jamk.fi

Toimilohko- ja kuvageneraattori Siemens TIA Portal -ohjelmointityökaluun

Matias Ruha

Opinnäytetyö
Toukokuu 2019
Tekniikan ala
Insinööri (AMK), sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Ruha, Matias	Opinnäytetyö, AMK	Toukokuu 2019
		Julkaisun kieli Suomi
	Sivumäärä 47+3	Verkkojulkaisulupa myönnetty: x

Työn nimi

Toimilohko- ja kuvageneraattori Siemens TIA Portal -ohjelmointityökaluun

Tutkinto-ohjelma

Insinööri (AMK), sähkö- ja automaatiotekniikan tutkinto-ohjelma

Työn ohjaaja(t)

Veli-Matti Häkkinen, Markku Ström

Toimeksiantaja(t)

PCS-Engineering Oy, PCS-Services Oy

Tiivistelmä

Siemens TIA Portal -ohjelmointityökalun käyttöä on pyritty tehostamaan ja parantamaan jatkuvasti niin, että työvaiheisiin kuluvaa aikaa voidaan vähentää tinkimättä työn laadusta. TIA Openness -ohjelmointirajapinta on yksi versiopäivityksien tuomista uusista ominaisuuksista. Se mahdollistaa omien sovelluksien liittämisen TIA Portal -ohjelmointityökaluun.

Toimilohkojen, toimilohkojen datalohkojen, faceplatejen ja toimilohkoikonien tekeminen, sekä tagien yhdistäminen ja kommentoiminen on nykyisillä menetelmillä käsin tehtävä työvaihe, joka toistetaan useita kertoja. Työvaihe on hidasta toistotyötä ja suunnittelijan siihen käyttämä työaika on pois sovellussuunnittelun tärkeämmistä työvaiheista. Työvaiheessa muodostuu myös paljon kirjoitusvirheitä.

Opinnäytetyön tavoitteena oli kehittää TIA Openness -ohjelmointirajapintaa käyttävä toimilohko- ja kuvageneraattori, jolla voidaan generoida ja automatisoida edellä mainittu työvaihe. Generoinnille sekä generaattorille asetettiin erilliset tavoitteet.

Toimilohko- ja kuvageneraattori kehitettiin Visual Studio -kehitysympäristössä. Generaattorin toiminnot ohjelmoitiin VB.NET -ohjelmointikielellä ja sille tehtiin käyttöliittymä. Generaattorin käyttöä testattiin biokaasulaitosprojektin toimilohkojen generoinnissa ja testiprojektissa. Testiprojektissa tehtiin 30:n piirin kaikki työvaiheet käsin ja generaattorilla. Suunnittelija kellotti molempien menetelmien aikana työvaiheisiin kuluvan ajan. Lopuksi tuloksia verrattiin toisiinsa.

Testaustuloksien perusteella generaattorilla saadaan säästettyä suunnittelijan työaikaa mittavasti ja vähennettyä virheiden määrää.

Avainsanat (asiasanat)

TIA Openness, TIA Portal, generoida, toimilohko, faceplate, toimilohkoikoni

Miscellaneous (Confidential information)

Liitteet 1-3 ovat salassa pidettäviä, joten ne on poistettu julkisesta työstä. Salassapidon peruste Julkisuuslain 621/1999 24§, kohta 17, yrityksen liike- tai ammat-tisalaisuus. Salassapitoaika viisi (5) vuotta, salassapito päättyy 8.5.2024.

jamk.fi

Description

Author(s)	Type of publication	Date
Ruha, Matias	Bachelor's thesis	May 2019
		Language of publication: Finnish
	Number of pages 47+3	Permission for web publication: x

Title of publication

Function block and screen generator for Siemens TIA Portal programming tool

Degree programme

Electric and Automation Engineering

Supervisor(s)

Häkkinen, Veli-Matti; Ström, Markku

Assigned by

PCS-Engineering Oy, PCS-Services Oy

Abstract

The usage of Siemens TIA Portal programming tool has been optimized and improved continuously so that work time can be reduced without compromising on the quality of work. TIA Openness application programming interface is one of the new features of version updates. It allows users to connect their own applications to the TIA Portal programming tool.

Creating, tagging and commenting function blocks, instance data blocks, faceplates and function block icons is a manual process repeatedly performed multiple times with the current methods. The work phase is slow repetitive work and the working time used by the designer is away from the more important work phases of application design. There are also many spelling mistakes created in the work phase.

The goal of the thesis was to develop a function block and screen generator using the TIA Openness application programming interface to generate and automate the work phase mentioned above. Separate goals were set for generation and generator.

The function block and screen generator were developed in a Visual Studio integrated development environment. The generator was programmed in VB.NET programming language and it was provided with an interface. The usage of the generator was tested in the biogas plant project and test project. The generator was used to generate all necessary function blocks in the biogas plant project. In the test project, designer performed every work phase for 30 circuits by hand and with generator. During both methods, the designer clocked the time used for each work phase. Lastly, the results were compared. Based on the test results, the generator can save designer's working time considerably and reduce the number of errors.

Keywords/tags (subjects)

TIA Openness, TIA Portal, generate, function block, faceplate, function block icon

Miscellaneous (Confidential information)

Appendixes 1-3 are confidential which have been removed from the public thesis. Grounds for secrecy: Act on the Openness of Government Activities 621/1999, Section 24, 17: business or professional secret. Period of secrecy is five years and it ends 8.5.2024.

Sisältö

Kuviot

1 Johdanto

1.1 Opinnäytetyön tausta ja tavoite

Siemens TIA Portal -ohjelmointityökalu on laajassa käytössä PCS-Engineering Oy:n sovellussuunnittelijoilla. Sen käyttöä on pyritty tehostamaan ja parantamaan jatkuvasti, jotta työn laatu pysyy korkeana ja työaikaa säästyisi eri työvaiheissa. Uusien versiopäivityksien mukana on tullut uusia ominaisuuksia, joihin perehtyminen ja joiden hyödyntäminen on kannattavaa tehokkaan työskentelyn kehittämiseksi ja ylläpitämiseksi. TIA Openness -ohjelmointirajapinta on yksi versiopäivityksien tuomista uusista ominaisuuksista. Sen avulla on mahdollista tehdä omia sovelluksia, jotka keskustelevat TIA Portal -ohjelmointityökalun kanssa. Toimeksiantaja halusi selvittää TIA Openness -ohjelmointirajapinnan hyötyjä ja mahdollisuuksia. Lyhyen selvityksen jälkeen kävi ilmi, että sillä on mahdollista tehdä muun muassa generointia. Siemens ei tarjoa valmista työkalua tai ratkaisua generointiin.

Uuden projektin alkaessa TIA Portal -ohjelmointityökalulla on jokaiselle moottoriohjaukselle, venttiiliohjaukselle ja mittaukselle luotava toimilohko, toimilohkon datalohko, toimilohkoikoni ja faceplate. Toimilohko kommentoidaan ja siihen yhdistetään tageja. Seuraavaksi toimilohkoikoni yhdistetään toimilohkoon kolmella tagilla, minkä jälkeen faceplate voidaan puolestaan yhdistää toimilohkoikoniin. Edellä mainitut työvaiheet ovat nykyisillä menetelmillä toistotyötä, joka on hidasta. Siihen käytetty työaika on pois tärkeämmistä sovellussuunnittelun työvaiheista. Myös virheiden määrä suurenee, kun työvaiheita on tehtävä käsin. Generointia voidaan hyödyntää toistuvissa työvaiheissa. Sillä voidaan vähentää toistettavan työn määrää, mikä puolestaan vähentää suunnittelijan siihen käyttämää aikaa. Toistuvien työvaiheiden automatisointi on tärkeää, sillä suunnittelija voi siirtyä tärkeämpiin sovellussuunnittelun työvaiheisiin nopeammin. Sovellussuunnittelu kokonaisuutena nopeutuu ilman resurssien lisäämistä.

Toimeksiantaja halusi TIA Openness -ohjelmointirajapinnalla toteutetun generaattorin, jolla toimilohkot, toimilohkojen datalohkot, toimilohkoikonit ja faceplatet saataisiin generoitua Excel-taulukosta TIA Portal -projektiin. Opinnäytetyön tavoitteena oli kehittää generaattori toistuvien työvaiheiden, siihen käytettävän työajan ja virheiden vähentämiseksi. Ongelman pohjalta muodostettiin tutkimuskysymyksiä.

Generaattorille asetettiin lisäksi omat tavoitteet. Sen oli toimittava kolmella eri ohjelmointityökalun versiolla. Generoinnin tulisi onnistua kaikenlaisilla toimilohkoilla eikä vain tyyppilohkokirjaston valmiilla toteutuksilla. Sen sijaan toimilohkoikonit ja faceplatet generoitaisiin tyyppikirjaston toteutuksista. Generaattorin tulisi olla käytettävyydeltään yksinkertainen ja nopea. Generointilista tulisi olla Excel-taulukkolaskentaohjelman formaatissa. Generoitavien toimilohkojen, toimilohkoikonien ja faceplatejen tulisi olla käyttövalmiita sovellukseen.

Henkilökohtaisena tavoitteena oli oppia enemmän TIA Openness -ohjelmointirajapinnan mahdollisuuksista automatisoida työvaiheita, Visual Basic ohjelmointia ja TIA Portal -ohjelmointityökalun tehokkaampaa käyttöä.

1.2 Aiheen rajaus

Ohjelmointityökaluksi rajattiin Siemensin kehittämä TIA Portal ja sen lisäosa TIA Openness. TIA Portal -toimilohkojen generoinnille ei ollut löytynyt valmista ratkaisua. TIA Openness ei tue käyttöliittymän suunnitteluun käytettävää SIMATIC WinCC Professional -versiota. Opinnäytetyössä ja siinä mainituissa projekteissa on käytetty SIMATIC WinCC Advanced -versiota. Generaattoriin kehitysympäristöksi valittiin Visual Studio ja ohjelmointikieleksi Visual Basic.NET oman kokemuksen ja Siemensin TIA Openness -ohjelmointirajapinnan edellytyksien takia. Excel-tiedoston formaatti rajattiin Excel-työkirja (.xlsx) muotoon.

1.3 Toimeksiantaja PCS-Engineering Oy

Opinnäytetyön toimeksiantaja oli PCS-Engineering Oy, joka on perustettu Oulussa vuonna 2004. Sen tytäryhtiö PCS-Services Oy on perustettu Jyväskylässä vuonna 2009 ja sijaitsee Mattilanniemessä. PCS-Engineering Oy on alallansa keskisuuri asiantuntijapalveluyritys ja tarjoaa asiakkailleen teollisuuden insinööritoimistopalveluita. Liiketoiminta muodostuu sähkö-, automaatio- ja instrumentointiteknisistä palveluista, kuten esimerkiksi sähkö-, automaatio-, prosessi- ja sovellussuunnittelusta. Toiminta prosessiteollisuudessa on laajaa. Vuoden 2017 liikevaihdosta 34 % oli peräisin sellu- ja paperiteollisuudesta, 23 % energiateollisuudesta ja 23 % metalliteollisuudesta. (Palvelut 2019; Project management Consulting Services. 2018.)

Asiakkaana on erikokoisia yrityksiä eri aloilta. Suurimmat asiakkaat tulevat sellu- ja paperiteollisuudesta, energiateollisuudesta ja metalliteollisuudesta. Pienempiä asiakkaita ovat esimerkiksi maatilat ja vedenpuhdistamot. PCS-Engineering Oy tarjoaa palveluitaan kansainvälisesti. Projekteja tehdään alihankintana ja myös suoraan asiakkaalle. (Palvelut 2019; Project management Consulting Services. 2018.)

Henkilöstö koostuu noin 50:stä ammattilaisesta, joista suurin osa on sähkö- ja automaatiotekniikan insinöörejä. Henkilöstöstä noin 30 työskentelee Oulun toimistossa ja noin 20 Jyväskylän toimistossa. (Palvelut 2019; Project management Consulting Services. 2018.)

1.4 Tutkimuskysymykset ja -menetelmät

Opinnäytetyön tutkimuskysymykset olivat seuraavat:

- Voiko TIA Openness -ohjelmointirajapinnan avulla toteutetulla generaattorilla generoida täysin valmiita toimilohkoja, toimilohkoikoneita ja faceplateja käytettäväksi sovellussuunnittelun seuraavissa työvaiheissa?
- Saadaanko generoinnilla vähennettyä selvästi suunnittelijan toistotyötä ja siihen käytettyä työaikaa?
- Saadaanko generoinnilla vähennettyä virheiden määrää?

Tutkimuskysymyksiin etsittiin ratkaisua tutkimuksen aikana suunniteltavalla ja ohjelmoitavalla generaattorilla ja yrityksen muiden suunnittelijoiden antamalla loppupalautteella generaattorin käytöstä.

Opinnäytetyössä käytettiin kehittämistutkimuksen tutkimusmenetelmiä. Kehittämistutkimus on monimenetelmällinen tutkimusmenetelmä. Siinä yhdistyvät kvalitatiiviset ja kvantitatiiviset tutkimusmenetelmät. Kehittämistutkimus voidaan toteuttaa myös käyttäen vain kvalitatiivista tutkimusmenetelmää. (Kananen 2015, 76; Kananen 2012, 19-26.) Kehittämistutkimuksessa on tarkoitus kehittää kohdetta ja saada aikaan muutos tai parannus. Kehitettävä kohde voi olla esimerkiksi tuote tai menetelmä. (Kananen 2015, 76; Kananen 2012, 13, 43.)

Tässä työssä käytettiin vain kvalitatiivista tutkimusmenetelmää. Kvantitatiivisella tutkimusmenetelmää käytetään, kun tutkimuksessa käsitellään lukumääriä, prosenttiosuuksia tai muita numeerisia tutkimusaineistoja. (Heikkilä 2014, 6-10; Vilpas 2013.) Työssä ei tarvittu kvantitatiivista tutkimusmenetelmää, koska numeerisia tutkimusaineistoja ei käytetty. Tässä työssä kehitettiin generaattori, jota ei aikaisemin ole ollut toimeksiantajan käytössä, ja tutkittiin sen vaikutuksia suunnittelijan toistotyön määrään, siihen käytettyyn aikaan ja siinä muodostuvien virheiden määrään.

Kvalitatiivisessa eli laadullisessa tutkimusmenetelmässä tutkimuskohteena oleva ilmiö pyritään ymmärtämään. (Kananen 2015, 128; Kananen 2012, 29.) Laadullisen tutkimuksen tärkeimmät aineistonkeräämisen menetelmät ovat havainnointi, haastattelu ja dokumentit. (Kananen 2015, 81; Kananen 2012, 93.) Tutkimusaineistoa käytetään ongelman ratkaisemiseksi. Aineistoa kerätään, kunnes ongelmaan on saatu ratkaisu. (Kananen 2015, 128)

Tämän työn luotettavuuden mittareina käytettiin reliabiliteettia ja validiteettia. Reliabiliteetilla tarkoitetaan tutkimustulosten pysyvyyttä eli sitä, ovatko tulokset toistettavissa. Validiteetilla puolestaan tarkoitetaan sitä, onko tutkimuksessa tutkittu oikeita asioita ja ovatko tehdyt päätökset oikeita. (Kananen 2015, 343; Kananen 2012 167-168; Saaranen-Kauppinen & Puusniekka 2006, 25-26.)

2 Aineiston kerääminen ja analyysi

2.1 Aineiston kerääminen

Tässä työssä primääriaineisto kerättiin kentältä havainnoinnilla ja haastatteluilla. Sekundääriaineistoa olivat erilaiset ohjelmointityökalun valmistajan ja toimeksiantajan aikaisempien projektien dokumentit.

