Ville Teittinen

Control System Framework automaatiojärjestelmien ohjelmistokehityksessä

Opinnäytetyö
Sähkö- ja automaatiotekniikka

2019

Kaakkois-Suomen
ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Ville Teittinen	Insinööri (AMK)	Kesäkuu 2019

Opinnäytetyön nimi

Control System Framework automaatiojärjestelmien ohjelmistokehityksessä

36 sivua

Toimeksiantaja

YSP Oy

Ohjaaja

Harri Kosonen

Tiivistelmä

YSP päätti kehittää Control System Frameworkin, jonka tarkoituksena oli standardisoida automaatiojärjestelmien ohjelmistokehitys. Kehitettävien ohjelmistojen arkkitehtuurit monimutkaistuivat huomattavasti ja toimintatavat kehittyivät, mutta CSF:n tuomista uudistuksista ei kuitenkaan ollut vielä helposti ymmärrettävää ja täysin kattavaa dokumentaatiota.

Opinnäytetyön aiheena oli dokumentoida YSP:n kehittämää CSF-standardia. Tavoitteena oli selkeä kokonaisuus, jonka avulla ymmärtää YSP:n integraatioympäristön, ohjelmistokehitysprosessin, ohjelmistot ja kirjastot.

Työ aloitettiin tutustumalla ensin PLC- ja SCADA-järjestelmien ohjelmistokehitykseen. Seuraavaksi syvennyttiin versionhallintajärjestelmään, joka on yksi tärkeimmistä työkaluista ohjelmistokehityksessä. Työssä tutkittiin eri versionhallintajärjestelmien toimintoja, ominaisuuksia ja eroja. Tämän jälkeen perehdyttiin yleisesti ohjelmistojen kehitys- ja integraatioprosesseihin. Lopuksi kuvattiin YSP:n ohjelmistojen rakennetta, integraatioympäristöä ja kehitysprosessia.

Työn tuloksena tuotettiin yrityksen sisäiseen käyttöön PLC- ja SCADA-kirjastojen käyttöohjeet, sekä PLC-ohjelmoinnin perehdytystehtävä. Tehtävän tarkoituksena on perehdyttää PLC-ohjelmointi Unity Pro XL -sovelluksella CSF-standardin mukaisesti. Lisäksi toteutettiin ratkaisu PLC-ohjelmien versiotietojen tallentamiseen.

Asiasanat

Infra, Automaatiojärjestelmä, Ohjelmistokehitys, PLC, SCADA

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Ville Teittinen	Bachelor of Engineering	June 2019

Thesis title

Control System Framework in software development for automation systems

36 pages

Commissioned by

YSP Oy

Supervisor

Harri Kosonen

Abstract

YSP developed Control System Framework in order to standardize software development for automation systems. Development methods evolved, and software architecture became essentially more complex, however no comprehensive documentation had yet been composed.

Commission of the thesis was to build up and refine documentation regarding the CSF standard. Objective was to document the integration environment, the development process, the software architecture and the libraries.

The thesis was initiated by studying PLC- and SCADA-systems in general. As second, one of the most essential tools in software development, the Version Control System was researched. Features and differences of different Version Control Systems were addressed. After that the development and integration processes of a software were generally studied. At last the software architecture, integration environment and development process of YSP were documented.

Manuals for PLC and SCADA libraries and an introduction exercise for PLC programming were produced as a result. Objective of the exercise is to familiarize a subject to PLC programming by the CSF standard. In addition, a solution was developed to record the version of PLC programs.

Keywords

Infra, Automation system, software development, PLC, SCADA

SISÄLLYS

TERMEJÄ JA KÄSITTEITÄ

Build	Ohjelmiston koontiversio
CSF	Control System Framework YSP:n kehittämä standardi automaatiojärjestelmien ohjelmistokehitykselle
HMI	Human Machine Interface Käyttöliittymä järjestelmän operointiin
ICS	Infra Control Systems Älykkäät infran ohjausjärjestelmät
PLC	Programmable Logic Controller Ohjelmoitava logiikka
SCADA	Supervisory Control and Data Acquisition Valvomo-ohjelmisto
VCS	Version Control System Versionhallinta

1 JOHDANTO

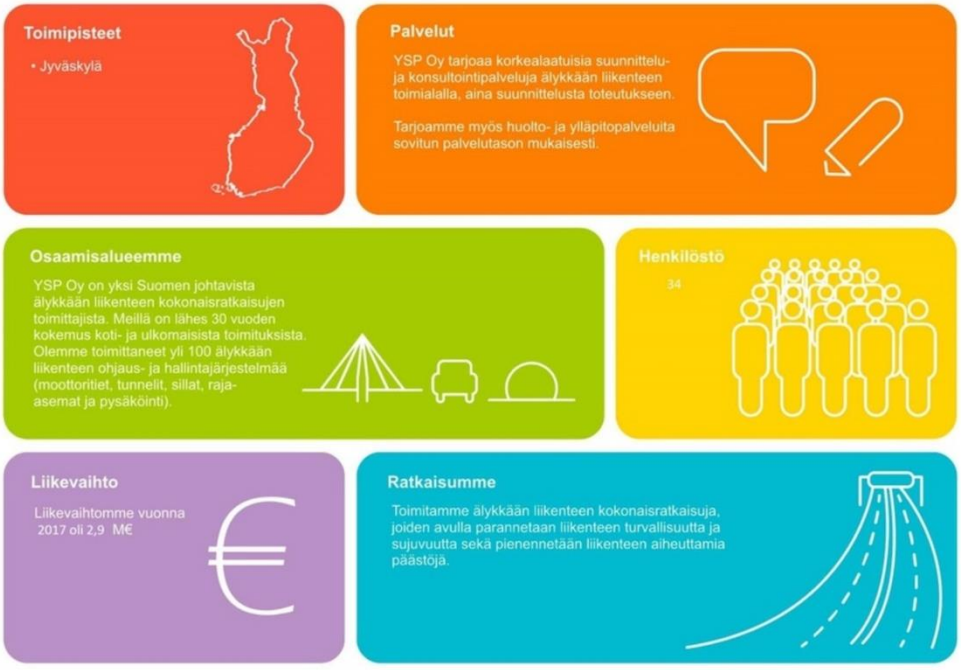
Modernien älykkään liikenteen ohjaus- ja valvontajärjestelmien ohjelmistoilta vaadittavien toimintojen ja ominaisuuksien lisääntyessä ohjelmistoista on tullut monimutkaisempia ja niiden kehityksestä entistä haasteellisempaa. Perinteisen mallin mukaan toteutetussa automaatiojärjestelmän ohjelmistokehityksessä havaittiin puutteita ja rajoituksia. Sovellukset jouduttiin lähes poikkeuksetta räätälöimään projektikohtaisesti sopiviksi. Perinteinen räätälöity malli todettiin epäkäytännölliseksi, joten YSP lähti kehittämään omaa kirjastoa nimeltä Control System Framework, jonka avulla oli tarkoitus standardisoida PLC ja valvomo-ohjelmointi ja yrityksen toimintamalli ohjelmistokehityksessä.

Opinnäytetyön tehtävänä on dokumentoida YSP:n ohjelmistokehitystä ja luoda selkeä paketti, joka auttaa lukijaa ymmärtämään YSP:n ohjelmistokehitysprosessin ja ympäristöt, sekä SCADA moduulit ja PLC lohkojen kirjastot. Tuotettava materiaali sisältää mm. selostusta YSP:n projektien rakenteesta, käyttöohjeet SCADA moduulien sekä PLC lohkojen kirjastoille ja perehdytyksen PLC ohjelmointiin.

Työssä tutustutaan ensin PLC- ja SCADA-järjestelmiin yleisesti. Lisäksi perehdytään versionhallintajärjestelmään, joka on yksi tärkeimmistä työkaluista ohjelmistokehityksessä. Sen jälkeen tutustutaan ohjelmiston kehitys- ja integraatioprosesseihin. Lopuksi käydään läpi ohjelmistokehitys YSP:n kehittämän CSF-standardin mukaisilla menetelmillä ja kirjastoilla.

2 TOIMEKSIANTAJAN ESITTELY

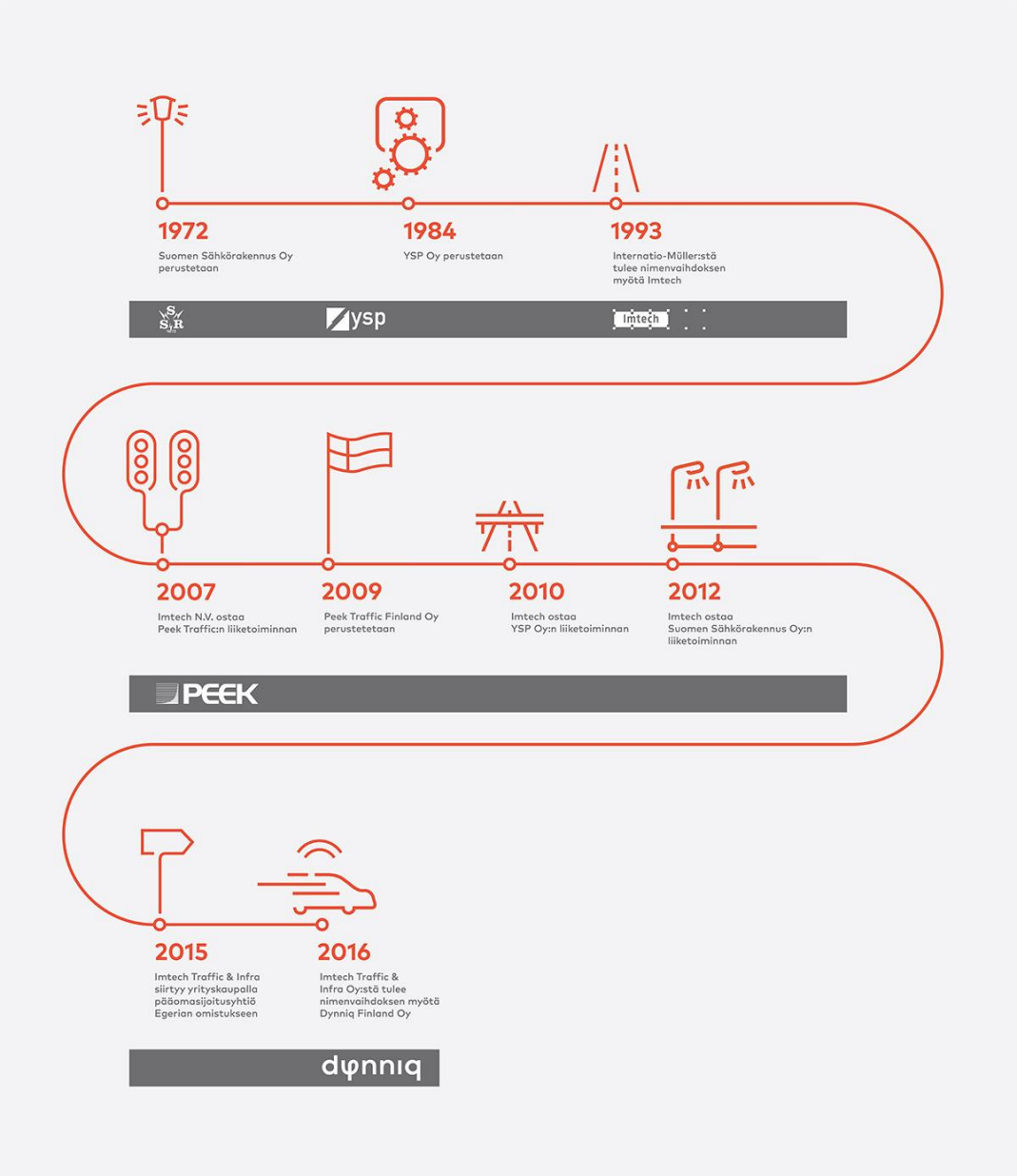
YSP Oy on jyväskyläläinen älykkäiden ohjausjärjestelmien asiantuntijayritys, joka keskittyy pääasiassa älykkään liikenteen kokonaisratkaisuihin ja on alallaan yksi Suomen johtavia yrityksiä. Yritys onkin toimittanut erilaisia liikenteen ohjaus- ja valvontajärjestelmiä lähes 30:n vuoden ajan, mukaan lukien yli 100 älykkään liikenteen järjestelmää. YSP toteuttaa järjestelmiä moniin erilaisiin kohteisiin, kuten moottori- ja maantiet, tunnelit ja avattavat sillat, tarjoten palveluita esisuunnittelusta toteutukseen sekä huolto- ja ylläpitotehtäviin. Kuvaan 1 on kerätty lyhyt yleiskatsaus yrityksestä. (YSP s.a.)



