Tampereen yliopisto

Ilkka Hannula

# AVOIMEN LÄHDEKOODIN KOMPONENTIN KÄYTTÖ INTEGRAATIOSSA VALTAKUNNALLISEEN TERVEYDENHUOLTOJÄRJESTELMÄÄN

Diplomityö
Tekniikan ja luonnontieteiden tiedekunta
prof. (emeritus) Hannu Koivisto
yliopisto-opettaja Mikko Salmenperä
Maliskuu 2020

# TIIVISTELMÄ

Ilkka Hannula: Avoimen lähdekoodin komponentin käyttö integraatiossa valtakunnalliseen terveydenhuoltojärjestelmään
Diplomityö
Tampereen yliopisto
Automaatiotekniikan DI-tutkinto-ohjelma
Huhtikuu 2020

---

Työssä käydään läpi avoimen lähdekoodin komponenttien valitsemista, käyttöönottoa, käyttämistä, tietoturvaa sekä ylläpitämistä ja päivittämistä. Työssä etsitään vastaukset kolmeen avoimen lähdekoodin komponentteja koskevaan tutkimuskysymykseen. Tutkimuskysymykset koskevat komponentin valintaa, ylläpitämistä ja päivittämistä sekä mainittuja asioita helpottavia toimintatapoja.

Teoriaosuudessa käsitellään avoimen lähdekoodin komponentteja, alkaen niiden kehityksestä ja kehittämiseen johtavista motivaatioista. Osuudessa käsitellään komponenttien lähdekoodin avoimuuden vaikutusta tietoturvaan, komponenttien kehitykseen liittyvää yhteisöä, komponenttien valintaa, avoimen lähdekoodin komponenttien lisenssejä, komponenttien käyttöönottoa ja käyttämistä sekä komponenttien päivittämiseen liittyviä ongelmia. Päivittämistä ja komponentin ylläpitämistä varten esitellään toimintatapoja, joilla ongelmia voidaan välttää. Toimintatapoja käsitellään yleisesti ohjelmistokehittämisen näkökulmasta ja myös versionhallinnan toimintamalleja avataan.

Työssä tehtävässä integraatiossa avoimen lähdekoodin lääkinnällisten kuvien katselin-komponentti otetaan käyttöön järjestelmään, joka on integroitunut käyttämään valtakunnallista terveydenhuoltojärjestelmää, Kanta-palveluita. Kanta-palveluista erityisesti Kuva-aineistojen arkisto on työn kannalta merkittävä. Atostekin eRA on Kanta-palveluita käyttävä järjestelmä, jonka lääkinnällisten kuvien välittämisestä ja näyttämisestä vastaavaan osajärjestelmään komponentin integraatio kohdistuu.

Integraation toteuttamisen lisäksi työssä käsitellään integraatiossa ilmenneitä haasteita ja ratkaisuja niihin. Työssä käydään läpi myös onnistuneen integraation jälkeistä avoimen lähdekoodin komponentin ylläpitämistä ja päivittämistä, sekä nostetaan esiin ongelmakohtia kuten muuttuneen toimintalogiikan sisältävän päivityksen käyttöönottoa.

Työn käytännön osuudessa esitellään myös avoimen lähdekoodin komponentin valitsemista varten luotu prosessi. Valitsemisprosessi perustuu iteraatioon ja sillä voidaan valita avoimen lähdekoodin komponentti, kun ehdokkaita on useita. Prosessissa on tarkoitus käydä ehdokkaita kokonaisvaltaisesti läpi ja tehdä harkitusti valinta parhaasta ehdokkaasta. Valitsemisprosessia käytetään myös työn integraatiossa käytettävän komponentin valintaan.

Avainsanat: Avoin lähdekoodi, päivittäminen, ylläpito, versionhallinta, terveydenhuolto

# ABSTRACT

Ilkka Hannula: Using open source component in integration to national healthcare system
Master Thesis
Tampere University
Master's Degree Programme in Automation engineering
April 2020

---

The work covers the selection, implementation, use, updating, security and maintenance of an open source component. This thesis aims to answers three research questions related to open source components. The research questions cover component selection, maintenance and upgrading, as well as procedures which facilitate the abovesaid actions.

The theory section introduces open source components by starting from their development and the motivations behind their development. The section also addresses the impact of component source code openness on their security, the component development community, component selection, open source component licenses, component deployment, and component update issues. Beneficial procedures related to component updates and maintenance procedures to avoid problems are discussed. The approaches are addressed from both software development and version control perspectives.

In the practical integration work, the open source medical image viewer component is deployed to a system which is integrated with the nationwide healthcare system, Kanta Services. "Kuva-aineistojen arkisto" is part of Kanta Services and is especially important for the integration. Atostek's eRA is a system that uses Kanta Services and of which subsystem is responsible for transmitting and displaying medical images. This subsystem is the target of the integration

In addition to the implementation of integration, the work presents with the found challenges and solutions considering the integration. The work also reviews the maintenance and updating of the open source component after successful integration, and highlights problem areas such as the introduction of updates with changed operating logic.

The practical part of the thesis introduces a process designed to facilitate the selection of an open source component. The selection process is based on iteration and can be used to select an open source component when multiple candidates exist. In the process, candidates are comprehensively reviewed, and a considered choice of component will be made. In this part the selection process is also used to select the component for the integration done in thesis.

Keywords: Open source, updating, maintenance, version control, healthcare

# ALKUSANAT

Työn puristamisesta loppuun asti haluan aluksi onnitella itseäni. Kiitokset työn mahdollistamisesta kuuluu Atostek Oy:lle ja ohjaajalleni Marko Leppäselle. Haluan kiittää myös koulun tarkastajia Hannu Koivistoa ja Mikko Salmenperää.

Työtä en olisi millään jaksanut tehdä ilman loistavaa tukea perheeltä ja ystäviltäni. Myös Itämerellä ja Kelan opintolainahyvityksellä on ollut iso vaikutus työn tekemiseen motivoimisella, kiitos siitä! Olipa homma.

Tampereella, 6.4.2020

Ilkka Hannula

# SISÄLLYSLUETTELO

# LYHENTEET JA MERKINNÄT

| | |
|---|---|
| CDA | potilasasiakirjadokumentin merkintää koskeva standardi |
| CE-merkintä | vakuutus, että tuote täyttää tuotetta koskevien direktiivien vaatimukset |
| C-STORE | DICOM-standardin mukainen tiedonsiirtoprotokolla |
| DICOM | lääkinnällisten kuvien standardi |
| eRA | Atostekin järjestelmä Kanta-liityntää helpottamaan |
| FDA | Yhdysvaltain elintarvike- ja lääkevirasto |
| Fimea | lääkealan turvallisuus- ja kehittämiskeskus |
| Git | hajautettu versionhallintajärjestelmä |
| GitHub | verkkosivu, joka tarjoaa Git-versionhallinnan projekteille paikan sekä graafisen käyttöliittymän |
| HL7 | terveydenhuollon standardeja kehittävä organisaatio |
| HTTP | protokolla WWW-palvelimien tiedonsiirtoon |
| IEC | kansainvälinen sähköalan standardointiorganisaatio |
| IIS | Microsoftin kehittämä palvelinohjelmistokokonaisuus, joka on tarkoitettu Windows-pohjaisiin palvelimiin |
| IHE | kansainvälinen yhteisö, joka määrittelee terveydenhuollon tietojärjestelmien profiileita standardeihin perustuen |
| ISO | kansainvälinen standardisoimisjärjestö |
| JavaScript | web-ympäristössä käytettävä dynaaminen komentosarjakieli |
| JSON | yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen |
| Kela | Kansaneläkelaitos |
| .NET | Microsoftin kehittämä ohjelmistokomponenttikirjasto |
| OSI | järjestö, joka pyrkii edistämään avoimen lähdekoodin ohjelmistojen käyttöä. |
| PACS | järjestelmä lääketieteellisten kuvien arkistointiin |
| REST | HTTP-protokollaan perustuva arkkitehtuurimalli |
| SSL | salausprotokolla tietoliikenteeseen IP-verkkojen yli |
| Subversion | versionhallintajärjestelmä |
| THL | Terveyden ja hyvinvoinnin laitos |
| Valvira | Sosiaali- ja terveysalan lupa- ja valvontavirasto |
| XCA | IHE-profiili, joka tukee muiden yhteisöjen hallussa olevien potilastietojen hakemista |
| XDS | tiedonvälitysmalli, joka määrittelee tavan välittää tietoa erilaisten potilastietojärjestelmien välillä |
| XUA | IHE-profiili käyttäjän tunnistautumiseen yli potilastietojärjestelmien rajojen |

# 1. JOHDANTO

Työn tavoitteena on löytää vastaukset avoimen lähdekoodin komponentteja käsitteleviin tutkimuskysymyksiin. Kuinka avoimen lähdekoodin komponentin päivittäminen ja ylläpito kannattaa hoitaa? Miten vähentää työmäärää avoimen lähdekoodin komponentin ylläpidossa? Kuinka valita avoimen lähdekoodin komponentti? Työssä keskitytään käsittelemään sellaisia avoimen lähdekoodin komponentteja, joiden lähdekoodi on yleisesti saatavilla ja joiden käyttäminen on ilmaista. Työssä teoriaa hyödynnetään käytännössä ja esitellään avoimen lähdekoodin komponentin integraatio sekä komponentin ylläpitämistä ja päivittämistä. Teorian pohjalta kehitetty prosessi esitellään ja sen perusteella tehdään avoimen lähdekoodin komponentin valinta.

Aluksi käydään läpi avoimen lähdekoodin komponenttien käyttöä, niiden etuja ja haasteita sekä vaatimuksia. Otettaessa käyttöön avoimen lähdekoodin komponenttia, on mietittävä, voidaanko komponenttia käyttää sellaisenaan vai tarvitaanko komponenttiin muutoksia. Koska komponenttia jatkokehitetään yhteisön toimesta sen omassa kehityshaarassa, se päivittyy tulevaisuudessa käyttökohteen ulkopuolella. Päivittäessä on syytä huomioida, miten oman kehityksen kanssa ristiriidassa olevat päivitykset käsitellään.

Osa päivityksistä on tärkeämpiä kuin toiset, esimerkiksi tietoturvan kannalta, mutta pelkästään näiden päivitysten käyttöönotto voi olla haastavaa ja erittäin työlästä. Päivitykset kun nojaavat usein myös vanhempiin päivityksiin, joita ei ole vielä otettu tai haluta ottaa toteutukseen mukaan. Päivitysten käyttöönoton helppoutta määrittelee myös komponentin laatuun liittyvät asiat, joita tarkastelemalla on mahdollista helpottaa päivittämisen työmäärää.

Isossa osassa avoimen lähdekoodin komponenttien hyödyntämisessä on komponenttien valinta erilaisten vertailukriteerien avulla. Tarkoitukseen sopiva ja laadukas komponentti on usein helpompi ylläpitää ja käyttää. Päivittäminen vaatii aina huolellisuutta ja päivityksen onnistuttua on integraation syytä testata.

Työhön kuuluvassa käytännön osuudessa, avoimen lähdekoodin DICOM-katselin-komponentti integroidaan terveydenhuoltojärjestelmään, joka on yhteydessä Kanta-palveluihin. Kanta-palvelut ovat Kansaneläkelaitos, Kelan koko Suomen kattava joukko digitaalisia sosiaali- ja terveydenhuollon palveluita. Kanta-palvelut ovat sekä sosiaali- ja terveydenhuollon toimijoita että kansalaisia varten. Kuva-aineistojen arkisto on Kanta-

palveluihin kuuluva palvelu, johon potilaan hoidon yhteydessä syntyneet kuvantamistutkimukset arkistoidaan. Työn integraatio liittyy erityisesti Kuva-aineistojen arkistoon.

eRA-palvelut on Atostekin kehittämä potilastietojärjestelmien Kanta-palveluihin liittymistä helpottava lääketieteen järjestelmäkokonaisuus. eRA-palvelut käyttävät Kanta-palveluiden rajapintoja, jolloin potilastietojärjestelmät voivat käyttää Kanta-palveluita eRA-palveluiden osien kautta. Näin potilastietojärjestelmien ei tarvitse käyttää suoraan Kanta-palveluiden rajapintoja ja ne selviävät toteutuksessa pienemmällä byrokratialla ja testauskuormalla mitä suora integraatio vaatisi. eRAImageServer on eRA-palveluiden osa, joka vastaa kommunikoinnista Kanta-palveluiden Kuva-aineistojen arkistoon.

Käytännön osuudessa avoimen lähdekoodin OHIF-viewer katselin-komponentti integroitiin eRAImageServerin osaksi hoitamaan Kuva-aineistojen arkistosta noudettujen kuvien näyttäminen. OHIF-viewer on JavaScriptillä toteutettu DICOM-katselin-komponentti, joka toimii selaimessa ja hakee kuvia erilliseltä DICOM-palvelimelta.

Lainsäädännön takia komponenttiin tehtiin muutoksia kuviin liittyvien tunnisteiden piilottamiseksi, koska nämä eivät saa näkyä katselimen käyttämissä hakuosoitteissa. Myös joitain toimintoja otettiin pois käytöstä, käyttöliittymää muokattiin ja katselimeen lisättiin suomen- ja ruotsinkieliset lokalisaatiot.

Työssä käydään läpi muuttuneen katselin-komponentin ylläpitämistä ja päivittämistä sekä nostetaan esiin käytännössä ilmenneitä ongelmia. Koska katselin-komponentti oli valittu jo ennen työn aloittamista, työssä ei käsitellä suuremmin sen valintaprosessia. Työssä esitellään kuitenkin integraatiossa käytettävän HTTP-palvelin-komponentin valinta.

Työn toinen luku keskittyy avoimen lähdekoodin komponentteihin, sekä niiden käyttämiseen ja kehittämiseen. Luvussa käsitellään myös komponenttien valintaa ja ylläpitämistä. Kolmannessa luvussa käsitellään komponentin päivittämistä ja tuodaan esiin mahdollisia keinoja välttää ja ratkaista ongelmia. Neljännessä luvussa käsitellään muuta työn kannalta oleellista teoriaa, kuten lääkinnällisen ohjelmiston määrittelyä ja DICOM-standardia. Viides luku käsittelee käytännön työn kannalta oleelliset järjestelmät ja niiden vaatimuksia. Kuudennessa luvussa käydään läpi työssä tehty integraatio ja hyödynnetään aiemmin käsiteltyä teoriaa käytännössä. Luvussa myös esitellään avoimen lähdekoodin komponentin valitsemiseen kehitetty prosessi ja käytetään sitä komponentin valinnassa.

# 2. AVOIMEN LÄHDEKOODIN KOMPONENTIT

Tässä luvussa käsitellään avoimen lähdekoodin komponentteja niiden käyttämisen, kehittämisen sekä ylläpitämisen näkökulmista. Luvussa käsitellään miten komponentin lähdekoodin avoimuus vaikuttaa niiden kehitykseen, tietoturvaan ja laatuun. Tämän lisäksi käydään läpi mitä avoimen lähdekoodin komponenttien käyttämisessä ja valinnassa kannattaa ottaa huomioon.

## 2.1 Avoimen lähdekoodin ohjelmistot

Avoimen lähdekoodin ohjelmisto tarkoittaa sellaista ohjelmistoa, jonka lähdekoodi on käyttäjän saatavilla ja muokattavissa, ohjelmiston lisenssin mukaisin ehdoin [1]. Avoimen lähdekoodin ohjelmisto ei kuitenkaan ota kantaa ohjelmiston maksullisuuteen vaan niitä voi olla sekä ilmaisia, että maksullisia. Työssä kuitenkin keskitytään käsittelemään sellaisia avoimen lähdekoodin ohjelmistoja ja komponentteja, joiden lähdekoodi on yleisesti saatavilla ja joiden käyttäminen on ilmaista.

Avoimen lähdekoodin ohjelmistossa on aina lisenssi, joka kertoo sen käytön mahdollisuuksista ja rajoitteista. Avoimen lähdekoodin ohjelmiston vastakohtana on suljetun lähdekoodin ohjelmisto, jossa lähdekoodi ei ole saatavilla.

Tyypillisesti avoimen lähdekoodin projekti alkaa, kun jollain henkilöllä tai ryhmällä ilmenee tarve jollekin uudelle ominaisuudelle tai täysin uudelle ohjelmistolle. Tämän jälkeen joku heistä kirjoittaa kyseisen ohjelmiston ja jakaa sen jonkun lisenssin alaisena muille, kenellä on samoja tarpeita [2]. Toki projekti voi alkaa muistakin syistä tai toisella tapaa.

Kun lisenssi sallii ohjelmiston lähdekoodin tutkimisen ja muokkaamisen, avaa se samalla mahdollisuuden muillekin löytää virheitä ohjelmistosta. Ohjelmiston jakaminen laajasti netissä tekee mahdolliseksi muun muassa useiden kehittäjien osallistumisen ohjelmiston koodin tuottamiseen, ominaisuuksien luomiseen, lähdekoodin parantamiseen, virheiden raportoimiseen, korjaamiseen sekä testaamisen.

Avoimen lähdekoodin kehityksen kannattajat usein nostavat esiin, että avoimella lähdekoodilla saavutetaan nopeampaa kehitystä. Ajatus tämän takana on se, että useammat kehittäjät voivat esimerkiksi kirjoittaa koodia, testata ohjelmistoa tai raportoida virheitä yhtäaikaisesti. Virheiden löytämismahdollisuus kasvaa sitä enemmän, mitä enemmän henkilöitä lähdekoodia tutkii, nopeuttaen samalla kehitystä. Vastaavasti suljetun lähdekoodin ohjelmistossa vain harvat näkevät lähdekoodin, joten muiden pitää

käyttää ohjelmistoa lähdekoodia näkemättä. Näin ongelman ilmaantuessa käyttäjällä ei ole mahdollista löytää sille varsinaista juurisyytä.

## 2.2 Avoimen lähdekoodin lisenssit

Tyypillisesti avoimen lähdekoodin komponentilla on aina jokin lisenssi, joka määrittelee sen käyttöä. Tässä esitellään kaksi vaihtoehtoista tapaa jaotella lisenssejä. Ensimmäinen tapa on sen mukaan, kuinka lisenssi rajoittaa komponentin muokatun version edelleen jakamista ja käyttöä. Lisenssit on helppo jakaa tuon ominaisuuden mukaan kolmeen kategoriaan seuraavaksi esitellyllä tavalla [3].

Vahva copyleft lisenssi on lisenssityyppi, joka määrittää, että komponenttia käytettäessä, myös siitä muokattu versio tai sitä käyttävä ohjelmisto on lisensoitava samalla tavalla. Kategoriaa pidetään rajoittavimpana avoimen lähdekoodin lisenssityypeistä. Komponentin käyttö merkitsee ohjelmiston niin sanottua saastumista lisenssiehdoilla, jonka takia tällaisen komponentin käyttöä usein pyritään välttämään kaupallisissa ohjelmistoissa. Esimerkkinä vahvan copyleftin lisenssistä on GNU General Public License, GPL lisenssi.

Heikko copyleft lisenssi on lisenssityyppi, joka myös määrittää, että komponenttia käyttävä ohjelmisto on lisensoitava samaan tapaan. Lisenssityyppi kuitenkin mahdollistaa tietyin ehdoin ohjelmiston julkaisemisen toisella lisenssillä. Luokkaa pidetään kohtalaisesti rajoittavana. Lisenssityyppi sallii lisenssin alaisia komponentteja yhdistettäväksi komponentteihin, jotka eivät ole saatavilla millään avoimen lähdekoodin lisenssillä. Esimerkkinä heikon copyleftin lisenssistä on GNU Lesser General Public License, LGPL lisenssi.

Non-copyleft lisenssi on vähiten rajoittava lisenssityyppi, joka ei velvoita komponentin käyttäjää lisensoimaan ohjelmistoaan vastaavasti kuin komponentti, kunhan tekijänoikeuden omistajalle annetaan tunnustus lisenssin mukaisesti. Tämän lisenssityyppiin komponentteja voidaan käyttää ohjelmistoissa, joissa halutaan välttää lisenssiehdoista saastumista. Esimerkkinä non-copyleft lisenssistä on Apache License Version 2.0.

