TURUN YLIOPISTO

PLC-koodin standardisointi ja versionhallinta

Automaatiotekniikka
Kone- ja materiaalitekniikan laitos
Kandidaatintutkielma

Laatija:
Riina Nyrkkö

21.4.2026
Turku

Kandidaatintutkielma

Tutkinto-ohjelma, oppiaine: Automaatiotekniikka
Tekijä: Riina Nyrkkö
Otsikko: PLC koodin standardisointi ja versionhallinta
Ohjaajat: Jari Böling, Kari Mahlberg
Sivumäärä: 28 sivua
Päivämäärä: 21.4.2026

Tämän kandidaatintutkielman tavoitteena on tarkastella ohjelmoitavien logiikoiden (PLC) ohjelmistosuunnittelun standardisointia ja versionhallinnan kehittämistä nykyaikaisessa teollisuusympäristössä. Tutkielman toimeksiantajana toimii Ioncor Oy (entinen Valmet Automotive EV Power Oy), jonka liiketoiminta keskittyy sähköajoneuvojen akkujärjestelmien kehittämiseen ja valmistamiseen. Akkujärjestelmien valmistukseen käytetään pitkälle automatisoituja valmistuslinjoja.

Tutkielman teoreettisessa osuudessa perehdytään kansainväliseen standardiin, joka määrittelee PLC-ohjelmoinnin perusteet, kielet ja rakenteen. Työssä käydään läpi standardin mukaiset ohjelmointikielet ja tarkastellaan sen modulaarista arkkitehtuuria. Erityisesti kiinnitetään huomiota siihen, miten tilaajayritys voi parantaa suunnittelun tehokkuutta ja vähentää virhealttiutta modulaarisia rakenteita hyödyntämällä. Modulaarisuuden hyötyihin on syvennytty osallistumalla tuotantokoodin kehittämiseen.

Empiirisessä osuudessa on käsitelty tilaajayrityksen siirtymää ohjelmistojen versionhallinnassa Siemensin Teamcenter -järjestelmään. Tutkielmassa tutkittiin siirtymän hyötyjä ja vanhan versiohallintatavan haasteita.

Tutkielma osoittaa, että koodin modulaarisuus ja versionhallinta ovat välttämättömiä edellytyksiä monimutkaisten automaatioprojektien hallitsemiseksi. Siirtyminen keskitettyyn versionhallintaan parantaa koodin eheyttä, helpottaa vianetsintää ja tukee ohjelmakirjastojen käyttöä.

Avainsanat: Logiikkaohjelmointi, standardisointi, versionhallinta, modulaarisuus, automaatio

Sisällys

1 Johdanto

Tämän tutkielman tarkoituksena on tarkastella PLC-koodin standardisoinnin merkitystä ja versionhallintajärjestelmän muutosta tilaajayrityksessä. Ohjelmistoprojektien kasvaessa ja useiden suunnittelijoiden ja kunnossapidon työskennellessä saman koodin parissa, koodin rakenteen yhtenäisyys ja muutoshistorian hallinta nousevat kriittisiksi tekijöiksi. Ilman selkeiden standardien noudattamista ja keskitettyä versionhallintaa projektit altistuvat inhimillisille virheille, kuten ohjelman ylikirjoittamiselle ja vaikeaselkoisuudelle.

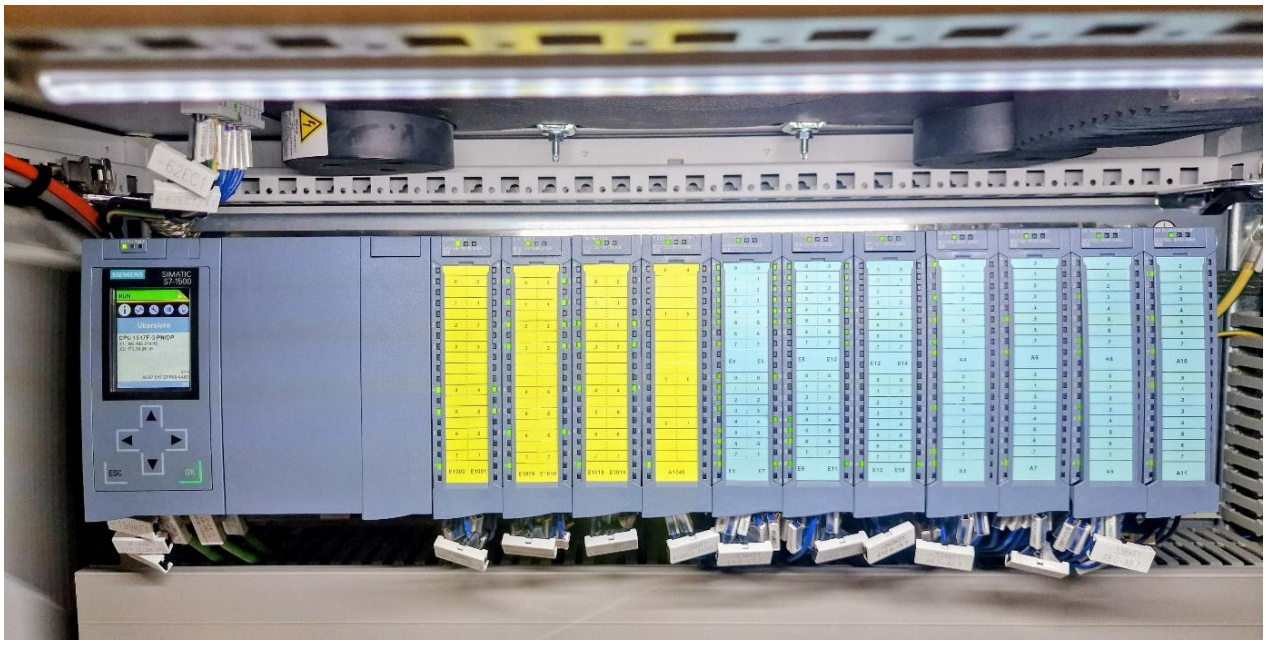
Tutkielman teoreettisessa osuudessa syvennytään kansainväliseen IEC 61131-3 -standardiin, joka määrittelee PLC-ohjelmoinnin perustan. Standardin tarkoituksena on yhtenäistää eri laitevalmistajien välisiä eroja ja tarjota raamit modulaariselle suunnittelulle. Standardisointi on edellytys ohjelmistojen uudelleenkäytettävyydelle ja hyödyntämiselle useassa projektissa.

Empiirisessä osuudessa tarkastellaan kohdeyrityksen siirtymää verkkolevypohjaisesta versionhallinnasta integroituun Teamcenter-versionhallintaan. Havainnot perustuvat ohjelmistoprojekteissa havaittuihin haasteisiin ja uuden järjestelmän tuomiin prosessiparannuksiin. Lisäksi osana tutkielmaa on osallistuttu PLC-koodin käyttöönottoon, vianetsintään, korjaamiseen ja kehittämiseen.

Tämän tutkielman tekniset ratkaisut ja esimerkit on toteutettu tilaajayrityksen Siemensin TIA Portal -ohjelmointiympäristössä. Käytännön toteutuksessa on hyödynnetty useampaa Siemens SIMATIC S7-1500 -sarjan ohjelmoitavaa logiikkaa (Kuva 1).