Kuva 1. YSP lyhyesti (YSP s.a)

2.1 Yrityksen historia

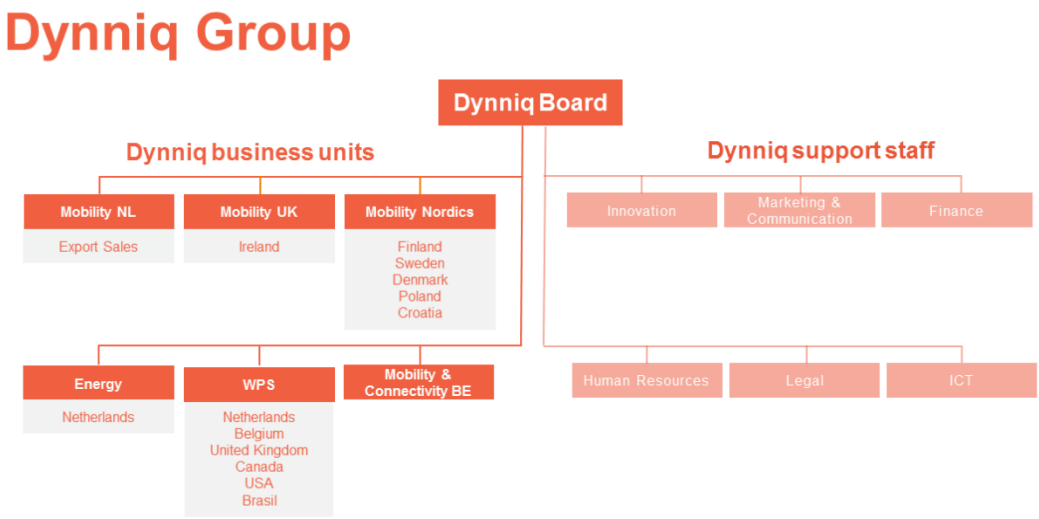
YSP Oy, eli Yleinen Sähköpalvelu Oy aloitti toimintansa sähkö- ja automaatiosuunnittelun parissa Jyväskylässä vuonna 1984. 90-luvun puolivälissä yrityksen toimintaa laajennettiin vastaamaan teknologian kehitystä. Uutena osaamisalueena ICT mahdollisti erilaisten ohjaus- ja valvontajärjestelmien kokonaisratkaisujen toteutuksen. (YSP s.a.) Vuonna 2010 Imtech osti YSP:n liiketoiminnan vahvistaakseen asemaansa Skandinavian alueella. Imtech Traffic & Infra divisioona siirtyi sijoitusyhtiö Egerian omistukseen elokuussa 2015 ja pian tämän jälkeen helmikuussa 2016 Imtech Traffic & Infrasta tuli nimenvaihdoksen seurauksena Dynniq. (Dynniq kotisivut s.a.) Kuvassa 2 on esitetty YSP:n kehitys kohti Dynniqiä (YSP s.a).



Kuva 2. YSP:n historia (YSP s.a.)

2.2 Dynniq-konserni

Dynniq on infra-alalla kansainvälisesti toimiva konserni, joka toimittaa kokonaisratkaisuja älyliikenteen, pysäköinnin ja energian osa-alueilla. Konsernilla on yhteensä yli 1800 työntekijää toimipisteissään Euroopassa ja Amerikassa. Kuvassa 1 on listattuna maat, joissa eri yksiköiden toimipisteet sijaitsevat. (Dynniq Sharepoint s.a.)



Kuva 3. Dynniqin eri yksiköiden toimipisteet (Dynniq sharepoint s.a)

3 AUTOMAATIOJÄRJESTELMIEN OHJELMISTOKEHITYS

3.1 PLC

PLC eli ohjelmoitava logiikka on automaatiojärjestelmien hallintaan tarkoitettu laite. Logiikan peruskomponentit ovat Virtalähde, I/O-moduuli, joka yhdistää hallittavan järjestelmän tai laitteen PLC:hen ja prosessori, joka suorittaa ohjelmoituja loogisia operaatioita. Ohjelmointi toteutetaan yleensä tietokonesovelluksella ja ladataan myöhemmin PLC-laitteeseen. (Unitronics s.a.)

IEC 61131-3 standardi määrittelee viisi PLC-ohjelmoinnissa käytettävää kieltä.

Function Block Diagram (lyh. FDB) on graafinen ohjelmointikieli, jossa ohjelmakoodi muodostuu toiminnallisista lohkoista, jotka yhdistyvät toisiinsa linkeillä (Schneider Electric 2011, 321–323).

Ladder Diagram (lyh. LD) on graafinen ohjelmointikieli, jossa ohjelmakoodi sijoitetaan kahden virtakiskon väliin muodostaen virtapiirejä (Schneider Electric 2011, 347–350).

Instruction List (lyh. IL) on tekstimuotoinen ohjelmointikieli, jossa ohjelmakoodi muodostuu allekkain rivikohtaisesti listatuista käskyistä (Schneider Electric 2011, 449–453).

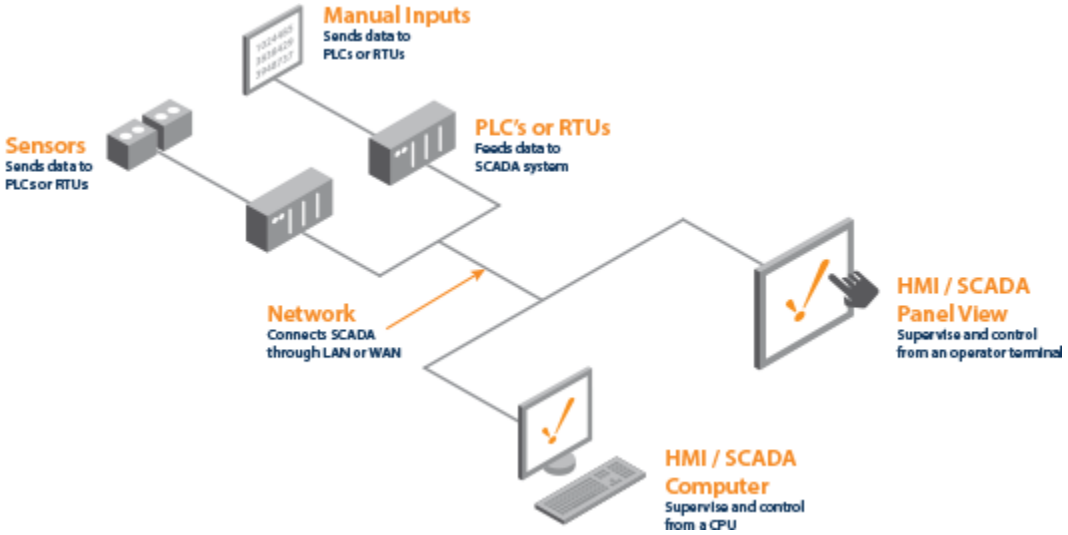
Structured Text (lyh. ST) on tekstimuotoinen ohjelmointikieli, jossa ohjelmakoodi muodostuu lausekkeista ja käskyistä (Schneider Electric 2011, 321–323).

Sequential Function Chart (lyh. SFC) on graafinen ohjelmointikieli, jossa ohjelma koostuu toiminnallisista lohkoista ja ehdoista, joiden täyttyessä siirrytään seuraavaan lohkoon (Schneider Electric 2011, 497–501).

Lisäksi standardi määrittelee muita PLC-ohjelmoinnin keskeisiä elementtejä, kuten erilaiset tietotyypit ja lohkot. Perustietotyyppi (eng. Elementary Data Type) on muuttujan arvolle annettava määritelmä, jonka avulla ohjelma ymmärtää miten muuttujaa tulee käsitellä. Kaikki perustietotyypit on määritetty standardissa. Johdetut tietotyypit (eng. Derived Data Type) ovat valmistajan tai käyttäjän määrittämiä rakenteita, jotka voivat esimerkiksi koostua useista eri tietotyypeistä. Lohkot (eng. Function Blocks) ovat ohjelman rakenteellisia osia, jotka muodostuvat tuloista, lähdöistä ja sisäisestä ohjelmakoodista. (IEC 61131-3: 2003.)

3.2 SCADA

SCADA tarkoittaa ohjelmistoa, jonka avulla voidaan reaaliaikaisesti valvoa ja hallita jotakin prosessia tai järjestelmää, sekä kerätä siitä tietoa. SCADA-ohjelmisto on osa järjestelmää, johon kuuluu yleensä HMI, SCADA-ohjelmisto, sekä oheislaite, kuten PLC. (Gupta 2011.) Kuvassa 4 on esitetty yksinkertainen malli SCADA-järjestelmästä (Automation Excellence s.a).