Lisenssien valintaa helpottamaan Open Source Inatiative, OSI on asettanut kymmenen minimivaatimusta avoimen lähdekoodin lisenssille, jotta ne voivat saada OSI Certified aseman. Toinen tapa jaotella lisenssejä on näiden OSI Certified lisenssien jakaminen rajoittaviin (LGPL, GPL ja MPL) ja salliviin lisensseihin (MIT, BSD, Apache). Näistä sallivat on luotu akateemisiin ympäristöihin. Kyseiset lisenssit sallivat, mutta eivät vaadi johdannaisten ohjelmistojen jakamista. Tämä tarkoittaa edelleen, että lähdekoodia

jatkokehittäneen ei tarvitse luovuttaa luomuksiaan lisenssin omistajille. Tätä voidaan pitää riskinä komponentin kehitykselle [4]. Toinen riski sallivissa lisensseissä on, että joku voi ottaa lähdekoodit ja kehittää niiden perusteella kilpailevan, mutta suljetun ratkaisun markkinoille.

MIT lisenssi on Massachusetts Institute of Technology yliopistossa kehitetty lisenssi ja se on yksinkertaisin avoimen lähdekoodin kehityksessä yleisesti käytetty lisenssi. Sen ainut rajoitus on, että lisenssin ja tekijöiden nimiä, eikä takuuehtoja saa poistaa komponentin yhteydestä. Berkeley Software Distribution, BSD lisenssi on pääpiirteiltään vastaava kuin MIT lisenssi. Apache lisenssi taas muistuttaa paljolti kahta edellistä, mutta uusi versio Apache 2.0 on juridisesti yksityiskohtaisempi ja siinä on avattu oletuksia, joita lisensseistä voidaan tehdä. Apache 2.0 lisenssi antaa mahdollisuuden käyttää johdannaisteoksissa muita lisenssejä. Nämä lisenssit sallivat kaupallisen käytön ja ne myös mahdollistavat lähdekoodin sulkemisen uudelleen.

Rajoittavissa lisensseissä ei ole edellä mainittuja sallivien lisenssien riskejä. Rajoittavat lisenssit ovat salliviin nähden paljon monimutkaisempia ja pidempiä. The General Public License, GPL on ilmeisesti eniten käytetty avoimen lähdekoodin lisenssi [4]. Siinä on osa, joka vaatii johdannaisten ohjelmistojen julkaisua saman lisenssin alla. The Lesser General Public License, LGPL on hieman vähemmän rajoittava kuin GPL. Sen käyttämään komponenttiin tehdyt suorat muutokset on julkaistava samalla tai GPL lisenssillä, mutta yhdistelmiä voidaan lisensoida myös muilla lisensseillä tai jopa omistusoikeudella. Mozilla Public License, MPL kehiteltiin vuonna 1998, kun Netscape päätti julkaista lähdekoodinsa. Myös MPL lisenssissä on vastavuoroisuusvaatimus, joka pakottaa kehittäjät antamaan kaikki lähdekoodimuutokset takaisin yhteisölle.

On olemassa paljon muitakin lisenssejä, joista osa on toisistaan edelleen kehitettyjä. Lisenssiin kannattaa tutustua huolella ennen komponentin käyttöönottamista. Avoimen lähdekoodin komponentin lisenssi ei vaikuta suoranaisesti komponentin käyttöön tai sen ominaisuuksiin. Komponentin lisenssi voi kuitenkin vaikuttaa välillisesti kehittäjien työtapoihin, kun esimerkiksi vahvan copyleft lisenssin komponentin käytön seurauksena on aiemmin suljetun ohjelmiston lähdekoodin avaaminen. Jatkossa käsiteltävät asiat koskevat yleisesti kaikkien lisenssien alaisia avoimen lähdekoodin komponentteja.

## 2.3 Yksittäisen kehittäjän syitä avoimen lähdekoodin kehittämiseen

Kehittäjän tasolla syitä ohjelmiston kehittämiseen avoimella lähdekoodilla tai avoimen lähdekoodin projektiin mukaan menemiselle ovat esimerkiksi halu oppia, halu luoda, tunne joukkoon kuulumisesta, ulkoinen motivaatio ja halu päästä mukaan projektiin [5].

Oppimisen halu on yksi potentiaalinen syy osallistua avoimen lähdekoodin ohjelmiston kehittämiseen. Koska kehittäjä pystyy tunnistamaan tarpeensa, on myös mahdollista, asettaa tavoite, johon oppimisessa pyritään. Tavoite voi olla esimerkiksi ohjelmiston komponentin valmistuminen tai mitä tahansa sen lopputulokseen vaikuttavaa.

Siinä missä kehittäjällä voi olla halu oppia, on ihmiselle ominaista myös halu luoda ja jakaa luomaansa. Tällöin luodaan jotain ja samalla opitaan tehdessä. Avoimen lähdekoodin kehityksessä voidaan esimerkiksi luoda ja kehittää ohjelmistoja tai työkaluja muiden käyttöön. Se taas saattaa tuottaa kehittäjälle mielihyvää ja tyytyväisyyden tunnetta, kun jotain on luotu siihen pisteeseen, että sitä voidaan käyttää ja siitä on muille hyötyä.

Kolmantena syynä on joukkoon kuulumisen tunne, sillä usein avoimen lähdekoodin kehittämistä ei tehdä yksin vaan taustalla on jonkinlainen yhteisö. Joukkoon kuuluminenkin on yksi ihmisen perustarpeista, minkä lisäksi toinen sosiaalinen tekijä on olla muille yhteisössä hyödyksi esimerkiksi jakamalla tietoa.

Motivaation syynä voi olla myös jokin ulkoinen tekijä, mistä helpoin esimerkki on oma tarve kehitettävälle ohjelmistolle tai tarve vaikuttaa sen kehityssuuntaan. Usein mahdollisuus käyttää tuotetta tai mahdollinen siitä saatava tunnustus yhteisöltä tai yhteisön ulkopuolella voi myös motivoida kehittäjää. Kehitystyöstä saatava maine voi muun muassa nostaa kehittäjän statusta ja yritykset voivat tätä kautta löytää työntekijöitä koodinäytteiden kera [6].

Avoimen lähdekoodin kehittäminen voi myös olla mahdollisuus päästä mukaan projektiin. Yksittäisellä kehittäjällä voi hyvin olla halua osallistua johonkin projektiin, kun oman projektin tekeminen tuntuu liian työläältä tai muuten epämotivoivalta. Kehittäjä osana yhteisöä voi pystyä ratkaisemaan haastaviakin ongelmia päästessään tilaan, jossa hän on täysin kiinni tekemisessään ja irti ympäristöstään.

## 2.4 Yrityksen syitä avoimen lähdekoodin kehittämiseen

Siinä missä yksittäisillä kehittäjillä on tiettyjä syitä avoimen lähdekoodin kehittämiseen, on myös yrityksillä omat syynsä. Yrityksellä näitä syitä voi olla esimerkiksi paremmat valmiudet tuottaa innovatiivisia tuotteita, palveluiden myynti avoimen lähdekoodin ohjelmiston ympärillä, tai kehityskustannusten lasku.

Avoimen lähdekoodin kehitys antaa yritykselle paremmat valmiudet innovatiivisten ratkaisujen tuottamiseen, koska kehityksessä on mukana myös muita kuin yrityksen työntekijöitä. Näin yritys saa yhteisön mukaan tuotekehitykseen ja saadaan laajempia näkökulmia kehitykselle ja hyödyntää yhteisön tuesta. Yhteisö tietää, mitä se on vailla,

mikä taas vähentää yrityksen arvailun tarvetta. On myös osoitettu, että yhteisöön luottavat yritykset ovat ennakoivampia uuden koodin julkaisussa. Tämä taas tuottaa nopeammin lisää koodia myös muilta, sillä innovaatioprosessi yhteisössä on kumulatiivinen [7].

Vaikka avoimen lähdekoodin tuotoksia saa vapaasti käyttää, ei se suoraan tarkoita, etteikö yritys voisi tehdä niillä liiketoimintaa. Tuotteiden tueksi voidaan hyvin luoda palveluita, joilla tulosta tehdään. Tällaisia palveluita voi olla esimerkiksi koulutukset, tekninen tuki, konsultointi tai vaikka sertifioinnit. Palveluita yrityksen on helppo tarjota, koska heiltä löytyy kehityksessä tullutta osaamista. Toisaalta muutkin voivat myydä vastavia palveluita, koska ohjelmisto on ilmainen, mutta tähän saadaan kilpailuetua juuri osaamisella tai materiaaleilla, mitä yrityksellä on hallussa. Täytyy kuitenkin muistaa, että vastaavia palveluita voi myydä, vaikka kyseessä ei olisi avoimen lähdekoodin ohjelmisto.

Julkaistessaan ohjelmistonsa avoimen lähdekoodin lisenssillä, on yrityksellä useita mahdollisuuksia säästää kustannuksissa. Näistä selvin on säästää kehityskustannuksissa, kun myös yrityksen ulkopuoliset työntekijät osallistuvat kehitykseen. Toinen hieman enemmän piilossa oleva hyöty tulee, kun käyttäjät tutkivat lähdekoodia, raportoivat virheistä ja toimivat testaajina. Kun lähdekoodiin pääsee käsiksi, on mahdollista, että käyttäjä pystyy itse selvittämään kohdan, jossa vika on. Näin käyttäjien avulla lähdekoodin laatu paranee avoimen lähdekoodin ohjelmistoissa.

Yritys saattaa myös päätyä julkaisemaan jonkun alun perin vain itselleen tarkoitetun osan avoimen lähdekoodin ohjelmistoa yleiseen jakoon. Tällä yritys pyrkii varmistamaan, että yhteensopivuus julkaistuun osaan säilytetään myös tulevissa versioissa ja ilman, että yritys pitäisi siitä itse huolen. Näin yrityksen motivaatio avoimen lähdekoodin kehittämiselle on kehitykselle tietyn suunnan näyttäminen.

Yhtenä syynä lähdekoodin avoimuudelle voi myös olla jonkin käytettävän komponentin lisenssi, joka pakottaa myös muun ohjelmiston avoimeksi. Tällainen tilanne saattaa muodostua, jos lisenssien kanssa ei olla tarkkoja. Komponentti voi myös olla kehityksen kannalta niin tärkeä, että sen käyttöön päädytään, vaikka samalla projekti saastuu komponentin lisenssistä.

Avoimen lähdekoodin kehittäminen voi yrityksen näkökulmasta olla hyvinkin erilaista, riippuen yrityksen suhteesta ohjelmistoon. Yritys voi kehittää ohjelmistoa itse ja saada apua yhteisöltä tai yritys voi olla itse osa yhteisöä ja osallistua ohjelmiston kehitykseen omalta osaltaan. Molemmissa tapauksissa syyt kehitykseen voivat olla samoja, mutta jälkimmäisessä yritystä voi myös motivoida valmis tai osittain valmis kokonaisuus itse tekemisen sijaan. Erääseen avoimen lähdekoodin kehitystä käsittelevään tutkimukseen

osallistuneista noin 40% sai työnantajaltaan palkkaa avoimen lähdekoodin kehitykseen osallistumisesta [8].

## 2.5 Tietoturva avoimen lähdekoodin ohjelmistossa

Ohjelmiston turvallinen toteutus perustuu siihen, että ohjelmisto on oikeasti toteutettu turvallisesti. Suljetun lähdekoodin tapauksessa kuitenkin lisäksi koettu turvallisuus perustuu myös siihen, että harva tietää kuinka toteutus toimii. Näistä ensimmäinen on tietysti toivotumpaa ja oikeaa turvallisuutta ja jälkimmäinen näennäistä turvallisuutta. Avoimen lähdekoodin tapauksessa toteutus on yleisesti saatavilla ja tarkasteltavissa, jolloin lähdekoodin avoimuuteen siirtyessä myös tuo näennäisen turvallisuuden menetys mietityttää.

Kun lähdekoodi on kaikkien saatavilla, on tietysti helppo perustella, että ohjelmisto alttiimpi kohdennetuimmille hyökkäyksille, koska myös kaikki virheet ovat kaikkien saatavilla. Toisaalta virheet on helpompi löytää ja vastaavasti avoimuus motivoi kehittäjää näkemään enemmän vaivaa tehdäkseen laadukkaampaa koodia [9]. Mikäli lähdekoodi ei ole avointa, on kuitenkin mahdollista, että lähdekoodi vuotaa johonkin ja sitä kautta hyökkääjä pääsee käsiksi helposti löytyviin haavoittuvaisuuksiin. Hyökkääjällä on myös mahdollisuuksia löytää haavoittuvaisuuksia, vaikka lähdekoodia ei olisi saatavilla.

Usein löydetyt haavoittuvaisuudet tulee kehittäjälle tietoon, kun löytäjä ilmoittaa niistä, jonka jälkeen ne saadaan korjattua. Mikäli kuitenkin hyökkääjä löytää haavoittuvaisuuden, hän todennäköisesti pitää sen omana tietonaan, että sitä ei korjattaisi. Tällainen tilanne avoimen lähdekoodin ohjelmistossa on epätodennäköisempi, koska virheet ja haavoittuvaisuudet on helpompi löytää myös yhteisön toimesta.

Tämän lisäksi, kun lähdekoodi on saatavilla, on myös yksittäisen käyttäjän mahdollista luoda ohjelmiston tietoturvasta halutessaan tarkka kuva. Jos taas lähdekoodi ei ole saatavilla, on kuvan tietoturvasta antaminen kehittäjän vastuulla, mikä ei ole täysin luotettavaa. Jotkin yritykset tilaavat selvityksiä suljetun ohjelmistonsa tietoturvasta ulkopuoliselta taholta, mutta yleensä selvitys koskee vain tiettyä ohjelmiston versiota. Usein selvitystä ei kustannussyistä uusita joka versiolle, jolloin selvitys on uusille versioille vanhentunut. Avoimen lähdekoodin ohjelmistossa taas tätä selvitystä ohjelmiston tietoturvasta tekee kehittäjän lisäksi myös muut.

Tietysti virheiden ja haavoittuvaisuuksien löytymisen lisäksi avoimen lähdekoodin ohjelmistoissa myös niiden korjaaminen onnistuu nopeammin, koska kuka vaan voi

tehdä korjauksen ja pyytää sen ottamista versionhallintaan muiden saataville. Näin käyttäjät voivat itse korjata tai valita korjauksia, jotka ovat heille tärkeimpiä, eikä heidän tarvitse odottaa, että kehittäjä julkaisee korjauspaketin. Tämä johtaa nopeampiin korjauspäivityksiin ja turvallisempaan toteutukseen. On myös osoitettu, että avoimen lähdekoodin ohjelmistoissa korjaukset julkaistaan nopeammin kuin muissa ohjelmistoissa [9].

## 2.6 Avoimen lähdekoodin komponentin valinta

Avoimen lähdekoodin komponentteja on saatavilla tuhansia ja suurimman hyödyn niistä saavuttaa, kun valitsee sopivimman. Tilanteessa, jossa etsinnässä on avoimen lähdekoodin komponentti ja tarvittavat toiminnallisuudet toteuttavia ehdokkaita on useita, on näistä valitseminen oikeastaan melko suoraviivaista. Valinta kannattaa tehdä muutaman helpon kriteerin avulla, joilla tarkastellaan esimerkiksi lisenssin laajuutta, komponentin sopivuutta, terveyttä ja laatua [10].

Aluksi huomio kannattaa tarkastella komponenttien lisenssejä. Jotkut lisenssit ovat melko liberaaleja ja antavat mahdollisuuden käyttää komponenttia laajasti, kunhan alkuperäinen kehittäjä mainitaan. Toiset lisenssit taas saattavat vaikuttaa ohjelmiston kehitykseen hankaloittavasti. Mikäli komponentti ei sovi lisenssinsä puolesta käytettäväksi, on se hylättävä, koska lisenssille ei voi jatkossa tehdä mitään.

Valinnassa kannattaa huomioida millaisella tahdilla komponentti päivittyy. Tulevista päivityksistä on vaikea löytää tarkkaa tietoa, mutta paras arvaus perustuu usein päivityshistoriaan. Päivityshistoriassa kannattaa kiinnittää huomiota päivitysten tahtiin, uusimman päivityksen tuoreuteen sekä päivitystahdin tasaisuuteen. Mikäli päivityksiä ei ole kuulunut hetkeen, voi olla myös todennäköistä, ettei sellaista tule kovin nopeasti jostain ongelmasta raportoitaessa. Tällaisessa tilanteessa kannattaa varautua, että kehitystä saattaa joutua tekemään itse.

Kannattaa myös tutustua päivityksiin liittyviin kuvauksiin, mitä niissä on tehty ja onko yhteensopivuus niissä säilynyt myös vanhempiin versioihin. Mikäli yhteensopivuus vanhempiin versioihin on usein katkennut, on se todennäköisempää myös jatkossa. Siitä seuraa, että päivitysten käyttöönotto on työläämpää ja päivityksistä jälkeen jääminen houkuttelevampaa.

Seuraavana tarkastellaan komponentin yhteisöä. Kannattaa selvittää, että mitä kaikkia kanavia yhteisöllä on käytössään ja mistä pystyy löytämään tietoa. Mahdollisia kanavia voi olla esimerkiksi keskustelupalsta, sähköpostilista, tehtävienhallinta työkalu ja wiki. Kanavat ovat erityisen tärkeitä, koska yleensä kehityksessä tarvittava ensimmäinen ja

ainoa apu löytyy juuri yhteisöltä. Kannattaa myös selvittää kuinka helposti komponentin käyttöön on mahdollista saada apua ja, että kuinka avoimesti ja mukavasti kysymysten kysyjiä kohdellaan. Entä osallistuuko yhteisössä kysymyksiin vastaamiseen ja keskusteluun pääasiassa yksi vai useampi henkilö.

Komponentin käytettävyyttä helpottaa huomattavasti, mikäli komponentista löytyy dokumentaatiota tai käyttöopas. Dokumentaatiossa kannattaa kiinnittää huomiota sen selkeyteen sekä selvittää onko siitä oikeasti apua komponentin käyttämisessä tai toimintoihin tutustumisessa. Mikäli taas käyttöopas löytyy, on syytä selvittää kuinka hyvä, selkeä ja kattava se on.

Apuna komponentin valinnassa voi myös käyttää työkavereiden tai yleisiä näkemyksiä ja esimerkiksi latausten tai komponenttiin liittyvän keskustelun määrä on hyvä mittari. Mitä suositumpi komponentti on, sitä todennäköisemmin netistä löytyy vastaus komponentin käytöstä nousevaan kysymykseen. Tämä on usein myös hyvä mittari komponentin laadulle. Siinä missä esimerkiksi hyvä päivittyvyys tai dokumentaatio nostaa komponentin laatua, nostaa usein myös sen laaja käyttö sitä. Toki komponentin koodin laatua voi itse tarkastella, mutta isolla ja aktiivisella käyttäjäkunnalla on taipumus parantaa komponentin laatua.

Viimeinen kriteeri on järkevä tässä vaiheessa, sillä komponentista saatava tieto ja komponentin elinvoima ovat tärkeimpiä edellytyksiä käyttöönotolle. Kriteeri on lähdekoodin laatu, tyyli ja helppous. Kriteerin tarkastelu on toki kehittäjästä kiinni, mutta yleensä selkeät erot komponenttien välillä on helppo löytää ja pienet erot ovat taas melko epäolennaisia. Kannattaa tutkia erityisesti sellaisia osia, joita tulisi itse käyttämään. Osia kannattaa tarkastella siinä mielessä, että ovatko ne ymmärrettäviä ja olisiko niitä mahdollisesti helppo ottaa käyttöön.

Komponentin valinnassa kannattaa huomioida sen käyttötarkoitus. Mikäli komponenttia käytetään yleisesti paljon, sen laatu tuskin on sellaista, että siitä koituisi merkittävää haittaa. Jos taas komponentin käyttö ei vaadi muutoksia itse komponenttiin, voi olla järkevä valinnassa joustaa jostain kriteeristä. Komponentin käyttäminen sellaisenaan on erittäin hyvä ylläpidettävyyden näkökulmasta ja sitä käsitellään tarkemmin seuraavassa luvussa.

## 2.7 Avoimen lähdekoodin komponentin käyttöönotto

Kun komponenttia ollaan ottamassa käyttöön, kannattaa kiinnittää huomiota muutamaan seikkaan. Myös käyttöönotossa voidaan nimittäin tehdä valintoja, joilla on vaikutusta tulevaisuudessa komponentin ylläpitoon ja käyttöön.

Käyttöönotossa ensimmäisenä kysymyksenä on, että otetaanko käyttöön lähdekoodit ja käännetään ne itse vai hyödynnetäänkö valmiiksi konekielelle käännettyä koodia. Jälkimmäisessä tapauksessa vältytään käännösprosessiin ja sen ylläpitoon liittyviltä mahdollisilta ongelmilta, mutta samalla menetetään mahdollisuus käännösvaiheen konfigurointeihin tai lähdekoodin pieniin muutoksiin. Konekielen koodia käytettäessä ollaan riippuvaisia kehittäjien tekemistä käännöksistä ja niiden loppuminen johtaa myös komponentin päivitysten loppumiseen. Lähdekoodia käytettäessä voidaan komponenttia hyödyntää myös muissa, konekielen koodien tuen ulkopuolella olevissa ympäristöissä. On myös huomattava, että mikäli komponenttiin tullaan tekemään muutoksia, niin silloin on valittava lähdekoodit. Työssä myös jatkossa keskitytään tähän vaihtoehtoon.