Havainnointi tapahtuu reaaliajassa. Sitä voidaan tehdä tuntematta ilmiötä tai suunnitellusti tarkistuslistan avulla. (Kananen 2015, 134; Saaranen-Kauppinen & Puusniekka 2006, 59.) Havainnoinnissa seurataan ilmiötä, esimerkiksi prosessia. Tuntemattomaan prosessiin on hyvä tutustua havainnoimalla, vaikka se olisikin dokumentoitu. (Kananen 2015, 135) Havainnoinnista on käytössä monia eri muotoja. Muodon valinta riippuu ilmiöstä ja siitä, kuinka havainnointi vaikuttaa ilmiöön. (Kananen 2015, 137; Saaranen-Kauppinen & Puusniekka 2006, 59-60.) Tässä työssä käytettiin havainnoinnin muotoina suoraa havainnointia ja osallistuvaa havainnointia. Ensimmäiseksi suunnittelijaa seurattiin toistotyön työvaiheiden aikana ja sen jälkeen osallistuttiin toistotyön työvaiheisiin. Havainnoinnin aikana kirjoitettiin muistiinpanoja.

Haastattelu sopii menetelmäksi, kun tutkitaan mielipiteitä. Sillä voidaan täydentää ja varmistaa aineistoa, joka on kerätty havainnoimalla. (Kananen 2015, 132, 143) Haastattelutyyppejä on useita, mutta ne luokitellaan karkeasti strukturoituihin haastatteluihin ja strukturoimattomiin haastatteluihin. Haastateltavana on hyvä olla henkilöitä, joita aihe koskettaa. (Kananen 2015, 144-145; Saaranen-Kauppinen & Puusniekka 2006, 53.)

Strukturoidut haastattelut ovat lomaketyyppisiä haastatteluja, joissa on valmiit kysymykset ja vastausvaihtoehdot. Haastattelussa kysymykset esitetään samassa järjestyksessä. Strukturoidussa haastattelussa yritetään minimoida haastattelijan vaikutusta aineistoon. (Kananen 2015, 145; Saaranen-Kauppinen & Puusniekka 2006, 56-57.)

Strukturoimaton eli avoin haastattelu on haastattelu, joka muistuttaa tavallista keskustelua. Haastattelun eteneminen on vapaata, mutta tarkoitus on kuitenkin pysyä tietyssä aihepiirissä. Haastattelija voi kuitenkin hienovaraisesti esittää lisäkysymyksiä ja ohjata keskustelua tarvittaessa. (Saaranen-Kauppinen & Puusniekka 2006, 54.) Työn aikana strukturoimattomia haastatteluja pidettiin kokeneemmille suunnittelijoille. Suunnittelijoita oli kaksi ja heidän yhteenlaskettu työkokemus alalta on toistakymmentä vuotta. Ensimmäisessä haastattelussa käytiin läpi alkutilannetta ja kartoitettiin tavoitteita. Jälkimmäisessä haastattelussa käytiin läpi kokemuksia ja mielipiteitä generaattorin käytöstä. Haastatteluja toteutettiin myös reaaliaikaisen keskustelupalvelun välityksellä.

Dokumentit olivat esimerkiksi tekstejä, kuvia ja videoita, jotka liittyivät tutkittavaan ilmiöön. Ne ovat jo tuotettuja tuotoksia. (Kananen 2015, 157-158.) Työssä hyödynnettiin ohjelmointityökalun ja ohjelmointirajapinnan kehittäjän sekä toimeksiantajan vanhojen projektien dokumentteja.

2.2 Aineiston analyysi

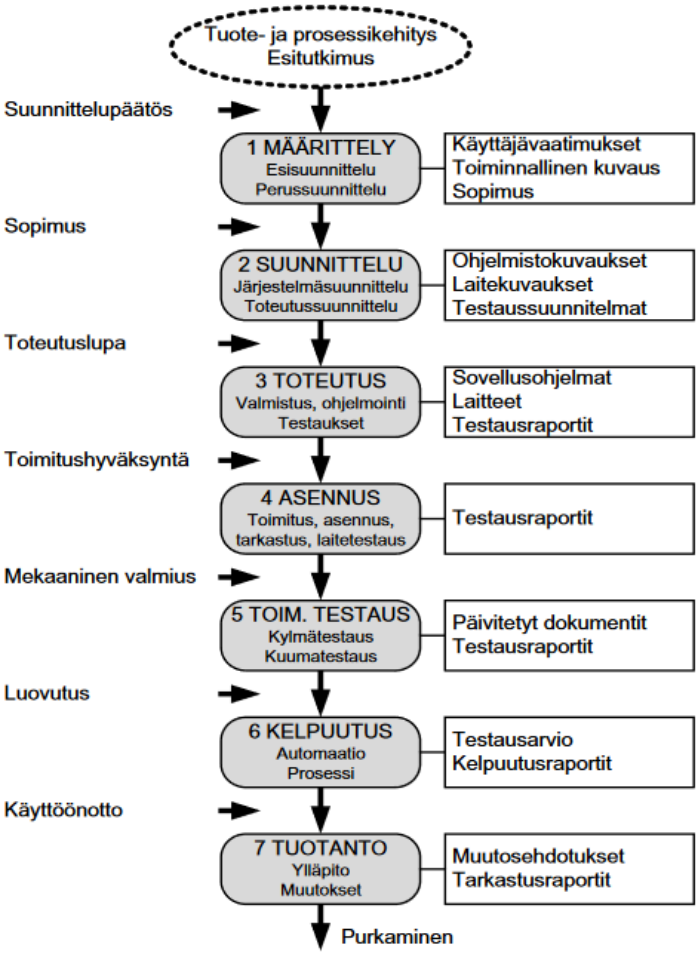
Kerättyyn aineistoon perehdyttiin lukemalla. Kerätylle aineistolle tehtiin sisällönanalyysi. Siinä etsitään aineistosta yhtäläisyyksiä ja eroja, tiivistäen kuvauksen ilmiöstä. (Saaranen-Kauppinen & Puusniekka 2006, 97.) Aineiston määrä ei ollut suuri, joten muita analyysin menetelmiä ei katsottu tarpeelliseksi käyttää.

3 Automaatiosuunnittelu

3.1 Yleistä

Automaatioprojektin tarkoituksena on toteuttaa kohteen automaatiojärjestelmä ja käyttöönotto ja kerätä tarvittavat tiedot järjestelmän ylläpitoon. Automaatiosuunnittelu tuottaa tarvittavan tiedon kohteen automaatiojärjestelmän toteuttamiseksi.

Suunnittelutoimintaa on kuvattu elinkaarimallilla (ks. kuvio 1). Automaatiosuunnittelu koostuu useista eri toimijoista ja tarkkuustasoista. Eri tasoja ovat suunnitteluliiketoiminnan taso, projektihallinnan taso ja suunnittelutehtävien taso. (Hirvonen, Hukki, Strömman, & Tommila 2010, 8.)



Kuvio 1. Automaatiosuunnittelun elinkaari (Hirvonen ym. 2010, 16)

Suunnittelu tuottaa joukon dokumentteja, jotka kuvaavat järjestelmää siten, että järjestelmän toteuttaminen, käyttöönotto ja ylläpito ovat mahdollista. Aluksi on selvitettävä asiakkaan ja käyttäjän tarpeet toiminnoista. Suunnittelu jaetaan erilaisiin suunnittelutehtäviin ja niitä suorittavat organisaatiot ja yksittäiset suunnittelijat. Suunnittelutehtävät sijoittuvat pääsääntöisesti eri elinkaarivaiheisiin, mutta osittain myös niiden välille. Ajoitus on projektikohtaista. Tehtävien tulos syntyy lähtötietojen,

suunnittelijan osaamisen ja projektin tavoitteiden avulla. Tulos arvioidaan itse tai kollegan kanssa, mutta vaativampien ja laajempien kokonaisuuksien tuloksien arviointiin voidaan järjestää virallisempi katselmus tai testaus. Aikaisempien tehtävien tuottamia tuloksia ja materiaaleja voidaan käyttää myöhempien tehtävien suorittamiseen. Suunnittelutehtäviä on erilaisia. Niitä ovat esimerkiksi laitemäärittelyt, ohjelmiston suunnittelu, ohjelmointi. (Hirvonen ym. 2010, 13-21.)

3.2 Määrittelyvaihe

Määrittelyvaiheen ensimmäinen askel on esisuunnittelu. Esisuunnittelussa asiakas määrittelee järjestelmän vaatimuksia ja tarpeita. Esisuunnittelun tuloksia ovat dokumentit, joissa asiakas kuvaa, mitä järjestelmän halutaan tekevän. Dokumentit sisältävät järjestelmän tarpeet, yleiskuvauksen, toiminnalliset ja ei-toiminnalliset vaatimukset, suunnittelun rajoitukset ja kehitysprosessin vaatimukset. (Ajo, Hakonen, Harju, Järvi, Kaskes, Lenardic, Niukkanen, Nurminen, Ritala, Tolppanen & Tommila 2012, 32-40.)

Esisuunnittelun jälkeen on tehtävä perussuunnittelu. Perussuunnittelun tarkoituksena on tuottaa tarkemmat toiminnalliset määrittelyt ja järjestelmän toteutusperiaatteet esisuunnittelun pohjalta. (Hirvonen ym. 2010, 40-47)

Määrittelyvaiheessa tuotetaan prosessikuvaus, joka sisältää PI-kaaviot, ajotapakuvaukset, eri konfiguraatiot ja käynnistyssekvenssit, lukitukset, eri tuotantotilanteet ja niiden vaihtoon liittyvät toimenpiteet sekä normaalit ja hätäpysäytykset. Lisäksi prosessikuvauksen perusteella syntyy automaatiojärjestelmän lähtötiedot eli I/O-lukumäärä, säätöpiirien määrä, toimilaitteiden mitoitustiedot, erilaisten näyttöjen määrä ja tiedot henkilöstöstä, laitoksesta ja muista asennuksista, jotka tulee ottaa huomioon. (Ajo ym. 2012, 32-47; Hirvonen ym. 2010, 20-21.)

3.3 Suunnitteluvaihe

Suunnitteluvaihe, yleensä toteutussuunnittelu, on yksi automaatiosuunnittelun elinkaaren vaiheista. Siinä tarkennetaan määrittelyvaiheen eli esi- ja perussuunnittelun tuloksia, jotta automaatiojärjestelmän laitteiston ja ohjelmiston toteutus voidaan aloittaa automaatiosuunnittelun elinkaaren toteutusvaiheessa. Suunnitteluvaiheessa toteutetaan järjestelmäsuunnittelu, mekaniikkasuunnittelu, laitteistosuunnittelu ja ohjelmistosuunnittelu. (Ajo ym. 2012, 48-55; Kippo & Tikka 2008, 99-101.)

Laitteistosuunnittelussa määritellään järjestelmän rakenne ja tarvittavat laitteet. Laitteilla tarkoitetaan prosessiohjaimia, niiden moduuleja, PC:itä, instrumentointia ja kaapelointia. Lisäksi suunnitellaan verkko, sen tarvitsemat laitteet, käyttöliittymä ja valvomon rakenne. (Ajo ym. 2012, 48-55.)

Ohjelmistosuunnittelu etenee laitteistosuunnittelun kanssa rinnakkain. Ohjelmistosuunnittelussa hyödynnetään määrittelyvaiheen ja laitteistosuunnittelun dokumentteja. Suunnittelun aikana on tiedettävä, mitä kenttälaitteita on tarkoitus ohjata ja miten niihin saa yhteyden. Ohjelmistosuunnittelussa suunnitellaan ohjelmiston rakenne, tietokannat ja käytettävät tyyppilohkot. Ohjelmistossa olisi hyvä käyttää tyyppilohkoja eli valmiita, testattuja ja vakioituja ratkaisuja. (Ajo ym. 2012, 48-55.)

4 TIA Portal -ohjelmointityökalu

4.1 Yleistä

TIA Portal eli Totally Integrated Automation Portal tarkoittaa ”täysin integroitua automaatioportaalia”. TIA Portal on Siemensin kehittämä ohjelmointityökalu teollisuusautomaation suunnitteluun. Ohjelmointityökaluun on yhdistetty logiikkojen ohjelmointiin käytettävä SIMATIC STEP 7, käyttöliittymien suunnitteluun käytettävä SIMATIC WinCC, turvaratkaisujen ohjelmoitiin käytettävä SIMATIC STEP 7 Safety Advanced ja taajuusmuuttajien ohjelmointiin käytettävä SINAMIC Startdrive. (TIA

Portal 2017; Totally Integrated Automation Portal 2013; Your gateway to automation… 2018.)

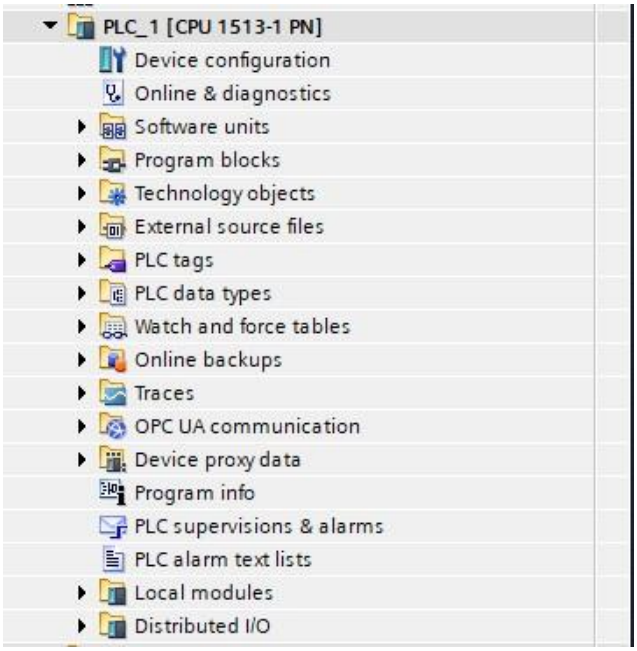
Vuonna 2011 julkaistiin versio 11, joka oli TIA Portalin ensimmäinen versio. Siinä yhdistyivät vain logiikkojen ohjelmointityökalu ja käyttöliittymien suunnittelutyökalu. Version 14 huoltopäivitys 1 julkaistiin vuonna 2017. Versio 15 julkaistiin myös vuonna 2017 ja versio 15.1 vuonna 2018. (Delivery release of TIA Portal V14 SP1 2017; Delivery release SIMATIC STEP 7 Professional V11 2011; Delivery release TIA Portal V15 2017; Delivery release TIA Portal V15.1 2018.)

4.2 Projektin hierarkia

Projektin hierarkian (ks. kuvio 2) ylimpänä on projektikansio. Kansio sisältää kaikki projektiin liittyvät laitteet, asetukset ja toiminnot. Toimintoja ovat projektin laitteiden ja niiden välisen yhteyden määrittäminen ja laitteiden yhteisten viestiluokkien ja tekstilistojen määrittäminen. Asetuksia ovat projektin dokumentoinnin asetukset ja projektin käyttäjän ja järjestelmän kieli- ja tekstiasetukset. (TIA Portal Tutorial #02 2012; Totally Integrated Automation Portal 2013, 4.)

