Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Konsta Ikonen
Henrik Hokkanen

Testiautomaation suunnittelu potilasmonitorin ohjelmiston validaatiolle

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tieto- ja viestintätekniikka
Insinöörityö
5.5.2021

Tekijä Otsikko Sivumäärä Aika	Konsta Ikonen, Henrik Hokkanen Testiautomaation suunnittelu potilasmonitorin ohjelmiston validaatiolle 56 sivua 5.5.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Hyvinvointi- ja terveysteknologia
Ohjaajat	Esa Kähkönen, Software Engineering Manager Mikael Soini, Yliopettaja

Insinöörityön tavoitteena oli suunnitella testiautomaation toteuttaminen potilasmonitorin ohjelmiston validaatiolle Robot Framework -testiautomaatiokehyksellä. Projektin kohteena olevalle potilasmonitorin ohjelmistolle tehdään säännöllisiä päivityksiä esimerkiksi tietoturvaan liittyen. Ohjelmiston validaatio tulee tehdä aina, kun markkinoilla olevalle ohjelmistolle tehdään muutoksia. Ohjelmiston validaatio on ennalta määritelty testikokonaisuus, jonka tarkoituksena on kattaa järjestelmän kriittiset toiminnot muun kattavan verifikaation lisäksi myös järjestelmän niiltä osa-alueilta, mihin muutoksia ei ole tehty.

Työ rajattiin sisältämään testiautomaatiolla suoritettavan ohjelmiston validaation testiryhmän valinta, näille testitapauksille ja muille testitiedostoille tehtävät muutokset sekä suunnitelma testiautomaatioympäristöstä, jossa testiryhmä pystytään suorittamaan. Testiryhmällä tarkoitetaan kaikkia erillisiä testitapauksia, jotka suoritetaan ohjelmiston validaatiossa. Tällä hetkellä työn kohteena olevaa potilasmonitoriohjelmistoa ei testata lainkaan testiautomaatiolla. Toisen potilasmonitoriprojektin testiautomaatiokokonaisuutta sekä valittua ohjelmiston validaation testiryhmää käytettiin lähteenä testiautomaation suunnitteluun ja tähän projektiin viitataan tekstissä lähdeprojektina.

Projektin aikana selvitettiin käytettävän lähde- sekä kohdepotilasmonitoriohjelmiston keskeiset eroavaisuudet ohjelmiston validaatiossa testattavien ominaisuuksien sekä vaatimuksien kannalta. Tämän pohjalta valittiin testiryhmä, jota voidaan kustannustehokkaasti hyödyntää kohdeohjelmistolle. Testiautomaation suorittamiseen työn kohteena olevalle potilasmonitorille vaaditaan tuki testiautomaatiota varten, mikä tullaan toteuttamaan tulevaisuudessa.

Työn lopputuloksena saatiin kattava raportti tarvittavista toimenpiteistä testiautomaation toteuttamiseen potilasmonitoriohjelmistolle sekä ohjelmiston validaation testiryhmä, jonka avulla verifioidaan ohjelmiston kriittisimmät toiminnot. Testiautomaation vuoksi pystytään säästämään jopa 37 päivän verifikaatiotyö yhden suorituskerran aikana. Testiryhmälle suunniteltiin testiautomaatioympäristö, jossa kaikki ohjelmiston validaation testiautomaatiotapaukset pystytään suorittamaan ilman manuaalista laitteiston vaihtoa testien aikana.

Avainsanat	Testiautomaatio, Robot Framework, Potilasmonitori, Ohjelmiston validaatio, FDA

Authors Title Number of Pages Date	Konsta Ikonen, Henrik Hokkanen Test Automation Plan for Patient Monitor Software Validation 56 pages 5 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Health Technology
Instructors	Esa Kähkönen, Software Engineering Manager Mikael Soini, Principal Lecturer

The goal of the thesis project was to create a test automation plan for patient monitor software validation with Robot Framework. The target patient monitor software has regular updates for instance to improve cyber security. For each update, a software validation is conducted before release. The software validation consists of a predetermined test set intended to cover all critical functions of the system also from system areas not affected by the changes. The software validation is performed in addition to a comprehensive verification during software development.

The project was limited to include a test set used in automated software validation, recognizing changes and modifications needed for the test files and a plan for the test automation environment. The test set refers to all test cases executed as part of software validation. Currently there is no test automation implemented for the target patient monitor software. The test automation files, and software validation test set of another patient monitor software were used as a source for planning the test automation and is referred to in the text as a source project.

During the project, the main feature and requirement differences between the source and the target patient monitor software were investigated. Based on this, a test set was selected which can be cost-effectively utilized for the target patient monitor software. Support is required to perform the test automation on the patient monitor. The support will be implemented in the future and is not part of this the scope of the present study.

The outcome of the study was a comprehensive report for the necessary modifications to implement the test automation for the patient monitor software, as well as the software validation test set to verify the most critical functions of the software. With test automation, it is possible to save up to 37 days of verification work during one execution. A test automation lab was designed to perform the test set where all software validation test cases can be performed automatically without any manual changes with test equipment used during the tests.

Keywords	Test automation, Robot Framework, Patient monitor, Software validation, FDA

Sisällys

Lyhenteet ja käsitteet

ATC	Automated Test Case. Automatisoitu testitapaus.
CSV	Comma separated values. Tiedostomuoto, jolla tallennetaan yksinkertaista taulukkomuotoista tietoa tekstitiedostoon.
FDA	Food and Drug Administration. Yhdysvaltain elintarvike- ja lääkevirasto.
GE	General Electric. Yhdysvaltalainen monialayritys.
Gitlab	Verkkopalvelu tiedostojen versionhallintaan.
EEG	Elektroenkefalografia. Aivosähkökäyrä on menetelmä aivojen sähköisen toiminnan mittaamiseen.
EKG	Elektrokardiogrammi. Sydänsähkökäyrä on sydämen toimintaan liittyviä sähköimpulsseja ja sitä kautta sydämen toimintaa kuvaava käyrä.
HTML	Hypertext Markup Language. Avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä.
IP	Invasiivipaine. Invasiivinen eli kajoava verenpaineen mittaus verisuoneen asetetun kanyylin avulla.
IP-osoite	Internet Protocol -osoite. Tietokoneen tai muun verkkoon liitetyn laitteen osoite, jolla laite tunnistetaan verkossa.
JSON	Java-script Object Notation. Avoimen standardin tiedostomuoto tiedonvälitykseen.
Ohjelmiston validaatio	FDA:n määrittelemä testi lääkinnällisille laitteille, osa Design Control -prosessia. Ennalta määritelty testiryhmä, joka verifioidaan ennen ohjelmiston julkaisua ohjelmistoon tehtyjen muutosten jälkeen.

reST	Re-structured Text. Tiedostomuoto tekstiedostoille pääasiassa Python-ohjelmointikielellä.
SPI	Surgical Pleth Index. Parametri, joka pulssioksimetrian sykeaallon amplitudin ja sydämen syketaajuuden avulla mittaa potilaan anestesiatilaa.
SpO2	Oxygen saturation. Ääreisverenkierron happisaturaatio eli happikylläisyys.
TAF	Test Automation Framework. Yrityksen rakentama testiautomaatiokehys monitorin verifiointiin tarvittaville työkaluille, liitännäisille ja skripteille.
TSV	Tab-Separated Values. Tekstimuoto tietojen tallentamiseksi taulukkorakenteeseen.
Verifikaatio	Ohjelmiston toimintojen vaatimustenmukaisen toiminnan todistaminen testaamalla ja dokumentoimalla tulokset. Testaus on aina verifikaatiota, kun noudatetaan ennalta määrättyjä proseduureja ja tulokset dokumentoidaan todisteiksi.
Versionhallinta	Ketterässä ohjelmisto- tai testiautomaatiokehityksessä muutokset yhdistetään ajoittain versionhallintaohjelmistossa säilytettyyn päähaaraan, jotta virheiden esiintyessä edellisiin versioihin voidaan palata tai versioita vertailla.
XML	Extensible Markup Language. Merkintäkielien standardi, joka määrittää tietojen merkintämuodon loogisella rakenteella.

1 Johdanto

Työn tavoitteena on suunnitella potilasmonitorin ohjelmiston validaatiotestauksen automatisointi Robot Frameworkilla. Työn tilaaja on maailman johtavan terveysteknologia-alan yritys GE Healthcaren tytäryhtiö GE Healthcare Finland Oy. Suomen päätoimipiste Helsingin Vallilassa on yksi GE Healthcaren johtavista potilasmonitoroinnin tuotekehityksen ja hallinnan osaamiskeskuksista, jossa sairaalalaitteita on kehitetty jo yli 45 vuoden ajan. GE Healthcare Finland Oy työllistää Suomessa noin 700 ihmistä, jotka edustavat yli 25 eri kansallisuutta. [1.]

Potilasmonitorien ohjelmistoille tehdään säännöllisiä päivityksiä muun muassa tietoturvallisuuden parantamiseksi. Aina ennen uuden päivityksen julkaisua potilasmonitorille suoritetaan ohjelmiston validaatio. Ohjelmiston validaatiossa verifioidaan järjestelmän kriittisimmät toiminnot ennalta määritetyn testiryhmän mukaan. Verifikaatiolla tarkoitetaan testausta, jossa dokumentoidaan objektiivisin todistein ohjelmiston vaatimustenmukainen toiminta. Ohjelmiston validaatio on osa FDA:n (U.S Food and Drug Administration) määrittelemää *Design Control* -prosessia, joka varmistaa laitteen luotettavan toiminnan ja on välttämätöntä tehdä myyntiluvan saamiseksi Yhdysvaltojen markkinoille.

Jokaisen lääkinnällisen laitteen tulee vastata sille asetettuja vaatimuksia viranomaismääräyksissä. Viranomaismääräyksien tarkoituksena on taata jokaisen laitteen luotettava toiminta ja potilasturvallisuus kaikissa tilanteissa. Potilasmonitori on toiminnoiltaan kompleksinen laite, jota käytetään laajasti terveydenhoidon ammattilaisten työkaluna potilaan tilan ja hoidon arvioimisessa. Tästä syystä potilasmonitorin toiminta tulee verifioida testaamalla sen kaikki toiminnot kattavin todistein, jotta voidaan varmistua siitä, että se toimii luotettavasti jokaisessa tilanteessa.

Testiautomaation suunnitteluun käytettiin toisen potilasmonitoriprojektin testiautomaatiokokonaisuutta sekä siihen valittua ohjelmiston validaation testiryhmää lähteenä. Projektin tavoitteena oli selvittää ja suunnitella, kuinka tämä olemassa oleva testijärjestelmä lähteenä käytetystä potilasmonitoriprojektista voitaisiin siirtää työn kohteena olevalle potilasmonitoriohjelmistolle. Tämän toteuttamiseksi vaadittiin perehtymistä testiautomaation toteutukseen Robot Frameworkilla. Lisäksi ymmärrys potilasmonitorin perustoiminnoista sekä testiautomaatioon liittyvistä yksityiskohdista oli keskeisessä roolissa, jotta

voitiin analysoida lähteenä olevan potilasmonitoriohjelmiston testiautomaatiota kohteena olevalle ohjelmistolle.

Tutkimuskysymys projektille on: “Mikä on tehokkain tapa automatisoida ohjelmiston validaatio potilasmonitoriohjelmistolle?” Tällä hetkellä työn kohteena olevan potilasmonitorin ohjelmiston validaatio koostuu ennalta määritetystä testiryhmästä, joka verifioidaan manuaalisesti. Testiryhmällä tarkoitetaan kaikkia erillisiä testitapauksia, jotka verifioidaan ohjelmiston validaatiossa. Tyypillisesti manuaalisesti suoritettava testaus on hitaampaa ja alttiimpaa inhimillisille virheille. Automatisoimalla ohjelmiston validaation testiryhmä lopullisen testauksen aika lyhenee, toistettavuus paranee sekä virheiden määrä pienenee.

Lähde- ja kohdeohjelmistojen välillä on eroja sekä toiminnallisuuden että ominaisuuksien suhteen. Tästä syystä oli tärkeää analysoida sekä dokumentoida erot, jotta voidaan luoda mahdollisimman tarkka suunnitelma ohjelmiston validaation automatisoinnista sekä ohjelmiston validaatiossa suoritettavasta testiryhmästä. Kohteena oleva ohjelmisto on jo julkaistu ja markkinoilla. Siksi on tärkeää myös kuvata tässä työssä sen muutoksia koskeva viranomaismääräysten mukainen prosessi. Projektiin kuului myös testiautomaatioympäristön suunnittelu ohjelmiston validaation testiryhmälle, joka sisältää vaadittavien laitteiden ja työkalujen kartoituksen sekä valinnan.

Työn lopputuloksena on suunnitelma tilaajalle, jota voidaan käyttää tukena automaatiotestijärjestelmän siirtämisessä sekä testiautomaation sovittamisessa ja toteutuksessa potilasmonitoriohjelmistolle. Tarkat kuvaukset työn sisällöstä ja ratkaisuista ovat salassa pidettävää tietoa, ja insinöörityössä kuvataan menetelmät yleisellä tasolla. Lopullisen työnantajalle toimitettavan suunnitelman tavoitteena on kuvata tarkasti automaatiotestien kattavuus ohjelmiston vaatimusten kannalta sekä kuvata tarkkaan huomioitavat seikat ja keskeiset erot näiden ohjelmistoversioiden välillä.

2 Potilasmonitori

Potilasmonitori on laite, joka on suunniteltu ja tarkoitettu mittaamaan potilaan vitaaliparametreja sekä elintoimintoja. Potilasmonitoria voidaan käyttää lyhytaikaisesti, kuten

leikkaussalissa tai pitkäaikaisesti kuten teho-osastolla. Jokaisessa olosuhteessa on ensiarvoisen tärkeää, että laite ilmoittaa elintoimintojen heikentymisestä käyttäjälle, jolloin voidaan aloittaa oikeanlaiset toimenpiteet potilaan elintoimintojen turvaamiseksi. [2, s. 947.]

Potilasmonitori sisältää monia eri algoritmeja vitaaliarvojen analysointia varten ja antaa näiden avulla käyttäjälle jatkuvaa tietoa potilaan tilasta numeerisilla arvoilla sekä visuaalisesti aaltomuodoilla. Potilasmonitori mittaa esimerkiksi sydänsähkökäyrää (EKG, elektrokardiogrammi), pulssioksimetriaa (SpO2, happisaturaatio), verenpaineita (IP, invasiivipaine), aivosähkökäyrää (EEG, elektroenkefalografia) ja hengityskaasuja. Näitä arvoja tulkitsemalla on mahdollista havaita potilaan elintoimintojen muutokset reaaliaikaisesti ja määrittää hoito nopeasti. Kuvassa 1 on GE Healthcaren Carescape B650-potilasmonitori, joka on yksi kolmesta eri monitorityypistä, jolle tässä työssä suunnitellaan ohjelmiston validaation automatisointia. Kyseinen potilasmonitori on modulaarinen, eli vitaaliparametrien mittausta varten potilasmonitoriin kiinnitetään eri mittausmoduuleita.

Kuva 1. GE Healthcare Carescape B650 -potilasmonitori. [3.]

Potilasmonitorissa käytetään eri ohjelmistopaketteja, joiden tarkoituksena on määrittää laitteen toiminta vastaamaan tiettyä käyttötarkoitusta tai käyttöympäristöä kuten esimer-

kiksi leikkaussalia tai vastasyntyneiden intensiivistä hoitoa varten. Jokaisella ohjelmistopaketilla on omat vakio- ja hälytysasetukset vastaamaan käyttötarkoituksen ja käyttöympäristön tarpeita. Potilasmonitorin osa toiminnoista on lisensoitu, eli niiden käyttöä varten tarvitaan erillinen lisenssi.

Lääkinnällisenä laitteena potilasmonitorin on täytettävä viranomaismääräysten sille asettamat vaatimukset. Vaatimusten määrittely ja niiden noudattaminen takaavat, että laite toimii luotettavasti sekä täyttää FDA:n viranomaismääräykset. Edellä mainitun tahon antamien määräysten täyttäminen vaaditaan myyntiluvan saamiseksi potilasmonitorille Yhdysvaltojen markkinoille. Viranomaismääräykset riippuvat kohdemarkkinoista ja vaatimusten määräytymisessä voi olla alueellisia eroja. Kunkin alueen viranomaismääräysten täyttäminen vaaditaan myyntiluvan saamiseksi kyseisille kohdemarkkinoille.

Potilasmonitorin vaatimustenmukainen toiminta tulee osoittaa verifioimalla eli testaamalla sen kaikki toiminnot. Verifioinnilla tarkoitetaan sitä, että laitteen toiminta testataan sekä tulokset dokumentoidaan objektiiviseksi todisteeksi toiminnan oikeellisuudesta. Verifikaatiosta dokumentoidaan siis aina objektiivinen todiste laitteen toiminnasta, kun taas testaaminen tarkoittaa sitä, että ohjelmiston toiminto testataan, mutta siitä ei välttämättä dokumentoida todistetta. Verifikaation tarkoituksena on siis dokumentoida riittävä määrä todisteita, joilla voidaan osoittaa viranomaisille laitteen vaatimustenmukainen toiminta. Kehityksen aikana suoritetun verifikaation lisäksi potilasmonitorinohjelmiston kriittisimmät toiminnot verifioidaan ohjelmiston validaatiossa ennen laitteen julkaisua. Ohjelmiston validaatiossa verifikaatio suoritetaan samalla tavalla kuin muu ohjelmiston kehityksen aikana suoritettu verifikaatio eli laitteen toiminta testataan ja nämä tulokset dokumentoidaan todisteiksi.

Vaatimustenhallinta on keskeisessä roolissa lääkinnällisen laitteen kehityksessä. Kunnollisella vaatimustenhallinnan työkalulla voidaan selkeästi seurata polkua asetetuista vaatimuksista aina objektiiviseen todisteeseen siitä, että laitteen toiminta täyttää sille asetetut vaatimukset. Objektiivinen todiste voidaan hankkia monella tapaa: testaamalla laitteen toiminta manuaalisesti ja kirjaamalla tulokset todisteeksi siitä, että laite toimii vaatimusten mukaisesti. Sama voidaan myös suorittaa testiautomaatiolla. Jos vaatimuksen toteutumista ei voida testaamalla osoittaa, voidaan ohjelmiston toiminta verifioida myös koodin tarkastelulla.

3 Potilasmonitorin verifiointi testiautomaatiolla

Suuri osa potilasmonitorin toiminnoista voidaan verifioida manuaalisesti potilasdatasimulaattoreilla käyttäen eri mittausmoduuleita. Potilasdatasimulaattorilla voidaan simuloida mittausmoduuleille eri parametrien mittausdataa. Osaan manuaaliseen verifikaatioon käytetään ohjelmistosimulaattoreita, joiden avulla simuloidaan potilasmonitoriin lähetettyä dataa ja esimerkiksi laitteiden vikatilanteisiin liittyviä hälytyksiä. Kun ohjelmistosta pitää verifioida osia tai toimintoja, joita verifikaatioinsinööri ei pysty tuottamaan, käytetään koodin analysointia *Code Inspection* -verifikaatioon. Manuaalinen ohjelmiston verifikaatio on tyypillisesti aikavievää, sekä tarkkuutta vaativaa työtä.

Tätä verifikaatioprosessia on mahdollista suoraviivaistaa suorittamalla verifikaatio testiautomaation avulla. GE Healthcare toteuttaa automaatiotestausta *Test Automation Frameworkilla* (TAF), joka koostuu Robot Framework testi- ja resurssitiedostoista sekä python-kirjastoista. Robot Framework on testiautomaatiokehys ohjelmistojen hyväksyntätason testaukseen. Edellä mainittu TAF käsittää yrityksen itseluomat työkalut ja skriptit, joita tarvitaan potilasmonitorin ohjelmiston eri osioiden ohjaukseen ja verifikaatioon.

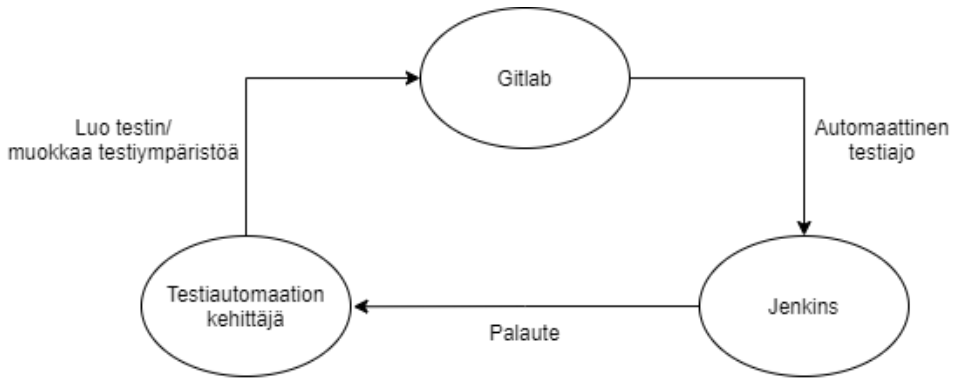
Kaikkien testiautomaatiossa käytettävien testi- ja mittauslaitteiden, simulaattorien ja ohjelmistojen tulee olla validoitu testikäyttöä eli verifikaatiota varten. Edellä mainittujen testityökalujen validoinnin tarkoituksena on arvioida ja dokumentoida virallisesti jokaisen käytettävän testityökalun ominaisuudet ja rajoitukset suhteessa sen käyttötarkoitukseen ja käyttöympäristöön [4]. Käytettävien testityökalujen validoinnilla siis varmistutaan siitä, että työkalu toimii luotettavasti ja soveltuu käyttötarkoitukseen verifikaatiossa. GE Healthcare validoi jokaisen käytettävän testityökalun, jotta sitä voidaan käyttää virallisessa verifikaatiossa.

