SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

TESTAUSKÄYTÄNTÖJEN RAKENTAMINEN JA SCADA TOTEUTUKSIEN SIMULOINTI

TEKIJÄ: Antti Pykäläinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Sähkötekniikan koulutusohjelma	
Työn tekijä Antti Pykäläinen	
Työn nimi Testauskäytäntöjen rakentaminen ja SCADA toteutuksien simulointi	
Päiväys 27.5.2020	Sivumäärä/Liitteet 40
Ohjaajat Sami Tiilikainen, tuntiopettaja ja Pasi Lepistö, lehtori	
Toimeksiantaja/Yhteistyökumppani KPA Unicon Oy, Joni Räsänen, tuotepäällikkö	

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää yrityksen PlantSys SCADA-järjestelmän testauskäytäntöjä. Testauksen tueksi oli määrä kehittää testausdokumentti, joka luo selkeän rakenteen testausprosessille ja helpottaa samalla testaajan työtä. Lisäksi testaamisen tueksi oli määrä tutkia mahdollisuuksia automatisoida testausprosessia osittain.

Opinnäytetyössä perehdyttiin aluksi SCADA-järjestelmien, testaamisen ja simuloinnin teoriaan yleisesti. Tällöin voitiin paremmin punnita simuloinnin hyötyjä ja haittoja osana testausprosessia. Testauksen teorian, erityisesti ohjelmistotestauksen teorian pohjalta kyettiin suunnittelemaan testausdokumentaation rakenne ja sisältö mahdollisimman tehokkaaksi. SCADA-järjestelmien teoriaa puolestaan hyödynnettiin rajaamaan ohjelmistotestauksen metodeista käytettäviksi ne, jotka ovat hyödyksi SCADA valvomo-ohjelmistojen testauksessa.

Opinnäytetyön tuloksena syntyi yhtenäinen testausdokumentti, joka yhdistää useita ohjelmistotestauksessa käytettyjä dokumentteja yhdeksi kokonaisuudeksi. Tämän testausdokumentin osat ovat: yleistiedot, testaussuunnitelma, testausaikataulu, testausyhteenveto sekä testauksen tulokset. Työn aikana jalostettiin myös testauskäytänteitä kehittämällä automatisoituja testausmenetelmiä. Simulointia ei käytännön testauksessa opinnäytetyön aikana hyödynnetty, sillä automaatiojärjestelmän ja laitosprosessin simulointiin tarvittavat resurssit olisivat suuremmat, kuin simuloinnista saatava hyöty.

Avainsanat
valvomo, testausdokumentaatio, simulointi, automaation ohjausjärjestelmä

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electrical Engineering			
Author Antti Pykäläinen			
Title of Thesis Building of Testing Practices and Simulation of SCADA Implementations			
Date	27 May 2020	Pages/Appendices	40
Supervisors Mr. Sami Tiilikainen, Lecturer and Mr. Pasi Lepistö, Senior Lecturer			
Client Organisation /Partners KPA Unicon Oy, Mr. Joni Räsänen, Product Manager			

Abstract

The goal of the thesis was to develop better testing practices for the PlantSys SCADA system development of the company. A testing document was to be created to aid in the testing process. In addition, ways to automatise some parts of the testing procedure were to be studied.

In the beginning of the thesis, the theory of SCADA systems, testing and simulation were familiarized with. The theory was utilized to weigh the positives and negatives of using simulation as a part of the testing process. Also, the theory of testing, especially of the software testing helped with the efficiency of the testing document creation. Theory of SCADA-systems helped to better prioritize the most fitting software testing practices to be used in the SCADA control room software testing.

As a result of the thesis, the testing documentation was created. The document combines multiple documents used in software testing into one compact document. The document includes the following elements: general information, a test plan, a test schedule, a testing summary, and the test results. During the thesis process, testing practises were also developed by introducing new automated testing procedures. Simulation was not applied in the practical testing as it was not cost efficient to simulate both the automation system and the plant process for testing purposes.

Keywords
control room, testing documentation, simulation, automation control system

SISÄLTÖ

SANASTO JA LYHENTEET

Binääriluku	Binäärijärjestelmä on kantalukujärjestelmä, jossa kaikki numerot esitetään lukujen yksi ja nolla avulla. Automaatiojärjestelmässä käytetään binärilukuja digitaalisten tulojen ja lähtöjen tilatietojen käsittelemisessä.
Data	Data tarkoittaa tietoa tai tietoja. Data sanaa käytetään yleisesti tietotekniikassa tarkoittamaan ohjelmistojen ulos antamia tai säilyttämiä tietoja.
DCS	Distributed Control System, eli hajautettu ohjausjärjestelmä. Hajautettu ohjausjärjestelmä on SCADA-järjestelmän kaltainen automaationohjausjärjestelmä, jonka avulla voidaan ohjata ja valvoa laitosprosesseja. Hajautetun ohjausjärjestelmän erona SCADA-järjestelmään verrattuna on sen paikallisuus. Kaikki hajautetun järjestelmän osat sijaitsevat samassa paikassa, eli ohjattavalla laitoksella, kun taas SCADA-järjestelmä tarjoaa mahdollisuuden myös etäohjaukseen laitoksen ulkopuolelta.
DDC	Direct Digital Control, eli suora digitaaliohjaus. Suoralla digitaaliohjauksella tarkoitetaan järjestelmiä, joissa laitteiden ohjaus tapahtuu suoraan tietokoneen kautta sen sijaan, että tietokoneella ohjattaisiin jotain logiikkaohjainta, kuten ohjelmoitavaa logiikkaa. Suoraa digitaalisäätöä hyödynnetään usein rakennusautomaatiossa esimerkiksi lämmitys- ja ilmastointijärjestelmien ohjaamiseen.
Modeemi	Modeemi on laite, jonka avulla dataa muutetaan muotoon, joka voidaan lähettää esimerkiksi internetin välityksellä. Modeemia käytetään myös lähetetyn tiedon vastaanottamiseen, sillä se muuttaa tulevan tiedon takaisin muotoon, jota esimerkiksi tietokone osaa käsitellä.
OPC	Open Platform Communications, eli avoimen alustan kommunikaatio. OPC on käytännössä ryhmä standardeja teollista tietoliikennettä varten. OPC-standardeja noudattamalla mahdollistetaan erilaisten laitteiden välinen kommunikaatio, kuten ohjelmoitavan logiikan ja tietokoneen välinen kommunikaatio. OPC-standardista on vanhempi OPC Classic versio ja uudistettu OPC UA, eli Unified Architecture versio.
Operaattori	Operaattorilla tarkoitetaan henkilöä, joka valvoo ja tarvittaessa ohjaa tai säätää laitosprosessia laitoksen valvomosta käsin.
Palvelin	Palvelin on laite tai tietokoneohjelma, jota käytetään tukena jonkin muun ohjelman käytössä. SCADA-järjestelmässä esiintyvät palvelimet pohjautuvat asiakas–palvelin malliin, jossa palvelin antaa tietoja ja asiakas vastaanottaa ne. Monesti palvelin voi toimia sekä asiakkaana että palvelimena.

PLC Programmable Logic Controller, eli ohjelmoitava logiikka. Pieni tietokone, jota käytetään reaaliaikaisten automaatioprosessien ohjaamiseen. Esimerkiksi lämpölaitosprosesseja ohjataan usein ohjelmoitavien logiikoiden avulla. Ohjelmoitavien logiikoiden ohjelmointi tapahtuu niille erikseen suunniteltujen ohjelmointikielien avulla. Yleisimmät logiikkaohjelmointikielet ovat Ladder Diagram, Function Block ja Structure Text.

Prosessiohjain Prosessiohjaimella tarkoitetaan opinnäytetyön yhteydessä yleisesti jotain laitosprosessien ohjaamiseen tyypillisesti käytettävää ohjauslaitetta, kuten etäpäätelaitetta (RTU) tai ohjelmoitavaa logiikkaa (PLC).

Protokolla Tietoliikenteessä protokollalla tarkoitetaan sääntöjärjestelmää, joiden mukaisesti kaksi laitetta välittää toisilleen tietoa. Noudattamalla protokollaa laitteiden välisessä viestinnässä, varmistetaan kummankin laitteen kyky lähettää ja vastaanottaa tietoa toisiltaan. Protokollien standardisoinnilla varmistetaan standardisoitavan protokollan vastaavan kaikkia tietoliikenteelle asetettuja vaatimuksia, kuten tiedon häviämisen havaitseminen ja estäminen.

Redundanssi Redundanssilla ja redundanttisuudella tarkoitetaan opinnäytetyön aiheyhteydessä laitteiden varmentamista. Esimerkiksi, jos automaatiojärjestelmässä jonkin prosessia ohjaavan logiikkaohjaimen, kuten ohjelmoitavan logiikan yhteydet varmistetaan useamman eri kommunikaatioväylän avulla, kutsutaan tätä varmennusta redundanssiksi. Jos taas esimerkiksi asennetaan kaksi palvelinta rinnakkain, jolloin toinen palvelin kykenee ylläpitämään SCADA-järjestelmää, jos ensimmäinen palvelin hajoaa, on kyseessä redundanttinen palvelin.

RTU Remote Terminal Unit, eli etäpäätelaite. Etäpäätelaitetta käytetään samaan tarkoitukseen kuin ohjelmoitavaa logiikkaa. Etäpäätelaitteiden ohjelmointi poikkeaa kuitenkin ohjelmoitavien logiikoiden ohjelmoinnista siinä, että joitain etäpäätelaitteita kyetään konfiguroimaan web-pohjaisilla applikaatioilla, tai tietokoneohjelmoinnissa käytettävillä ohjelmointikielillä kuten C++ tai Visual Basic, kun taas ohjelmoitavia logiikoita ohjelmoidaan niiden ohjelmointiin erikseen suunnitelluilla kielillä, kuten Ladder, Function Block tai Structure Text.

SCADA Supervisory Control and Data Acquisition, eli valvonta ja tietojen keruu. SCADA on ohjausjärjestelmä, joka koostuu tietokoneista, tietoliikenneverkostosta, valvomonäyttöjistä ja automaatiojärjestelmästä, joita käytetään jonkin laitosprosessin ohjaamiseen. SCADA-järjestelmien tietoliikenne perustuu usein tagitietokannan käyttämiseen logiikkaohjaimien ja SCADA-palvelimen välisen tietoliikenteen toteuttamiseksi.

SQL Structured Query Language on relaatiotietokantoja varten tehty kyselykieli, jolla relaatiotietokantoihin voidaan tehdä muutoksia, lisäyksiä tai hakea tietota. SQL-tietokantaa varten on olemassa erilaisia palvelimia, kuten Microsoft SQL Server ja MySQL. SCADA-järjestelmissä käytetään SQL-tietokantoja historiadatan säilyttämiseen.

Standardi Standardi on liuta sääntöjä ja ohjeita, jonka jokin järjestö on kehittänyt. Standardeja noudattamalla voidaan todistaa esimerkiksi tuotteiden valmistuksessa tuotteen laatu ja turvallisuus. Standardit eivät kuitenkaan ole pakollisia noudattaa, mutta joitain standardeja voidaan määrätä laissa noudatettavaksi. Toisaalta esimerkiksi tietotekniikassa standardeja on tärkeää noudattaa, jotta laitteiden välinen kommunikaatio olisi mahdollista.

Tagi Monimutkaisemmissa ohjelmoitavien logiikoiden ohjelmissa hyödynnetään niin kutsuttua tagitietokantaa, joka sijaitsee ohjelmoitavan logiikan muistissa. Tagilla tarkoitetaan nimeä, joka on annettu jollekin yksittäiselle muuttujalle, joka on logiikan muistissa. Tagi sisältää muuttujan nimen lisäksi tiedon muuttujan datatyypistä, joka PLC-ohjelmistossa voi olla esimerkiksi kokonaisluku, binäärimuuttuja tai merkkijono.

Tietokanta Tietokannalla tarkoitetaan kokoelmaa dataa, joka on järjestetty jonkin rakenteen mukaisesti. Opinnäytetyössä keskitytään relaatiotietokantoihin, joita varten on kehitetty oma kielensä SQL. Automaatiojärjestelmän tagit sijaitsevat myös omassa tietokannassaan, josta niihin päästään reaaliaikaisesti käsiksi valvomo-ohjelmistolla SCADA-palvelimen kautta.

1 JOHDANTO

Opinnäytetyön tarkoituksena on kehittää testausdokumentti KPA Uniconin PlantSys SCADA-järjestelmän valvomojen testaamiseksi. Opinnäytetyön aihe valikoitui työskennellessäni kyseisessä yrityksessä opintojeni ohella. Aihe on yritykselle ajankohtainen, sillä yrityksellä ei ole aiempaa virallista testausdokumenttia SCADA-järjestelmien testausta varten. Yrityksen toiveena oli kehittää testausdokumentti, jonka avulla saadaan tehostettua yrityksen testausprosessia. Testausprosessin osaksi haluttiin luoda myös testausympäristö, jota hyödyntämällä voitaisiin turvallisesti käyttää automaattitestaustyökaluja SCADA-valvomoiden testaamiseen.

Opinnäytetyön teoriaosuudessa pohjustetaan automaation ohjausjärjestelmien, testauksen sekä simuloinnin teoriaa. Automaation ohjausjärjestelmien osalta käydään läpi yleisimmin käytettyjen automaation ohjausjärjestelmien teoriaa, jotta voidaan verrata niitä SCADA-järjestelmän kanssa. Itse SCADA-järjestelmän rakenteeseen ja toimintaan työssä keskitytään automaation ohjausjärjestelmistä eniten. Testaukseen liittyvässä teoriassa perehdytään tarkemmin ohjelmistotestaukseen ja erityisesti siihen, miten ohjelmistotestauksen käytäntöjä voidaan soveltaa SCADA-järjestelmien testauksessa. Ohjelmistotestauksen teoriassa tutustutaan erityisesti sen dokumentointiin, sekä turvallisuuden kannalta kriittisten järjestelmien, kuten SCADA-järjestelmien testaamiskäytäntöihin. Lopuksi käsitellään yleisellä tasolla simuloinnin ja simulaatioiden teoriaa. Siihen ei kuitenkaan paneuduta syvällisemmin, sillä simulaation käyttö testausmenetelmänä koettiin liian aikaa vieväksi toteuttaa opinnäytetyön laajuus huomioon ottaen.

Teoriaosuuden jälkeen kerrotaan tarkemmin, miten testausympäristön ja testausdokumentin suunnittelu toteutettiin. Kuvaillaan, kuinka testausympäristö luotiin ja miten sitä voidaan hyödyntää. Lisäksi esitellään opinnäytetyön tuloksena syntyneen testausdokumentin rakenteesta ja millä tavoin sitä tullaan käyttämään osana valvomoiden testausprosessia. Testausympäristön yhteydessä kerrotaan myös työn aikana kehitetystä automaattitestaustyökalusta, jota voidaan hyödyntää valvomoiden hälytysten testaamisessa.