Kuva 4. Yksinkertainen SCADA-järjestelmä (Automation Excellence s.a)

Järjestelmää ohjataan pääosin automaattisesti PLC:lle määritettyjen ohjelman, asetusarvojen ja järjestelmän palauttamien arvojen perusteella. SCADA-ohjelmistolla voidaan vaikuttaa PLC-ohjelman toimintaan muuttamalla esimerkiksi asetusarvoja tai tekemällä ohjauskomentoja, joilla voidaan ohittaa automaattiohjaukset. Järjestelmän operointiin voidaan käyttää erillistä HMI-laitetta tai SCADA-ohjelmistoon kuuluvaa käyttöliittymää. (Gupta 2011.)

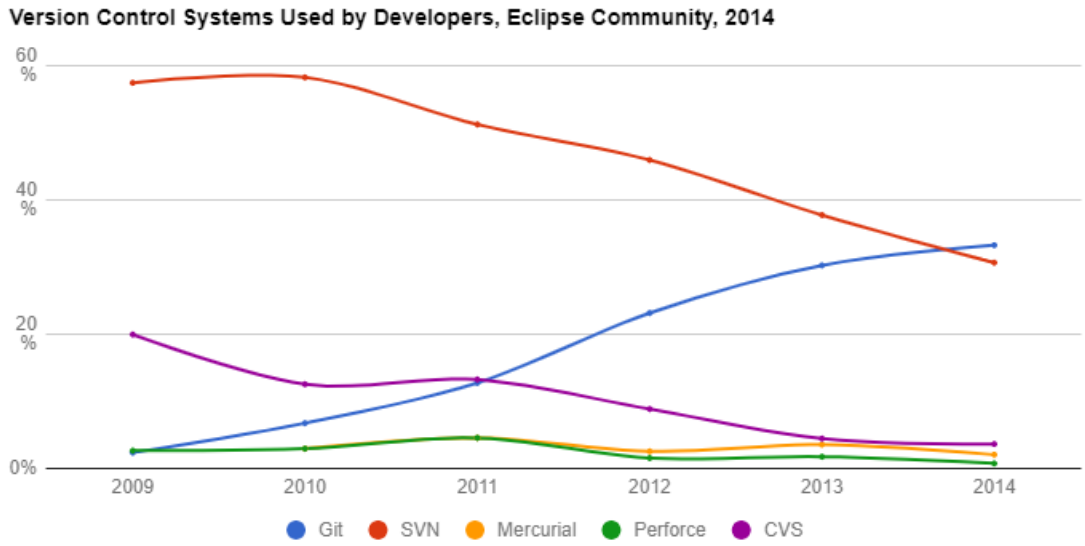
SCADA-ohjelmistojen kehitykseen käytetään integroitua kehitysympäristöä, eli tietokonesovellusta, jonka avulla voidaan toteuttaa valvomon kaikki perusominaisuudet. Tärkeimpiä ominaisuuksia ovat rajapinnat ulkoisille laitteille, järjestelmästä kerättävän tiedon käsittely, järjestelmän ohjaus ja käyttöliittymä, jonka avulla järjestelmää voidaan valvoa ja ohjata. (Sielco Sistemi s.a.)

4 VERSIONHALLINTAJÄRJESTELMÄ

Useiden versioiden tallettaminen manuaalisesti kopioimalla ja nimeä muuttamalla on erittäin työläs ja virhealtis menetelmä ylläpitää versiohistoriaa. (Somasundaram 2013.) Manuaalisesti toteutettujen versioiden välillä on myös työlästä pitää tarkkaa kirjaa niiden välissä tehdyistä muutoksista. Ratkaisu löytyy versionhallinnasta, joka on keskeinen työkalu ohjelmistokehityksessä. (fournova s.a.) Versiohallinnalla tarkoitetaan järjestelmää, joka tallentaa säilöön vanhat versiot ja versioiden välillä tehdyt muutokset, mahdollistaen tehtyjen muutosten tarkistelun ja tiedoston minkä tahansa vanhan version palauttamisen (Somasundaram 2013).

Ensimmäisenä toimivana ratkaisuna kehitettiin paikallinen versionhallinta. Yksi suosituimmista paikallisen versionhallinnan järjestelmistä oli Revision Control System (RCS), joka kirjaa tiedostoihin tehtävät muutokset vain käyttäjän paikalliselle levylle. Muutokset tallentuvat päivityspaketteina, joiden avulla tiedoston tietty versio voidaan luoda uudelleen lisäämällä tiettyä ajanhetkeä ennen tuotetut paketit järjestyksessä. (Somasundaram 2013.)

Paikallinen versionhallinta ei kuitenkaan tarjonnut ratkaisua toiselle olennaiselle ongelmalle, joka oli yhteystyön ja tiedostojen jakamisen rajoittuneisuus. Ensimmäisen sukupolven jälkeen onkin kehitetty kaksi uutta yhteistyötä edistävää järjestelmää, joista ensimmäinen on keskitetty ja toinen hajautettu versionhallinta. (Somasundaram 2013.) Toista sukupolvea edustava SVN on pitkään ollut suosituin järjestelmä, mutta kolmannen sukupolven järjestelmä Git on kuitenkin ajan myötä jättänyt sen taakseen. Kuvassa 3 näkyy Eclipse Communityn vuosittaisen kyselyn tulokset vastaajien useimmissa projekteissa käyttämästä ensisijaisesta lähdekoodin hallintajärjestelmästä. (RhodeCode s.a.)



Kuva 5. Eri versionhallintajärjestelmien suosion kehitys (RhodeCode s.a)

4.1 Versionhallinnan käsitteitä

Varasto (eng. repository) on versionhallinnan ydinkomponentti, johon tallentuu kaikki järjestelmän data. Asiakasohjelmat ovat yhteydessä varastoon ja voivat kirjoittaa tai lukea tietoja. Kun varastoon kirjoitetaan uutta tietoa, niin vanhat versiot jäävät muistiin ja ne voidaan myöhemmin palauttaa. Varastossa säilytetään myös lokia muutoksista, jossa on mm. tiedot muutosten ajankohdista ja tekijöistä. (Collins-Sussman, Fitzpatrick & Pilato 2011, 1.)

Työkopio (eng. Working Copy) on paikalliselle levylle tallennettu kopio tietystä versiosta. Käyttäjä voi hakea työkopioon korvaavan version varastosta tai viedä työkopiosta varastoon uuden version. Käyttäjä ei siis tee muutoksia suoraan varastoon, vaan ne tehdään työkopioon. Työkopion ja varaston väliseen kommunikaatioon käytetään asiakasohjelmaa. (Collins-Sussman, Fitzpatrick & Pilato 2011, 2.)

Taulukossa 1 on esitetty keskitetyn ja hajautetun versionhallintajärjestelmän tärkeimmät perustoiminnot. Taulukossa näkyy toiminnon nimi suomeksi ja englanniksi, sekä versionhallintajärjestelmä, jossa toimintoa voidaan käyttää. Viimeisessä sarakkeessa on kerrottu lyhyesti mitä toiminnolla tehdään.

Taulukko 1. Keskitetyn ja hajautetun versionhallintajärjestelmän perustoimintoja (Sink 2011)

eng.	suom.	Keskitetty	Hajautettu	Toiminto
Create	Luoda	x	x	Toiminnolla luodaan uusi varasto.
Checkout	Kuitata	x	x	Kopioi varastosta tiedostojen uusimmat versiot täysin uudeksi työkopioksi.
Commit	Viedä	x	x	Vie työkopioon tehdyt muutokset varastoon uudeksi versioksi
Update	Tuoda	x	x	Tuo varastosta uusimman version ja yhdistää sen paikalliseen työkopioon.
Clone	Kloonata		x	Luo uuden täysin identtisen instanssin varastosta.
Push	Työntää		x	Työntää paikallisesta varastosta muutokset toiseen varaston instanssiin.
Pull	Vetää		x	Kopioi toisesta varaston instanssista muutokset paikalliseen varastoon.

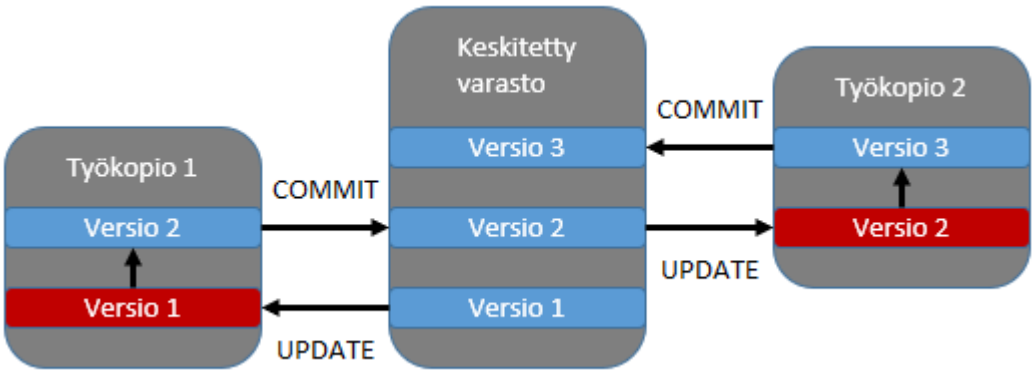
4.2 Keskitetty versionhallinta

Keskitetyssä versionhallinnassa (kuva 6) rajoittunut pääsy varastoon on ratkaistu sijoittamalla se keskitetylle palvelimelle, johon käyttäjät ovat yhteydessä internetin yli. Muiden tekemiä muutoksia voidaan seurata helpommin ja uusimmat versiot ovat heti kaikkien saatavilla, koska varasto on kaikille yhteinen. (Somasundaram 2013.) Valvojalle on myös selkeästi vaivattomampaa hallinnoida oikeuksia keskitetyllä palvelimella, kuin useiden käyttäjien paikallisilla varastoilla (Chacon & Straub 2014).

Kuva 6. Keskitetyn versionhallinnan rakenne

Keskitetyn varaston käytöllä on edellä mainittujen etujen lisäksi myös huonojakin ominaisuuksia. Jos palvelin tai yhteys palvelimeen lakkaa toimimasta, niin käyttäjillä on käytössään vain viimeisimmät palvelimelta haetut versiot, eikä varastoon voida myöskään tallettaa uusia versioita. Pahimmillaan palvelimen korruptoituessa tai hajotessa voidaan jopa menettää koko versiohistoria tai osa siitä, jos sitä ei ole asianmukaisesti varmuuskopioitu. (Somasundaram 2013.)

Kuvassa 7 on esitetty keskitetyn versionhallinnan työnkulku yksittäisellä tiedostolla, jota työstää kaksi henkilöä. Kuvassa sinisellä merkityt versiot ovat prosessin jälkeen tallessa olevia ja punaiset versiot välivaiheita, jotka häviävät prosessin aikana. Mustat nuolet kuvaavat prosessin vaiheita.