Kun verrataan manuaalista ja automatisoitua verifikaatiota, manuaalisessa verifikaatiossa testaaja voi löytää poikkeavuuksia testattavan ominaisuuden ulkopuolelta. Testiautomaatio keskittyy vain ennalta määritettyihin ominaisuuksiin, jotka on kirjoitettu komentoina testitapauksiin. Yksi testaaja pystyy myös suorittamaan useita eri testitapauksia samanaikaisesti, ja niitä voidaan suorittaa vuorokauden ympäri.

GE Healthcare käyttää testiautomaatiota laajalti uusimmilla potilasmonitorin ohjelmistoversioilla. On ollut tapauksia, joissa testausautomaatio tunnistaa virheet toistamisen

kautta, jotka olisi voinut jäädä havaitsematta manuaalisella verifikaatiolla. Manuaalinen verifikaatio on paljon hitaampaa ja vaatii enemmän työtunteja, mikä lisää kustannuksia ja ohjelmiston julkaiseminen vaatii enemmän aikaa. Kunkin ohjelmistoprojektin testausautomaation tarve tulisi laskea testiautomaatioympäristön luomisesta ja testitapausten suorittamisesta ja verrata todellisiin testaustarpeisiin. Lisäksi tulee huomioida testiautomaatiossa tarvittavien laitteistojen kustannukset sekä testiympäristön rakentamiseen kuluva aika.

Testiautomaatiota varten voidaan luoda erillinen testiympäristö, jossa testijärjestelmää eli testitapauksia, testikirjastoja ja muita testaamiseen käytettyjä tiedostoja voidaan suorittaa ja niiden toimivuutta testata testattavalle ohjelmistolle. Hyvin koottu testiautomaatiojärjestelmä koostuu useista kirjastoista ja resurssitiedostoista, joita hyödynnetään laajasti eri testitapauksissa. Muutoksien tekeminen yhteen kirjastoon tai resurssitiedostoon voi vaikuttaa toisen testitapauksen tulokseen. Tästä syystä testiautomaatiojärjestelmä on suositeltavaa rakentaa niin, että sen toimivuus pystytään tarkastamaan jokaisen implementoidun muutoksen jälkeen. Kuvassa 2 esitellään jatkuvan kehityksen kulku toimintakaaviona, missä työkalu, jossa testitiedostoja kehitetään, on yhdistetty automaatiopalvelimeen, joka käynnistää automaattisesti testin ohjelmistolle varmistaakseen, ettei testitiedostoissa ole virheitä.



Kuva 2. Jatkuvan kehityksen toimintakaavio. [5.]

Testitiedostoja kehitetään versionhallintatyökalussa kuten esimerkiksi GitLab, johon testiautomaation kehittäjä yhdistää muutokset ja muutetuille testitiedostoille suoritetaan testi sen toimivuuden tarkastamiseksi automaatiopalvelimen kuten Jenkinsin kautta.

Jenkinsiin on määritetty testiryhmä, joka suoritetaan aina, kun testitiedostoihin yhdistetään muutoksia ja testiautomaation kehittäjä saa tästä palautteen. Jenkins on esimerkki automaatiopalvelimesta, jonka avulla ohjelmisto voidaan rakentaa ja testata automaattisesti. Jatkuvan integraation menetelmässä kehitysvaiheessa kaikki tehdyt muutokset integroidaan vähintään kerran päivässä, ja jokainen integraatio tarkistetaan virheiden havaitsemiseksi automatisoidulla testillä [6].

Jatkuvan integroinnin menetelmä auttaa testiautomaation kehittäjiä työskentelemään samanaikaisesti eri ominaisuuksien kanssa sekä suorittamaan toistettavia testejä uusille versioille. Välitön palaute muutosten jälkeen vähentää riskejä rikkinäisiin versioihin, edistää nopeampaan kehitykseen sekä laadukkaampiin ohjelmistoversioihin.

4 Robot Framework

Robot Framework on testiautomaatioympäristö ohjelmistojen hyväksyntätason ohjelmistotestaukseen ja kehittämistyöhön. GE Healthcarella tätä testiautomaatioympäristöä käytetään potilasmonitoriohjelmiston testaukseen ja on keskeisessä roolissa projektin kohteena olevan potilasmonitorin ohjelmistolle suoritettavassa testiautomaation suunnittelussa. Robot Frameworkin avoimen lähdekoodin ohjelma on julkaistu Apache License 2.0:lla. Robot Frameworkin on alun perin kehittänyt suomalainen Nokia Siemens Networks ja sen kehitystä sponsoroi Robot Framework Foundation, jonka jäseninä on yrityksiä kuten Cisco, OP, Veikkaus, Finnair, Reaktor, OURA, Botlabs, ja Robocorp. [7.]

Robot Framework käyttää helppoa ihmisen luettavaa syntaksitekniikkaa. Tämä mahdollistaa itse testitapausten kirjoittamisen ilman kattavaa ymmärrystä ohjelmoinnista. Kirjoitetuille syntakseille koodataan oikeat komennot yleensä resurssitiedostoihin, joista testattavaa ohjelmistoa ohjataan.

Robot Frameworkia voidaan käyttää testaamaan esimerkiksi verkkosivuja, iOS- sekä Android-puhelinapplikaatioita, tietokantoja ja tulkkien avulla eri kielillä kirjoitettuja ohjelmistoja. Robot Frameworkin etu verrattuna muihin testauskehyksiin on sen ison yhteisön tuki sekä lokitiedostot. Suuret ja kuvaavat lokitiedostot mahdollistavat testien tulosten selkeän tarkastelun sekä virheiden havaitsemisen.

4.1 Avainsanaohjattu testaus

Robot Framework käyttää testien kuvaamiseen avainsanoja, jotka kirjoitetaan taulukkomaisesti. Robot Framework testit voivat olla avainsanaohjattuja (Esimerkkikoodi 1), käytösohjattuja tai dataohjattuja testitapauksia. Taulukoiden formaatti voi olla tekstitiedosto (Esimerkkikoodi 1), HTML (Hypertext Markup Language), TSV (tab-separated values) tai reST (reStructuredText).

```
*** Settings ***
Documentation     A test suite for valid login.
...
Resource          LoginResource.robot
Test Setup        Open Browser
Test Teardown     Close Browser

*** Test Cases ***
Valid Login
    [Documentation]   Verifies that login page works
    [Tags] Requirement_1
    Open Browser To Login Page
    Input Username    admin
    Input Password    secret
    Submit Credentials
    Welcome Page Should Be Open
```

Esimerkkikoodi 1. Robot Framework -esimerkkikoodi tekstitiedostomuodossa.

Testitiedosto koostuu tyypillisesti neljästä eri osiosta, jotka ovat *Settings* (asetukset), *Variables* (muuttujat), *Test Cases* (testitapaukset) ja *Keywords* (avainsanat). Settings-kohdassa alustetaan testissä käytettävät kirjastot ja resurssit sekä *Test Setup* (testin alustus) ja *Test Teardown* (testin lopetus). Tarvittaessa koko testikokonaisuus voidaan alustaa *Suite Setup* ja lopettaa *Suite Teardown* -komennoilla.

Settings-kohdassa määritetään testissä käytettävät testikirjastot, resurssitiedostot ja erilliset tiedostot, joihin on kirjoitettu esimerkiksi muuttujia. *Test Setup* on komento, joka koostuu avainsanoista, jonka testi suorittaa ensimmäisenä, esimerkiksi ‘avaa verkkoselain’. *Test Teardown* -komennon testi suorittaa jokaisen testitapauksen lopuksi, esimerkiksi “sulje verkkoselain”. *Test Setup*- ja *Test Teardown* -komentojen tarkoituksena on vähentää toistoa eri testitapauksille.

Jos testitiedosto koostuu useista testitapauksista (*Test Case*), voidaan testattava ohjelmisto alustaa suorittamalla *Suite Setup* -komento ennen yhdenkään testitapauksen suorittamista. Vastaavasti testikokonaisuus voidaan lopettaa suorittamalla *Suite Teadown* -komento. Kun testikokonaisuuden kaikki testitapaukset vaativat saman lähtökohdan ja aloituskomennot, jolloin toiston sekä testausajan vähentämiseksi on järkevää käyttää Suite-komentoja. Esimerkiksi testikokonaisuudessa voi olla usea eri testi, joka testaa verkkosivun eri toimintoja. Tällöin koko testikokonaisuus *Test Suite* voidaan alustaa avaamalla verkkoselain *Suite Setup* -komennolla.

Muuttujat voidaan myös kirjoittaa suoraan testitiedoston *Variables*-kohtaan joko pienillä kirjaimilla, jolloin muuttujaa voi käyttää vain testitiedostossa, tai isoilla kirjaimilla, jolloin muuttuja on *Global Variable* eli käytettävissä myös muissa testiin liittyvissä tiedostoissa. Robot Frameworkilla muuttujien käyttö ei ole välttämätöntä, mutta niiden avulla voidaan vähentää toiston tarvetta ja niin sanottua kovakoodaamista, jossa muuttujat kirjataan suoraan avainsanoihin eli lähdekoodiin. Muuttujien kovakoodaamista tulisi välttää, jotta testitapausta voi muokata suoraan komentoriviltä testiä suoritettaessa tai *Variables*-kohdasta kaikkialle, missä muuttujaa kutsutaan. Testin kovakoodaaminen myös vaikeuttaa sen muokkaamista, jos jokin ominaisuus ohjelmassa vaihtuu. Tällöin kovakoodattu osuus tulee muokata jokaiseen käytettyyn paikkaan. Resurssitiedostoon (*Resource*) kirjataan korkean tason avainsanoja sekä muuttujia. Kirjastot sisältävät alimman tason avainsanoja ja ovat eniten vuorovaikutuksessa testattavan järjestelmän kanssa [8].

Test Case -kohta suorittaa testin komennot hyödyntäen vain joko tiedostossa olevia, viitatuissa kirjastoissa tai resurssitiedostoissa olevia avainsanoja. Avainsanat eli suoritettavat komennot määritetään *Keywords*-kohdassa. Mikäli testin laajuus on kapea, eikä testissä käytettäviä avainsanoja tarvita muualla, silloin voidaan pitää kaikki osiot yhdessä tiedostossa. On kuitenkin suositeltavaa luoda oma resurssitiedosto avainsanoille. Tällä tavoin voidaan samoja avainsanoja käyttää monissa eri testeissä ja vältytään toistolta. Riippuen testin kohteesta on monesti tarpeellista myös luoda oma rajapinta, jolla ohjataan kohteena olevaa ohjelmistoa, kuten tässä insinöörityössä.

Testitapaus kutsuu avainsanoja ja muuttujia resurssitiedostosta, josta vastaavat avainsanat sekä muuttujat löytyvät. Esimerkkikoodissa 2 on Robot Framework -resurssitiedosto, joka sisältää viittauksen SeleniumLibrary-kirjastoon, käytettäviä muuttujia sekä testitapauksesta kutsuttavia avainsanoja.

```
*** Settings ***
Documentation     Resource file for Login test.
...               Includes keywords and libraries.
...
Library           SeleniumLibrary

*** Variables ***
${BROWSER}          Firefox
${USERNAME}         admin
${PASSWORD}         secret
${LOGINPAGE URL}    http://www.minunsivu.fi/
${MAINPAGE URL}     http://www.minunsivu.fi/welcome.html

*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGINPAGE URL}    ${BROWSER}
    Login Page Should Be Open

Login Page Should Be Open
    Title Should Be    Log in to discover world

Input Username
    [Arguments]    ${username}
    Input Text     username_field    ${username}

Input Password
    [Arguments]    ${password}
    Input Text     password_field    ${password}

Submit Credentials
    Click Button    login_button

Welcome Page Should Be Open
    Location Should Be    ${MAINPAGE URL}
    Title Should Be        Welcome Page
```

Esimerkkikoodi 2. Robot Framework -esimerkkikoodi resurssitiedostosta.

Resurssitiedosto esimerkkikoodissa 2 oleva avainsana ‘Login Page Should Be Open’ käsittää ihmislukuisesti ymmärryksen, mitä avainsanalla testataan. ‘Title Should Be’ on SeleniumLibraryn komento, joka on alustettu avainsanassa tarkastamaan, vastaako annettu syöte ‘Log in to discover world’ -sivun otsikkoa. Jokainen avainsana tarkistaa ehdon toteutumisen ja antaa testaajalle tiedon tuloksesta. Esimerkkikoodissa 2 oleva komento ‘Title Should Be’ testaa, onko sivun otsikko ‘Welcome Page’.

Testitapaus voi olla hyvin korkean tason tyyppinen sekä kuvaava, jotta myös henkilö, joka ei tunne Robot Framework -ohjelmointia saa käsityksen, mitä avainsanat testaavat. On myös hyödyllistä käyttää testattavan ominaisuuden avainsanoja kuvaamaan, mitä testijärjestelmä testaa ohjelmasta. Tämä mahdollistaa myös piilottamaan monimutkaisemmat ohjelmakoodit, toiminnallisuudet sekä skriptit alemmille tasoille avainsanojen taakse. [9.]

4.2 Dataohjattu testaaminen

Avainsanaohjattujen testien lisäksi käytetään paljon myös dataohjattuja testitapauksia. Dataohjatussa testitapauksessa suoritetaan *Test Template* -oletusavainsana useaan kertaan käyttäen eri muuttujia, jotka on määritetty *Test Cases* -kohdassa. Dataohjatussa testitapauksessa ajettu testi pysyy jokaisella kerralla samana, mutta käytetyt muuttujat vaihtuvat. Esimerkkikoodi 3 on esimerkki dataohjatusta testitapauksesta, jossa annetaan neljän eri käyttäjän toimivat käyttäjänimet sekä salasanat ja testataan kirjautumista suorittamalla 'Login With All Page Users' -avainsana.

```
*** Settings ***
Documentation     This test verifies that page can be accessed ...
with all users
...
Resource          ../resources/login_resources.robot
Suite Setup       Open Login Page
Suite Teardown    Close Browser
Test Template     Login With All Page Users

*** Test Cases ***    USERNAME    PASSWORD
User1                 admin       secret
User2                 fielduser   fieldpass
User3                 service     servicepass
User4                 basicuser   userpass

*** Keywords ***
Login With All Page Users
    [Arguments]         ${user}    ${pass}
    Input Text          ${USERNAME_INPUT}    ${user}
    Input Password      ${PASSWORD_INPUT}    ${pass}
    Click Button        Login
    Title should be     Welcome page
    Click Link          ${LOG_OUT}
```

Esimerkkikoodi 3. Robot Framework -esimerkkikoodi dataohjatusta kirjautumistestistä eri käyttäjänimillä ja salasanoilla.

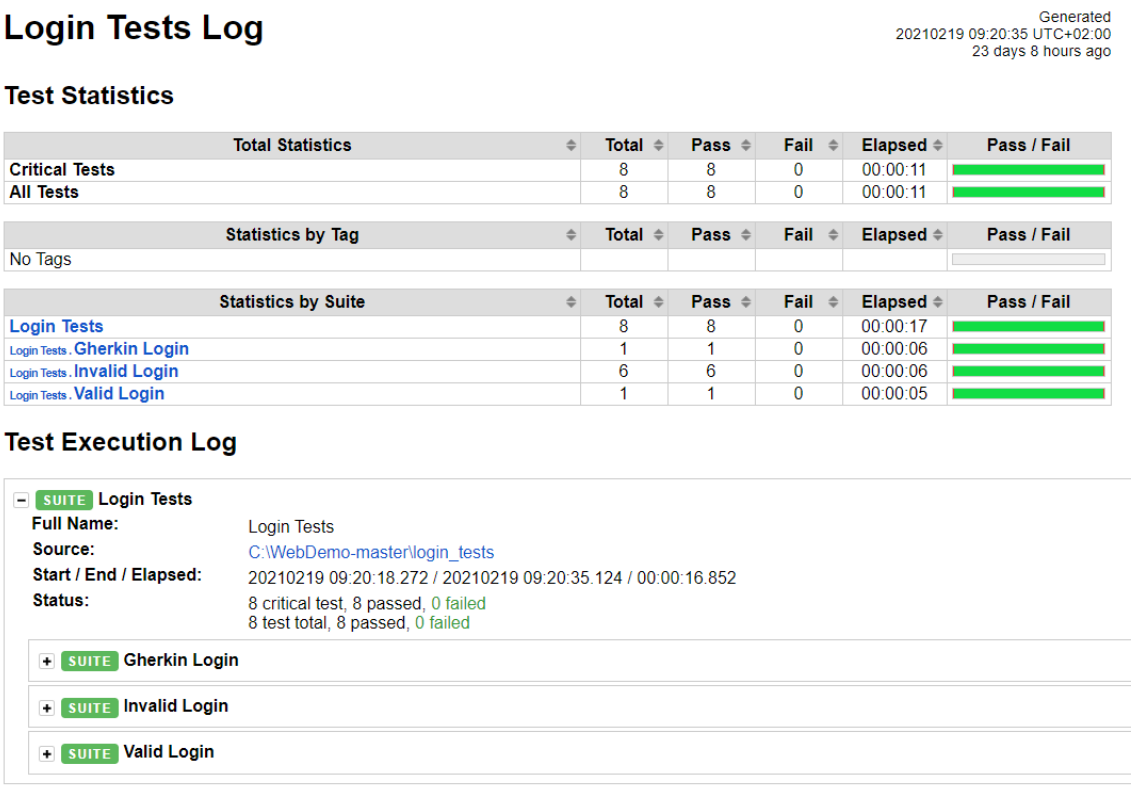
Dataohjattu testi vähentää toistoa testitapausta kirjoitettaessa ja mahdollistaa samankaltaisten testien suorittamisen useilla eri muuttujilla. Tämän lisäksi testien ylläpidettävyys on helpompaa, koska muutoksia ei tarvitse tehdä yksitellen jokaiselle testitapaukselle. Mikäli Esimerkkikoodi 3:ssa olisi enemmän testattavia käyttäjiä (*user*), voitaisiin nämä tiedot koostaa csv-tiedostoon ja hakea tiedot sieltä, mikä selkeyttää testin suorittamista ja ylläpitämistä entisestään.

Robot Framework on tehty Python-ohjelmointikielellä, mutta tukee myös Java-ohjelmointikieltä Jython-tulkin avulla. Laajat testausominaisuudet Robot Frameworkilla tarjoavat

Pythonilla tai Javalla kirjoitetut testikirjastot, joita käyttäjä voi itse räätälöidä tai luoda. Robot Frameworkille löytyy paljon ulkoisia ladattavia kirjastoja, esimerkiksi SeleniumLibraryn web-pohjaiseen ja Java GUI:n testaukseen. [8.] SeleniumLibrary käyttää WebDrivereita verkkoselaimen hallintaan.

4.3 Testiraportti

Kun testitapaus on ajettu, Robot Framework palauttaa kattavan selvityksen testin suorituksesta kahdessa osassa: Loki sekä Raportti. Molemmat tiedostot ovat HTML-muodossa. Kuvassa 3 on kirjautumistestistä saatu lokitiedosto.

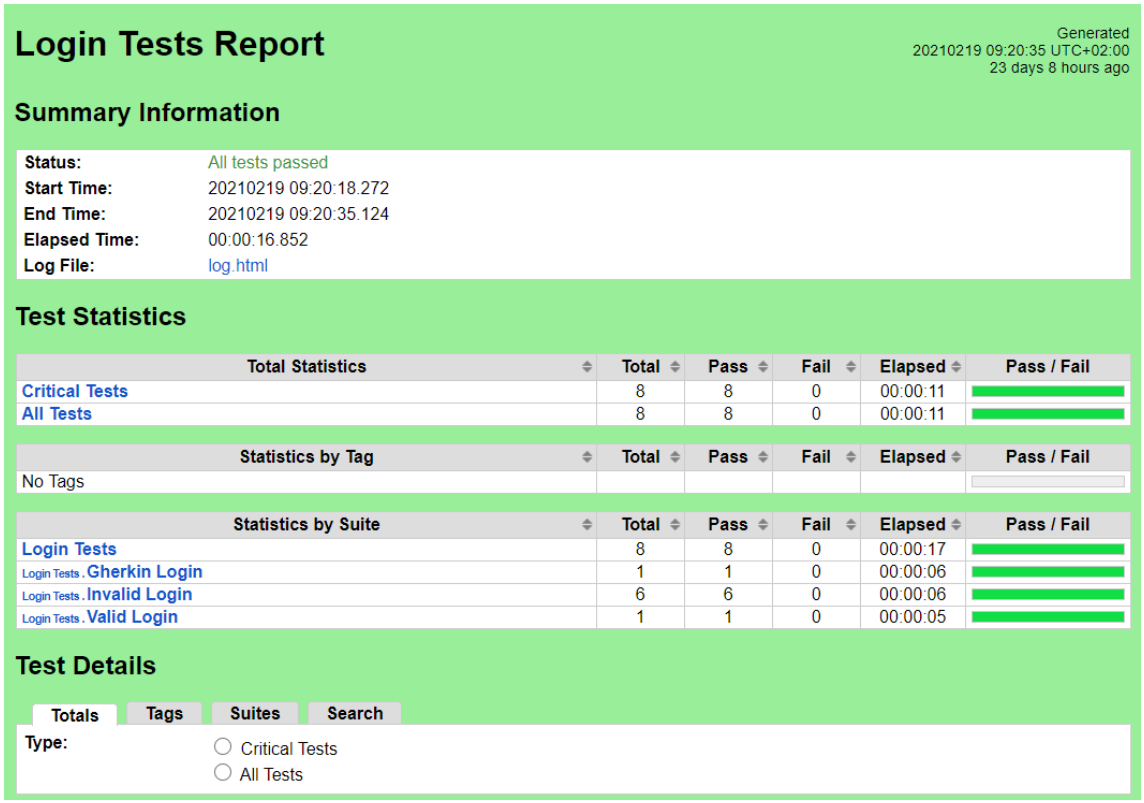


Kuva 3. Robot Framework -testin log.html-tiedosto.