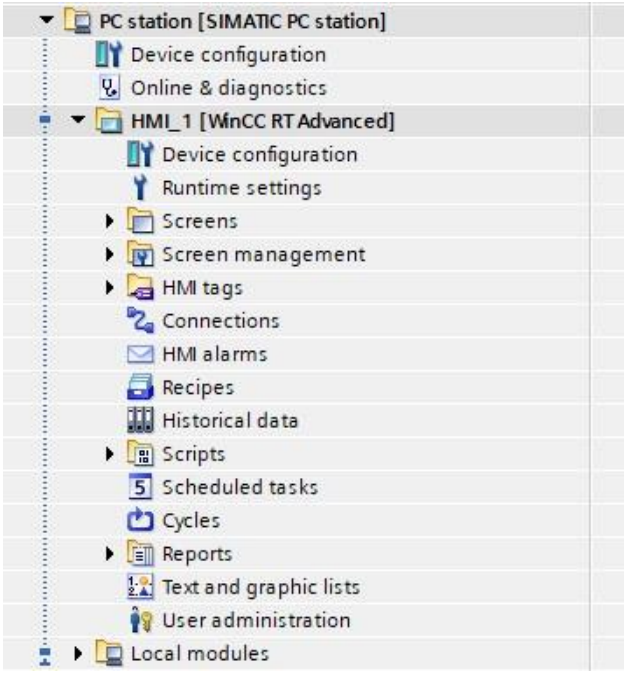
Kuvio 2. TIA Portal -ohjelmointityökalun projektin hierarkia

Jokaisella laitteella on hierarkiassa oma kansionsa ja laitekohtainen kansiorakenne. Ohjelmoitavan logiikan kansiorakenne sisältää ohjelmayksiköt, ohjelmalohkot, turvaparametrit, teknologiaobjektit, ulkoiset lähdetiedostot, tagit, datatyypit, taulukot tagien monitorointiin ja trendit (ks. kuvio 3). Ohjelmalohkojen kansiossa käyttäjä voi luoda uusia kansioita ja organisoida niiden avulla ohjelmansa paremmin. (TIA Portal Tutorial #02 2012; Totally Integrated Automation Portal 2013, 4.)



Kuvio 3. Ohjelmoitavan logiikkaohjaimen hierarkia TIA Portal -ohjelmointityökalussa

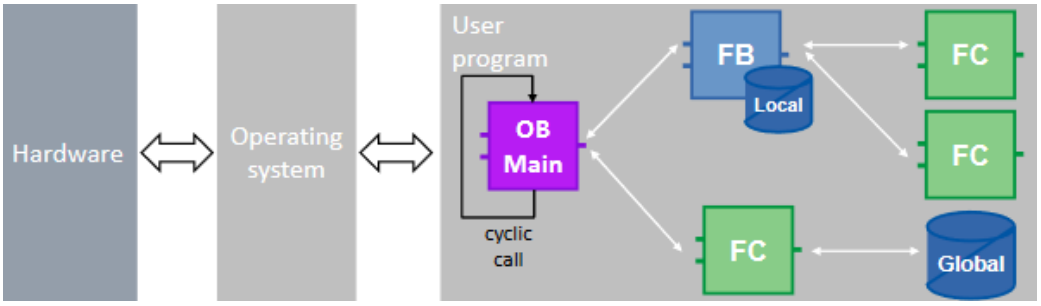
Paneelien ja PC-pohjaisten käyttöliittymien kansiorakenne sisältää laitekonfiguroinnin, valvomonäytöt, valvomokuvien hallinnan, tagilistat, yhteyden luonnin, hälytyksien luonnin, reseptien hallinnan, historiallisen datan käsittelyn, skriptit, ajastetut toiminnot, valvomokuvien päivityssyklit, raportit, teksti- ja grafiikkalistat ja käyttäjähallinnon (ks. kuvio 4). (TIA Portal Tutorial #02 2012) Valvomokuvien kansiossa käyttäjä voi luoda uusia kansioita ja organisoida valvomokuvia samalla tavalla kuin ohjelmoitavan logiikan kansiossa ohjelmalohkoja.



Kuvio 4. HMI:n hierarkia TIA Portal -ohjelmointityökalussa

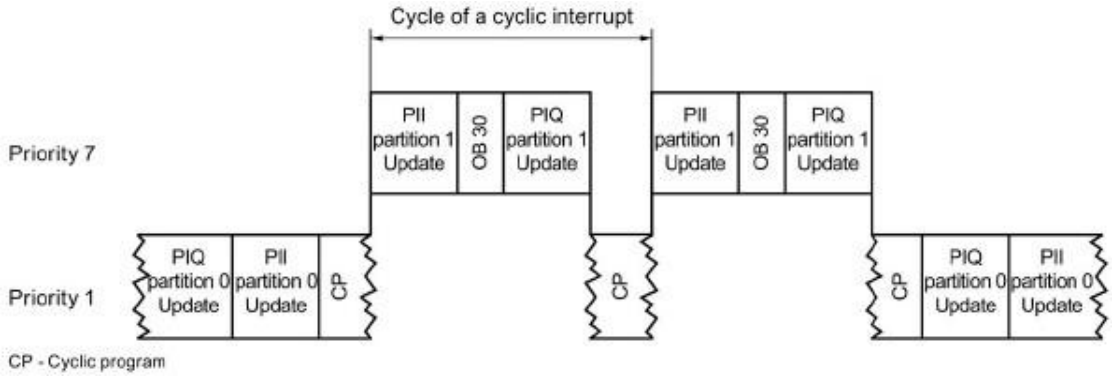
4.3 Ohjelman rakenne

Käyttäjän ohjelma muodostuu ohjelmalohkoista, jotka ladataan logiikkaan ja suoritetaan siellä aina syklisesti (ks. kuvio 5). Lohkoja on neljää eri tyyppiä: organisointilohkoja, toimilohkoja, funktioita ja datalohkoja. Lohkot voivat sisällänsä myös kutsua muita lohkoja. (Programming Guideline for S7-1200/S7-1500 2014, 36-37.)



Kuvio 5. Käyttöjärjestelmä ja käyttäjän ohjelma (Programming Guideline for S7-1200/S7-1500 2014, 36)

Organisointilohkot toimivat käyttöliittymänä käyttöjärjestelmän ja käyttäjän ohjelman välillä. Organisointilohkoja voidaan käyttää määrittelemään logiikkaohjaimen käynnistämisen toiminnot, syklisen ohjelman prosessoinnin, keskeytysohjatun ohjelman prosessoinnin ja virheiden käsittelyn. Syklisessä ohjelman prosessoinnissa organisointilohkot suoritetaan vuoron perään OB-numeron mukaan. Syklin käsittelyajan pituus riippuu ohjelman suuruudesta. Ajoittain on tarve suorittaa tietty rutiini, joka on riippumaton syklisen ohjelman käsittelyajasta. Silloin käytetään korkeamman prioriteetin syklistä keskeytystä. Syklinen keskeytys on keskeytys, joka käynnistää jaksoittaisilla aikaväleillä syklisen keskeytyksen organisointilohkon suorittamisen (ks. kuvio 6). Aikaväli on määriteltävissä ja se voi olla kymmenestä millisekunnista viiteen sekuntiin. Kun syklisen keskeytyksen organisointilohko on suoritettu, jatketaan syklisen ohjelman prosessointia keskeytyspisteestä. (Berger 2014, 205-206; Programming Guideline for S7-1200/S7-1500 2014, 37-39.)

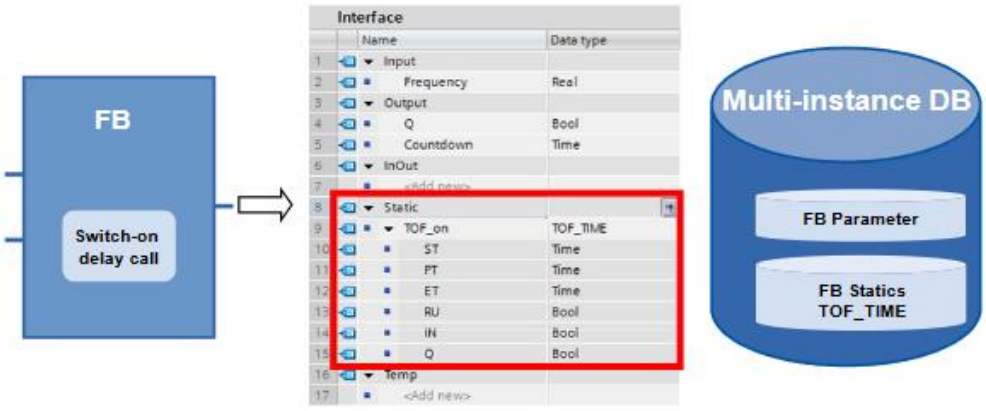


Kuvio 6. Syklisen keskeytyksen sykli (Cycle and response times 2018)

Funktiot eli FC:t ovat lohkoja ilman syklistä muistia. Lohkon arvot eivät tallennu sykliltä toiselle. Funktioita käytetään esimerkiksi silloin, kun samaa laskutoimitusta tarvitaan useita kertoja ohjelman eri osissa. Funktioilla voidaan esimerkiksi toteuttaa matemaattisia laskutoimituksia. (Programming Guideline for S7-1200/S7-1500 2014, 40-41.)

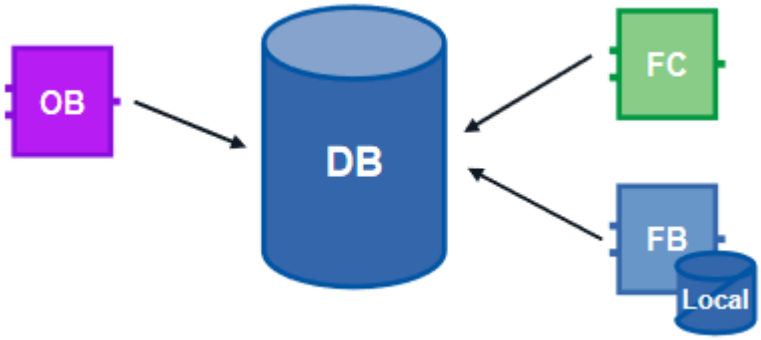
Toimilohkot eli FB:t ovat lohkoja syklisellä muistilla, joihin arvot tallentuvat pysyvästi. Toimilohkoja käytetään aliohjelmiin ja käyttäjäohjelman rakenteeseen. Toimilohkon kutsua sanotaan instanssiksi. Instanssin data tallennetaan instanssidatalohkoon eli

DB:hen. Jokaisella toimilohkolla on oma instanssidatalohko, joka toimii syklisenä muistina. Instanssidatalohkon rakenne määritellään toimilohkossa. Jos toimilohkossa kutsutaan toista toimilohkoa, voidaan data tallentaa kutsutun toimilohkon datalohkoon. Tätä kutsutaan multi-instanssiksi. (Programming Guideline for S7-1200/S7-1500 2014, 42-45.)



Kuvio 7. Multi-instanssi (Programming Guideline for S7-1200/S7-1500 2014, 43)

Globaalidatalohko on datalohko, jota voi hyödyntää kaikki muut lohkotyypit (ks. kuvio 8). Globaalidatalohkoon voidaan tallentaa muuttujien arvoja koko käyttäjäohjelmasta. Sen rakenne voi koostua satunnaisista datatyypeistä. (Programming Guideline for S7-1200/S7-1500 2014, 45-46.)



Kuvio 8. Globaalidatalohko (Programming Guideline for S7-1200/S7-1500 2014, 46)

Ohjelmoidessa kohdejärjestelmän toiminta on suositeltavaa jakaa pienemmiksi kokonaisuuksiksi toimilohkojen avulla siten, että samoja toimilohkoja voidaan käyttää useasti. Näistä toimilohkoista voidaan tehdä vakioratkaisuja luomalla tyyppilohkokirjasto ja tallentamalla ne sinne. Tyyppilohkokirjaston toimilohkoja voidaan käyttää useasti projektin aikana ja jatkossa tulevissa projekteissa. Esimerkkejä tyyppilohkokirjaston toimilohkoista ovat moottorinohjaustoimilohko (ks. kuvio 9) ja mittauksen monitorointitoimilohko. (Programming Guideline for S7-1200/S7-1500 2014, 36-37, 50-51.)

Toimeksiantajan moottorinohjaustoimilohkolla asetetaan moottorin ohjaustapa käsi- tai automaattiohjaukselle. Toimilohkoon viedään käynti-, pysäytys-, lukitus- ja hälytysehdot sekä takaisinkytkentä moottorilta. Toimilohkosta puolestaan tuodaan käyntikäsky, pysäytyskäsky tai hälytystieto. Toimilohko ohjaa tuoduilla tiedoilla toimilohkon lähtöjä ja siten moottoria lukittumaan, käynnistymään tai pysähtymään. Ohjaustila ja käyntitiedot viedään valvomoon.

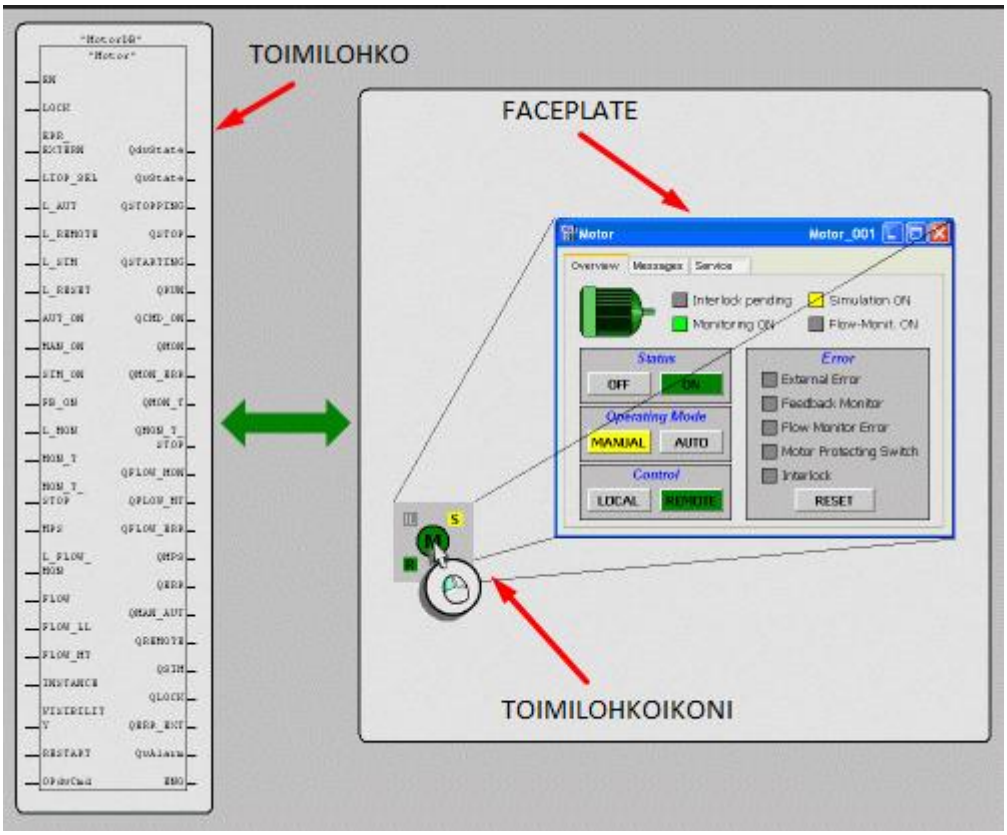
Mittauksen monitorointitoimilohkolla tarkastellaan, tarvitseeko mittauksesta tehdä hälytys vai onko mittausarvo hälytysrajojen sisäpuolella. Toimilohkolle tuodaan mittaustieto, hälytysrajat ja hälytysviiveet. Toimilohkosta tuodaan ulos tarvittaessa hälytystieto. Lopuksi mittausarvo viedään valvomoon, ja tarvittaessa myös hälytystieto. Kun toimilohko on tallennettu kirjastoon, voidaan jokaisessa analogiamittauksessa käyttää kyseistä toimilohkoa.