Vaikka ohjelmoinnin periaatteet pohjautuvat kansainväliseen standardiin, voi eri laitevalmistajien välillä olla eroja standardin implementoinnissa. Tässä esitellyissä ohjelmointitavoissa on käytetty lähteinä suoraan mm. Siemensin manuaaleja, jotka saattavat poiketa muiden laitevalmistajien arkkitehtuurista.

Tutkielmassa on hyödynnetty tekoälyä kieliasussa ja lähteiden löytämiseen. Kirjoitettu teksti ja ajatukset ovat kuitenkin kirjoittajan omia.



Kuva 1. Tutkielmassa käytettiin vastaavanlaisia Siemens SIMATIC S7-1500 -sarjan ohjelmoitavia logiikoita. Kuvassa näkyy myös tyypillinen kokoonpano, johon kuuluu prosessorin lisäksi tulo- ja lähtömoduulit. Keltaisen väriset yksiköt ovat turvapuolen moduuleja.

1.1 Tilaajayritys

Tutkielman tilaajayrityksenä toimii IONCOR Oy, joka tunnettiin aiemmin nimellä Valmet Automotive EV Power Oy. Yrityksen juuret ulottuvat vuonna 1968 perustettuun, pitkästä autonvalmistushistoriastaan tunnettuun Valmet Automotive Oyj -konserniin.

Yhtiö perustettiin vastauksena sähköistyvän liikenteen murrokseen: sen ydinliiketoiminta keskittyy korkeateknologisten akkujärjestelmien kehittämiseen ja valmistamiseen. Toiminta alkoi Uudessakaupungissa vuonna 2018, ja se laajeni nopeasti: jo vuonna 2019 yritys avasi akkutehtaan Salon entisiin Nokian matkapuhelinvalmistuksen tiloihin. Pian tämän jälkeen tuotanto laajeni myös Saksan Kirchardtiin.

IONCOR Oy irtaantui omaksi yhtiökseen vuonna 2024, jolloin se jatkoi toimintaansa Valmet Automotiven tytäryhtiönä. Omistuspohja muuttui edelleen vuonna 2025, kun yritys myytiin ja suurimmaksi omistajaksi tuli Finnish Minerals Group. Nykyisin yritys kehittää innovaatioita raskaan liikenteen, hyötyajoneuvojen ja liikkuvien työkoneiden sähköistämiseen.

Akkujärjestelmien valmistus on vaativa prosessi, joka edellyttää pitkälle vietyä ja tehostettua automatisointia. Korkea automaatioaste on välttämätön, jotta tuotannossa saavutetaan vaadittu tarkkuus, toistettavuus ja turvallisuus. Tuotantomäärät ja vaatimukset muuttuvat jatkuvasti, mikä edellyttää jatkuvaa linjojen laajentamista, päivittämistä ja rakentamista.

Ohjelmiston rooli ympäristössä on kriittinen. Jotta tuotanto voi mukautua nopeasti muutoksiin, PLC-ohjelmakoodin on oltava luonteeltaan modulaarista ja hyvin dokumentoitua. Siirrettävällä koodilla varmistetaan, että aiemmin testattu ja hyväksytty logiikka on hyödynnettävissä myös myöhemmissä projektin vaiheissa.

2 Ohjelmoitava logiikka

Teollisuusautomaatiossa ohjelmoitavan logiikan (*Programmable Logic Controller*, PLC) kehitys on kulkenut 1960-luvun relelogiikan korvaajista kohti monimutkaisia ja moderneja ohjausjärjestelmiä. Alun perin PLC-järjestelmät suunniteltiin kestämään kovia tehdasolosuhteita, kuten pölyä, kuumuutta ja sähköisiä häiriöitä, mikä erotti ne perinteisistä tietokoneista jo varhaisessa vaiheessa [1]. Kansainvälinen standardi IEC 61131-1 määrittelee ohjelmoitavan logiikan ”digitaalisesti toimivaksi elektroniseksi järjestelmäksi, joka on suunniteltu kestämään teollisuusympäristössä ja joka käyttää ohjelmoitavaa muistia käyttäjälähtöisten ohjeiden sisäiseen tallentamiseen tiettyjen toimintojen, kuten logiikan, sekvensoinnin, ajoituksen, laskennan ja aritmetiikan toteuttamiseksi erilaisten koneiden tai prosessien ohjaamiseksi” [2].

2.1 Siirtymä relelogiikasta ohjelmalliseen ohjaukseen

Ennen PLC-laitteiden yleistymistä teollisuuden ohjaus perustui fyysisiin releisiin, kytkimiin ja mekaanisiin ajastimiin. Relelogiikan käytössä oli kuitenkin monia ongelmia ja suurten relekaappien ylläpitoon käytettiin paljon manuaalista työtä. Asennusvaiheessa releet vaativat runsaasti johdotustöitä, ja myöhemmin pienikin muutos prosessissa saattoi tarkoittaa satojen johtojen uudelleenkytkentää. Mekaaniset komponentit olivat vikaherkkiä ja vianetsintä vaikeaa [1].

PLC:n kehitys tarkoitti uutta aikakautta teollisuusautomaatiossa. Sen keskeisin etu oli ohjelmoitavuus: logiikka voitiin muuttaa ilman kalliita ja aikaa vieviä muutoksia fyysiseen laitteistoon, mikä vähensi merkittävästi seisokkiaikoja ja ylläpitokustannuksia. Tämä teknologinen harppaus oli merkittävä etu monimutkaisten prosessien tarkempaan, tehokkaampaan ja luotettavampaan automatisointiin [3].

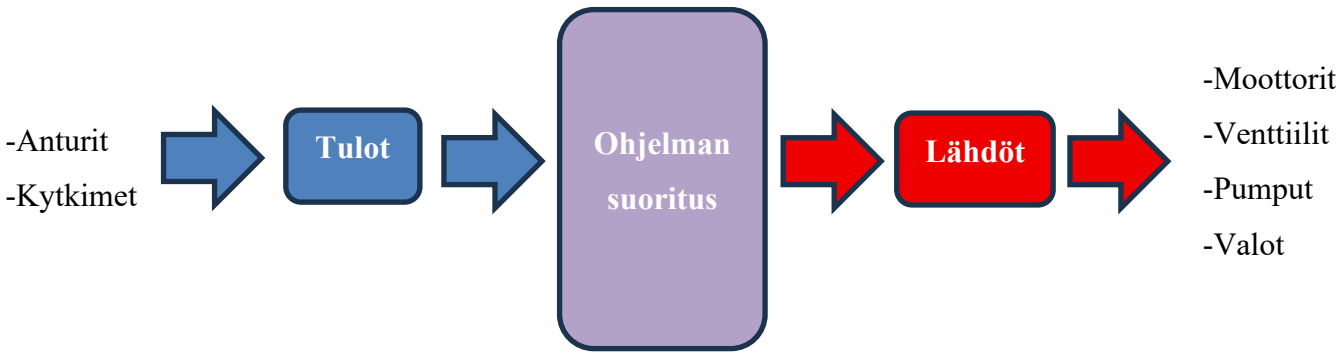
2.2 Tekninen perusta

PLC koostuu yksinkertaisimmillaan seuraavista laitteista:

- virtalähde
- keskusyksikkö (*Central Processing Unit*, CPU)
- muisti
- tulo- ja lähtömoduulit (I/O)
- kommunikaatiomoduulit

I/O-moduulien kautta PLC vastaanottaa kenttälaitteiden (esim. antureiden) signaaleja ja ohjaa toimilaitteita (esim. moottoreita ja venttiilejä). Signaalit voivat olla digitaalisia tai analogisia. PLC:n muistiin ladataan ohjelma, minkä jälkeen ohjain lukee tulosignaalit ja ohjaa lähtösignaaleja ohjelman määrittelemällä tavalla [1].