1.1 KPA Unicon

KPA Unicon on pieksämäkeläinen perheyritys, joka on toiminut vuodesta 1990 asti vihreän energiantuotannon ja käytön parissa. KPA Unicon suunnittelee, rakentaa ja toimittaa lämpö- ja voimalaitoksia ympäri maailmaa. Lisäksi yritys tuottaa operointi- ja kunnossapitopalveluita, sekä toimii osaomistajana eräissä laitoksissa. KPA Unicon tarjoaa myös asiantuntijapalveluita etäkäytön, kunnossapidon, käyttöönoton ja prosessioptimoinnin saralla. (KPA Unicon a. 2020-5-12.)

KPA Uniconilla on nykyään toimipisteitä Pieksämäellä, Helsingissä, Tampereella, Kuopiossa, Haapavedellä ja Kempeleellä. Lisäksi KPA Uniconilla on kaksi pajaa, toinen Kiuruvedellä ja toinen Lapualla. KPA Uniconin alueelliset sivukonttorit sijaitsevat puolestaan Venäjällä, Chilessä, Kroatiassa, Ranskassa, Iso-Britanniassa ja Ruotsissa. (KPA Unicon a. 2020-5-12.)

Yritys työllistää tällä hetkellä yli 240 henkilöä Suomessa ja ulkomailla. KPA Uniconissa työskennellään monipuolisissa työtehtävissä energia-, prosessi-, sähkö-, ja valmistustekniikan parissa. KPA Uniconilla työskentelee insinöörien lisäksi myös monien muiden alojen osaajia. (KPA Unicon b. 2020-5-12.)

1.2 PlantSys

PlantSys Operation on Inductive Automationin Ignition -ohjelmointiympäristöllä kehitetty biolämpö- ja -voimalaitoksille suunniteltu valvomojärjestelmä. Lisäksi PlantSys tuoteperheeseen kuuluvat Maintenance, Reporting ja Material Flow moduulit. (KPA Unicon c. 2020-5-12.)

PlantSys Operation tarjoaa käyttäjäystävällisen visuaalisen käyttöliittymän, joka sopeutuu kaikille eri kokoluokkien puhtaan energian laitoksille. PlantSys Operation tarjoaa kaiken laitokselta saatavan informaation reaaliaikaisena trendejä, hälytyksiä ja mittauksia varten. (KPA Unicon c. 2020-5-12.)

PlantSys Maintenance on kattilalaitoksille suunniteltu kunnossapitojärjestelmä. Maintenance-järjestelmällä voidaan aikatauluttaa kaikki laitoksen huollot ja jakaa ne huoltajien kesken. Tällöin kaikki pysyvät ajan tasalla siitä, mitkä laitteistot ovat jo huollettu ja mitkä ovat vielä huoltamatta. Kunnossapitojärjestelmään voidaan myös asettaa laitteille eliniät, joiden perusteella järjestelmä luo automaattisesti huoltotehtäviä sitä tarvitseville laitteille. (KPA Unicon c. 2020-5-12.)

PlantSys Reporting kerää ja tiivistää kaiken laitosautomaatiolta saatavan tiedon loogiseksi kokonaisuudeksi. Reporting-ohjelmistolla kyetään tarkastelemaan laitoksen tehokkuutta ja muita mittausarvoja tuntikeskiarvoina, tai tarvittaessa jopa minuutin välein. PlantSys Reporting on saatavilla kaikkialla, missä on toimiva internet-yhteys. (KPA Unicon c. 2020-5-12.)

Material Flow moduulilla laitoksilla voidaan valvoa laitoksille tulevia polttoainetoimituksia. Material Flow-moduulilla polttoaineen kuljettaja kykenee kuittaamaan laitokselle tuodessaan toimittamansa polttoaineen, jolloin laitoksen reaaliaikaisen polttoainetilanteen seuraaminen helpottuu merkittävästi. (KPA Unicon c. 2020-5-12.)

1.3 Ignition

Inductive Automationin Ignition SCADA on Java pohjainen SCADA-järjestelmien suunnitteluun ja käyttämiseen tarkoitettu ohjelmistokokonaisuus. Standardi Ignition paketin mukana tulevat moduulit sisältävät OPC UA ajurit, Designerin, jolla suunnitellaan valvomoita, sekä rajattoman määrän clienttejä, joilla voidaan avata Ignition Designerilla suunniteltuja valvomoita. (Inductive Automation, 2020-5-12.)

Ignition Designer on valvomoiden suunnitteluun tarkoitettu ohjelmointiympäristö. Ignition Designer on suunniteltu toimimaan kaikilla käytetyimmillä käyttöjärjestelmillä. Ignition Designerin etuna on myös, että sen lataaminen ja käyttäminen on ilmaista. Tämän mahdollistaa Inductive Automationin tarjoama kahden tunnin kokeilujakso, jonka voi uusia rajattomasti. Näin ollen lisenssin voi hankkia vasta, kun valvomo on jo valmis käyttöönotettavaksi. (Inductive Automation, 2020-5-12.)

Ignitionilla suunnitellun valvomon voi avata keskuspalvelimen kautta millä tahansa siihen yhteydessä olevalla laitteella. Ignitionin lisenssi ostetaan palvelinkohtaisesti. Tältä palvelimelta voidaan hakea tietoa niin monelle päätelaitteelle, kuin vain palvelimen suorituskyky antaa myöten. Ignitionin palvelimelle PLC:ltä tulevat tiedot voidaan halutessaan tallentaa erilliseen tietokantaan, jolloin historiatietoja voidaan säilyttää rajattomasti. Ignition tukeekin kaikkia suosituimpia relaatiotietokantoja, kuten Microsoft SQL Server, Oracle ja MySQL. (Inductive Automation, 2020-5-12.)

2 YLEISIMMÄT AUTOMAATION OHJAUSJÄRJESTELMÄT

Tässä kappaleessa käsitellään yleisesti automaationohjausjärjestelmien merkitystä osana laitosten automaatiota. Lisäksi perehdytään tarkemmin yleisimpiin erilaisiin automaationohjausjärjestelmiin ja niiden eroihin. Nämä kolme yleisintä tässä kappaleessa käsiteltävää automaationohjausjärjestelmää ovat DCS, DDC ja SCADA. Opinnäytetyössä keskitytään SCADA-järjestelmiin, joten kappaleessa keskitytään vertailemaan muiden järjestelmien yhtäläisyyksiä ja eroavaisuuksia SCADA-järjestelmän kanssa.

Teollisuuden ohjausjärjestelmien tarkoituksena hallita instrumentaatiota, jota käytetään teollisuuden prosesseissa. Teollisuuden ohjausjärjestelmät voidaan luokitella seuraavien toiminnallisuuksien mukaan, joista niillä voi olla yksi tai useampia. Nämä toiminnallisuudet ovat tarkkailu, monitorointi ja ohjaus. (Ackerman 2017, 9-11.)

Tarkkailulla tarkoitetaan järjestelmän kykyä viestittää prosessitietoja operaattorin nähtäväksi esimerkiksi valvomonäyttöjen välityksellä. Tällöin operaattori voi tarkkailla prosessin kulkua ja tarvittaessa säätää prosessia, jotta prosessin normaali toiminta ei häiriintyisi tai keskeytyisi. Monitorointi puolestaan poikkeaa tarkkailusta siinä määrin, että monitoroinnissa automaationohjausjärjestelmä reagoi prosessiarvojen muutoksiin ja tekee toimenpiteitä automaattisesti. Näitä automaattisia toimenpiteitä ovat esimerkiksi hälytysten näyttäminen operaattorille, tai jopa tarvittaessa koko prosessin keskeyttäminen. Kolmas toiminto, eli ohjaustoiminto puolestaan tarkoittaa yleisesti koko järjestelmän ja/tai prosessin ohjaamista aina yksittäisistä toimilaitteista ja venttiileistä suuriin moottoreihin ja puhaltimiin. Tämä ohjaus voi tapahtua joko operaattorin käskystä nappia painamalla, tai epäsuorasti, kun operaattori säätää prosessin asetusarvoa, jonka seurauksena automaatiojärjestelmä säätää laitteita uuden asetusarvon saavuttamiseksi. (Ackerman 2017, 9-11.)

DCS (Distributed Control System), eli hajautettu ohjausjärjestelmä ja SCADA ovat nykyään melko samankaltaisia useimmilta toiminnallisuuksiltaan. Perinteisesti SCADA järjestelmiä käytettiin laajoille maantieteellisille alueille ulottuvien prosessien ohjaukseen, kun taas DCS keskittyi yksittäisten laitosten ohjaukseen. (Ackerman 2017, 15.) Hajautetussa ohjausjärjestelmässä tiedonkeruu ja ohjaustoiminnot toteuttavat lukuisat mikroprosessoripohjaiset yksiköt, jotka ovat sijoitettu lähelle niiden ohjaamia laitteistoja (Bailey ja Wright 2003, 15). DCS järjestelmäarkkitehtuurissa keskitytään redundanttisuuteen ja se ilmenee järjestelmän kaikissa osissa aina ohjaimista ja antureista redundanttiseen palvelimeen asti (Ackerman 2017, 15). SCADA-järjestelmän suunnittelussa redundanttisuus tulee erikseen huomioida, kun halutaan varmentaa prosessin keskeytymätön toiminta.

DCS-järjestelmissä kaikki laitteet HMI-näytöistä prosessiohjaimiin kuuluvat jonkin yhden yrityksen tuoteperheen alle. Tällöin kaikki laitteet tukevat samoja patentoituja kommunikaatiotekniikoita ja näin ollen kaikki DCS-järjestelmän laitteet toimivat suoraan keskenään. Esimerkiksi ohjelmoitavia logiikoita käytettäessä turvaudutaan erillisiin kommunikaatioratkaisuihin, jotta PLC:t saadaan kommunikoimaan keskenään. (RealPars a. 2018.)

Suoralla digitaalisäädöllä (DDC) tarkoitetaan ohjausprosessia, jossa mikroprosessori ylläpitää sisäistä tietokantaa, joka sisältää ohjattavan järjestelmän tiedot. Näiden tietojen perusteella säädin säätää prosessiasetuksia olosuhteiden muutosten mukaisesti. Tarkemmin, DDC on kolmevaiheinen signaalin muokkausprosessi, jossa tulevat informaatiosignaalit muutetaan analogisesta digitaalisiksi, jolloin mikroprosessoriohjain ymmärtää tulevan analogisen signaalin mukana tulevan tiedon. Tämän jälkeen mikroprosessori, eli tietokone, antaa ohjauskomennot pohjautuen vastaanottamaansa tietoon ja tämä käsky muunnetaan binääritiedosta analogiseksi lähtötiedoksi, jonka ohjattava laite puolestaan ymmärtää. (Coffin 1999, 1.) Suorassa digitaalisäädössä järjestelmää ohjaavan tietokoneen toiminta on kriittisessä roolissa järjestelmän toiminnan kannalta. Tämän vuoksi suoraa digitaalisäätöä ei käytetä sellaisten prosessien ohjaamisessa, joissa ohjausjärjestelmän vioittuminen voisi aiheuttaa suuria kustannuksia. (Chaudhuri ja Chaudhuri 2012, 236.)

3 SCADA JÄRJESTELMÄ

SCADA:lla (Supervisory Control and Data Acquisiton) viitataan telemetriaan ja tiedonkeruuseen. Tämä kerätty tieto tarvittaessa käsitellään, eli tehdään tarvittavat laskutoimitukset, jonka jälkeen tieto esitetään operaattorille valvomonäytöillä. Näitä mittaustietoja tarkastelemalla operaattori voi tarpeen mukaan säätää ohjattavaa prosessia. (Bailey ja Wright 2003, 2.)

SCADA-järjestelmiä on ollut olemassa niin kauan, kuin on ollut ohjausjärjestelmiäkin. Varhaisissa SCADA-järjestelmissä hyödynnettiin tiedon keruuta mittareilla, valoilla ja nauhakaavioilla. Näiden järjestelmien ohjaus tapahtui manuaalisilla painikkeilla ja vivuilla. (Bailey ja Wright 2003, 1.)

Näissä perinteisissä anturilta–paneelille tyyppisissä SCADA järjestelmissä oli omat etunsa. Tällaisen järjestelmän luomiseen ei tarvittu ollenkaan ohjelmointia, tai laskentatehoa omaavia laitteita. Samalla tällaiseen järjestelmään oli useimmiten helppoa lisätä uusia yksinkertaisia laitteita, kuten kytkimiä tai ilmoitusvaloja. (Bailey ja Wright 2003, 2.)

Perinteisessä SCADA järjestelmässä oli kuitenkin huonotkin puolensa. Tällaisessa järjestelmässä johdotuksien määrä kasvoi helposti hallitsemattomaksi, kun järjestelmän koko kasvoi. Järjestelmästä kerättävä tieto oli myös alkeellista, eikä sitä kyetty varastoimaan suuria määriä. Kaikki prosessin valvonta tapahtui paikallisessa valvomossa, joten etävalvontamahdollisuuksia ei ollut, jonka vuoksi prosessia täytyi olla koko ajan valvomassa paikan päällä. (Bailey ja Wright 2003, 2.)

Modernilla reaaliaikaisella SCADA järjestelmällä laitoksen ja sen prosessien ohjaaminen mahdollistaa niiden toiminnan optimoimisen. Samalla reaaliaikaisella valvonnalla lisätään tehokkuuden lisäksi luotettavuutta ja turvallisuutta. Kaiken tämän ansiosta laitoksen operointikustannukset ovat pienemmät, kuin vanhoissa ei-automatisoiduissa järjestelmissä. (Bailey ja Wright 2003, 12.)