4.3.1 Toimilohkojen yhdistäminen valvomoon

Tyyppilohkokirjaston toimilohkojen tiedot tuodaan valvomonäytölle faceplatejen avulla. Esimerkiksi moottorinohjaustoimilohkon ohjauspaneelifaceplatessa on moottorin käyntitiedot, lukitustiedot ja painikkeet käsiohjauksille. Faceplate on ikään kuin vakioitu ratkaisu valvomossa, kuten tyyppilohkokirjaston toimilohkot ovat ohjelmassa. Ilman faceplatea, jokaiseen käyttöobjektiin eli painikkeeseen ja tekstikenttään pitäisi viedä tieto erikseen. Faceplate on konfiguroitu näyttö- ja käyttöobjektien ryhmä. Sen käytöstä hyötyy monin tavoin sovellussuunnittelun aikana. Esimerkiksi faceplatelle voidaan viedä vain yksi tagi, joka on käyttäjän määrittelemä datatyyppi. Se

sisältää kaikki käyttöobjekteilta toimilohkoon tuotavan ja toimilohkosta käyttöobjektiin vietävän tiedon. Tagimäärän vähentämisellä myös virheiden määrät vähenevät. Lisäksi, jos esimerkiksi moottorin toimilohkon ohjauspaneeliin (ks. kuvio 9) tarvitsee tehdä muutos, riittää kun muutoksen tekee faceplateen. Silloin muutos päivittyy kaikkiin moottorin ohjauspaneeleihin. Projektissa moottoreita on kymmeniä, joten muutos olisi muutoin tehtävä erikseen kaikkiin ohjauspaneeleihin. (Basics on HMI Faceplates 2017, 6-10.) Toimeksiantajan faceplatet tehdään valvomoon pop-up-ikkunoiksi. Pop-up-ikkunat aukaistaan prosessin valvomokuvan toimilohkoikoneista.

Toimilohkoikoni on objekti valvomokuvassa. Moottorille, venttiilille ja mittaukselle on omat toimilohkoikoninsa. Esimerkiksi moottorin toimilohkoikoni on ympyrä, jossa on M-kirjain keskellä. Ympyrän väri muuttuu riippuen moottorin tilasta: käy, pysähtynyt tai lukittunut. Ympyrän vieressä on M- tai A-kirjain, joka viittaa moottorin ohjaustapaan manual tai auto (käsi- tai automaattiohjaus). Toimilohkoikonia klikkaamalla saadaan aukaistua faceplate pop-up-ikkuna (ks. kuvio 9).

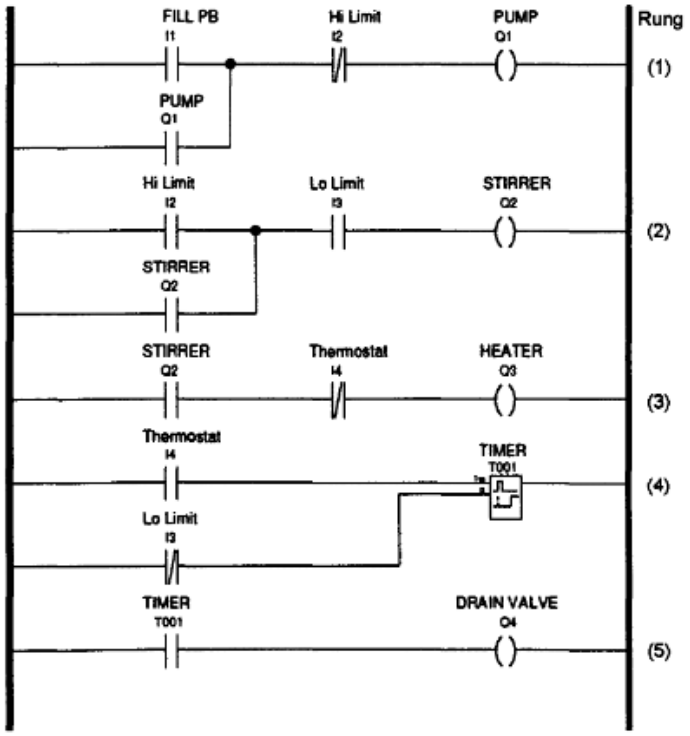


Kuvio 9. Siemensin mallitoteutus moottorinohjaustoimilohkosta, toimilohkoikonista ja faceplatesta

4.4 Ohjelmointikielet

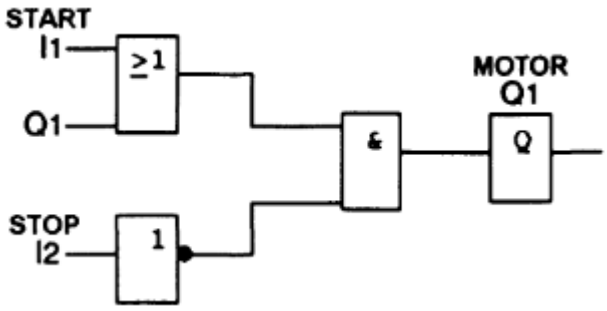
Jokainen lohko voidaan ohjelmoida sovellussuunnittelijan valitsemalla ohjelmointikielellä. Ohjelmointikieliä on viisi erilaista: ladder diagram (LAD) eli tikapuukaavio, function block diagram (FBD) eli toimilohkokaavio, statement list (STL) eli käskylista, structured control language (SCL) eli rakenteellinen teksti ja graph (S7-GRAPH) eli vuokaavio. Ohjelmointikielet täyttävät kansainvälisen sähköalan standardointiorganisaatio IEC:n standardin IEC 61131-3 vaatimukset. (Automaatio-ohjelmistot 2016; Programming Guideline for S7-1200/S7-1500 2014, 9; Standards Compliance according to IEC 61131-3 2013, 1.)

TIA Portal -ohjelmointiympäristön ladder diagram (LAD) vastaa standardin IEC 61131-3 ladder diagram (LD) -ohjelmointikieltä. Tikapuukaavio on ohjelmoitavan logiikan graafinen ohjelmointikieli. Nimensä se on saanut siitä, että se muistuttaa ulkonäöltään tikapuita (ks. kuvio 10). LD-ohjelmointikieli on johdettu relekaavioista, joilla esitetään sähkömekaanisia relejärjestelmiä. LD-ohjelmointikielessä vasemmassa reunassa oleva pystyviiva kuvaa positiivista virtakiskoa. Pystyviivaan on yhdistetty vaakaviivoja, joissa on kytkimiä ja suoritettava toiminto. Viivoilla kuvataan tehon virtausta vasemmalta oikealle. Kytkimiä ja viivoja voi asettaa myös rinnakkain. LD-ohjelmointikieltä sovelletaan pääasiassa totuusarvo-tyyppisille muuttujille ja loogisille operaatioille, mutta sillä on kuitenkin mahdollista tehdä myös esimerkiksi laskutoimituksia. (John & Tiegelkamp 2010, 147-154; Standards Compliance according to IEC 61131-3 2013, 1; Walker 2012, 53-54, 174-176.)



Kuvio 10. Tikapuukaavio (LD) (Walker 2012, 57)

Function block diagram (FBD) eli toimilohkokaavio on toinen ohjelmoitavan logiikan graafinen ohjelmointikieli. Se on alunperin lähtöisin signaalinkäsittelytekniikasta, jossa kokonais- ja desimaaliluvut ovat tärkeitä. Toimilohkot ovat koodiyksiköitä, jotka ovat kirjoitettu jollain toisella IEC 61131-3 standardin mukaisella ohjelmointikielellä. Toiminnot voivat olla yksinkertaisia bittioperaattoreita tai monimutkaisempia laskutoimituksia. Graafisesti ne esitetään nelikulmaisina laatikoina, joiden suoristusjärjestys muodostetaan yhdistämällä laatikoita toisiinsa pysty- ja vaakaviivoilla (ks. kuvio 11). Laatikoissa eli lohkoissa viivat yhdistyvät tulo- ja lähtöpinneihin. Pinneihin voi kytkeä myös muuttujia ja muuttumattomia arvoja. Tulo- ja lähtöpinneihin tulevat arvot käytetään toimilohkon toiminnossa. (John & Tiegelkamp 2010, 134-137; Standards Compliance according to IEC 61131-3 2013, 1; Walker 2012, 60.)



Kuvio 11. Toimilohkokaavio (FBD) (Walker 2012, 60)

TIA Portal -ohjelmointiympäristön statement list (STL) vastaa standardin IEC 61131-3 instructions list (IL) -ohjelmointikieltä. Käskylista on alemman tason tekstimuotoinen ohjelmointikieli, joka muistuttaa Assembly-ohjelmointikieltä (ks. kuvio 12). IL-ohjelmointikieli muodostuu sarjasta ohjeita. Ohje on yhden rivin mittainen käsky ohjelmoitavalle logiikalle. Se muodostuu yleensä operaattorista ja operaation tyypistä riippuen yhdestä tai useammasta kohdemuuttujasta. Luettavuuden ja jäsentämisen parantamiseksi on IL-ohjelmointikielessä käytettävissä tyhjät rivit ja käskyjen kommentointi. Otsikointia puolestaan käytetään ohjelman suorittamisen aikana hyppyihin, joilla vaikutetaan suoritusjärjestykseen. (John & Tiegelkamp 2010, 100-101; Standards Compliance according to IEC 61131-3 2013, 1; Walker 2012, 61.)

```
LD    I1   <Load register with the value of input 1 (START)>
OR    Q1   <Logical OR with output 1 (MOTOR latch contact)>
ANDN  I2   <Logical AND with the inverse of input 2 (STOP)>
OUT   Q1   <Set/Reset output 1 (MOTOR)>
END
```

Kuvio 12. Käskylista (IL) (Walker 2012, 61)

TIA Portal -ohjelmointiympäristön structured control language (SCL) vastaa standardin IEC 61131-3 structured text (ST) -ohjelmointikieltä. Rakenteellinen teksti on korkeamman tason tekstimuotoinen ohjelmointikieli, joka muistuttaa paljon PASCAL-

ohjelmointikieltä. ST-ohjelmointikielessä voidaan käyttää osoituksia ja loogisia operaattoreita kuten muissa ohjelmointikielissä, mutta myös monimutkaisempia silmukkarakenteita kuten FOR- ja WHILE-silmukoita, valintarakenteita kuten IF- ja CASE-valintoja ja aliohjelmia. Kieli muodostuu lauseista, joilla voidaan suorittaa monimutkaisiakin toimintoja hyvin tiivistetysti (ks. kuvio 13). Lauseet erotellaan toisistaan puolipilkulla. ST-ohjelmointikielessä voidaan käyttää kommentteja ja sisennyksiä luettavuuden ja jäsentämisen parantamiseksi. (John & Tiegelkamp 2010, 116-119; Standards Compliance according to IEC 61131-3 2013, 1; Walker 2012, 62.)

```
1 IF (#I1 OR #Q1) AND NOT #I2 THEN
2     #Q1 := TRUE;
3 END_IF;
```

Kuvio 13. Rakenteellinen teksti (SCL)

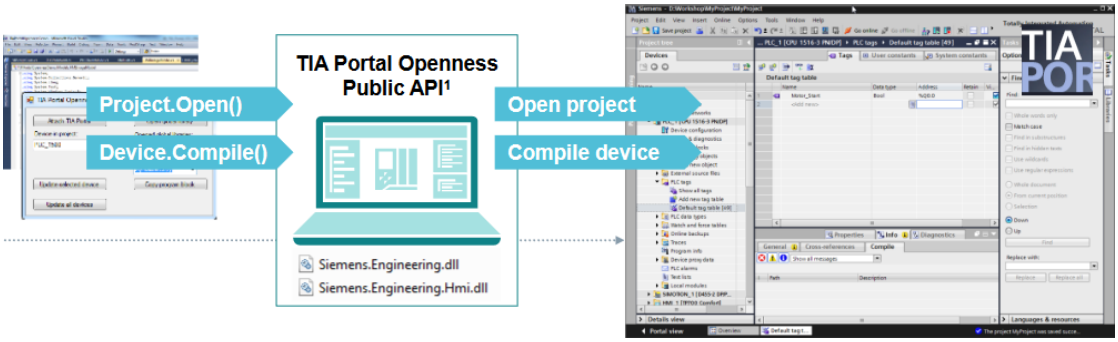
TIA Portal -ohjelmointityökalun GRAPH-ohjelmointikieli vastaa standardin IEC 61131-3 sequential function chart (SFC) -ohjelmointikieltä. Vuokaavio on graafinen ohjelmointikieli, joka on johdettu tunnetuista tekniikoista, kuten Petri-net:stä tai sekvenssimenetelmästä. SFC-ohjelmointikieli suunniteltiin jakamaan monimutkainen ohjelma pienempiin ja hallittavampiin osiin ja kuvaamaan näiden osien välistä suoritusjärjestystä. Etenkin askelmaiset prosessit soveltuvat erittäin hyvin SFC-ohjelmointikielelle. Kieli koostuu askeleista ja siirroista. Askeleeseen siirryttäessä askel muuttuu aktiiviseksi. Askeleen ollessa aktiivinen, suoritetaan sarja ohjeita eli käskyjä. Käskyjä toistetaan, kunnes askeleesta siirrytään pois ja askel menee passiiviseksi. Askeleiden välillä käytetään siirtoa. Siirron suorittamiseksi täytyy siirtoehdon totuusarvo muuttua ”tosi” tilaan. Yksinkertaista yhdistelmää askeleista ja siirroista kutsutaan sekvenssiksi. Ohjelmakoodissa sekvenssit voivat olla vaihtoehtoisia tai rinnakkaisia toisilleen. (John & Tiegelkamp 2010, 169-176; Standards Compliance according to IEC 61131-3 2013, 1; Walker 2012, 63.)

5 TIA Openness -ohjelmointirajapinta

5.1 Yleistä

TIA Openness, joskus myös TIA Portal Openness, on ohjelmointirajapinta TIA Portal -ohjelmointityökalun SIMATIC STEP 7:lle ja SIMATIC WinCC:lle. Sitä on kehitetty TIA Portal V12 SP1 -version julkaisusta alkaen eli vuodesta 2015. Se on ollut saatavilla TIA Portal V13 SP1 -versiosta alkaen ja sisältynyt STEP 7- ja WinCC-asennusmediaan. (TIA Portal Openness 2019, 4-5; TIA Portal Openness 2017a.)

TIA Openness käyttää DLL-tiedostoja eli jaettuja kirjastotiedostoja päästäkseen käsiksi TIA Portal -ohjelmointityökalun objekteihin ja toimintoihin (ks. kuvio 14). DLL-tiedostoja voidaan käyttää ulkoisessa kehitysympäristössä, jossa voidaan luoda omia sovelluksia automatisoimaan suunnittelutehtäviä. (TIA Portal Openness 2019; TIA Portal Openness 2017a.)