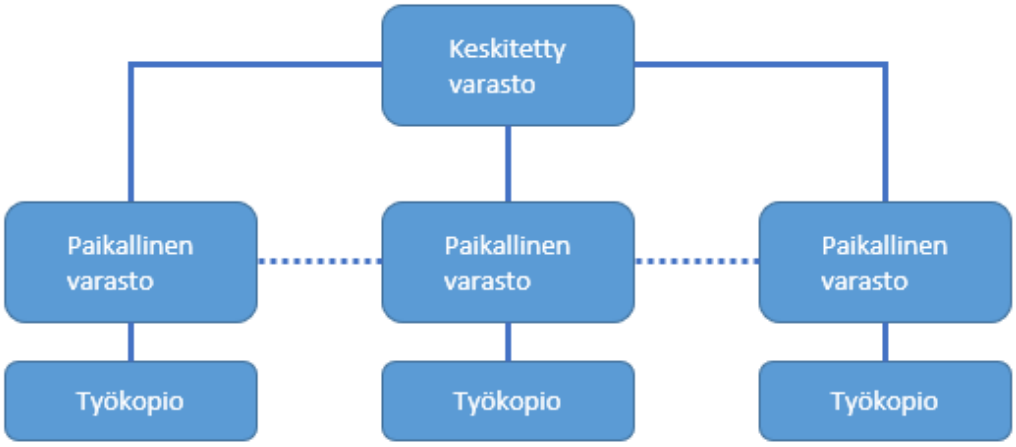


Kuva 7. Keskitetyn versionhallinnan työnkulku

Yksinkertaisimmillaan työn kulku muodostuu kolmesta vaiheesta. Henkilö 1 tuo keskitetystä varastosta muutokset paikalliseen työkopioonsa, tekee tiedostoon tarvittavat muutokset ja vie muokatun tiedoston takaisin keskitettyyn varastoon. Tämän jälkeen henkilö 2 voi työstää samaa tiedostoa tuomalla version 2 omaan paikallaiseen työkopioonsa. (Collins-Sussman, Fitzpatrick & Pilato 2011.)

4.3 Hajautettu versionhallinta

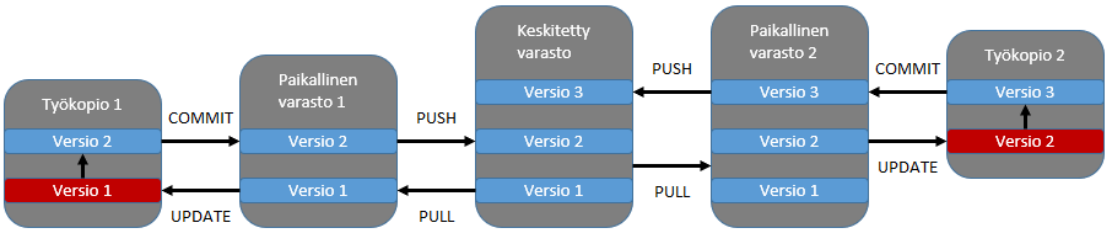
Hajautettu versionhallinta kehitettiin käytännössä yhdistämällä keskitetty ja paikallinen varasto tarkoituksena hyödyntää molempien vahvuuksia. Hajautetussa järjestelmässä varasto tallennetaan sekä keskitetylle palvelimelle että jokaisen käyttäjän paikalliselle levylle. Hajautetulla rakenteella käyttäjät voivat tehdä muutoksia varastoon ilman yhteyttä keskitettyyn palvelimeen ja työntää ne sinne myöhemmin. Paikalliseen varastoon tehtävät operaatiot ovat myös huomattavasti nopeampia kuin keskitetylle palvelimelle tehtävät, sillä niitä ei tarvitse tehdä internetin yli. (Somasundaram 2013.) Kuvassa 8 on esitetty hajautetun versionhallinnan rakenne yksinkertaisimmillaan.



Kuva 8. Hajautetun versionhallinnan rakenne

Yhtenä tärkeänä versionhallinnan elementtinä on jokaiselle käyttäjälle varattu henkilökohtainen työskentelytila. Henkilökohtaisessa työtilassa käyttäjä voi keskittyä täysin omaan työskentelyynsä, riippumatta muiden tekemistä muutoksista. Hajautetussa versionhallinnassa jokaisella käyttäjällä on oma paikallinen varasto, jonka avulla voidaan hyödyntää versiohistoriaa, eikä tuotoksia tarvitse jakaa muille kehittäjille ennen kuin se on tarpeellista. (Sink 2011.)

Kuvassa 9 on esitetty hajautetun versionhallinnan työnkulku yksittäisellä tiedostolla, jota työstää kaksi henkilöä. Kuvassa sinisellä merkityt versiot ovat prosessin jälkeen tallessa olevia ja punaiset versiot välivaiheita, jotka häviävät prosessin aikana. Mustat nuolet kuvaavat prosessin vaiheita.



Kuva 9. Hajautetun versionhallinnan yksinkertainen työnkulku

Koska hajautetussa version hallinnassa jokaisella käyttäjällä on oma paikallinen varasto, aloitetaan työnkulku vetämällä tiedosto keskitetystä varastosta omaan paikalliseen varastoon. Paikallisesta varastosta tuodaan muutokset työkopioon, tiedostoon tehdään muutoksia ja muokattu tiedosto viedään takaisin paikalliseen varastoon. Paikallisesta varastosta täytyy vielä työntää muutokset keskitettyyn varastoon, jolloin ne ovat muiden käyttäjien saatavilla. (Ernst 2018.)

5 KEHITYSPROSESSI

Perinteisen projektimallin mukaan toteutettu ohjelmistokehitys oli usein hidasta ja eikä kyennyt vastaamaan muutokseen tehokkaasti. Nykyään ohjelmistokehityksessä yleensä käytetäänkin ketterän kehityksen menetelmiä, joiden tavoitteena on nopeuttaa kehitysprosessia ja parantaa tuotosten laatua. Ketterä kehitys tarkoittaa the Agile Manifesto -julistuksessa esitettyjä arvoja ja periaatteita edustavia toimintamalleja ja menetelmiä. (Smartsheet s.a.)

5.1 Ketterän kehityksen arvot

Ketterän kehityksen pohjana on yksilöiden muodostama tiimi, jolla on kehityksen edellyttämät ammattitaito, ympäristö ja työkalut. Tiimin toiminta on itseohjautuvaa, eli se pyrkii saavuttamaan annetut tavoitteet parhaaksi kokemallaan tavalla. Kehitystiimin sisäinen kommunikaatio on avainasemassa, sillä kaikki yksilöt yhdessä vastaavat kehityksen etenemisestä. (Smartsheet s.a.)

Tiimin työskentelyä ohjaa alusta asti säännöllinen ja jatkuva arvon tuottaminen. Arvolla tarkoitetaan täysin valmista osakokonaisuutta ja lopulta kokonaista ohjelmistoa. Kehityksessä tuotettu arvo kuvastaa ensisijaisesti edistystä kohti päämäärää. Tehokkaan kehityksen kannalta on elintärkeää minimoida turhan työn määrä. (Smartsheet s.a.)

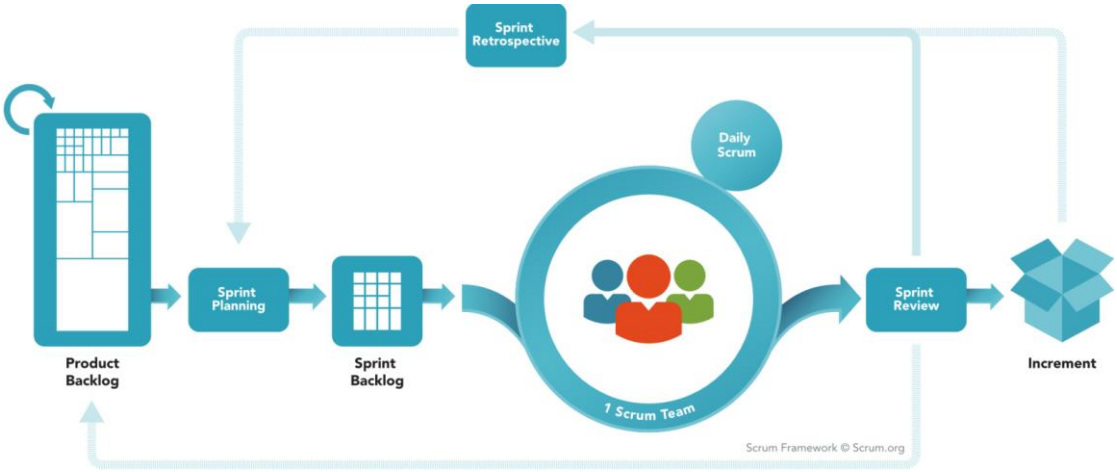
Yhteistyö asiakkaan kanssa on säännöllistä koko kehitysprosessin ajan. Ohjelmiston vaatimuksia ei määritetä yksityiskohtaisesti ennen kehityksen aloittamista, vaan asiakkaalle ja mahdollisesti tuotteen käyttäjille annetaan mahdollisuus vaikuttaa tuotteen kehitykseen koko prosessin ajan. Näin varmistetaan, että tuote vastaa aidosti asiakkaan tarpeita ja muutoksiin voidaan reagoida tehokkaammin. (Smartsheet s.a.)

Yhtenä ketterän kehityksen tärkeänä elementtinä onkin kyky toimia muuttuvassa ympäristössä ja jopa pyrkiä jatkuvaan muutokseen. Yksityiskohtaisesti suunnitellun kokonaisuuden sijaan keskitytään pienempiin osakokonaisuuksiin kerrallaan. Muutokset tulee nähdä projektia kehittävänä ja asiakkaalle lisäarvoa tuottavana prosessina. (Smartsheet s.a.)

5.2 Ketterän kehityksen menetelmät

Ketterän kehityksen menetelmillä tarkoitetaan tiettyjä sovittuja toimintamalleja ja tapoja, joista kaikki yksittäisen kehitystiimin jäsenet ovat sopineet ja sitoutuneet noudattamaan niitä (Agile Alliance s.a). Ketterä kehitys on kuitenkin kokonaisuutena haastava hallita ja lukuisia valmiita viitekehyksiä erilaisista menetelmistä on saatavilla, joten niiden pohjalta on hyvä aloittaa kehittämään omaa toimintaa (Shore & Warden 2008).