Kuvan 3 log.html sisältää testin kokonaisläpäisyn lisäksi jokaisen avainsanan läpäisyn tai hylkäyksen sekä jokaiseen vaiheeseen käytetyn ajan. Testitapauksessa suoritettiin kolme eri testiä: Gherkin Login, Invalid Login sekä Valid Login. Osa testeistä sisälsi useamman testitapauksen. Yhteensä suoritettiin 8 erillistä testiä, jotka kaikki saivat tuloksen

‘PASS’ eli läpäisty. Kaikki testitapaukset voidaan log.html-tiedostossa avata osiin ja nähdä yksityiskohtaisesti vaiheissa suoritetut ohjaukset ja komennot sekä niiden saamat palautteet. Selkeästi luettavissa oleva loki sekä raportti helpottavat testiautomaation kehittämistä laitteelle tai sivustolle sekä löytämään ohjelmistosta vikoja. Testin tuloksen ollessa hylätty näkyy lokista, mikä testin osa ei saanut läpäisyä, mikä avainsana puuttuu testistä tai mitä arvoa ei löytynyt.

Lokitiedoston lisäksi testistä saadaan kuvassa 4 oleva raportti, josta näkee yleiskatsauksen suoritetusta testistä.



Kuva 4. Robot Framework -testin report.html-tiedosto.

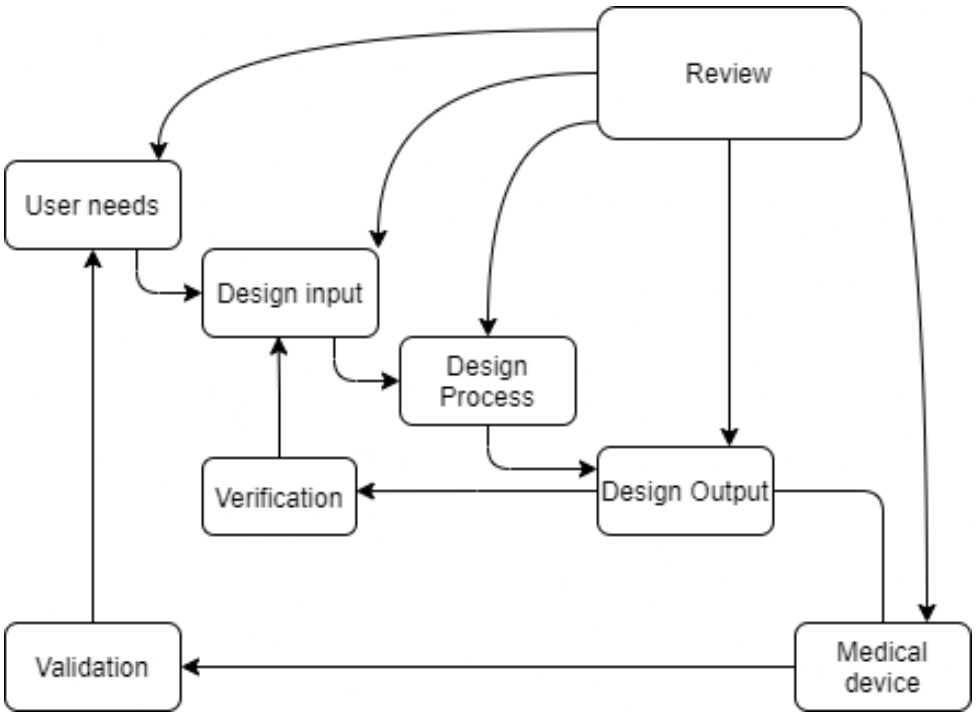
Raportti on linkitetty lokitiedostoon, joka helpottaa navigointia lokitiedoston yksityiskohtaisempaan informaatioon. Raportin ulkoasu on tehty käyttäjälle mahdollisimman selkeäksi käyttämällä koko taka-alan väriä kuvastamaan testin tulosta.

4.4 Tunnisteet

Robot Framework käyttää testien luokitteluun Tageja eli tunnisteita. Näiden käyttö on vapaavalintaista, mutta niitä on suositeltavaa käyttää yksilöimään sekä selkeyttämään testattavaa ominaisuutta tai esimerkiksi vaatimusta. Loppuraportti näyttää tunnisteet tilastoina, esimerkiksi testattujen vaatimusten lukumäärän, läpäisyt sekä hylkäykset tunnisteittain. Tunnisteita voi myös käyttää testitapauksia suorittaessa sisällyttämään tai poistamaan osia testistä. Selkeä testitapausten luokittelu auttaa järjestelmälliseen vaatimusten testaukseen sekä varmistamaan, että kaikki kriittiset testit on suoritettu ja testien kokonaistulos on läpäisty. [8.]

5 Suunnittelun hallintaprosessi (*FDA Design control process*)

Tässä luvussa kuvataan FDA:n määrittelemä *Design Control* -prosessi ja vaatimusten määräytyminen lääkinnälliselle laitteelle. *Design Control* -prosessin tarkoituksena on taata lääkinnällisen laitteen kehityksessä vaadittavien vaatimusten täyttyminen. Prosessi määrittelee kehityksen aikana vaadittavan dokumentoinnin ja menettelytavat laitteen oikean toiminnan takaamiseksi ja osoittamiseksi. Laitteen käyttötarkoitus määrittää sen tuoteluokan, jonka mukaan myös vaatimukset määräytyvät. FDA:n mukaan jokaisen laitteen, joka asettuu luokkaan II tai III, tulee noudattaa *Design Control* -prosessia myyntiluvan saamiseksi Yhdysvaltain markkinoille. Työn tarkoitus on luoda suunnitelma potilasmonitoriohjelmiston validaatiolle, joka on osa *Design Control* -prosessin *Design Change* -vaihetta. Potilasmonitori kuuluu käyttötarkoituksensa vuoksi tuoteluokkaan II [10] ja prosessi kuvataan tämän tuoteluokan mukaan. *Design Control* -prosessi on vain yksi osa FDA:n *CFR - Code of Federal Regulations Title 21*:ta, jota jokaisen lääkinnällisen laitteen valmistajan tulee noudattaa Yhdysvaltain markkinoilla. [11] Kuvassa 5 on visualisoitu *Design Control* -prosessin olennaisimmat vaiheet laitteen kehityksen kannalta.



Kuva 5. *Design control* -prosessia kuvaava kaavio [12.]

Jokaisen vaiheen jälkeen suoritetaan *Design Review* (kuvassa *Review*). Tämän tarkoituksena on varmistaa, että tuotesuunnittelu on turvallista, tehokasta ja varmistaa aikataulussa eteneminen. *Design Review* on siis erittäin tärkeä osa *Design Control* -prosessia, sillä sen avulla varmistetaan se, että laite on turvallinen käyttää sekä vastaa sille asetettuja vaatimuksia. Näiden arviointien (*Design Review*) tulokset tulee kirjata DHF (*Design History File*) -dokumenttiin, jota tulee ylläpitää koko laitteen elinkaaren ajan. [13.]

Koko laitteen kehitys lähtee suunnittelusta, jossa kuvataan laitteen käyttötarkoitus sekä käyttäjien tarpeet mahdollisimman tarkasti. Käyttötarkoitus kuvaa kliinistä ongelmaa, jonka laite pyrkii ratkaisemaan, esimerkiksi diagnosoimaan, hoitamaan, ehkäisemään tai lievittämään sairauksia. Käyttötarkoitus tulee rajata mahdollisimman tarkasti siten, että se kuvaa selkeästi ja yksinkertaisesti, mihin laitetta on tarkoitus käyttää. Käyttäjien tarpeilla (*Indications For Use*) on tarkoitus pyrkiä kuvaamaan millaisessa ympäristössä, miten ja kenen toimesta laitetta tullaan käyttämään. Käyttötarkoituksen sekä käyttäjien tarpeiden määrittely toimii pohjana laitteen suunnittelussa (kuvassa *User Needs*) ja tämän mukaan määräytyy myös viranomaismääräyksissä käytettävä tuoteluokka.

Laitteen lakisääteiset vaatimukset määräytyvät suoraan tuoteluokan mukaan. Tästä syystä näiden tarkka määrittely on tärkeää ja nämä täytyy dokumentoida huolellisesti suunnitteluvaiheessa. Suunnitteluvaiheessa tulee laatia ja dokumentoida suunnitelma (*Design Plan*), jossa kuvataan selkeästi päämäärät ja tavoitteet kehityksessä. Dokumentissa tulee myös selkeästi kuvata kaikki suunnittelun ja kehityksen kriittiset vaiheet sekä määrittää tarvittavat arvioinnit (*Review*) jokaiselle vaiheelle. Dokumentti tulee pitää ajan tasalla koko laitteen elinkaaren aikana, kun tehdään muutoksia prosessin johonkin vaiheeseen. Tämän dokumentoinnin tavoitteena on taata, että kehityksen jokaisessa vaiheessa tuotetta kehitetään vaatimusten mukaisesti. [14.]

Seuraava vaihe on määrittää tuotteelle vaadittavat syötteet (kuvassa *Design Input*), minkä tarkoituksena on määritellä laitteen toiminnallisuus ja muut keskeiset seikat käyttötarkoituksen ja käyttäjien tarpeiden mukaan. *Design Input* tarkoittaa lyhyesti sanottuna laitteen fyysisiä ja suorituskykyominaisuuksia, jotka ovat pohjana laitteen suunnittelussa. Esimerkkinä fyysisestä ominaisuudesta voisi olla, että laitteen tulee olla kannettava, eli silloin tulee määritellä tarkoin, kuinka paljon laite saa maksimissaan painaa. Kun taas suorituskykyyn viittaava ominaisuus voisi olla esimerkiksi se, että laitteen tulee suorittaa ja analysoida mittaus tietyssä ajassa. Kun kyseiset seikat on määritelty tarkoin, niistä koostetaan laitteelle vaatimukset. Laitteelle laadittujen vaatimusten tulee olla selkeästi määritelty sekä kattaa kaikki turvallisuuteen sekä laitteen oikeanlaiseen toimintaan liittyvät seikat. Vaatimuksia määriteltäessä on myös tärkeää ottaa huomioon laitteen tuoteluokkaa koskevat määräykset vaatimuksissa, jotta voidaan täyttää oikeanlainen vaatimustenmukaisuus viranomaismääräyksien suhteen. [13.]

Kun käyttötarkoituksen ja käyttäjien tarpeiden pohjalta määritellyt vaatimukset on tehty, arvioitu ja hyväksytty voidaan aloittaa laitteen kehittäminen (kuvassa *Design Process*). Kehityksessä tulee huomioida tarkasti aiemmin määritellyt vaatimukset ja laite tulee rakentaa näiden pohjalta. *Design Output* tarkoittaa *Design Input* -kohdassa määriteltyjen ominaisuuksien pohjalta luotua suunnitelmaa, jotka ovat esimerkiksi piirustuksia, vaatimuksia ja luomisohjeita. *Design Output* kuvaa kaikki osat, joista ohjelmiston rakentuu. *Design output* sisältää dokumentit, suunnitelman kuvaukset, analyysit, testiproseduurit sekä testien tulokset, jotka tuotetaan suunnittelutoiminnan aikana. Lopullinen *Design Output* osoittaa, että suunnitelma täyttää *Design Input* -kohdan vaatimukset. Kehityksen aikana jokainen toteutettu ominaisuus verifioidaan (kuvassa *Verification*). Verifikaation

tarkoituksena on varmistaa objektiivisin todistein, että ominaisuus on toteutettu vaatimusten mukaisesti. Verifikaation avulla siis varmistetaan, että laite on kehitetty oikein. Kun laitteen vaatimustenmukainen toiminta on osoitettu dokumentoiduin ja objektiivisin todistein, arvioidaan vielä, kuinka *Design Output* vastaa *Design Inputtia*. [15.]

Kun *Design Output* -arviointi on valmis, voidaan siirtyä lähemmäksi laitteen julkaisua (kuvassa *Medical Device*). Ennen tätä vaaditaan kuitenkin *Design Transfer* -prosessi ja laitteen validaatio. *Design Transfer* -prosessin tarkoituksena on arvioida ja selvittää vaadittavat yksityiskohdat laitteen sarjatuotantoa ajatellen. Kehityksen aikana rakennettu prototyyppi ei välttämättä vastaa komponenteiltaan lopullista tuotetta. Siksi on tärkeää arvioida ja määrittää sarjatuotanto niin, että lopullinen tuote on toiminnoiltaan sama prototyypin kanssa. Tämän jälkeen viimeinen vaihe on laitteen validaatio (kuvassa *Validation*). Validaation tarkoituksena on osoittaa, että käyttäjien tarpeet on täytetty dokumentoiduin objektiivisin todistein. [15.]

Kun markkinoilla olevalle laitteelle tarvii tehdä muutoksia, tulee noudattaa FDA:n *Design Control* -prosessin *Design Change* -vaatimuksia. Jokainen tehtävä muutos tulee määrittää, dokumentoida, validoida, tarkistaa ja hyväksyttää ennen muutoksen implementointia. Projektissa muutoksen kohteena on potilasmonitorin ohjelmiston tietoturvaan liittyvät parannukset sekä mahdollisten muiden toiminnallisuuksien parantaminen. Kun muutostarpeet on tunnistettu, arvioitu ja hyväksytty ne implementoidaan. Muutostarvetta arvioidessa tulee huomioida myös se, ettei tehty muutos vaikuta ohjelmiston muiden osien toimintaan. Jokainen yksittäinen muutos verifioidaan erikseen, jolla varmistetaan, että muutos on implementoitu oikein. Lisäksi verifioidaan muutoksesta mahdollisesti riippuvaiset toiminnot ja varmistetaan niiden oikeanlainen toiminta. [16.]

Kun kaikki ohjelmistolle tehdyt yksittäiset muutokset on arvioitu, implementoitu ja verifioitu, suoritetaan koko järjestelmän kattava ohjelmiston validaatio. Validaation laajuus koostuu ennalta määrätystä testiryhmästä, joka kattaa järjestelmän kaikki osa-alueet, myös niiltä osin, mihin muutoksia ei ole tehty. Näin varmistutaan siitä, että järjestelmä toimii luotettavasti ja ettei mikään yksittäinen muutos ole vaikuttanut ennalta arvaamattomalla tavalla ohjelmiston muihin osa-alueisiin. Seuraavassa luvussa käydään läpi menetelmät tämän ohjelmiston validaation testiryhmän valitsemiseksi työn kohteena olevalle potilasmonitoriohjelmistolle.

6 Työn vaiheet ja tulokset

Tässä luvussa kuvataan työn vaiheet sekä niiden tärkeimmät havainnot ja tulokset. Tarkat kuvaukset vaiheiden sisällöstä, ratkaisuista ja tuloksista ovat salassa pidettävää tietoa, ja luvussa kuvataan menetelmät yleisellä tasolla.

6.1 Projektin alustus

Projektin lähtökohtana oli suunnitella potilasmonitoriohjelmistolle Robot Framework -testiautomaatiokehyksellä toteutettava ohjelmiston validaatio. Nykyinen ohjelmiston validaatio suoritetaan manuaalisella verifikaatiolla. Ohjelmiston validaation tarkoituksena on suorittaa ennalta määritetty testiryhmä muun kattavan järjestelmä- ja ohjelmistotason verifikaatiotestauksen lisäksi ja varmistaa ohjelmiston kriittisimpien toimintojen toimivuus. Potilasmonitoriohjelmiston kehityksen aikana suoritetaan paljon kohdennettua verifikaatiotestausta ohjelmiston osiin, joihin muutoksia tehdään, kun taas ohjelmiston validaation testiryhmä pysyy vakiona ja kattaa laajasti potilasmonitorin tärkeimpiä toimintoja, myös niiltä osin, joihin muutoksia ei tehdä.

Validaatio tulee suorittaa aina ennen virallista ohjelmiston julkaisua ja jakelua käyttöön. Tämän jälkeen valmistaja kykenee osoittamaan viranomaisille varmistuksen siitä, että markkinoille tuotu laite toimii luotettavasti ja asetettujen viranomaismääräysten mukaisesti. Ohjelmiston validaation lisäksi suunnittelu, verifikaatio, testaus, jäljittäminen, kokoonpanon hallinta ja muut ohjelmistotuotannon näkökohdat tukevat päätelmää, että ohjelmisto voidaan julkaista markkinoille. Tämä johtopäätös perustuu useisiin tarkastuksiin ja analyyseihin, joita suoritetaan ohjelmiston koko elinkaaren ajan.

Työn kohteena olevalle potilasmonitoriohjelmistolle ei ole tällä hetkellä olemassa testiautomaatiota. GE Healthcarella on uudempia potilasmonitoriohjelmistoprojekteja, joille suuri osa verifikaatiosta suoritetaan testiautomaatiolla. Uudemman potilasmonitoriohjelmiston testiautomaatiojärjestelmää sekä ohjelmiston validaation testisuunnitelmaa käytetään lähteenä koko projektin ajan ja viitataan tästä eteenpäin lähdeohjelmistona tai lähdeprojektina. Tavoitteena on hyödyntää mahdollisimman paljon jo olemassa olevaa testiautomaatiojärjestelmää tästä lähdeprojektista, jonka perusteella muutetaan koh-

teena olevan potilasmonitoriohjelmiston validaation testisuunnitelma verifioitavaksi testiautomaatiolla. Lähdeohjelmisto perustuu suurelta osin kohteena olevaan potilasmonitoriohjelmistoon sekä sen toimintoihin ja siksi yhtäläisyyksiä löytyy monilta osa-alueilta. Lähteenä käytetty potilasmonitoriohjelmisto sisältää kuitenkin uusia ominaisuuksia ja uusia parametreja, verifikaatiossa käytettyjä simulaattoreita sekä mittausmoduuleita, joita kohteena oleva ohjelmisto ei tue.

Kohteena oleva potilasmonitoriohjelmisto on julkaistu ja käytössä eri markkina-alueilla. Ohjelmistolle tehdään säännöllisiä päivityksiä sen elinkaaren aikana ja suurin painopiste näillä päivityksillä on tietoturvaan liittyvien ominaisuuksien parantaminen. Suunnitelman luominen testiautomaatiolle sisältää pelkästään ohjelmiston validaatiossa olevat testitapaukset sekä näiden ohjelmistoarkkitehtuurisen alijärjestelmien kattavuuden.

Julkaistavia tietoturvapäivityksiä tullaan tekemään arviolta kaksi kertaa vuodessa vielä monen vuoden ajan, joten ohjelmiston validaatio tullaan suorittamaan kohteena olevalle ohjelmistolle useita kertoja. Yksi tietoturvapäivitys voi kattaa suuren määrän muutoksia ohjelmistoon. Automatisoimalla tämä prosessi ja suunnittelemalla luvussa 3 esitetty jatkuvan integraation testijärjestelmä, ohjelmiston validaation testiryhmä voidaan suorittaa muutoksia tehdessä jokaiselle versiolle välittömästi ilman suuria työmääriä testaajalta. Lähdeohjelmistossa on paljon eroja työn kohteena olevaan ohjelmistoon, jonka takia testiautomaatiojärjestelmää ei voida hyödyntää suoraan ja nämä erot on kirjattu myöhemmissä luvuissa. Tavoitteena oli kuitenkin hyödyntää mahdollisimman paljon lähdeprojektin testiautomaatiojärjestelmää.