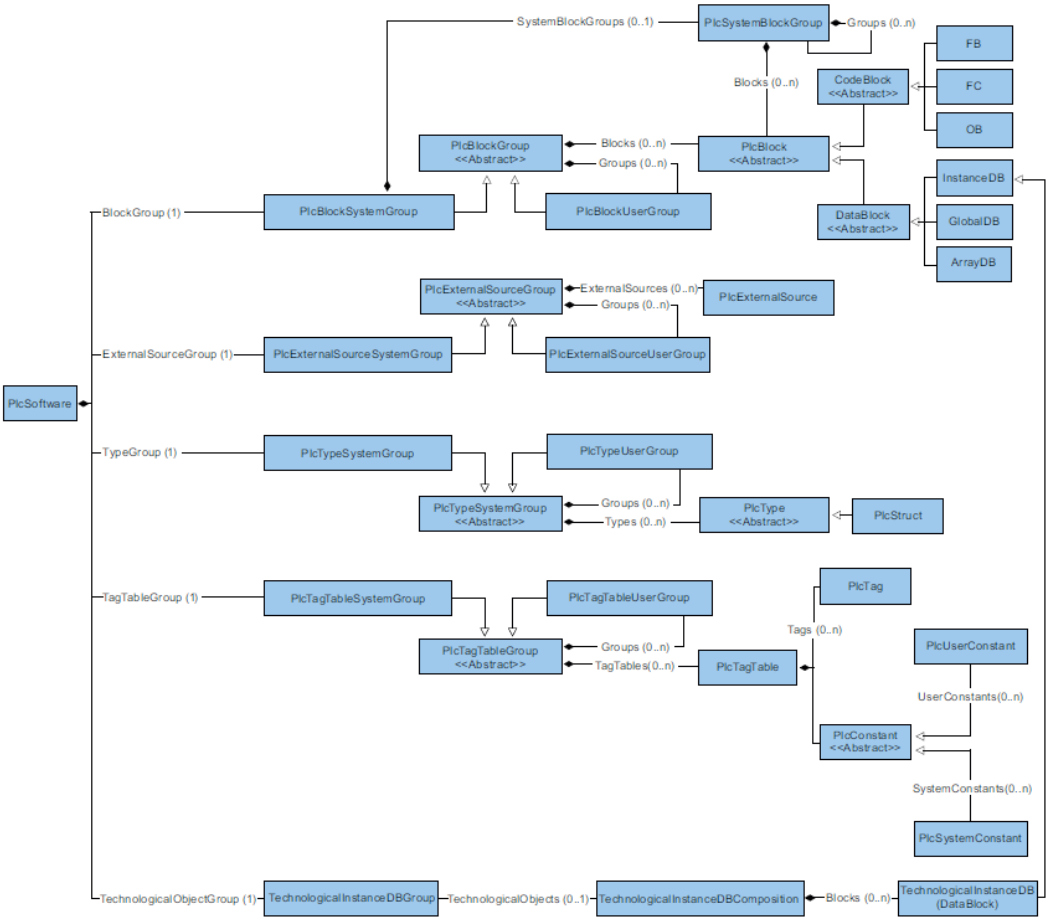


Kuvio 14. TIA Opennesin toimintaperiaate (TIA Portal Openness 2017a)

Ohjelmointirajapinta ja mahdollisuus omien sovelluksien käytölle on tehty vähentämään suunnittelijan toistuvia työtehtäviä. Automatisointi lisää tehokkuutta suunnittelutyöhön ja vähentää virheitä. Esimerkkisovelluksia, joita voidaan kehittää, ovat erilaiset koodigeneraattorit valvomokuville ja ohjelmoitavan logiikan ohjelmalohkoille sekä vertailija, joka vertailee projektia globaaliin kirjastoon ja päivittää projektin automaattisesti. (TIA Portal Openness 2019; TIA Portal Openness 2017a; TIA Portal Openness 2017b.)

5.2 Hierarkia

TIA Openness -ohjelmointirajapintaa käyttämällä päästään käsiksi TIA Portalin objekteihin ja toimintoihin. TIA Openness objektit ja toiminnot ovat hierarkiassa kuten TIA Portal -ohjelmointityökalussakin, mutta ne hieman poikkeavat toisistaan. TIA Openness hierarkiassa ohjelmoitavan logiikan data on PlcSoftware-objektin alla, joka vastaa TIA Portalissa ohjelmoitavan logiikan kansiota. PlcBlockGroup-objektin alla puolestaan on TIA Portalin Program blocks -kansion data. Kuviossa 15 on kaavio, joka esittää TIA Openness -ohjelmointirajapinnan ohjelmoitavan logiikan objektimallin.

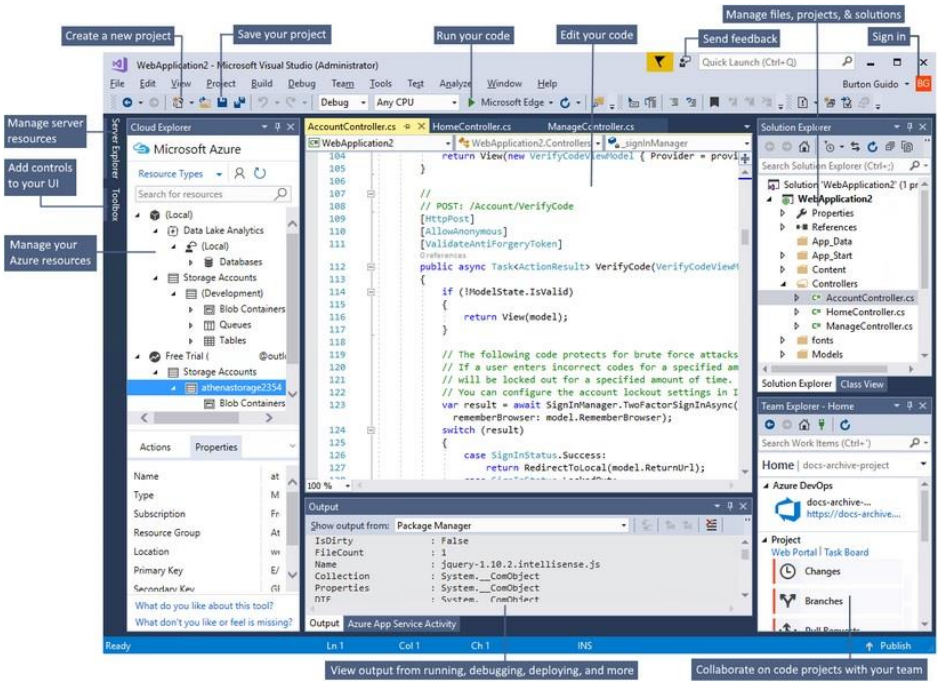


Kuvio 15. TIA Openness objektimalli ohjelmoitavasta logiikasta (Openness 2018, 56)

5.3 Visual Studio

Siemensin TIA Openness sovelluksien ohjelmointi edellyttää, että ne kehitetään Microsoft Visual Studio 2015 Update 1- tai uudemmassa kehitysympäristössä, joka tukee .NET 4.6.2 -versiota. Lisäksi tarvitaan osaamista Visual Studiosta, C#- ja VB.net -ohjelmointikielistä sekä .NET-ohjelmistokomponenttikirjastosta. (Openness 2018, 25)

Visual Studio on Microsoftin kehittämä kehitysympäristö. Se on kattava kokoelma erilaisia sovelluskehitystyökaluja ja -palveluita (ks. kuvio 16). Sillä voidaan kirjoittaa, muokata, koota, rakentaa ja testata koodia. Sitä käytetään Windows-, verkko-, pilvi-, Android- ja iOS-sovelluksien kehittämiseen. Visual Studiolla voidaan kehittää myös pelejä PC-, Android-, ja iOS-alustoille. Yleisen editorin ja virheenkorjausohjelman lisäksi Visual Studion työkaluja ovat muun muassa kääntäjät, koodin viimeistelytyökalut ja graafisen suunnittelun työkalut. (Welcome to the Visual Studio IDE 2019.)



Kuvio 16. Visual Studion käyttöliittymä (Welcome to the Visual Studio IDE 2019)

5.3.1 Ohjelmointikielet

C# on Microsoftin kehittämä korkeamman tason yksinkertainen, olio-orientoitu ja tyyppiturvallinen ohjelmointikieli. Se on ECMA-organisaation standardoima kieli. C#:n juuret ovat C-kielien perheessä ja se on perheen ensimmäinen komponenttisuuntautunut ohjelmointikieli. C# käyttää .NET-ohjelmistokomponenttikirjastoa. C# muistuttaa paljon C++ ja Java-ohjelmointikieliä ja siksi sen kirjoittaminen on luontevaa C, C++, Java ja JavaScript-ohjelmoijille (ks. kuvio 17). (A Tour of the C# Language 2016; Lahtonen 2016; Sivonen 2004.)

```
1   using System;
2
3   namespace HelloWorld
4   {
        0 references
5       class Program
6       {
            0 references
7           static void Main(string[] args)
8           {
9               Console.WriteLine("\nWhat is your name? ");
10              var name = Console.ReadLine();
11              var date = DateTime.Now;
12              Console.WriteLine($"\nHello, {name}, on {date:d} at {date:t}!");
13              Console.Write("\nPress any key to exit...");
14              Console.ReadKey(true);
15          }
16      }
17  }
18
```

Kuvio 17. Hello world -ohjelma C#-ohjelmointikielellä (Build a C# Hello World… 2017)

Visual Basic on Microsoftin kehittämä korkeamman tason olio-orientoitu ja tyyppiturvallinen ohjelmointikieli. Se pohjautuu Microsoftin kehittämään BASIC-ohjelmointikieleen. Visual Basic julkaistiin erillisenä versioon 6.0 asti, minkä jälkeen se tuli osaksi .NET-ohjelmistokomponenttikirjastoa. Visual Basic on suunniteltu helposti lähestyttäväksi ja opittavaksi kieleksi. Sen syntaksi muistuttaa englantia, mikä edistää koodin selkeyttä ja luettavuutta (ks. kuvio 18). (Vick & Wischik 2017; Visual Basic n.d; Visual Basic Guide 2018.)

```
1   Imports System
2
    0 references
3   Module Program
        0 references
4       Sub Main(args As String())
5           Console.WriteLine(vbCrLf + "What is your name? ")
6           Dim name = Console.ReadLine()
7           Dim currentDate = DateTime.Now
8           Console.WriteLine($"{vbCrLf}Hello, {name}, on {currentDate:d} at {currentDate:t}")
9           Console.Write(vbCrLf + "Press any key to exit...")
10          Console.ReadKey(True)
11      End Sub
12  End Module
13
```

Kuvio 18. Hello world -ohjelma Visual Basic -ohjelmointikielellä (Build a Visual Basic… 2017)

5.4 XML-tiedostot

TIA Openness -ohjelmointirajapinta tukee projektitietojen tuontia TIA Portal -ohjelmointiympäristöstä omaan sovellukseen ja vientiä omasta sovelluksesta TIA Portaliin XML-tiedostojen avulla. (Openness 2018, 35)

XML on World Wide Web Consortiumin (W3C) kehittämä merkintäkieli. Se on joustava tekstimuoto, joka on johdettu ISO 8879 standardin Standard Generalized Markup Language (SGML) -metakielestä. Suunnittelutavoitteina olivat muun muassa helppokäyttöisyys internetissä, yhteensopivuus monenlaisiin sovelluksiin ja asiakirjojen helppolukuisuus ja käsiteltävyys. (Extensible Markup Language (XML) 1.0 2008.)

XML määrittelee asiakirjojen koodaamiselle syntaksin, jota ihmiset ja koneet voivat lukea (ks. kuvio 19). Se käyttää tunnisteita, jotka määrittelevät asiakirjan rakenteen ja sen, miten asiakirja tulisi tallentaa ja kuljettaa. Käyttäjä voi luoda omia merkintäsymboleja sisällön kuvaamiseksi, kun taas esimerkiksi verkkosivujen ohjelmointiin käytettävä Hypertext Markup Language (HTML) käyttää ennalta määriteltyjä merkintäsymboleja kuvaamaan verkkosivujen sisältöä. (Gavin 2018.)

```
      <ProgrammingLanguage>FBD</ProgrammingLanguage>
    </AttributeList>
    <ObjectList>
      <MultilingualText ID="4" CompositionName="Comment">
        <ObjectList>
          <MultilingualTextItem ID="5" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text />
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
      <MultilingualText ID="6" CompositionName="Title">
        <ObjectList>
          <MultilingualTextItem ID="7" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text>Analogia sisääntulo ja skaalaus</Text>
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
    </ObjectList>
  </SW.Blocks.CompileUnit>
  <SW.Blocks.CompileUnit ID="8" CompositionName="CompileUnits">
    <AttributeList>
```

Kuvio 19. TIA Portal -projektista tuodun mittauksen monitorointilohkon koodia

6 Generaattorin toteutusvaiheet

6.1 Perehtyminen

Ensimmäisenä perehdyttiin tutkimusongelmaan eli toimilohkojen, toimilohkoikoneiden ja faceplatejen tekemisen toistuviin työvaiheisiin. Aineistoa kerättiin suorahavainnoinnilla ja osallistuvalla havainnoinnilla. Ensiksi suunnittelijaa seurattiin toimilohkojen, toimilohkoikoneiden ja faceplatejen tekemisen toistuvien työvaiheiden aikana ja seuraavaksi osallistuttiin näihin toistuviin työvaiheisiin. Havainnoinnin aikana kirjoitettiin muistiinpanoja ja käytiin keskustelua ongelmaan liittyen.

Havainnoinnin jälkeen kerättiin sekundääriaineistoa TIA Portal -ohjelmointiympäristön ja TIA Openness -ohjelmointirajapinnan dokumenteista. TIA Openness -dokumenteissa funktiot ovat sekä selitetty englanniksi että esitetty C#-ohjelmointikielellä.

Siemensin tukisivuilta löytyi myös malliprojekti, jolla pystyi lisäämään laitteen TIA Portal -projektiin.

6.2 Esisuunnittelu

Esisuunnittelu aloitettiin heti opinnäytetyön aihe-ehdotuksen hyväksynnän jälkeen. Perehtymisen aikana tehdyn havainnoinnin pohjalta pidettiin strukturoimattomia haastatteluita suunnittelijoille, joita aihe kosketti. Haastateltavat suunnittelijat olivat toimeksiantajan työntekijöitä, joilla on useita vuosia työkokemusta TIA Portal -ohjelmointiympäristössä työskentelystä. Haastatteluissa käytiin läpi alkutilannetta ja kartoitettiin generaattorin ja generoinnin tavoitteita.

Haastatteluiden jälkeen pidettiin lyhyt aloituspalaveri, jossa käytiin haastatteluaineistoa läpi. Haastatteluaineistoon kerättiin yksityiskohtaisempaa tietoa toimilohkojen, toimilohkojen datalohkojen, toimilohkoikonien ja faceplatejen generoitavista kohdista ja siitä, missä muodossa generoitava data olisi. Haastatteluissa suurimmaksi puheenaiheeksi nousi generoinnin soveltuvuus eri projekteihin. Projekteissa on eri tavalla toteutettuja toimilohkoja, joskus myös asiakkaan omia ratkaisuja. Generointi ei saisi olla riippuvainen toimilohkon toteutustyylistä, vaan sen pitäisi olla mahdollista eri suunnittelijoiden ja asiakkaiden ratkaisuilla. Lisäksi eri projekteissa saatetaan käyttää eri TIA Portal -versioita, mikä ei saisi olla este generoinnille.