PLC-järjestelmien arkkitehtuuri perustuu sykliseen suoritukseen. Aluksi ohjain tarkistaa oman tilansa ja antaa virheen, jos laitteistossa on jotain vikaa. Sen jälkeen se lukee tulot (inputit) ja tallennetaan ne PLC:n muistiin. Kun arvot on tallennettu, ohjelmalogiikka suoritetaan ylhäältä alaspäin näiden tallennettujen tulojen perusteella. Ohjelman suorituksen jälkeen lähtöarvot kirjoitetaan ohjelman mukaisiksi (Kuva 2). Tämän jälkeen sykli aloitetaan alusta ja näitä neljää sykliä toistetaan jatkuvasti. Koko ohjelma siis suoritetaan jokaisella kierroksella. Syklien suoritusajat mitataan usein millisekunneissa [4].



Kuva 2. Ohjelmoitavan logiikan looginen rakenne, joka havainnollistaa tulosignaalien ja lähtösignaalien välistä prosessia ohjauslogiikassa

2.3 PLC-laitteisto verrattuna PC-pohjaiseen ohjaukseen

PLC:ssä on selkeitä etuja PC-pohjaiseen ratkaisuun verrattuna. Se ei vaadi käyttäjältä suurta ymmärrystä korkean tason ohjelmointikielistä ja tietokoneiden toiminnasta, vaan on käyttäjäystävällisempi muun muassa graafisten ohjelmointikielien ansiosta. Logiikkaohjain on myös suunniteltu teollisuuskäyttöön, joten se on rakennettu kestämään tärinää, pölyä sekä lämpötila- ja kosteusvaihteluita [1]. Sen yksi tärkeimpiä ominaisuuksia onkin hyvä luotettavuus ja häiriönsietokyky [5].

PLC-laitteet on varustettu teollisuusstandardien mukaisilla I/O-liitännöillä, joiden modulaarinen rakenne mahdollistaa järjestelmän laajentamisen tarpeen mukaan ilman koko ohjainyksikön vaihtamista. PLC ei myöskään sido suunnittelijaa tiettyyn ohjelmakieleen, vaan se tukee useampia ohjelmointikieliä. Laitteet suorittavat myös jatkuvaa itsediagnostiikkaa, minkä ansiosta käyttäjän on helppo paikallistaa ongelma vikatilanteissa [5].

3 PLC-ohjelmoinnin standardisointi

Standardisointi on keskeinen työkalu PLC-ohjelmoinnin ohjelmiston laadun ja uudelleenkäytettävyyden parantamiseksi. Ennen yleisen standardin kehittämistä PLC-valmistajien välillä oli suuria eroja. Yhtenäisen standardin käyttöönotto on mahdollistanut ohjelmoijien siirtymisen valmistajalta toiselle ilman, että ohjelmakoodi tai tiedonsiirto täytyy kirjoittaa kokonaan uudelleen [6].

3.1 IEC 61131

IEC 61131 on *International Electrotechnical Comissionin* (IEC) kehittämä kansainvälinen standardisarja ohjelmoitaville logiikkaohjaimille. Se on kymmenosainen [7] ja tarkoitettu viitteelliseksi ohjeeksi valmistajille [6]. Standardin osat on esitetty taulukossa 1.

Taulukko 1. IEC 61131 -standardin osat [8].

Nro	**Otsikko**	**Selitys**
1	General information	PLC-järjestelmien keskeiset ominaisuudet
2	Equipment requirements and tests	PLC-laitteiden tekniset vaatimukset
3	Programming languages	PLC-ohjelmoinnin ohjelmointikielet, syntaksi ja semantiikka
4	User Guidelines	Ohjeet toimittajan ja käyttäjän väliseen viestintään
5	Communications	PLC:n kommunikointi muiden laitteiden kanssa
6	Functional Safety	PLC- ja oheislaitteiden toiminnallisen turvallisuuden vaatimukset
7	Fuzzy control programming	Sumea säätöohjelmointi standardin mukaisilla kielillä
8	Guidelines for the application and implementation of programming languages	Ohjeet ohjelmointikielten käyttöönottoon ja toteutukseen
9	Single-drop digital communication interface for small sensors and actuators	Pienten antureiden ja toimilaitteiden kommunikaatiorajapinta
10	PLC open XML Exchange Format	XML-pohjainen tiedostomuoto standardin mukaisten projektien vientiin ja tuontiin

3.2 IEC 61131 -standardin mukaiset ohjelmointikielet

IEC 61131-3 on standardin osista kaikkein tärkein ja käytetyin, sillä siinä määritellään PLC-ohjelmoinnin ohjelmointikielet, niiden syntaksi ja semantiikka. Standardissa ohjelmointikieliä on yhteensä 5:

- Structured Text (ST)
- Instruction List (IL)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Sequential Function Chart (SFC)

Näistä Structured Text ja Instruction List ovat tekstipohjaisia ohjelmointikieliä, kun taas Ladder Diagram, Function Block Diagram ja Sequential Function Chart ovat graafisia [9].

3.2.1 Structured Text

Structured Text muistuttaa logiikkaohjaimien viidestä ohjelmointikielestä kaikkein eniten korkean tason ohjelmointikieltä, kuten Pascalia ja C:tä (Kuva 3). Samoja ohjausrakenteita ovat mm. IF…THEN, CASE, sekä se, että silmukat ja rivit päätetään puolipisteeseen. ST soveltuu erityisen hyvin monimutkaiseen ohjelmointiin, joka sisältää trigonometriaa, differentiaali- ja integraalilaskentaa sekä data-analyysiä, ja mahdollistaa tiiviin, selkeästi jäsennettävän koodin. Se on myös tekstipohjaisen luonteensa vuoksi usein nopeampi suorittaa kuin graafiset ohjelmointikielet. Haittapuolena ST voi olla vieras huolto- ja kunnossapitohenkilöstölle, minkä vuoksi sitä harvoin käytetään tuotantosolun suorassa sekvenssiohjauksessa vaan ST rajataan taustarutiineihin, johon huoltotoimenpiteet eivät yleensä kohdistu [9]. Tämmöisiä rutiineja ovat mm. kommunikaatiolohkot eri teollisuuslaitteiden kanssa.

```
1 FOR #tiCounter := 1 TO 36 DO
2     IF "DB_FG01_OEM".arQDataLowerHousing[#tiCounter] = '$00'
3     THEN "DB_FG01_OEM".arQDataLowerHousing[#tiCounter] := ' '
4         ;
5     END_IF;
6 END_FOR;
```