Seuraavana kannattaa miettiä kuinka laajasti komponenttia tullaan hyödyntämään jatkossa. Hyödynnetäänkö komponentista pari riviä omassa koodissa, käytetäänkö sitä kokonaan omana osaohjelmistona, integroidaanko se komponentiksi järjestelmään vai jotain siltä väliltä. Kannattaa pyrkiä käyttämään komponenttia, kuin sitä on tarkoitettu, jolloin se toimii varmemmin. Mikäli kuitenkin esimerkiksi muutosten takia komponenttia ei voida käyttää normaalisti, saattaa vaihtoehtoisia tapoja löytyä, joilla käyttäminen voi onnistua.

Voi myös olla hyödyllistä selvittää kuinka komponentin kehitykseen pääsee mukaan. Kehityksessä mukana ollessa pystyy nimittäin vaikuttamaan kehityksen suuntaan. Tästä syystä useat firmat, jotka käyttävät paljon avoimen lähdekoodin komponentteja, pyrkivät muodostamaan syvempiä suhteita avoimen lähdekoodin projekteihin. Kun kehityksessä esimerkiksi on mukana oman yrityksen väkeä, on helpompi saada mukaan omaa agendaa ja tarvittavia muutoksia.

## 2.8 Komponentin kompleksisuuden vaikutus kehitykseen

Ohjelmistokehityksessä ei ole mitenkään epätavallista, että henkilö, joka muokkaa lähdekoodia, ei ole ollut mukana alkuperäisessä kehityksessä. Seurauksena tästä iso osa ohjelmoijan näkemästä vaivasta menee alkuperäisen lähdekoodin ymmärtämiseen. Lähdekoodin tutkiminen ja ymmärtäminen sisältää esimerkiksi ohjelmiston eri komponenttien välisten suhteiden ja niiden logiikan tutkimista. Usein tässä on kyse myös ison datamäärän käsittelystä, jota tarvitsee pystyä seulomaan, että voi keskittyä tärkeimpiin alueisiin. Ohjelmiston kasvaessa myös lähdekoodin ymmärtämiseen kuluva työmäärä kasvaa rajusti. Lähdekoodin ymmärtäminen on ohjelmiston ylläpidossa isossa roolissa ja on esitetty, että siihen menee enemmän kuin puolet ylläpitoon käytetystä ajasta [2].

Yleisesti monimutkaisissa projekteissa on paljon sisältöä, mitä voidaan muuttaa ja korjata. Kun lähdekoodi on selkeää ja helppoa, on myös sen ylläpitäminen helpompaa. Kun lähdekoodi on kompleksinen, kehittäjä käyttää enemmän aikaansa ymmärtääkseen ja oppiakseen lähdekoodia. Avoimen lähdekoodin ohjelmistoissa kehittäjä voi hypätä projektiin mukaan tai projektista pois, milloin haluaa, sillä hänellä ei ole velvoitetta projektiin. Lähdekoodin yksinkertaisuudesta on erityisesti etua avoimen lähdekoodin ohjelmistoissa, koska se helpottaa uusien kehittäjien liittymistä mukaan. Projektiin mukaan liittymisen pitäisi olla mahdollisimman helppoa, jos on pienikään mahdollisuus, että kehittäjällä haluaisi mukaan.

## 2.9 Avoimen lähdekoodin komponenttien kehitykseen varautuminen

Kehityksellä on aina jokin suunta, johon kehittäjät ja kehityksen rahoittavat vaikuttavat, mutta tulevat muutokset eivät aina ole välttämättä toivottuja tai odotettuja. Jollain tavalla tähän kehitykseen on pystyttävä varautumaan ja lisähaasteen tuo omat muutokset, joita komponentin käyttöönotossa on mahdollisesti tehty. Näin komponentin kehitys tapahtuu yhteisön omassa kehityshaarassa, kun taas ohjelmistossa hyödynnetty versio komponentista kulkee omassaan. Ongelma on siis se, että päivityksen tullessa vaihtoehtona on omien muutosten uudelleen integrointi jokaiseen komponentin versioon tai pysyminen vanhemmassa versiossa [10].

Helpointa on tietysti välttää avoimen lähdekoodin komponentin muokkaamista omiin tarpeisiin. Näin vältytään ylläpitämästä omaa versiota komponentista. Muutosten välttämiseksi voi esimerkiksi yrittää käyttää komponentin eri asetuksia tai muuttaa omaa toteutusta, joka komponenttia käyttää. Jos kuitenkin komponenttiin tehtävät muutokset ovat välttämättömiä, niin kannattaa ne pitää mahdollisimman pieninä ja lähdekoodin tyyliä vastaavana.

Käyttöönottaessa komponenttia on syytä miettiä, että kuinka komponentin kehitystä tullaan seuraamaan. On mahdollista esimerkiksi pysyä tietyssä vakaassa versiossa, ottaa jokainen uusi versio käyttöön tai vaikka ottaa uusi versio aina kun se sisältää tietoturvapäivityksiä. Tietoturvapäivitykset voi ottaa myös manuaalisesti omien muutosten tapaan käyttöön vanhempaan versioon, mutta se voi olla vaivalloisempaa ja manuaalisesti tehdyt muutokset seuraavat mukana jatkossa omien muutosten tapaan. Päivittäessä kannattaa käyttää versionhallintaa apuna ja jokaisen uuden version voi ensiksi ottaa omaan haaraan, mikä helpottaa lokaalien muutosten uudelleen integrointia.

Mikäli muutokset ovat yleispäteviä, on järkevää pyytää niiden mukaan ottamista komponentin kehitysprojektiin. Näin muutokset voivat päätyä osaksi komponentin

virallista kehityshaaraa. Silloin myös muut voivat hyötyä muutoksista ja ne saadaan siirrettyä omasta kehityshaarasta pois, mikä helpottaa päivittämistä jatkossa. Jos muutokset tulee osaksi komponentin virallista kehityshaaraa, näyttävät ne jatkossa kehitykselle suuntaa ja on todennäköistä, että ne tulevat olemaan mukana myös jatkossa.

On myös mahdollista, että käytettävä komponentti perustuu tiukasti johonkin tiettyyn versioon toisesta komponentista. Tällöin käytettävän komponentin lisäksi ollaan sidoksissa myös tuohon tiettyyn versioon riippuvuutena olevasta komponentista. Tästä voi seurata, että riippuvuutena olevaan komponenttiin tulee tietoturvapäivitys, mutta päivitystä oteta käytettävään komponenttiin. Toinen mahdollinen ongelmatilanne on, jos komponentin käyttämän riippuvuuden toinen versio on jossain muussa käytössä olevassa komponentissa riippuvuutena. Tällöin tulee tilanne, että toteutuksessa on riippuvuuksia jonkin komponentin kahteen tai useampaan versioon, mikä voi johtaa isompiin ongelmiin. Kannattaa siis huomioida myös komponentin riippuvuudet, kun varautuu komponentin kehitykseen.

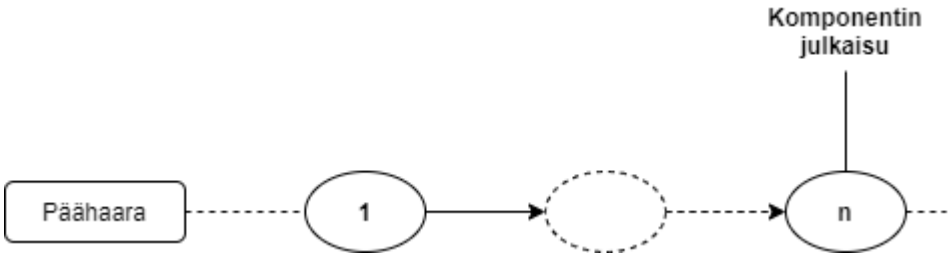
# 3. PÄIVITTÄMISTÄ HELPOTTAVAT KÄYTÄNNÖT

Luvussa nostetaan esiin ongelmia, joita avoimen lähdekoodin komponenttien käytössä ilmenee ja käydään läpi käytäntöjä, joiden avulla komponenttien ylläpitoa voidaan helpottaa. Erityisesti käsitellään avoimen lähdekoodin komponenttien päivittämistä. Luvussa oletetaan, että komponentin käyttöönottoa ei ole tehty ilman omia muutoksia, jolloin sen päivittäminen ei ole aivan yksinkertaista. Käytännöt ovat päivittämisen suhteen yleispäteviä, eikä ole merkitystä onko päivittäminen välttämätöntä tai kuinka suuria päivitykset ovat.

## 3.1 Uusien päivitysten käyttöönotto versionhallinnasta

Versionhallinta on tärkeä työkalu ohjelmistokehityksessä ja sillä voidaan helpottaa päivitysten käyttöönottoa. Erityisesti voidaan ajatella, että versionhallinnalla saavutetaan etua käytettäessä kehityshaaroja. Niiden avulla uuden päivityksen käyttöönotossa ja yhdistämisessä voidaan ehkäistä ongelmia jo toimivan toteutuksen kanssa. Työn integraatiossa käytettävä komponentti on saatavissa Git-versionhallinnalla GitHubista. Gitillä päivitysten mukaan ottamista käsitellään seuraavaksi.

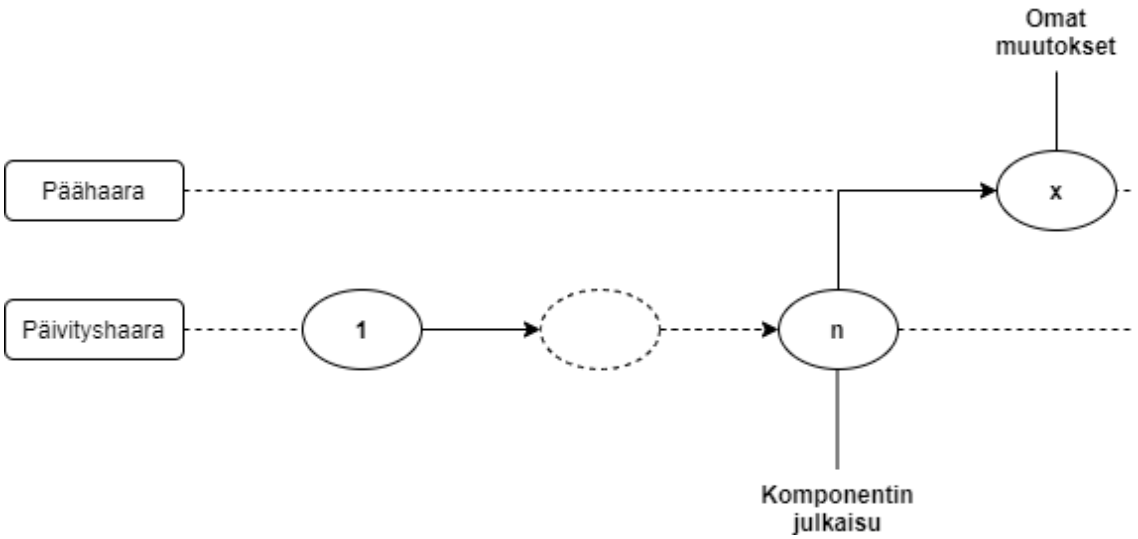
Käyttäessä Git-versionhallintaa aloitetaan siitä, että avoimen lähdekoodin komponentista otetaan klooni paikalliseen versionhallintaan. Samalla asetetaan lähteeksi tietovarasto, josta jatkossa haetaan päivitykset. Kuvassa 1. on esitetty tilanne, jossa komponentti on otettu omaan versionhallintaan, mutta sen koodi on vielä alkuperäistä. Tilanteessa vaiheiden 1 ja n välillä on määrittelemätön määrä tietovarastoon tehtyjä muutoksia. 1 kuvaa komponentin ensimmäistä julkaisua ja n kuvaa kohtaa, joka sisältää komponentin sen hetkisen uusimman julkaisun sellaisenaan ilman muutoksia. Käytettävät kehityshaarat on merkattu kuviin.




***Kuva 1.*** *Alkutilanteessa päähaarassa on komponentin kehityshaaran julkaisu*

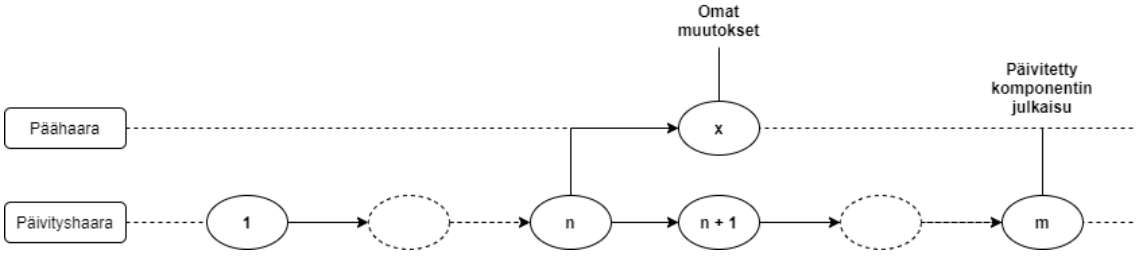
Tässä vaiheessa otetaan komponentin päivityksiä varten käyttöön uusi haara, jossa komponentin virallista kehitystä hallitaan. Pysytään kuitenkin vielä kehityksen päähaarassa, jonne tehdään omat muutokset. Muutokset pusketaan mukaan omaan

versionhallintaan, tilanne kuvassa 2, johon on merkattu x:llä omat muutokset sisältävä versio.




***Kuva 2.*** *Omat muutokset tehdään päähaarassa.*

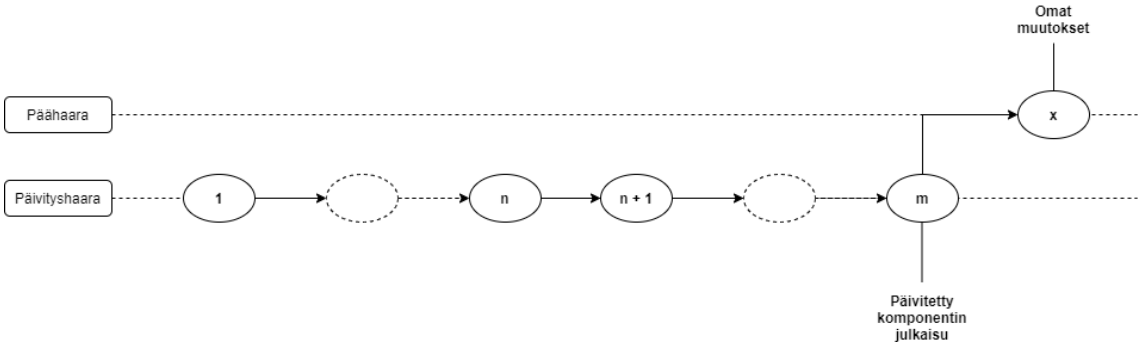
Käytössä on päähaara, josta löytyy omat muutokset, sekä toinen haara, josta löytyy käytettävä avoimen lähdekoodin komponentti, ilman muutoksia. Tilanteessa omia muutoksia on tehty ja seuraavaksi halutaan ottaa komponenttiin tulleet päivitykset käyttöön. Siirrytään komponentin päivitystä varten luotuun haaraan ja haetaan sinne komponentin julkaisutietovarastosta päivitykset. Julkaisutietovarasto on aiemmin määritelty haaran lähdetietovarastoksi. Näin päädytään tilanteeseen, että päivityshaarassa on päivittynyt versio ilman muutoksia ja päähaarassa vanha versio omilla muutoksilla, tilanne esiteltynä kuvassa 3. Kuvassa n + 1 kuvaa julkaisua, joka on seuraava julkaisu n:n jälkeen, m taas kuvaa uusinta julkaisua, joka on määrittelemättömän määrän julkaisua n edelle oleva julkaisu.




***Kuva 3.*** *Päähaarassa omat muutokset ja päivityshaarassa päivitetty komponentti*

Seuraavaksi siirrytään takaisin omat muutokset sisältävään päähaaraan ja tehdään päivityshaaraan kohdistuva rebase-komento. Rebase-komento tekee haarojen yhdistämisen niin, että päivitetyn komponentin päälle ollaan yhdistämässä päähaarassa olevat omat muutokset. Näin läpikäytäviä muutoksia on yleensä vähemmän ja

yhdistäessä mahdolliset päällekkäisistä muutoksista johtuvat ongelmat on helpompi ratkoa. Yhdistämisessä myös verrataan omia muutoksia päivittyneeseen lähdekoodiin, jolloin on helpompi huomata, mikäli jokin toiminnallisuus ei näytäkään toimivan kuin sen pitäisi. Kun yhdistäminen on tehty loppuun, päädytään kuvan 4. tilanteeseen, joka vastaa kuvan 2. tilannetta, jossa päivityshaara on luotu ja komponenttiin omat muutokset kulkevat erillään päähaarassa. Erona kuvien tilanteissa on oikeastaan vain komponentista käytettävä julkaisu. Tässä vaiheessa komponentti on siis päivittyneempi ja vaiheiden n ja m välissä on komponenttiin haetut päivitykset. Omat muutokset ovat edelleen päähaarassa, missä kehittäminen myös jatkossa tapahtuu.




***Kuva 4.*** *Päivitykset on saatu mukaan ja tila vastaa toisen kuvan tilannetta.*

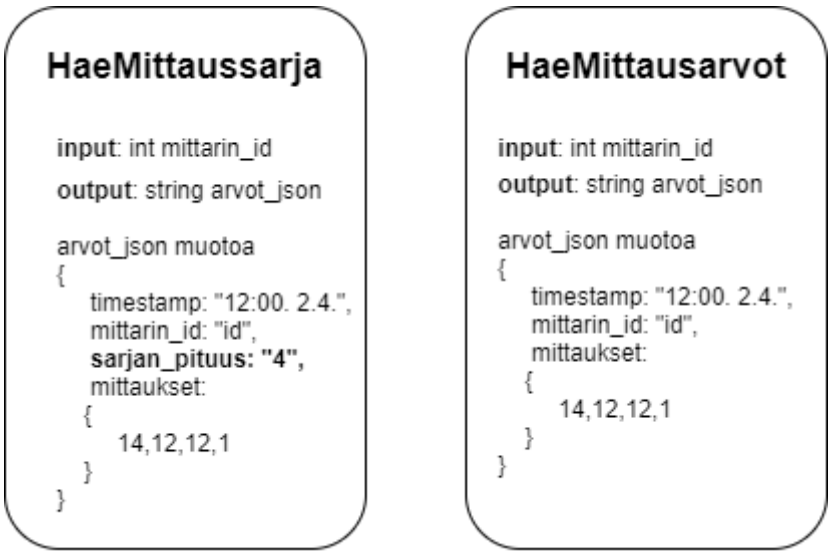
Koska versionhallinta määräytyy avoimen lähdekoodin komponentin versionhallinnan mukaan, voi olla tilanteita, joissa käytössä on Subversion, mutta haluaisi mieluummin käyttää Git:tiä. Tällaisessa tilanteessa on mahdollisuus käyttää jotain Gitin Subversion bridgeä ja hoitaa päivittäminen samaan tapaan kuin Gitillä [11]. Gitin Subversion bridgellä Subversionia voidaan käyttää Gitin tapaan, jolloin päivittäminen on helpompaa.

## 3.2 Ongelmatilanteet päivityksen käyttöönoton seurauksena

Seuraavaksi esitellään mahdollinen ongelmatilanne, joka voi seurata onnistuneen päivitysten käyttöönoton seurauksena. Ongelma voi seurata myös vaikka päällekkäisyyksien kanssa mitään ongelmaa yhdistäessä ei ilmennyt. Ongelma on myös mahdollinen, mikäli lähdekoodin muokkaaminen on kierretty tekemällä muutos komponentin ulkopuolelle omaan toteutukseen.

Käsitellään ensiksi tilanne, jossa avoimen lähdekoodin komponenttiin on tehty oma muutos. Muutos on kohdistunut pyyntöön A, jolla komponentti pyytää toiselta järjestelmältä tietoa. Pyynnön palauttamaan listaukseen on lisätty integraation kannalta oleellinen arvo. Komponentissa on myös pyyntö B, jolla voi tehdä saman asian, mutta se on ainakin näennäisesti toisia tilanteita varten. Pyyntöön B ei ole tehty vastaavaa

parametrin lisäystä, sillä tällä hetkellä sitä ei käytetä. Mahdollinen pyyntöjen ero esiteltynä kuvassa 5, jossa HaeMittaussarja kuvaa pyyntöä A ja HaeMittausarvot pyyntöä B.