Haastatteluaineiston läpikäymisen jälkeen asetettiin generaattorille ja generoinnille tavoitteet. TIA Openness -ohjelmointirajapinnalla toteutetun generaattorin tavoitteena oli saada toimilohkot, toimilohkojen datalohkot, toimilohkoikonit ja faceplatet generoitua Excel-tiedostosta TIA Portal -projektiin. Sen oli toimittava niillä kolmella ohjelmointityökalun uusimmalla versiolla, jotka ovat käytössä meneillä olevissa projekteissa. Generoinnin tavoitteena puolestaan oli vähentää suunnittelijan käyttämää työaikaa toistuviin työvaiheisiin ja vähentää virheiden määrää automatisoimalla toimilohkojen, toimilohkojen datalohkojen, toimilohkoikoneiden ja faceplatejen luominen, tagien yhdistäminen ja kommentointi. Generoinnin tulisi onnistua kaikenlaisilla

toimilohkoilla, eikä vain tyyppilohkokirjaston valmiilla toteutuksilla. Toimilohkoikonit ja faceplatet puolestaan generoitaisiin tyyppikirjaston toteutuksilla.

Toimeksiantaja ei asettanut rajoitteita Excel-tiedoston muotoilulle eikä generaattorin käyttöliittymälle. Ohjelmointikieleksi valikoitui Visual Basic opinnäytetyöntekijän oman kokemuksen ja TIA Openness -ohjelmointirajapinnan edellytyksien takia.

6.3 Perussuunnittelu

Perussuunnittelu aloitettiin suunnittelemalla generaattorin toimintaperiaatetta asetettujen tavoitteiden pohjalta. Kun generaattori aukaistaan, ensimmäiseksi on valittava se TIA Portal -ohjelmointityökalun versio, johon generaattori yhdistetään. Generaattorista olisi hyvä olla mahdollista aukaista TIA Portal -ohjelmointityökalu, sillä sen voi TIA Openness -dokumenttien mukaan halutessaan avata ilman käyttöliittymää. Kun TIA Portal on aukaistu, on seuraavaksi aukaistava projekti. Projekti pitäisi olla mahdollista aukaista, tallentaa ja sulkea generaattorilla, koska aikaisemmassa vaiheessa TIA Portal on voitu aukaista ilman käyttöliittymää. On myös mahdollista, että TIA Portal -prosessi ja -projekti ovat aukaistuna jo ennen generaattoria. Generaattori tulisi olla mahdollista yhdistää myös valmiiksi avattuun TIA Portal -prosessiin ja -projektiin.

Kun generaattori on yhdistetty, voidaan suorittaa haluttu toiminto eli laitteen lisäys, toimilohkojen generointi tai toimilohkoikonien ja faceplatejen generointi. Lisättäessä laitetta projektiin täytyy generaattorille syöttää tarvittavat tiedot: laitteen nimi, Siemensin artikkeli- ja versionumero. Annetut tiedot syötetään ohjelmaan ja projektiin luodaan uusi laite.

Jotta toimilohkojen generointi onnistuisi, tarvitaan jokin toimilohko generointipohjaksi. Yksi tavoitteista oli generoida eri suunnittelijoiden ja asiakkaiden toimilohkoja, joten toimilohko täytyy ensiksi suunnitella ja toteuttaa TIA Portalilla. Suunniteltu toimilohko tuodaan TIA Portalista generaattorilla XML-tiedostona käyttäjän valitsemaan kansioon. Kansiossa olevasta toimilohkosta tulisi generoida toimivia toimilohkoja eli

tiedoston sisältöä olisi muokattava ja sitten tallennettava se uudeksi tiedostoksi useita kertoja peräkkäin. Generoinnin jälkeen generoidut XML-tiedostot tulisi viedä takaisin TIA Portal -projektiin.

Tavoitteena oli generoida toimilohkojen ja toimilohkojen datalohkojen lisäksi myös faceplateja ja toimilohkoikoneita valvomon graafiseen käyttöliittymään. Faceplatet ovat TIA Portal -projektissa valvomonäytön kuvia. Jokaiselle moottorinohjaustoimilohkolle, venttiilinohjaustoimilohkolle ja mittaustoimilohkolle on oma faceplate, joka yhdistetään toimilohkoon yhdellä tai useammalla tagilla. Faceplatejen generointi oli mahdollista toteuttaa samalla tavalla kuin toimilohkojen ja toimilohkojen datalohkojen.

Toimilohkoikonit puolestaan ovat objekteja valvomonäytön kuvassa. Niiden generointi ei onnistu samalla tavalla kuin toimilohkojen ja faceplatejen, joten tässä sovellettiin niin sanottuja valmiiksi tehtyjä generointipohjia (ks. liite 1). Jokaiselle toimilohkoikonille tehtiin oma generointipohja. Generointipohjassa toimilohkoikoneita on 50 kappaletta, ja niiden muutettavat tiedot ovat yksilöllisiä.

Seuraavaksi alkoi käyttöliittymän suunnittelu. Käyttöliittymä muodostettiin kahdesta ikkunasta: version valintaikkunasta ja generoinnin ikkunasta. Valintaikkunassa (ks. liite 2) on pudotusvalikko, painike ja konsoli. Pudotusvalikossa on vaihtoehtoina kolme uusinta TIA Portal -versiota. Käyttäjä valitsee TIA Portal -version, johon generaattori aiotaan yhdistää. Painikkeella puolestaan siirrytään generointi-ikkunaan ja lähetetään versiotieto versionkäsittely toimintoon. Konsoliin tulostuu käyttäjälle ohjeita, virheilmoituksia ja generoinnin työvaiheita. Konsoli löytyy sekä version valintaikkunasta että generoinnin ikkunasta.

Generoinnin ikkunan (ks. liite 3) pohjaksi valikoitui TIA Openness -malliprojektin toteutus. Sitä laajennettiin ja muokattiin sopivaksi opinnäytetyössä toteutettavalle generaattorille. Ikkuna koostuu viidestä ryhmästä: TIA Portal, Project, Hardware, Blocks ja Screens. Ryhmät on jaettu toimintojen mukaan. *TIA Portal* -ryhmässä on valintanapit TIA Portalin käyttöliittymälle ja kaksi painiketta, joilla voi aukaista ja sulkea TIA Portal -prosessin. *Project*-ryhmässä puolestaan on painikkeita projektinhallintaan.

Projektin voi aukaista, tallentaa ja sulkea. Generaattorin voi yhdistää myös projektiin, joka on jo aukaistu aikaisemmin ennen generaattoria. *Hardware*-ryhmässä on tekstikenttiä, joihin kirjoitetaan laitteen nimi, Siemensin tuoteartikkelinumero ja versio. Painikkeella voidaan lisätä haluttu laite projektiin. *Blocks*-ryhmän alla on toimilohkojen generointiin tarvittavat objektit. Tekstikenttiä on kaksi. Ensimmäiseen kirjoitetaan projektin ohjelmoitavan logiikan nimi, jossa toimilohko sijaitsee. Toiseen kirjoitetaan toimilohkon nimi, josta generoidaan muita toimilohkoja. Painikkeita on neljä: toimilohkon tuominen projektista, generoitavan tiedon lataaminen ohjelmaan, generoitavan toimilohkon lataaminen ohjelmaan ja generoitujen toimilohkojen vieminen takaisin TIA Portal -projektiin. *Screens*-ryhmän alla on vastaavanlaiset tekstikentät ja painikkeet kuin *Blocks*-ryhmässä, mutta lisäksi tekstikenttä valvomokuvan numerolle ja valintapainike valvomokuvan tyypille. Valvomokuvien eri tyyppejä ovat normaali kuva, pop-up kuva tai toimilohkoikonin generointipohja. Valintapainikkeilla vaikutetaan generointitietoihin ja -tyyliin, sillä eri tyypit generoituvat hieman eri tavalla.

6.4 Ohjelmointi

Ohjelman rakenne koostuu pääosin aliohjelmista, joita kutsutaan käyttöliittymän painikkeilla. Aliohjelmien väliseen tiedonsiirtoon käytettiin globaaleja muuttujia.

6.4.1 Versiokäsittely ja ohjelmointirajapinnan luominen

TIA Openness -generaattorin oli toimittava kolmella uusimmalla TIA Portal -versiolla. TIA Openness -rajapinta muodostetaan DLL-tiedostoilla, joiden avulla päästään käsiksi TIA Portalin objekteihin ja toimintoihin. Jokaisella versiolla on omat DLL-tiedostonsa, jotka sijaitsevat kyseisen version asennuskansiossa. Käyttäjä on asennusvaiheessa määrittänyt asennuskansion polun ja polku tallennetaan rekisteriavaimeen. Jotta oikea versio TIA Portalista saataisiin aukaistua, on käyttäjältä kysyttävä versiota. Käyttöliittymän version valintaikkunassa käyttäjä valitsee sen TIA Portal -version, johon generaattori halutaan yhdistää. Valinta lähettää valitun version rekisteriavaimen polun AssemblyResolve-aliohjelmaan, joka lukee rekisteriavaimen ja lataa valitun version DLL-tiedostot rekisteriavaimeen tallennetusta polusta.

Seuraavaksi generointi-ikkunasta aukaistaan TIA Portal tai yhdistetään auki olevaan TIA Portal -prosessiin. Jos käyttäjä haluaa yhdistää jo auki olevaan TIA Portal -prosessiin, TIA Portal Openness -palomuuri kysyy käyttäjältä, hyväksytäänkö generaattorin yhdistäminen TIA Portal -prosessiin vai ei.

6.4.2 Projektitiedostojen tuonti ja vienti

Suunniteltu ja käännetty toimilohko tai kuva tuodaan TIA Portal -projektista generointipohjaksi. Generointipohjasta generoidaan useampia toimilohkoja tai kuvia, jotka viedään TIA Portal -projektiin. Projektitietojen tuonti ja vienti tapahtuu tekstimuotoisilla XML-tiedostoilla.

Ennen kuin toimilohkoa tai kuvaa voidaan tuoda TIA Portalista, täytyy käyttäjältä kysyä sen nimi ja sijainti eli laitteen nimi, jonka alla objekti sijaitsee. Ohjelma etsii ensimmäiseksi käyttäjän ilmoittaman laitteen eli ohjelmoitavan logiikan, PC-valvomon tai HMI-paneelin. Laitteen alla voi olla useita kansioita ja kansioita voi olla myös sisäkkäin. Jotta toimilohko tai kuva löytyisi, on jokainen kansio ja sen sisältö käytävä läpi. Tämä tapahtuu rekursiivisella funktiolla. Kun haluttu toimilohko tai kuva on löydetty, voidaan se tuoda Export-funktiolla. Export-funktio kääntää TIA Portalissa olevan objektin XML-tiedostoksi ja tallentaa sen käyttäjän valitsemaan kansioon.

Generoinnin jälkeen samaan kansioon on luotu x määrä uusia toimilohkoja tai kuvia. Määrä riippuu generontilistan pituudesta. Generoidut toimilohkot tai kuvat viedään TIA Portal projektiin Import-funktiolla. Import-funktio kääntää XML-tiedostot TIA Portalin objekteiksi.

6.4.3 XML-tiedostojen generointi

Käyttäjä valitsee haluamansa XML-tiedoston generointipohjaksi. Toimilohkon tai kuvan XML-tiedosto ladataan generaattorille käyttöliittymän painikkeella ja XML-tiedoston sisältö luetaan muuttujaan. Seuraavaksi generoitava tieto on ladattava generaattorille käyttöliittymän painikkeella. Generoitava tieto ladataan Excel-tiedostosta kaksiulotteiseen taulukkoon.

Generointi perustuu etsi ja korvaa- ja tallenna nimellä -toimintoihin. Generaattori tunnistaa Excel-taulukon suuruuden. Excel-taulukon ylimmän rivin soluissa on merkkijonoja, jotka löytyvät XML-tiedostosta. Nämä merkkijonot etsitään muuttujasta, johon XML-tiedoston sisältö on ladattu, ja korvataan saman sarakkeen alemman rivin solun merkkijonolla. Jokainen sarake edustaa yhtä muutettavaa tietoa ja jokainen rivi edustaa ns. yhtä toimilohkoa tai kuvaa (ks. kuvio 20). Kun kaikki sarakkeet on käyty läpi, luodaan muuttujasta eli muokatusta XML-tiedoston sisällöstä uusi tiedosto. Tiedoston nimeksi kirjoitetaan toimilohkon nimi. Ensimmäinen generoitu toimilohko on valmis. Nämä toiminnot sisältyvät FOR-silmukkaan. Jos Excel-tiedostossa on useampia rivejä, luetaan alkuperäisen XML-tiedoston sisältö uudelleen muuttujaan ja tehdään edellä esitetyt toiminnot uudelleen käyttäen aiempaa alemman rivin solujen merkkijonoja.

		Merkkijono 1	Merkkijono 2	Merkkijono 3	Merkkijono 4	Merkkijono 5
		A	B	C	D	E
Malli.xml	1	POSITIO	NIMI	KOMMENTTI	111.1	222.2
LI-1000.xml	2	LI-1000	Säiliön pinta	Pintamittaus ultraääni	0.0	100.0
TI-1001.xml	3	TI-1001	Säiliön lämpötila	Lämpötilamittaus PT-100	0.0	150.0
PI-1002.xml	4	PI-1002	Säiliön paine	Painemittaus	0.0	6.0
FI-1003.xml	5	FI-1003	Säiliön tulovirtaus	Virtausmittaus	0.0	60.0

Kuvio 20. Excel-taulukon malli

6.5 Testaus

6.5.1 Projektin toimilohkojen generointi

Suunnittelija loi tyyppilohkokirjaston toteutuksilla toimilohkon, jossa oli kaikki tarvittava taajuusmuuttajaohjatun moottorin ohjaamiseen ja sen hälytysten ja lukitusten käsittelyyn. Toimilohkoa on käytetty biokaasulaitos-projektissa, jossa on noin 40 taajuusmuuttaja ohjattua moottoria. Toimilohkoon syötettiin generoinnin aikana positiokoodi, piirin nimi ja I/O-osoitteet.

6.5.2 Testiprojekti käsin ja generaattorilla

Generaattorin testaamista varten luotiin testiprojekti. Testiprojektissa oli tarkoituksena tehdä generointilista ja generoida valmiista mittaustoimilohkosta ja moottoritoimilohkosta kuvitteellisen panimoyhtiön kymmenelle käymisastialle lämpötilamittaus, pintamittaus ja moottoriohjattu jäähdytyskäyttö. Lisäksi kullekin toimilohkolle generoidaan faceplate ja valvomokuvaan toimilohkoikoni. Nämä työvaiheet kellotetaan. Testiprojektissa pääsee testaamaan kaikkia eri generointi mahdollisuuksia. Generoinnin jälkeen tehdään toimilohkot, faceplatet ja toimilohkoikonit käsin kellottaen työvaiheisiin kuluva aika. Lopuksi vertaillaan, kuinka paljon aikaa kului generoimalla ja käsin tekemällä.

6.6 Käyttöohjevideot

Käyttöohjeeksi tehtiin videoita. Opinnäytetyöntekijä nauhoitti näyttöään ja kommentointiaan videolle samalla, kun generoi testiprojektiin tarvittavat toimilohkot, toimilohkojen datalohkot, toimilohkoikonit ja faceplatet. Tässä tunnin mittaisessa videossa esitellään testiprojekti, generaattori ja generoinnin eri vaiheet eli koko prosessi alusta loppuun. Lisäksi nauhoitettiin kommentoimattomia ja lyhyempiä videoklippejä eri generoinnin vaiheista ja tyyleistä.