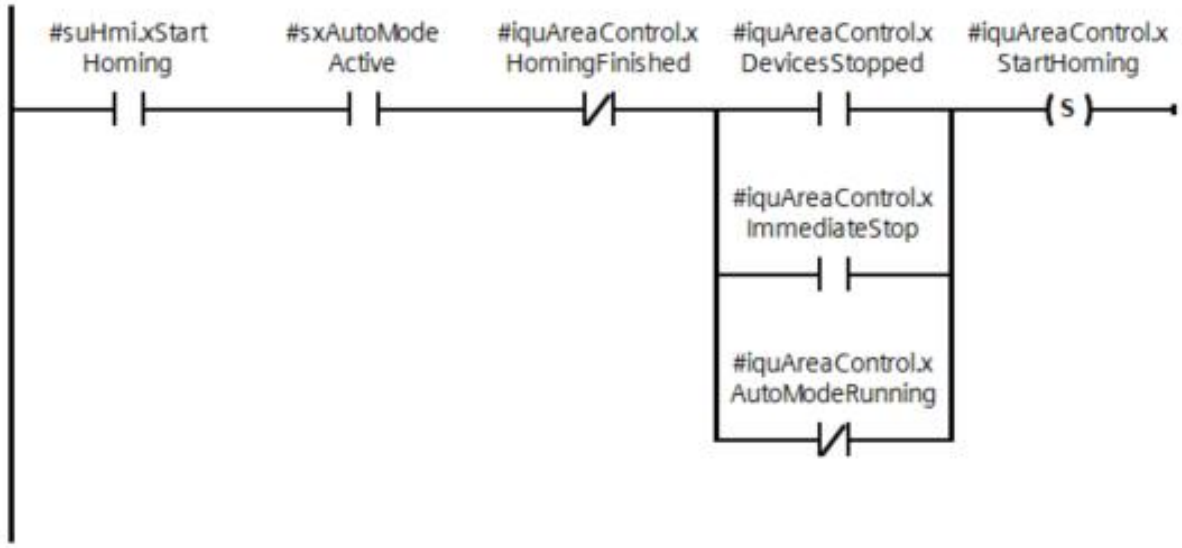
Kuva 3. Esimerkki Structured Text-ohjelmointikielestä, jossa tarkistetaan FOR-silmukalla, onko taulukossa (array) NULL-arvoja ja korvataan ne välilyönneillä.

3.2.2 Instruction List

Instruction List on matalan tason tekstipohjainen ohjelmointikieli, jossa yksi koodirivi vastaa yhtä operaatiota. Se muistuttaa enemmän assembly-ohjelmointikieltä kuin muita moderneja ohjelmointikieliä [1]. IL on tyypillisesti suoritusajaltaan nopea ja muistitehokas, mutta se on vähemmän visuaalinen kuin muut kielet, mikä voi vaikeuttaa ohjelman lukemista, mikäli lukijalla ei ole ohjelmointitaustaa. Modernit PLC:t ovat suorituskyvyltään ja nopeudeltaan niin tehokkaita, että IL:n hyödyt jäävät vähemmän merkittäviksi haittoihin verrattuna [9]. IL luokiteltiin käytöstä poistetuksi IEC 61131-3:n 3. painoksessa vuonna 2013 ja poistettiin kokonaan 4. painoksesta vuonna 2025 [10] [7].

3.2.3 Ladder

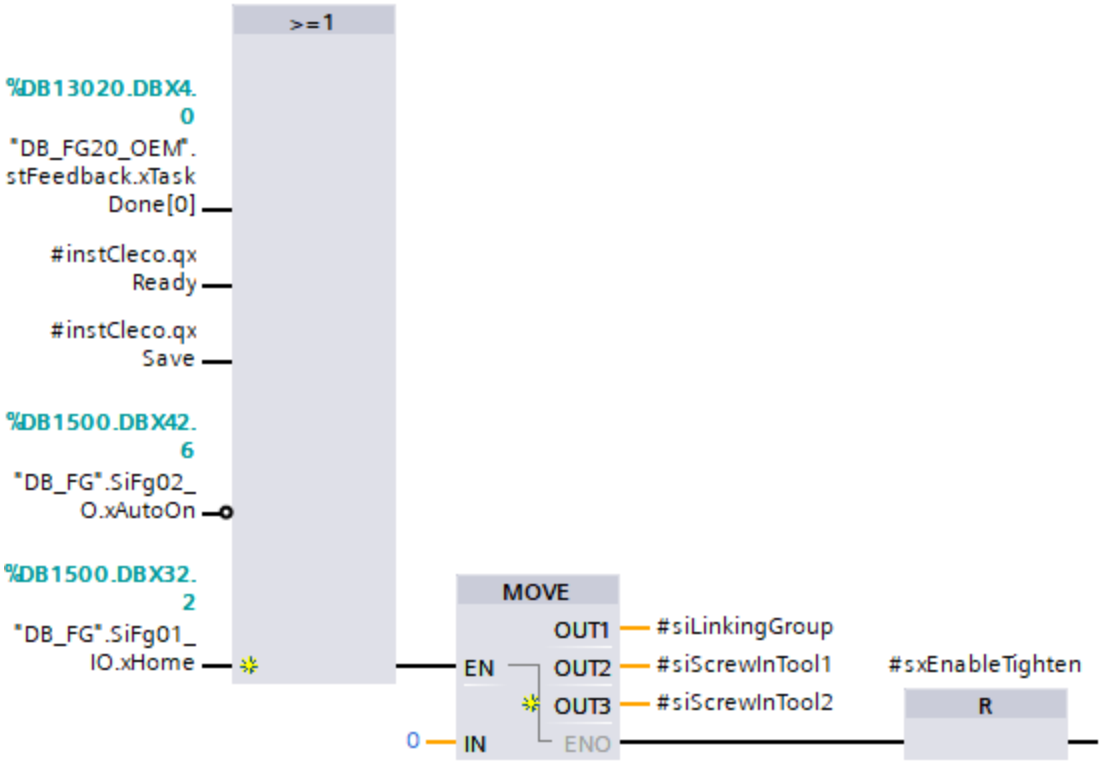
Kaikista IEC 61131-3 -standardin mukaisista PLC-ohjelmointikielistä Ladder (suomeksi välillä myös Tikapuukaavio) on käytetyin. Se on graafinen ohjelmointikieli, jonka suosio perustuu siihen, että sitä voi ymmärtää, vaikkei olisikaan tottunut perinteiseen tekstipohjaiseen ohjelmointiin [9]. Ladder perustuu vanhaan relelogiikkaan [1]. Ladderissa koodi koostuu mm. kontakteista, keloista, ajastimista ja funktioista. Kuvan 4 esimerkissä näkyy yksittäinen verkko, joka toteuttaa kotiajon manuaalisen käynnistyslogiikan. Verkossa hyödynnetään sarjaan kytkettyjä sulkeutuvia ja avautuvia koskettimia sekä rinnakkaista haaroitusta. Lopussa muuttuja asetetaan päälle (set-funktio).



Kuva 4. Esimerkki kotiinajosekvenssin kutsumisesta Ladder-ohjelmointikielellä.

3.2.4 Function Block Diagram

Function Block Diagram (FBD, suomeksi myös Toimintalohkokaavio) on PLC-ohjelmoinnin toinen graafinen ohjelmointikieli. Toisin kuin Ladder, jossa ohjelmakoodi esitetään relelogiikalla, FBD esitetään valmiiksi määrätyillä lohkoilla, kuten JA, TAI ja ajastimet. Se on erityisen suosittu prosessiohjauksessa [3]. FBD ja Ladder ovat loogiselta perusrakenteeltaan lähes identtisiä, sillä molemmat kuvaavat signaalivirtaa ja Boolen logiikkaa visuaalisessa muodossa. Tästä syystä esimerkiksi Siemensin kehitysympäristössä ohjelmoinnin konversio toimii saumattomasti näiden kahden kielen välillä. Käytännössä koko logiikka on mahdollista kääntää kielestä toiseen ilman, että sisäinen suorituslogiikka tai datan käsittely muuttuu. Kuvassa 5 Ladder-kielellä ohjelmoitu verkko käännettynä FBD:ksi.