6.1.1 Työn rajaus

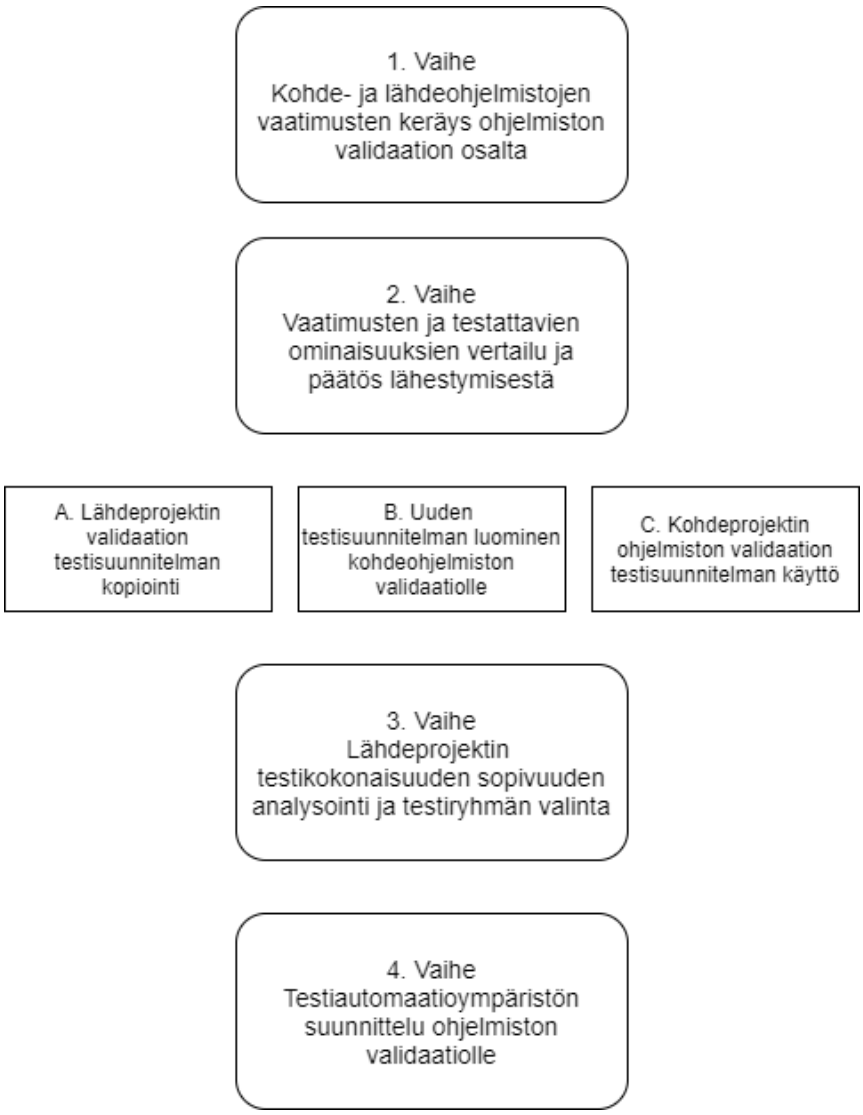
Projekti aloitettiin suunnittelulla ja rajaamalla työ kattamaan pelkästään ohjelmiston validaation testiryhmän valinta ja testiympäristön suunnittelu. Tavoitteena oli valita testiryhmä ohjelmiston validaatiolle, jossa testien muokkaaminen lähdeprojektista kohdeprojektille vaatisi mahdollisimman vähän työtä ja jossa kaikki potilasmonitorin kriittisimmät toiminnot ohjelmistoarkkitehtuurista verifioitaisiin. Tälle testiryhmälle suunniteltiin lisäksi käytettävä testiautomaatioympäristö. Sekä testiryhmän valinnassa että testiautomaatioympäristön suunnittelussa on useita eri huomionarvioisia päätöksiin vaikuttavia yksityis-

kohtia, joihin syvennytään tässä luvussa. Samassa alustuksessa myös suunniteltiin aikataulu sekä luotiin tarvittavat dokumentit kuten esimerkiksi päiväkirja, johon kirjattiin jokaisen päivän kohdalle suunnitellut ja toteutuneet tavoitteet, työn rajaus, hyödynnettävä lähdeprojekti sekä yhteyshenkilöt yrityksestä ohjaamaan, konsultoimaan ja arvioimaan tehtyjä analyyseja.

Jotta testiautomaatiota Robot Frameworkilla pystytään suorittamaan potilasmonitorille, täytyy rakentaa tuki potilasmonitoriohjelmistoon. Tuen avulla pystytään ohjaamaan potilasmonitoria testiautomaatiolla, palauttamaan potilasmonitorin käyttöliittymästä näkyviä elementtejä toimintojen verifioimiseen sekä kirjaamaan tietoa lokitiedostoihin. Työ ei sisällä monitorituen rakentamiseen liittyviä yksityiskohtia, eikä työn kohteena olevalla potilasmonitorilla ole työn teon hetkellä rakennettua tukea testiautomaatiolle.

6.1.2 Vaiheiden määrittely

Projekti jaettiin alustuksessa kuvassa 6 esitettyihin vaiheisiin. Tämän tarkoituksena oli selkeyttää projektin kulkua ja määrittää jokaiselle vaiheelle riittävät mittarit, minkä avulla voidaan seurata tavoitteiden toteutumista ja siirtyä seuraavaan vaiheeseen.



Kuva 6. Projektin vaiheet.

Ensimmäisessä vaiheessa kerättiin kohde- ja lähdepotilasmonitorin ohjelmiston validaatiossa verifioidut vaatimukset, jotta näitä pystytään käyttämään vertailuun sekä testitapauksia muokatessa testitapausten luokitteluun. Toisessa vaiheessa vertailtiin lähde- ja kohdeprojektien ohjelmiston validaatiossa verifioituja vaatimuksia sekä testikokonaisuutta lähtökohdan valitsemiseksi uuden testisuunnitelman luontiin. Kolmannessa vaiheessa tutkittiin yksityiskohtaisesti valitun lähtökohdan mukaisesti testitapauksia ja testikokonaisuuden sopivuutta kohdehjelmistolle. Neljännessä vaiheessa valittiin alustava testiryhmä, jonka suorittamalla ohjelmiston validaatiossa katetaan järjestelmän ohjelmistoarkkitehtuurin alijärjestelmät eli kriittisimmät toiminnot sekä suunniteltiin testiympäristö testiryhmän suorittamiseen.

Testiryhmällä viitataan testitapauksiin, jotka suoritetaan ohjelmiston validaatiossa. Testiautomaatioympäristöllä tarkoitetaan konkreettista testilaboratoriota, missä testattava laite on kiinnitetty eri testilaitteistoon ja tietokoneeseen, josta suoritetaan Robot Frameworkilla testitapauksia. Testisuunnitelmassa listataan ohjelmiston validaatiossa verifioidut testitapaukset sekä katetut ohjelmistoarkkitehtuurin alijärjestelmät.

6.1.3 Testiautomaatiolla saavutettu hyöty

Kohteena olevan ohjelmiston validaatio kattaa tällä hetkellä työmäärällisesti yhteensä 40 työpäivän manuaalisen verifikaatiotyön kolmelle eri potilasmonitorityypille, jossa testataan ohjelmiston useita eri osa-alueita käyttäen eri testitilanteita. Lähdeohjelmiston validaation lähtökohtana on ollut tämän prosessin nopeuttaminen kokoamalla ohjelmiston validaation testiryhmä, jonka pystyy suorittamaan korkeintaan yhdessä päivässä per monitori ja verifioi silti potilasmonitoriohjelmiston kaikki kriittisimmät toiminnot. Koska tietoturvapäivityksiä tullaan tekemään säännöllisesti, tullaan myös ohjelmiston validaatio suorittamaan työn kohteena olevalle potilasmonitorille useasti. Taulukossa 1 kuvataan laskelmaa testiautomaation tuomasta ajallisesta hyödystä, jos ohjelmiston validaatio tullaan suorittamaan vielä 10 kertaa.

Taulukko 1. Esimerkki ajallisesta säästöstä työpäivissä suorittamalla ohjelmiston validaatio testiautomaatiolla.

Ohjelmiston validaatio manuaalisesti	40	työpäivää
Ohjelmiston validaatio testiautomaatiolla	3	työpäivää
Ohjelmiston validaatio suoritetaan	10	kertaa
Manuaalinen verifikaatio kesto yhteensä	400	työpäivää
Testiautomaatio kesto yhteensä	30	työpäivää
Aikaa säästetty	370	työpäivää

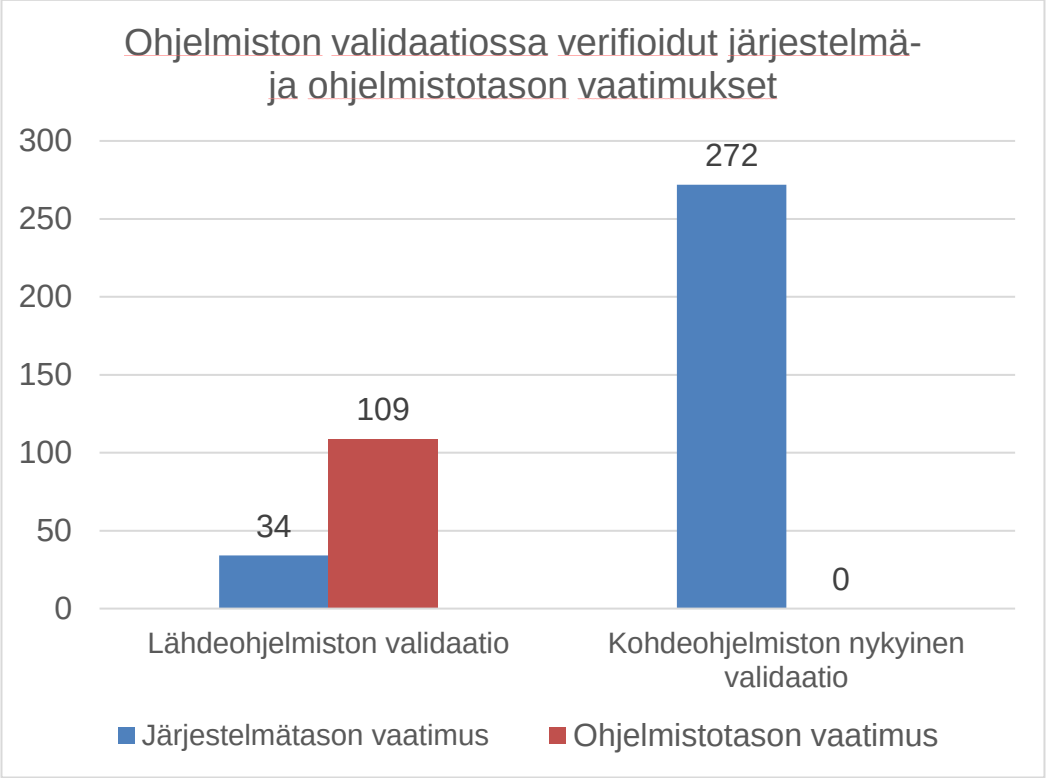
Ohjelmiston validaation manuaalinen verifikaatio kestää noin 40 työpäivää ja testiautomaatiolla suoritettavan ohjelmiston validaation tavoitekesto on yhteensä 3 päivää. Pel-

kästään yhden kerran suorittamalla säästetään jo 37 päivän verifikaatiotyö. Jos ohjelmiston validaatio tullaan suorittamaan testiautomaatiolla 10 kertaa, säästetään jo 370 työpäivän verifikaatiotyö. Muita hyödyntäviä tekijöitä on verifikaatioinsinöörin vapautuminen muihin työtehtäviin, kun testiautomaatio voidaan suorittaa ilman jatkuvaa seurantaa. Lisäksi testiautomaatiolla ohjelmistoa pystytään testaamaan jatkuvasti kehityksen aikana, jolloin ohjelmiston ollessa valmis pystytään luottamaan virallisen ja dokumentoitavan ohjelmiston validaation läpäisyyn ensimmäisellä kerralla eikä yllätyksiä ilmesty enää tässä vaiheessa.

6.2 Vaatimusten ja tiedon keräys

Ensimmäinen vaihe projektissa oli kerätä sekä kohteena että lähteenä käytetyn potilasmonitoriohjelmistojen vaatimukset, jotka ohjelmiston validaatiotestit verifioivat. Vaatimuksilla tarkoitetaan järjestelmä- sekä ohjelmistotason vaatimuksia, mitkä määrittävät potilasmonitorin ohjelmiston toiminnot. Järjestelmätason vaatimus on niin sanottu ylätason vaatimus, joka määrittää järjestelmän toiminnan yleisellä tasolla. Ohjelmistotason vaatimus määrittelee tarkemmin ohjelmiston toimintaa koskevia yksityiskohtia. Järjestelmätason vaatimus voi olla esimerkiksi ‘Järjestelmän tulee tukea sykkeen hälytysrajojen muokkaamista’ ja ohjelmistotason vaatimus esimerkiksi ‘Korkean sykkeen hälytyksen tulee aktivoitua, kun potilaan syke on noussut yli määritetyn hälytysrajan’.

Kohdeohjelmiston vaatimukset koostettiin komentosarjalla vaatimustenhallintatyökalusta ja käsiteltiin luettavaan muotoon. Tällä tavoin saatiin rajattua vain ohjelmiston validaatiossa verifioitavat vaatimukset. Lähdeprojektin ohjelmiston validaatiossa verifioidut vaatimukset ja kattavuus tarkistettiin testisuunnitelmasta. Testisuunnitelman testiryhmän automaatiotestit sisälsivät luokittelun siitä, mitkä vaatimukset mikäkin yksittäinen testi täyttää. Suunnitelma kattoi suoritettavat automaatiotestit, sekä mitä ohjelmistoarkkitehtuurin alijärjestelmiä nämä kattoivat. Kuvassa 7 on esitelty vaatimusten keräämisen tulokset, joista havaittiin, että potilasmonitoriprojektien välillä on eroja ohjelmiston validaatiossa.



Kuva 7. Ohjelmiston validaatiossa verifioidut järjestelmä- ja ohjelmistotason vaatimukset lähde- ja kohdeohjelmiston validaatiossa.

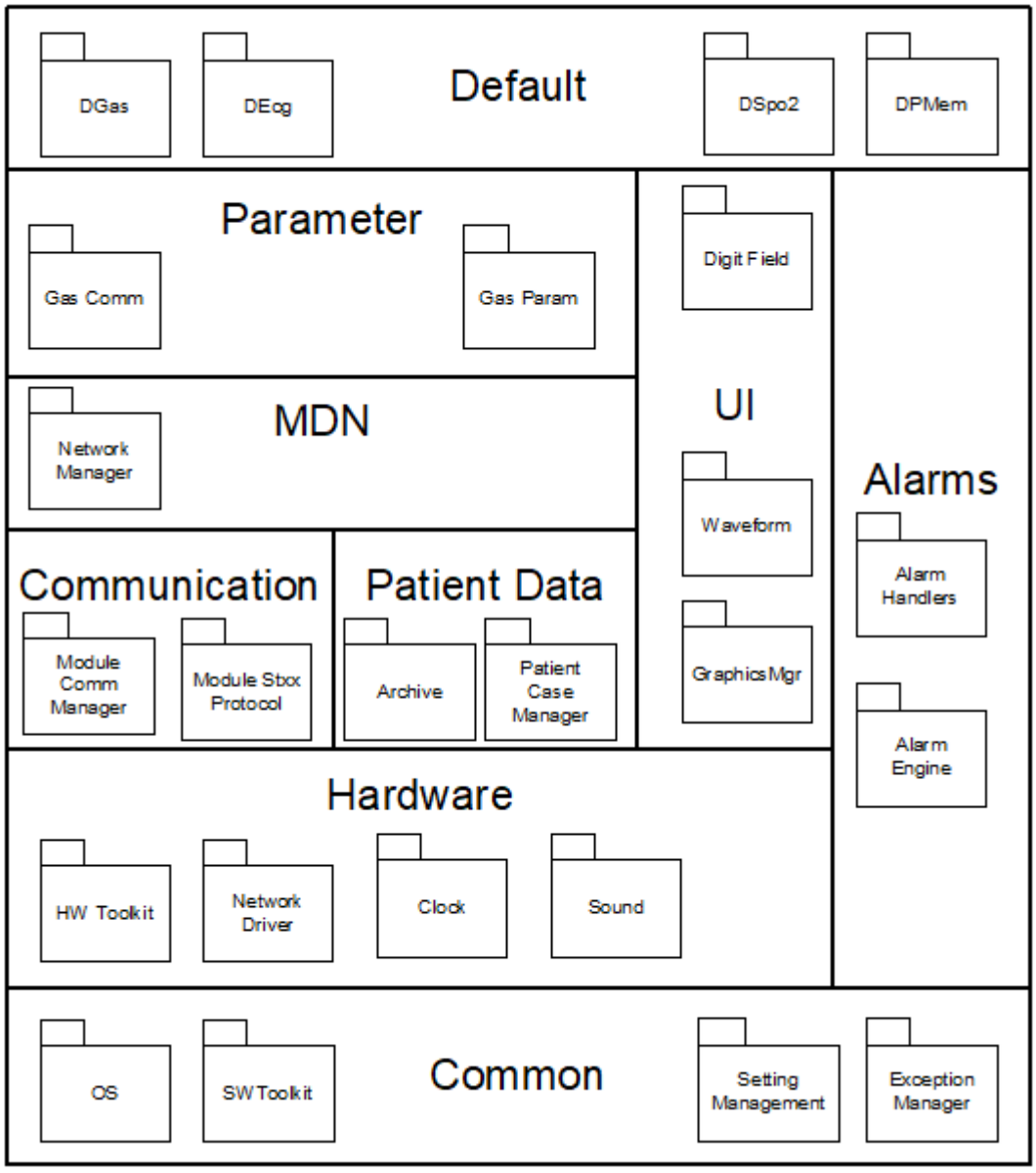
Lähdeprojektin ohjelmiston validaatio verifioi 34 järjestelmätason vaatimusta ja 109 ohjelmistotason vaatimusta, kun taas kohdeprojektissa verifioitiin 272 järjestelmätason vaatimusta. Olennaista on, että *FDA Design Controls* ei määrittele, mitä vaatimuksia ohjelmistosta tulee verifioida validaatiossa. Olennaisinta on kattaa ohjelmiston alijärjestelmien kriittisimmät toiminnot, jotta varmistutaan tuotteen luotettavasta toiminnasta. Vaatimukset voivat siis olla niin järjestelmätason kuin ohjelmistotason vaatimuksia.

6.3 Vaatimusten vertailu ja alustavan testiryhmän valinta jatkotarkasteluun

Vaatimukset koottiin vertailua varten, jotta saadaan ymmärrys, kuinka paljon samoja vaatimuksia ohjelmiston validaatio verifioi ja kuinka monelle eri vaatimukselle löytyy lähdeprojektista hyödynnettävä testiautomaatiolla verifioitava testitapaus. Tärkeintä oli ymmärtää vaatimusten puolesta näiden ohjelmistoprojektien erot.

Ohjelmiston validaation testiryhmä valitaan kattamaan kaikki keskeiset alijärjestelmät, kuten esimerkiksi oletusarvot, potilastiedot, graafinen käyttöliittymä, hälytykset, verkkoviestintä, huoltokäyttöliittymä, mittausmoduulien toiminta sekä eri parametriarvojen mittaus. Vaatimusten vertailu vahvisti sen, että molempien ohjelmistojen validaatio kattoi näitä samoja alijärjestelmiä potilasmonitoriohjelmistosta.

Vaatimuksien vertailusta havaittiin, että kohdeohjelmiston nykyinen manuaalinen validaatio verifioi näitä osa-alueita useilla eri muuttujilla, kun taas lähdeprojektissa tätä on suoraviivaistettu käyttämällä pienempää määrää muuttujia. Esimerkiksi kohdeprojektin validaatiossa potilasdatan lähetys potilasmonitoriverkossa eri potilasvalvontakeskuksiin ja muihin monitoreihin testataan useilla eri parametreillä, mikä selittää osaltaan vaatimuksien suurempaa määrää, joita kohdeprojektin manuaalinen validaatio verifioi. Ohjelmiston validaatio on kattavan verifikaation lisäksi suoritettava testiryhmä, jolla tehdään lisävarmistus kriittisimpien toimintojen toimivuudesta. Tärkeintä on kattaa eri alijärjestelmät ja monitorin kriittisimmät toiminnot. Kuvassa 8 on esitelty kohdeprojektin ohjelmistoarkkitehtuurin eri alijärjestelmät, joiden toimivuus verifioidaan ohjelmiston validaatiossa.

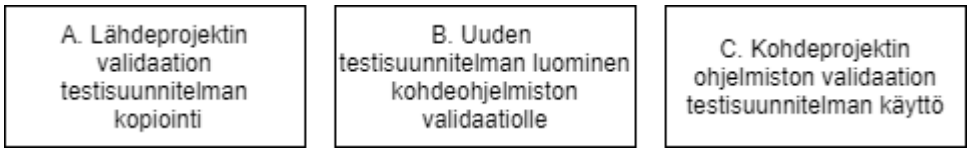


Kuva 8. Rakenteellinen näkymä työn kohteena olevan potilasmonitorin ohjelmistoarkkitehtuurista ja alijärjestelmistä.

Arkkitehtuuri koostuu eri alijärjestelmistä, joiden toiminta on riippuvaista toisistaan. Kuvassa järjestelmä on jaettu eri osiin, jotka sisältävät kuvauksen alijärjestelmistä. Eri alijärjestelmien toiminto on useasti riippuvainen toisista. Esimerkiksi parametrien (kuvassa *Parameter)* alijärjestelmä käsittelee parametrilta saatavan mittausdatan ja välittää sen edelleen käyttöliittymälle, joka sisältää kentät numeerisille arvoille sekä tarvittaessa aal-

tomuodoille, kuten esimerkiksi EKG:stä näytetään sen aaltomuoto sekä sykearvo. Oletusarvoista (kuvassa *Default*) verifioidaan monitorin asetuksien, hälytysrajojen sekä hälytysprioriteettien oletusarvoja. Osalla näistä on erikseen määrättyjä arvoja, joihin niiden tulee palautua esimerkiksi tehdasasetusten palautuksen jälkeen tai päättämällä potilastilanne ja aloittamalla uusi. Käyttöliittymän (kuvassa *UI, User Interface*) alijärjestelmästä verifioidaan, että monitori näyttää parametrien aaltokäyrät, arvot sekä muut grafiikat oikein. Hälytysten (kuvassa *Alarms*) alijärjestelmästä verifioidaan esimerkiksi eri hälytyksien aktivoituminen, indikaattorit, prioriteettien eskaloituminen sekä määritetyt viiveet.

Testiryhmää valittaessa oli tärkeää huomioida näiden edellä mainittujen alijärjestelmien kattavuus, ja siksi testiryhmän valinnan vaihtoehtoina käytettiin niin lähde- kuin kohdeprojektin ohjelmiston validaation testisuunnitelman testiryhmiä. Kuvassa 9 on esitelty eri lähestymistavat ohjelmiston validaation testisuunnitelman ja testiryhmän valintaan.



Kuva 9. Vaihtoehdot testiautomaatiolla verifioitavan ohjelmiston validaation alustavan testisuunnitelman luontiin.