7 Tulokset

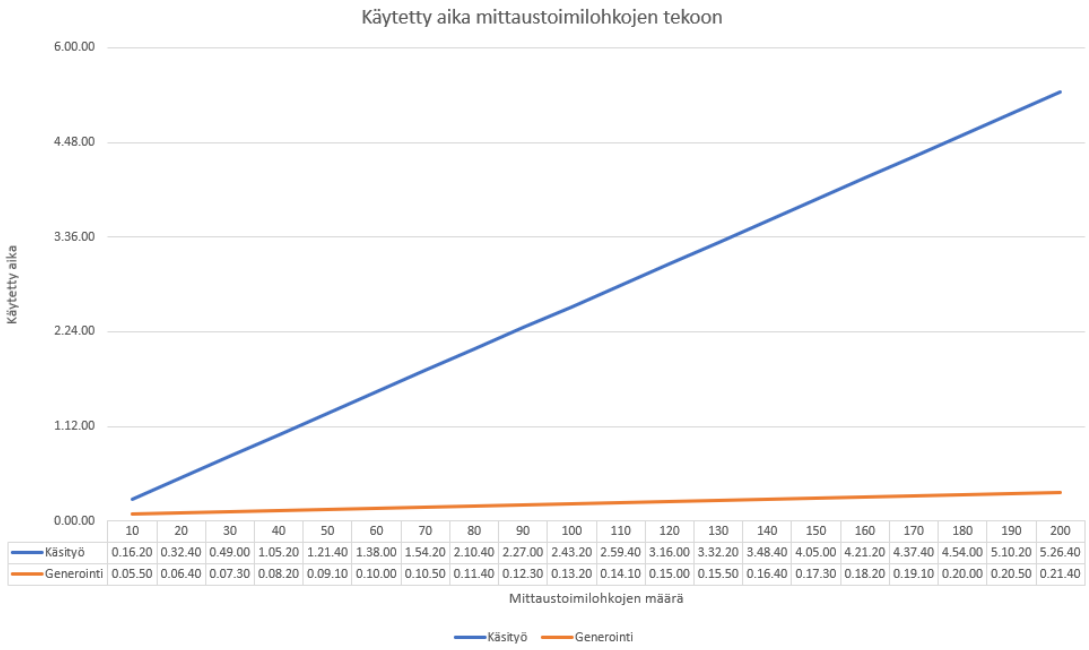
Opinnäytetyössä kehitettiin toimilohko- ja kuvageneraattori, jolla yritettiin ratkaista ongelma ja saada vastauksia tutkimuskysymyksiin. Generaattorin toimivuutta arvioitiin haastattelemalla loppukäyttäjiä ja kellottamalla heidän sekä opinnäytetyöntekijän työskentelyä. Generaattori täytti sille asetetut tavoitteet, opinnäytetyön ongelmaan löytyi ratkaisu ja tutkimuskysymyksiin saatiin vastaukset.

Generaattorille oli useita eri tavoitteita. Sen oli toimittava kolmella eri ohjelmointityökalun versiolla. Generaattorilla onnistuttiin generoimaan TIA Portalin versioissa 14

SP1, 15 ja 15.1. Generointidata syötettiin generaattorille Excel-tiedostosta, joka oli myös yksi generaattorin tavoitteista.

Tavoitteeksi oli määritetty, että generaattorilla voidaan generoida toimilohkot, toimilohkojen datalohkot, toimilohkoikonit ja faceplatet käyttövalmiiksi. Generaattorilla onnistuttiin generoimaan käyttövalmiita tuotoksia, tosin yleensä projektin lähtötiedot eivät ole niin tarkat, että esimerkiksi toimilohkoon ei tarvitsisi enää palata sovellussuunnittelun muissa vaiheissa.

Testiprojektin tekemiseen osallistui opinnäytetyöntekijän lisäksi kaksi suunnittelijaa. Testiprojektiin kuluneet ajat ovat keskiarvoja kolmesta testaussuorituksesta. Testiprojektissa 20 mittaustoimilohkon tekemiseen kului käsityönä 32 minuuttia ja 40 sekuntia eli yhden toimilohkon tekemiseen meni minuutti ja 38 sekuntia. Mittaustoimilohko sisälsi kahdeksan kohtaa, joihin tehtiin muutoksia: mittauspiirin positio, kommentti, I/O-osoite, mittausalueen alaraja, mittausalueen yläraja, ylärajahälytyksen hälytysraja ja alarajahälytyksen hälytysraja. Samojen mittaustoimilohkojen tekemiseen generaattorilla kului kahdeksan minuuttia. Aikaan sisällytettiin myös generoinnin alkutoimenpiteet ja generointilistan tekeminen. Generoimalla aikaa säästettiin yhteensä 24 minuuttia ja 40 sekuntia eli minuutti ja 14 sekuntia per toimilohko. Tuloksien avulla on tehty laskelma työajan säästöstä suuremmilla toimilohkomäärillä (ks. kuvio 21). Käsityön määrä on suoraan verrannollinen kuluneeseen aikaan. Generoinnissa alkutoimenpiteet vievät suuremmissa toimilohkomäärissä vain muutamia kymmeniä sekunteja enemmän aikaa, jolloin yhden toimilohkon generointiin kuluva aika pienenee. Kuvion 21 laskelmassa alkutoimenpiteille on laskettu keskimäärin viisi minuuttia aikaa ja toimilohkon generoinnille sekä TIA Portaliin viemiselle yhteensä viisi sekuntia per toimilohko.

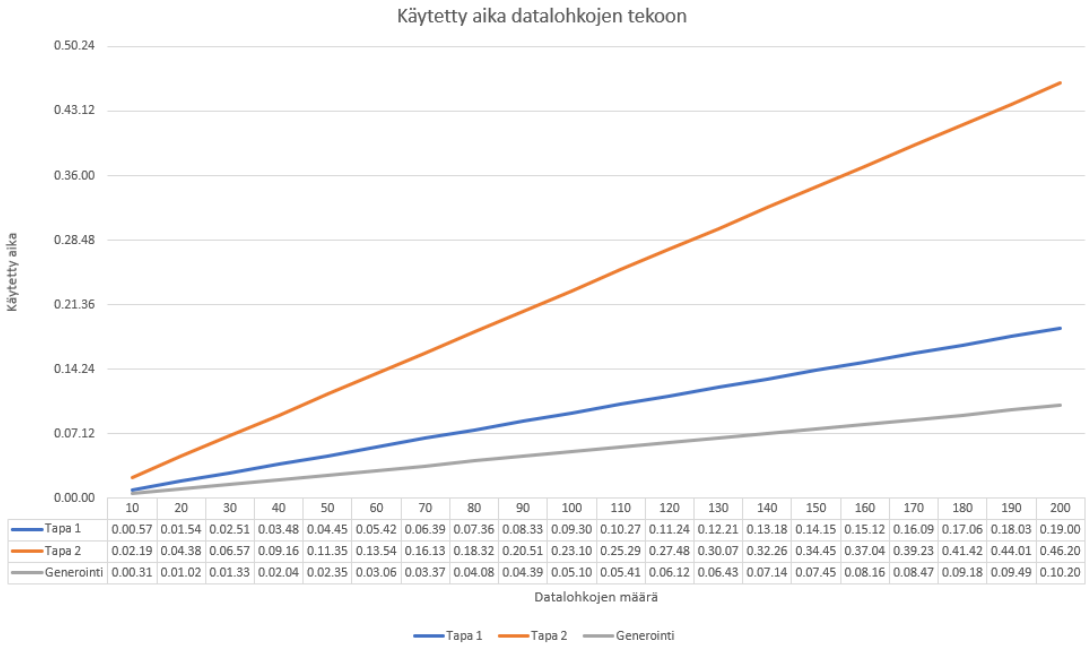


	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
Käsityö	0.16.20	0.32.40	0.49.00	1.05.20	1.21.40	1.38.00	1.54.20	2.10.40	2.27.00	2.43.20	2.59.40	3.16.00	3.32.20	3.48.40	4.05.00	4.21.20	4.37.40	4.54.00	5.10.20	5.26.40
Generointi	0.05.50	0.06.40	0.07.30	0.08.20	0.09.10	0.10.00	0.10.50	0.11.40	0.12.30	0.13.20	0.14.10	0.15.00	0.15.50	0.16.40	0.17.30	0.18.20	0.19.10	0.20.00	0.20.50	0.21.40

Kuvio 21. Laskelma mittaustoimilohkoihin käytettävästä ajasta

Virheitä käsityöllä oli aikeissa tulla kaksi, joista yksi huomattiin ja korjattiin. Generaattorille virheitä ei saatu aikaiseksi yhtään.

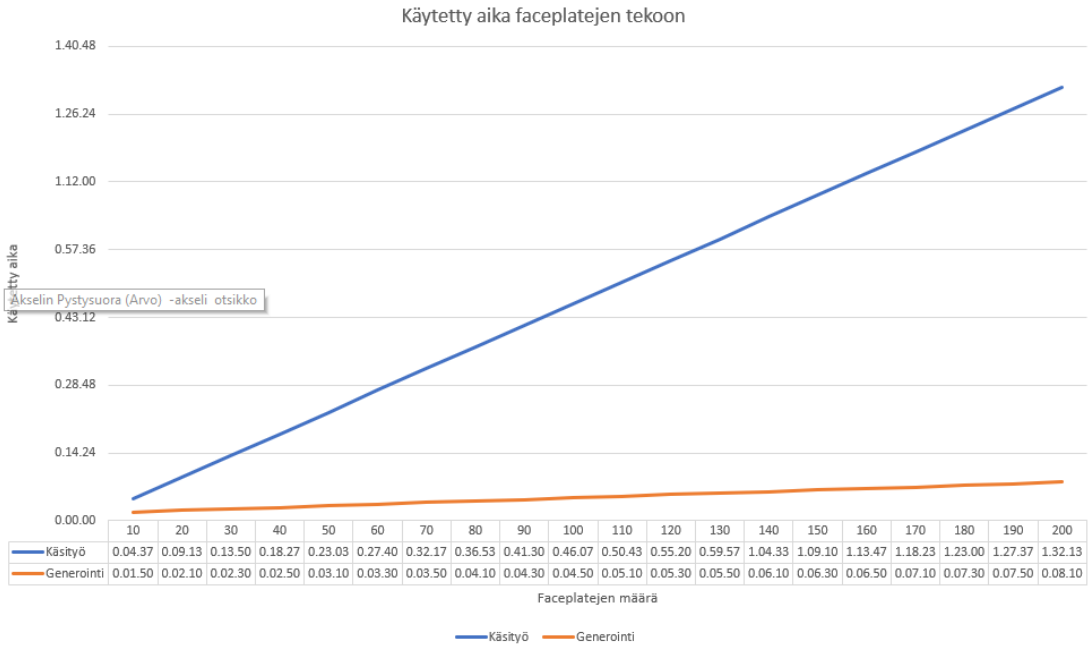
Vastaavanlainen vertailu tehtiin myös 30 toimilohkon datalohkolla. Käsityöllä datalohkot luodaan raahaamalla toimilohko organisointilohkoon tai yhdistämällä tyhjä datalohko toimilohkoon. Mittaustoimilohkoille 20 datalohkon tekemiseen käsityöllä ja ensimmäisellä tavalla kului aikaa minuutti ja 54 sekuntia. Moottoritoimilohkoille kymmenen datalohkon tekemiseen kului jälkimmäisellä tavalla kaksi minuuttia ja 19 sekuntia. Keskimäärin ensimmäisellä tavalla yhden datalohkon tekeminen kulutti aikaa 5,7 sekuntia ja toisella tavalla 13,9 sekuntia. Generoimalla aikaa kului kokonaisuudessaan minuutti ja 33 sekuntia, joka on 3,1 sekuntia per datalohko. Generoimalla aikaa kului vähemmän kuin käsityöllä. Tuloksien avulla datalohkoille on tehty vastaavanlainen laskelma kuin toimilohkoille (ks. kuvio 22). Generointilistalle ei datalohkojen kohdalla lasketa aikaa, koska lista on sama kuin toimilohkoille.



	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
Tapa 1	0.00.57	0.01.54	0.02.51	0.03.48	0.04.45	0.05.42	0.06.39	0.07.36	0.08.33	0.09.30	0.10.27	0.11.24	0.12.21	0.13.18	0.14.15	0.15.12	0.16.09	0.17.06	0.18.03	0.19.00
Tapa 2	0.02.19	0.04.38	0.06.57	0.09.16	0.11.35	0.13.54	0.16.13	0.18.32	0.20.51	0.23.10	0.25.29	0.27.48	0.30.07	0.32.26	0.34.45	0.37.04	0.39.23	0.41.42	0.44.01	0.46.20
Generointi	0.00.31	0.01.02	0.01.33	0.02.04	0.02.35	0.03.06	0.03.37	0.04.08	0.04.39	0.05.10	0.05.41	0.06.12	0.06.43	0.07.14	0.07.45	0.08.16	0.08.47	0.09.18	0.09.49	0.10.20

Kuvio 22. Laskelma datalohkoihin käytetystä ajasta

Toimilohkojen ja toimilohkojen datalohkojen jälkeen vertailua tehtiin faceplateille. Kymmenelle mittauslohkolle ja 20 moottoritoimilohkolle tehtiin faceplate. Faceplatejen tekemiseen käsityöllä aikaa kului kokonaisuudessaan 13 minuuttia ja 50 sekuntia, joka on keskimäärin 27,7 sekuntia per faceplate. Generaattorilla faceplateihin käytettiin kokonaisuudessaan aikaa kaksi minuuttia ja 30 sekuntia, josta 50 sekuntia meni listojen tekemiseen. Generointi itsessään kulutti 52 sekuntia eli 1,73 sekuntia per faceplate. Generoinnilla säästettiin aikaa myös faceplatejen tekemisessä. Tuloksista tehtiin laskelma faceplateihin käytetystä ajasta (ks. kuvio 23). Laskelmassa listan tekoon on varattu minuutti ja 30 sekuntia ja generoinnille kaksi sekuntia per faceplate.



Kuvio 23. Laskelma faceplateihin käytetystä ajasta

Lopulta toimilohkoikoneille tehtiin samanlainen vertailu kuin edeltäville. Käsin tekemällä 30 toimilohkoikonille kellotettiin 11 minuuttia ja 30 sekuntia, joka on 23 sekuntia per toimilohkoikoni. Generaattorilla tämä työvaihe puolestaan suoritettiin viidessä minuutissa ja 44 sekunnissa. Listojen tekemisen osuus tästä on kolme minuuttia. Generointiin itsessään käytettiin 5,8 sekuntia toimilohkoikonia kohden. Generoinnilla säästettiin aikaa myös toimilohkoikonien kanssa. Laskelmassa listan tekoon on varattu kolme minuuttia ja 30 sekuntia ja generoinnille seitsemän sekuntia per toimilohkoikoni (ks. kuvio 24).