***Kuva 5.*** *Esimerkki pyynnöistä, joilla erona paluuarvon sisällössä yksi arvo.*

Päivityksen seurauksena komponentin toteutus muuttuu niin, että myös pyyntöä B käytetään erikoistapauksissa. Päivitys otetaan onnistuneesti käyttöön ja myös omat muutokset saadaan hallittua järkevästi versionhallinnalla. Ohjelmisto kääntyy ja näyttää testailun mukaan toimivan oikein. Tässä vaiheessa jää kuitenkin huomaamatta, että eräässä erikoistilanteessa komponentti on siirtynyt käyttämään pyynnön A sijasta pyyntöä B. Muutos johtaa ohjelman kaatumiseen, koska pyynnön mukana ei palaudu oleellista arvoa.

Edellä mainitun esimerkin kaltainen tilanne voi myös tapahtua, jos komponentti sisältää jo alun perin pyynnöt A ja B, joissa on tuo oleellinen palauttaman listauksen arvo erona. Näin voitaisiin olla tilanteessa, että käytössä olisi täysin alkuperäinen avoimen lähdekoodin komponentti, jonka päivittäminen rikkoisi integraation. Seuraavaksi esitellään tapoja, joilla tämän kaltaisia ongelmia voidaan välttää.

## 3.3 Päivitysten aiheuttamien ongelmien välttäminen

Mahdollisuuksia välttää päivityksistä tulevia ongelmia on ainakin kaksi. Ensimmäinen on, muutoslokin käyminen läpi, jolloin voidaan huomata, mikäli jokin toiminta on muuttunut. Mikäli halutaan olla vielä tarkempia, voidaan käydä jokainen päivitettyyn versioon liittyvä versionhallinnassa oleva yksittäinen päivitys läpi. Näin nähdään vielä tarkemmin, että onko tärkeille alueille kohdistunut muutoksia.

Aina ensimmäinen vaihtoehto ei ole mielekäs tai edes mahdollinen, mikäli muutoksia on tullut niin paljon, että niiden läpi käyminen on todella työlästä. Toinen toimintatapa, mikä ei ole kuitenkaan vaihtoehtoinen ensimmäiselle, vaan olisi hyvä tehdä joka tapauksessa, on testaus. Kun tiedetään mihin osiin ohjelmaa omat muutokset ovat kohdistuneet, on mahdollista päivitysten käyttöönoton jälkeen suorittaa kohdennettua testausta juuri kyseiselle alueelle. Näin pystytään melko tehokkaasti testaamaan komponenttia ja löytämään mahdollisia päivityksen aiheuttamia ongelmia. Etenkin jos testit toteutetaan huolella ja niitä tehdessä mietitään sekä alkuperäistä toteutusta, että omia muutoksia.

Ongelmien välttämiseksi regressiotestaus on hyvä työkalu. Regressiotestauksessa testisetillä pyritään todentamaan, että vanha toteutus toimii muutosten kanssa. Etenkin automaattisena regressiotestauksesta voi olla mittavaa hyötyä. Täytyy kuitenkin huomata, että täysin kattavien testien tekeminen on mahdotonta, eikä aina automaattista testausta ei ole helppoa toteuttaa. Riittävän kattavasta testisetistä voi tulla uusien ominaisuuksien jälkeen vajaa, jolloin enää pelkkä regressiotestaus ei auta paljastamaan ongelmia kaikkialta.

Testaaminen on vaikeampaa, jos integroitava komponentti, sisältää jotain herkästi muuttuvaa ja testien kannalta olennaista, kuten käyttöliittymän. Jos komponentin käyttöliittymä muuttuu hieman ja integraatio vielä toimii täysin, voi testit silti epäonnistua. Tällöin automaattiset testit on korjattava, että niillä voidaan todentaa integraation toimiminen oikein. Toisaalta jos testitapaukset ovat määritelty, on ne mahdollista testata manuaalisesti ja todentaa haluttu toiminta, vaikka käyttöliittymä muuttuisi hieman.

Jos tutkinta tai testaus löytää ongelmia, on mahdollista, että korjaus on helposti tehtävissä tai ei. Tässä vaiheessa kannattaa perehtyä ongelman juurisyyhyn ja tehdä analyysi, että kuinka helposti toteutus olisi uudelleen muokattavissa toimimaan halutulla tavalla. Voi nimittäin olla, että taustalla on uudessa versiossa jonkin ominaisuuden muokkaaminen tai kokonaan käytöstä poisto. Tällaisista muutoksista on toivottavasti myös tiedotettu aiemmin jollain yhteisön tiedotuskanavalla. On myös hyvä pysähtyä miettimään, mihin suuntaan komponenttia ollaan jatkossa kehittämässä ja onko vastaavat ongelmat tulevaisuudessa todennäköisiä.

Toisaalta aiemmin esitetyn tilanteen kaltaisien ongelmien välttämiseen on myös muita mahdollisuuksia, jotka alkavat käytettävän komponentin valinnasta, eivätkä rajoitu vain avoimeen lähdekoodiin. Komponenttia käyttäessä kannattaa aina kiinnittää huomiota tietoihin ja varoituksiin vanhentuvista toiminnallisuuksista. Jo ennen, kun jokin ominaisuus on vanhentumassa, saattaa siitä olla keskustelua tai tietoa yhteisön keskustelukanavilla. Parhaassa tapauksessa käyttäjiltä myös kysytään mielipidettä

tuleviin muutoksiin. Kehittäjät ja yhteisö voivat saada luotettavuutta käyttämällä kehittämisessä järkeviä toimintatapoja. Myös kehityksessä itse mukana oleminen ja lähdekoodin hyvin tunteminen vähentävät todennäköisyyttä päivitysten aiheuttamille ongelmille.

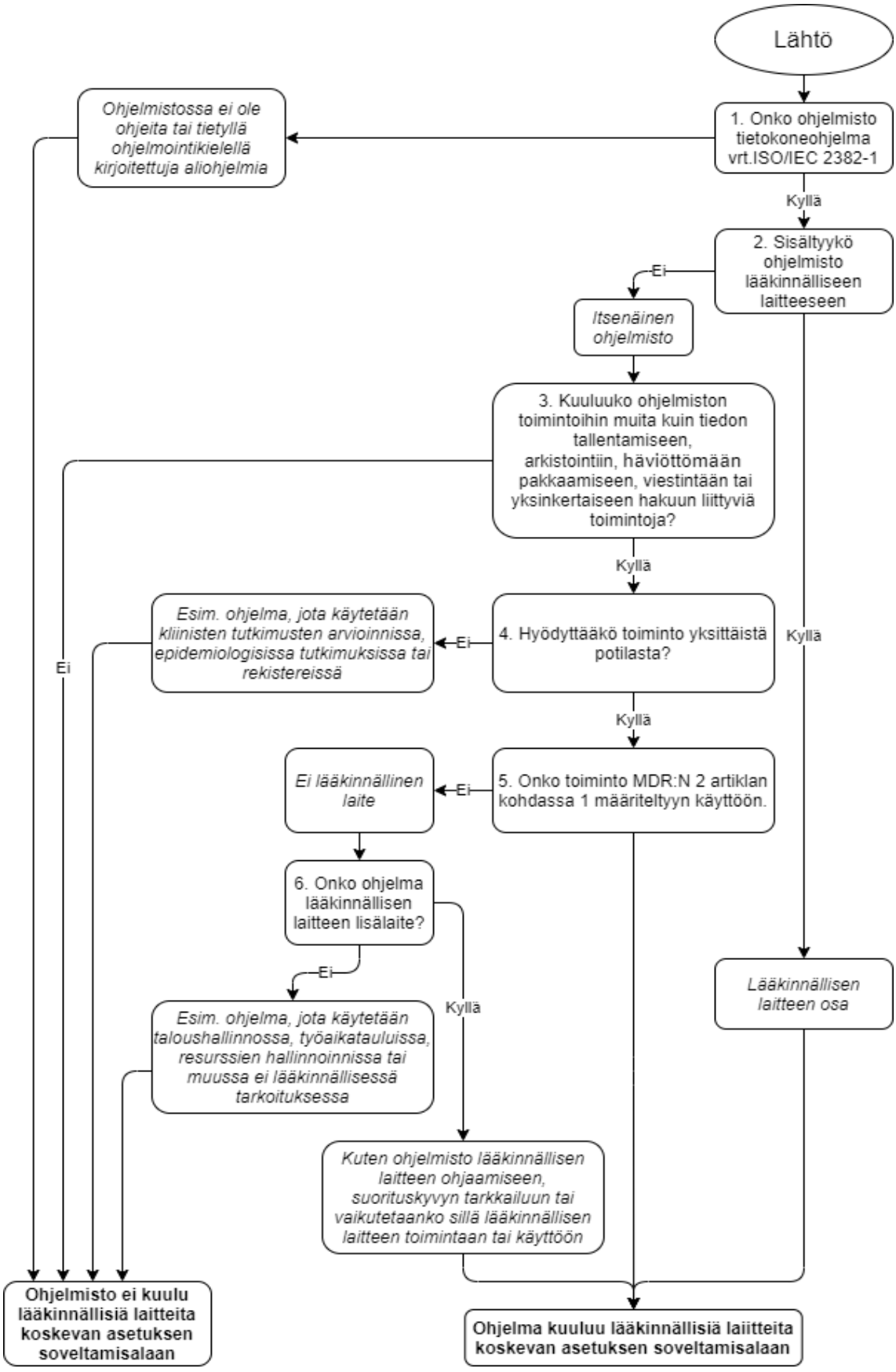
# 4. STANDARDIT JA ASETUKSET

Tässä luvussa esitellään työn integraatioon liittyviä standardeja ja asetuksia. Luvussa esitellään ensiksi lääkinnällisen ohjelmiston määrittelemistä, jonka perusteella määritellään, liittyykö työssä tehtävä integraatio lääkinnälliseen ohjelmistoon. Sen jälkeen käydään läpi integraation käyttämiä IHE-profiileja ja DICOM-standardia.

## 4.1 Lääkinnällinen ohjelmisto

EU-asetuksen 2017/745, mukaan lääkinnällinen laite, on määritelmä, joka on enimmäkseen ohjelmiston kehitystä koskeva asetus [12]. Tässä työssä lääkinnällisellä ohjelmistolla tarkoitetaan ohjelmistoa, mikä on asetuksen mukainen lääkinnälliseksi laitteeksi luokiteltu ohjelmisto. EU-asetus kohdistaa enemmän vaatimuksia ohjelmiston kehitysprosessille kuin itse ohjelmistolle [13]. EU-asetuksen lisäksi lääkinnällisen ohjelmiston tapauksessa kannattaa myös täyttää Yhdysvaltain elintarvike- ja lääkevirasto, FDA:n säännöksen asettamat vaatimukset, mikäli halutaan päästä EU:n ulkopuolelle, myös Yhdysvaltojen markkinoille.

Määritellessä onko ohjelmisto lääkinnällinen ohjelmisto, voidaan standardista käyttää esimerkiksi kuvan 6. mukaista tulkintaa. Tulkintaan kuuluu kuusi kysymystä, joiden avulla määrittely voidaan tehdä [14]. Erityisesti tulkinnan kohdan 3. kysymys on merkittävä. ”Kuuluuko ohjelmiston toimintoihin muuta kuin tiedon tallentamista, arkistointiin, häviöttömään pakkaamiseen, viestintään tai yksinkertaiseen hakuun liittyviä toimintoja?”




***Kuva 6.*** *Kaavio, joka auttaa määrittelemään, onko ohjelmisto lääkinnällinen [14].*

Lääkinnällinen ohjelmisto luokitellaan eri luokkiin sen sisältämän riskin perusteella. Matalan riskin omaavat tuotteet ovat luokkaa I, kohtalaisen riskin tuotteen IIa tai IIb ja korkeimman riskin tuotteet luokkaa III. Käytännössä on erityisen harvinaista, että lääkinnällisen luokan ohjelmisto olisi luokkaa I. Luokkaan I kuuluu lääkinnällisiä laitteita kuten stetoskooppeja ja luokalle CE-merkin voi myöntää itse. Lääkinnällisten ohjelmistojen voidaan ajatella olevan aina vähintään luokkaa IIa, jolloin CE-merkin voi myöntää itse ilmoitetun laitoksen, Notified Bodyn tarkastaessa toimintaa. Ilmoitettu laitos on EU-maan nimeämä organisaatio arvioimaan tuotteiden vaatimustenmukaisuutta ennen tuotteiden markkinoille päätymistä [15].

Työn kannalta kiinnostavin luokka on IIa, joka tuo mukanaan vaatimuksia, jotka voidaan täyttää esimerkiksi taulukossa 1. esiteltyjen standardien käyttöönotolla. Standardit ohjaavat ohjelmistokehitysprosessia ja niiden vaatimuksia ovat esimerkiksi kaikkien moduulien ja kirjastojen taulukointi sekä riskienhallinta-arviointi. Dokumentaatiolle on vaatimus, että koodimuutoksista pitää olla jäljitettävä yhteys tehtäviin, joita ne koskevat ja tehtävistä edelleen yhteys niitä koskevaan vaatimukseen [16]. Vaatimuksilla tarkoitetaan järjestelmävaatimuksia, jotka voivat tulla asiakasvaatimuksista tai viranomaisvaatimuksista. Dokumentaatio tarvitaan, että voidaan myöhemmin jäljittää mikä versio jollain käyttäjällä on ollut käytössä, kun ongelma on ilmennyt. Tästä taas voidaan jäljittää edelleen mitkä muutokset ja vaatimukset ovat olleet ongelman taustalla. Dokumentaatio tulee säilyttää, vaikka versionhallinta tai tehtävienhallintaohjelmisto ajettaisiin alas.

Taulukko 1. *Lääkinnällisen ohjelmiston luokan IIa vaatimusten täyttävät standardit.*

| Standardi | Numero | Selite |
|---|---|---|
| Laadunhallintajärjestelmä | ISO 13485 | Osoittaa laadunhallintajärjestelmän olevan arvioitu sekä vastaavan vaatimuksia ja asiakkaiden tarpeita. |
| Riskienhallintastandardi | ISO 14971 | Kuvaa riskienhallintaa potilasturvallisuus-riskien näkökulmasta. |
| Käytettävyysstandardi | IEC 62366 | Lääketieteellisiin laitteisiin sovellettava käytettävyysstandardi. |
| Prosessistandardi | IEC 62304 | Määrittelee lääkinnällisten ohjelmistojen elinkaariprosessit ja elinkaarivaatimukset. |

Yleisesti voidaan ajatella, että lääkinnällisen ohjelmiston asema auttaa tuotteen markkinoinnissa ja myynnissä. Lääkinnällisen ohjelmiston on myös sallittua tehdä asioita, joita normaali ohjelmisto ei saa tehdä. Esimerkkinä tällaisesta on sykemittarin mahdollisuus lääkinnällisenä ohjelmistona kertoa poikkeavasta rytmistä tai kammiovärinästä. Jos sykemittari ei ole lääkinnällinen ohjelmisto, ei se voi kertoa tuollaista, vaikka se samalla ohjaisi lääkärille. Teoriassa on mahdollista, että sykemittarista voisi tulostaa EKG-käyrän, jonka voi viedä lääkärille tulkittavaksi, mutta tähän toimintaan mittari ei saisi tarpeen mukaan ohjata. Ensiksi ohjelmistolle tulisi määrittää käyttötarkoitus, jonka jälkeen voitaisiin EU-asetuksesta lukea, onko kyseessä lääkinnällinen ohjelmisto ja mitä luokkaa. On nimittäin mahdollista, että sama tuote eri käyttötarkoituksella on lääkinnällinen ohjelmisto ja toisella käyttötarkoituksella myytynä se ei ole.

Sertifioinnin kustannukset ovat isot ja usein prosessia jarruttaa myös byrokratian pelko. Näiden syiden takia yritykset saattavat pelätä ohjelmiston lääkinnälliseksi ohjelmistoksi määrittelemistä. Osa sertifioinnin vaatimuksista vaikuttaa myös yrityksen toimintaan yleisesti, kuten laatujärjestelmä. Laatujärjestelmä ei koske vain lääkinnällisen ohjelmiston tekoa vaan ohjelmiston tekoa yrityksessä ylipäätänsä. Laatujärjestelmä ei ole oikeastaan enää edes kilpailuetu, vaan enemmänkin pakollinen toimintatapa.

Päätöstä ohjelmiston lääkinnälliseksi ohjelmistoksi määrittelystä ei kuitenkaan voi tehdä ilman perusteita, sillä mikäli lääkinnällisen ohjelmiston vaatimusten täyttämiselle ei ole perusteita, eivät Valvira ja Fimea hyväksy ilmoitusta. Mikäli taas laitetta tai ohjelmistoa pystyy helposti käyttämään määriteltyjen käyttötapausten ohi väärin ja niin, että ohjelmisto toimii kuten lääkinnällinen ohjelmisto, on tästä mahdollista saada seurauksia. Lääkinnällisen ohjelmiston määrittely perustuu kuitenkin tulkintaan standardista, jolloin siitä on mahdollista tehdä eri tulkintoja. Ohjelmistot koostuvat usein eri moduuleista, joiden voidaan ajatella olevan omia ohjelmistoja. Näin jokainen ohjelmiston moduuli voi olla erikseen lääkinnällinen ohjelmisto, jolloin määritelmä voi myös koskea vain osaa koko ohjelmiston moduuleista.

## 4.2 Liittyykö työn integraatioon lääkinnällinen ohjelmisto?

Työssä tehtävä integraatio kohdistuu Kanta-Palveluita käyttävään eRA järjestelmään ja tarkemmin sen osaan eRAImageServeriin. Integraatioon liittyvistä järjestelmistä ja niiden osista kerrotaan tarkemmin seuraavassa luvussa. Tämä tarkkuus riittää kuitenkin käsittelemään kysymyksen, liittyykö työn integraatioon lääkinnällinen ohjelmisto.

Tällä hetkellä eRA ei ole lääkinnällinen ohjelmisto, koska sen käyttö ei sisällä lääkinnälliselle ohjelmistolle ominaisia toiminnallisuuksia. Jotkin osat eRAsta saatetaan tulevaisuudessa määritellä lääkinnälliseksi ohjelmistoksi ja merkitä CE-merkinnällä, mikäli eRAn käyttötarkoitus muuttuu. Oletettavasti muutokset tulevat tapahtumaan eri osiin eRAssa kuin eRAImageServeriin. Ei myöskään ole todennäköistä, että eRAa tultaisiin käsittelemään kokonaisuudessa lääketieteellisenä ohjelmistona. Mahdollinen muutos eRAn käyttötarkoituksesa ei siis todennäköisesti vaikuta eRAImageServeriin.

Todennäköisesti eRA kuuluisi lääkinnällisenä ohjelmistona luokkaan IIa. ISO 9001-laatujärjestelmä vaikuttaa jo tällä hetkellä eRAn ja eRAImageServerin kehitykseen, jolloin muutos lääkinnälliseksi ohjelmistoksi ja ISO 13485-standardin alaiseksi ei toisi suurta muutosta, mutta työmäärä sen toteuttamiseen olisi iso. Ei myöskään ole todennäköistä, että eRAImageServer tulisi itsessään olemaan lääkinnällinen ohjelmisto, sillä sen suunniteltuihin toiminnallisuuksiin ei kuulu lääkinnälliselle ohjelmistolle ominaisia toimintoja.

## 4.3 IHE-profiilit

Kuva-aineistojen arkisto ja eRAImageServer käyttävät määritelmän mukaan IHE-profiileja, joiden avulla toiminnallisuudet toteutetaan. Integrating Healtcare Enterprise, IHE on kansainvälinen yhteisö, joka määrittelee standardeihin perustuvia profiileja, joita hyödynnetään terveydenhuollon järjestelmissä [17]. Suomen lainsäädännön ja kansallisen terveydenhuollon järjestelmien arkkitehtuuriperiaatteiden takia näitä toiminnallisia kokonaisuutta tukevia profiileja käytetään myös Kuva-aineistojen arkistossa [18]. Monissa profiileissa on paljon soveltamisvaihtoehtoja ja optioita, joita käytetään tarpeen mukaan.