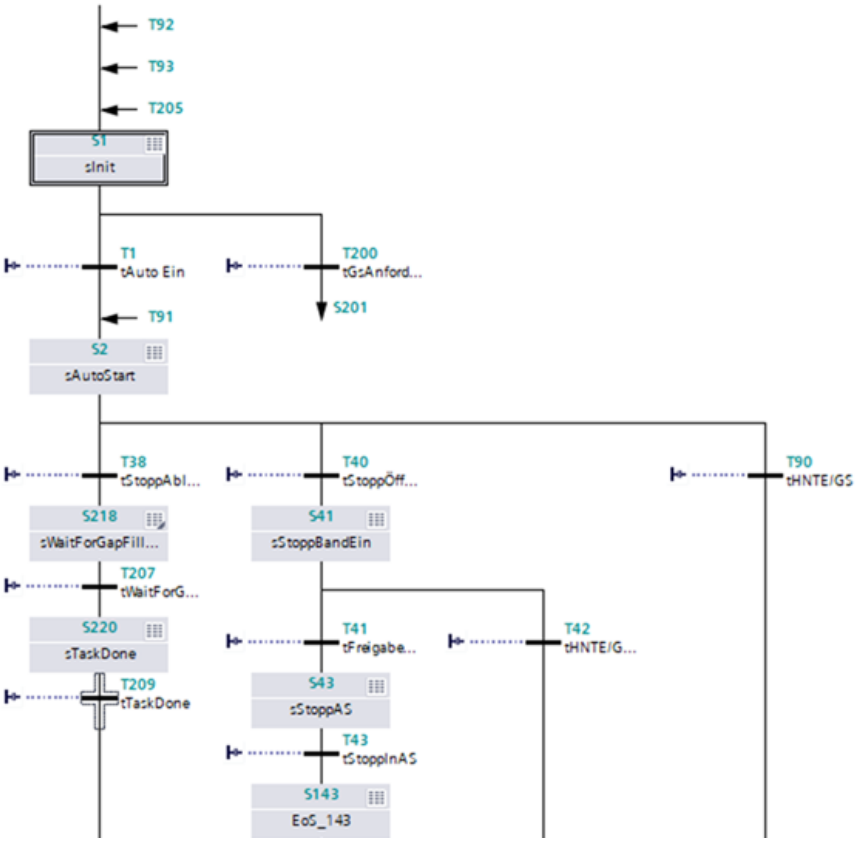
Kuva 5. Ladder-ohjelmointikielellä ohjelmoitu verkko käännetty Function Block Diagrammiksi.

3.2.5 Sequential Function Chart

Sequential Function Chart (SFC) on kolmas graafinen ohjelmointikieli, ja se muistuttaa vuokaaviota [4]. Sitä käytetään monivaiheisten sekvenssien hallintaan. SFC perustuu Grafcet-standardiin ja se on teoreettisesti johdettu Petri-verkoista, joita käytetään hajautettujen järjestelmien ohjaamiseen [11].

SFC koostuu vaiheista (Steps) ja siirtymistä (Transitions). Vaiheet edustavat järjestelmän tiloja, joihin on kytketty tiettyjä toimintoja (Actions). Nämä toiminnot voivat olla yksinkertaisia bittiohjauksia tai monimutkaisempia alirutiineja, jotka on kirjoitettu muilla IEC-kielillä. Siirtymät sisältävät ehtoja, joiden on toteuduttava, jotta ohjauslogiikka siirtyy seuraavaan vaiheeseen [11].

Käytännön työssä on huomattu, että SFC:n keskeinen etu on se, että sillä voidaan visualisoida sekvenssin järjestystä ja hallita rinnakkaisia kokonaisuuksia. Se mahdollistaa monimutkaisten prosessien purkamisen hallittaviin osiin. SFC on korvaamaton työkalu kunnossapitohenkilöstölle vianetsinnässä, sillä ohjelmointityökalu korostaa aktiivisen vaiheen reaaliajassa. Tällöin on helppo havaita, mihin ehtoon sekvenssi on pysähtynyt. Kuvassa 6 näkyy katkelma erään automaattisolun sekvenssistä.



Kuva 6. Osio sekvenssikaaviosta Sequential Function Chartilla

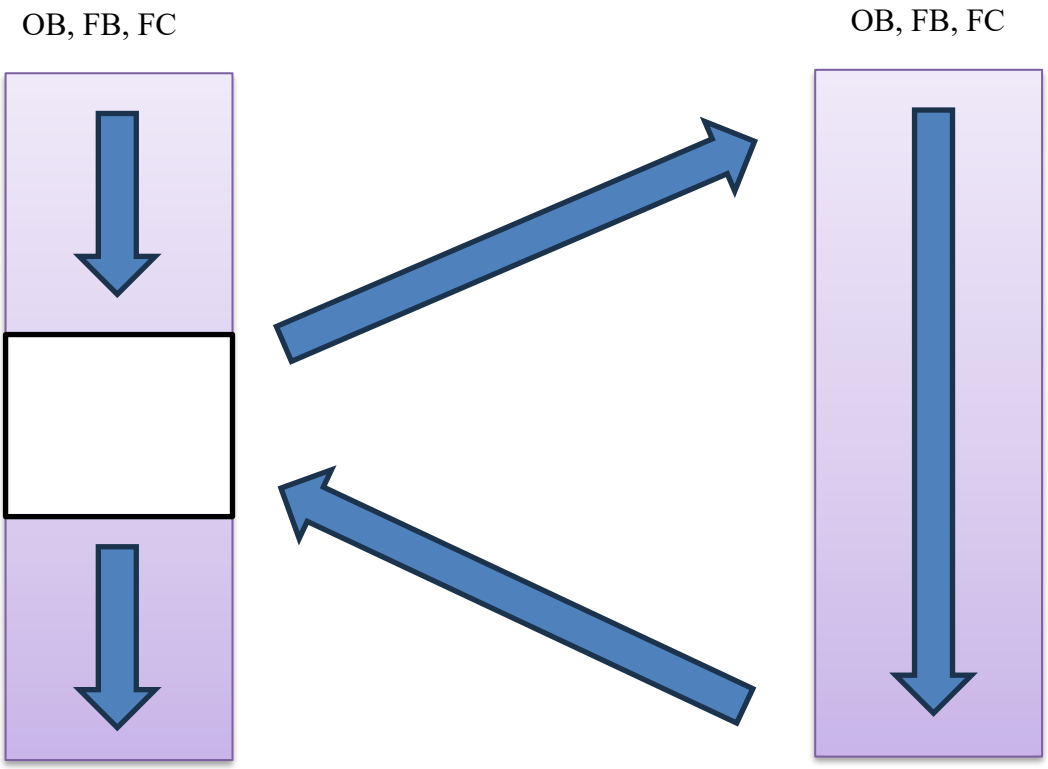
4 Ohjelmiston modulaarinen rakenne ja globaalit kirjastot

Nykyaikaista automaatio-ohjelmointia helpottaa ohjelmistokoodin modulaarinen arkkitehtuuri. Se parantaa koodin käytettävyyttä, testattavuutta ja ylläpidettävyyttä. Modulaarisuuden tarkoituksena on jakaa monimutkainen ohjausjärjestelmä pienempiin osakokonaisuuksiin, mikä vähentää virhealttiutta, nopeuttaa uusien järjestelmien suunnittelua ja helpottaa vianhakua [1]. Tässä luvussa tarkastellaan ohjelmiston rakennekomponentteja ja niiden hallintaa, sekä tapoja hyödyntää modulaarisuutta ohjelmointikoodissa.