Scrum on yksi suosituista ketterän kehityksen viitekehyksistä. Scrumin kehitysprosessi (kuva 10) on iteratiivinen ja inkrementaalinen, eli useasti toistuva prosessi, jonka aikana tuotetaan jokin valmis osa lisää lopulliseen tavoitteeseen. Scrumin kehitysprosessin toteuttajana toimii Scrum-tiimi, joka työstää tuotoksia ja osallistuu ennalta määrättyihin tapahtumiin iteraation, eli toistettavan jakson, aikana. (Schwaber & Sutherland 2017.)



Kuva 10. Scrumin kehitysprosessi (Scrum.org)

Scrumin tuotoksiin kuuluu kehitysjono (eng. product backlog), sprintin kehitysjono (eng. sprint backlog) ja inkrementti (eng. increment). Inkrementti tarkoittaa iteraation aikana täysin valmiiksi saatuja osia, jotka edistävät lopullisen tavoitteen saavuttamista. Kehitysjono on tehtävälista, joka määrittelee kaikki tiedossa olevat kehityskohdat, joilla saavutetaan lopullinen tavoite. Sprintin kehitys jono vastaavasti on yhden iteraation tehtävälista. (Schwaber & Sutherland 2017.)

Scrum-tiimi koostuu tuoteomistajasta, scrummasterista ja kehitystiimistä. Tuoteomistaja on yksi henkilö, jonka vastuulla on ylläpitää kehitysjonoa. Kehitystiimi on joukko yksilöitä, jotka yhdessä toteuttavat kehitysjonolle määritettyjen tehtävien mukaisia inkrementtejä. Scrummasterin tehtävä on käytännössä tukea scrumin toteutumista tiimin toiminnassa. (Schwaber & Sutherland 2017.)

Scrumin tapahtumat muodostuvat iteraatiojaksosta eli sprintistä ja sen aikana pidettävistä palavereista. Sprintin alussa on sprintin suunnittelu, jossa scrum-tiimi päättää sprintin kehitysjonolle siirrettävät kohdat, jotka voidaan saada sprintillä täysin valmiiksi. Sprintin aikana pidetään päivittäin lyhyt palaveri, jonka aikana käydään läpi mitä jokainen on päivän aikana tehnyt, mitä tekee seuraavaksi ja onko esiintynyt ongelmia, tarkoituksena seurata sprintin etenemistä säännöllisesti. Sprintin lopussa on katselmointi, jossa tarkastellaan sprintin inkrementtiä, ja retrospektiivi, jonka tarkoituksena on tarkastella ja kehittää scrum-tiimin toimintaa. (Schwaber & Sutherland 2017.)

6 INTEGRAATIOPROSESSI

Tässä luvussa esitellään yleisesti integraatioprosessia perustuen Vilkmanin (2010) tietojärjestelmätieteen kandidaatin tutkielmaan ”Jatkuva integraatio ohjelmistokehityksessä”.

Stavidroun (1999) määritelmässä integraatiolla tarkoitetaan prosessia, jonka tarkoituksena on tuottaa eheä ja vaatimukset täyttävä ohjelmisto, yhdistämällä siihen kuuluvat erilliset komponentit. Binderin (1999) mukaan integraatioprosessissa kuuluu myös varmistaa kaikkien komponenttien, eli ohjelmistotuotteen osakokonaisuuksien, kuten esimerkiksi funktioiden, moduulien ja alijärjestelmien, toimivan keskenään yhdessä.

Kehittyneet ohjelmistotuotteet sisältävät yleensä lukuisia, jopa tuhansia komponentteja, joiden kehitys ei ole sidottu aikaan, paikkaan tai henkilöstöön. Täysin erillään kehitetty komponentti voi esimerkiksi olla avoimen lähdekoodin sovellus. (Kim, Park, Yun & Lee 2008.)

Päivitettäessä vanhoja tai lisättäessä täysin uusia komponentteja, tulee integraatio toteuttaa erityisen huolellisesti, sillä toiminnallisuuden muutokset yhdessäkin komponentissa voivat jo aiheuttaa odottamattomia ongelmia koko ohjelmistotuotteen toiminnalle (Kim ym. 2008). Integraatio kannattaakin toteuttaa mahdollisimman aikaisessa vaiheessa, jolloin todennäköisyys ongelmien esiintymiselle on pienempi ja niiden korjaaminen helpompaa (van der Storm 2007).

6.1 Jatkuva Integraatio

Jatkuva integraatio on joukko käytäntöjä, joiden tarkoituksena on välttää pitkät ja monimutkaiset integraatiovaiheet. Käytäntöjen avulla pyritään yksinkertaistamaan ja osittain automatisoimaan integraatioprosessiin liittyviä toimenpiteitä. Tarkoituksena on mahdollistaa integraation suorittaminen säännöllisesti, kun muutoksia on vähän ja mahdollisten virheiden löytäminen ja korjaaminen on huomattavasti helpompaa. (Fowler 2006.)

Kaikki ohjelmiston lähdemateriaali tulee säilyttää samassa tietolähteessä, kuten esimerkiksi versionhallintajärjestelmän palvelimella. Tietolähteessä säilytettään kaikki se lähdemateriaali, jota tarvitaan toimivan koontiversion luomiseen millä tahansa koneella. Lähdemateriaalin buildaaminen toimivaksi koontiversioksi on usein monimutkainen ja turhaa aikaa kuluttava prosessi, joka voidaan automatisoida. Automaattisella buildaamisella vältytään ylimääräisiltä virheiltä ja koontiversiolle voidaan samalla suorittaa automaattitestejä. (Fowler 2006.)

Kehittäjien tulee pyrkiä viemään tekemänsä muutokset päähaaraan säännöllisesti. Kun muutoksia on tehty vain vähä, on ongelmien selvittäminen ja korjaaminen kivuttomampaa. Lisäksi mahdolliset eri kehittäjien tekemien muutosten väliset konfliktit voidaan havaita aikaisemmin. Muutokset yhdistetään ensin omalla koneella uusimpaan päähaaraan ja buildataan. Onnistuneen koontiversion lähdemateriaalit viedään yhteiseen tietolähteeseen. (Fowler 2006.)

Ohjelmisto tulee buildata säännöllisesti erillisellä integraatiopalvelimella. Buildaaminen voidaan suorittaa manuaalisesti tai automaattisesti automaatiopalvelimen avulla. Automaatiopalvelin hakee automaattisesti tehdyt muutokset, buildaa ohjelmiston ja suorittaa automaattitestauksen. Buildaamisessa tai testeissä ilmenneet ongelmat tulee korjata heti niiden ilmaantuessa, koska tarkoituksena on ylläpitää päähaarassa viimeisen toimivan koontiversion lähdemateriaalia. (Fowler 2006.)

Jatkuvassa integraatiossa tarvitaan yleensä ohjelmistolle useita erilaisia ympäristöjä eri tehtäviin, joten on suositeltavaa, että ohjelmisto asennus eri ympäristöihin toteutetaan automaattisesti skriptien avulla. On myös tärkeää, että viimeisin buildattu ja toimiva ohjelmisto on kaikkien saatavilla ja käytettävissä vaivattomasti. Lisäksi ohjelmiston nykyinen tila ja siihen tehdyt muutokset tulee olla kaikkien nähtävillä, jolloin kehityksen etenemistä on helpompi seurata. (Fowler 2006.)

7 OHJELMISTOKEHITYS YSP: LLÄ

7.1 Control System Framework

Ennen Control System Frameworkin (lyh. CSF) tarjoamaa pohjaa ohjelmistokehittäjät työskentelivät pääasiassa itsenäisesti. Ratkaisut ja toiminnot olivat enimmäkseen projektikohtaisia. Ohjelmistojen kehitys oli hidasta ja laajojen eroavaisuuksien vuoksi testausautomaation kehitys käytännössä lähes mahdotonta. CSF on ohjausjärjestelmien kehitykselle tarkoitettu viitekehys, jonka tavoitteena on optimoida YSP:n ohjelmistokehitystä luomalla yhtenäinen malli, jonka pohjalta ohjelmistokehitystä toteutetaan. Sen keskeisenä periaatteena hyödyntää resursseja tehokkaammin ja jakaa niitä eri yksiköiden välillä.

ICS-projekteissa siirryttiin käyttämään mikropalveluarkkitehtuuria. Mikropalveluarkkitehtuurilla toteutettu ohjelmisto koostuu itsenäisistä komponenteista, joilla on jokaisella oma yksittäinen tehtävä. Komponentit kommunikoivat rajapintojen kautta keskenään muodostaen kokonaisen ohjelmiston.

PLC- ja SCADA-ohjelmistojen uudistetun arkkitehtuurin tavoitteena oli kehitys kohti geneeristä muotoa, jolloin valmiita komponentteja voidaan käyttää uusissa projekteissa ilman muutoksia. Lisäksi komponenttien toiminnallinen kehitys ja lisätyt ominaisuudet voidaan päivittää huomattavasti helpommin vanhempiin ohjelmistoihin.

CSF on kuitenkin vielä suhteellisen tuore käytäntö ja sen tuomat muutokset kehittyvät jatkuvasti. Ohjelmistojen uudistunut arkkitehtuuri on selkeästi vanhaa mallia monimutkaisempi ja vaatii ohjelmiston kehittäjiltä laajempaa tuntemusta järjestelmästä. Arkkitehtuurin monimutkaistumisen seurauksena ohjelmiston ymmärtäminen on uudelle kehittäjälle haastavaa, eikä järjestelmää ole dokumentoitu vielä kattavasti.

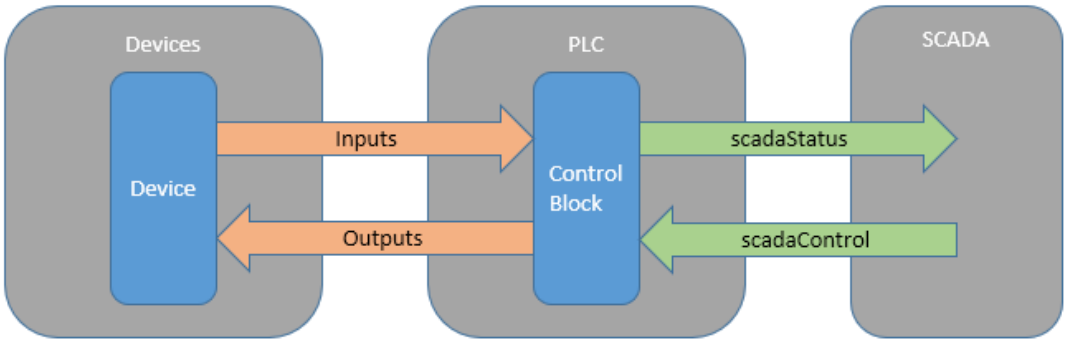
7.2 PLC-ohjelmistokehitys

YSP:llä käytetään PLC-ohjelmistojen kehityksessä pääasiassa Schneider Electricin Unity Pro XL -sovellusta. Ohjelmointi toteutetaan Structured Textillä, joka on standardin mukainen kirjoitettava ohjelmointikieli. Tekstimuotoinen oh-

jelmakoodi on helpommin siirrettävissä sovelluksen ulkopuolelle ja sitä voidaan tarvittaessa generoida tai kääntää muille sovelluksille helpommin. Poikkeuksena sekvenssiohjauksissa käytetään Sequential Function Chartia, sillä se tarjoaa niiden hallinnan kannalta hyödyllisiä ominaisuuksia.