Health Level Seven International, HL7 on voittoa tavoittelematon, vuonna 1987 perustettu kansainvälinen järjestö, jonka tavoitteena on luoda potilastiedon käsittelyyn liittyviä standardeja [19]. Clinical Document Architure Release 2, CDA R2 on HL7:n julkaisema dokumenttien merkintää koskeva standardi. CDA R2-asiakirjat ovat xml muotoisia ja voivat sisältää kaiken tyyppistä kliinistä sisältöä. Tyypillinen CDA R2-asiakirja on esimerkiksi hoitokertomus tai kuvantamisraportti [20].

IHE on määritellyt joukon kuvantamiseen liittyviä sisältöprofiileja ja näistä kuvantamistutkimuksia koskevia sisältöprofiileja edellytetään Kuva-aineistojen arkistossa käytettäväksi. Näiden lisäksi Kuva-aineistojen arkistossa hyödynnetään CDA R2-asiakirjoja koskevia sisältöprofiileja, joille Kanta-hankkeen yhteydessä on laadittu HL7-määrittelyt. Kuvantamisen kertomusasiakirjoissa käytetään HL7:n määrittelemiä

olemassa oleva sisältömuotoja. Myös IHE:n luonnostason profiileita hyödynnetään Kuva-aineistojen arkiston ensimmäisen vaiheen edellyttämältä osalta. Näihin on päädytty, koska niiden arvioidaan sisältyvän viralliseen versioon 1-2 vuoden kuluessa ja niiden käytössä on etua tulevasta IHE-profiilien mukaisuudesta verrattuna mahdollisiin omiin määrittelyihin [21].

Cross-Enterprise Document Sharing for Imaging profiili, XDS-I.b määrittelee Kuva-aineistojen arkistoon arkistoiduille kuvantamistutkimuksille manifestin muodostamisen sekä XDS-tietovarastoon tallentamisen ja edelleen XDS-rekisteriin rekisteröinnin. Profiili määrittelee kolme vaihtoehtoa lausunnon rakenteelle ja tallennukselle, mutta Kuva-aineistojen arkistossa näiden sijasta käytetään potilastiedon arkiston yhteydessä määriteltyä kuvantamistutkimuksen asiakirjaa ja sen potilastiedon arkistoon tallentamista, jossa asiakirjaa ei tallenneta tietovarastoon.

Muita Kuva-aineistojen arkistossa hyödynnettäviä profiileja ovat mm. Cross-Enterprise Document Sharing, XDS.b, jota käytetään asiakirjojen hakemiseen ja siirtämiseen, Cross-Commonity Access, XCA, joka hoitaa hajauttamisen XDS-ympäristössä sekä Cross-Enterprise User Assertion, XUA, joka puolestaan hoitaa kontekstitietojen välittämisen [18].

## 4.4 DICOM ja PACS

DICOM tulee sanoista Digital Imaging and Communications in Medicine ja tarkoittaa kansainvälistä standardia lääkinnälliseen kuvantamiseen [22]. DICOM tarjoaa kuvantamisen, datan siirron, tallentamisen ja näyttämisen kattavan protokolan, joka on suunniteltu kattamaan kaikki käytännöt nykyaikaisessa lääketieteessä. Standardi määrittelee kaiken tarvittavan kuvien diagnostisesti tarpeeksi tarkkaan näyttämiseen ja kuvien prosessointiin ja sen tavoitteena on olla perustavanlaatuinen standardi lääkinnällisessä kuvantamisessa.

DICOM-standardi myös määrittelee DICOM Information Modelin, mallin, jonka tarkoitus on koostaa reaalimaailman tietoja potilaasta. Malli määrittelee erilaisia attribuutteja, joita potilaasta kuvan yhteyteen tallennetaan. Mahdollisia määriteltyjä attribuutteja on yli 2000 ja sellainen voi olla esimerkiksi potilaan ikä, sukupuoli tai paino. Standardi myös määrittelee attribuuttien Value Representationin, tallennusmuodon, joita on yhteensä 27. Esimerkkejä tallennusmuodosta ovat päivämäärä, aikaleima ja teksti.

DICOM-standardia käytetään lähes kaikissa radiologian, kardiologian ja sädehoidon laitteissa. Standardi on yksi laajimmille levinneistä terveydenhuollon tiedonvälityksen

standardeista ja sitä on kehitelty jo yli 35 vuotta [23]. DICOM-standardi on tunnustettu Internal Organization for Standardization, ISO:n toimesta ISO 12052 standardiksi.

DICOM-standardi on kehittänyt lääkinnällistä kuvantamista, josta esimerkkeinä voidaan mainita, vaikka standardin universaali käyttö lääkinnällisessä kuvantamisessa, erinomainen kuvanlaatu kuvien diagnostista katselua varten ja tuki kuvien mukana kuvantamisen kannalta erittäin tärkeiden parametrien tallentamiseen.

Picture Archiving and Communication Systems, PACS tarkoittaa lääketieteellisten kuvien arkistointiin tarkoitettua järjestelmää, joka koostuu laitteistoista ja ohjelmistoista. PACS:n tavoitteena on parantaa toiminnan tehokkuutta ja samalla edesauttaa paremmassa diagnostiikassa. PACS hoitaa kuvien tallentamisen niin, että kuvia on mahdollista tarkastella myös muualla kuin kuvat tuottaneessa yksikössä. PACS on tarkoitettu DICOM-standardia noudattavien kuvien tallentamiseen.

DICOM-laite toteuttaa vain sen tarvitsemat osat DICOM-standardista ja tästä syystä laitteen mukana tulee DICOM Conformation Statement dokumentti, joka kertoo millä laajuudella laite tukee DICOM-Standardia. Esimerkiksi DICOM-arkisto tukee standardia arkistoinnin laajuudella, DICOM-tulostin taas tulostamisen vaatimalla laajuudella ja niin edelleen [22].

# 5. JÄRJESTELMIEN KUVAUKSET

Luvussa käydään läpi järjestelmät, joihin käytännön osuuden integraatio kohdistuu. Käytännön osuudessa otetaan käyttöön avoimen lähdekoodin komponentti, joka vastaa lääketieteellisten kuvien näyttämisestä. Luvussa esitellään integraatioon liittyvät järjestelmät ja niiden osat sekä olemassa olevan integraatio suhteet. Aluksi esitellään kelan Kanta-palvelut, joka on kokonaisuus, jota eRA käyttää. eRA taas on järjestelmä, jonka osaan varsinainen käytännön osuuden integraatio kohdistuu. eRA käsitellään aluksi kokonaisuutena ja lopuksi tarkennetaan osaan, johon työn integraatio suoraan vaikuttaa.

## 5.1 Kanta-palvelut & Kuva-aineistojen arkisto

Kanta-palvelut on Kansaneläkelaitos, Kelan, sosiaali- ja terveydenhuollon palvelukokonaisuus. Alun perin Kanta oli yhteinen nimitys Kansalliselle Terveysarkistolle, terveydenhuollon ja apteekkien valtakunnallisille tietojärjestelmäpalveluille [24]. Kanta-palvelut on kuitenkin laajentunut sisältämään yhä enemmän palveluita sähköisen lääkemääräyksen lisäksi, eikä nimeä kirjoiteta enää auki.

Kanta-palvelujen tehtävä on taata potilaan tietojen saatavuus hoitotilanteessa aina ajantasaisena. Kanta-palvelut siis huolehtivat sujuvasta tiedonkulusta terveydenhuollossa. Kanta-palvelut on tarkoitettu erityisesti terveydenhuollon toimijoille, mutta niistä on myös paljon hyötyä yksittäisellä kansalaisella, sillä ne mahdollistavat paremman terveydenhuollon. Kanta-palvelut on koko ajan kehittyvä palvelukokonaisuus, johon sisältyy tällä hetkellä seuraavat sähköiset palvelut [25].

**Omakanta** on kansalaisten henkilökohtaiseen käyttöön tarkoitettu palvelu. Palvelun avulla käyttäjä voi katsella sähköisiä lääkemääräyksiään ja niiden tilaa sekä tarkastella omia terveystietojaan. Avohoidosta löytyy lähes kaikki kirjatut potilastiedot ja osastohoidon osalta näkyvissä on tietojen yhteenveto [26]. Käyttäjä voi Omakannassa myös hallinnoida suostumuksiaan ja potilastietojen luovutusta koskevia kieltoja ja tai tahdonilmaisuja.

**Resepti-palvelu** on palvelu, jossa kaikkien reseptien määräys ja toimitus toimii. Paperi- tai puhelinreseptejä määrätään vain poikkeustapauksissa. Palvelu mahdollistaa esimerkiksi lääkärin potilaalle kirjoittamat sähköiset reseptit ja niiden tallentamisen. Resepti-palveluun pääsee myös apteekkien tietojärjestelmillä, jolloin lääkkeiden toimittaminen on mahdollista mistä tahansa apteekista. Potilaan suostumuksella lääkäri

voi tarkastella potilaan kokonaislääkitystä mikä ehkäisee lääkkeiden haitallisia yhteisvaikutuksia. Omakannasta potilas voi itse tarkastella omia reseptejään [26].

**Lääketietokanta** sisältää tarpeelliset tiedot lääkkeistä ja korvattavuuksista, lääkkeiden määräämistä ja toimittamista varten. Näitä tietoja ovat esimerkiksi tuotteen hinta ja tuotteen mahdollinen korvattavuus jollain toisella lääkevalmisteella. Lääketietokannasta löytyy myös lääkkeeseen liittyviä tietoja kuten vaikuttava aine, vahvuus, pakkauksen koko ja yksikkö sekä ATC-koodi [26].

**Potilastiedon arkisto** on sähköinen terveydenhuollon palvelu. Se mahdollistaa potilastietojen keskitetyn arkistoinnin ja säilyttämisen sekä tietojen siirtämisen järjestelmästä toiseen, koska tiedot tallennetaan teknisesti yhtenevässä muodossa. Palveluun tallennetaan myös tietoja potilaan suostumuksista, kielloista ja tahdonilmaisuista. Potilastiedon arkistoa käytetään aina potilastietojärjestelmällä.

**Vanhojen potilastietojen arkistointi.** Vanhat potilastiedot voidaan arkistoida säilytettäväksi ainoastaan Potilastiedon arkistoon, jolloin ne voidaan poistaa lähdejärjestelmästä. Palvelu ei sisällä lähdetietojärjestelmästä tietojen hakua, eikä niiden standardointia. Tämän vuoksi potilasasiakirjojen toimitus pitää tehdä ennalta määrätyssä formaatissa.

**Sosiaalihuollon asiakastiedon arkisto** on valtakunnallinen sosiaalihuollon tietojärjestelmä. Se mahdollistaa sosiaalihuollon asiakastietojen pysyvän keskitetyn arkistoinnin ja säilyttämisen, sekä tietojen aktiivisen käytön. Sosiaalihuollon asiakastiedon arkistoa käytetään aina asiakasjärjestelmällä. Tallennetut asiakirjat ovat tällä hetkellä vain rekisterinpitäjän omassa käytössä, mutta tulevaisuudessa ne tulevat myös muiden rekisterinpitäjien käyttöön [27].

**Terveydenhuollon todistusten välitys** on palvelu, jonka avulla terveydenhuollon ammattihenkilöiden kirjoittamia todistuksia ja lausuntoja on mahdollista välittää terveydenhuollon ulkopuolisille organisaatioille, kuten Kelan etuuskäsittelyyn suoraan potilastietojärjestelmästä [28].

**Kelain** on verkkopalvelu, jossa lääkäri tai hammaslääkäri voi tehdä lääkemääräyksen esimerkiksi lääkärin ammattioikeuden perusteella. Kelain on tarkoitettu käytettäväksi tilanteisiin, kun yksityiskäytössä Kanta-palveluihin liitettyä potilastietojärjestelmää tai muuta sähköistä palvelua ei ole saatavissa [29].

**Kanta-asiakastestipalvelu** on Kanta-palveluihin liitettävien tietojärjestelmien toimittajille, näiden asiakastestaaja organisaatioille sekä apteekeille tarkoitettu palvelu, jossa testataan Kanta-palveluiden toimivuutta tietojärjestelmissä sekä niiden toimivuutta niitä käyttävillä organisaatioilla. Asiakastestiympäristö on Kanta-palveluita vastaava

ympäristö, jossa on tarkoituksena hoitaa kaikki testaaminen. Näin pyritään vähentämään mahdollisuutta, että viallisia tietoja tai rakenteita päätyisi Kanta-palveluiden tuotannossa olevaan tietokantaan.

Kanta-palveluihin sisältyy myös **Kuva-aineistojen arkisto**, jonka toteuttaminen perustuu asiakastietojen sähköistä käsittelyä koskevaan lakiin ja sosiaali- ja terveysministeriön asetukseen. Kuva-aineistojen arkisto toteutetaan vaiheittain ja sen toteuttaminen on vielä kesken. [30]. Arkiston ensimmäiset käyttöönotot tapahtuivat syksyllä 2018 ja ne koskivat kuvien arkistointia radiologisista tutkimuksista. Koska käytännön työn integraatio koskee nimenomaan Kuva-aineistojen arkistoa, käsitellään sitä seuraavaksi vielä tarkemmin.

## 5.2 Kuva-aineistojen arkisto

Kuva-aineistojen arkisto on Kanta-palveluihin kuuluva palvelu, johon potilaan hoidon yhteydessä syntyneet kuvantamistutkimusten kuva-aineistot arkistoidaan. Näitä kuva-aineistoja on esimerkiksi röntgen- ja magneettikuvat sekä niihin liittyvät pyynnöt, merkinnät ja lausunnot. Palvelun avulla hoidetaan varsinaisten potilaan kuvantamistutkimusten haun ja tallentamisen lisäksi myös Potilastiedon arkistoon tallennettujen kuvantamisen hoitoasiakirjojen haku. Palvelua käyttävät sekä kuvantamistutkimuksia tekevät, että tutkimuksia lausuvat terveydenhuollon organisaatiot. Palvelun käyttäjällä tulee olla käytössään Potilastiedon arkisto sekä järjestelmä, joka soveltuu asiakirjojen arkistointiin sekä hakemiseen [31]. Tämän lisäksi kuva-aineistojen arkistoon liittyviltä järjestelmiltä vaaditaan sertifiointi, jonka osana on läpäistävä Kelan yhteistestaus.

Yhtenäinen Kuva-aineistojen arkisto helpottaa kuvien saatavuutta yksityisten ja julkisten palveluntarjoajien välillä valtakunnallisesti. Mitä useampi terveydenhuollon organisaatio käyttää palvelua, sitä enemmän arkistosta on hyötyä. Arkisto auttaa kuvien välittämisessä organisaatioiden välillä tehden siitä nopeampaa, helpompaa ja tietoturvallisempaa. Palvelu myös edellyttää standardisoituja kuvantamistutkimusten merkintätapoja ja näin parantaa tietojen yhteiskäyttöä sekä virheettömyyttä. Kuva-aineistojen arkisto mahdollistaa vanhoista alueellisista tai paikallisista arkistoista luopumisen, mikä säästää kustannuksia. Arkisto myös mahdollistaa helpomman arkistoitujen vertailukuvien hakemisen, samalla kun se vähentää manuaalista arkiston hallinnointia [32].

Potilas itse voi tutustua Omakannan kautta kuvantamistutkimuksia koskeviin lausuntoihin, mutta ei näe tutkimusten kuva-aineistoja. Arkiston mahdollistama tietojen

yhteiskäyttö ja saatavuus myös ehkäisee potilaalle tehtäviä päällekkäisiä kuvantamistutkimuksia ja parantaa potilasturvallisuutta vähentäen esimerkkisi röntgenkuvien aiheuttamaa säteilykuormaa [33]. Arkiston kuvat ovat potilaan luvalla vain terveydenhuollon ammattihenkilöstön käytössä, mikä parantaa potilaan tietosuojaa ja tietoihin pääsyn voi myös kieltää.

Terveyden ja hyvinvoinnin laitos, THL sekä Kela kehittävät Kuva-aineistojen arkistoa yhdessä tietojen monipuoliseen hyödyntämiseen. Heillä on jatkossa tarkoitus saada mukaan myös silmälääkäreiden ja suun terveydenhuollon tuottamia kuvia tuottajien omista arkistoista. Vastaavasti myös kartoitetaan, voitaisiinko säteilytietoja arkistoida järkevästi. Näin terveydenhuollon ammattihenkilöstö pystyisi Kuva-aineistojen arkiston avulla seuraamaan potilaaseen kohdistunutta säteilyrasituksen määrää, kun arvioi tarvetta uusille kuvantamistutkimuksille. Tämän lisäksi käynnissä on myös tekninen määrittely sydänfilmeille ja näkyvän valon kuville.

Kuva-aineistojen arkisto on Kanta-palveluiden osuus, johon käytännön osuuden integraatio kohdistuu. Työssä ei integroiduta suoraan Kanta-palveluihin, vaan tehdään integraatio Kanta-palveluita käyttävään järjestelmään, josta kerrotaan tarkemmin seuraavaksi.

## 5.3 eRA-Palvelut

Atostekin eRA-palvelut ovat lääketieteellinen järjestelmäkokonaisuus, joka helpottaa ja nopeuttaa terveydenhuollon tai sosiaalihuollon toimijan Kanta-palveluihin liittymistä ja niiden käyttöä. eRA-palvelut voidaan integroida osaksi potilastietojärjestelmää, mutta niiden käyttäminen onnistuu myös web-käyttöliittymän kautta ilman potilastietojärjestelmää [34]. Liittymällä eRA:n kautta Kanta-palveluihin, välttää pitkän hyväksymisprosessin sekä laajan integrointi- ja testaustaustyön. Sovellusten ja tietojärjestelmien integraatio eRA-palveluiden avulla Kanta-palveluihin tapahtuu ilman raskasta yhteistestausta tai sertifiointeja yhden rajapinnan kautta [35].

eRA-palveluiden avulla voidaan hakea ja esittää Kanta-palveluihin tallennettua tietoa, sekä tallentaa Kanta-palveluihin edelleen uutta tietoa. eRA-palveluihin kuuluu tällä hetkellä potilastiedon arkistoon liittyvä eRA Arkisto, sähköisiin lääkemääräyksiin liittyvä eRA Resepti, lääkärin todistuksiin ja -lausuntoihin liittyvä eRA Lomakkeet, sosiaalihuollon asiakastiedon arkistoon liittyvä eRA Kansa, ostopalveluihin liittyvä eRA Osva ja tulevaisuudessa myös kuva-aineistojen arkistoon liittyvä eRA Kvarkki. Näiden lisäksi eRA on saatavina myös laboratoriojärjestelmille tietojen Kanta-palveluihin siirtoa

varten. eRA Siirrot taas mahdollistavat organisaatioille vanhojen potilastietojen siirron Kanta-arkistoon [34].

eRAn käyttämiseen tarvitaan Valviran ja Väestörekisterikeskuksen myöntämä terveydenhuollon ammatti- tai toimikortti. Kortilla hoidetaan käyttäjän tunnistautuminen ja sähköinen allekirjoitus. Kortin käyttämistä varten tarvitaan kortinlukija sekä Atostekin kortinlukijaohjelmisto eRA SmartCard. Ohjelmisto on saatavilla Windowsille, OS X:lle, iOS:lle sekä Androidille [34].