3.1 Miksi SCADA?

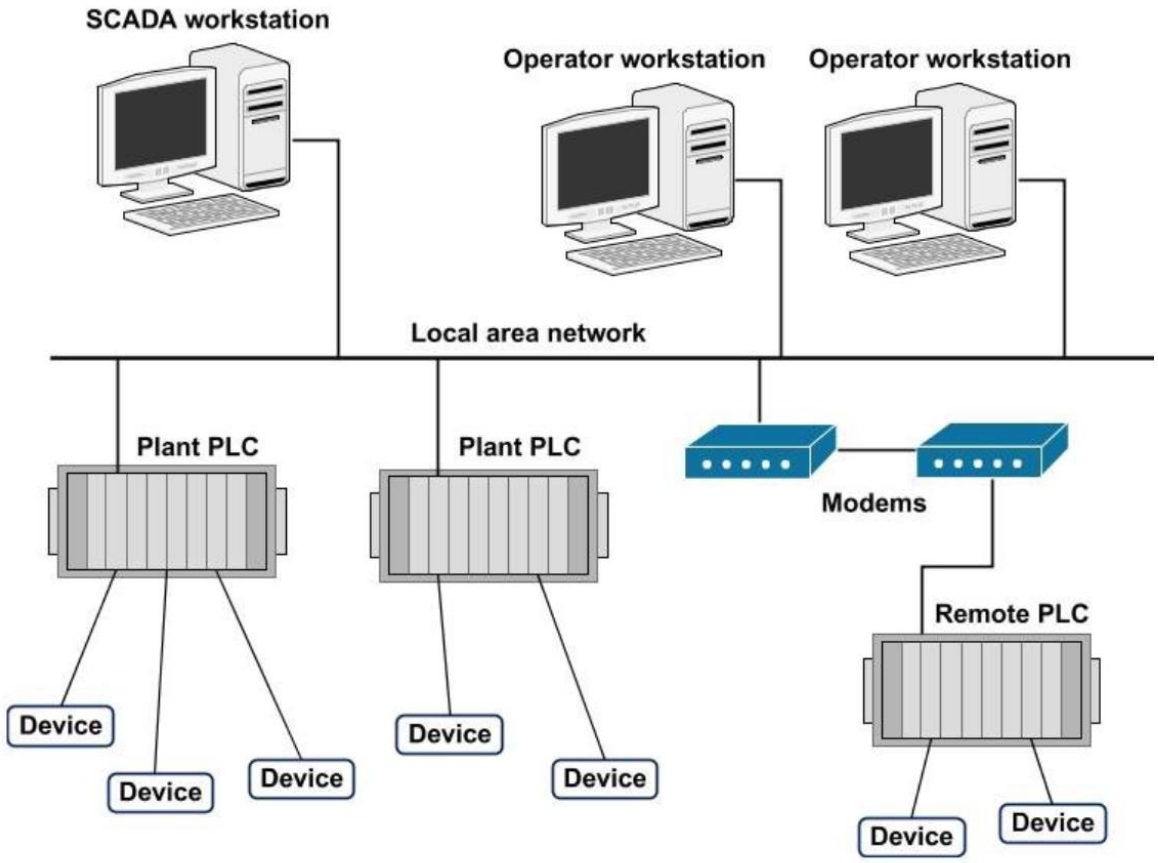
SCADA-järjestelmällä operaattori kykenee ohjaamaan laaja-alaistakin prosessia keskitetystä sijainnista. SCADA:n hyödyllisyyttä parantaa vielä entisestään, jos prosessi, järjestelmä tai jokin niiden osa on vaikeasti tavoitettavassa sijainnissa. (Boyer 2010, 17.) SCADA järjestelmään kyetään tallentamaan valtavia määriä dataa. Siten järjestelmään voidaan liittää suuria määriä sensoreita. SCADA-järjestelmän vahvuus on myös siinä, että mittaustietoja kyetään tarkastelemaan etäyhteyksillä muualtakin, kuin itse operoitavalta laitokselta. Lisäksi SCADA-järjestelmän antamaa dataa kyetään hyödyntämään monipuolisemmin, kun dataa voidaan käsitellä ja tarkastella käyttäjän haluamilla tavoilla. (Bailey ja Wright 2003, 3.)

SCADA-järjestelmän käyttäminen parantaa laitoksen hyötysuhdetta, kun tuotantotietoja seuraamalla voidaan optimoida laitoksen käyttöä tehokkaammaksi. Prosessin turvallisuus paranee, kun kyetään paremmin hallitsemaan ja tarkkailemaan prosessia. Samalla henkilökunnan tehokkuus paranee, kun he pystyvät paremmin havainnoimaan ja säätämään prosesseja. (Bailey ja Wright 2003, 17.) SCADA järjestelmä laajentaa samalla operaattorin mahdollisuutta havainnoida ja ohjata prosessia. Suurissa prosesseissa yksittäisen operaattorin olisi mahdotonta tarkkailla koko prosessia samanaikaisesti, ilman valvomojärjestelmän apua. (Boyer 2010, 18-19.)

SCADA-laitevalmistajat tukevat avoimia kommunikaatioprotokollia. Tämän vuoksi eri valmistajien valmistamista laitteista koostuva SCADA-järjestelmä on mahdollista toteuttaa. (RealPars b. 2019.) Tämä puolestaan halventaa SCADA-järjestelmän suunnittelua, sillä useampia eri järjestelmän osia voidaan kilpailuttaa, toisin kuin ei-avoimissa patentoiduissa järjestelmissä, joissa kaikki järjestelmän komponentit hankitaan yhtenä kokonaisuutena.

3.2 Rakenne

Tyypillinen SCADA-järjestelmä koostuu operaattorin työasemista, SCADA-palvelimista, kommunikaatioverkoista, ohjelmoitavista kenttäohjaimista, kenttälaitteista ja signaaleista (McCrady 2013, 11). SCADA-järjestelmissä käytetään tyypillisesti prosessiohjaimena PLC:tä (Programmable Logic Controller), eli ohjelmoitavaa logiikkaohjainta tai RTU:ta (Remote Terminal Unit), eli etäpäätelaitetta. Kummankin laitteen käyttäminen on mahdollista myös samassa SCADA-järjestelmässä ja siihen voidaan turvautua esimerkiksi silloin, kun tarvitaan RTU:ta jonkin vaikeasti saavutettavan sijainnin yhdistämiseksi järjestelmään langattomasti. (RealPars c. 2018.)



Kuva 1. Tyypillinen SCADA-järjestelmäarkkitehtuuri (McCrady 2013, 12).

Lämpö- ja voimalaitosprosesseissa hyödynnetään usein Kuva 1 mukaista SCADA-arkkitehtuuria, joka pohjautuu ohjelmoitavilla logiikoilla toteutettuun automaatiojärjestelmään. Kuvan järjestelmässä erillisen RTU:n käytön sijasta on kytketty PLC etäisesti SCADA järjestelmään modeemeilla. Lämpö- ja voimalaitoksilla RTU-laitteiden hyödyntäminen on vähäisempää, koska suurin osa biolämpö-, ja -voimalaitosten laitteistoista sijoittuvat kattilarakennuksen sisälle tai sen läheisyyteen.

Pääasema koostuu yhdestä tai useammasta operaattorin työasemasta, jotka ovat yhdistettynä kommunikaatiojärjestelmään (Bailey ja Wright 2003, 46). Operaattorin työasema, jota kutsutaan usein myös HMI:ksi (Human Machine Interface), eli ihmisen ja koneen väliseksi käyttöliittymäksi, vaatii ohjelmointia, jossa linkitetään prosessin tietokanta graafisiin elementteihin operaattorin valvomonäytöillä. Näillä graafisilla näytöillä operaattori voi esimerkiksi ohjata laitteita tai hakea tietoa prosessin tilasta. SCADA-ohjelmistoon kuuluu prosessinäyttöjen lisäksi historiatietojen, hälytys- ja tapahtumalokien valvomista varten luodut näytöt. (McCrady 2013, 15.)

Laitokselle tulevan SCADA-ohjelmiston suunnittelussa tärkeää on huomioida sen suorituskyky ja kustannustehokkuus kyseisellä laitoksella, mutta ohjelmiston laajennettavuus tulevaisuuden tarpeisiin on vielä tärkeämpää. SCADA-ohjelmiston toteutuksessa tulee siis käyttää skaalattavaa arkkitehtuuria, jotta tarvittavat laajennukset ja laitteistomuutokset laitoksella ovat helposti lisättävissä SCADA-ohjelmistoon. (Bailey & Wright 2003, 68.)

SCADA ohjelmistot voidaan jakaa kahteen pääkategoriaan: patentoituihin ja avoimiin. Patentoidut järjestelmät ovat automaatiojärjestelmävalmistajien itse luomia järjestelmiä, joilla voidaan ottaa yhteys valmistajan omaan automaatiojärjestelmään. Avoimet SCADA järjestelmät puolestaan mahdollistavat yhteyden muodostamisen lukuisten valmistajien automaatiojärjestelmiin. (Bailey ja Wright 2003, 5.) PlantSys-ohjelmisto lukeutuu avoimiin SCADA-järjestelmiin. Hyödyntämällä OPC-UA kommunikaatioprotokollaa laitosautomaation ja PlantSys järjestelmän välillä, voidaan PlantSys SCADA-järjestelmä kytkeä lukuisiin eri automaatiojärjestelmiin.

Useimmissa SCADA-järjestelmissä on yksi, tai jopa kaksi palvelintietokonetta SCADA järjestelmän ylläpitämistä varten. Pienissä prosesseissa, jotka sisältävät vain muutamia logiikkaohjaimia, voidaan palvelimen rooli hoitaa valvomotietokoneella, mutta suuremmissa prosesseissa vaaditaan erillinen tietokone hoitamaan palvelimen tehtäviä. Ajan mittaan prosessista kertyviä historiatietoja säilytetään palvelimella tietokantojen muodossa. Lisäksi palvelimella säilytetään reaaliaikaista prosessidataa samaisessa tietokantamuodossa. Palvelintietokone hoitaa myös kaiken SCADA-verkostossa tapahtuvan kommunikaation logiikkaohjaimien ja valvomotietokoneiden välillä. Logiikkaohjaimet ohjaavat omaa prosessin osaansa ja keräävät siitä tietoa. Nämä tiedot SCADA-palvelin vastaanottaa ja päivittää niillä prosessin historiatietokantoja. (McCrady 2013, 18.)

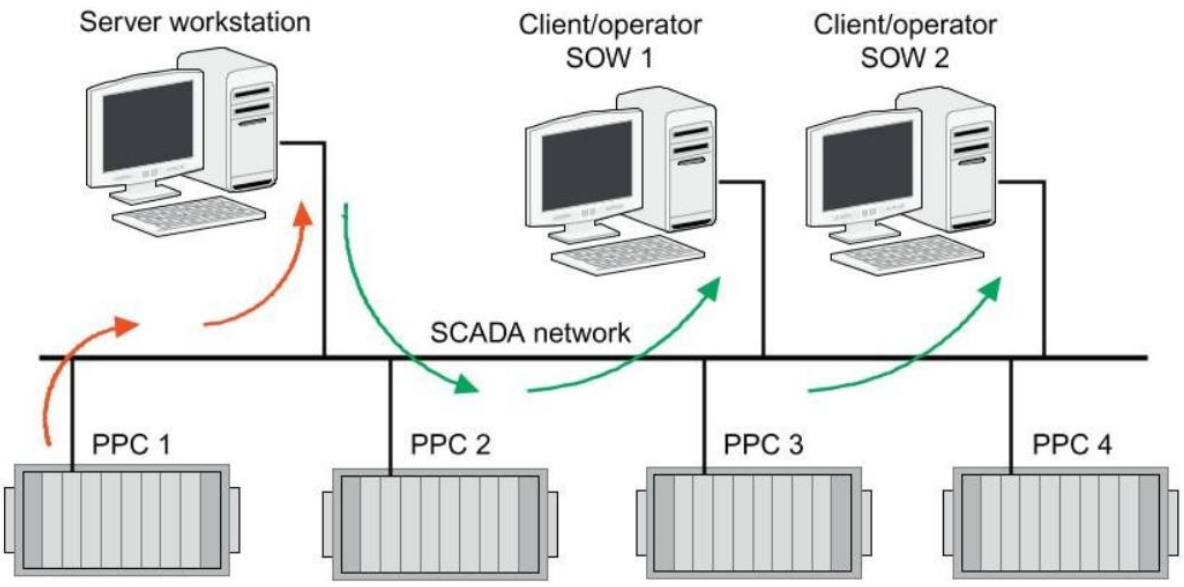
SCADA-järjestelmää ylläpitävän palvelimen toiminta tulee varmentaa. Yksi tapa varmentaa palvelimen toiminta on käyttää kahta rinnakkaista palvelinta, jolloin toisen palvelimen vioittuessa toinen palvelin kykenee ylläpitämään SCADA- järjestelmää. Myös muita kriittisiä komponentteja, kuten kytkimet, joiden kautta palvelimien viestintäyhteydet viedään operaattorin työasemille, sekä ohjelmoitaville logiikoille. Tätä varmennustapaa kutsutaan redundanssiksi ja se on hyvä ottaa huomioon myös laitoksen tietoliikenneverkoston suunnittelussa valittaessa käytettävää verkkotopologiaa, eli verkon perusrakennetta. (McCrady 2013, 155.)

3.3 Kommunikaatio

Prosessiohjaimilla on olemassa vain neljä erilaista datatyyppiä, jotka ohjaimelle tulevat tai siltä lähtevät. Nämä tyypit ovat digitaalinen tulo ja lähtö, sekä analoginen tulo ja lähtö. (McCrady 2013, 3.) Prosessiohjaimen ja kenttäinstrumentaation välinen kommunikaatio toteutetaan näillä digitaalisilla ja analogisilla signaaleilla. Monet instrumentit lähettävät tilatietonsa etäpäätelaiteelle tai ohjelmoitavalle logiikalle sähköisenä viestinä samaa kuparilankaa pitkin, jolla se on kytketty etäpäätelaitteeseen tai ohjelmoitavaan logiikkaan. Tässä tapauksessa esimerkiksi analogisen pinnanmittausanturin mittaustieto muutetaan ampeeritiedoksi, jossa tyhjä säiliö vastaa 4 mA:n virtaa ja täysi säiliö puolestaan 20 mA:n virtaa. (Boyer 2010, 66-67.) Analogista signaalia voidaan kuvastaa myös tasajännitteen avulla, esimerkiksi jännitevälillä 1-5 VDC. Digitaalisella signaalilla puolestaan on kaksi eri tilaa, jota kuvastetaan sähkövirran avulla. Virta joko kulkee, eli signaali on true (tosi), tai virta ei kulje ja signaali on false (epätosi). Ohjelmassa tämä signaali ilmenee arvona yksi (tosi) tai nolla (epätosi). (McCrady2013, 3.)

Prosessiohjaimelta lähteviä analogisia arvoja voidaan hyödyntää moottorien, säädettävien venttiilien ja kaikkien muiden muuttujilla ohjattavien laitteiden ohjaukseen. Analogisia tuloarvoja puolestaan voidaan käyttää esimerkiksi lukituksissa, kuten kattilan paloprosessin sammuttamisessa, jos kattilan lämpötila-anturille asetettu yläraja-arvo ylittyy. (McCrady 2013, 4.) Analogisia tulotietoja käytetään myös suoraan prosessiarvojen, kuten lämpötilan esittämiseen valvomonäytöllä. Digitaalisia lähtöarvoja puolestaan käytetään yleisesti moottorien käynnistämiseen ja sammuttamiseen, sekä niiden ohjauksen päälle tai pois asettamiseen (McCrady 2013, 4). Digitaalisia tulotietoja puolestaan ovat monet pinnankorkeus, pyörintä ja tukosvahtien tilatiedot.