4.1 Ohjelmointiyksiköt

PLC:n ohjelmointiyksiköitä kutsutaan POU:ksi (*Program Organization Unit).* POU:ta on kolmea eri tyyppiä, joilla on omat roolinsa ohjelman suorituksessa [12]. Ohjelman suoritusta on esitetty tarkemmin kuvassa 7 lohkojen kutsun aikana.

- **Organisaatioblokki (OB).** Organisaatioblokit eivät ole varsinaisesti lohkoja muiden joukossa, vaan ne toimivat käyttöjärjestelmän ja käyttöohjelman rajapintana. RUN-tilassa käyttöjärjestelmä suorittaa ohjelmakierto-OB:n eli OB1 jokaisen syklin jälkeen uudestaan. Ohjelmakierto-OB toimii alimmalla prioriteetilla, muut tapahtumaohjatut OB-lohkot, kuten häiriöiden hallintaan tarkoitetut diagnostiikkakeskeytykset, voivat keskeyttää suorituksen tarvittaessa.
- **Funktioblokki (FB).** Funktioblokki käyttää datablokkeja tai instanssidatablokkeja staattisten muuttujien tallentamiseen. Se tarvitsee aina kutsun jostain toisesta lohkosta. Funktioblokin suurin hyöty modulaarisuuden näkökulmasta on se, että sillä voidaan luoda useita instansseja. Tällöin yksi koodilohko mahdollistaa usean samanlaisen laitteen ohjauksen, esimerkiksi yhdellä sylinterilohkolla voidaan ohjata useampaa samanlaista sylinteriä. Tällöin jokaisen sylinterin tila tallennetaan omaan instanssidatablokkiin, jossa tilatiedot säilyvät suorituskertojen välillä.
- **Funktio (FC).** Funktioita käytetään operaatioiden suorittamiseen, kuten matemaattisiin ja loogisiin operaatioihin. Toisin kuin funktioblokilla, funktioilla ei ole omaa muistia, vaan se käyttää väliaikaismuuttujia, joiden arvot häviävät suorituksen jälkeen. Jotta dataa saadaan talteen, se täytyy tallentaa globaaliin muistiin. Myös funktiota täytyy kutsua erikseen.



Kuva 7. Ohjelmalohkojen välinen vuorovaikutus ja suorituksen siirtyminen. Ohjelman suoritus siirtyy kutsuvasta lohkosta (vasemmalla) kutsuttuun lohkoon (oikealla). Kutsutun lohkon suorituksen jälkeen hallinta palautuu takaisin alkuperäiseen lohkoon [12].

4.2 Instanssit ja muistinhallinta

Ohjelmointiympäristössä koodi ja muistinhallinta pidetään tyypillisesti erillään. Luvussa 4.1 mainittujen funktioblokkien instanssien dataa voidaan Siemensin ohjelmointiympäristössä tallentaa kolmella eri muodolla: yksittäisinstanssina, multi-instanssina ja parametri-instanssina. Dataa voidaan tallentaa myös pysyvään tai väliaikaiseen muistiin.

4.2.1 Instanssit

Yksittäisinstanssissa funktioblokille luodaan aina oma instanssidatablokki [13], lyhenteenä käytetään iDB tai DI. Esimerkiksi jos funktioblokilla ohjataan useampaa samanlaista

sylinteriä, jokaiselle sylinterille luodaan oma instanssidatablokki. Jos ohjattavia sylintereitä on paljon, projektipuusta voi tulla turhan pitkä ja sekava. Se sopiikin paremmin tapauksiin, joissa instansseja ei ole montaa ja ohjelma on yksinkertainen.

Multi-instanssissa kaikkien funktioblokkien data tallennetaan kutsuvan funktioblokin instanssidatablokkiin. Tämä vähentää erillisten datablokkien määrää ja tekee ohjelmasta selkeämpää ja siistimmän näköistä [13].

Parametri-instansseissa lohkon instanssia ei määritetä kiinteästi kutsun yhteydessä. Sen sijaan instanssi välitetään lohkolle rajapinnan kautta InOut-parametrinä. [13]. Tämä mahdollistaa sen, että ohjelmoija voi itse määrittää, mikä lohko omistaa instanssin datan. Parametri-instanssien käyttö vähentää datan siirtelyä ja kopioimista datablokeista, kun funktioblokki voi hyödyntää usean eri datablokin sisältöä.