	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
Käsityö	0.03.50	0.07.40	0.11.30	0.15.20	0.19.10	0.23.00	0.26.50	0.30.40	0.34.30	0.38.20	0.42.10	0.46.00	0.49.50	0.53.40	0.57.30	1.01.20	1.05.10	1.09.00	1.12.50	1.16.40
Generointi	0.04.40	0.05.50	0.07.00	0.08.10	0.09.20	0.10.30	0.11.40	0.12.50	0.14.00	0.15.10	0.16.20	0.17.30	0.18.40	0.19.50	0.21.00	0.22.10	0.23.20	0.24.30	0.25.40	0.26.50

Kuvio 24. Laskelma toimilohkoikoneihin käytetystä ajasta

8 Johtopäätökset ja pohdinta

Testaamisesta saadun datan ja loppukäyttäjien palautteen perusteella voidaan todeta, että ongelmaan löytyi ratkaisu ja tutkimuskysymyksiin vastaukset. Tutkimuskysymyksiä oli kolme. Ensimmäisessä kysymyksessä pohdittiin, voiko TIA Openness -ohjelmointirajapinnan avulla toteutetulla generaattorilla generoida täysin valmiita toimilohkoja, toimilohkojen datalohkoja, faceplateja ja toimilohkoikoneita käytettäväksi sovellussuunnittelun seuraavissa työvaiheissa. Vastaus on kyllä, mutta toimilohkojen kohdalla vain harvoin projektin lähtötiedot ovat alussa niin kattavat, että niitä ei tarvitsisi generoinnin jälkeen muokata. Toinen kysymys oli, saadaanko generoinnilla vähennettyä selvästi suunnittelijan toistotyötä ja siihen käytettyä työaikaa. Tähän vastaus saadaan testiprojektilla tehdyistä kellotuksista. Generoinnilla aikaa säästettiin koko projektissa 60 minuuttia ja 46 sekuntia, vaikka generoitavien objektien määrät olivat vähäisiä. Laskelmien perusteella laajemmissa projekteissa generoinnin hyöty on suurempi. Lisäksi toistuvat työvaiheet saatiin lähes kokonaan poistettua. Viimei-

senä kysymyksenä oli, saadaanko generoinnilla vähennettyä virheiden määrää. Testauksen aikana virheiden määrä oli hyvin pieni käsintehdyssä projektissa. Generaattorilla tehdyssä projektissa virheitä ei ilmennyt lainkaan. Virheiden määrää onnistuttiin siis vähentämään generoinnilla.

Generaattorin tavoitteisiin pääsy varmistettiin jo toteutuksen aikana, sillä se toteutettiin siten, että se täyttää sille asetetut tavoitteet. Lisäksi opinnäytetyöntekijän henkilökohtaisiin tavoitteisiin päästiin. Ohjelmointi Visual Basic -ohjelmointikielellä kehittyi runsaasti sekä lisäksi TIA Portalin -ohjelmointityökalun käytön tehokkuus lisääntyi opinnäytetyön aikaisen tutkimisen ja generaattorin kehittämisen ansiosta.

Testaamiseen osallistui kaksi suunnittelijaa opinnäytetyöntekijän lisäksi. Testaajien määrä oli vähäinen, mutta riittävä toteamaan generaattorin toimivuuden ja tuloksien luotettavuuden. Testaaminen oikeassa projektissa sekä testiympäristössä on mielestäni paras tapa saada realistisia tuloksia. Generoinnin tuloksiin eniten vaikuttava tekijä oli ehdottomasti kohteen generoitavien tietojen määrä. Suurin osa ajasta menee generointilistan tekemiseen. Generaattorin testaustuloksien osalta luotettavuusmittarina käytetty reliabiliteetti käy toteen. Testaustulokset ovat toistettavissa.

Tutkimusmenetelmänä kehittämistutkimus oli mielestäni sopivin. Muutokseen tähtäävällä kehittämisellä saavutettiin mittavia hyötyjä. Toistettavat työvaiheet automatisoitiin generaattorilla ja säästetty työaika pystytään käyttämään tärkeämpiin työvaiheisiin.

Työn tavoitteita ja toteuttamista pohdittiin haastatteluilla. Haastattelut olivat oikea tapa kerätä aineistoa havainnoinnin lisäksi, sillä haastateltavat olivat kokeeneempia suunnittelijoita ja generaattorin loppukäyttäjiä. Haastattelujen kohderyhmä oli oikea, mutta haastateltavien määrä jäi pieneksi. Generaattorille asetetut tavoitteet täyttyivät ja generaattorilla saatiin ratkaisu ongelmaan ja tutkimuskysymyksiin.

Tietoperustaa kirjoittaessa lähdekriittisyyteen on kiinnitetty erityisesti huomiota. Lähteinä käytettiin valmistajan materiaaleja, alan kirjallisuutta ja tutkimuksia. Tieto-

perustassa käsitellään tarpeellisia aiheita generaattorin ja sovellussuunnittelun ympäriltä. Tietoperustassa on käytetty useampia lähteitä tiedon oikeellisuuden varmistamiseksi ja luotettavuuden lisäämiseksi. Onnistunut lopputulos vahvistaa tietoperustan sisällön riittävyyttä ja luotettavuutta. Tietoperustan osalta tutkimuksen luotettavuusmittarina toimiva validiteetti käy mielestäni toteen, koska tutkimuksessa on tutkittu oikeita asioita. Tämän vahvistaa toimiva generaattori ja positiiviset testaustulokset.

Jatkokehitysideana generaattoriin voisi lisätä laitteiden, kuten ohjelmoitavien logiikkaohjaimien ja taajuusmuuttajien lisäämisen TIA Portal -projektiin ja niiden HW-tietojen generoinnin. Automaatiojärjestelmässä ja TIA Portal -projektissa saattaa olla useampi ohjelmoitava logiikkaohjain ja niissä kussakin kymmeniä I/O-kortteja. Lisäksi taajusmuuttajia voi olla kymmeniä. Laitteiden lisääminen ja niiden HW-konfiguroiminen generoimalla säästäisi suunnittelijan aikaa isommissa projekteissa.

Lähteet

Ajo, R., Hakonen, S., Harju, H., Järvi, J., Kaskes, K., Lenardic, E., Niukkanen, E., Nurminen, T., Ritala, P., Tolppanen, M. & Tommila, T. 2012. Laatu automaatiossa – Parhaat käytännöt. Automaatiosuunnittelun opas. Suomen Automaatioseura. Viitattu 26.2.2019.
https://www.automaatioseura.fi/site/assets/files/1367/laatuautomaatiossa.pdf.

A Tour of the C# Language. 2016. Microsoftin C#-ohjelmointikielen dokumentaatio. Microsoft. Viitattu 24.3.2019.
https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/index.

Automaatio-ohjelmistot. 2016. Siemens. Viitattu 10.3.2019.
http://www.siemens.fi/fi/industry/teollisuuden_tuotteet_ja_ratkaisut/tuotesivut/automaatiotekniikka/automaatio_ohjelmistot.htm.

Basics on HMI Faceplates. 2017. SIMATIC Comfort Panels, Runtime Advanced and WinCC (TIA Portal). Siemensin ohje. Viitattu 14.4.2019.
https://cache.industry.siemens.com/dl/files/632/68014632/att_924738/v2/68014632_faceplates_instruction_doku_v14_en.pdf.

Berger, H. 2014. Automating with SIMATIC S7-400 inside TIA Portal: Configuring, Programming and Testing with STEP 7 Professional. Hoboken: John Wiley & Sons.

Build a C# Hello World application with the .NET Core SDK in Visual Studio 2017. Microsoftin .NET Core -kehitysalustan dokumentaatio. Viitattu 24.3.2019.
https://docs.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio.

Build a Visual Basic Hello World application with the .NET Core SDK in Visual Studio 2017. Microsoftin .NET Core -kehitysalustan dokumentaatio. Viitattu 24.3.2019.
https://docs.microsoft.com/en-us/dotnet/core/tutorials/vb-with-visual-studio.

Cycle and response times. 2018. SIMATIC S7-1500, S7-1500R/H, ET 200SP, ET 200pro. Siemensin ohje. Viitattu 13.4.2019.
https://cache.industry.siemens.com/dl/files/558/59193558/att_895996/v1/s71500_cycle_and_reaction_times_function_manual_en-US_en-US.pdf?download=false.

Delivery release of TIA Portal V14 SP1. 2017. Siemensin tiedote. Viitattu 1.3.2019.
https://support.industry.siemens.com/cs/document/109744304/delivery-release-of-tia-portal-v14-sp1?dti=0&pnid=14672&lc=en-WW.

Delivery release SIMATIC STEP 7 Professional V11. 2011. Siemensin tiedote. Viitattu 1.3.2019.
https://support.industry.siemens.com/cs/document/49639088/delivery-release-simatic-step-7-professional-v11?dti=0&pnid=14672&lc=en-WW.

Delivery release TIA Portal V15. 2017. Siemensin tiedote. Viitattu 1.3.2019.
https://support.industry.siemens.com/cs/document/109752224/delivery-release-tia-portal-v15?dti=0&pnid=14672&lc=en-WW.

Delivery release TIA Portal V15.1. 2018. Siemensin tiedote. Viitattu 1.3.2019.
https://support.industry.siemens.com/cs/document/109758794/delivery-release-tia-portal-v15-1?dti=0&pnid=14672&lc=en-WW.

Extensible Markup Language (XML) 1.0. 2008. W3C Recommendation 26 November 2008. World Wide Web Consortiumin (W3C) suositus. 5. p. Viitattu 20.3.2019.
https://www.w3.org/TR/xml/.

Gavin, B. 2018. What Is An XML File (And How Do I Open One)?. Artikkeli How-To-Geek -teknologiaverkkolehdessä. Viitattu 20.3.2019.
https://www.howtogeek.com/357092/what-is-an-xml-file-and-how-do-i-open-one/.

Heikkilä, T. 2014. Tilastollinen tutkimus. 9. uud. p. Helsinki: Edita Publishing. Viitattu 12.4.2019.
http://www.tilastollinentutkimus.fi/1.TUTKIMUSTUKI/KvantitatiivinenTutkimus.pdf.

Hirvonen, J., Hukki, K., Strömman, M. & Tommila, T. 2010. Automaatiosuunnittelun prosessimalli – Yhteiset käsitteet verkottuneen suunnittelun perustana. Perustuu vuosina 2005-2007 tehtyyn ANTI-2-tutkimusprojektiin. Suomen Automaatioseura. Viitattu 26.2.2019.
https://www.automaatioseura.fi/site/assets/files/1367/automaatiosuunnittelun_prosessimalli.pdf.

John, K.-H. & Tiegelkamp, M. 2010. IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids. 2nd ed. Heidelberg: Springer

Kananen, J. 2015. Opinnäytetyön kirjoittajan opas: Näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä: kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kippo, A. & Tikka, A. 2008. Automaatiotekniikan perusteet. Helsinki: Edita.

Lahtonen, T. 2016. C#-perusteet. Informaatioteknologian tiedekunnan luentomateriaali. Jyväskylä: Jyväskylän yliopisto. Viitattu 24.3.2019.
http://appro.mit.jyu.fi/gko/luennot/luento1/#TOC4.

Openness. 2018. Automating creation of projects. Siemensin järjestelmäohjekirja. Viitattu 16.3.2019.

https://cache.industry.siemens.com/dl/files/163/109477163/att_926042/v1/TIAPortalOpennessenUS_en-US.pdf.

Palvelut. 2019. PCS-Engineering Oy yrityksen kotisivut. Viitattu 16.2.2019. https://pcs-engineering.fi/palvelut/.

Project management Consulting Services. 2018. PCS-Engineering Oy yrityksen esite.

Programming Guideline for S7-1200/S7-1500. 2014. STEP 7 (TIA Portal) and STEP 7 Safety in TIA Portal. Siemensin ohje. Viitattu 6.3.2019. http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s71200/programming-guideline-for-s71200-s71500_2014-09_en.pdf.

Saaranen-Kauppinen, A. & Puusniekka, A. 2006. KvaliMOTV – Menetelmäopetuksen tietovaranto. Verkko-oppikirja. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 20.2.2019. https://www.fsd.uta.fi/fi/tietoarkisto/julkaisut/kvalimotv.pdf.

Sivonen, V.-M. 2004. Ohjelmointikielen periaatteet: C#-kieli. Tietojenkäsittelytieteen laitoksen seminaariesitelmä. Helsinki: Helsingin yliopisto. Viitattu 24.3.2019. https://www.cs.helsinki.fi/u/pohjalai/k04/ohpe/seminar/Sivonen-CSharp.pdf.

Standards Compliance according to IEC 61131-3. 2013. Siemensin tukisivusto. Viitattu 13.3.2019. https://support.industry.siemens.com/cs/attachments/50204938/IEC_61131_compliance_e.pdf.

TIA Portal. 2017. Teollisuusautomaation ohjelmistoalusta. Viitattu 1.3.2019. http://www.siemens.fi/fi/industry/teollisuuden_tuotteet_ja_ratkaisut/tuotesivut/tia_portal.php.

TIA Portal Openness. 2019. Introduction and Demo Application. Siemensin TIA Portal Openness esittely. Viitattu 16.3.2019. https://support.industry.siemens.com/dl/files/692/108716692/att_984738/v1/108716692_TIA_Openness_GettingStartedAndDemo_V15_1_en.pdf.

TIA Portal Openness. 2017a. Siemensin TIA Portal Openness diaesitys Digitalization Days 2017 -tapahtumassa. Viitattu 16.3.2019. https://www.digitalizationdays.ch/uploads/pdf/01_TIA_Portal_Openness.pdf.

TIA Portal Openness. 2017b. Siemensin ohjelmistoesittely verkkosivuilla. Viitattu 16.3.2019. https://www.industry.siemens.com/topics/global/en/tia-portal/software/details/Pages/openness.aspx.

TIA Portal Tutorial #02. 2012. Project tree. Opetusvideo. Youtube-videopalvelu. Julkaistu 24.7.2012. Viitattu 6.3.2019. https://www.youtube.com/watch?v=Xw1EJ9Lo_Tk.

Totally Integrated Automation Portal. 2013. Siemensin lehdistötiedote. Viitattu 1.3.2019.
http://www.siemens.fi/pool/products/industry/iadt_is/tuotteet/tia_portal/tia_portal_brochure.pdf.

Vick, P. & Wischik, L. 2017. The Microsoft® Visual Basic® Language Specification. Microsoftin Visual Basic ohjelmointikielen määrittely. Microsoft. Viitattu 24.3.2019.
http://ljw1004.github.io/vbspec/Visual%20Basic%20Language%20Specification.pdf?raw=true.

Vilpas, P. 2013. Kvantitatiivinen tutkimus. Moniste. Vantaa: Metropolia ammattikorkeakoulu. Viitattu 12.4.2019.
https://users.metropolia.fi/~pervil/kvantsu/Moniste.pdf.

Visual Basic. N.d. PCMag-tietokonekuvalehden verkkosivujen tietosanakirja. Viitattu 24.3.2019
https://www.pcmag.com/encyclopedia/term/53986/visual-basic.

Visual Basic Guide. 2018. Microsoftin Visual Basic -ohjelmointikielen dokumentaatio. Viitattu 24.3.2019
https://docs.microsoft.com/en-us/dotnet/visual-basic/

Walker, M. J. 2012. The programmable logic controller: its prehistory, emergence and application. Väitöskirja The Open University -yliopistoon. The Open University. Viitattu 13.3.2019.
http://oro.open.ac.uk/54687/1/594090.pdf.

Welcome to the Visual Studio IDE. 2019. Microsoftin Visual Studio -kehitysympäristön dokumentaatio. Viitattu 24.3.2019.
https://docs.microsoft.com/fi-fi/visualstudio/get-started/visual-studio-ide?view=vs-2017.

Your gateway to automation in the digital enterprise. 2018. Totally Integrated Automation Portal. Siemensin esite. Viitattu 1.3.2019.
https://c4b.gss.siemens.com/resources/images/articles/dffa-b10461-01-7600.pdf.

Liitteet

Liite 1. Moottoritoimilohkoikonin generointipohja (salassa pidettävä)

Liite 2. Generaattorin version valintaikkuna (salassa pidettävä)

Liite 3. Generaattorin generointi-ikkuna (salassa pidettävä)