Instrumentit eivät kykene häiritsemään toistensa ja etäpäätelaitteen välistä kommunikaatiota, sillä jokainen instrumentti on kytkettynä etäpäätelaitteeseen erillisellä kaapelilla. Lisäksi, jos etäpäätelaitteen on tarve välittää tietoa kenttäinstrumentille, on sitä varten yleensä oma erillinen kaapelinsa etäpäätelaitteen ja instrumentin välillä, jolloin etäpäätelaite ei myöskään häiritse instrumentilta tulevaa viestintää. (Boyer 2010, 67.) Pääaseman ja etäpäätelaitteiden tai ohjelmoitavien logiikoiden välisessä kommunikaatiossa hyödynnetään master-slave periaatetta. Master-laite, eli pääasema on ainoa laite, joka voi aloittaa keskustelun pääaseman ja etäpäätelaitteen välillä ja slave-laite, eli etäpäätelaite vain vastaa, kun pääasema kutsuu sitä. (Boyer 2010, 120.) Pääaseman, tarkemmin SCADA-palvelimen, ja etäpäätelaitteiden välisessä kommunikaatiossa kaikille mittaus- ja tilatiedoille täytyy olla olemassa yksilöllinen nimitys. Tätä nimitystä kutsutaan SCADA- ja automaatiojärjestelmien yhteydessä yleisesti tagiksi. (McCrady 2013, 32.)



Kuva 2. Havaintoesimerkki SCADA-järjestelmän tietoliikenteestä (McCrady 2013, 135).

Palvelintietokone tai palvelimena toimiva työasema vastaa viestinnästä logiikan ja SCADA-järjestelmän välillä. Kaikki operaattorin työasemilla näkyvä tieto kulkee tämän palvelimen kautta. Kuva 2 tätä viestintää on havainnollistettu nuolien avulla. Punaisella nuolella kuvastetaan prosessiohjainten lähettämää dataa. Vihreillä nuolilla puolestaan kuvastetaan palvelimen käsittelemää ja työasemille välittämää dataa. (McCrady 2013, 134-135.)

3.4 SCADA järjestelmän liittäminen automaatiojärjestelmään

Kun suunnitellaan SCADA järjestelmän liittämistä jo olemassa olevaan infrastruktuuriin, tulee kartoittaa tarkasti jo olemassa oleva kommunikaatioinfrastruktuuri. Tällöin voidaan säästää huomattavasti, kun kyetään hyödyntämään jo valmiina olevia kommunikaatioväyliä. Täytyy kuitenkin muistaa, että uuden järjestelmän kytkemisessä tulee olla varovainen, jotta jo valmiina olevat yhteydet eivät häiriinny uusien lisäämisestä. (Bailey ja Wright 2003, 10.)

Jos SCADA järjestelmä yhdistetään kokonaan uuteen automaatiojärjestelmään, on puolestaan huomioitava järjestelmän toiminnallisuus ja laajuus suhteessa järjestelmän kokoon. Erityistä huomiota tulee kiinnittää järjestelmän suorituskykyyn ja eheyteen. (Bailey ja Wright 2003, 10.)

Kun päivitetään automaationohjausjärjestelmää, voidaan ohjattavan järjestelmän iän mukaan joutua uusimaan myös laitteita. Vanhojen automaatiojärjestelmien tagijärjestelmät ovat joskus erittäin lyhyissä yksittäisiin numero ja kirjainsarjoihin perustuvissa muodoissa. Jos automaatiojärjestelmää ei päivitetä, on automaatio-ohjelmoijan suotavaa päivittää järjestelmän tagit, jotta tagit olisivat merkityksellisempiä. (McCrady 2013, 7-8.) Tällöin tagien liittäminen SCADA järjestelmään helpottuu, sillä tagien nimistä kyetään päättelemään, mitä tietoa kyseinen tagi välittää.

PlantSys SCADA-järjestelmä yhdistetään automaatiojärjestelmään hyödyntämällä Ignitionin OPC UA-moduulia, mikä mahdollistaa kommunikaation PlantSys SCADA-palvelimen ja automaation välillä. Automaatiojärjestelmän mittaus- ja tilatiedot välitetään logiikalta OPC UA protokollaa tukevalle palvelimelle, kuten KEPServerEX-palvelimelle. KEPServerEX-palvelimelta tiedot puolestaan voidaan hakea Ignition-palvelimelle, jolta tiedot lopulta saadaan näkyviin PlantSys SCADA-järjestelmässä.

KEPServerEX on liitäntäalusta, jonka avulla kaikki automaatiojärjestelmän tieto voidaan keskittää yhdeksi kokonaisuudeksi. Alustan suunnittelun avulla käyttäjä voi yhdistää, hallita, monitoroida ja ohjata monipuolisesti eri automaatiolaitteita ja -ohjelmistoja KEPServerEX käyttöliittymän kautta. KEPServerEX-palvelin tukee OPC-standardia, joten sen yhdistäminen lukuisiin eri järjestelmiin on helppoa. KEPServerEX mahdollistaa myös eri laitevalmistajien laitteiden välisen suoran laitteelta–laitteelle tapahtuvan kommunikaation, mikä ei välttämättä suoraan olisi mahdollista. (Kepware 2020-05-10.)

OPC UA on vuonna 2008 julkaistu alustariippumaton palveluorientoitunut arkkitehtuuri, joka integroi OPC Classicin määritelmät yhdeksi laajennettavissa olevaksi kehykseksi. OPC UA protokollan etuina ovat sen yhteensopivuus useimpien eri valmistajien ohjelmoitavien logiikoiden kanssa. Lisäksi OPC UA on yhteensopiva kaikkien yleisimpien käyttöjärjestelmien, kuten Linuxin, Windowsin, Androidin tai Apple OSX:n kanssa. (OPC Foundation 2020-03-20.)

4 TESTAAMINEN YLEISESTI

Testaamisella tarkoitetaan yksinkertaisuudessaan joukkoa toimintoja, joiden tavoitteena on löytää vikoja tuotteessa tai järjestelmässä (Homés 2012, 8). Testausta voidaan hyödyntää laaja-alaisesti eri alojen kehitys ja suunnitteluprojekteissa. Testaamisen päätavoitteena on pienentää yllättävien suunnittelukustannuksien syntymisen ja tuotteen viallisuuden riskiä. (Everett ja McLeod 2007, 9.)

Testaustauksen päätavoitteen voi jakaa vielä tarkemmin neljään päävaiheeseen. Ensimmäinen vaihe on tunnistaa kehitysvaiheeseen liittyvät riskit, jotka testaamalla ovat pienennettävissä. Toisessa vaiheessa suoritetaan testausta, jolla tunnistettuja riskejä minimoidaan. Kolmas tavoite on tietää, milloin testausta on suoritettu riittävästi. Neljäs tavoite on pitää testausta vakioituna osana koko suunnitteluprojektia. Testaamisen suunnittelussa on tärkeää myös miettiä testaamisen järkevyyttä. Jos testaamisen kustannukset ylittävät selvästi testaamisella estettävän riskin aiheuttamat mahdolliset kustannukset, ei kyseisen asian varalta testaaminen ole kovinkaan järkevää. (Everett ja McLeod 2007, 9-10)

Testaamisesen suorittaa usein ihminen, joko manuaalisesti itse testaamalla, tai käyttämällä jotain testaamista varten tehtyä työkalua tai apuvälinettä. Tällöin testauksen onnistumiseen vaikuttavat myös tehtyjen testien lisäksi testaaja, joka testit on tehnyt. Jos testaaja on esimerkiksi kokematon, voi häneltä testatessa jäädä huomioimatta sellaisia epäkohtia, joita kokeneempi testaaja saattaisi ottaa huomioon. Toisaalta jos testaaja testaa samaa asiaa pitkiä aikoja, voi testaaja väsyä testaamiseen ja näin ollen testaamisen laatu heikkenee. Jos testaaja puolestaan on itse suunnitellut tai valmistanut testaamansa asian, voi hänen olla vaikeampi nähdä vikoja testatessaan, sillä tämä tarkoittaisi, että hän on itse tehnyt virheen testattavaa asiaa suunnitellessaan.

5 OHJELMISTOTESTAUS

Testausta suunnitellessa tulee pyrkiä vähentämään testauksen määrää siten että testauksen suorittaminen olisi taloudellisesti kannattavaa. Testauksessa ei siis tule edes pyrkiä täydellisyyteen, sillä se ei käytännössä olisi edes mahdollista. (Homès 2012, 12.) Testaaminen mahdollistaakin vain vikojen havaitsemisen ohjelmistossa. Testaamisella ei kuitenkaan voida osoittaa sitä, etteikö ohjelmistossa olisi ollenkaan virheitä (Sommerville 2007, 539; Homès 2012, 11). Ohjelmistotestauksen tavoitteena onkin vakuuttaa ohjelmiston toimivan riittävän hyvin, että se voidaan ottaa tuotantokäyttöön (Sommerville 2007, 539).

Tarkemmin sanottuna ohjelmistotestauksen tavoitteena on osoittaa ohjelman kehittäjälle ja asiakkalle, jolle ohjelmisto on menossa, että ohjelmiston toiminta vastaa sille asetettuja vaatimuksia. Jos testataan tiettyyn tarkoitukseen räätälöitä sovellusta, tulisi testata kaikkia ohjelmistolle asetettuja vaatimuksia vähintään yhdellä erillisellä testillä. Yleisemmillä ohjelmistoilla testaukseen tulisi sisältyä testi jokaista ohjelmistoon tulevaa ominaisuutta varten. (Sommerville 2007, 538.)

Toisena tavoitteena testaukselle on jo aiemmin mainittu vikojen tai virheiden löytäminen ohjelmistosta. Tällä testauksella yritetään löytää kaikenlaisia ei haluttuja järjestelmän toimintoja. Tällaisia toimintoja ovat esimerkiksi järjestelmän kaatuminen, virheelliset laskelmat tai datan korruptoituminen. (Sommerville 2007, 538.) Järjestelmän kaatumisella tarkoitetaan sitä, että järjestelmä lopettaa toimimisen kesken käytön, ilman että käyttäjä niin haluaa. Datan korruptoituminen puolestaan tarkoittaa sitä, että dataan tulee ei-haluttuja muutoksia, jotka pahimmassa tapauksessa muuttavat datan käyttökelvottomaksi.

Testaaminen on työläs ja kallis prosessi. Sen vuoksi testaamisen avuksi on kehitetty automaattisia työkaluja, jotka tarjoavat laajan skaalan erilaisia toimintoja, jotka halventavat testauksesta aiheutuvia kustannuksia. Esimerkkejä tällaisista testausta helpottavista automaatiotyökaluista ovat testidatageneraattorit, jotka hakevat dataa jostain valmiista tietokannasta, tai luovat uutta dataa jonkin kaavan mukaisesti. Testidatageneraattorin tehtävä on luoda valmista dataa, joka on oikeassa muodossa syötettäväksi testattavaan ohjelmaan. Toisena esimerkkinä on simuloida laitteistoa, jota ohjelmistolla on tarkoitus ohjata, tai jolla sitä on tarkoitus ajaa. (Sommerville 2007, 561-563.)

5.1 Testausympäristöt

Testausympäristöllä voidaan havainnoida ohjelmiston toimintaa asiakkaan näkökulmasta, ennen kuin se viedään tuotantoon, eli asiakkaan käytettäväksi. Testausympäristön tavoitteena on saada testattava ohjelmisto toimimaan mahdollisimman todenmukaisesti, mutta samalla toimia erillään tuotantoympäristöstä. (Everett ja McLeod 2007, 157-158.) Mitä lähempänä testausympäristön toiminta on oikeaa tuotetta, sitä parempi se on testaustuloksia ajatellen. Testaustulosten luotettavuus onkin riippuvainen juuri siitä, miten samanlaiseksi testausympäristö saadaan tuotantoympäristöön verrattuna. (Everett ja McLeod 2007, 158.)

Tavallinen testausympäristö sisältää testaustyökalut, testausdatan ja muut tarvittavat asiat, joita ohjelmiston testaamiseen tarvitaan. Testausympäristössä harvoin kuitenkaan toteutetaan järjestelmätestausta. Järjestelmätestaukseen käytetäänkin usein tuotantoympäristöä tai simulaatiota tuotantoympäristöstä. (Chemuturi ja Cagley 2010, 108.)

Jos kuitenkin käytetään järjestelmätestausympäristöä, tulee sen olla täydellinen kopio tuotantoympäristöstä, jotta voidaan testata miten ohjelma käyttäytyisi oikeassa tilanteessa. Järjestelmätestauksessa testaaja pääsee muuttamaan vain testausdataa tarvittaessa. Testausympäristöä ja järjestelmätestausympäristöä ei kuitenkaan aina eritellä. Joskus yhdistetäänkin järjestelmätestausympäristö ja tavallinen testausympäristö yhtenäiseksi testausympäristöksi. Tämä valinta tehdään ohjelmiston kehittäjän tarpeiden pohjalta. (Chemuturi ja Cagley 2010, 110.)

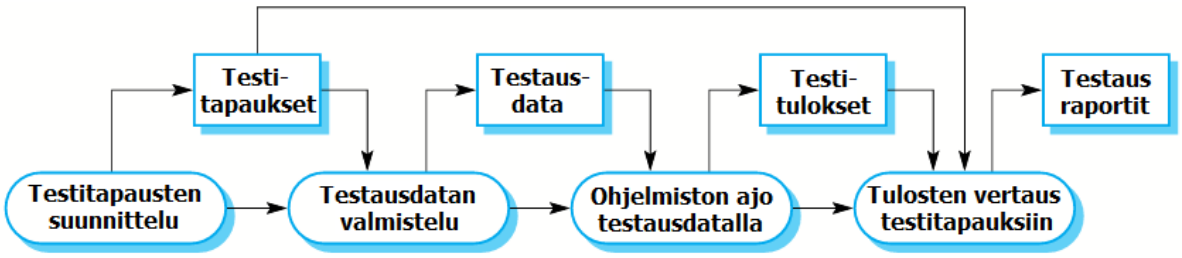
5.2 Kriittisten järjestelmien validointi

Kaikkien turvallisuuden kannalta kriittisten järjestelmien validoinnissa on paljon samaa, kuin muissa korkeaa luotettavuutta vaativissa järjestelmissä. Kriittisten järjestelmien validointi ja varmennusprosessissa tuleekin demonstroida, että järjestelmä vastaa määrittelyjä ja järjestelmän palvelut ja toiminta vastaavat asiakkaan vaatimuksia. Kriittisille järjestelmille tulee kuitenkin tehdä vieläkin tarkempaa testausta, jotta voidaan todistaa järjestelmän olevan luotettava. (Sommerville 2007, 567.) Turvallisuuden kannalta kriittisiä järjestelmiä testataankin tarkemmin kuin useimpia normaalikäyttöön suunniteltuja järjestelmiä (Homés 2012, 13).

Kriittisten järjestelmien turvallisuuden varmistamisessa päämäärä poikkeaa luotettavuuden varmistamisesta. Järjestelmän luotettavuutta voidaan arvioida erilaisilla mittareilla. Turvallisuutta ei voida kuitenkaan suoraan mitata millään konkreettisella mittarilla. Tämän vuoksi kriittisen järjestelmän turvallisuuden arvioinnissa käytetäänkin apuna mahdollisia todisteita järjestelmän toiminnallisuudesta. Usein järjestelmän turvallisuuden takeena on se, että järjestelmää kehittävä yritys on aikaisemmin kehittänyt samankaltaisia järjestelmiä, jotka ovat todettu turvallisiksi. (Sommerville 2007, 574.)