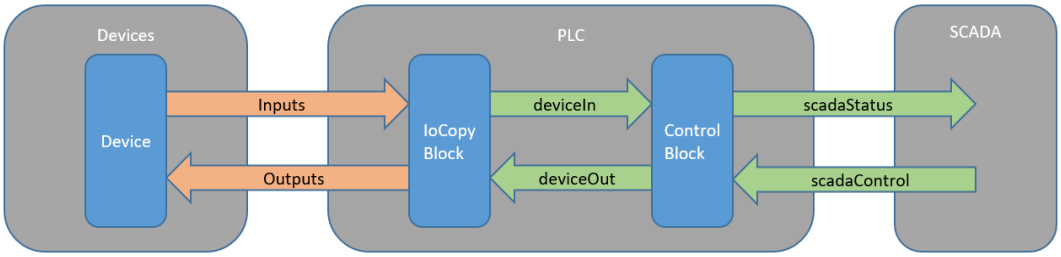
7.2.1 Ohjelman arkkitehtuuri

Perinteisen mallin mukainen PLC-ohjelma oli rakenteeltaan hyvin yksinkertainen (kuva 11). Ohjauslohko (ControlBlock) oli yhdistetty suoraan laitteen lähtöihin, tuloihin ja valvomoon. Yksinkertaisesta rakenteestaan huolimatta sen mukautuvuus laitetasolla tehtäviin muutoksiin oli heikko, minkä seurauksena muokkauksia jouduttiin tekemään jokaiseen ohjelmiston osaan laitteelta valvomoon asti.



Kuva 11. PLC-ohjelman perinteinen arkkitehtuuri

Uudistetussa arkkitehtuurissa (kuva 12) laitteen ja ohjauslohkon väliin on lisätty uusi komponentti (IoCopyBlock), jonka tarkoituksena on vähentää laitetasolla tapahtuvien muutosten vaikutusta koko ohjelmistoon. Lisäksi rakenteellinen muutos luo uusia mahdollisuuksia ohjelmiston testaamisen ja simuloinnin kehittämiselle.



Kuva 12. PLC-ohjelman uudistettu arkkitehtuuri

Kuvassa harmaat alueet vastaavat järjestelmän eri osia, joiden sisällä on sinisellä niihin kuuluvat komponentit. Nuolet ovat puolestaan komponenttien välillä kulkevaa tietoa; oranssit nuolet vastaavat yksittäisiä arvoja ja vihreät nuolet johdettuja tietotyyppejä.

7.2.2 Tietotyypit ja lohkot

Jokaiselle laitetyypille tarvitaan pääsääntöisesti neljä johdettua tietotyyppiä ja kaksi johdettua lohkoa. Johdettujen tietotyyppien ja lohkojen on tarkoitus olla yhteensopivia samaan laitetyyppiin kuuluvien erilaisten laitteiden välillä. Yksittäinen johdettu tietotyyppi koostuu kaikista muuttujista, joiden arvot viedään kahden komponentin välillä samansuuntaisesti. Pääsääntöisesti käytettävät tietotyypit ja niiden käyttö on esitetty taulukossa 2.

Taulukko 2. Ohjelmistoissa käytettävät johdetut tietotyypit

Tietotyyppi	Lähtö	Tulo	Käyttötarkoitus
deviceIn	IoCopyBlock	ControlBlock	Laitteelta PLC:lle tuotavat tilatiedot
deviceOut	ControlBlock	IoCopyBlock	PLC:ltä laitteelle vietävät ohjaukset
scadaStatus	ControlBlock	SCADA	Ohjauslohkon käsittelemät laitteen tilatiedot valvomolle
scadaControl	SCADA	ControlBlock	Valvomolta PLC:lle tuotavat ohjausarvot

IoCopyBlock-lohkon tehtävä on linkittää laitteen tulot deviceIn-tietotyyppiin ja lähdöt deviceOut-tietotyyppiin. **ControlBlock**-lohkoon on ohjelmoitu laitteen toiminnallisuus, joka kattaa tilatietojen käsittelyn valvomolle haluttuun muotoon, laitetyypille ominaiset automaattiohjaukset ja niihin vaikuttavat asetukset, sekä valvomolta tulevat käsiohjaukset. Lisäksi lohkoon on integroitu laitteen toiminta simulointitilassa.

7.2.3 Kirjasto

PLC-Kirjasto on lohkojen ja tietotyyppien hallintaan tarkoitettu työkalu. Valmiit komponentit viedään kirjastoon, josta ne voidaan hakea toiseen projektiin. Kir-

jastoa ylläpidetään versionhallinnan keskitetyllä palvelimella, jolloin se on kaikkien kehittäjien saatavilla. Lisäksi versionhallinnan lokien avulla voidaan tarkastella komponentteihin tehtyjä muutoksia ja tarvittaessa perua lohkoihin tehtyjä päivityksiä.

Kirjaston ideaalisena tavoitteena on sisältää geneeriset komponentit, joiden pohjalta on valmius toteuttaa täysin kokonaisen tunneliprojektin kaikki laitteet kattava PLC-ohjelmisto.

7.2.4 Ohjelman rakenne

Jokaisen laitetyypin johdetusta tietotyypistä muodostetaan muuttujataulukko, jonka koko määräytyy kyseisen laitetyypin laitteiden määrän mukaisesti (Taulukko 2).

Taulukko 3. Laitetyypin muuttujataulukoiden rakenne

	deviceIn	deviceOut	scadaStatus	scadaControl
Laite 1	deviceIn[0]	deviceOut[0]	scadaStatus[0]	scadaControl[0]
Laite 2	deviceIn[1]	deviceOut[1]	scadaStatus[1]	scadaControl[1]
…	…	…	…	…
Laite n	deviceIn[n]	deviceOut[n]	scadaStatus[n]	scadaControl[n]

Laitteiden ohjelmointiin tarvitaan pääasiassa jokaiselle laitetyypille kaksi osiota. IoCopy_laitetyyppi-osioissa kutsutaan IoCopyBlock-lohkot ja Control_laitetyyppi-osiossa ControlBlock-lohkot jokaiselle saman laitetyypin laitteelle.

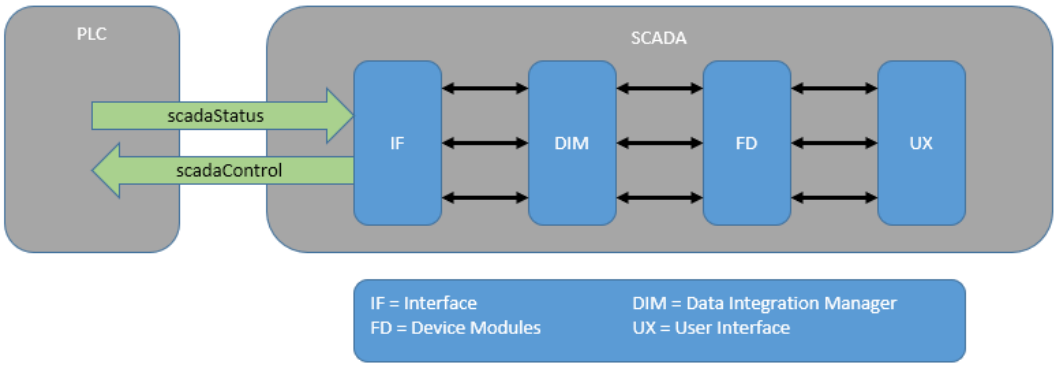
Ohjelman simulointi perustuu vahvasti uudistetun arkkitehtuurin tuomiin etuihin. Simulointitilassa IoCopy-osiot kytketään pois ohjelmakierrosta, minkä seurauksena muu ohjelmisto irtautuu IO-tasosta. Tämän ansiosta ohjelmistoa voidaankin suorittaa simulointitilassa jo projektin alkuvaiheessa, vaikka laitteiden tuloja ja lähtöjä ei olisikaan määritetty. Simulointitilaa ohjataan simulointimuuttujalla, jonka tilatieto viedään kaikille ohjelman laitteiden ohjauslohkoille. Ohjauslohkoihin integroidun simulointiosion ohjelma suoritetaan, kun simulointi on päällä.

7.3 SCADA-ohjelmistokehitys

SCADA-ohjelmistokehityksessä käytetään Siemensin SIMATIC WinCC Open Architecture -sovellusta. Valvomon ohjelmointi toteutetaan manuaalisesti kirjoitetuilla Control Script -ohjelmakoodeilla. Script Wizard -työkalua ei käytetä, koska käsin kirjoittamalla ohjelmakoodista saadaan huomattavasti selkeämpää.

7.3.1 Arkkitehtuuri

SCADA-ohjelmiston uudistetussa arkkitehtuurissa valvomon sisäinen rakenne on standardisoitu ja erotettu PLC-rajapinnasta erillisellä moduulilla, jolla tasojen väliset datapointit yhdistetään (kuva 13).



Kuva 13. SCADA-ohjelmiston uudistettu arkkitehtuuri

Rajapintana toimivan IF-tason datapointit voidaan generoida UnityManager-työkalulla suoraan PLC-ohjelmasta tuodun muuttujataulukon pohjalta ja niiden päärakenne on yhden laitetyypin kaikille laitteille sama. FD-tason datapointtien rakenne on standardisoitu, eli se on kaikilla ohjelmiston laitteilla aina samanlainen. Data Integration Manager -moduuli linkittää IF-tason datapointit FD-tason datapointteihin. Kaikille graafisen käyttöliittymän (UX) laitteille käytetään samaa laitepohjaa, johon lisätään toiminnot laitetyyppikohtaisilla moduuleilla.

7.3.2 Moduulit

Moduulit ovat SCADA-ohjelmiston tiedostoista koostuvia paketteja, jotka muodostavat ohjelmistoon jonkin toiminnon, ominaisuuden tai osakokonaisuuden.

Moduulien tarkoituksena on yksinkertaistaa tuotetun ohjelmakoodin uudelleenkäyttöä. Kehitetyt moduulit tallennetaan versionhallinnassa ylläpidettävään kirjastoon. Moduulien hallintaa varten on kehitetty erilliset työkalut.