Mahdollisuutena oli joko käyttää kokonaan lähdeprojektin ohjelmiston validaation testiryhmää ja muokata testitapaukset sopiviksi kohdeohjelmistolle, luoda uusi testisuunnitelma ja testiryhmä kohdeohjelmiston validaatiolle hyödyntäen lähdeprojektin testijärjestelmää tai käyttää kohdeprojektin ohjelmiston validaation testisuunnitelmaa ja etsiä näille automaatiotestit lähdeprojektista tai luoda uudet.

Jokaisessa vaihtoehdossa oli tiettyjä piirteitä, joiden toteutuessa vaihtoehtoa olisi johdonmukaista hyödyntää. Jotta vaihtoehtoa A eli lähdeprojektin testisuunnitelmaa voisi hyödyntää kokonaan, tulisi testattavien vaatimusten ja testiautomaatiojärjestelmässä käytettyjen moduulien, parametrien, ohjelmiston toimintojen sekä simulaattoreiden olla yhteensopivia tai helposti vaihdettavissa yhteensopiviin kohdeprojektille. Tämä analyysi ja tulokset on kirjattu lukuun 6.4. Tällä vaihtoehdolla testikokonaisuus olisi valmiiksi hyödynnettävissä ja muokattavissa sekä voitaisiin yksinkertaisesti varmistua siitä, että testisuunnitelma kattaa ohjelmistoarkkitehtuurin eri alijärjestelmät.

Hybridi-vaihtoehdossa B muodostettaisiin uusi testiryhmä kohdeohjelmiston validaatiolle. Tämä tarkoittaa, että hyödynnetään mahdollisimman paljon olemassa olevia testitapauksia lähdeprojektin ohjelmiston validaatiosta ja sopimattomien testien tilalle tulisi joko löytää tai luoda uudet testitapaukset. Tämä vaihtoehto antaisi eniten vapauksia projektin tekemiseen sekä testitapausten valitsemiseen. On kuitenkin mahdollista, että osan testitapauksista ollessa sopimattomia kohdeohjelmistolle olisi tarvittavaa tehdä uusi analyysi testiryhmän arkkitehtuurisesta kattavuudesta eri alijärjestelmille sekä pätevyydestä ohjelmiston validaation testiryhmäksi.

Vaihtoehto C on se, että käytetään kohdeprojektin olemassa olevaa ohjelmiston validaation testisuunnitelmaa, jossa nykyiset testit on suoritettu manuaalisesti. Testitapaukset tulee etsiä lähdeprojektista ja muokata sopiviksi tai luoda näille vaatimuksille uudet testiautomaatiolla suoritettavat testitapaukset. On myös mahdollista supistaa vaatimusten määrää ja kohdistaa se kattamaan laajasti alijärjestelmät, mutta vähentää turhaa toistoa. Tämä vaihtoehto voi osoittautua haastavaksi, jos lähdeprojektin testiautomaatiojärjestelmästä näille vaatimuksille sopivia testejä ei löydy. Tässä tapauksessa testien tekeminen veisi paljon aikaa, ja projektin päätarkoitus eli kustannustehokkuus kärsisi. Tämä vaihtoehto oli mukana suunnitelmaa luodessa, mutta vaihtoehtoja analysoidessa osoittautui nopeasti, että tämä vaihtoehto tulisi viemään kaikista eniten aikaa ja siksi karsiutui pois jo aikaisessa vaiheessa.

Koska tärkeintä ohjelmiston validaatiolle on kattaa ohjelmistoarkkitehtuurin eri alijärjestelmät ja kriittisimmät toiminnot, on haastavaa suunnitella ohjelmiston validaatio pelkästään katettujen vaatimusten puolesta. Mikäli tarkoituksena olisi ollut luoda testiautomaatio alusta alkaen kohdeohjelmistolle ilman lähdeprojektista hyödynnettävää testijärjestelmää, olisi ollut mahdollista lähestyä suunnitelman tekoa enemmän vaatimusten näkökulmasta käyttäen vaihtoehtoa C.

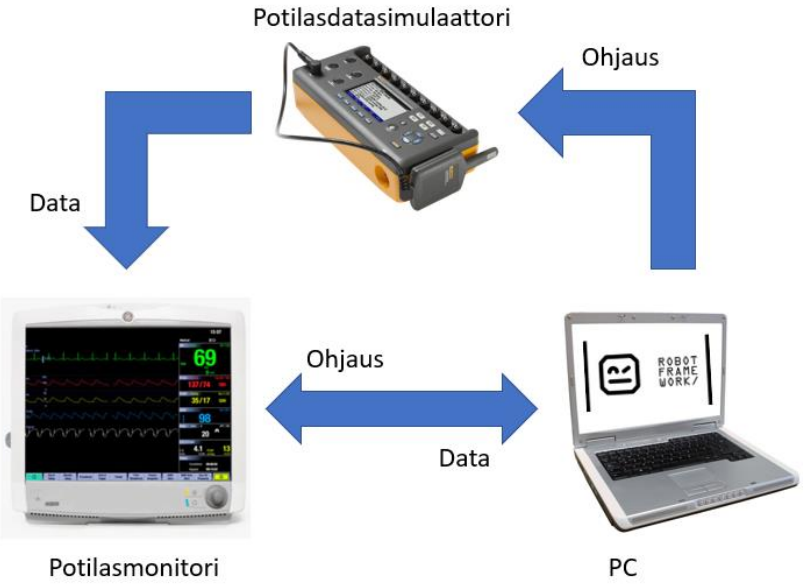
Lähdeohjelmistossa käytetään potilasmonitoriverkossa tiedonsiirtoon vain yhtä verkkoprotokollaa, jonka avulla potilasmonitori siirtää verkon yli dataa keskuksiin sekä toisille potilasmonitoreille. Työn kohteena oleva potilasmonitoriohjelmisto pystyy lähettämään dataa verkossa kahdella eri verkkoprotokollalla, koska se on yhteensopiva kahden eri keskuksen kanssa, joita markkinoilla on käytössä. Tämä eroavaisuus oli tiedossa, kun suunnitelmaa käytiin luomaan ja oli selkeästi nähtävissä vaatimuksia vertaillessa. Koska testitapauksia tälle verkkoprotokollalle ei ole olemassa, päätettiin jo tässä vaiheessa

suorittaa ohjelmiston validaatiossa tämän osan testaaminen manuaalisella verifikaatiolla. Lisäksi lähdeohjelmiston testiryhmästä puuttuivat testitapaukset, joissa tulostetaan verkon yli tulostimeen esimerkiksi vitaaliparametrien aaltokäyriä tai arvoja. Haluttiin myös tälle lisätä manuaalinen verifikaatio. Manuaalisen verifikaation testiryhmä on kirjattu luvussa 6.4.4.

Suurin osa vaatimuksista, joita lähdeohjelmiston validaatio kattoi, oli myös vaatimuksia työn kohteena olevalle potilasmonitoriohjelmistolle. Lähtökohtaisesti siirryttiin yksittäisten testitapausten tarkasteluun käyttäen perustana kokonaan lähdeprojektin testisuunnitelmaa eli vaihtoehtoa A. Tämä testiryhmä sisältää 14 eri testitiedostoa, joiden sisällä on useita testitapauksia. Osa testitapauksista vaatii muutoksia eriävien vaatimusten takia, mutta vaatimusten ja testijärjestelmän tutkimuksen jälkeen on selvää, että on nopeampaa vaihtaa yksittäinen testitapaus toiseen kuin etsiä uudelleen kaikki testitapaukset, jotka sopivat kohdeohjelmistolle.

6.4 Lähdeprojektin testikokonaisuuden analysointi

Jotta testitapauksia voidaan hyödyntää työn kohteena olevalle potilasmonitorille, täytyy suunnitella testijärjestelmä (Kuva 10), jota testiautomaation suorittamiseen tarvitaan. Koko testiautomaatioympäristö koostuu pääpiirteittäin tietokoneella olevista Robot Frameworkilla toteutetuista testitapauksista, kirjastoista, python-skripteistä sekä konkreettisesta testattavasta laitteistosta. Kuvassa 10 on testiautomaatioympäristö, jossa potilasmonitoria voidaan testata. Kun testin kohde on erillinen laite eikä tietokoneella oleva ohjelmisto, täytyy sille rakentaa erillinen testiautomaatioympäristö, jossa sitä testataan.



Kuva 10. Esimerkki testiympäristöstä testiautomaation suorittamiseen potilasmonitorille.

Kuvan 10 testiympäristössä tietokone ohjaa potilasmonitoria, ja potilasmonitori lähettää dataa tietokoneelle. Tämän datan avulla verifioidaan, että potilasmonitori reagoi oikein testitapauksissa asetettuihin tiloihin. Potilasmonitorille lähetetään dataa potilasdatasimulaattorilta, jota tietokone myös ohjaa. Potilasdatasimulaattorista kiinnitetään mittauskaapeleita potilasmonitorissa oleviin mittausmoduuleihin. Testiympäristöissä käytetään myös tietokoneella ohjattavia ohjelmistosimulaattoreita, joiden avulla voidaan simuloida vitaalitoimintojen lisäksi moduulien vikatilanteita.

Testitapauksista selvitettiin testien suorittamiseen tarvittava laitteisto sekä testin sopivuus työn kohteena olevalle ohjelmistolle. Testiryhmän eli erillisten testitapausten sopivuuden analysoinnin lisäksi testijärjestelmän muiden osa-alueiden sopivuus täytyi analysoida. Yhtenä isoimpana ongelmana oli lähde- ja kohdepotilasmonitorien erilainen huoltokäyttöliittymä.

6.4.1 Huoltokäyttöliittymä

Huoltokäyttöliittymää hyödynnetään testiautomaatiossa esimerkiksi alustamaan potilasmonitori eri käyttöasetuksiin sekä palauttamaan testitapausten välillä potilasmonitori tehdasasetuksiin. Useat testit hyödyntävät tätä huoltokäyttöliittymää ja tämän käyttö tulee olemaan myös erittäin tärkeää kohteena olevan ohjelmiston validaatiossa alustamaan testejä oikeisiin asetuksiin ja tiloihin. Kun yksittäisiä testitapauksia aloitettiin tutkimaan, oli huoltokäyttöliittymä usean testitapauksen alustuksessa käytössä. Tämän takia ohjelmistolle päätettiin luoda konseptitodistus eli *proof-of-concept* -tyylinen testitapaus Robot Frameworkilla. *Proof-of-concept*-testillä tarkoitetaan prototyyppiä testitapauksesta, jonka suorittamisella varmistetaan, että työn kohteena olevan ohjelmiston huoltokäyttöliittymää pystytään ohjaamaan Robot Frameworkin komennoilla.

Koska kohteena olevan potilasmonitorin ohjelmiston huoltokäyttöliittymä erosi suurilta osin lähdeohjelmistossa käytettävästä, oli tärkeää selvittää, kuinka tämän ohjaus voidaan toteuttaa. Huoltokäyttöliittymää voidaan hallita kahdella eri tavalla suoraan monitorista käyttäen potilasmonitoriohjelmistoon sisäänrakennettua selainta tai etähallinnalla tietokoneelta käyttäen verkkoselainta. Näiden välillä ei ole eroja toiminnallisuuden suhteen, eli kummallakin on mahdollista säätää asetukset samalla tavalla ja saavuttaa sama lopputulos. Koska projektin aikana monitorin tukea testiautomaatiolle ei ollut vielä toteutettu, oli mahdollista toteuttaa huoltokäyttöliittymän hallinta etänä käyttäen verkkoselainta, mikä ei vaadi potilasmonitorilta tukea testiautomaatiolle. Tällä tavoin oli mahdollista luoda *proof-of-concept*-tyylinen varmistus siitä, että etähallinnan toiminnallisuus on toteutettavissa myös lopullisessa testien implementaatiossa osana muita testejä.

Robot Frameworkilla on mahdollisuus käyttää ulkoista SeleniumLibrary-kirjastoa, joka on tarkoitettu web-testaukseen. SeleniumLibrary-kirjasto sisältää valmiit avainsanat web-elementtien ohjaukseen ja hallintaan, joka suoraviivaistaa testien kirjoittamista. SeleniumLibraryn käyttö perustuu eri web-elementtien ohjaamiseen, jotka täytyy paikallistaa kirjastosta löytyvin avainsanoin käyttäen esimerkiksi XPath-syntaksia tai eri HTML-elementtien tunnisteita. Kun haluttu elementti on paikallistettu, voidaan sille määrittää toiminto, kuten esimerkiksi klikkaa painiketta tai syötä salasana sille kuuluvaan tekstikenttään. Robot Frameworkin ja SeleniumLibraryn lisäksi tarvitaan jokaiselle selaimelle oma WebDriver-liitännäinen, jotta SeleniumLibrary tunnistaa käytettävän selaimen.

Huoltokäyttöliittymän hallinnan toteutus suoritettiin Firefox-selaimella käyttäen sille tarkoitettua gecko-WebDriver liitännäistä. [17.]

Huoltokäyttöliittymän etäohjauksen toiminnallisuutta aloitettiin rakentamaan asteittain. Aluksi luotiin ensimmäinen testitiedosto sisäänkirjautumistestille ja sille oma resurssitiedosto, johon talletettiin kaikki usein käytetyt muuttujat. Monitorin huoltoliittymään etäyhteyden luomiseksi tarvittiin monitorin IP-osoite. Huoltoliittymälle on eri käyttäjäprofiileja, joilla jokaiselle on omat kirjautumistunnukset. Ensimmäiseksi pyrittiin toteuttamaan pelkästään yksinkertainen sisäänkirjautuminen huoltoliittymään service-käyttäjällä. Tätä varten jokainen vaadittava web-elementti kuten käyttäjätunnus- ja salasanatekstikenttä tuli paikallistaa käyttäen SeleniumLibraryn mukaisia lokaattoreita. Johdonmukaisimmaksi vaihtoehdoksi osoittautui XPath-lokaattorin käyttö. Sen avulla voidaan määrittää jokainen elementti selkeästi. Kun tarvittavat elementit oli paikallistettu, osoitettiin näille toiminnot kuten kirjoita käyttäjätunnus sekä salasana ja paina sisäänkirjautumispainiketta.

Huoltokäyttöliittymä koostuu eri välilehdistä ja painikkeista. Jokaiselle tarvittavalle web-elementille luotiin oma muuttuja, johon tallennettiin tämän sijainti XPath-formaatissa. Kaikille käytettäville huoltoliittymän elementeille luotiin omat muuttujat, joiden arvot sisältävät sijainnin XPath-formaatissa. Esimerkkikoodissa 4 on alustettu salasanatekstikentälle muuttuja, jonka arvo on elementin sijainti XPath-formaatissa.

```
${PASSWORD_INPUT}        //input[@name='pass']
```

Esimerkkikoodi 4. Esimerkki muuttujasta, jolla on Xpath-arvo.

Nämä muuttujat koostettiin yleiseen resurssitiedostoon, jotta näitä voidaan hyödyntää jatkossa helposti toiminnallisuutta lisätessä.

Koska huoltokäyttöliittymä on salasanoin suojattu, oli sisäänkirjautumisen testaamisen luonti edellytys muiden toiminnallisuuksien testaamisille. Kun tämä oli onnistuneesti toteutettu, oli mahdollista testikokonaisuutta laajentaa lisää. Monissa eri testitapauksissa huoltokäyttöliittymää käytetään vaihtamaan monitorin ohjelmistopaketti. Eri ohjelmistopaketeilla on omat asetuksensa vastaamaan tiettyä käyttötarkoitusta tai käyttöympäristöä kuten leikkaussali tai vastasyntyneiden intensiivinen hoito. Tästä syystä seuravaaksi

tuli toteuttaa ohjelmistopaketin vaihto. Koska aiemmin oli luotu resurssitiedosto, joka sisältää tarvittavat muuttujat web-elementeille, oli toiminnallisuutta helppo laajentaa luomalla oma testitiedosto ohjelmistopaketin vaihdolle.

Esimerkkikoodi 5 sisältää kaksi erillistä testitapausta, joista ensimmäinen tarkistaa, että kaikki ohjelmistopaketit ovat saatavilla ja toinen vaihtaa valitun ohjelmistopaketin.

```
*** Settings ***
Documentation     This test verifies that SW packages can be
changed.
Resource          ../resources/resource.robot
Test Setup        Open Login Page
Test Teardown     Close Browser

*** Test Cases ***
ATC all sw packages are available
    [Documentation]    Verifies that all sw packages are shown.
    [Tags] Example requirement
    Login To ServiceInterface
    Navigate To Sw Packages
    Verify That All Sw Packages Are Available

ATC OR sw package can be selected
    [Documentation]   Verify that OR sw package can be selected
    Login To ServiceInterface
    Navigate To Sw Packages
    Select Sw Package      ${OR_PACKAGE}
    Restart Monitor
    Login To ServiceInterface
    Check Sw Package
    Page Should Contain    ${MON_TYPE}ORP
```

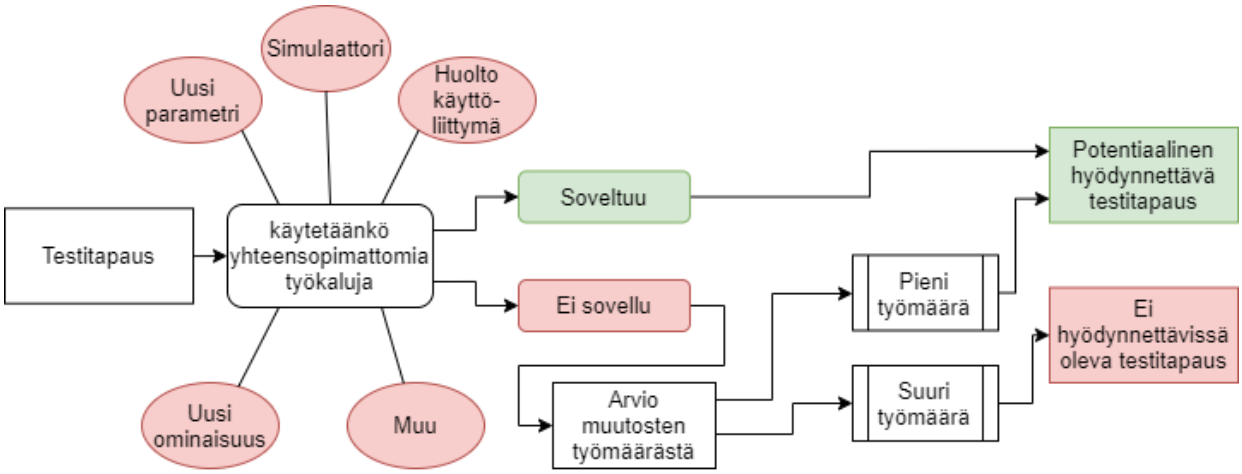
Esimerkkikoodi 5. Huoltokäyttöliittymän ohjaamisen testaamiseksi luotu testitapaus.

Settings -kohdassa kutsutaan aiemmin luotua resurssitiedostoa, joka sisältää kaikki tarvittavat muuttujat (V*ariables)* ja avainsanat (*Keywords*) testien suorittamiseksi. Samalla on myös määritelty yksittäisen testitapauksen *Test Setup* -komento, jossa testin aluksi avataan verkkoselain ja siirrytään monitorin IP-osoitteeseen huoltokäyttöliittymään kirjautumiseksi. *Test Teardown* -komento sulkee verkkoselaimen testitapauksen lopuksi. Ensimmäinen testitapaus kirjautuu sisään huoltokäyttöliittymään oikeilla kirjautumistunnuksilla ja navigoi ohjelmistopakettien valikkoon, jossa tarkastetaan, että kaikki paketit ovat saatavilla. Toinen testitapaus kirjautuu sisään huoltokäyttöliittymään ja valitsee ennalta määritetyn ohjelmistopaketin, joka on esimerkkikoodissa valittu leikkaussaliksi (${*OR_PACKAGE}*). Koska monitori joudutaan käynnistämään uudelleen muutoksen ak-

tivoimiseksi, testi käynnistää monitorin uudelleen ja odottaa 90 sekuntia ennen uutta kirjautumista. Tämän jälkeen vahvistetaan huoltokäyttöliittymän tiedoista, että oikea ohjelmistopaketti on aktivoitu.

6.4.2 Yksittäisten testitapausten analysointi

Kun huoltokäyttöliittymän ohjaamisesta saatiin varmuus, käytiin jokainen testitapaus läpi muidenkin yhteensopimattomien työkalujen tunnistamiseksi ja testitapaukset jaoteltiin hyödynnettävissä oleviin, ja suuren työmäärän muutoksia vaativat ei hyödynnettävissä oleviin. Kuvassa 11 esitellään prosessi, jonka avulla tutkitut testitapaukset luokiteltiin hyödynnettävissä oleviin testitapauksiin.



Kuva 11. Yksittäisen testitapauksen yhteensopivuuden analysointi.

Testitapauksien käyttäessä yhteensopimattomia työkaluja suoritettiin arvio työmäärästä muutosten tekoon, jotta testitapausta pystyttäisiin hyödyntämään kohteena olevalle ohjelmistolle. Jos työmäärä ei ollut suuri, hyväksyttiin nämä testitapaukset potentiaalisiksi ohjelmiston validaation testeiksi. Nämä olivat esimerkiksi käytettyjen simulaattoreiden tai moduulien vaihtamista toiseen. Kun muutoksia tarkastettiin, arvioitiin samalla, muuttuuko testitapaus alkuperäisen tarkoituksen tai katettujen vaatimusten osalta.