4.2.2 Datatyypit

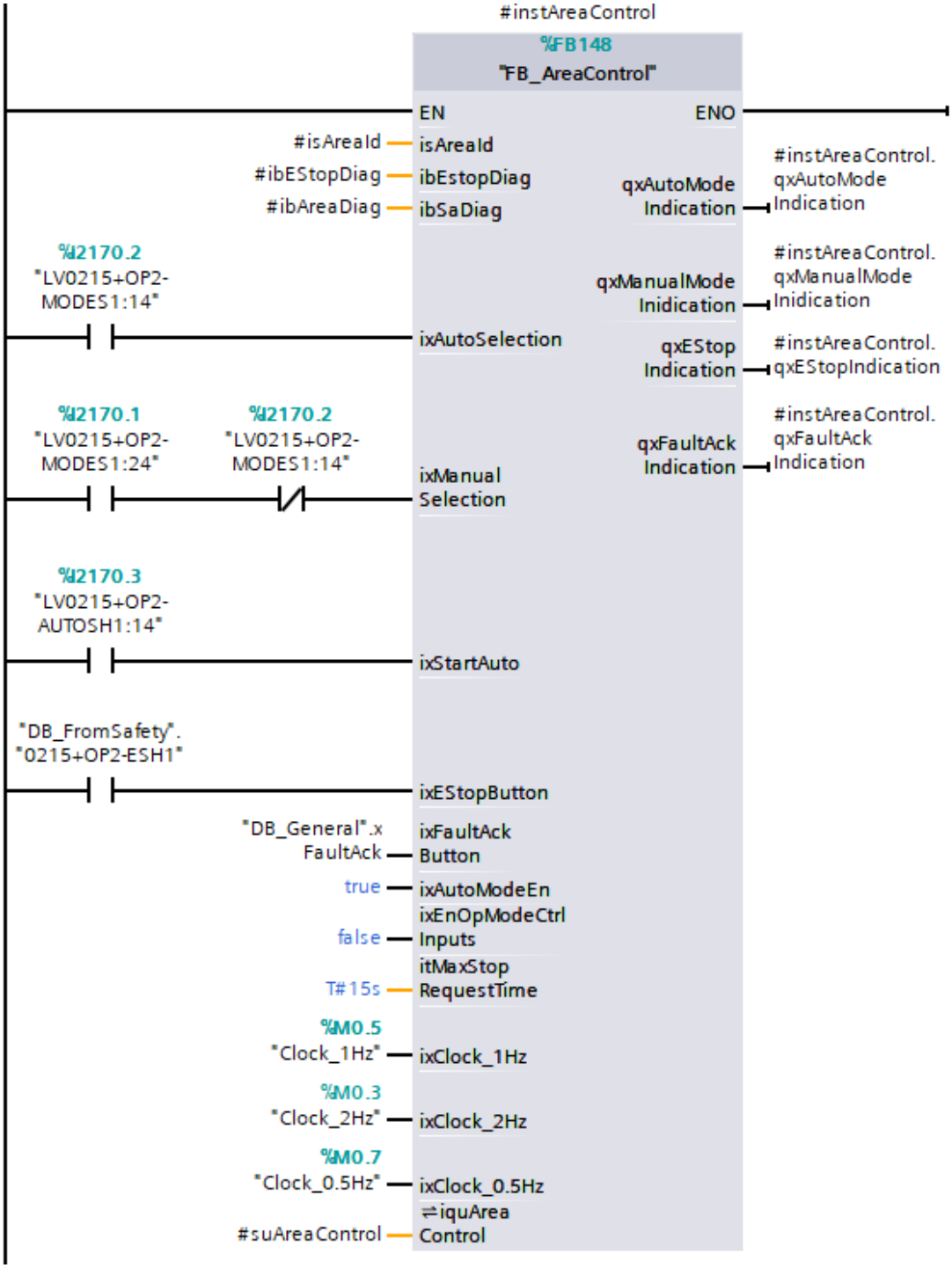
Siemensin ohjelmointiympäristössä on käytössä standardin määrittämät perusdatatyypit. Standardiin kuuluvat muun muassa bitit ja bittijonot (BOOL, BYTE, WORD), kokonaisluvut (INT, DINT, UINT), liukuluvut (REAL, LREAL) sekä aika- ja merkkijonotyypit (TIME, STRING, CHAR).
Perusdatatyyppien lisäksi käyttäjän on mahdollista luoda itsemääriteltyjä datatyyppejä, nk. User Defined Data Type (UDT). Käyttäjän määrittelemään datatyyppiin voidaan ryhmitellä useita erityyppisiä muuttujia yhden otsikon alle. Tällöin esimerkiksi toimilaitteen data pidetään yhtenä kokonaisuutena irrallisten muuttujien sijaan. UDT on myös mahdollista kirjastoida, mikä mahdollistaa sen siirrettävyyden projektista toiseen. Jos käyttäjän määrittelemään datatyyppiin tarvitsee tulevaisuudessa tehdä muutoksia, kuten lisätä uusi muuttuja, muutos tehdään kerran UDT-määritelmään. Määritelmästä muutos päivittyy automaattisesti kaikkiin ohjelman osiin.
Toinen monimutkaisempi datatyyppirakenne on Struct. Myös siinä yhden otsikon alle on mahdollista koota erityyppisiä muuttujia. Toisin kuin UDT, Struct ei ole modulaarinen ja siirrettävä rakenteeltaan. Jos Structiin täytyy tehdä muutos, se ei päivity muihin samanlaisiin tyyppeihin automaattisesti. Tällöin muutos täytyy manuaalisesti tehdä kaikkiin niihin Structeihin, jotka halutaan päivittää.

4.3 Globaalit kirjastot modulaarisuuden hallinnassa

Globaalit kirjastot ovat erinomainen tapa säilyttää modulaarisia ohjelmistokomponentteja, kuten funktioblokkeja, funktioita ja käyttäjän määrittämiä datatyyppejä. Näitä komponentteja voidaan kopioida suoraan projektista toiseen, mikä nopeuttaa uuden ohjelmakoodin käyttöönottoa.

Globaaleihin kirjastoihin tallennettaviksi kirjastolohkoiksi sopivat esimerkiksi toimilaitteiden ohjaus- ja kommunikointilohkot sekä automaattisolujen ohjauslohkot, sillä ne toimivat aina suurin piirtein samalla tavalla. Kirjastolohko toteutetaan lokaaleilla muuttujilla, jotta lohko olisi siirrettävissä. Globaalit muuttujat sekä tulo- ja lähtöosoitteet voidaan tuoda rajapintaan, josta lohko kirjoittaa niille vastaavat lokaalit muuttujat.

Kuvassa 8 näkyy esimerkki kirjastoidusta lohkosta, jolla ohjataan automaattisolua. Automaattisoluja ja ohjauslogiikoita on tuotantolinjoilla tyypillisesti useita, joten niiden ohjaus on järkevää kirjastoida, jotta ohjauslohko on helppo kopioida projektista toiseen. Instanssin rajapinnassa näkyvät tulot osoitteineen ja lohkon kirjoittamat lähtömuuttujat. Aluetta on tarkoitus ohjata fyysisillä painikkeilla, joten jokaisen alueen instanssin rajapintaan on tuotava sen alueen kyseiset tulot painikkeilta.



Kuva 8. Tulot ja lähdöt tuodaan kirjastoidun instanssin rajapintaan, jossa ne määrätään lokaaleihin muuttujiin.

5 Versionhallinta

Versionhallinta on ohjelmistotekninen menetelmä, jonka avulla hallitaan tuotteen elinkaaren aikana syntyviä ohjelmistoversioita ja niiden välisiä muutoksia. Sen tavoitteena on tarjota menetelmä muutosten seuraamiseen ja virheiden peruuttamiseen. Versionhallinnan rooli on elintärkeä erityisesti monen käyttäjän ohjelmointiympäristössä, jossa rinnakkainen muutosten hallinta ja koodin eheyden varmistaminen ovat kriittisiä osa-alueita operatiivisen toiminnan jatkuvuuden kannalta [14].

Tilaajayrityksessä päivitettiin versionhallintajärjestelmä verkkolevypohjaisesta tallennuksesta Siemensin Teamcenter -järjestelmään. Seuraavissa alaluvuissa tarkastellaan näiden kahden menetelmän ominaispiirteitä ja analysoidaan järjestelmämuutoksen hyötyjä.

5.1 Perinteinen verkkolevypohjainen hallinta ja sen haasteet

Tilaajayrityksessä on perinteisesti hyödynnetty jaettua verkkolevyä ohjelmointiprojektien arkistointiin. Vaikka menetelmä on teknisesti yksinkertainen, se on osoittautunut nykyisessä olomuodossaan kankeaksi ja riittämättömäksi. Keskeiset haasteet on saatu tilaajayrityksen käytännön kokemuksen pohjalta ja ne voidaan jakaa seuraaviin kategorioihin:

- Heikko jäljitettävyys. Verkkolevypohjaisessa tallennuksessa muutoshistoria on usein puutteellinen. Uuden version tallentaminen ei luo merkintää siitä, mitä koodissa on muutettu ja kuka muutoksen on tehnyt. Ainut jäljitettävä ominaisuus on edellinen muutosaika.

- Samanaikaisen muokkauksen konfliktit. Useampi ohjelmoija voi avata ja muokata samaa projektitiedostoa samanaikaisesti. Tämä voi johtaa pahimmassa tapauksessa ylikirjoitustilanteisiin ja tuotannon pysähtymiseen.

- Hankaluudet koodin arkistoinnissa. Myös huoltohenkilöstöllä on pääsy ohjelmistoihin huoltotoimenpiteiden takia ja ilman riittävää perehdytystä tiedostoja voi jäädä lukituiksi käyttöoikeusvirheiden takia. Arkistointi on myös vaivalloista ja saattaa unohtua helposti.