SCADA järjestelmät ovat teollisuuden ohjausjärjestelmiä, joita käytetään kriittisissä teollisuusympäristöissä, joissa virheet voivat johtaa hengenvaarallisiin tilanteisiin. Tällaisia teollisuuden ohjausjärjestelmien ohjaamia prosesseja ovat esimerkiksi kemikaalien tuotantoprosessit, joissa sekoitetaan kemikaaleja halutulla suhteella, tai veden lisääminen kattilaan oikeassa määrin. (Jaswal 2014, 158.)

5.3 Testausdokumentaatio



Kaavio 1. Ohjelmistotestausprosessin malli (Sommerville 2007, 539.)

Yllä olevassa kaaviossa 1 on esitetty tyypilliset ohjelmistotestauksen vaiheet. Ennen kuvassakin esiintyvää testitapausten suunnittelua tehdään testaussuunnitelma. Testaussuunnitelma sisältää kuvauksen testausprosessin päävaiheista, ohjelmistolle asetetut vaatimukset, testattavat asiat, testausaikataulun, tavan testauksen tulosten järjestelmälliseen tallennukseen, laitteisto ja ohjelmistovaatimukset sekä mahdolliset testausrajoitteet, kuten mahdolliset tulevat henkilöstövajeet testauksen aikana. Pienemmille järjestelmille voidaan käyttää vähemmän virallista testaussuunnitelmaa, mutta testaamiseen tarvitaan kuitenkin jonkinlainen virallinen dokumentti, jonka tehtävänä on tukea testauksen suunnittelua. (Sommerville 2007, 521.)

Testaussuunnitelma pysyy harvoin samanlaisena koko ohjelmiston kehityksen ajan. Testaussuunnitelmaa päivitetäänkin ohjelmistokehityksen aikana tapahtuvien muutosten mukaisesti koko ohjelmiston elinkaaren ajan. Testaussuunnitelmiin voi aiheuttaa muutoksia esimerkiksi viivytykset muissa tuotannon vaiheissa, jotka aiheuttavat viivästystä testauksen aloittamiselle. Tällöin suunnitelmaa joudutaan muuttamaan, jotta testaaja tietää milloin testaaminen on mahdollista aloittaa ja osaa siirtyä muihin tehtäviin alustavasti suunnitellun testaamisen sijaan. (Sommerville 2007, 521.)

Kun testaussuunnitelma on valmis, voidaan siirtyä itse testaamiseen. Testaaminen aloitetaan testitapausten suunnittelulla. Testitapauksella tarkoitetaan kokonaisuutta, joka sisältää sarjan syötettäviä arvoja, toteuttamisen edellytykset, odotetut tulokset ja toteuttamisen jälkiehdot (Homès 2012, 139). Eri ohjelmiston osia on helpompi testata eri vaiheessa ohjelmiston kehitystä, joten testitapaukset on hyvä määritellä jokaiselle tuotantovaiheelle erikseen, miettien juuri mitä asioita, milloinkin kannattaisi testata. Testitapaukset suunnittelemalla voidaan miettiä, miten eri ohjelmiston osa-alueita tullaan testaamaan. (Everett ja McLeod 2007, 83-84.)

Testitapausten laatimisen jälkeen valmistellaan testausdata. Normaalisti ohjelmoijat valmistelevat itse testidatan ohjelman testausta varten. Tämä on testidatan luomiselle loogisin lähestymistapa, sillä ohjelman suunnittelija tietää itse parhaiten, millaista dataa ohjelmaan tullaan syöttämään ja missä muodossa datan pitäisi olla. Ohjelmiston kehittäjä on myös paras henkilö luomaan testitapauksia kehittämälleen ohjelmistolle, koska ohjelmiston kehittäjä on itse suunnitellut, miten ohjelmiston tulisi toimia. (Sommerville 2007, 81.)

Testauksen jälkeen tehdään testiraportti. Testiraportti sisältää tiedon, mitä testejä tehtiin, mitkä niiden tulokset olivat ja antaa arvion ohjelman toiminnallisuudesta testien perusteella. Testiraportissa arvioidaan erikseen jokaisen järjestelmän osa-alueen toimintaa ja myös sitä, miten kattavasti ohjelmiston eri osa-alueet on testattu. Testausraportissa on myös uniikki tunniste, jonka avulla testausraportti voidaan eritellä, sekä listan henkilöistä, joiden tulisi hyväksyä testauksen tulokset esimerkiksi allekirjoittamalla testausraportin. (Homés 2012, 234.)

5.4 SCADA-järjestelmien testaaminen

Nykypäivän SCADA-järjestelmien koko ja monimutkaisuus voi vaihdella suuresti eri järjestelmien välillä. Tämän vuoksi järjestelmällinen lähestymistapa on paras SCADA-järjestelmien toiminnan varmistamiseksi. Testausta on hyvä suorittaa sekä sisäisesti SCADA-järjestelmän suunnitteluyrityksessä, että asiakkaan kanssa itse laitoksella. Suorittamalla huolellinen testaus jo ennen SCADA-järjestelmän toimittamista, voidaan säästää selvästi kustannuksissa vähentämällä laitoksella tapahtuvaa testausta. Yrityksen sisäisesti tekemä testaus lyhentää myös koko projektin valmistumisaikaa. SCADA-järjestelmän käyttöönottaminen on aina laitosprojektin viimeisten vaiheiden joukossa, joten SCADA-järjestelmää ei voida kokonaan testata ennen kuin laitoksen kaikki muut laitteistot on asennettu. Näin ollen, mitä lyhempi aika käyttöönoton yhteydessä tehtävä testaus on, sitä nopeammin koko projekti saadaan toimitettua. (McCrady 2013, 157-159.)

SCADA-järjestelmien ja automaatiojärjestelmien toiminnan varmistamiseksi asiakas vaatii usein tehtäväksi FATin (Factory Acceptance Test), eli tehdashyväksyntätestauksen, ennen järjestelmän toimitusta laitokselle. Tässä testauksessa järjestelmän toimintaa testataan oikeiden laitteiden sijasta simuloimalla automaation tuloliitäntöjen signaaleja. Tehdashyväksyntätestauksen seurauksena tehdään usein myös vielä muutoksia järjestelmän toiminnankuvaukseen, koska järjestelmän toiminnan hahmottaminen on helpompaa simuloitujen tuloarvojen avulla. Nämä FATin jälkeiset muutokset ovat yleensä sopimusehtojen ulkopuolella, mutta niiden toteuttaminen on silti välttämätöntä järjestelmän oikeaoppisen toiminnan takaamiseksi. (McCrady 2013, 27.)

Kun SCADA-järjestelmää käyttöönotetaan, on tärkeää testata, että järjestelmän toiminta vastaa sille asetettuja määrittelyjä. Tämä testaus suoritetaan lopullisesti joko itse laitoksella, johon järjestelmä tulee, tai ennen kuin se kuljetetaan laitokselle. Käytännössä testauksen tavoitteena on todentaa, että ohjelmiston logiikka toimii halutulla tavalla. (McCrady 2013, 38-39.) Lopputestauksessa on sitä parempi, mitä enemmän testausta saadaan suoritettua ennen järjestelmän toimitusta, sillä laitoksella järjestelmän testaamiseen joudutaan käyttämään tuotantoversiota. Järjestelmän käyttöön ottaminen onkin sitä jouhevampaa, mitä vähemmän sitä joudutaan testaamaan itse laitoksella. (McCrady 2013, 159.)

6 SIMULAATIO

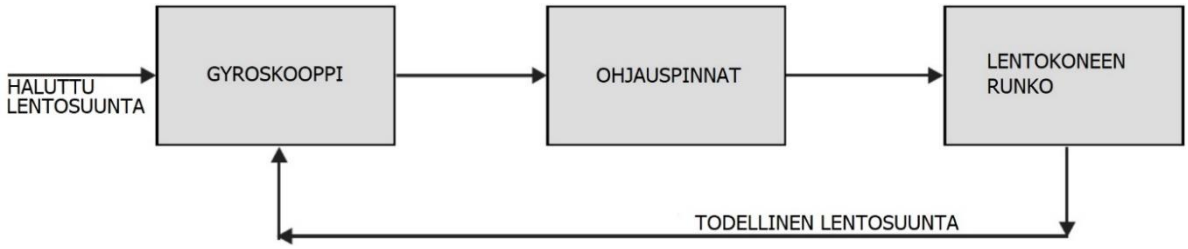
Simulaatiolla tarkoitetaan dynaamisten mallien käyttämistä kokeiden tekemiseksi, tiedon keräämiseksi tai jopa viihdemuotona. Simulaatio on kehittynyt ei-laskennallisesta simulaatiosta nykypäivän tietokoneistettuun simulaatioon. Ennen tietokoneita, simulaatio pohjautui ajatuskokeisiin ja pienoismallien avulla tehtäviin kokeisiin. (Tolk & Ören 2017, 12-13.)

Simuloinnissa jonkin reaalimaailman järjestelmän toimintaa pyritään mallintamaan mahdollisimman tarkasti. Kun puhutaan simuloinnista nykypäivänä, viitataan lähes aina tietokonesimulaatioon. Yksi määritelmä tietokonesimulaatiolle on, että järjestelmän dynaamista käyttäytymistä kuvastetaan siirtämällä se tilasta toiseen hyvin määriteltyjen käyttöohjeiden mukaisesti. (Raczynski 2006, 1.)

Järjestelmällä tarkoitetaan jotain asiaa tai esinettä, joka suorittaa jonkin toiminnon. Järjestelmä voidaan jakaa pienempiin osioihin, joilla kaikilla on oma toimintonsa, joka edesauttaa koko järjestelmän toimintaa. Jokaiselle järjestelmän osiolle voidaan määritellä niiden ominaisuudet, joitten perusteella niiden kykyä suoriutua voidaan mitata. Esimerkkinä järjestelmästä voisi olla autopilotilla lentävä lentokone. Tämän järjestelmän osioina ovat gyroskooppi, lentokoneen runko ja lentokoneen ohjauspinnat. Näiden järjestelmän osioiden ominaisuudet ovat gyroskoopin asetukset, nopeus ja ohjauspintojen kulmat. Järjestelmän toiminto kokonaisuudessaan on lentäminen. (Singh 2008, 1-2.)

Joskus järjestelmän toiminta riippuu ympäristöstä. Jos kyseessä on tällainen järjestelmä, kutsutaan järjestelmää eksogeeniseksi. Jos taas järjestelmän toiminta ei riipu ympäristötekijöistä ollenkaan, sitä kutsutaan endogeeniseksi. Lentokoneen lentäminen on eksogeeninen järjestelmä, sillä sen lentämiseen vaikuttavat ulkoiset tekijät, kuten sää. Staattinen malli lentokoneesta puolestaan on endogeeninen. (Singh 2008, 2.)

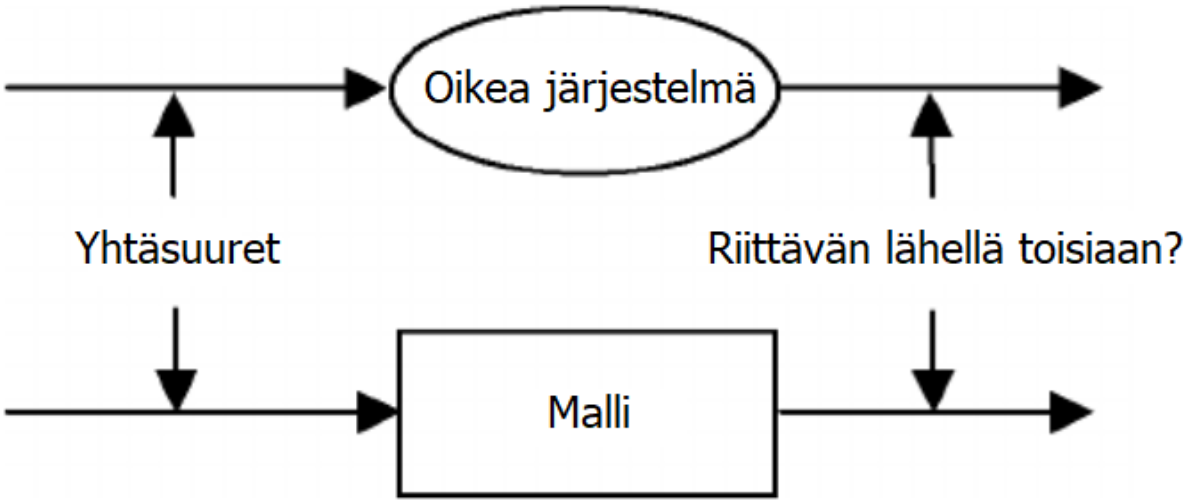
Ensimmäinen askel simulaatiossa on rakentaa järjestelmästä malli. Mallissa joudutaan aina tekemään kompromisseja, sillä malli on aina yksinkertaisempi kuin reaalimaailman järjestelmä, jota sillä kuvataan. Kun simulaatiomallia rakennetaan, valitaan järjestelmän osioiden ominaisuudet, eli muuttujat, jotka siinä otetaan huomioon sen perusteella, mitä mallin avulla halutaan tutkia. (Raczynski 2006, 2-3.)



Kuva 3. Autopilotilla lentävästä lentokoneesta tehty simulaatiomalli (Singh 2008, 4).

Kuva 3 on mallinnettu autopilotilla lentävää lentokonetta. Malli on ajatuksen tasolla yksinkertainen, sillä kyseinen järjestelmä voidaan jakaa kuvan mukaisiin kolmeen osioon. Gyroskooppi havainnoi eron halutun lentosuunnan ja todellisen lentosuunnan välillä. Sen jälkeen gyroskooppi välittää tämän tiedon pohjalta käskyn lentopinnoille, jotka ohjaavat lentokoneen rungon oikeaan lentosuuntaan. (Singh 2008, 4.)

Simuloinnissa ja mallinnuksessa mallien pätevyys on yksi keskeisimmistä ongelmista. Jos simulaatiomallin pätevyydestä ei voida olla varmoja, ovat simulaation avulla saatavien tulosten luotettavuuskin kyseenalainen. Simulaatiossa pyritään riittävään tarkkuuteen, tällä tarkoitetaan reaalimaailman järjestelmän tulosten ja simulaation tulosten olevan tarpeeksi lähellä toisiaan. Tämä riittävä tarkkuus on kuitenkin mallintajan päätettävissä. Jotta simulaatiomallin tulokset olisivat vertailukelpoisia, täytyy simulaation tuottaa pitkälläkin aikavälillä samankaltaisia tuloksia kuin reaalimaailman järjestelmä tuottaisi samassa tilanteessa. Tätä voidaan tutkia vertaamalla historiadataa sekä oikeasta järjestelmästä, että simuloidusta järjestelmästä. (Raczynski 2006, 3-5.) Tätä simulaatiomallin pätevyyden tarkastelua havainnollistetaan alla olevassa Kuva 4.



