.fi/uutiset/3-10754572 (25.4.2019)
[6] https://www.hs.fi/kaupunki/art-2000006101890.html (11.5.2019)
[7] https://www.hs.fi/kaupunki/art-2000006133204.html (6.6.2019)
[8] https://yle.fi/uutiset/3-10818476 (6.6.2019)
[9] https://www.hs.fi/kaupunki/art-2000006182751.html (24.7.2019)
[10] https://www.hs.fi/kaupunki/art-2000006183299.html (24.7.2019)
[11] https://www.hs.fi/kaupunki/art-2000006314151.html (19.11.2019)
[12] https://www.hs.fi/kaupunki/art-2000006316375.html (21.11.2019)
[13] https://yle.fi/uutiset/3-11158411 (23.1.2020)
[14] https://www.hs.fi/kaupunki/art-2000006388717.html (30.1.2020)
[15] https://www.hs.fi/kaupunki/art-2000006389630.html (30.1.2020)
[16] https://www.hs.fi/kaupunki/art-2000006391154.html (31.1.2020)
[17] https://yle.fi/uutiset/3-11187969 (1.2.2020)
[18] https://www.hs.fi/kaupunki/art-2000006396374.html (5.2.2020)
[19] https://www.hs.fi/kaupunki/art-2000006420216.html (26.2.2020)
[20] https://yle.fi/uutiset/3-11229349 (28.2.2020)
[21] https://www.hs.fi/kaupunki/art-2000006430604.html (6.3.2020)
[22] https://www.hs.fi/kaupunki/art-2000006442854.html (17.3.2020)
[23] https://yle.fi/uutiset/3-11262294 (17.3.2020)
[24] https://yle.fi/uutiset/3-11261588 (17.3.2020)
[25] https://yle.fi/uutiset/3-11261427 (17.3.2020)
[26] https://www.hs.fi/kaupunki/art-2000006453369.html (26.3.2020)
[27] https://www.hs.fi/kaupunki/art-2000006462316.html (3.4.2020)
[28] https://www.hs.fi/kaupunki/art-2000006463034.html (3.4.2020)
[29] https://www.hs.fi/kaupunki/art-2000006465793.html (6.4.2020)
[30] https://yle.fi/uutiset/3-11295252 (7.4.2020)

Boeing:

[31] https://yle.fi/uutiset/3-10684378 (12.3.2019)
[32] https://yle.fi/uutiset/3-10685022 (12.3.2019)
[33] https://yle.fi/uutiset/3-10685369 (12.3.2019)
[34] https://www.hs.fi/ulkomaat/art-2000006033724.html (13.3.2019)
[35] https://yle.fi/uutiset/3-10689958 (14.3.2019)
[36] https://yle.fi/uutiset/3-10689286 (14.3.2019)

[37] https://www.hs.fi/ulkomaat/art-2000006037131.html (15.3.2019)
[38] https://yle.fi/uutiset/3-10694237 (18.3.2019)
[39] https://www.hs.fi/talous/art-2000006039197.html (18.3.2019)
[40] https://yle.fi/uutiset/3-10699553 (21.3.2019)
[41] https://www.hs.fi/talous/art-2000006046456.html (24.3.2019)
[42] https://www.hs.fi/talous/art-2000006050474.html (27.3.2019)
[43] https://yle.fi/uutiset/3-10712335 (28.3.2019)
[44] https://www.hs.fi/ulkomaat/art-2000006051833.html (29.3.2019)
[45] https://yle.fi/uutiset/3-10717606 (1.4.2019)
[46] https://www.hs.fi/ulkomaat/art-2000006055956.html (2.4.2019)
[47] https://yle.fi/uutiset/3-10721317 (3.4.2019)
[48] https://yle.fi/uutiset/3-10723930 (5.4.2019)
[49] https://yle.fi/uutiset/3-10725830 (5.4.2019)
[50] https://yle.fi/uutiset/3-10726024 (6.4.2019)
[51] https://yle.fi/uutiset/3-10728426 (8.4.2019)
[52] https://yle.fi/uutiset/3-10735271 (12.4.2019)
[53] https://yle.fi/uutiset/3-10738935 (14.4.2019)
[54] https://yle.fi/uutiset/3-10745998 (18.4.2019)
[55] https://www.hs.fi/ulkomaat/art-2000006076262.html (18.4.2019)
[56] https://www.hs.fi/talous/art-2000006083210.html (25.4.2019)
[57] https://www.hs.fi/talous/art-2000006087176.html (28.4.2019)
[58] https://www.hs.fi/talous/art-2000006088154.html (29.4.2019)
[59] https://www.hs.fi/talous/art-2000006108494.html (17.5.2019)
[60] https://www.hs.fi/ulkomaat/art-2000006111002.html (19.5.2019)
[61] https://www.hs.fi/talous/art-2000006115527.html (23.5.2019)
[62] https://yle.fi/uutiset/3-10812588 (2.6.2019)
[63] https://www.hs.fi/ulkomaat/art-2000006143034.html (14.6.2019)
[64] https://yle.fi/uutiset/3-10850162 (26.6.2019)
[65] https://www.hs.fi/ulkomaat/art-2000006155109.html (27.6.2019)
[66] https://www.hs.fi/talous/art-2000006165934.html (7.7.2019)
[67] https://yle.fi/uutiset/3-10878427 (15.7.2019)
[68] https://www.hs.fi/talous/art-2000006174860.html (16.7.2019)
[69] https://www.hs.fi/talous/art-2000006183377.html (24.7.2019)
[70] https://yle.fi/uutiset/3-11145957 (7.1.2020)
[71] https://yle.fi/uutiset/3-11153047 (10.1.2020)
[72] https://www.hs.fi/ulkomaat/art-2000006290439.html (30.10.2019)
[73] https://www.hs.fi/talous/art-2000006368078.html (10.1.2020)
[74] https://www.hs.fi/tiede/art-2000006401220.html (8.2.2020)
[75] https://www.hs.fi/ulkomaat/art-2000006434222.html (10.3.2020)