## 5.4 eRA ja eRAImageServer

eRA-palveluihin kuuluu kehittäjän näkökulmasta projekteja, joita ovat esimerkiksi eRA ja eRAImageServer. Näiden lisäksi palveluihin kuuluu myös tämän työn kannalta hieman merkityksettömämpiä osia, jotka toteuttavat muita osia eRA-palveluista tai simuloivat Kelan Kanta-palveluita. Lyhyesti eRA:sta puhuttaessa tarkoitetaan jatkossa eRA projektia, joka toteuttaa ison osan eRA-palveluista ja jonka ympärille kokonaisuus lopuista osista rakentuu. eRA toteuttaa eRA-palveluiden web-käyttöliittymän ja integraatiorajapinnan, jotka ovat käyttäjän kannalta eRA-palveluiden tärkeimmät ja näkyvimmät osat. Lääkäri voi eRAn web-käyttöliittymästä avata potilaalle palvelutapahtuman potilaan tietojen käsittelyä varten. Palvelutapahtumassa lääkäri voi esimerkiksi katsella potilaan vanhoja potilastietoja, luoda potilaalle uuden lausunnon tai luoda sähköisen lääkemääräyksen. Myös lääketieteellisten kuvantamisasiakirjojen käsittelyn ja katselun on tarkoitus tapahtua tätä kautta.

eRAImageServer on eRA-palveluiden osa, joka hoitaa lääketieteellisten kuvien hakemisen, välittämisen ja näyttämisen. Myös eRAImageServer on yksittäinen projekti, mutta toimiakseen se tarvitsee eRAn, jonka kautta sitä käytetään. Alussa käyttö on web-pohjaista, mutta tulevaisuudessa käyttämistä varten eRAan toteutetaan myös integraatiorajapinnat. Osana eRAImageServeria tulee olemaan katselin-komponentti, jonka varsinainen kehitys tapahtuu erillään omassa projektissa.

eRAImageServer on Visual Studiolla kehitettävä ja .NET:llä toteutettu projekti. Koska projekti vastaa DICOM-kuvien välittämisestä ja välillä kuvat saattavat olla hyvinkin isoja, ei ohjelmistoa voida suorittaa Microsoftin Internet Infromation Services, IIS palvelinohjelmistokokonaisuudessa. DICOM-kuvien koot saattavat olla jopa kymmeniä megatavuja ja sen lisäksi useamman kuvan välittämisen pitäisi onnistua samalla kertaa. IIS:ssä useampien noin isojen kuvien välittäminen kerralla aiheuttaisi ongelmia, eikä se ole oletuksena edes mahdollista. DICOM-palvelimen lisäksi eRAImageServerillä valittiin toteutettavaksi HTTP-pyyntöjä vastaanottavat WWW-palvelimet. HTTP-pyynnöillä

käytetään eRAImageServerin DICOM-rajapintaa, jonka kautta katselin-komponentti hakee kuvien sisällön. Varsinainen katselin-komponentti on myös saatavilla palvelimella, johon kuvia varten generoidaan tilapäisiä linkkejä, eRAn käytettäväksi. Luvun lopussa on esitetty koko integraatio kuvilla varustettuna.

## 5.5 OHIF-viewer komponentti

Open health imaging foundation, lyhyemmin OHIF on voittoa tavoittelematon avoimen lähdekoodin lääkinnällisen kuvantamisen ratkaisuihin keskittynyt organisaatio. Organisaation OHIF-ohjelmistokehys on tarkoitettu lääkinnällisen kuvantamisen järjestelmien kehitystä varten [36]. OHIF-viewer on ohjelmistokehykseen kuuluva katselin-komponentti, joka toimii selaimessa ja näyttää lääkinnällisiä DICOM-kuvia. Organisaation tavoitteena on tarjota yksinkertainen ja yleiskäyttöinen DICOM-kuvien katselin, jota voidaan edelleen laajentaa spesifisiin tarkoituksiin [37].

Katselin-komponentti voidaan liittää DICOM-standardia tukeviin kuva-arkistoihin ja sillä voidaan katsella kaksi- tai kolmiulotteisia kuvia. Katselin-komponentista on olemassa organisaation ylläpitämä ja suoraan katseltavissa oleva demo heidän sivuillaan. Komponentti on lisensoitu myös kaupalliset toteutukset sallivalla MIT-lisenssillä. Avoimen lähdekoodin ohjelmistokehityksellä organisaatio toivoo saavansa tukea myös muilta kehittäjiltä ja yhteisöltä.

Kuvapalvelimelta haetaan kuvien lisäksi myös niiden metatietoja. Kuvien näyttäminen tapahtuu selaimessa ja kuvat haetaan suoraan kuvapalvelimelta. Katselin-komponenttia käyttäessä käyttäjällä tarvitsee olla yhteys kuvapalvelimeen. OHIF-viewer pyörii selaimessa ja sillä on mahdollista katsella kuvia niin tietokoneella kuin tableteilla ja älypuhelimillakin [38]. OHIF-vieweristä ei löydy Conformation Statementtia ja tieto sen DICOM-standardin osien täyttämisestä puuttuu sen dokumentaatiosta.

OHIF-vieweristä on olemassa kaksi versiota, jotka molemmat ovat toteutettu Javascriptilla. Ensimmäinen toteutus, 1.0 pyörii Meteor sovellusalustalla ja sen kehitys on lopetettu [39]. Toisessa versiossa, 2.0 taas on siirrytty käyttämään React ohjelmistokehystä [40]. Versioista uudempaa käytetään työn käytännön osuudessa.

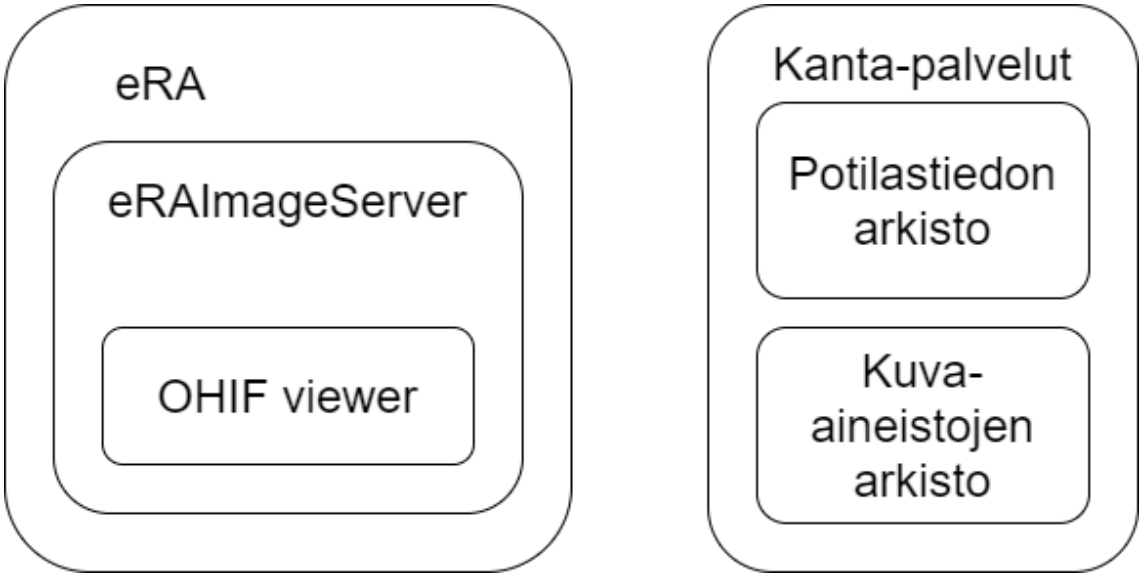
Julkaisut komponentista tapahtuvat automaattisesti versionhallintaan tehdyn päivityksen perusteella. Julkaisut regressiotestataan, jonka jälkeen ne päivitetään OHIF-Viewerin demopalvelimelle. Käyttäjät saavat julkaisut käyttöönsä joko GitHubista tai Node Package Managerista, NPM [41].

OHIF-viewer valikoitui projektissa käytettäväksi katselimeksi, koska se oli valinta hetkellä paras varteenotettava web-pohjainen DICOM-katselin-komponentti. Katselin-

komponentin oli oltava ilmainen avoimen lähdekoodin komponentti, jonka lisenssi ei estä kaupallista käyttöä. Muita valintakriteereitä olivat muun muassa säännöllinen ylläpito, katselimen laajuus sekä käyttö osana muitakin ohjelmistoja. Tämän lisäksi haluttiin, että komponenttia on optimoitu tarpeeksi, että sillä onnistuu myös isompien DICOM-kuvien käsittely.

## 5.6 Integraatiokokonaisuus

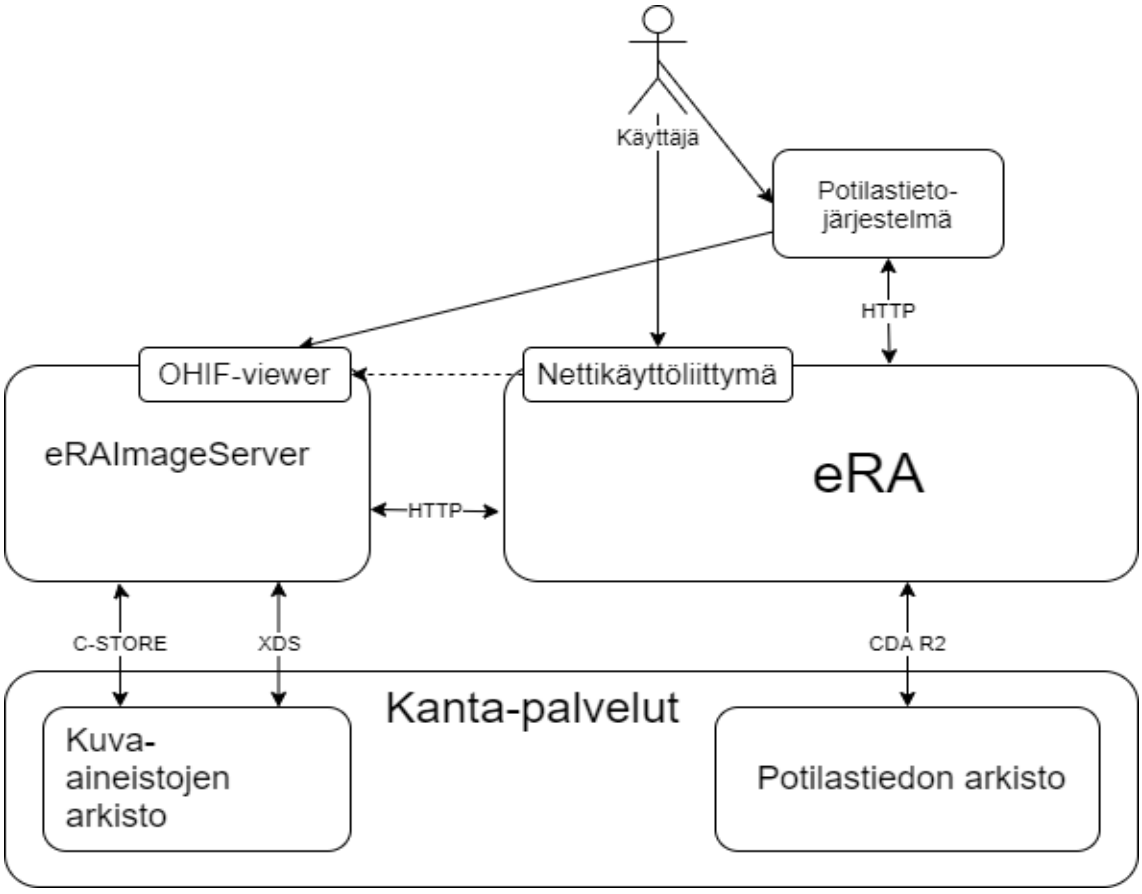
Integraatiota koskevat komponentit on jo erillään esitelty, mutta niiden välisiä suhteita ei ole vielä täysin esitelty. Kuvassa 7. on esitelty integraation oleelliset järjestelmät eRA ja Kanta-palvelut, sekä niiden sisältämät osajärjestelmät. eRAImageServerin käyttö tapahtuu aina eRAn kautta ja seuraavaksi esitellään koko toimintaketju.




***Kuva 7.*** *Integraation kannalta oleelliset osajärjestelmät*

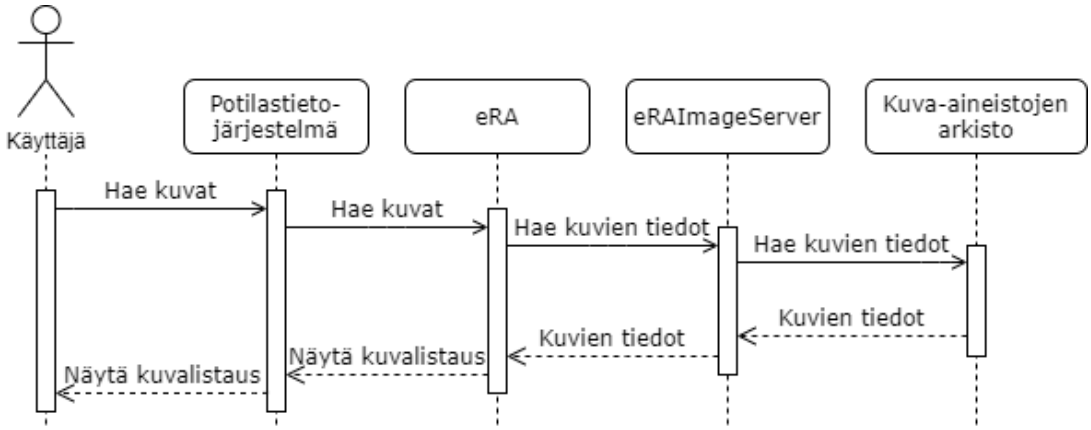
Aluksi käyttäjä käyttää eRAa joko potilastietojärjestelmän kautta tai suoraan nettikäyttöliittymästä. Potilastietojärjestelmä kommunikoi eRAn kanssa HTTP-liikenteellä REST rajapinnan avulla. Myös linkki OHIF-vieweriin välittyy tätä kautta. Nettikäyttöliittymässä käyttäjälle aukeaa katselin uuteen välilehteen, integraatio on avattu kokonaisuudessaan kuvassa 8.

eRA käyttää HTTP-liikennettä kommunikoidessaan eRAImageServerin kanssa ja huolehtii CDA R2-asiakirjojen välityksestä Potilastiedon arkistoon. eRAImageServer taas kommunikoi XDS profiilien ja DICOM-standardin määrittelemän C-STORE protokollan mukaan Kuva-aineistojen arkiston kanssa.



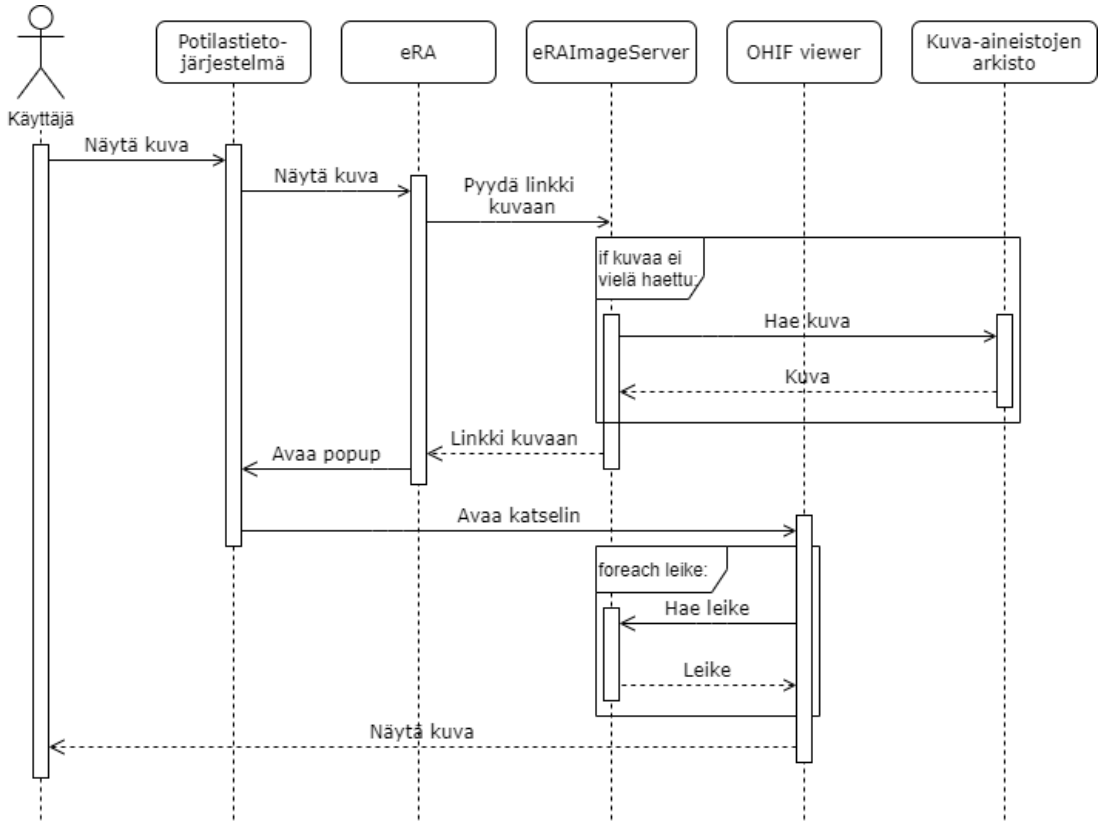

***Kuva 8.*** *Integraatiokuva koko järjestelmästä*

Kuvan näyttämistä varten käyttäjän on aluksi pyydettävä eRAa hakemaan eRAImageServeriltä kuvia, joko potilastietojärjestelmän kautta tai ilman. Tämän jälkeen eRAImageServer hakee kuvien tiedot Kuva-aineistojen arkistosta ja palauttaa ne eRAlle edelleen käyttäjän tai potilastietojärjestelmän saataville. Sekvenssikaavio kuvien hakemisesta on kuvassa 9.




***Kuva 9.*** *Kuvien haun sekvenssikaavio*

Käyttäjä valitsee potilastietojärjestelmän kautta eRAsta näytettävän kuvan, kun kuvien tiedot ovat haettu ja näkyvillä on kuvalistaus. Valinnan jälkeen eRAImageServer tarkistaa, onko tietojen lisäksi varsinainen kuva haettu Kuva-aineistojen arkistosta. Jos kuvaa ei ole vielä haettu, niin se haetaan. Kuvan haun jälkeen eRAImageServer generoi tilapäisen katselulinkin eRAlle. eRA avaa linkin uuteen välilehteen, jolloin katselin-komponentti aukeaa potilastietojärjestelmässä. Katselin-komponentti hakee kuvasta leikkeet näytettäväksi eRAImageServeriltä, käyttäen HTTP-pyyntöjä. Kun leikkeet on haettu, on katselin-komponentti valmiina ja käyttäjä voi selata ja katsella kuvan leikkeitä, kuten kuvan 10. sekvenssikaaviossa on esitelty.




***Kuva 10.*** *Kuvan näyttämisen sekvenssikaavio*

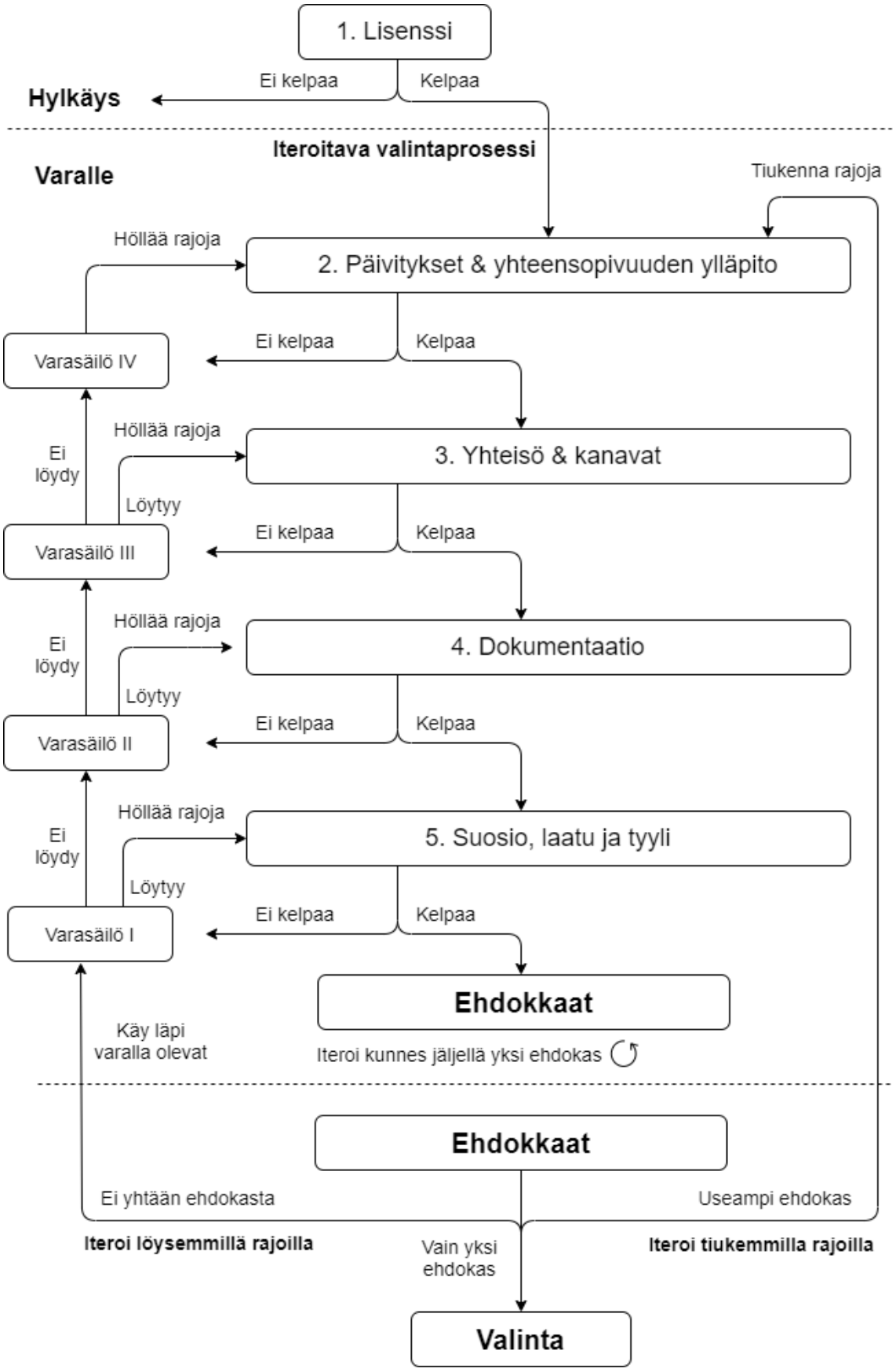
# 6. KOMPONENTIN INTEGRAATIOPROSESSI

Tässä luvussa käydään läpi työssä toteutettu avoimen lähdekoodin komponentin integraatioprosessi, alkaen komponentin valinnasta ja päättyen komponentin ylläpitämiseen. Aluksi esitellään teorian pohjalta luotu prosessi avoimen lähdekoodin komponentin valintaan ja seuraavaksi tehdään sen pohjalta integraatiossa mukana olevan HTTP-palvelin-komponentin valinta. Työssä integroidaan DICOM-katselin-komponentti osaksi olemassa olevaa Kanta-palveluita käyttävää eRA terveydenhuoltojärjestelmää. Katselin-komponentti ei ole itse suoraan yhteydessä Kuva-aineistojen arkistoon, mutta näyttää eRAImageServerin sieltä hakemia kuvia. Työn alkaessa oli haku Kanta-palveluista jo toteutettu ja toteutettavana oli katselimen integraatio.