Kuva 4. Mallin pätevyyden tarkastelu tulevien ja lähtevien arvojen perusteella (Raczynski 2006, 5).

7 TESTAUSKÄYTÄNTÖJEN RAKENTAMINEN JA KEHITTÄMINEN

Testauskäytäntöjen kehittäminen oli kaksiosainen projekti. Kehitystyön tavoitteena oli luoda yhtenäinen testausdokumentaatio, jota kyettäisiin soveltamaan kaikkiin tuleviin valvomoprojekteihin. Toisena osana oli luoda testausympäristö, jossa valvomojärjestelmiä pystytään testaamaan vapaamuotoisesti ilman vaaraa, että tuotantoon menevä versio voisi mennä sekaisin. Testausympäristön kehityksen mukana selvitettiin myös erilaisia mahdollisuuksia hyödyntää automaattitestaustyökaluja ja simulaatiota osana testausympäristöä testausprosessin helpottamiseksi.

7.1 Testausympäristön ja -dokumentin suunnittelu

Testausympäristön suunnittelussa oli tärkeää huomioida jo käynnissä olevat laitokset. Tämän vuoksi todettiin testausympäristön toteuttamisen olevan suotuisampaa paikallisesti, yhdistämättä testausympäristöä portaaliin, jossa todelliset projektit sijaitsevat. Tällä varmennetaan se, että testausvaiheessa ei pääse vahingossakaan sotkemaan valmiita projekteja, vaan kaikki testausta varten tehdyt muutokset tilatietoihin tai mittausarvoihin ovat vain testaajan omalla koneella.

Testausympäristöä suunnitellessa todettiin simuloinnin olevan opinnäytetyön aikatauluun nähden liian laaja kokonaisuus. Simulointia varten olisi täytynyt keksiä tapa simuloida logiikan toimintaa virtuaalisesti, koska hälytys ja mittaustiedot tulevat logiikan tietokantapalvelimelta ja tälle palvelimelle olisi erittäin hankalaa luoda järkevää dataa ilman simulaation hyödyntämistä. Tämä simulointi veisi valtavasti aikaa jo yhden projektin testaamisessa, joten tämä vaihtoehto jätettiin toteuttamatta osana opinnäytetyötä

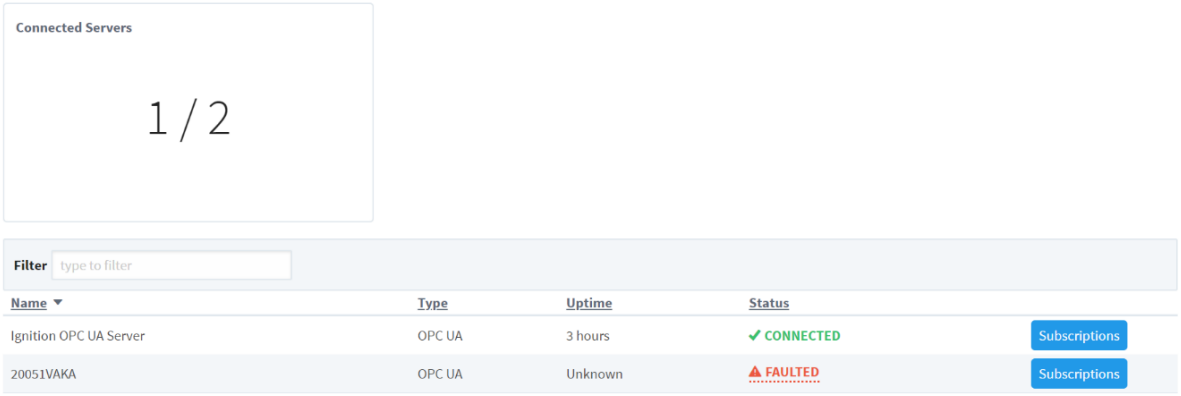
Testausympäristön suunnittelun lisäksi suoritettiin myös päivitetyn testausdokumentaation suunnittelu. Uuden testausdokumentaation tavoitteena on luoda selkeä yleinen testauskäytäntö, jota kyettäisiin soveltamaan kaikkiin testausympäristössä testattaviin projekteihin. Testausdokumentaation suunnittelussa päädyttiin tekemään testausdokumentaatiosta kaksiosainen, joista ensimmäinen osio sisältää välttämättömät testit, jotka ovat määrä minimissään suorittaa jokaiselle valvomoprojektille. Toiseen osioon puolestaan tulisi laajempaa testausta, jota voitaisiin toteuttaa, jos testaamiseen jää ylimääräistä aikaa välttämättömän testauksen jälkeen.

Testausdokumentin suunnittelussa tärkeää oli saada siitä mahdollisimman pitkälle sovellettavissa oleva jokaiseen projektiin. Tällöin uuden projektin testausta suunnitellessa voidaan noudattaa testausdokumentin valmista pohjaa. Dokumenttiin tarvitseekin vain lisätä tai poistaa tarvittaessa kohtia, kun luodaan uuden projektin testausdokumenttia.

Testausdokumentin on myös määrä toimia apuna suunnittelussa, sillä testausdokumentin suunnitelma osiossa listataan kaikki yleisimmät operaattorin näytöt eri kattilatyypeille. Lisäksi testaussuunnitelmaan lisätään tarpeen mukaan projektikohtaisia lisänäyttöjä. Testausdokumentti onkin suunniteltu siten, että se on määrä alustavasti täyttää jo projektin alussa samalla, kun koko projektin aikataulutusta suunnitellaan, jotta testausdokumenttiin kyetään täyttämään teknisessä spesifikaatiossa määritellyt näytöt ja toiminnallisuudet.

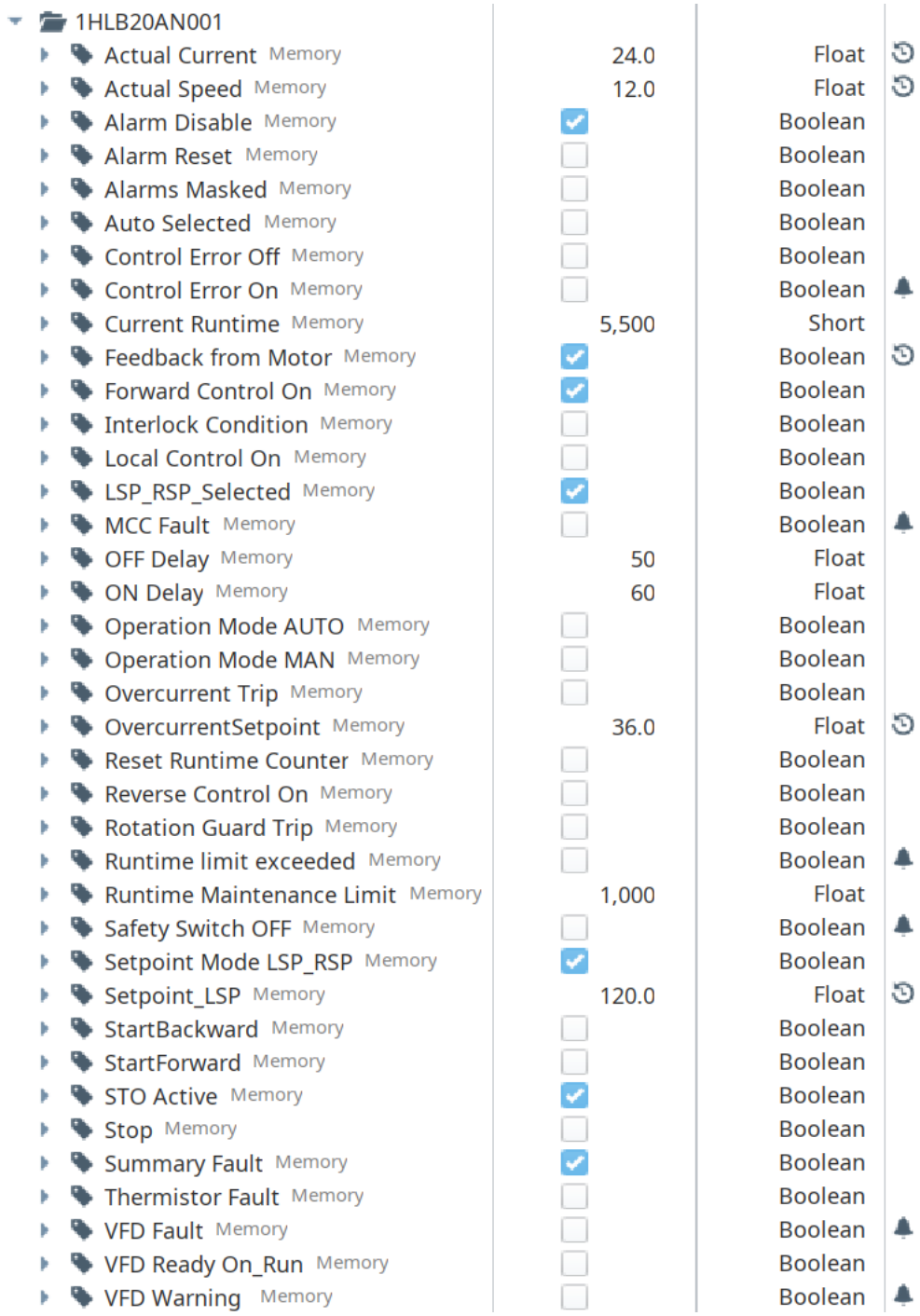
7.2 Testausympäristön rakentaminen

Testausympäristön rakentaminen aloitettiin tuomalla keskeneräisen laitosprojektin valvomonäytöt Ignition-portaalista paikalliseksi versioksi työasemalle. Tämä tapahtui ottamalla oikeasta projektista varmuuskopio, joka tuotiin localhost-portaaliin. Localhost-portaalilla tarkoitetaan portaalia, johon ei voida yhdistää internetin välityksellä, vaan siihen saadaan yhteys vain siltä koneelta, jolla se sijaitsee.



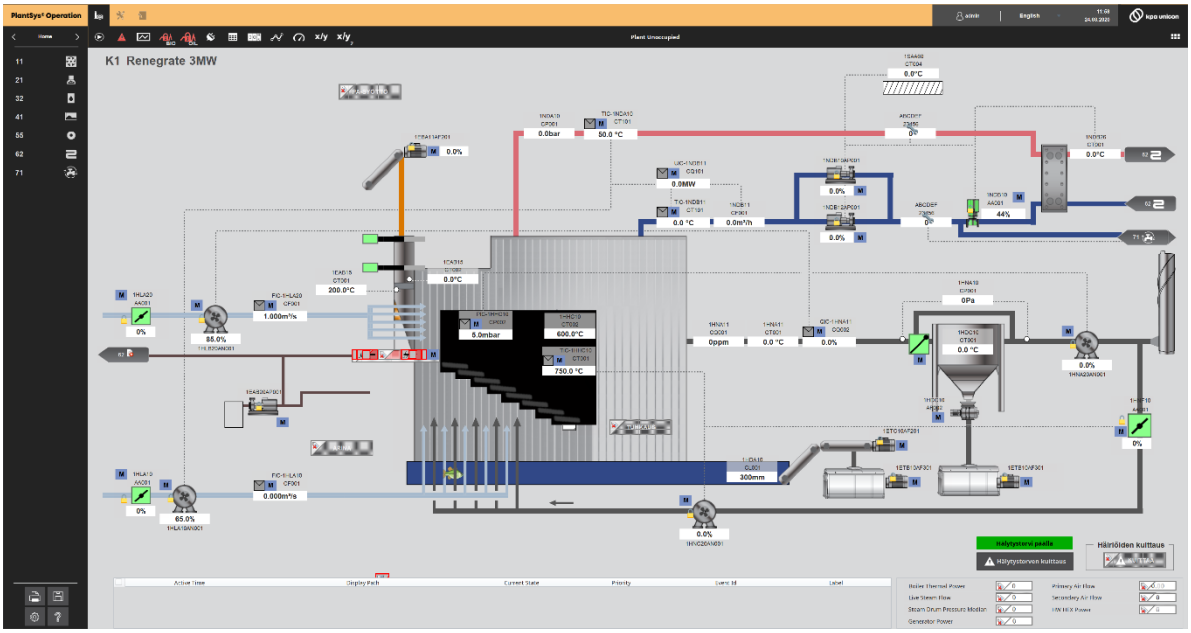
Kuva 5. Ignitionin OPC UA -yhteydet. (Kuvankaappaus testausympäristön ”Ignition Gateway” -webportaalista)

Kuten Kuva 5 voidaan nähdä, ei ole yhteyttä oikean projektin automaation tietokantapalvelimeen. Tämän voi nähdä siitä, että projektin ”20051VAKA” OPC UA palvelimen statustietona on faulted. Koska yhteyttä automaatiojärjestelmään ei ole, ei testausympäristöön myöskään saada reaaliaikaisia mittaus- ja hälytystietoja. Kuitenkin automaatiojärjestelmän tagilista vietiin oikeasta projektista ja tuotiin paikalliseen projektiin muistitageina, jotka sisälsivät vientihetkellä OPC UA palvelimella olevat mittausarvot ja tilatiedot.



Kuva 6. Yhtä puhallinta varten PLC:llä olevat tagit. (Kuva otettu Ignition Designer -sovelluksesta testausympäristöstä.)

Näin ollen testausympäristössä kyetään muokkaamaan manuaalisesti tagien tietoja, jolloin voidaan nähdä, ovatko tagit kytketty oikein valvomonäytöillä oleviin laitteisiin. Kuten Kuva 6 voidaan kuitenkin havaita, on jopa yhdellä laitteella olemassa lukuisia tageja, jolloin jo yhden laitteen läpikohtainen testaus voi viedä runsaasti aikaa. Kuva 6 laite on sekundääri-ilmapuhallin, joka on biokattilan palamisprosessin kannalta kriittinen laite, joten on luonnollista, että sen toimintaa ohjataan ja seurataan tarkasti useilla eri tageilla, kuten lukuisilla eri vikailmoitustageilla. Kuvasta nähdään myös, että käytössä olevat tagit ovat memory, eli muistitageja, jolloin ne ovat vain staattisia arvoja, jotka olivat PLC:n tietokantapalvelimen muistissa tagien tallennushetkellä.



Kuva 7. Keskeneräinen PlantSys -valvomonäyttö viistoarinan palamisprosessin valvomiseen. (Kuva otettu testausympäristöstä)

Koska laitos ei ole vielä käynnissä, ei mittareilla tai laitteilla ole oikeita tilatietoja, vaan tiedot ovat tällä hetkellä testausvaiheessa erikseen syötetty eri laitteille. Kuten Kuva 7 voidaan havaita, monta laitetta ja mittauspistettä on myös vielä ilman tagitietoja tai positiota. Tämä johtuu siitä, että niitä ei ole vielä määritetty automaatiojärjestelmään, tai prosessikaavioihin. Niistä onkin vain tieto, että ne saatetaan olla lisäämässä.