ModuleCreator on työkalu, jolla voidaan hallita kirjastossa olevia moduuleja. ModuleCreatorilla voidaan luoda uusia moduuleja tai muokata vanhoja, sekä julkaista niitä kirjastoon.

ModuleManager on työkalu, jolla voidaan graafisen käyttöpaneelin kautta hallita projektin moduuleja. Paneelista näkee listattuna kaikki kirjastoidut moduulit, niiden versiot ja kuvaukset. Paneelin kautta voidaan asentaa, poistaa ja päivittää projektin moduuleja kirjastosta.

Moduulikirjaston käyttöohjeessa käsitellään tarkemmin työkalujen ominaisuuksia ja toimintoja. Lisäksi ohjeessa käsitellään kirjaston sisältöä, moduulien rakennetta ja tärkeimpiä moduuleja. Arkkitehtuurillisesti oleellisia moduuleja ovat UnityManager, Data Integration Manager ja Laitemoduulit.

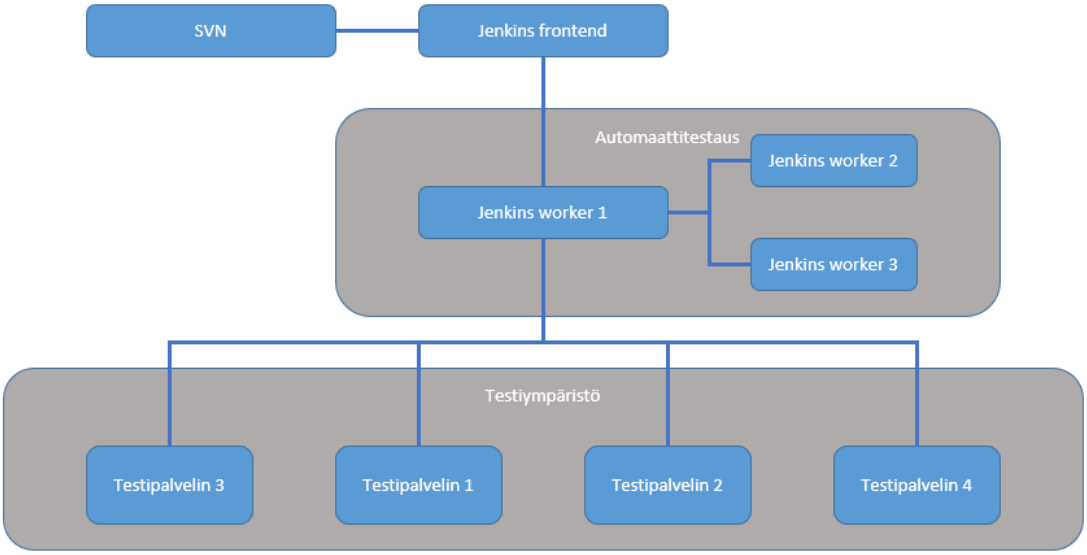
UnityManager on työkalu, jolla voidaan generoida valvomon PLC-rajapinnan datapointit. Unity Pro -sovellukseen määritetyt muuttujat viedään Export-toiminnolla XML-tiedostoon. UnityManager parsii tiedostosta tarpeelliset tiedot, joiden avulla rajapinta generoidaan.

Data Integration Manager on integraatiomoduuli, jolla yhdistetään rajapintana toimiva IF-taso ja graafisen käyttöliittymän pohjana toimiva FD-taso. DIM-moduuli voidaan konfiguroida erillisellä JSON-tiedostolla.

Laitemoduuli on laitetyyppikohtainen moduuli, joka sisältää laitteen ominaisuudet ja toiminnot. Laitemoduulin avulla luodaan kaikki laitetyypin laitteet graafiseen käyttöliittymään.

7.4 Integraatioympäristö

YSP:n integraatioympäristö rakentuu kaikkiaan yhdeksästä palvelimesta (kuva 14). Kuvassa harmaalla on merkitty integraatiotesteihin käytettävät palvelinryhmät ja sinisellä yksittäiset palvelimet.



Kuva 14. YSP:n integraatioympäristön palvelimet

Tällä hetkellä käytössä on keskitetty versionhallintajärjestelmä Apache Subversion. SVN-asiakasohjelmaksi suositellaan avoimen lähdekoodin TortoiseSVN-sovellusta. Tulevaisuudessa tullaan mahdollisesti siirtymään hajautettuun versionhallintajärjestelmään Git:iin.

Automaatiopalvelimena toimii avoimen lähdekoodin Jenkins, jonka käytössä on yhteensä neljä palvelinta. Jenkins frontendillä on järjestelmän graafinen käyttöliittymä, johon voidaan ottaa yhteys selaimella, ja se koordinoi worker-palvelinten toimintaa. Worker 1 suorittaa varsinaiset automaatiopalvelimelle määritetyt tehtävät ja Workerit 2-3 toimivat suorittivat logiikkaohjelmistoja simulaattoreissa.

Testiympäristö koostuu neljästä palvelimesta. Palvelimilla 3-4 suoritetaan logiikkasimulaattoreita ja palvelimilla 1-2 niiden valvomo-ohjelmistoja. Testiympäristö on tarkoitettu kehittäjien käyttöön ohjelmistojen testaukseen simulointitilassa. Testipalvelimia voidaan käyttää yrityksen sisäisessä verkossa RDP-etätyöpöytäyhteydellä.

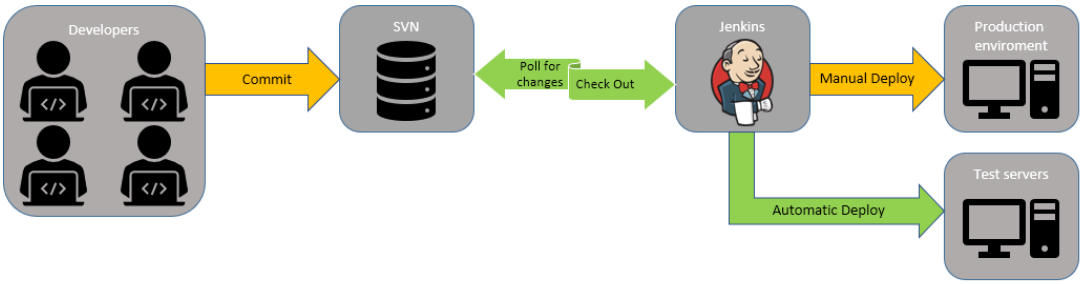
7.5 Kehitysprosessi

YSP:n ohjelmistokehityksessä sovelletaan scrumia, joka on ketterän kehityksen viitekehys. Tavoitteena on luoda tehdyllä työllä kokonaan valmiita osakokonaisuuksia, mutta kehitysprosessille ei kuitenkaan ole vielä löytynyt täysin

vakiintunutta mallia, vaan se muuttuu ja kehittyy vielä. Scrumin yhtenä keskeisenä elementtinä olevien kehitysjonojen hallintaan käytetään Atlassianin Jira-alustaa. Jirassa kehitysjonolle lisättäviin tehtäviin merkitään tehtävän osa-alue, jonka perusteella tehtävät jakaantuvat kehittäjille.

Tavoitteena on, että kehittäjät varaavat kehitysjonolta aina uuden oman osa-alueensa mukaisen tehtävän, joka on prioriteetiltaan korkeimpana. Tehtävät jaetaan vielä pienemmiksi osatehtäviksi, jotka ovat selkeästi määritettäviä osia tehtävästä. Sprintillä on tarkoitus suorittaa pääasiassa vain kehitysjonolta varattuja tehtäviä ja näiden osatehtäviä.

Kehittäjien henkilökohtaisena työtilana toimii SVN:stä tuotu paikallinen työkopio. Tuotettujen inkrementtien työnkulku integraatioympäristössä on esitetty kuvassa 15. Harmaat laatikot ovat prosessin toiminnallisia osia, vihreät nuolet automaattisesti osien välillä tapahtuvia siirtymiä, ja oranssit nuolet manuaalisia siirtymiä.



Kuva 15. Kehitysprosessi integraatioympäristössä

Työkopioon tuotettu inkrementti viedään keskitettyyn varastoon. Vientioperaation kommenttikenttään tulee Jiran tehtävätunnus ja englanninkielinen kuvaus viennin syystä ja tehdyistä muutoksista. Tehtävätunnuksen avulla SVN-loki saadaan Jirassa näkyviin tehtävän tietoihin.

Jenkins kyselee määritetyn ajan välein SVN:ltä muutoksia ja tuo ne tarvittaessa paikalliseen työkopioonsa. Jenkinsiin luodaan ohjelmistoille projektikohtaisesti omat tehtävät, joiden mukaan se suorittaa automaattisesti määrättyjä toimintoja. Tehtävien määritykset voidaan pääosin kopioida aiemmalta projektia muokaten tehtävän kuvaus ja projektin polku SVN:ssä.

PLC-projektille Jenkins ei varsinaisesti suorita mitään toimintoja, vaan se arkistoidaan ja julkaistaan, jonka jälkeen Jenkins ajaa ohjelman testipalvelimille.

SCADA-projektille Jenkins suorittaa koontiskriptit, joiden tuotoksena saadaan SCADA-ohjelmiston koontiversiot eri työpisteille. Valvomo käynnistyy testiautomaatiopalvelimelle ja suorittaa ohjelmoidut automaattitestit uusinta simulaattoriin ladattua PLC-ohjelmistoa vasten. Testauksen jälkeen projekti arkistoidaan ja julkaistaan testituloksiin perustuvalla luokituksella. Testeistä onnistuneesti suoriutunut ohjelmisto ajetaan testipalvelimille.

8 TYÖN TULOKSET

8.1 Dokumentaatio

Opinnäytetyön tuloksena saatiin yrityksen sisäiseen käyttöön tarkoitettuja dokumentteja; WinCC OA -moduulien käyttöohje, Unity Pro XL -kirjaston käyttöohje ja Unity Pro XL -perehdytys. Dokumentaation ensisijaisena tarkoituksena on esim. perehdyttää täysin uusi tai eri työtehtävissä ollut työntekijä YSP:n kehittämän CSF-standardin mukaiseen PLC-ohjelmointiin Unity Pro XL -sovelluksella ja opastaa PLC- ja SCADA-kirjastojen käytössä.

WinCC OA -moduulien käyttöohjeessa kuvaillaan yleisesti mitä moduulit ovat, miten ne rakentuvat ja miten moduulikirjasto muodostuu. Lisäksi käydään läpi moduulien käyttöön tarvittavat työkalut sekä niiden ominaisuuksia ja toimintoja. Lopuksi ohjeistetaan moduulien luominen ja käyttäminen ohjelmistossa vaiheittain.