Testitapausten analysoinnista paljastui yksityiskohtia, jotka vaativat muokkauksia tai työkalujen vaihtoa toisiin, jotta testit voidaan siirtää kohdeohjelmistolle. Keskeisiä yksityiskohtia olivat esimerkiksi:

- yhteensopimaton ohjelmistosimulaattori
- yhteensopimaton mittausmoduuli
- hälytysäänten kirjaus lokitiedostoihin
- resurssitiedostot muuttujille
- testien alustukseen käytetyt init-tiedostot
- elementtien tunnisteet ohjelmiston ohjaukseen.

Ohjelmistosimulaattori, jolla simuloitiin pulssioksimetriaa sekä SPI (*Surgical Pleth Index*) -parametria ei ole yhteensopiva kohdeohjelmistolle. SPI-parametria käytetään anestesiassa kuvaamaan nukutetun potilaan kipureaktiota numeerisesti. Tällä tavoin voidaan määrittää leikkauksen aikana nukutetun potilaan anestesian riittävyyttä. [18.] Käytetty ohjelmistosimulaattori vaatii tietyn mittausmoduulin, joka ei myöskään ole yhteensopiva kohdeohjelmistolle. Näille yhteensopimattomille laitteille tutkittiin vaihtoehtoista ratkaisua. Koska testitapaukset on kirjoitettu korkean ja matalan tason avainsanoilla, on testitapaukset helppo vaihtaa toiselle simulaattorille. Testitapaukset sisälsivät korkean tason avainsanoja kuten esimerkiksi ‘Simulate SpO2 Pulse Rate 98’ ja ohjelmistosimulaattorin tarkoin määrätty toiminta oli kirjattu sen ohjaukseen liittyviin resurssitiedostoihin. Muutokseen riittää käytettävän simulaattorin testikirjaston vaihto haluttuun ja syntaksin tarkistus, jotta testitapaus on suoritettavissa toisella simulaattorilla. Hyvin rakennettu testijärjestelmä on helposti muokattavissa esimerkiksi tällaisten tilanteiden kohdalla.

Lähdeprojektin testeissä käytetty lisämoduuli on yhteensopimaton kohdeprojektille ja vaatii moduulin vaihdon sopivaan. Kohdeprojektissa käytetään tämän moduulin sijasta toista mittausmoduulia, joka tukee testiryhmälle tarvittavia toimintoja ja sisältää kytkennät EKG-, SpO2-, invasiivipaine-, lämpötila- ja verenpainemittauksille. Muutoksia testitapauksiin käytetyn mittausmoduuliin takia ei tarvita, koska potilasmonitori vastaanottaa dataa simulaattorista moduulin kautta, eikä mittausmoduulia ohjata ollenkaan testitapauksissa.

Testiryhmä myös sisälsi testejä hälytysäänten verifiointiin, mikä vaati ohjelmistolta hälytystilanteista tietynlaista kirjausta ohjelmiston lokitiedostoihin. Aluksi näitä testitapauksia oltiin pudottamassa pois valitusta testiryhmästä, koska kohdeohjelmisto ei kirjaa lokeihin hälytysääniä samoilla yksityiskohdilla. Hälytysäänten kirjaus lokitiedostoihin on kuitenkin olennainen toiminto verifioida ohjelmiston validaatiossa ja tätä hyödynnetään myös muissa testitapauksissa, joissa verifioidaan eri hälytyksen aktivoitumista. Mahdollista oli

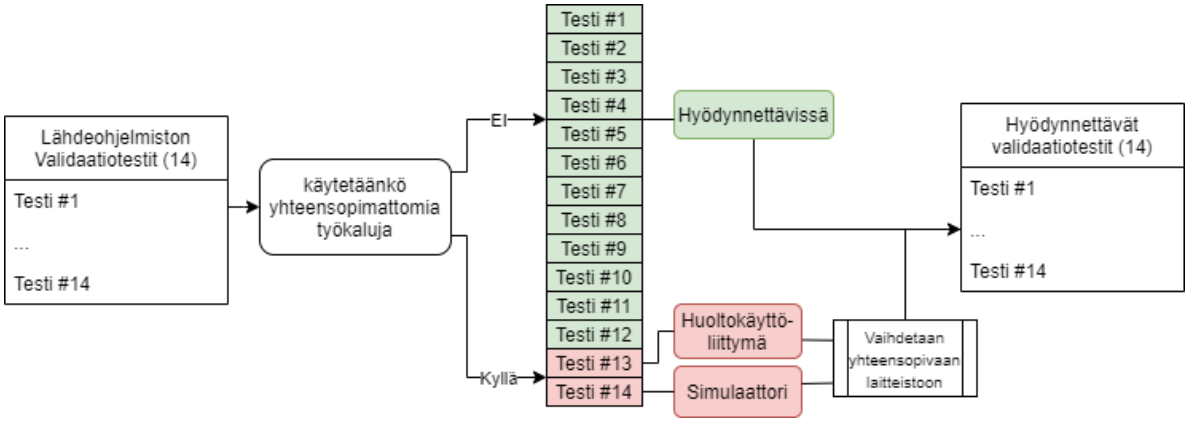
myös suorittaa pelkästään hälytysäänten verifikaatio manuaalisesti muun testiautomaatiolla verifioitavan testiryhmän lisäksi. Konsultoinnin ja työn määrän arvioinnin jälkeen pidettiin kuitenkin parhaimpana vaihtoehtona lisätä potilasmonitoriohjelmistoon hälytysäänten kirjaaminen hälytystilan aikana potilasmonitorin lokitiedostoihin. Tämä muutos ohjelmistoon on mahdollista toteuttaa pienellä työmäärällä ja siten pystytään edelleen automatisoida mahdollisimman paljon ohjelmiston validaation testiryhmästä.

Testitapaukset käyttivät erillistä resurssitiedostoja muuttujille. Näitä oli esimerkiksi asetusten oletusarvot, sallitut hälytysprioriteetit ja potilasmonitorin ohjaukseen liittyvät tunnisteet. Muuttujat näille tiedostoille vaihdettiin vastaamaan kohdeohjelmiston arvoja.

Testin alustukseen käytetään _init_ tiedostoja, jotka sisältävät *Suite Setup*- ja *Suite Teardown* -komennot. Osa testitapauksista alustettiin hyödyntämään yllä mainittua yhteensopimatonta ohjelmistosimulaattoria, ja näiden testien alustukset täytyy vaihtaa käyttämään yhteensopivaa potilasdatasimulaattoria.

Robot Framework käyttää navigointiin elementtien, eli painikkeiden ja valikoiden tunnisteita pystyäkseen ohjaamaan potilasmonitoria. Työn kohteena olevasta ohjelmiston lähdekoodista ja lokitiedostoista tarkastettiin, onko kaikkiin käyttöliittymän elementteihin lisätty yksilöivä tunniste keskittyen valikkorakenteisiin, jotka oletettavasti ovat käytössä ohjelmiston validaation testitapauksissa. Puuttuvia tunnisteita oli noin 40, jotka koostettiin listaksi, jotka ohjelmistonkehittäjä lisää ohjelmistoon monitorituen toteutuksen yhteydessä.

Kun kaikki testitapaukset oli analysoitu, saatiin kokonaiskuva, kuinka hyvin lähdeprojektin ohjelmiston validaation testiryhmää pystytään hyödyntämään kohdeohjelmistolle. Kuvassa 12 on esitelty analyysin tulokset.



Kuva 12. Uuden ohjelmiston validaatiotestitapausten hyödyntäminen kohdeohjelmistolle.

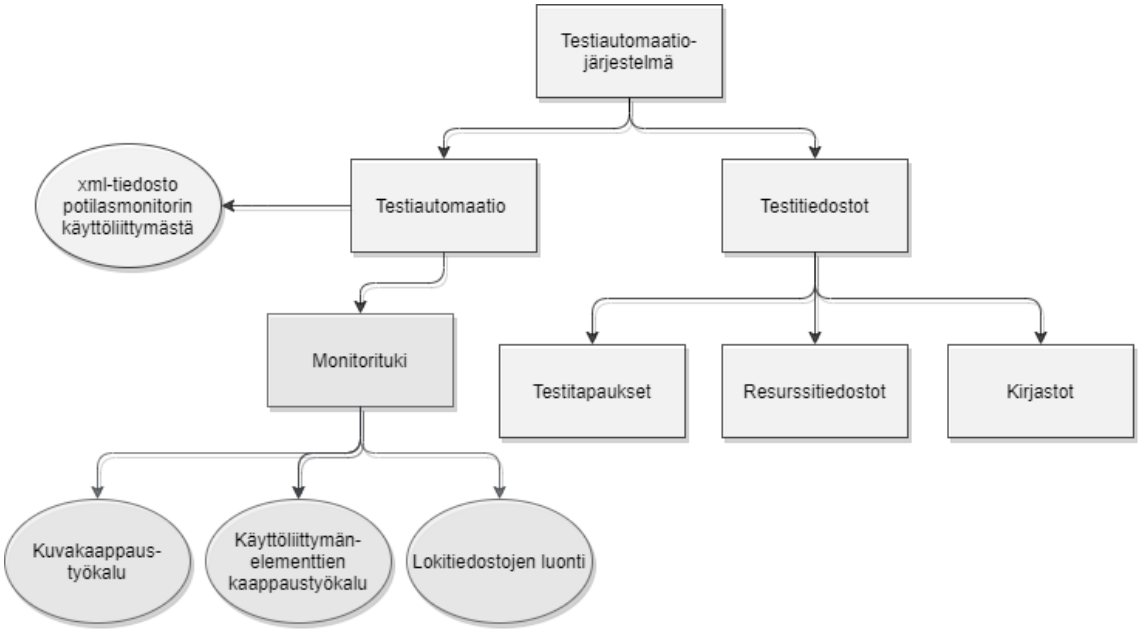
Lähdeprojektin 14:sta ohjelmiston validaatiossa olevista testitiedostoista oli 12 hyödynnettävissä niin, että vain pieniä muutoksia vaadittiin testitapausten muokkaamiseen sopivaksi kohdeohjelmistolle. Näitä muutoksia olivat esimerkiksi yllä mainitut elementtien tunnisteet, hälytysäänten lokikirjauksen muutos ohjelmistoon ja resurssitiedostojen muuttujien muokkaus. Kahdessa testitapauksessa, joissa muutoksien arvioitiin olevan melko suuria, arvioitiin muutosten tekemisen olevan kuitenkin pienempi työmäärä kuin etsiä toinen testitapaus, joka kattaisi saman alijärjestelmän. Toinen käytti ohjelmistosimulaattoria, joka ei ole yhteensopiva kohteena olevalle potilasmonitorille, ja toinen testitapaus hyödynsi testin alustuksen lisäksi testissä laajasti lähdeohjelmiston huoltokäyttöliittymää.

6.4.3 Monitorituki ja työkalu monitorin ohjaukseen testiautomaatiossa

Automaatiotestien suorittamiseksi potilasmonitorille tarvitaan tuki, joka implementoidaan suoraan potilasmonitorin ohjelmistoon. Monitorituki on järjestelmä, joka mahdollistaa automaattisen vuorovaikutuksen monitorin ja Robot Framework -testikirjastojen välillä. Tämä tukityökalu tarjoaa myös kuvakaappaustyökalun kirjaamaan lokitiedostoihin tarvittavia yksityiskohtia sekä työkalun kaappaamaan käyttöliittymässä näkyvät elementit JSON-tiedostomuodossa.

Tunnistettu muutos testitapauksiin oli käyttöliittymän elementtien yksilöivien tunnisteiden vaihtaminen kohdeprojektin mukaisiin tunnisteisiin. Elementit ovat esimerkiksi painik-

keita, vierityspalkkeja tai erilaisia valikoita, joiden avulla muutetaan asetuksia ja navigoidaan potilasmonitorin valikoissa. Näytöllä näkyvästä datasta pystytään verifioimaan monitorin näytöstä käyttöliittymän ja elementtien kaappaus työkalulla. Tämä työkalu on osa monitoritukea, joka monitorille luodaan, jotta automaatiotestejä pystytään suorittamaan. Monitorituki tullaan implementoimaan potilasmonitorille testisuunnitelman hyväksyttämisen jälkeen, jolloin varmuus testiautomaation implementoinnin tuomista hyödyistä on varmistettu. Kuvassa 13 on esitelty testiautomaatiojärjestelmä, jonka osa monitorituki on ja jonka avulla Robot Frameworkilla testitapauksia pystytään suorittamaan potilasmonitorille.



Kuva 13. Testiautomaatiojärjestelmä testiautomaation suorittamiseen potilasmonitorille.

Testitiedostojen lisäksi tarvitaan muita testiautomaation työkaluja. Potilasmonitorin etäohjaukseen tarvitaan xml-tiedostoa potilasmonitorin käyttöliittymästä ja elementeistä puukaaviona, koska tätä ohjelmistoa ei pystytä esimerkiksi web-pohjaisille sivuille tarkoitetulla SeleniumLibrarylla ohjaamaan. Tämä tiedosto on eräänlainen kartta ohjelmiston valikkorakenteesta, ja sitä käytetään testiautomaatiossa navigointiin sekä käyttöliittymän hallinnointiin. Suunnitelman tekemiseen kuului myös tämän xml-dokumentin luominen. Lähdeprojektin xml-tiedosto koostui yli 5000 rivistä yksilöiden lähes jokaisen käyttöliittymän elementin, ja sitä käytettiin pohjana xml-tiedoston rakentamiseen kohteena olevalle potilasmonitoriohjelmistolle. Käyttöliittymästä tehty xml-tiedosto kuvaa käyttöliittymän

valikot, painikkeet ja muut kohteet solmuina eli *nodeina. Node* sisältää vähintään kolme attribuuttia, *name* eli yksilöivä nimi, jolla testitapauksista solmua kutsutaan, *ui_name* on ohjelmistossa oleva yksilöivä tunniste, ja *type* kertoo elementin tyypin.

Kuvassa 14 on osa xml-tiedostosta hälytysvalikon rakenteesta solmuina. Koska ohjelmiston validaatiossa navigointia tarvitaan vain osaan valikkorakenteesta, priorisoitiin muutokset näihin tiedossa oleviin valikkorakenteiden osuuksiin. Koska navigaatio käyttää puurakennetta, eli navigoi solmujen *Node* kautta halutun elementin luokse, muokattiin xml-tiedosto noudattamaan työn kohteena olevan potilasmonitoriohjelmiston valikkorakennetta. Halutun elementin ollessa usean painikkeen takana, käy navigaatio läpi polun kaikki elementit ja tiedoston tulee noudattaa tarkasti tätä valikkorakennetta. Esimerkiksi navigoidakseen kuvassa 14 olevan ALRMLMT_DEFAULT -painikkeen sisältävään valikkoon täytyy ensin navigoida ALARMS sekä ALARMS_ALRMLMT painikkeiden kautta.

```
<!-- Home button -->
<node name="HOME" ui_name="Normal\nScreen:MainMenu" type="close" />
<!-- Alarm setup -->
<node name="ALARMS" ui_name="Alarms\nSetup:MainMenu" type="menu">
    <!-- Alarm setup => Alarm limits -->
    <node name="ALARMS_ALRMLMT" ui_name="Tab_Alarm\nLimits:Alarms Setup" type="tab">
        <node name="ALRMLMT_PREVIOUS_TAB" ui_name="Prev:Alarm\nLimits:Alarms Setup" type="button" />
        <node name="ALRMLMT_AUTO" ui_name="Auto Limits:Alarm\nLimits:Alarms Setup" type="button" />
        <node name="ALRMLMT_DEFAULT" ui_name="Default Limits:Alarm\nLimits:Alarms Setup" type="button" />
        <node name="ALRMLMT_UNDO" ui_name="Undo Settings:Alarm\nLimits:Alarms Setup" type="button" />
        <node name="ALRMLMT_CLOSE" ui_name="Close:Alarm\nLimits:Alarms Setup" type="close" />
        <node name="ALRMLMT_NEXT_TAB" ui_name="Next:Alarm\nLimits:Alarms Setup" type="button" />
    </node> <!-- Alarm setup => Alarm limits close -->
    <!-- Alarm setup => Alarm Priorities -->
    <node name="ALARMS_ALRMPRIO" ui_name="Tab_Alarm\nPriorities:Alarms Setup" type="tab">
    ...
```

Kuva 14. Kuvakaappaus työn kohteena olevan potilasmonitorin käyttöliittymästä puurakenteena xml-formaatissa.

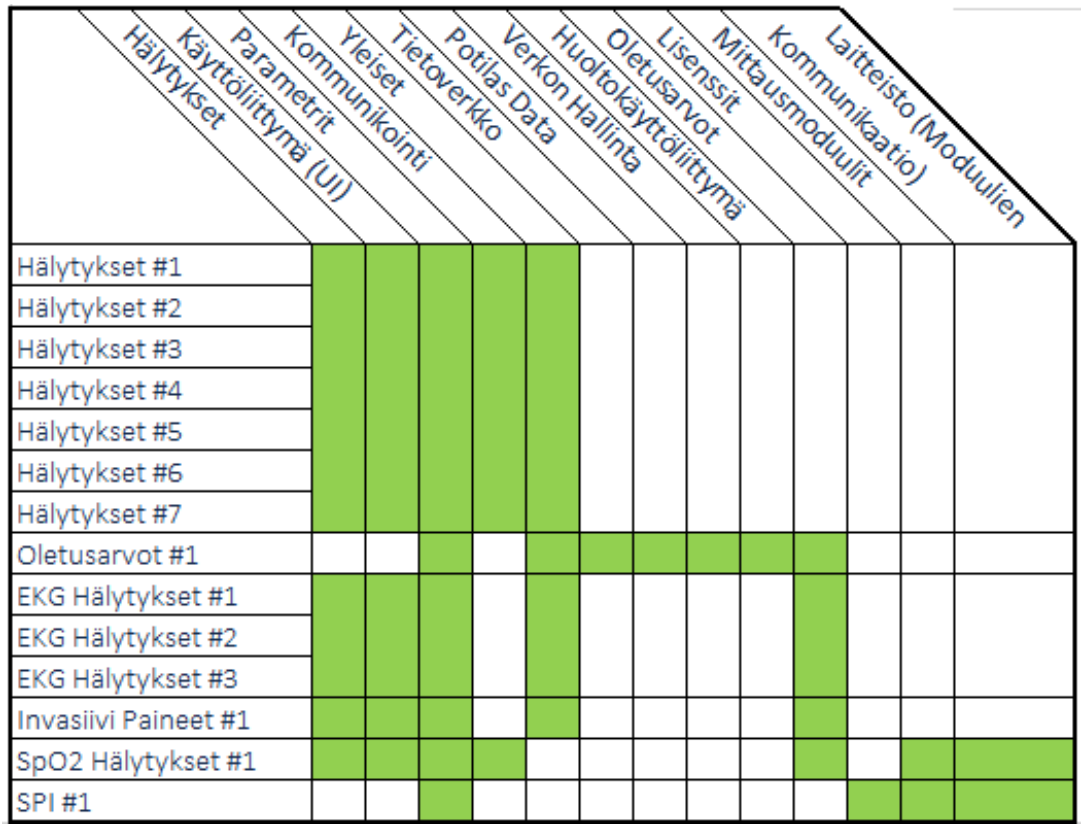
Luotua xml-tiedostoa käytetään vasta, kun potilasmonitorille luodaan tuki testiautomaatiota varten. Sen teko oli kuitenkin tehtävissä projektin aikana, koska xml-tiedostossa käytettyjen attribuuttien arvot sekä potilasmonitorin valikkorakenne on etukäteen nähtävissä ja kirjoitettavissa xml-tiedostoon.

6.4.4 Alustava testiryhmä ja ohjelmistoarkkitehtuurinen kattavuus

Testitapauksien yhteensopivuuksien tarkastamisen ja testijärjestelmälle tarvittavien muutosten tunnistamisen jälkeen 14 testitiedostoa valittiin ohjelmiston validaation testiryhmäksi. Testitiedostot sisältävät useita eri yksittäisiä testitapauksia, jotka kattavat yhteensä 143 potilasmonitorin vaatimusta. Tälle testiryhmälle tiedostettiin, että muutoksia tarvitaan esimerkiksi navigointiin käytettyjen tunnisteiden vaihtamiseen oikeisiin kohdeprojektille. Lisäksi testitapauksissa käytetyt yksilöivät luokittelut täytyy vaihtaa vastaamaan kohdeohjelmiston vaatimuksia. Nämä eroavaisuudet on kuitenkin erittäin helppo havaita ja muokata, kun potilasmonitorissa on tuki testiautomaatiota varten ja testitapauksia pystytään suorittamaan.

Potilasmonitoriohjelmistosta puuttui joidenkin valikoiden kohdalla tärkeiden elementtien yksilöivät tunnisteet, joita käytetään potilasmonitorin ohjaukseen sekä käyttöliittymästä näkyvien arvojen ja tietojen verifiointia varten. Puuttuvat tunnisteet kirjattiin ylös ohjelmistokehittäjälle lisättäväksi ohjelmistoon samalla, kun automaatiotestituki monitorille toteutetaan. Lisäksi tunnistettu hälytysäänten kirjaus lokitiedostoihin merkattiin ratkaistavaksi ohjelmistokehittäjälle.

Tämä valittu testiryhmä kattaa potilasmonitorin ohjelmistoarkkitehtuurin eri alijärjestelmät ja kriittisimmät toiminnot. Kuvassa 15 on esitelty ohjelmiston validaatioon valitut testitiedostot vasemmalla ja näiden kattamat potilasmonitoriohjelmiston alijärjestelmät ylhäällä.



Kuva 15. Ohjelmiston validaatiossa katetut potilasmonitorin ohjelmistoarkkitehtuurin eri alijärjestelmät.