Kaikkien laitteiden ja hälytysten toiminnan testaaminen erikseen manuaalisesti on kuitenkin työlästä. Vaihtoehtoina onkin tehdä alustava testaussuunnitelma, jossa testataan pistokoemaisesti satunnaisten laitteiden toimintaa tarkemmin, sekä silmäillään kaikki valvomonäytöt läpi erikseen selkeiden virheiden varalta. Jos testaamiseen on kuitenkin tarjolla enemmän resursseja, voidaan suorittaa tarkempaa testausta. Tähän testaukseen sisällytetään esimerkiksi tagilistan muokkaaminen massana muuttamalla esimerkiksi kaikki hälytystagit samaan tilaan. Kun muutetaan kaikki nämä tagit yhtaikaisesti true-tilaan, voidaan tarkastella valvomonäytöt lävitse ja todeta kaikkien laitteiden olevan vikatilassa, jos laitteiden hälytykset näkyvät oikein SCADA-järjestelmässä.

Testausympäristön avulla ei kuitenkaan kyetä testaamaan tarkemmin hälytysten toiminnallisuutta. Esimerkiksi jos jonkin hälytyksen tulisi aktivoitua, kun vaikkapa lämpötila ylittää sille asetetun raja-arvon. Koska testausympäristöllä ei ole yhteyttä OPC UA-palvelimeen, ei hälytysten aktivoitumista voida kuitenkaan testata. Esimerkiksi, jos asetetaan testausympäristössä jonkin mittauksen arvo suuremmaksi kuin sen raja-arvo olisi tulisi oikeasti järjestelmän antaa ylärajahälytys kyseiseltä anturilta. Testausympäristössä tätä ei kuitenkaan tapahdu, koska hälytykset käsitellään ohjelmoitavan logiikan ohjelmistossa, eikä SCADA-järjestelmässä.

Kuva 8. Hälytysten testaaminen Python-ohjelmalla (Kuva otettu testausympäristöstä ja Pythonin integroidusta kehitysympäristöstä.)

Hälytysten testaamiseen kehitettiin Kuva 9 mukainen lyhyt Python-ohjelmointikielellä kirjoitettu koodipätkä. Koodin avulla voidaan Ignitionista ulos vietyyn .json tiedostomuotoiseen tagilistaan muokata kaikkien hälytysten tilaksi joko true tai false sen mukaan, halutaanko hälytykset asettaa päälle vai pois. Kun hälytykset aktivoidaan tällä metodilla, antaa Python-ohjelma tiedon, montako hälytystagia ohjelma muokkasi. Kun tämä tiedetään, voidaan tarkastella valvomon hälytyslokissa olevaa hälytysten määrää ohjelman muokkaamiin hälytyksiin. Jos määrät ovat samat, voidaan todeta kaikkien hälytysten toimivan valvomo-ohjelmistossa. Kuva 8 esitetään, miten hälytysten määriä voidaan verrata. Valvomon hälytyslokista valitaan kaikki aktiiviset hälytykset ja verrataan niitä Python-ohjelman muokkaamien hälytysten määrään. Kuva 8 tapauksessa määrät ovat samat, jolloin voidaan todeta testausympäristössä olevan projektin hälytysten näkyvän oikein valvomon hälytysnäytöllä. Samalla kun kaikki hälytykset aktivoidaan, voidaan silmäillä hälytykset läpi mahdollisten kirjoitusvirheiden varalta hälytysten nimissä.

```
import json

#Change these
input_json = "input.json" #the file you want to modify
output_json = "output.json" #name of the modified file

# input file
with open(input_json) as f:
    data = json.load(f)

#print(data["tags"][0]["tags"])

# Get PLC Tags
# 4001, 4002,....
tags_1 = data["tags"][0]["tags"]



# Iterate PLC Tags
alarm_count = 0
for folder in tags_1:
    # Get eg 4001 tags, ....
    try:
        tags_2 = folder["tags"]
        for folder2 in tags_2:
        # Get 1EAB10AA101... tags
            tags_3 = folder2["tags"]
            for memory in tags_3:
                alarm = memory.get("alarms")
                if alarm != None:
                        memory["value"] = False #Change this to True/False to enable/disable alarms
                        alarm_count += 1
    except KeyError: #This takes into account the alarms, that are not under device folders such as 1EAB10AA101 etc.
        for memory2 in tags_2:
            alarm2 = memory2.get("alarms")
            if alarm2 != None:
                memory2["value"] = False #Also change this value to True/False
                alarm_count += 1



# output file
with open(output_json,"w") as f:
    json.dump(data, f, indent=2, ensure_ascii=False)
print("JSON dumped, the number of alarms modified = ", alarm_count)
```

Kuva 9. Hälytysten testaamiseen kirjoitettu Python-koodi (Kuva otettu Pythonin integroidusta kehitysympäristöstä.)

Testausympäristöä voidaan hyödyntää myös jo valmiiden projektien päivittämisessä uuteen ohjelmistoversioon. Kun Ignition-projektin versiota päivitetään voi ilmentyä rikkinäisiä graafisia elementtejä. Tämä johtuu siitä, että nämä kyseiset komponentit sisältävät rikkinäisiä linkityksiä. Nämä rikkinäiset komponentit eivät näkyneet aiemmissa Ignition-ohjelmistoversioissa, mutta uusimmassa versiossa ne ovat näkyvissä. Vialliset komponentit ovat nopeita korjata, mutta jos päivitetään tuotannossa olevaa valvomoa, ovat ne parempi korjata testausympäristössä. Tällöin nämä komponentit eivät aiheuta hämmennystä valvomoa käyttävälle operaattorille.

7.3 Testausdokumentin rakenne ja käyttö

Opinnäytetyön aikana tuotettiin myös testausdokumentti, jota voidaan käyttää testaamisen ohjeistamisessa ja avustamisessa. Testausdokumentissa testausmetodeja ja käytäntöjä on jaettu useampaan osioon. Testauksessa on pakollinen osio, joka sisältää vähittäisvaatimukset testaukselle, jota jokaiselle projektille tulisi tehdä. Pakollinen osio on jaettu vielä kahteen osioon yleisiin, sekä projektikohtaisiin testeihin.

Testausdokumentaatiossa päädyttiin ratkaisuun, jossa kaikki testauksen vaiheet ovat tiivistetty yhteen dokumenttiin testauksen helpottamiseksi. Koska testattavana ei ole suoranaisesti ohjelmisto, joka olisi kehitetty alusta asti koodaamalla, vaan ohjelmisto on tehty Ignitionin valvomoiden suunnitteluun tarkoitetulla Vision moduulilla. Ohjelmiston testausta varten tehtyyn dokumentaatioon onkin tiivistettynä yleistiedot, testaussuunnitelma, testausaikataulu, testausyhteenveto sekä testaustulokset.

Ohjelmistot, joita PlantSys -valvomon rakentamiseen tarvitaan, ovat Ignition, KepserverEX ja MySQL. Laitteistoihin kuuluvat yleisesti palvelin, jolla tietokanta sijaitsee, valvomotietokone, jolla ohjataan prosessia sekä Tosibox lukko, jolla voidaan muodostaa tarvittaessa salattu etäyhteys laitoksen valvomotietokoneelle tai palvelimelle. Kuva 10 on esimerkki mahdollisesta laitteistosta, jolla PlantSys SCADA -järjestelmä voitaisiin toteuttaa.

1.1. Hardware setup

Server type	Dell PowerEdge R340
Total memory	16 GB UDIMM ECC
Total disk space	5x 600 GB
Workstation	Lenovo M720Q TINY
Monitors	1x Voxicon G32QHD IPS
Monitor connectivity	DisplayPort, HDMI 1.4, VGA, 4-port USB 3.0 HUB
Network backup storage	Buffalo Terastation 1200 2x 1 TB
Connectivity	DELL EMC DNN1124T Tosibox Lock 200

Kuva 10. Esimerkki tyypillisestä PlantSys -laitteistosta. (Kuva testausdokumentin yleistieto-osiosta.)

Suuremmissa laitoksissa käytössä voi olla useampia valvomotietokoneita, jos valvomoon halutaan enemmän näyttöjä, kuin mitkä toimisivat yhdellä tietokoneella. Pienemmissä laitoksissa puolestaan palvelin ja operaattorin työasema voidaan yhdistää, eli operaattorin työasema toimii samalla myös PlantSys SCADA-palvelimena. Järjestelmien toiminnan varmistamiseksi voidaan myös asentaa kaksi palvelinta rinnakkain, jolloin toinen palvelimista on redundanttinen. Tällöin, jos toinen palvelimista hajoaa, voidaan laitosta ajaa yhden palvelimen varassa.

Testaussuunnitelmassa eritellään yleisesti testauksen tavoitteet, testausmetodit sekä testauksen läpäisykriteerit. Testausmetodeissa eritellään myös eri kattilatyyppien tyypilliset valvomoikkunat, sekä mahdolliset lisäikkunat, jotka ovat projektikohtaisia. Tällöin testaussuunnitelmaa pystytään samalla käyttämään apuna testauksessa, kun voidaan tarkastaa tästä listauksesta kaikkien vaadittujen valvomonäyttöjen olevan mukana valvomoprojektissa. Testaussuunnitelman mukaisesti testaus on hyväksytysti läpäisty, kun kaikki pakolliset testit on suoritettu ja kaikki testauksessa löydetyt virheet ja viat on korjattu ja korjausten jälkeen toiminta on varmennettu.

Testausdokumentin ja erityisesti testausaikataulun on tarkoitus olla projektin selkäranka. Kun projekti pistetään alulle, samalla luodaan myös projektille testidokumentti. Tällöin kirjataan testausdokumenttiin päivämäärä, jona dokumentti on luotu. Samalla dokumenttiin lisätään kaikki projektikohtaiset lisäykset ja tehdään alustava aikataulu koko projektin testaamiselle. Testauksen aikatauluttaminen projektin etenemisen kanssa on tärkeää, sillä testaamista ei kannata aloittaa ennen, kuin valvomo on tarpeeksi lähellä lopullista toteutusta. Jos testaus aloitetaan liian aikaisin, ilmenee testauksessa liikaa sellaisia vikoja, jotka johtuvat ainoastaan siitä, että suunnittelija ei ole ehtinyt vielä esimerkiksi aloittaa kaikkien näyttöjen rakentamista tai siitä, että automaatiosta ei ole vielä annettu tarpeeksi tietoja valvomon suunnittelijalle.

3. Test schedule

Testing timetable

Testing phase	Date	Signature
Creating the preliminary test plan and test schedule	1.3.2020	Antti Pykäläinen
Creating the testing environment for the project	20.3.2020	Antti Pykäläinen
First round of mandatory (and advanced) tests	20.3.2020	Antti Pykäläinen
Fixing and testing defects of first testing	27.3.2020	Antti Pykäläinen
Updating the testing environment to match the latest project version		
Final tests before FAT		
Fixing and testing defects of final tests		

Kuva 11. Esimerkki testausdokumentin aikatauluosion täydentämisestä

Kuva 11 on mallinnettu, miten aikataulua tultaisiin täyttämään oikean projektin edetessä. Aikataulun päivittäisikin aina kyseisen vaiheen testauksen tehnyt henkilö. Ensimmäisessä vaiheessa, eli dokumentaation luomisessa on mukana useampia henkilöitä, joten silloin voitaisiin merkitä kaikkien dokumentin laadinnassa mukana olleiden nimet dokumenttiin. Testausympäristön päivittäminen on olennaista uusia testauskertoja varten, koska suunnittelijan on turhaa tehdä testausympäristössä löytyneiden virheiden korjauksia testausympäristössä. On parempi korjata löytyneet virheet oikeassa projektissa ja jatkaa projektia eteenpäin niin pitkään, kunnes on järkevää testata valvomoa uudestaan.

4. Test summary (mandatory)

4.1. General mandatory testing

	Test ID	Definition	Results	Signature & date
General tests	4.1.1	Check all screens for correct language	Pass	Antti Pykäläinen 27.3.2020
	4.1.2	Check proper function of menus	Pass	Antti Pykäläinen 27.3.2020
	4.1.3	Check if pens function properly	Pass	Antti Pykäläinen 20.3.2020
	4.1.4	Visual confirmation of all operator screens	Fail	Antti Pykäläinen 20.3.2020

Kuva 12. Esimerkki testausdokumentin testausyhteenvedon käytöstä

Testausdokumenttiin täytetään yleiset testattavat asiat Kuva 12 mukaiseen taulukkoon. Jos näissä testeissä ilmenee joitain vikoja tai virheitä, eritellään ne erilliseen testaustulokset taulukkoon. Testaustulokset taulukossa on osio myös virheiden korjaamiseen. Tähän osioon voidaan merkata tapa, jolla virhe korjattiin ja päivämäärä, jolloin virhe on korjattu.

Testiyhteenvedossa on myös osio niin kutsutuille edistyneemmälle testaukselle, jolla tarkoitetaan testausta, joka on aikaa vievämpää, kuin pakollinen testaus ja jota voidaan suorittaa, jos testaajalle jää aikaa tehdä tarkempaa testausta. Tälle testaukselle ei varata erikseen aikaa, sillä pakollinen testaus suunnitellaan tarpeeksi kattavaksi valvomon toiminnan varmistamiseksi. Kuten teoriaosuudessakin mainitaan, liika testaaminen ei ole myöskään hyvästä, koska sillä saavutettava hyöty on pienempi kuin siihen kuluvat resurssit.