## 6.1 Avoimen lähdekoodin komponentin valintaprosessi

Seuraavana esitellään työssä kehitetty komponenttien valintaprosessi. Prosessissa komponentin valinta tehdään erilaisten kriteerien perusteella ja niistä ensimmäisenä on aina lisenssi. Lisenssi on hylkäävä kriteeri, sillä mikäli lisenssi ei täytä käytön kannalta tarvittavia ehtoja, on komponentin hyödyntäminen mahdotonta. Lisenssiin ei pysty itse vaikuttamaan, mutta komponenttia taas on mahdollista muokata. Esimerkiksi huono dokumentaatio ei estä komponentin käyttöä kokonaan, vaikka saattaa hidastaa käyttöönottoa merkittävästi.

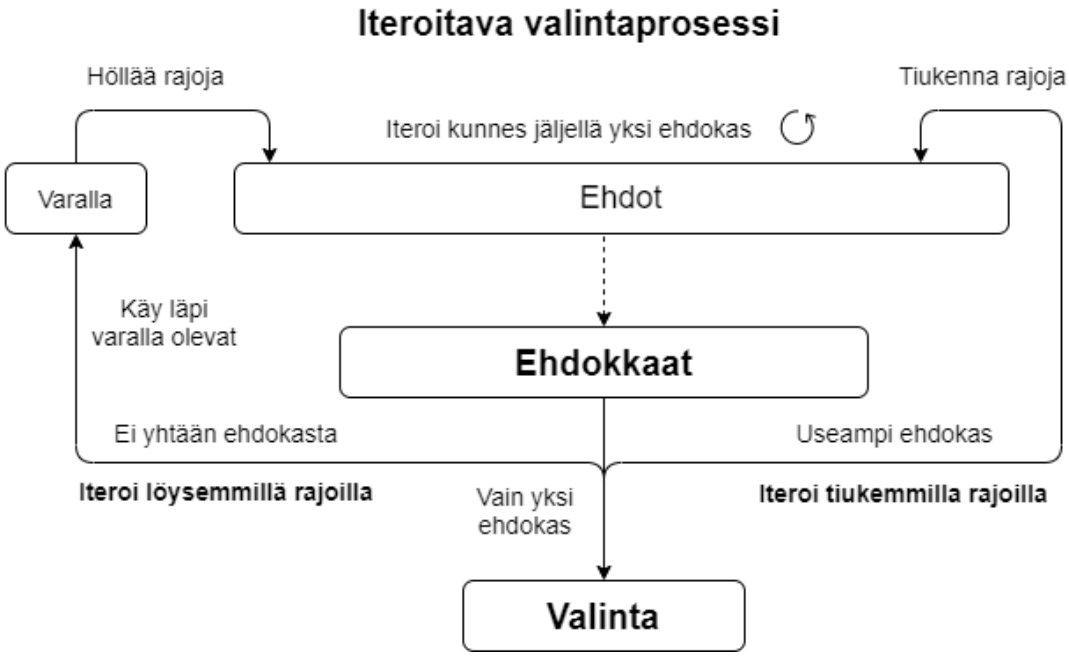
Seuraavana valintaprosessissa mietitään mitä kriteereitä valinnassa halutaan käyttää ja missä järjestyksessä. Näistä kriteereistä luodaan iteroitavan valintaprosessin aikana valintaa ohjaavia ehtoja, eikä alussa kriteerit ole tarkkoja ehtoja. Kuvassa 11 nämä kriteerit ovat 2.-5.  ja ne ovat muutettavissa ja täydennettävissä tarpeen mukaan ja niitä voi halutessaan myös lisätä tai vähentää.




***Kuva 11.*** *Komponentin valintaprosessi*

Prosessin aluksi asetetaan kriteereille ehdot, joiden perusteella voidaan määritellä, kelpaako komponentti käytettäväksi. Tämän jälkeen jokainen lisenssiehdon täyttänyt ehdokas kulkee kierroksen läpi vastaten ehtoihin. Ehdon kohdalla komponentti kelvatessaan jatkaa seuraavaan ehdolle. Jos taas komponentti ei läpäise ehtoa, laitetaan se varasäilöön talteen.

Lopulta kun kaikki komponentit on käyty läpi, iteroidaan prosessia, kunnes jäljellä on vain yksi komponentti. Mikäli mahdollisia komponentteja on enemmän kuin yksi, iteraatiota toteutetaan niin, että kierroksen läpäisseille komponenteille tehdään uusi kierros tiukennetuin ehdoin. Jos taas valintaprosessin läpäisseitä komponentteja ei ole yhtään, mennään kriteereitä läpi käänteisessä järjestyksessä, kunnes jonkin kriteerin varasäilöstä löytyy komponentteja. Näille komponenteille käydään sitten uudelleen ehtoja läpi loppua kohden löysemmillä rajoilla. Prosessia jatketaan samalla tavalla, tarvittaessa edelleen iteroiden. Valinta tehdään, kun jäljellä jää yksittäinen komponentti, joka valitaan, kuva 12.




***Kuva 12.*** *Valintaprosessin iteroitava osuus*

Valintaprosessi ei ota kantaa ehtojen määrittelyyn, vaan ne tulee jokaisen prosessia käyttävän määritellä itse. Oikeastaan ehtojen määrittely on oma prosessinsa, johon vaikuttaa aina tilanne. Tämän lisäksi ehtojen määrittelyssä auttaa kokemus niin avoimen lähdekoodin komponenttien käytöstä kuin valinnastakin. Ehtojen määrittelyssä on järkevintä edetä rauhassa pienillä kiristyksillä. Iteroiminen ja kokemus ohjaavat järkevään ehtojen kiristämiseen.

Ehtoja määritellessä kannattaa huomioida myös komponentin tarve omassa kehityksessä. Mikäli kyse on esimerkiksi vain pienestä osasta, jota ei todennäköisesti tarvitse jatkossa muuttaa, ei ehkä ole kannattavaa hylätä komponenttia laajan dokumentaation tai käyttöoppaan puuttumisen takia. Tällöin integraatio on varmasti työläämpää, mutta ei mahdotonta. Vastaavasti pieneen osaan käytettävän komponentin pitkät päivitysvälit tai yhteisön epäaktiivisuus on pieniä syitä hylkäämiseen, kun vaihtoehtona on komponentin toteuttaminen kokonaan itse.

Ehtojen asettaminen aluksi löysäksi edesauttaa kaikkien ehdokkaiden käymistä läpi ainakin kertaalleen kaikilta osa-alueilta. Näin iteraatiossa ehtoja kiristämällä tehty päätös on tiedostavampi. Jo yhdellä iteraatiokierroksella pääsee hyvin kärryille komponenteista ja niiden välisistä eroista eri osa-alueilla, koska jokaisen ehdokkaan kohdalla on jokaista kriteeriä selvitettävä ainakin jonkin verran.

Katsoessa kokonaisuutta voi komponenttien eroista löytyä jotain ratkaisevaa, jonka perusteella päätös lopulta halutaan tehdä. Komponenttien kriteerien täyttämisestä voi tehdä taulukon, jonka avulla on helpompi päättää rajoja seuraavalle iteraatiokierrokselle. Kun komponentit on kerran käyty läpi, on ne seuraavilla kierroksilla nopeampi käydä läpi ja osalla iteraatiokierroksista voi käyttää vain osittaista prosessia.

Useammalla iteraatiokierroksella voidaan myös estää valitsemasta komponenttia vain yksittäisen ehdon perusteella. Joltain komponentilta voi esimerkiksi puuttua dokumentaatio kokonaan, mutta sillä taas on hyvä päivitystaajuus. Jos alkuvaiheessa asetetaan ehdoksi lyhyt päivitysväli, voidaan päätyä valitsemaan kyseinen komponentti, jolla ei ole ollenkaan dokumentaatiota, mutta joka ainoana täytti tiukan ehdon päivityksistä.

Lopuksi kannattaa käydä valinnan järkevyys läpi. Hyvä ajatus on käydä valitun komponentin osa-alueet läpi ja tarkistaa, että niissä ei ole käytön kannalta ongelmia. Kannattaa myös katsoa mistä syistä muut komponentit on hylätty ja mikäli komponentin hylkäys mietityttää, on mahdollista iteraatioon palata uusilla ehdoilla, jos vain prosessista on tehty palaamisen mahdollistavat muistiinpanot.

## 6.2 Komponentin valinta luodun prosessin avulla

Käytännön osuudessa integroitava katselin-komponentti oli valittu jo projektin aikaisemmassa vaiheessa ennen tämän työn aloitusta, joten sen valinnassa edellä esiteltyä prosessia ei ole hyödynnetty. Integraatioon vahvasti liittyvä avoimen lähdekoodin HTTP-palvelin-komponentti sen sijaan vaihdettiin työn aikana paremmin tarkoitukseen sopivaan. Komponentti toteuttaa eRAImageServerille palvelimen, jolla

esimerkiksi katselin-komponenttia käytetään. Komponentin valinta käydään läpi seuraavaksi.

Pääsyy komponentin vaihtoon oli saada komponentin käyttämä HTTP-liikenne vaihdettua salattuun HTTPS-liikenteeseen eli SSL-tuki oli komponentille ehdoton vaatimus. Vaatimuksena komponentille oli myös, että se on avoimen lähdekoodin komponentti, jonka lisenssi ei rajoita komponentin käyttämistä kaupallisessa sovelluksessa, eikä saastuta projektin lisenssiä.

Mahdollisia komponenttiehdokkaita valinnan iteraatioprosessiin löytyi MIT lisenssillä varustetut Nancy [42] ja Embedio [43] sekä Grapevine [44], jossa on Apache lisenssi. Valinnassa päädyttiin käyttämään valintaprosessin mukana esiteltyjä kriteerejä ja järjestystä sellaisenaan, jolloin ensimmäinen kriteeri oli ”Päivitykset & yhteensopivuuden ylläpito”. Kriteerin ehdoksi määriteltiin, että komponentin piti olla päivittynyt viimeisen puolen vuoden sisään ja päivityksiä tuli olla kaiken kaikkiaan ainakin viisi. Päivittyvyys oli komponentin valintaan annetuissa ohjeissa korostettu ehto, joten ehtoa ei voitu alkuun määritellä kovin löysästi.

Komponenteista ensimmäinen, Nancy ei läpäissyt ehtoa vaan päätyi tässä vaiheessa varalle. Embedio taas läpäisi ehdon kirkkaasti, koska se oli saanut useita päivityksiä viimeisen parin kuukauden aikana ja oli muutenkin päivittynyt tiheään. Versionumeroinnin perusteella päivityksissä ei ollut kuitenkaan aina säilynyt yhteensopivuus. Grapevine läpäisi ehdon niukasti, sillä edellisestä päivityksestä oli rajaksi määritelty puoli vuotta.

Seurana kriteerinä oli ”Yhteisö & kanavat”, jonka ehdoksi määriteltiin, että hakemalla komponentin teknologialla, nimellä ja eräällä ohjelmointiyhteisöllä tulisi netistä löytyä joitain hakutuloksia. Tämän ehdon molemmat komponenteista läpäisivät. Tässä vaiheessa ei haluttu asettaa tiukkoja ehtoja, sillä vaihtoehtoja oli jäljellä vain kaksi ja iteroimalla saa kyllä valittua lopulta paremman. Pitämällä rajat löysinä, haluttiin käydä molemmat komponentit läpi ja tehdä valinta kokonaisvaltaisemmin.

Kriteerille ”Dokumentaatio” asetettiin ehdoksi, että komponentin käytöstä pitäisi löytyä helppo esimerkki ja lisäksi jotain hieman spesifimpiä tilanteita. Sellaiset löytyi molemmista komponenteista, mutta Embediossa dokumentaatio oli selkeästi laajempi. Kriteeriä ”Suosio, laatu ja tyyli” ei otettu ollenkaan mukaan, koska kun ehdokaskomponentteja oli niin vähän, että sitä ei haluttu painottaa valinnassa ollenkaan. Ensimmäiseltä iteraatiokierrokselta siis kaksi ehdokasta selviytyi loppuun asti, Embedio ja Grapevine. Näistä ensimmäinen oli ollut parempi sekä päivitysten, että

dokumentaation suhteen, kun taas jälkimmäinen ei ollut osoittautunut edukseen missään kriteerissä.

Iteraation seuraavalla kierroksella määriteltiin aluksi tiukemmat rajat päivitysten tiheydelle, jolloin Embedio valikoitui komponenteista käytettäväksi. Itseasiassa sama valinta olisi tapahtunut myös dokumentaatiokriteeriä tiukentamalla. Lopputulosta tarkastellessa pääteltiin, että prosessi oli tuottanut toivotun tuloksen, eikä iteraatioon tarvinnut palata, sillä valinta oli selkeästi perusteltu.

Kun komponentteja on jäljellä kaksi, joista toinen komponentti on useammalla osa-alueella parempi ja toinen ei yhdelläkään, on valinta helppo ja selkeä, kuten esittelemässäni valinnassa. Aina ei kuitenkaan ole näin selkeä tilanne, jolloin iteraatio ja kriteerien painotus korostuu. Mikäli valinta yhtään epäilyttää, kannattaa iteraatio tehdä maltillisesti.

## 6.3 Katselin-komponentin käyttöönotto ja muutokset

Tässä luvussa käydään läpi työn käytännön osuudessa toteutettu avoimen lähdekoodin DICOM-katselimen integraatio ja sen vaiheet, sekä integraatiossa katselin-komponenttiin tehtyjä muutoksia. Luvussa käydään myös läpi komponentin hyödyntämisessä ilmenneitä käytännön ongelmia ja teoriaosuudessa selvinneiden asioiden hyödyntämistä käytännössä.

Lähtötilanteessa eRAImageServerin ja OHIF-viewerin aikaisemman 1.0 version integraatio oli aikoinaan jo toteutettu. Katselin-komponenttiin oli tehty muutoksia, joiden avulla esim. potilastiedot oli piilotettu käytettävistä osoitteista ja lokalisaatiota oli toteutettu. Katselin-komponentti oli toiminut eRAImageServerin vanhan version kanssa, mutta toteutus oli jo yli puolitoista vuotta vanha. eRAImageServeriin oli ehtinyt tulla päivityksiä, joten ei ollut selvää pitäisikö katselimen tällä hetkellä vieläkin toimia tai minkä version kanssa integraatio oli toteutettu.

Kun projektin osat saatiin ajoon, ei integraatio eRAImageServeriin täysin toiminut. Katselin-komponentti avautui eRAImageServerin avulla, mutta kuvien hakeminen ei onnistunut. Tutkimalla eRAImageServeriä ja käyttämällä selaimen kehitystyökaluja, pystyttiin selvittämään, millaista liikennettä integraatiossa komponenttien välillä kulki. OHIF:n demopalvelimella taas pyöri uudempi 2.0 versio, jonka avulla pääteltiin, kuinka uudemmassa versiossa kuvien hakeminen tapahtui. Tämän lisäksi eRAImageServerin vanha toteutus antoi suuntaa, millaisena se oli alun perin toiminut.

Katselin-komponentista on olemassa vain löyhä dokumentaatio, joka olettaa katselimen integroitumista PACS:iin. Dokumentaatiossa ei myöskään avata kuvien hakutoimintaa,

tai kerrota mitä osia DICOM-standardista toteutus hyödyntää, joten siitä ei ollut apua integraatiossa. Koska integraatioon oli yritetty tutustua, katselimeen tehtyjen muutoksien syitä selvitetty sekä integraatiota yritetty saada tuloksetta toimimaan päätettiin vaihtaa käyttämään OHIF-viewerin uudempaa 2.0 versiota, johon olisi kuitenkin jossain vaiheessa siirrytty.

Aluksi haluttiin varmistua, että kuvien haku toimisi sellaisenaan, joten eRAImageServerille tehtiin testimielessä muutokset, joiden avulla se luovutti yksittäisen kuvan metatiedot ja kuvan, kun niitä tietyn muotoisella osoitteella kysyttiin. Tähän löytyi apua OHIF-viewerin demopalvelimelta, kehitystyökalujen avulla.

Kun kuvien haku saatiin sellaisenaan toimimaan ja potilaiden tunnisteet yritettiin vaihtaa pois, törmättiin uuteen ongelmaan. Ketju metatiedoista kuvien hakemiseen ei onnistunut, jos kuvassa oli useampia kuin yksi leike. Tämä kuitenkin ratkesi, kuvan metadatan hakua hieman muuttamalla. Tämän jälkeen tarvittavat omat muutokset, kuten potilastunnisteiden piilotus tehtiin ja ne toimivat myös useamman leikkeen kuvilla, jolloin integraatio oli testaamista vaille valmis.

## 6.4 Katselimen jatkokehitys ja päivitykset

Työn suorittamisen aikana esiin nousi kysymys, että miten katselin-komponentin päivitykset hoidetaan. Ongelmallisemman tilanteesta teki, että komponenttia oli räätälöity. Näin oli mahdollista, että kun komponenttia oltiin päivittämässä, jotkin päivitykset kohdistuisivat juuri räätälöityihin kohtiin ja olisivat päällekkäisiä.

Onnistuneeseen päivittämiseen piti aluksi ottaa päivitykset mukaan ja sen jälkeen toteutusta oli muokattava edelleen, että omat muutokset säilyivät käytössä. Tulevien päivityksen käyttöönoton kannalta päädyttiin, että tämä oli kriittistä tehdä, koska uusin version haluttiin pitää integraation käytössä.

Päivitysten mukana myös komponentin kuvien hakemisen logiikka muuttui. Varsinaisesti tämä ei kuitenkaan aiheuttanut itse päivittämisen aiheuttavaa ongelmaa, sillä ongelma huomattiin jo etukäteen. Ongelma löytyi teoria osuudessa esitetyllä tavalla, tutkimalla muutoksia. Itse muutokset aiheuttivat työtä, mutta päivittäminen oli helppoa, kun ongelma huomattiin ajoissa.

Päivittämisessä hyödynnettiin myös toista teoriaosuudessa esiin nostettua keinoa, testausta, ongelmien välttämiseksi. Automaattisia testejä projektissa ei ollut käytössä, mutta testausta oli helppo painottaa muutoksiin käyttämällä testitapauksia, jotka keskittyivät kuvien hakemiseen ja näin ollen suurella todennäköisyydellä toivat esiin

ongelmia. Erityisen tärkeää oli testata kuvien avausta erilaisilla kuvilla ja kuvasarjoilla, jolloin myös erikoisempien tapausten ongelmat nousivat esiin.

Tulevaisuudessa päivityksiin varaudutaan niin, että tietyin väliajoin käydään läpi komponentin kehityshaaran päivitykset ja katsotaan niiden tarpeellisuus projektin näkökulmasta. Mikäli päivittäminen koetaan tarpeelliseksi, selvitetään kuinka paljon siihen mennessä tulleet päivitykset ovat ristiriidassa oman toteutuksen kanssa.