Testiautomaatiolla suoritettujen testitapausten lisäksi ohjelmistolle haluttiin tehdä manuaalisella verifikaatiolla varmistus kriittisten toimintojen varmistamiseksi, joita testiautomaatiolla ei pystytä suorittamaan. Tämä testiryhmä tullaan suorittamaan testiautomaatiolle verifioitavan testiryhmän yhteydessä ja on osa virallista ohjelmiston validaation testisuunnitelmaa. Tämä testiryhmä sisältää verifikaatiotestauksen seuraaville potilasmonitorin toiminnoille:

- verkkoprotokolla #1
- verkkoprotokolla #2
- tulostaminen tietoliikenneverkon kautta
- konkreettiset hälytysäänet eri hälytysprioriteeteillä
- huoltokäyttöliittymä.

Kyseinen monitoriohjelmisto tukee kahta eri verkkoprotokollaa, joita käytetään monitorien ja potilaskeskusten väliseen viestintään. Verkkoprotokolla #1, jota sekä työn koh-

teena oleva kuin lähteenä käytetty potilasmonitoriohjelmisto tukee, verifioidaan lähdeprojektin testitapauksissa lähettämällä ja potilasmonitorista vastaanottamalla dataa tämän verkkoprotokollan avulla. Tämän lisäksi ohjelmistolle haluttiin lisätä manuaalinen verifikaatio arvojen näkymisestä tätä verkkoprotokollaa käyttävässä potilaskeskuksessa. Manuaaliseen verifikaatioon kirjattiin suoritettavaksi tutkiva testaus, jossa lähetetään potilasmonitorilta vitaaliparametrien aaltokäyriä ja arvoja sekä potilaan tietoja tähän potilaskeskukseen ja verifioidaan näiden oikea välittyminen tällä verkkoprotokollalla.

Vaatimusten ja eri ohjelmistoprojektien toimintojen vertailusta havaittiin työn kohteena olevan potilasmonitoriohjelmiston tukevan myös toista verkkoprotokollaa #2, jota lähdeohjelmisto ei tue. Koska tämän potilaskeskuksen ja verkkoprotokollan toiminnan testaamiseen ei ole olemassa valmiita testitapauksia on tehokkainta suorittaa toimivuuden verifiointi manuaalisesti. Manuaaliseen verifikaatioon kirjattiin suoritettavaksi tutkiva testaus, jossa siirretään potilasdataa kyseiseen potilaskeskukseen ja verifioidaan oikeiden arvojen ja asetusten näkyminen molemmissa.

Työn kohteena olevan ohjelmiston edellinen manuaalinen ohjelmiston validaatio verifioi potilasmonitorista tulostamista kattavasti eri muuttujilla. Potilasdatan tulosteen verifiointi testiautomaatiolla ei ole mahdollista ainakaan nykyisellä laitteistolla ja ohjelmistolla. Koska tulostaminen tietoliikenneverkon yli oli verifioitu aikaisemmassa testisuunnitelmassa, haluttiin myös tälle suorittaa tutkiva testaus manuaalisesti ohjelmiston validaatiossa.

Ohjelmistoon lisättiin toiminnallisuutta hälytysäänten verifioimiseen potilasmonitorin lokitiedostoista. Tämä oli tärkeää myös, koska osa muista testiautomaatiossa hälytyksiä verifioivista testitapauksista hyödynsi tätä ominaisuutta. Hälytysääniä pystyisi verifioimaan testiautomaatiolla lokitiedostojen lisäksi muun muassa vertailemalla, kuinka paljon potilasmonitorin kaiutin kuluttaa virtaa. Manuaalisen verifikaation testiryhmään lisättiin kuitenkin tutkiva testaus, jossa verifikaatioinsinööri aiheuttaa potilasmonitorille hälytyksiä eri prioriteeteilla ja kuuntelee hälytysten aktivoitumista ja hälytysääniä. Näin testiautomaatiossa verifioidun hälytysten kirjautumisen lokitiedostoihin lisäksi verifioidaan oikeiden hälytysäänten kuuluminen käyttäjälle.

Huoltokäyttöliittymälle tehtiin työn aikana *proof-of-concep*t-tyylisiä testejä, millä varmistuttiin, että testitapaukset, jotka käyttävät potilasmonitorin huoltokäyttöliittymää, pystytään sulauttamaan osaksi testiautomaatiota toteutusvaiheessa. Näiden luonti oli välttämätöntä, jotta osa testiautomaatiossa suoritettavista testitapauksista pystytään suorittamaan sekä alustamaan potilasmonitori testiautomaatiolla oikeisiin asetuksiin testitapauksia varten. Huoltokäyttöliittymä on tärkeä osa potilasmonitoria etenkin käyttöönottovaiheessa, ja siksi sen oikeanlainen toiminta on osa potilasmonitorin kriittisimpiä toimintoja. Tämän vuoksi manuaaliseen verifikaatioon lisättiin tutkiva testaus huoltokäyttöliittymän oikean toiminnan varmistamiseksi.

Manuaalisesti suoritettavassa tutkivassa testauksessa määrätään aika, joka testaajan tulee käyttää toiminnon vaatimustenmukaisen toiminnan varmistamiseen. Manuaalisen verifikaation lisääminen testisuunnitelmaan ei haitannut työn tavoitetta automatisoida ohjelmiston validaatio. Suuri osa ohjelmistolle suoritettavasta validaatiosta suoritetaan testiautomaatiolla ja manuaalisen verifikaatiotestauksen määrä on erittäin pieni, tarkoin määritettävissä sekä nopeasti suoritettavissa.

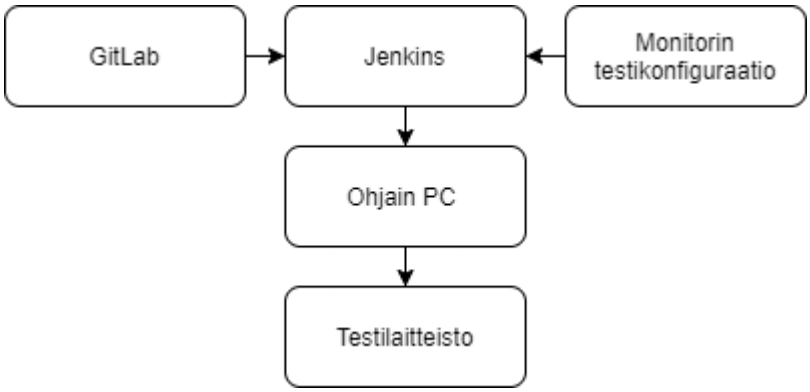
Lopullinen ohjelmiston validaation testiryhmä kattaa testiautomaatiolla verifioitavat 143 vaatimusta sekä viiden eri toiminnon verifioinnin manuaalisesti. Koko testiryhmä pystytään suorittamaan yhdelle potilasmonitorityypille yhdessä päivässä, jolloin ohjelmiston validaatio kestää kolmelle potilasmonitorityypille suorittaessa yhteensä kolme päivää.

6.5 Testiautomaatiojärjestelmän suunnittelu

Kun lähdeprojektin testikokonaisuus oli analysoitu ja kohdeprojektin testiautomaatiolla suoritettava testiryhmä valittu, voitiin aloittaa testiautomaatiojärjestelmän suunnittelu ja vaadittavan laitteiston kartoitus. Järjestelmän suunnittelu aloitettiin konsultoimalla aiheen asiantuntijoita, jossa selvitettiin tarvittavat kriteerit järjestelmälle. Näitä kriteereitä varten pyrittiin rakentamaan järjestelmä tukemaan jatkuvaa integraatiota. Järjestelmä pyrittiin rakentamaan mahdollisimman suoraviivaiseksi. Lisäksi pyrittiin välttämään manuaalista työtä testien suorituksen aikana.

Testijärjestelmän toteutuksessa tarvitaan jatkuvan integraation työkaluja kuten GitLabia ja Jenkinsiä. GitLab on versionhallintatyökalu, mihin kaikki automaatiotestaukseen liittyvät tiedostot, kuten testitapaukset, tallennetaan. Versionhallinnan avulla on helppo tehdä muutoksia testitapauksiin ja tarvittaessa voidaan palata aiempaan toimivaan versioon ongelmien syntyessä. Projektia varten GitLabiin luotiin oma hakemisto, jonne kopioitiin aluksi kaikki lähdeprojektin tiedostot. Tämän jälkeen hakemistoa siistittiin ja kaikki tarpeettomat testitiedostot, kirjastot sekä resurssitiedostot poistettiin. Jenkins on automaatiopalvelin, jonka avulla ohjelmistoa voidaan testata automaattisesti.

Kuva 16 havainnollistaa testijärjestelmän rakennetta jatkuvan integraation mukaisesti.



Kuva 16. Jatkuvan integraation mukainen testiautomaatiojärjestelmä

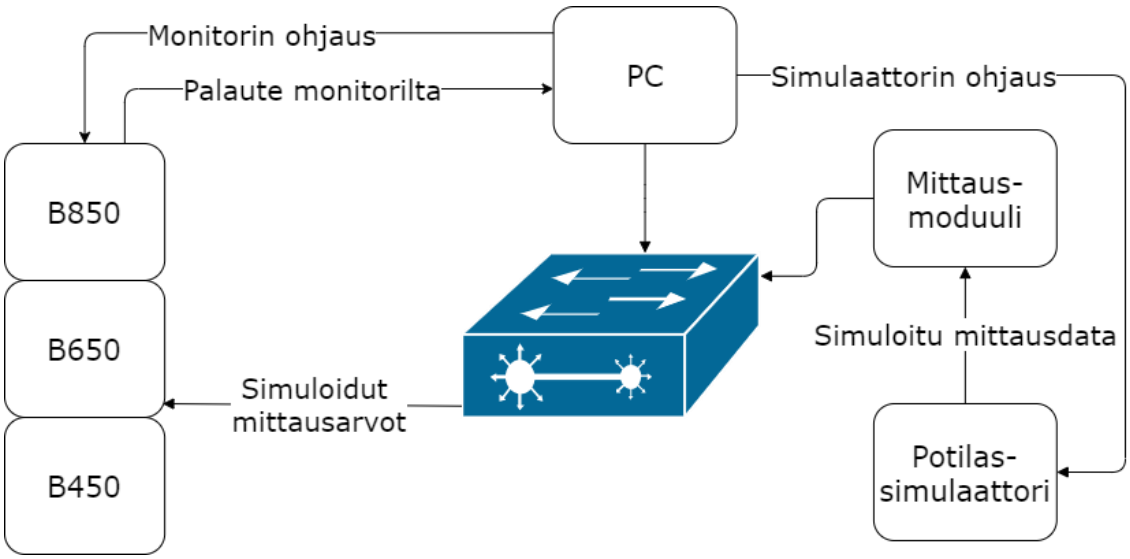
Jenkinsin avulla suoritetaan jokainen testi tai testiryhmä ja nämä haetaan GitLabista. Jenkinsiin määritellään tämän lisäksi jokaisen käytettävän testin konfiguraatio, jotta pystytään ohjaamaan oikeaa testilaitteistoa ja käyttämään tarvittavia muuttujia. Tämä konfiguraatiotiedosto määrittää ohjaintietokoneen käytön, joka suorittaa sille määritetyt testit. Tietokone lähettää kaikki komennot testilaitteistolle ja vastaanottaa testien tulokset, jotka ohjataan takaisin Jenkinsiin. Tuloksista muodostetaan luettava raportti, jossa on tulokset ‘PASS’ tai ‘FAIL’, sekä jokaisen avainsanan ja tunnisteen yksityiskohtaisempi informaatio.

Testijärjestelmän ohjauksen jatkuvan integraation mukaisesti rakentaminen mahdollistaa testien automaattisen toiston, versionhallinnan sekä järjestelmällisesti koostetun raportoinnin. Näistä raporteista voidaan tarkastella yksittäisten testien tai testikokonaisuuksien tuloksia ja testin suoritukseen kulunutta aikaa.

6.5.1 Alustavan testiautomaatioympäristön kartoittaminen

Testiautomaatioympäristön suunnitelman pohjana toimi lähdeprojektin testiautomaatioympäristö. Tavoitteena oli korvata yhteensopimattomat laitteet ja simulaattorit kohdeohjelmistolle yhteensopivilla. Tavoitteena oli rakentaa ympäristö vastaamaan lähdeprojektin testiautomaatioympäristöä, tietyin poikkeuksin. Aiemmissa analyyseissä oli tiedostettu ennakkoon, että osa toiminnallisuudesta joudutaan korvaamaan, kuten tietyt ohjelmistosimulaattorit. Järjestelmää suunniteltaessa konsultoitiin automaatiosta vastaavaa insinööriä. Konsultaation avulla saatiin muodostettua selvä käsitys järjestelmän rakenteesta sekä vaadittavasta laitteistosta. Testilaitteisto tulisi koostumaan kaikista kolmesta eri monitorityypistä, mittausmoduulista, potilasdatasimulaattorista, hallittavasta kytkimestä sekä tietokoneesta, joilla ohjataan kaikkia näitä. Kaikki nämä laitteet kytketään hallittavaan kytkimeen, jolloin eri kytkentäportteja voidaan hallita ja siten määrittää yksitellen, mitä monitoria testataan sekä mitä mittausmoduuleita ja simulaattoreita käytetään milloinkin.

Kuvassa 17 esitetään hallittavan kytkimen toimintaperiaate. Tietokonetta käytetään hallitun kytkimen porttien ohjaamiseen, jolloin voidaan määrittää erikseen, minkä portin kautta tietoa käsitellään.



Kuva 17. Hallittavan kytkimen käyttö.

Kaikki testilaitteet paitsi potilassimulaattori on kytketty kytkimeen. Tämä mahdollistaa sen, että testilaitteistoa ei tarvitse vaihtaa monitorityyppien välillä vaan kytkin ohjaa mittausmoduulilta saatavaa dataa testin alaiselle potilasmonitorille. Kuvassa vasemmalla puolella on kuvattu käytettävät eri monitorityypit: B850, B650 ja B450, jotka kaikki on kytketty kytkimeen. Käytettävä monitori palauttaa tietoliikenneverkon kautta tietokoneelle vaadittavia tietoja esimerkiksi lokitiedostojen muodossa, joilla monitorin toiminta verifioidaan. Kytkimen avulla on siis mahdollista määrittää eri kombinaatiot, millä verifikaatiota suoritetaan. Normaalisti näiden mittausmoduulien vaihdot jouduttaisiin tekemään manuaalisesti, mutta hallittavan kytkimen avulla se on mahdollista automatisoida, jolloin säästytään manuaaliselta työltä. PC:llä ohjataan myös potilassimulaattoria, jolla simuloidaan erilaisia fysiologisia tiloja. Simulaattoria ohjataan PC:llä ja käytettävät mittauskaapelit on kytketty mittausmoduuliin. Tällä tavalla voidaan hallita yksitellen, mitä monitoria käytetään sekä milloin mittausmoduuli on kytkettynä osaksi järjestelmää.

Alustavassa suunnitelmassa pyrittiin testaamaan kaikki samat alijärjestelmät kuten lähdeprojektissa. Tämän osalta haasteelliseksi osoittautui kuitenkin lisenssinalainen SPI-parametri. Kyseisen parametrin mittaus tapahtuu näiden ohjelmistoversioiden välillä eri tavalla ja niissä käytetään eri mittausmoduuleja. Tämän parametrin mittaamiseksi olisi tarvittu mittausmoduuli, jonka kiinnittäminen osaksi testijärjestelmää suoraan ei ollut mahdollista yhteensopimattoman liittimen vuoksi.

Ongelmaa yritettiin ratkaista käyttämällä ulkoista moduulikehystä, johon tämä moduuli pystytään kytkemään. Moduulikehys on ulkoinen laite, jota käytetään B850-monitorityypin kanssa mittausmoduulien kiinnittämiseen. Kahdella muulla monitorityypillä tämä kehys on integroitu. Näillä kahdella monitorityypillä, joissa on integroitu kehys, ongelmaksi muodostui riittävän virran saanti. Selvityksen jälkeen tuli ilmi, ettei tätä kehystä tueta, koska kyseiset monitorityypit eivät pysty syöttämään kehykselle tarpeeksi virtaa. Tämän lisäksi testeissä ilmeni, että vaikka moduulikehyksen kanssa käytettiin lisävirtaa, toiminta oli epävakaata ja mittaukset katkeilivat kyseiseltä moduulilta. Tämän johtopäätös oli se, että kyseistä moduulia ei voida liittää hallittuun kytkimeen, jolloin testien suorittaminen tällä moduulilla vaatisi enemmän manuaalista työtä testien välissä moduulien vaihdon takia. Koska tavoitteena oli automatisoida tämä prosessi mahdollisimman laajalti, päädyttiin etsimään toista ratkaisua. Tästä johtuen alustavaan suunnitelmaan jouduttiin tekemään muutoksia, jotka on kuvattu luvuissa 6.5.2 ja 6.5.3

6.5.2 Kartoittamisessa esiintyneiden ongelmien ratkaisu

Koska alkuperäisen suunnitelman mukaista testiautomaatiojärjestelmää ei ollut järkevää toteuttaa, jouduttiin pohtimaan toista ratkaisua. Ongelmaa lähestyttiin pohtien, miten SPI-parametria käyttävä testitapaus voitaisiin korvata. Aluksi analysoitiin, minkä ohjelmiston alijärjestelmän tämä parametri kattaa. Kyseisen testitapauksen tarkoituksena oli tarkistaa lisenssinalaisen parametrin toiminta, ja siksi kyseinen testitapaus oli valittu osaksi testiryhmää. Arvioinnin ja konsultointien jälkeen tultiin lopputulokseen, jossa tämä testitapaus korvataan toisella lisenssin alaisella parametrilla. Johdonmukaiseksi valinnaksi osoittautui lisenssi, joka mahdollistaa vastasyntyneiden intensiivihoidon ohjelmistopaketilla suoritettavan pulssioksimetrian monitoroinnin erillisellä mittausmoduulilla. SPI-testitapaus päädyttiin korvaamaan tällä lisenssiin alaisella parametrillä, sillä se on vastaavuudeltaan identtinen ohjelmiston arkkitehtuurin alijärjestelmän kannalta. Vastaavaa testitapausta ei löytynyt lähdeprojektista, joten tämä joudutaan tekemään myöhemmin testiautomaation toteutusvaiheessa.

Varmistukseksi muutoksen laajuudesta sekä toteutuksen järkevästä työmäärästä tehtiin siitä proof-of-concept-tyylinen varmistus kuten aiemmin. Tämä toteutettiin laajentamalla aiemmin luotuja huoltokäyttöliittymän testejä. Myös aiemmin luotua resurssitiedostoa laajennettiin sekä muuttujien että avainsanojen osalta. Tarvittava toiminnallisuus sisälsi edellä mainitun lisenssin poiston ja aktivoimisen. Esimerkkikoodissa 6 on toteutettu vaadittavan lisenssin poisto ja lisäys.

```
*** Settings ***
Documentation    Verifies that E-SpO2 Module Neonatal license can
...              be enabled

Resource          ../resources/resource.robot
Suite Setup       Activate NICU Package
Suite Teardown    Activate ICU Package
Test Setup        Open ServiceInterface
Test Teardown     Close Browser

*** Test Cases ***
Remove E-SpO2 Module Neonatal license
    Navigate To Host Licenses
    Remove License    ${PSPO}
    Restart Monitor
    Login To ServiceInterface
    Navigate To Host Licenses
    List Selection Should Be    ${PSPO}    DISABLED

Enable E-SpO2 Module Neonatal license
    Navigate To Host Licenses
    Activate License   ${PSPO}    @{ACTIVATE_PSPO}
```

```
    Restart Monitor
    Login To ServiceInterface
    Navigate To Host Licenses
    List Selection Should Be    ${PSPO}    ENABLED


*** Keywords ***
Activate NICU Package
    Open Login Page
    Login To ServiceInterface
    Navigate To Sw Packages
    Select Sw Package    ${NICU_PACKAGE}
    Restart Monitor
    Close Browser

Activate ICU Package
    Open Login Page
    Login To ServiceInterface
    Navigate To Sw Packages
    Select Sw Package    ${ICU_PACKAGE}
    Restart Monitor
    Close Browser

Open ServiceInterface
    Open Login Page
    Login To ServiceInterface
```