6. Test results (mandatory)

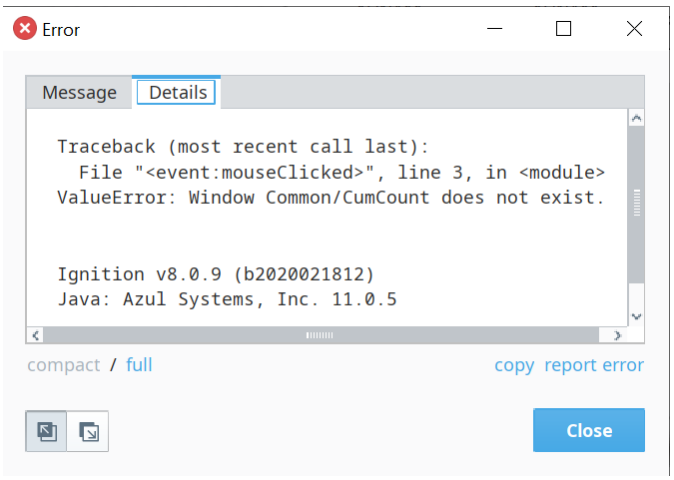
6.1. General system testing

	Defect ID	Severity of Defect	Summary of Defect	Signature & Date	Summary of Fix	Re-tested by & Date
General tests	1-4.1.1	Major	Wrong language in a device's name (Screen 21, 1HNC20AN001)	Antti Pykäläinen 20.3.2020	Translated device name to English	Antti Pykäläinen 27.3.2020
	2-4.1.2	Major	Clicking Cumulative Counters menu icon returns an error	Antti Pykäläinen 20.3.2020	Added the cumulative counters window to the project	Antti Pykäläinen 27.3.2020
	3-4.1.4	Major	Link from screen 21 to 61 does not work	Antti Pykäläinen 20.3.2020	Fixed the link by correcting a typo	Antti Pykäläinen 27.3.2020
	4-4.1.4	Minor	Graphical error in a pipeline (Screen 21, top-right)	Antti Pykäläinen 20.3.2020		

Kuva 13. Esimerkki testausdokumentin testaustulokset -osiosta mallivirheineen

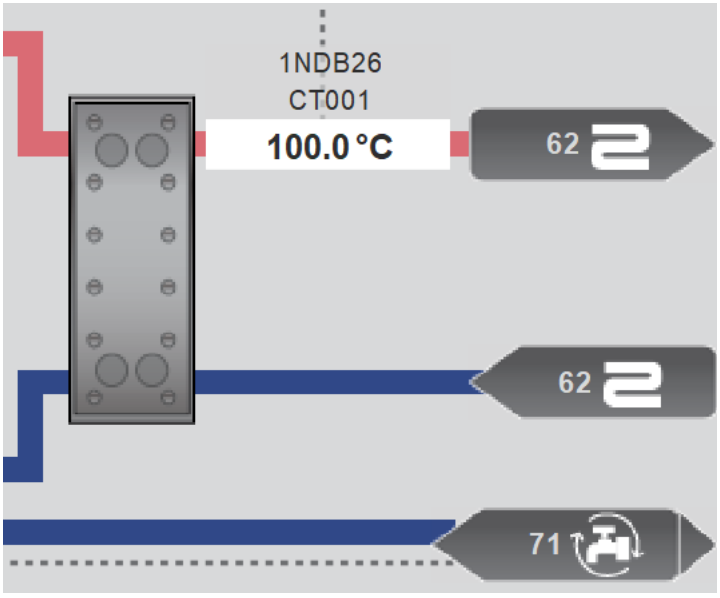
Kuva 13 esitellään puolestaan testauksen tulososion käyttämistä. Kuten jo aiemmin mainittu, testaustulokset -osion tarkoituksena on kasata kaikki testauksessa ilmenneet virheet. Testausyhteenvedossa kohta 4.1.4, eli valvomonäyttöjen visuaalinen tarkastus on merkattu esimerkkitapauksessa hylätyksi, koska siitä on merkattu testaustuloksiin virhe. Koska tätä virhettä ei ole vielä merkattu korjatuksi, pysyy kyseinen testi hylättynä. Kun tämä virhe korjataan, merkkaa virheen korjaaja ja testaaja samalla myös testiyhteenvetoon kyseisen testin läpäistyksi

Testitulokset -taulukkoon luokitellaan jokaiselle löydetylle virheelle sen vakavuus. Tämän tarkoituksena on auttaa suunnittelijaa priorisoimaan virheitä, jotka ovat haitallisimpia valvomokokonaisuuden kannalta. Nämä luokat ovat critical, major ja minor. Critical luokituksen saavat virheet, jotka estävät valvomon toiminnan, tai voivat aiheuttaa merkittävän turvallisuusriskin. Major luokituksen saavat virheet, jotka huonontavat asiakkaan käyttökokemusta tai ovat selkeästi havaittavissa. Minor luokittelun saavat virheet, jotka eivät vaikuta ohjelmiston käyttöön ja joita ei välttämättä edes huomaa, ilman tarkempaa tarkastelua.



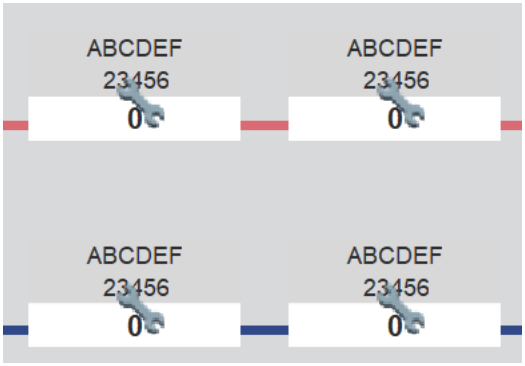
Kuva 14. Cumulative counters ikkunan avaamisesta tuleva virheilmoitus. (Kuva otettu testausympäristöstä.)

Testaustuloksissa ilmennyt esimerkkivirhe, jonka Defect ID on 2-4.1.2, on havainnollistettu Kuva 14. Tämä virheilmoitus johtuu siitä, että projektissa ei yksinkertaisesti ole vielä kyseistä ikkunaa tai painikkeeseen asetettu tiedostopolku ei ole oikea. Testattavana olevassa projektissa tilanne oli ensimmäinen, eli kyseistä valvomoikkunaa ei vielä ollut lisätty valvomoprojektiin. Hieman vastaavanlaisesti testaustuloksissa on listattu vika 3-4.1.4, jossa linkki kahden eri ikkunan välillä ei toimi. Tässä tapauksessa kumpikin ikkuna oli jo projektissa, mutta linkitys ikkunoiden välillä oli viallinen kirjoitusvirheen vuoksi.



Kuva 15. Testaustuloksissa kuvattu graafinen virhe 4-4.1.4. (Kuva otettu testausympäristöstä.)

Kuva 15 havainnollistetaan aikaisemmassa Kuva 13 mainittua graafista virhettä. Kuten voidaan nähdä, ei alimman putken kohdistus ole aivan keskellä. Tämän näkee siitä, kun vertaa alinta putkea kahteen ylempään. Virhe on järjestelmän toiminnan kannalta aika mitätön, eikä sitä myöskään huomaa kovinkaan helposti, kun valvomonäyttöä katselee kokonaisena, eikä tällä tavoin, kuin Kuva 15 on tehty. Tämän vuoksi virhe onkin saanut minor luokituksen.



Kuva 16. Positiotieto puuttuu mittauksesta. (Kuva otettu testausympäristöstä.)

Kun projekteja testataan samalla, kun ne ovat vielä suunnitteluvaiheessa, on tärkeää huomioida joidenkin näyttöjen olevan vielä keskeneräisiä. Esimerkiksi Kuva 16 on mittauksia, joiden olemassaolosta ei ole vielä varmuutta. Nämä mittaukset tulevat joko saamaan position ja tiedot logiikalta, tai ne tullaan poistamaan ylimääräisinä. Tällaisia mittauksia ei olisi järkevää merkitä testausraporttiin, koska suunnittelija tietää jo valmiiksi niiden olevan keskeneräisiä. Jos niitä alkaisi merkitsemään, tuottaisi se turhaa työtä sekä testaajalle, että suunnittelijalle, koska varhaisessa vaiheessa projektilla voi olla ilman positiotunnusta olevia laitteita runsaasti. Positioinnilla tarkoitetaan sitä, kun prosessisuunnittelussa jokaiselle laitteelle annetaan yksilöllinen numeroista ja kirjaimista koostuva tunniste.

Kun projekti on valmis, viimeistellään myös testausdokumentaatio. Valmis testausdokumentaatio toimitetaan muiden projektiasiakirjojen lomassa loppuasiakkaalle. Toimittamalla kattava testausdokumentaatio projektista, lisätään asiakkaan luottoa valvomojärjestelmän toimivuuteen. Samalla asiakas saa ammattitaitoisemman vaikutelman projektin suunnittelu- ja toteutusprosessista.

8 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella tilaajalle uusi toimiva testausdokumentti, joka tukisi koko valvomoprojektia myös suunnittelun osalta. Lisäksi työssä oli määrä luoda testausympäristö ja hyödyntää sitä osana testausprosessia. Testausdokumentin suunnittelu tapahtui tiiviissä yhteistyössä tilaajan kanssa, jotta testausdokumentti vastaisi mahdollisimman hyvin tilaajan tarvetta. Dokumentin luonnissa konsultoitiin myös valvomoiden testausta suorittaneita henkilöitä, joka omalta osaltaan konkretisoi testausta, jota valvomonäytöille yleensä tehdään.

Työn aikana perehdyttiin teoreettisesti simuloinnin hyödyntämiseen testauksessa. Simulointi osoittautui kuitenkin käytännössä erittäin aikaa vieväksi, joten sen käyttäminen testausympäristön kanssa testauksen tehostamiseksi todettiin opinnäytetyön rajallisen aikataulun vuoksi kannattamattomaksi. Simuloinnissa olisi kuitenkin selviä hyötyjä. Esimerkiksi simuloimalla automaatiojärjestelmää, voitaisiin paremmin todentaa hälytysten ja mittarien toimintaa.

Testauksen automatisointiin löytyi lopulta valitettavan vähän uusia apukeinoja. Monet automaattitestauksen työkalut vaativat enemmän ohjelmointiosaamista, mitä opinnäytetyön tekijällä olisi ollut. Toisaalta useat valmiit automaattitestausvälineet ovat myös melko hintavia, joka puolestaan hankaloitti niiden hyödyntämistä mahdollisina apuvälineinä testaamisessa. SCADA järjestelmien testaukseen liittyvässä kirjallisuudessa keskityttiin enimmäkseen SCADA-järjestelmien tietoturvan testaamiseen, esimerkiksi tietokantamurtojen osalta, eikä niinkään itse järjestelmän toiminnan testaamiseen.

Opinnäytetyö on kokonaisuutena onnistunut. Opinnäytetyön tuloksena syntynyt testausdokumentti on hyödyllinen apuväline projektien testauksen aikatauluttamiseen. Lisäksi testausdokumentin avulla on helppo pitää kirjaa siitä, mitä on testattu ja mitä on vielä testaamatta. Testausdokumentin avulla valvomon suunnittelija voi myös tarkastaa, onko kaikki projektissa suunnitellut näytöt jo osana projektia ja ovatko ne valmiina.

Automaattitestaukseen opinnäytetyön aikana saatiin kehitettyä valitettavan vähän työkaluja. Opinnäytetyössä mainittu hälytysten massatestaus olikin ainoa toteutukseen asti saatettu automaattinen testityökalu. Jos opinnäytetyön aihetta jatkojalostettaisiin, voitaisiinkin keskittyä enemmän testauksen automatisoinnin kehittämiseen. Tätä automatisointia voitaisiin lähteä toteuttamaan automaattitestaussovellusten, kuten Robot Frameworkin avulla yhdistämällä pienempiä automaattitestejä yhdeksi laajemmaksi kokonaisuudeksi. Testausympäristön käyttöä voisi myös jatkojalostaa. Ignitionilla pystyy tekemään lyhyitä ohjelmia Python-ohjelmointikielellä, joten tätä saattaisi myös kyetä hyödyntämään testauksen automatisoinnissa. Yksi varteenotettava vaihtoehto testauksen automatisointiin olisi luoda satunnaista dataa OPC-palvelimelle, joka liitettäisiin valvomojärjestelmään. Tämän avulla voitaisiin testata, näkyvätkö arvot oikein valvomonäytöillä ja aktivoituvatko oikeat hälytykset asetettujen rajaarvojen ylittyessä.

Simulointia voitaisiin myös jatkojalostaa enemmän. Jos resursseja löytyy tarpeeksi, voitaisiin simuloida sekä laitoksen palamisen prosesseja, että automaatiojärjestelmän toimintaa. Tämän avulla saataisiin hyötyä valvomoiden testaamisen lisäksi myös automaatiojärjestelmän testaamiseen, kun voitaisiin paremmin varmentaa automaatio-ohjelman toimivuus. Prosessisimulaatiolla voitaisiin myös parantaa laitosten suunnitteluprosessia prosessitekniikan osalta, kun voitaisiin paremmin testata laitoksen laitteiden mitoituksia.

LÄHDELUETTELO

ACKERMAN, Pascal 2017. Industrial Cybersecurity. Birmingham: Pack Publishing, Limited.

BAILEY, David ja WRIGHT, Edwin 2003. Practical SCADA for Industry. Oxford: Elsevier Science & Technology.

BOYER, Stuart 2010. SCADA: Supervisory Control and Data Acquisition. International Society of Automation and Technology

CHAUDHURI, Uttam R. ja CHAUDHURI, Utpal R. 2012. Fundamentals of Automatic Process Control. Boca Raton: Taylor & Francis Group.

CHEMUTURI, Murali ja CAGLEY, Thomas 2010. Mastering Software Project Management: Best Practices, Tools and Techniques. Fort Lauderdale: J. Ross Publishing.

COFFIN, Michael J. 1999. Direct Digital Control for Building HVAC Systems. New York: Springer Science+Business Media, LCC.

EVERETT, Gerald D. ja Jr. McLEOD, Raymond. 2007. Software Testing: Testing Across the Entire Software Development Life Cycle. Hoboken: IEEE Computer Society Press.

HOMÉS, Bernard 2012. Fundamentals of Software Testing. London: John Wiley & Sons, Incorporated.

Inductive Automation 2020. Ignition: The New SCADA [verkkoaineisto]. [viitattu 2020-05-12]. Saatavissa: https://inductiveautomation.com/scada-software/

JASWAL, Nipun 2014. Mastering Metasploit. Birmingham: Packt Publishing, Limited.

Kepware 2020. KEPServerEX Product Overview [verkkoaineisto]. [viitattu 2020-05-10]. Saatavissa: https://www.kepware.com/en-us/products/kepserverex/

KPA Unicon a. Tunne KPA Unicon [verkkoaineisto]. [viitattu 2020-05-12]. Saatavissa: https://careers.kpaunicon.com/tunne-kpa-unicon/

KPA Unicon b. Työn merkityksellisyys houkuttelee osaajia KPA Uniconille [verkkoaineisto]. [viitattu 2020-05-12]. Saatavilla: https://careers.kpaunicon.com/tarinat/tyon-merkityksellisyys-houkuttelee-osaajia-kpa-uniconille/

KPA Unicon c. Interact [verkkoaineisto]. [viitattu 2020-05-12]. Saatavissa: https://www.kpaunicon.com/interact/

McCRADY, Stuart G. 2013. Designing SCADA Application Software: A Practical Approach. London: Elsevier.

OPC Foundation 2020. What is OPC? [verkkoaineisto] [viitattu 2020-03-20]. Saatavissa: https://opcfoundation.org/about/what-is-opc/

RealPars a. 2018. What is the Difference Between PLC and DCS? [video]. Saatavilla: https://www.youtube.com/watch?v=iF99iKlDpxA

RealPars b. 2019. What is SCADA? [video]. Saatavilla: https://www.youtube.com/watch?v=nlFM1q9QPJw

RealPars c. 2018. What is RTU? [video]. Saatavilla: https://www.youtube.com/watch?v=Ax1jTp2dl9M

RACZYNSKI, Stanislav 2006. Modeling And Simulation: The Computer Science of Illusion. Chichester: John Wiley & Sons, Ltd.

SINGH, V.P. 2008. System Modeling and Simulation. New Delhi: New Age International Ltd.

SOMMERVILLE, Ian 2007. Software Engineering. Harlow: Pearson Education Limited.

TOLK, Andreas ja ÖREN, Tuncer 2017. The Profession of Modeling and Simulation: Discipline, Ethics, Education, Vocation, Societies and Economics. Hoboken: John Wiley & Sons.