Tämän jälkeen tehdään arvio, millaisen työmäärän päivittämisen yhteydessä tehtävät muutokset vaatisivat ja olisiko päivittäminen järkevää. Jos päivittämiseen päädytään, päivitykset otetaan mukaan Git-versionhallinnalla teoriassa esitellyllä tavalla. Päivityksiä mukaan ottaessa, myös yhdistämisvaiheessa muutoksia tarkastellaan, koska silloin on mahdollista löytää helposti ongelmia aiheuttavia muutoksia lähdekoodista.

Päivitysten jälkeen toteutusta testaamalla varmistetaan toteutuksen toiminta. Etenkin erikoistapauksia pyritään painottamaan ja testitapaukset dokumentoidaan, jotta testaus on jatkossakin yhtä kattavaa. Tulevaisuudessa saatetaan mukaan ottaa automaattisia testejä, joiden avulla mahdollistuu päivittäminen useammin, koska työmäärä pienenee.

Myös automaattisten regressiotestien luomista mietittiin, mutta niistä jouduttiin luopuman, koska työmäärä testien toteuttamisessa ja ylläpitämisessä olisi isompi kuin manuaalisessa testaamisessa. Kuten teoriaosuudessa käsiteltiin, on automaattisten regressiotestien ylläpitäminen työlästä integraatiossa käytettävän komponentin kanssa, koska komponentti määrittää katselimen käyttöliittymää.

Kehityksen aikana myös huomattiin, että katselin-komponentin käyttöliittymä ehti päivittyä useamman kerran, mikä olisi tehnyt automaattisista testeistä vanhentuneita. Vanhentuneet testit pitäisi päivittää ennen kuin niistä on taas hyötyä. Regressiotestien lisäksi toteutusta pitäisi muutenkin testata myös manuaalisesti jonkin verran, sillä integraatio on laaja ja kaiken testaavat testit on mahdotonta tehdä.

## 6.5 Käytännön osuuden yhteenveto

Käytännön osuudessa toteutettiin integraatio, jossa tuotiin esiin ja hyödynnettiin teoriaosuudessa käsiteltyjä asioita. Integraatiossa hyödynnettiin avoimen lähdekoodin komponenttia, joka otettiin käyttöön eRA-palveluihin, jotka edelleen integroituvat Kanta-palveluihin.

Integraatiossa käytettävien järjestelmien esittelyn jälkeen käsiteltiin työssä luotua valintaprosessia, joka on tarkoitettu avoimen lähdekoodin komponentin valintaan. Prosessi hyödyntää yksinkertaista iteraatiorunkoa, jossa ehdokaskomponentteja

käydään läpi eri kriteerien avulla ja iteraatiokierroksilla ehtoja tiukennetaan, tavoitteena on valita komponenteista perustellusti paras vaihtoehto. Työssä esiteltiin prosessin runko, mutta varsinaisten ehtojen määrittely on aina tapauskohtaista.

Ehtojen määrittely vaikuttaa kuitenkin lopputulokseen, ja maltillisella ehtojen kiristämisellä ja iteraation noudattamisella pääsee todennäköisesti hyvään lopputulokseen. Valintaprosessista esiteltiin myös esimerkki, jossa valittiin HTTP-palvelin komponentti. Esimerkkivalinnassa nousi hyvin esiin prosessille tyypillisiä piirteitä, kuten löysien alkuehtojen tärkeys integraation kannalta, sekä ensimmäisen kierroksen perusteella päivitettävien ehtojen merkittävyys.

Jo käytännön työn alkuvaiheessa törmättiin ongelmiin katselin-komponentin päivitysten kanssa, kun haluttiin ottaa käyttöön päivitys, joka sisälsi uudemman logiikan kuvien hausta. Muitakin päivityksiä oli siihen mennessä tullut ja ne kaikki päädyttiin ottamaan mukaan, ilman päivitysten tarkempaa valikointia. Päivittäminen tarkoitti, että katselin-komponenttia tuli muokata uusilta osilta lisää, jolloin kuviin liittyvät tunnisteet saatiin pidettyä edelleen piilossa. Uusien päivitysten tutkimista helpotti OHIF-vieweristä löytyvä demopalvelin, jolle uusin versio komponentista päivittyy. Demon avulla esimerkiksi päivitettyä kuvienhakulogiikkaa oli helpompi tutkia.

Myös käytännön osuus osoitti, että päivittämistä ennen tehtävä päivitysten sisällön tarkasteleminen on hyödyllistä, työssä se tehtiin pääasiassa demopalvelimen avulla. Päivityksiä tarkastellessa kannattaa olla luova, koska jos jostain pystyy päättelemään päivitysten tuomia ongelmakohtia, on siitä aina hyötyä.

Käytännön osuudessa tehty komponentin päivittäminen on hyvä esimerkki, kuinka ongelmakohdan löytämällä ja ongelmia ennakoimalla on mahdollista säästää resursseja. Toisaalta myös testaaminen on tärkeää ja käytännön osuuden tilanteessa oli oikeastaan kyse juuri ennakoivasta testaamisesta, joka täydentyi korjauksen jälkeen myös varsinaisella päivityksen testaamisella.

# 7. JOHTOPÄÄTÖKSET

Työssä käsiteltiin avoimen lähdekoodin komponenttien ylläpitämistä ja päivittämistä. Käsittely tapahtui erityisesti näkökulmasta, jossa komponentista on käytössä jokin komponentin julkaisusta hieman muokattu versio. Näin ylläpitäessä on huolehdittava, että tehdyt muutokset säilyvät toimivina jatkossakin ja, että komponentin normaali käyttö ei häiriinny muutoksista. Työssä löydettiin erilaisia toimintatapoja, joiden avulla ylläpitämistä näissä tilanteissa voidaan helpottaa.

Työssä kehitettiin ja esiteltiin myös valintaprosessi, jolla avoimen lähdekoodin komponentin valinta voidaan hoitaa, kun ehdokkaita on useampi. Valintaprosessi on työn ulkopuolella uudelleen käytettävissä oleva prosessi, joka soveltuu kaikenlaisten avoimen lähdekoodin komponenttien valintaan. Prosessi perustuu iteraatioon ja siinä komponentit käyvät läpi eri kriteereille määriteltyjä ehtoja. Prosessissa käytettävät ehdot ja kriteerit ovat käyttötapauksen perusteella muokattavissa. Työssä on myös esitelty yleiskäyttöiset kriteerit, joista valintaehtoja muodostetaan.

Prosessissa käydään kaikkia ehdokkaita samanaikaisesti läpi ja vastataan kriteeri kerrallaan komponentille määriteltyyn riittävyysehtoon. Kierroksessa käydään kaikki kriteerit läpi ja ehdokkaat päätyvät joko jatkoon tai varalle sen mukaan, kuinka ne täyttävät kriteerin riittävyysehdot. Prosessin kierroksia iteroidaan palaamalla kriteereihin uudelleen, kunnes jäljellä on yksi komponentti, joka valitaan. Aluksi riittävyysehdot ovat löysempiä, mutta niitä kiristetään loppua kohden. Ehtojen rauhallinen kiristäminen takaa, että iteraatiossa huomioidaan laajasti komponentteja, mutta keskitytään kuitenkin eniten niihin ominaisuuksiin, joita komponentin valitsija arvostaa eniten. Prosessia esiteltiin myös esimerkkivalinnan muodossa.

Komponentin ylläpitämisen helpottamiseksi kannattaa sen päivittämisestä tehdä suunnitelma. Suunnitelma voi esimerkiksi määrittää päivittämistahdin, millaisia päivityksiä priorisoidaan, ottaa kantaa versionhallinnan käyttötapoihin päivittämisessä tai määritellä testaamista päivittämisen jälkeen. Suunnitelman lisäksi päivitysten läpikäymisellä ja sisällön tuntemisella on mahdollista estää ongelmia ja helpottaa niiden ratkaisemista.

Kun komponentti on käytössä, kannattaa seurata sen kehitystä oman toteutuksen ulkopuolella. Komponentin saadessa uusia päivityksiä on hyötyä, jos sen päivittämisestä on tehty suunnitelma, jonka perusteella voidaan tehdä päätös päivittämisestä tai päivittämättä jättämisestä. Kun päivitykset päätetään ottaa toteutukseen mukaan voi

versionhallinnan järkevällä käyttämisellä pienentää käyttöönoton työmäärää ja samalla pystyy tarkastamaan omien muutosten ristiriitaisuutta.

Päivitysten käyttöönottoa omaan kehitykseen voidaan helpottaa myös versionhallinnalla. Git-versionhallinnan rebase-toiminto muuttaa muutosten näennäistä järjestystä ja selventää versionhallinnan historiaa sekä helpottaa yhdistämistä. Kun omassa kehityksessä tehtyjen muutosten jälkeen komponentin kehityshaaraan on tullut päivityksiä, rebase-toiminnolla muutokset haetaan mukaan. Yhdistäminen tapahtuu kuin komponentin päivitys olisi pohjalla ja omat muutokset tehty viimeisenä, jolloin niiden käsittely on selkeämpää. Rebase-toiminto auttaa teknisessä mielessä, mutta ei poista riskiä muutoksien aiheuttamista ongelmista eikä tarvetta päivitysten käyttöönoton jälkeiselle testaukselle.

Omien muutoksien lisäksi päivittäminen on erityisen haastavaa, jos päivittyvä versio ei ole takaisinpäin yhteensopiva. Näin tulee tilanne, jossa joidenkin toimintojen käyttämistä pitää muokata päivityksen yhteydessä. Myös komponentin logiikan muuttuminen voi johtaa vastaavaan tilanteeseen. Koska näiltä ongelmilta voi olla hankala välttyä, tarjotaan työssä niiden ratkaisuun kahta toimintatapaa.

Ensimmäisenä ja ennaltaehkäisevänä keinona on päivitysten tarkka läpikäyminen, joka sisältää niin päivityksiin liittyvät muutoslokit kuin varsinaisen lähdekoodin muutokset. Päivittäessä kannattaa olla mahdollisimman tietoinen mitä muutoksia päivitykset sisältävät. Ongelmallisia muutoksia voi olla esimerkiksi toiminnallisuuden logiikan muuttuminen tai jonkin toiminnon kokonaan käytöstä poistuminen.

Toinen toimintatapa on toteutuksen testaaminen ja se pyrkii paljastamaan toteutukseen jo päätyneitä ongelmia. Päivitysten mukaan saaminen ongelmitta ei vielä tarkoita, että kaikilta ongelmilta on varmasti vältytty. Ohjelma saattaa kääntyä ja kaikki näyttää olevan kunnossa, mutta muutokset voi vaikuttaa johonkin tilanteeseen, jossa toteutus ei enää toimi toivotusti. Testaaminen kannattaa hoitaa mahdollisimman laajasti ja lopputulosta parantaa, jos testaus huomioi kohdat mihin komponentin omat muutokset kohdistuvat ja käytön erikoistapaukset.

Työssä on annettu esimerkki päivittämiseen liittyvästä piilevästä ongelmasta ja osoitettu, että tuollainen ongelma ei tarvitse komponenttiin omia muutoksia, eikä rajoitu vain avoimen lähdekoodin komponenttien käyttöön. Yleisesti tällaisten ongelmien todennäköisyys kasvaa käytettävän komponentin koon ja käytön sekä testien vähäisyyden mukaan ja päivittäminen ilman päivitysten sisällön tuntemista edesauttaa näiden ongelmien ilmaantumista.

Avoimen lähdekoodin komponentin ylläpitämiseen ja päivittämiseen voidaan vaikuttaa yksinkertaisesti jo komponentin valinnalla. Komponentteja valittaessa kannattaa pyrkiä, että niitä ei tarvitsisi muokata, koska silloin niiden päivittäminen on huomattavasti helpompaa. Aina muokkaamiselta ei kuitenkaan voida välttyä, jolloin muutokset kannattaa yrittää minimoida.

Avoimen lähdekoodin komponentin ylläpitämistä ja päivittämistä esitettiin työssä myös käytännön integraatiolla. Integraatiossa otettiin käyttöön DICOM-kuvien katselin-komponentti, johon tehtiin myös integraatiota varten omia muutoksia. Työssä esiteltiin muutoksia vaatineen komponentin päivittämistä ja ongelmatilanteiden ratkaisemista.

# LÄHTEET


[1] Reynolds, C. J., & Wyatt, J. C. (2011). Open source, open standards, and health care information systems. *Journal of medical Internet research*, *13*(1), e24.

[2] Midha, V., Singh, R., Palvia, P., & Kshetri, N. (2010). Improving open source software maintenance. *Journal of Computer Information Systems*, *50*(3), 81-90.

[3] Sen, R., Subramaniam, C., & Nelson, M. L. (2008). Determinants of the choice of open source software license. *Journal of Management Information Systems*, *25*(3), 207-240.

[4] Lindman, J., Paajanen, A., & Rossi, M. (2010, September). Choosing an open source software license in commercial context: A managerial perspective. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 237-244). IEEE.

[5] Baytiyeh, H., & Pfaffman, J. (2010). Open source software: A community of altruists. *Computers in Human Behavior*, *26*(6), 1345-1354.

[6] Oreg, S., & Nov, O. (2008). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in human behavior*, *24*(5), 2055-2073.

[7] Andersen-Gott, M., Ghinea, G., & Bygstad, B. (2012). Why do commercial companies contribute to open source software?. *International journal of information management*, *32*(2), 106-117.

[8] Lakhani, K. R., & Wolf, R. G. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects.

[9] Increased security through open source https://dl-acm-org.libproxy.tuni.fi/citation.cfm?id=1188921 Hoepman, J. H., & Jacobs, B. (2007). Increased security through open source. *Communications of the ACM*, *50*(1), 79-83.

[10] Spinellis, D. (2011). Choosing and using open source components. *IEEE software*, *28*(3), 96-96.

[11] Git and Other Systems - Git as a Client, Pro Git book Saatavissa (viitattu 5.3.2020): https://git-scm.com/book/en/v2/Git-and-Other-Systems-Git-as-a-Client

[12] Guidance documet Medical Devices, European Commission, 2020. Saatavissa (viitattu 6.3.2020) https://ec.europa.eu/docsroom/documents/17921/attachments/1/translations/en/renditions/pdf

[13] EUR-Lex, EUROOPAN PARLAMENTIN JA NEUVOSTON ASETUS (EU) 2017/745 Saatavissa (viitattu 3.4.2020): https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:32017R0745&from=FI

[14] Ohjeasiakirja Itsenäisten ohjelmistojen määrittely ja luokittelu, Euroopan komissio, 2012. Saatavissa (viitattu 3.4.2020): https://www.fimea.fi/documents/160140/764653/sw_luokitteluohje_2012-03-13.pdf/0cce0166-d286-369a-5122-b2c9d0967ca2?t=1578988385487

[15] Notified bodies, European Comission, 2020. Saatavissa (viitattu 16.3.2020) https://ec.europa.eu/growth/single-market/goods/building-blocks/notified-bodies_en

[16] ISO 13485:2016(en), ISO Saatavissa (viitattu 3.4.2020): https://www.iso.org/obp/ui#iso:std:iso:13485:ed-3:v1:en

[17] HL7 Finland, 2020. Saatavissa (viitattu 8.1.2020): http://www.hl7.fi/sig-toiminta/ihe-sig/

[18] Kuva-aineistojen arkisto (Kvarkki) tekninen määrittely, Kanta, 2018. Saatavissa (viitattu 8.1.2020): https://www.kanta.fi/documents/20143/108477/Kuva-aineistojen+arkisto+tekninen+m%C3%A4%C3%A4rittely.pdf/978528a4-396d-9da8-f202-50151acdc820

[19] About HL7, HL7 International, 2020. Saatavissa (viitattu 28.2.2020): http://www.hl7.org/about/index.cfm?ref=nav

[20] CDA® Release 2, HL7 International, 2010. Saatavissa (viitattu 28.2.2020): http://www.hl7.org/implement/standards/product_brief.cfm?product_id=7

[21] Lääketieteellisen kuvantamisen kansalliset toiminnalliset määrittelyt, THL, 2019. Saatavissa (viitattu 8.1.2020): http://www.julkari.fi/bitstream/handle/10024/138508/Valtakunnallinen%20terveydenhuollon%20kuva-aineistojen%20arkisto_Toiminnallinen%20maarittely%20versio%201_4%20THL.pdf?sequence=1&isAllowed=y

[22] Pianykh, O. S. (2012). What is DICOM?. In *Digital Imaging and Communications in Medicine (DICOM)* (pp. 3-5). Springer, Berlin, Heidelberg.

[23] About DICOM, Saatavissa (viitattu 28.2.2020): https://www.dicomstandard.org/about/

[24] Tutustu Kansalliseen Terveysarkistoon verkossa, KELA, 2009. Saatavissa (viitattu 13.12.2019): https://www.kela.fi/-/tutustu-kansalliseen-terveysarkistoon-verkossa

[25] Mitä Kanta-palvelut ovat, Kanta, 2019. Saatavissa (viitattu 13.12.2019): https://www.kanta.fi/mita-kanta-palvelut-ovat

[26] Anssi Haikonen. 2017. Potilastietojen standardointi ja siirto kansalliseen sähköiseen arkistoon. Diplomityö. Tampereen teknillinen yliopisto.

[27] Sosiaalihuollon asiakastiedon arkisto, Kanta, 2020. Saatavissa (viitattu 16.12.2019): https://www.kanta.fi/ammattilaiset/sosiaalihuollon-asiakastiedon-arkisto

[28] Terveydenhuollon todistusten sähköinen välitys nopeuttaa palvelua, Kanta, 2019. Saatavissa (viitattu 16.12.2019): https://www.kanta.fi/blogi/-

/asset_publisher/1QjC602jKPR6/content/terveydenhuollon-todistusten-sahkoinen-valitys-nopeuttaa-palvelua

[29] Kelain, Kanta, 2020. Saatavissa (Viitattu 10.3.2020): https://www.kanta.fi/ammattilaiset/kelain

[30] Kuva-aineistojen arkisto, Kanta, 2020. Järjestelmäkehittäjille Saatavissa (viitattu 13.12.2019): https://www.kanta.fi/jarjestelmakehittajat/kuva-aineistojen-arkisto

[31] Kuva-aineistojen arkisto, Kanta, 2020. Saatavissa (viitattu 13.12.2019): https://www.kanta.fi/ammattilaiset/kuva-aineistojen-arkisto

[32] Kuva-aineistojen arkisto, THL, 2019. Saatavissa (viitattu 16.12.2019): https://thl.fi/fi/web/tiedonhallinta-sosiaali-ja-terveysalalla/kanta-palvelut/terveydenhuollon-kanta-palvelut/kuva-aineistojen-arkisto

[33] Kanta-palvelujen Kuva-aineistojen arkisto on valmis otettavaksi käyttöön, Kanta, 2018. Saatavissa (13.12.2019): https://www.kanta.fi/tiedote/-/asset_publisher/cf6QCnduV1x6/content/kanta-palvelujen-kuva-aineistojen-arkisto-on-valmis-otettavaksi-kayttoon-

[34] eRA-palvelut, Atostek, 2020. Saatavissa (viitattu 13.12.2019): https://www.atostek.com/era/

[35] Lauri Koriseva. 2018. Jatkuvan toimituksen käyttöönotto ohjelmistoprojektissa. Diplomityö. Tampereen teknillinen yliopisto.

[36] Open Health Imaging Foundation, OHIF, 2017 Saatavissa (viitattu 20.12.2019): http://ohif.org/

[37] OHIF Viewer Introduction, OHIF. Saatavissa (viitattu 20.12.2019): https://docs.ohif.org/

[38] Chang, Y. H., Foley, P., Azimi, V., Borkar, R., & Lefman, J. (2016). Primer for image informatics in personalized medicine. *Procedia engineering*, *159*, 58-65.

[39] OHIF Viewer, Version 1.0.0 Introduction, OHIF. Saatavissa (viitattu 5.3.2020): https://docs.ohif.org/history/v1/

[40] OHIF Viewer, Architecture, OHIF. Saatavissa (viitattu 5.3.2020): https://docs.ohif.org/architecture/

[41] OHIF Viewer, Continous Integration (CI) Saatavissa (viitattu 5.3.2020): https://docs.ohif.org/development/continous-integration.html

[42] Nancy, 2010. Saatavissa (viitattu 20.2.2020) http://nancyfx.org/

[43] Embedio, Github Saatavissa (viitattu 20.2.2020): https://github.com/unosquare/embedio

[44] Grapevine, Github Saatavissa (viitattu 20.2.2020): https://github.com/sukona/Grapevine