Esimerkkikoodi 6. Esimerkkikoodi lisenssin hallintatestistä

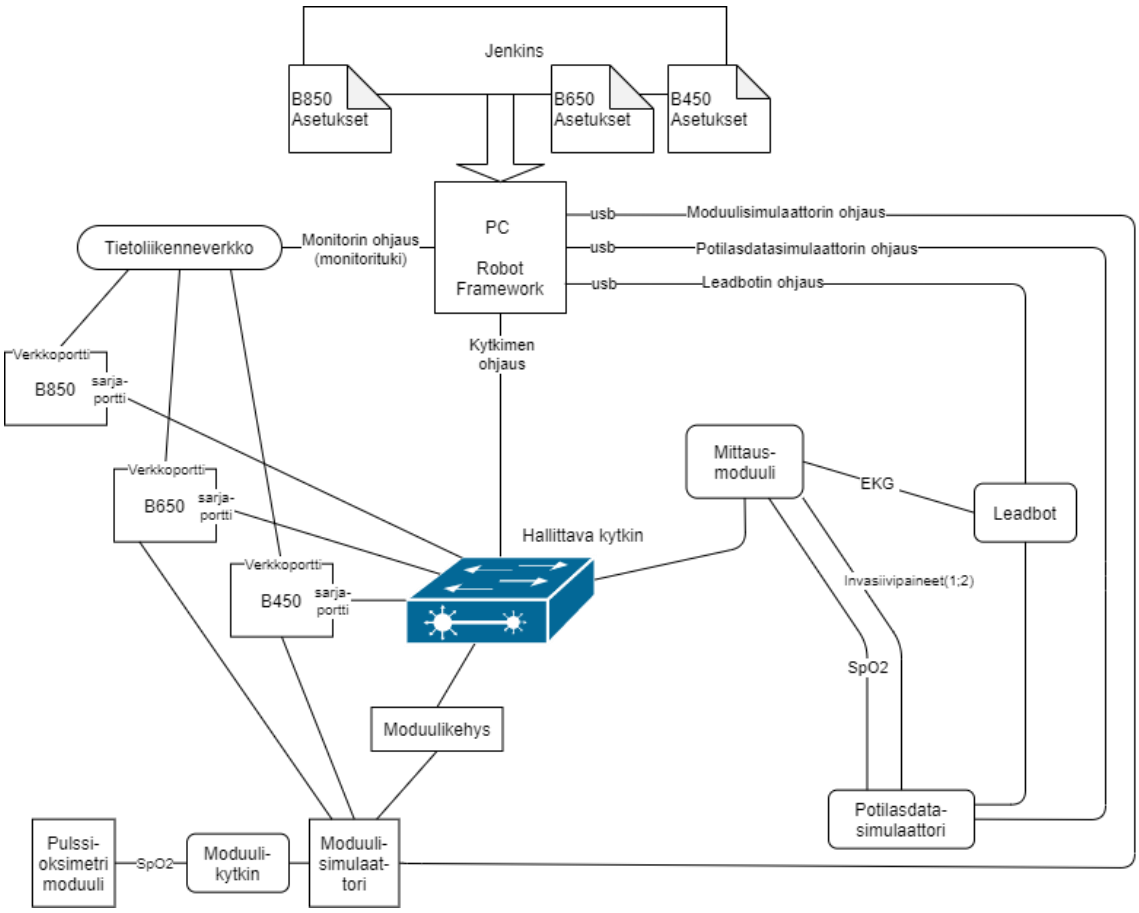
Testissä on käytetty alustuksena aiemmasta esimerkistä poiketen myös *Suite Setup*- ja *Suite Teardown* -komentoja. *Suite Setup* -komento suoritetaan ennen varsinaisia testejä ja *Suite Teardown* -komento suoritetaan taas viimeisen testitapauksen jälkeen. Esimerkkikoodin testejä varten *Suite Setup* -komento aktivoi monitorin ‘vastasyntyneiden teho-osasto' -ohjelmistopaketin. Tämä vaaditaan, jotta käytettävää lisenssiä voidaan testata, sillä kyseinen lisenssi mahdollistaa pulssioksimetrian käytön kyseisellä ohjelmistopaketilla erilliseltä mittausmoduulilta. Kun vaadittu ohjelmistopaketti on valittu, monitori käynnistetään uudelleen sen aktivoimiseksi. Ohjelmistopaketin aktivoinnin jälkeen kirjaudutaan huoltokäyttöliittymään, jossa lisenssi poistetaan käytöstä. Tämän jälkeen monitori käynnistetään uudelleen muutosten aktivoimiseksi, jonka jälkeen tarkistetaan vielä, että lisenssin tila on *‘Disabled’* eli pois käytöstä. Seuraava testi tekee saman toiseen suuntaan, eli aktivoi lisenssin ja syöttää sille vaadittavan aktivointikoodin. Uudelleenkäynnistyksen jälkeen jälleen tarkistetaan, että lisenssin tila on ‘*Enabled*’ eli aktivoitu. Näiden testien päätteeksi suoritetaan *Suite Teardown* -komento, jossa aktivoidaan oletuksena käytettävä ‘teho-osasto’-ohjelmistopaketti edellisen tilalle.

Testien tarkoituksena ei ollut luoda varsinaista testitapausta itse lisenssin alaiselle parametrille vaan tavoitteena oli saada varmistus siitä, että lisenssien hallitsemiseen tarvittava toiminnallisuus pystyttiin toteuttamaan huoltokäyttöliittymälle. Tällä testillä pystyttiin varmistumaan siitä, että huoltokäyttöliittymän vaadittava ohjaus ja hallinta on mahdollista toteuttaa käyttäen aiemmin luotuja testejä sekä resurssitiedostoja. Huoltokäyttöliittymän hallintaan liittyvien testien toiminnallisuutta voidaan hyödyntää myöhemmin toteutusvaiheessa ja sulauttaa osaksi muuta testiryhmää.

6.5.3 Testiautomaatiojärjestelmän suunnitelma

Kun kartoituksen yhteydessä esiintyneet haasteet oli ratkaistu, hahmoteltiin lopullinen rakenne testiautomaatioympäristölle. SPI-parametria verifioivan testin korvaamisen jälkeen oli mahdollista toteuttaa testiympäristö tavoitteiden mukaisesti. Lopullinen rakenne testiautomaatioympäristölle mahdollistaa jatkuvan testaamisen ilman suurta manuaalista työtä testien suorituksien välissä, sekä sen, että vaadittavan laitteiston määrä on optimoitu. Tämä tarkoittaa sitä, että simulaattoreita ja mittausmoduuleita voidaan hyödyntää kaikille eri monitorityypeille, eikä jokaiselle tarvita erikseen omaa. Tämä on erittäin huomionarvoinen seikka, sillä tämän avulla voidaan säästää huomattava summa testiympäristön rahallisissa laitteiston kustannuksissa.

Kuvassa 18 on kuvattu kaaviona testiautomaatioympäristön lopullinen rakenne. Jokainen testi suoritetaan Jenkinsistä käsin, johon on määritelty jokaisen monitorin konfiguraatiota vastaavat asetukset (kuvassa B850, B650 ja B450 asetukset). Näillä asetuksilla määritetään, mitä monitoria ohjataan tietokoneen avulla Robot Frameworkilla. Monitorin ohjauksella tarkoitetaan myöhemmässä vaiheessa toteutettavaa tukea, joka vaaditaan testien suorittamiseen.



Kuva 18. Testiautomaatiojärjestelmän rakenne.

Kuvassa tietoliikenneverkon kautta ohjataan monitoria, kun taas sarjaportti vastaanottaa simuloitua potilasdataa, joka tulee hallitun kytkimen kautta. Tietokoneella (kuvassa PC) hallitaan myös, mistä hallitun kytkimen portista tietoa vastaanotetaan milloinkin. Tällä tavoin voidaan kohdentaa, mikä monitori vastaanottaa mittauksia ja siten samalla myös simuloida tilannetta, jossa mittausmoduuli olisi irrotettu. Hallitun kytkimen käyttö ja hallinta mahdollistaa kaikkien siihen kytkettyjen simulaattorien käytön erikseen jokaiselle testattavalle monitorille. Tietokoneella ohjataan myös potilasdata-, moduuli- ja leadbot-simulaattoria. Potilasdatasimulaattorilla voidaan simuloida eri parametreja ja niiden fysiologisia tiloja. Moduulisimulaattoria käytetään moduulikytkimen ja pulssioksimetriamittausmoduulin kanssa simuloimaan sen erilaisia teknisiä tiloja, kuten mahdollista vikatilaa tai muuta teknistä tilaa. Leadbot-simulaattorilla voidaan simuloida erilaisia EKG-kytkentöjen tiloja, kuten esimerkiksi jonkin tietyn mittauselektrodin irtikytkentää. Näiden simulaattorien käyttö mahdollistaa valitun testiryhmän suorittamisen ilman manuaalista työtä testien välillä. B850-tyypin monitori vaatii myös oman ulkoisen moduulikehyksen,

kun taas kahdessa muussa tyypissä tämä on integroitu. Näihin kehyksiin kytketään moduulisimulaattori käytettävän moduulikytkimen ja pulssioksimetriamittausmoduulin kanssa.

Testiautomaatioympäristön suunnitellussa pyrittiin minimoimaan manuaalisen työn tarve testien välillä. Kuitenkaan tiettyjä rajoituksia ei pystytty ratkaisemaan, koska käytettävien eri monitorimallien rakenne eroaa hieman toisistaan. Samalla haluttiin minimoida käytettävän laitteiston määrä kustannussyistä. Testilaitteiston määrä vaikuttaa suoraan testiautomaatioympäristön kustannuksiin ja tässä pyrittiin löytämään optimaalinen ratkaisu kustannustehokkuuden kannalta. Näistä syistä johtuen testiautomaatioympäristöstä yksi moduulisimulaattori joudutaan manuaalisesti vaihtamaan eri monitorityyppien testien välillä. Kyseinen työmäärä on kuitenkin kohtuullinen eikä siten vie paljon aikaa ja ohjelmiston validaation testiryhmä pystytään suorittamaan yhdelle monitorityypille ilman manuaalista työtä testitapausten välillä.

6.6 Loppuraportin laatiminen

Kaikki tässä raportissa esitellyt löydökset koostettiin yrityksen sisäisessä viestinnässä käytettävälle sivustolle loppuraportiksi. Näitä tietoja hyödynnetään siinä vaiheessa, kun varsinaista automaatiotestausta voidaan alkaa toteuttamaan vaadittavan monitorituen valmistuttua. Kyseisessä raportissa läpikäydään yksityiskohtaisesti läpi kaikki keskeiset huomioon otettavat seikat. Tämänkaltaisen raportin luonti itse insinöörityön lisäksi on tarpeellista myös siksi, että osa aineistosta on hyvin yksityiskohtaista, jota ei siksi voida tässä raportissa kuvata riittävän tarkasti sen hyödyntämiseksi testiautomaation toteutuksessa. Yrityksen sisäisen raportin tarkoituksena on saada selkeä käsitys testiautomaation toteuttajalle siitä, mitkä keskeiset seikat tulee huomioida testiautomaation implementoinnissa.

7 Yhteenveto

Insinöörityön tavoitteena oli luoda suunnitelma potilasmonitorin ohjelmiston validaatiotestauksen automatisoinnista käyttäen Robot Framework -testiautomaatiokehystä.

Suunnitelman luontia varten hyödynnettiin GE Healthcarella toteutetun toisen ohjelmistoprojektin testiautomaatiojärjestelmää lähteenä. Lähdeprojektissa testiautomaatiota käytetään laajasti ohjelmiston verifikaatioon. Lähde- ja kohdeohjelmistoprojektien erojen vuoksi testijärjestelmää ei pystytä suoraan hyödyntämään, vaan kaikkien testijärjestelmän eri osa-alueiden sopivuus ja tarvittava työmäärä muokkauksiin täytyi arvioida.

Suunnitelman luonti aloitettiin koostamalla ohjelmisto- ja järjestelmävaatimukset kummankin ohjelmiston validaation testilaajuudesta. Vaatimusten koostaminen ja niiden vertailu olivat keskeisessä osassa testilaajuuden määrittelyssä. Näiden avulla saatiin selkeä kuva ohjelmistojen yhtäläisyyksistä ja eroavaisuuksista. Molempien potilasmonitoriohjelmistojen ohjelmiston validaatiota koskevat vaatimukset analysoitiin ja dokumentoitiin. Analyyseissä keskityttiin havaitsemaan keskeiset erot ohjelmistojen välillä, joilla on suora vaikutus testien kattavuuteen. Vaatimusten vertailun tuloksien perusteella oli mahdollista tunnistaa kohdeohjelmistolle sopimattomat testitapaukset.

Kun kaikki nämä eroavaisuudet oli tunnistettu ja dokumentoitu, voitiin siirtyä kohdeohjelmiston testiryhmän tarkasteluun. Testiryhmän jokainen testitapaus analysoitiin tarkasti näiden keskeisten erojen löytämiseksi. Jokainen eroavaisuus ja sille vaadittava muutos dokumentoitiin suunnitelmaan, jotta nämä pystytään muuttamaan suunnitelman pohjalta tehtävässä toteutuksessa. Analyysin aikana havaittiin, että osa muutoksista voidaan toteuttaa jo projektin aikana. Nämä muutokset tehtiin projektin aikana, jotta voidaan säästää työtaakkaa myöhemmässä vaiheessa. Kun testiryhmään kuuluvat testitapaukset oli analysoitu, aloitettiin testiautomaatioympäristön suunnittelu. Testiautomaatioympäristön suunnittelussa keskeisiä tavoitteita oli laitteiston määrän optimointi, manuaalisen työn minimointi ja jatkuvan integraation mahdollistavien työkalujen käyttö.

Työn tuloksena saatiin erittäin kustannustehokas suunnitelma siitä, miten työn kohteena olevan potilasmonitorin ohjelmiston validaatio voidaan toteuttaa testiautomaatiolla, jossa käytetään järkevästi hyödyksi lähdeprojektin testijärjestelmää sekä testitiedostoja. Suunnitelmassa on listattu ohjelmiston validaatiossa testiautomaatiolla verifioitavat 14 eri testitapausta, jotka kattavat ohjelmiston kriittisimmät toiminnot eri alijärjestelmistä. Testiautomaatiolla suoritettavien testitapausten lisäksi ohjelmiston validaatiossa tullaan suorittamaan viiden eri ohjelmiston ominaisuuden verifikaatio myös manuaalisesti. Tämä testiryhmä sisältää testitapauksia, joita testiautomaatiolla ei pystytä tai ei ollut järkevää tä-

män projektin aikana lähteä toteuttamaan. Manuaalisen verifikaation lisääminen ohjelmiston validaation testiryhmään ei haitannut työn tavoitetta. Lähes kaikki ohjelmiston validaation testiryhmästä suoritetaan testiautomaatiolla ja manuaalisen verifikaation määrä on pieni, tarkoin määritetty ja nopeasti suoritettavissa esimerkiksi samalla, kun testiautomaatiota suoritetaan taustalla toisessa testiympäristössä.

Testisuunnitelma sisältää suoritettavan testiryhmän lisäksi ohjelmistoon sekä testitapauksiin tarvittavat muutokset sekä kuvauksen testiympäristöstä, jonka avulla tämä testiryhmä pystytään testaamaan yhdelle monitorityypille ilman manuaalista testilaitteiston vaihtoa testitapausten välissä. Suunnitelma sisältää kaikki työn aikana tunnistetut muokkaustarpeet, mutta kattavan analyysin lisäksi toteutusvaiheessa voi nousta ongelmia, joita ei pystytty tunnistamaan pelkästään testijärjestelmää tutkiessa. Nämä ongelmat ovat jäljitettävissä Robot Frameworkin raporteista ajamalla testitapaukset sen jälkeen, kun potilasmonitorille on luotu tuki testiautomaatiota varten. Suunnitelmaan luotiin ohjeet testiautomaation toteuttajalle esimerkiksi ohjelmiston ohjaukseen liittyvien ongelmien ratkaisemiseksi.

Testiautomaation avulla ohjelmiston validaatio pystytään suorittamaan huomattavasti nopeammin kuin manuaalisella verifikaatiolla. Alustavan laskennan avulla mahdollinen ajallinen säästö työmäärässä on kymmenen suorituskerran aikana 370 työpäivää. Ohjelmiston validaation testiryhmä kattaa edelleen kaikki potilasmonitoriohjelmiston kriittisimmät toiminnot. Ohjelmiston validaation testiryhmälle suunniteltu testiautomaatioympäristö mahdollistaa potilasmonitorin testaamisen jatkuvasti kehityksen aikana jokaiselle versiolle. Tämä tuo suurta arvoa kehitystyöhön, kun mahdolliset ongelmat tämän testiryhmän suorittamiseen pystytään havaitsemaan välittömästi. Myöskään testitapauksien suorittaminen ei vaadi jatkuvaa seurantaa, joten verifikaatioinsinööri vapautuu näitä suorittaessa muihin työtehtäviin, esimerkiksi suorittamaan manuaalisen verifikaation testit.

Lopullinen ohjelmiston validaation testiryhmä kattaa testiautomaatiolla verifioitavat 143 vaatimusta sekä viiden eri toiminnon verifioinnin manuaalisesti. Koko testiryhmä eli testiautomaatiolla suoritettavat testitapaukset sekä manuaalisesti suoritettava tutkiva testaus pystytään suorittamaan yhdelle potilasmonitorityypille yhdessä päivässä, jolloin ohjelmiston validaatio kestää kolmelle potilasmonitorityypille suorittaessa yhteensä kolme päivää.

Insinöörityö aloitettiin tammikuussa 2021, ja työ valmistui huhtikuussa 2021. Kokonaiskesto oli reilu kolme kuukautta täyspäiväistä työtä. Projektin läpiviemiseen vaadittiin työn tekijöiltä laajaa ymmärrystä Robot Frameworkista, testiautomaatiosta sekä testiautomaatioympäristön suunnittelusta. Tutkimalla ensin testikokonaisuuden pystyttiin luomaan pätevä suunnitelma, jota käyttämällä lopputulos saavutetaan kustannustehokkaasti. Ilman suunnitelman tekoa toteutusvaihe kestäisi huomattavasti pidempään, koska tunnistettuja ongelmia tulisi hiljalleen toteutuksen eri vaiheissa. Nyt iso osa testiautomaatiokokonaisuuden siirtoon liittyvistä ongelmista on tunnistettu valmiiksi ja yksittäisten ongelmien ratkaisuun voidaan suoraan siirtyä toteutusvaiheessa. Myös tarkka suunnitelma testiautomaatioympäristössä käytetyistä laitteista helpottaa tarvittaessa muokkaamaan testitapauksia hyödyntämään tätä laitteistoa.

Työn lopputuloksena oli raportti yrityksen sisäisessä viestinnässä käytettävälle sivustolle. Tässä erillisessä raportissa on kuvattu yksityiskohtaisesti kaikki havaitut ja tiedostetut muutostarpeet. Raportin tarkoitus on myös antaa käsitys yritykselle siitä, kuinka paljon työtä ja muutoksia suunnitelman mukaisen testiautomaation toteuttaminen vaatii kohteena olevalle potilasmonitorinohjelmistolle. Tätä suunnitelmaa tullaan hyödyntämään sen jälkeen, kun potilasmonitorinohjelmiston ohjaksen tuki testiautomaatiota varten on toteutettu.

Lähteet

1 Yrityksen tiedot. Verkkoainesto. GE Healthcare Finland Oy <https://www.ge-healthcare.fi/about/about-ge-healthcare-systems>. Luettu: 2.3.2021.

2 Rüdiger, Kramme; Klaus-Peter, Hoffmann; Robert S, Pozos. 2011. Springer Handbook of Medical Technology. E-Kirja. Springer-Verlag Berlin Heidelberg.

3 GE Healthcare Carescape B650-potilasmonitori. Kuva. <https://www.ge-healthcare.com.sg/products/patient-monitoring/carescape-monitor-b650>. Avattu 14.3.2021.

4 Tool Validation Requirements for Medical Device Development. Verkkoaineisto. Medical Systems Consult. <http://www.medicalsysconsult.com/blog/tool-validation-requirements-for-medical-device-development.html>. Luettu 2.4.2021.

5 What is CI/CD? Continuous Integration & Continuous Delivery. Verkkoaineisto. Guru99. <https://www.guru99.com/continuous-integration.html>. Luettu 6.3.2021.

6 Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Continuous_integration>. Luettu 6.3.2021.

7 Robot Framework. Verkkosivu. <https://robotframework.org/>. Luettu 6.3.2021.

8 Robot Framework User Guide Version 3.2.2. Verkkodokumentti. <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>. Luettu 6.3.2021.

9 Ville, Kari. 2011. Robot Frameworkin validointi. Insinöörityö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta. <https://www.theseus.fi/bitstream/handle/10024/30970/insinoorityo_ville_kari.pdf?sequence=1&isAllowed=y>. Luettu 6.3.2021.

10 Product Classification. Verkkoaineisto. U.S. Food & Drug Administration. <https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfPCD/classification.cfm?id=891>. Päivitetty 29.3.2021. Luettu 30.3.2021.

11 CFR - Code of Federal Regulations Title 21. Verkkoaineisto. U.S. Food & Drug Administration. <https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=820&showFR=1&subpartNode=21:8.0.1.1.12.1> Päivitetty 10.11.2020. Luettu 30.3.2021.

12 Design Control Guidance For Medical Device Manufacturers. Verkkoaineisto. U.S. Food & Drug Administration. <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/design-control-guidance-medical-device-manufacturers>. Päivitetty 10.9.2018. Luettu 30.3.2021.

13 Speer, Jon. 2018. The Ultimate Guide To Design Controls For Medical Device Companies. Verkkoaineisto. Greenlight guru. <https://www.greenlight.guru/blog/design-controls#design-reviews>. 07.05.2018. Luettu 30.3.2021.

14 Speer, Jon. 2015. 2 Things You Need To Do Asap When Starting A Medical Device Project. Verkkoaineisto. Greenlight guru. <https://www.greenlight.guru/blog/starting-a-medical-device-project>. Päivitetty 17.02.2015. Luettu 30.3.2021.

15 Tartal, Joseph. Design Controls. Verkkoaineisto. U.S. Food & Drug Administration. <https://www.fda.gov/media/116762/download>. Luettu 30.3.2021.

16 Johner, Christian. Design Changes: Examples and Requirements. Verkkoaineisto. Johner Institute. <https://www.johner-institute.com/articles/regulatory-affairs/and-more/design-change/>. Luettu 1.4.2021.

17 SeleniumLibrary. Verkkodokumentti. <https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html>. Luettu 6.4.2021.

18 Yli-Hankala, Arvi. 2009. Uusi anestesian mittari: Surgical Pleth Index. Verkkoaineisto. Duodecim. <https://www.duodecimlehti.fi/duo98224>. Luettu 12.4.2021.