Erica-järjestelmä:

[76] https://yle.fi/uutiset/3-9446945 (9.2.2017)
[77] https://yle.fi/uutiset/3-10065982 (11.2.2018)
[78] https://yle.fi/uutiset/3-10478874 (27.10.2018)

[79] https://yle.fi/uutiset/3-10643398 (12.2.2019)
[80] https://www.hs.fi/kaupunki/art-2000006117359.html (24.5.2019)
[81] https://yle.fi/uutiset/3-10870655 (10.7.2019)
[82] https://yle.fi/uutiset/3-10973821 (16.9.2019)
[83] https://www.hs.fi/kaupunki/art-2000006286048.html (26.10.2019)

Espoon kaupunki, Tiera:

[84] https://www.hs.fi/kaupunki/art-2000005889405.html (5.11.2018)
[85] https://www.hs.fi/kaupunki/art-2000005894182.html (9.11.2018)
[86] https://yle.fi/uutiset/3-10503875 (12.11.2018)
[87] https://www.hs.fi/kaupunki/art-2000005897204.html (12.11.2018)
[88] https://www.hs.fi/paivanlehti/13112018/art-2000005897067.html (13.11.2018)
[89] https://yle.fi/uutiset/3-10836443 (17.6.2019)

HUS:

[90] https://yle.fi/uutiset/3-9883304 (13.10.2017)
[91] https://yle.fi/uutiset/3-9920602 (7.11.2017)
[92] https://yle.fi/uutiset/3-9920981 (7.11.2017)
[93] https://www.hs.fi/kaupunki/art-2000005439965.html (7.11.2017)
[94] https://www.hs.fi/kotimaa/art-2000005441623.html (8.11.2017)
[95] https://yle.fi/uutiset/3-9922412 (8.11.2017)
[96] https://yle.fi/uutiset/3-9921807 (8.11.2017)
[97] https://www.hs.fi/kaupunki/art-2000005441052.html (8.11.2017)
[98] https://yle.fi/uutiset/3-9924221 (9.11.2017)
[99] https://www.hs.fi/paivanlehti/09112017/art-2000005441750.html (9.11.2017)
[100] https://www.hs.fi/kaupunki/art-2000005442274.html (9.11.2017)
[101] https://www.hs.fi/paakirjoitukset/art-2000005441440.html (9.11.2017)
[102] https://yle.fi/uutiset/3-9950280 (27.11.2017)
[103] https://www.hs.fi/kaupunki/art-2000005619978.html (27.3.2018)
[104] https://yle.fi/uutiset/3-10321878 (2.8.2018)
[105] https://yle.fi/uutiset/3-10591381 (11.1.2019)
[106] https://www.hs.fi/kaupunki/art-2000005962066.html (11.1.2019)
[107] https://yle.fi/uutiset/3-10741765 (15.4.2019)
[108] https://www.hs.fi/kaupunki/art-2000006072935.html (15.4.2019)

Jobiili:

[109] https://www.hs.fi/kotimaa/art-2000005439826.html (7.11.2017)
[110] https://yle.fi/uutiset/3-9919922 (7.11.2017)

[111] https://yle.fi/uutiset/3-9920510 (7.11.2017)
[112] https://yle.fi/uutiset/3-9926249 (10.11.2017)

Junaliikenne:

[113] https://yle.fi/uutiset/3-9417159 (22.1.2017)
[114] https://yle.fi/uutiset/3-9417185 (22.1.2017)
[115] https://www.hs.fi/kotimaa/art-2000005056111.html (22.1.2017)
[116] https://yle.fi/uutiset/3-10111395 (10.3.2018)
[117] https://www.hs.fi/kotimaa/art-2000005599542.html (10.3.2018)

Jyväskylän yliopisto:

[118] https://www.hs.fi/kotimaa/art-2000005408995.html (14.10.2017)

Jyväskylän kotihoito:

[119] https://yle.fi/uutiset/3-10037765 (24.1.2018)

Kajaani:

[120] https://yle.fi/uutiset/3-9671285 (21.6.2017)

Kanta-palvelut:

[121] https://www.hs.fi/kotimaa/art-2000005225435.html (24.5.2017)

Kela:

[122] https://www.hs.fi/kotimaa/art-2000005247834.html (9.6.2017)
[123] https://yle.fi/uutiset/3-10299225 (11.7.2018)
[124] https://yle.fi/uutiset/3-11166986 (20.1.2020)

Lifecare:

[125] https://www.hs.fi/kaupunki/art-2000005573666.html (19.2.2018)
[126] https://yle.fi/uutiset/3-10098433 (2.3.2018)
[127] https://yle.fi/uutiset/3-10099559 (2.3.2018)
[128] https://yle.fi/uutiset/3-10099460 (2.3.2018)
[129] https://yle.fi/uutiset/3-10105601 (7.3.2018)
[130] https://www.hs.fi/kotimaa/art-2000005594758.html (7.3.2018)
[131] https://www.hs.fi/kaupunki/art-2000006032161.html (12.3.2019)
[132] https://yle.fi/uutiset/3-10114548 (14.3.2018)
[133] https://yle.fi/uutiset/3-10117720 (15.3.2018)

[134] https://yle.fi/uutiset/3-10120621 (16.3.2018)
[135] https://yle.fi/uutiset/3-10119960 (17.3.2018)
[136] https://yle.fi/uutiset/3-10122968 (19.3.2018)
[137] https://yle.fi/uutiset/3-10161544 (16.4.2018)
[138] https://yle.fi/uutiset/3-10165266 (18.4.2018)
[139] https://www.hs.fi/kaupunki/art-2000005691289.html (23.5.2018)
[140] https://yle.fi/uutiset/3-10494970 (6.11.2018)
[141] https://www.hs.fi/kaupunki/art-2000005914268.html (28.11.2018)
[142] https://yle.fi/uutiset/3-10587209 (8.1.2019)
[143] https://yle.fi/uutiset/3-11069990 (15.11.2019)
[144] https://yle.fi/uutiset/3-11073817 (18.11.2019)
[145] https://yle.fi/uutiset/3-11184302 (30.1.2020)

Maksuhäiriömerkinnät:

[146] https://yle.fi/uutiset/3-9791022 (22.8.2017)

Oriola:

[147] https://www.hs.fi/kotimaa/art-2000005356324.html (6.9.2017)
[148] https://www.hs.fi/kotimaa/art-2000005356975.html (7.9.2017)
[149] https://www.hs.fi/kotimaa/art-2000005360036.html (8.9.2017)
[150] https://www.hs.fi/talous/art-2000005368163.html (14.9.2017)
[151] https://www.hs.fi/talous/art-2000005414909.html (19.10.2017)
[152] https://www.hs.fi/talous/art-2000005414399.html (19.10.2017)
[153] https://www.hs.fi/talous/art-2000005422460.html (25.10.2017)
[154] https://www.hs.fi/talous/art-2000005436039.html (3.11.2017)
[155] https://www.hs.fi/talous/art-2000005466135.html (27.11.2017)
[156] https://www.hs.fi/talous/art-2000005494829.html (18.12.2017)
[157] https://yle.fi/uutiset/3-10357434 (17.8.2018)
[158] https://www.hs.fi/talous/art-2000006380632.html (22.1.2020)

Poliisi:

[159] https://yle.fi/uutiset/3-9395364 (10.1.2017)
[160] https://yle.fi/uutiset/3-9698048 (29.6.2017)
[161] https://yle.fi/uutiset/3-9703699 (4.7.2017)
[162] https://www.hs.fi/kotimaa/art-2000006170649.html (11.7.2019)
[163] https://yle.fi/uutiset/3-10954127 (4.9.2019)

Rovaniemi:

[164] https://yle.fi/uutiset/3-11115195 (12.12.2019)

S-pankki:

[165] https://www.hs.fi/talous/art-2000006063680.html (8.4.2019)
[166] https://yle.fi/uutiset/3-10728822 (8.4.2019)
[167] https://yle.fi/uutiset/3-10729185 (9.4.2019)

Tampere:

[168] https://yle.fi/uutiset/3-11053797 (6.11.2019)

THL:n rokotuskattavuus:

[169] https://yle.fi/uutiset/3-9784677 (22.8.2017)

Uber:

[170] https://www.hs.fi/ulkomaat/art-2000005622395.html (29.3.2019)

Visa:

[171] https://www.hs.fi/talous/art-2000005703982.html (1.6.2018)

Verottaja:

[172] https://yle.fi/uutiset/3-10918492 (12.8.2019)