Unity Pro XL -kirjaston käyttöohjeessa käydään aluksi vaiheittain läpi tietotyyppien ja lohkojen muodostaminen. Tämän jälkeen ohjeistetaan kirjaston käyttöönotto, sekä tutustutaan kirjaston rakenteeseen, sisältöön ja käyttöön. Kirjaston käyttö ohjeistetaan vaiheittain eri tilanteissa. Lopuksi on ohjeet versionhallinnan käytöstä kirjaston yhteydessä.

Unity Pro XL -perehdytyksessä opastetaan yksinkertaisen PLC-ohjelman toteutus CSF:n mukaisesti Unity Pro XL -sovelluksella. Perehdytettävä toteuttaa

ohjeiden mukaisia komponentteja, joista lopulta rakentuu toimiva ohjelma, joka julkaistaan YSP:n integraatioympäristössä. Perehdytystä varten kehitettiin malliksi PLC-ohjelma, jonka mukaan ohjeistus on toteutettu.

8.2 PLC-ohjelman versiointi

Nykyinen menetelmä PLC-ohjelman versiotietojen tallentamiseen ei sisällä haluttuja ominaisuuksia. Versiotietojen tallennuksen tavoitteena on selkeyttää tuotannossa suoritettavan ohjelman jäljentämistä vastaavaan versioon yrityksen versionhallinnassa. Versiotiedot halutaan saada näkyviin versionhallintaan ja ohjelman sisäisen muuttujan avulla valvomoon.

Käytössä oleva Unity Pro -ohjelmiston sisäänrakennettu ominaisuus päivittää ohjelman versiota aina kun se buildataan. Ohjelmaa joudutaan kuitenkin buildaamaan esimerkiksi käyttöönottotestauksessa, jolloin versio päivittyy, vaikka ohjelmaa ei varsinaisesti muutetakaan. Tästä johtuen tuotantoympäristössä olevan PLC-ohjelman versio ei yleensä vastaa versionhallinnasta löytyviä versioita. Ohjelman versionumero ei myöskään kasva loogisesti ohjelman kehittyessä. Lisäksi sisäänrakennetun ominaisuuden tallettamia versiotietoja ei pääse lukemaan avaamatta sovellusta.

Versioinnin parantamiseksi kehitettiin mahdollista ratkaisua, jonka tavoitteena oli syöttää PLC-ohjelman sisäisiin muuttujiin tietoa Windowsin komentorivillä suoritettavan skriptin avulla. Unity Pro -ohjelmisto tarjoaa kuitenkin hyvin rajalliset mahdollisuudet suorittaa toimintoja komentoriviltä, eikä tiedon vieminen suoraan komentoriviltä ohjelmaan ollut mahdollista.

8.2.1 Versioinnin kehitys

Versiotietojen tuonti ohjelmaan toteutettiin kehittämällä lohko, jolla luetaan PLC:n muistikortilla olevan tiedoston sisältö ja jaetaan se ohjelman sisäisiin muuttujiin tämänhetkisiksi arvoiksi. Ohjelman suorittaminen tietokoneella simulaattorissa käyttää näennäismuistikorttia, jonka polku paikallisella levyllä voidaan määrittää Windowsin rekisteriin.

Ongelma oli luettujen tietojen tallentaminen ohjelmaan pysyvästi. Tietojen lukemisen jälkeen ne tallentuivat vain muuttujien tämänhetkisiksi arvoiksi, eli ne

katosivat, kun ohjelman suoritus aloitettiin alusta. Tämänhetkiset arvot voidaan tallentaa lähtöarvoiksi asettamalla määrätty järjestelmäbitti päälle. Operaatio tehdään vain muuttujille, joille on määritetty ohjelmassa ”Save”-asetus päälle.

Haastavinta ratkaisussa oli versioinnin toteutus täysin automaattisesti komentorivillä suoritettavalla skriptillä. Prosessin automatisoinnissa hyödynnettiin Schneiderin UIUmas-työkalua, joka toimii rajapintana Quantum ja Premium PLC alustoille. UIUmas voidaan käynnistää ja sillä voidaan suorittaa määrättyjä perustoimintoja Windowsin komentoriviltä. Ratkaisun kannalta oleellisia toimintoja olivat ohjelman lataaminen simulaattoriin, suorittaminen ja lataaminen simulaattorista koneelle.

8.2.2 Tuotetut ratkaisut

Kehityksen aikana versioinnille toteutettiin kolme erilaista toimintamallia, joista ensimmäinen oli täysin automaattinen ja se toteutettiin automaatiopalvelimella. Kun versionhallintaan ladataan uusi versio PLC-ohjelmasta, automaatiopalvelin tuo sen paikalliselle levylle ja suorittaa määritetyt toiminnot. Automaatiopalvelimelle määritetään komentoriviltä suoritettava skripti, jolla ohjelmaan lisätään versiotiedot ja viedään sen jälkeen versionhallintaan.

PLC-ohjelmaa ei kuitenkaan aina ladata tuotantoon integraatioympäristön kautta, joten tarvittiin toimintamalli, joka toimii kehittäjän koneella. Ensimmäinen kehittäjän koneella suoritettava versiointi oli myös lähes täysin automaattinen. Ohjelman versiointi suoritetaan skriptillä, jonka nimi täytyy vaihtaa aina vastaamaan ohjelman nimeä. Skripti suorittaa ensin ohjelman versioinnin simulaattorissa ja avaa ohjelman sen jälkeen normaalisti Unity Pro -ohjelmistolla.

Versioinnin suorittaminen aina ohjelmaa käynnistäessä kesti kuitenkin huomattavan pitkään ja sen todettiin hidastavan sovelluksen käynnistystä. Ongelmasta päästiin eroon luopumalla osittain prosessin automaattisuudesta. Kolmannessa toimintamallissa suoritetaan tietokoneen käynnistyksen yhteydessä skripti, joka kirjoittaa versiotiedot näennäismuistikortin käyttämään polkuun.

Kehittäjä suorittaa ohjelman simulaattorissa, jolloin ohjelma lukee käynnistyksen yhteydessä luodut versiotiedot ja asettaa ne simulaattoriin lähtöarvoiksi. Lopuksi kehittäjä klikkaa valikosta toimintoa, joka siirtää simulaattorin lähtöarvot ohjelmaan.

LÄHTEET

YSP. s.a. YSP kotivisut. WWW-sivusto. Saatavissa: https://www.ysp.fi/ [viitattu 7.4.2019].

Vilkman, V. 2010. Jatkuva integraatio ohjelmistokehityksessä. PDF-dokumentti. Saatavissa: https://jyx.jyu.fi/bitstream/handle/123456789/24657/Vesa.Vilkman.pdf?sequence=1&isAllowed=y [viitattu 10.3.2019].

Dynniq kotisivut. s.a. Dynniq. WWW-sivusto. Saatavissa: https://dynniq.com/ [viitattu 7.4.2019].

Dynniq sharepoint. s.a. Dynniq. Sisäinen WWW-sivusto. [viitattu 13.4.2019].

Somasundaram, R. 2013. Git: Version Control for Everyone. E-kirja. Saatavissa: https://books.google.fi/books?id=mD0YT2dcFQkC&printsec=copyright&hl=fi&source=gbs_pub_info_r#v=onepage&q&f=false [viitattu 20.4.2019]

RhodeCode. s.a. Version Control Systems Popularity in 2016. WWW-sivu. Saatavissa: https://rhodecode.com/insights/version-control-systems-2016 [viitattu 22.4.2019]

Collins-Sussman, B., Fitzpatrick, B. & Pilato, C. 2011. Version Control with Subversion. E-kirja. Saatavissa: http://svnbook.red-bean.com/en/1.7/svn-book.pdf [viitattu 22.4.2019]

Chacon, S. & Straub, B. 2014. Pro Git. E-kirja. Saatavissa: https://link.springer.com/content/pdf/10.1007%2F978-1-4842-0076-6.pdf [viitattu 22.4.2019]

Sink, E. 2011. Version Control by Example. E-kirja. Saatavissa: https://ericsink.com/vcbe/html/index.html [viitattu 25.4.2019]

fournova. s.a. Learn Version Control with Git. E-kirja. Saatavissa: https://www.git-tower.com/learn/git/ebook/en/desktop-gui/introduction [viitattu 28.4.2019]

Agile Alliance. s.a. Agile 101. WWW-sivu. Saatavissa: https://www.agilealliance.org/agile101/ [viitattu 22.5.2019]

Unitronics. s.a. What is PLC. WWW-sivu. Saatavissa: https://unitronicsplc.com/what-is-plc-programmable-logic-controller/ [viitattu 22.5.2019]

Schneider Electric. 2011. Unity Pro Program Languages and Structure Reference Manual. PDF-dokumentti. Saatavissa: http://users.isr.ist.utl.pt/~jag/courses/api13/docs/PLC_2_Unity_Reference.pdf [viitattu 22.5.2019]

Sielco Sistemi. s.a. What is SCADA. WWW-sivu. Saatavissa: https://www.sielcosistemi.com/en/what-is-scada.html [viitattu 28.5.2019]

Gupta, K. 2016. Introduction to SCADA system & Security. WWW-dokumentti. Saatavissa: https://www.slideshare.net/Kunalgupta137/introduction-to-scada-63680926 [viitattu 29.5.2019]

Automation Excellence. s.a. What is SCADA? WWW-sivu. Saatavissa: http://www.automationexcellence.com/scada.html [viitattu 29.5.2019]

Ernst, M. 2018. Version control concepts and best practices. html-dokumentti. Saatavissa: https://homes.cs.washington.edu/~mernst/advice/version-control.html [viitattu 30.5.2019]

Shore, J. & Warden, S. 2008. The Art of Agile Development. E-kirja. Saatavissa: https://poetiosity.files.wordpress.com/2011/04/art_of_agile_development.pdf [viitattu 30.5.2019]

Fowler, M. 2006. Continous Integration. WWW-artikkeli. Saatavissa: https://martinfowler.com/articles/continuousIntegration.html [viitattu 30.5.2019]

Smartsheet. s.a. Comprehensive Guide to the Agile Manifesto. WWW-sivu. Saatavissa: https://www.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto [viitattu 1.6.2019]

Scrum.org. s.a. What is scrum. WWW-sivu. Saatavissa: https://www.scrum.org/resources/what-is-scrum [viitattu 1.6.2019]

Schwaber, K & Sutherland, J. 2017. Scrum-opas. PDF-dokumentti. Saatavissa: https://scrumwell.files.wordpress.com/2018/03/2017-scrum-guide-fi-v1-02.pdf [viitattu 1.6.2019]

IEC 61131-3. 2003. PDF-dokumentti. Saatavissa: http://d1.amobbs.com/bbs_upload782111/files_31/ourdev_569653.pdf [viitattu 2.6.2019]