- Kommentointiin ja dokumentointiin täytyy käyttää muita työkaluja.

5.2 Teamcenter

Siemensin Teamcenter tarjoaa integroidun versionhallintamahdollisuuden TIA Portal Teamcenter Gateway -rajapinnan kautta. Teamcenter-ratkaisun ytimessä on kyky hallita projekteja Teamcenterin nimikkeinä (items) ja niiden revisioina. Tämä mahdollistaa paremman projektien hallinnan, jossa jokainen ohjelmistoversio jää muutoshistoriaan. Ohjelmistoversioita on myös mahdollista kommentoida, mikä auttaa seuraavaa kehittäjää seuraamaan, mitä muutoksia kussakin revisiossa on tehty.

Kun projekti avataan Teamcenteristä, järjestelmä lataa sen paikalliseen välimuistiin ja lukitsee projektin (Check-out). Tämä mekanismi poistaa aiemmassa luvussa 5.1 mainitun samanaikaisen muokkauksen riskin, sillä järjestelmä estää muita käyttäjiä avaamasta projektia muokkaustilassa ennen lukituksen vapauttamista. Tämä ei kuitenkaan estä muita käyttäjiä avaamasta projektia lukutilassa.

Ennen Teamcenterin käyttöönottoa suoritettiin ohjelmiston versiopäivitys yhteensopivuuden ja uusimpien integraatio-ominaisuuksien varmistamiseksi. Tämä tehtiin lataamalla uuden version projektin ohjelmisto ja laiteohjelmisto PLC:lle, jonka jälkeen se tallennettiin uutena nimikkeenä Teamcenteriin.

5.2.1 Keskeiset hallintakomennot

Teamcenterin lataus tuo TIA Portalin projektipuuhun uuden Teamcenter-valikon, joka sisältää kriittiset toiminnot projektin hallintaan [15].

- Save as new item: Tätä toimintoa käytetään, kun projekti tuodaan ensimmäistä kertaa Teamcenteriin. Käyttäjä voi valita nimiketyypin (item type), määrittää tunnuksen (item ID) ja valitsee kansion, johon projekti tallennetaan.

- Save as new revision: Kun olemassa olevaan projektiin on tehty jotain muutoksia ja se halutaan julkaista takaisin muiden muokattavaksi, siitä tallennetaan uusi revisio. Vanhat versiot säilyvät edelleen tallessa Teamcenterissä, jossa niihin on helppo palata tarvittaessa. Vanhoista versioista näkyy myös kehittäjän laatima muutoslista.

- Check-out: Kun projekti halutaan varata muokattavaksi, valitaan Check-out. Check-out-tilassa olevaa projektia muut eivät voi avata muokkaustilassa, vaan ainoastaan

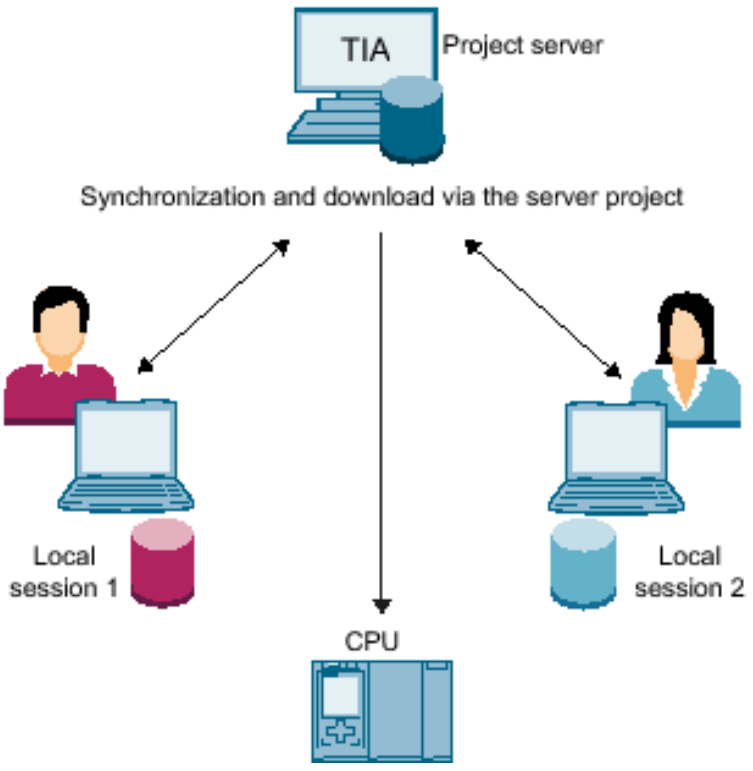
lukutilassa. Teamcenterissä on helposti nähtävillä, kenellä muokkausoikeudet ovat käytössä.

- Check-in: Muutosten tallennuksen ja uuden revision luomisen jälkeen projekti voidaan laittaa Check-in-tilaan. Tällöin muut ohjelmoijat pääsevät taas muokkaamaan projektia.

5.2.2 Rinnakkaissuunnittelu

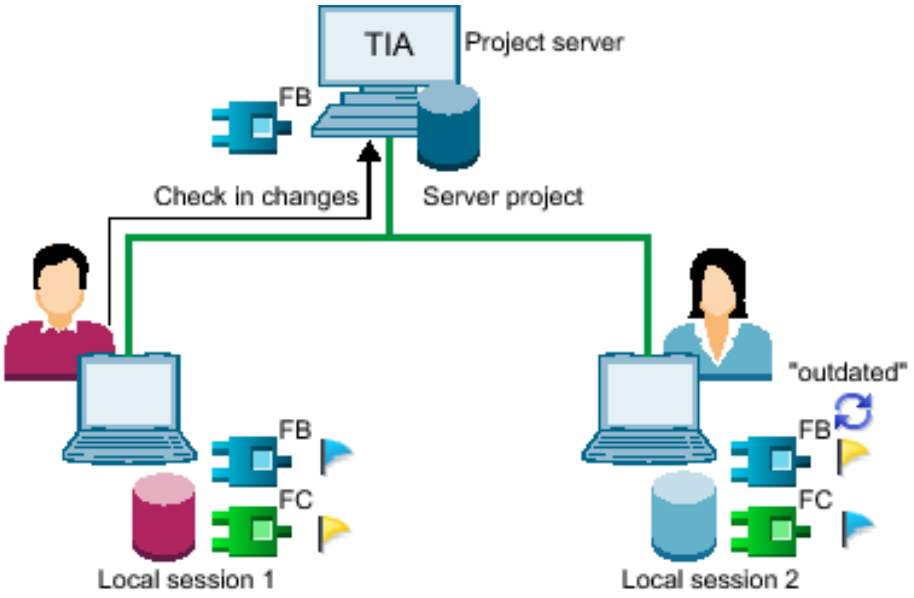
Vaikka projektin lukitus on kätevä työkalu estämään samanaikaiset muokkaukset, usein isommissa projekteissa useampi ohjelmoija työskentelee saman ohjelmistoprojektin parissa. Teamcenter-integraatiota voidaan käyttää yhdessä TIA Portalin Team Engineering –toiminnon kanssa, mikä mahdollistaa rinnakkaissuunnittelun [15].

Rinnakkaissuunnittelussa projektia hallitaan erillisellä palvelimella ja ohjelmoijat luovat omia paikallisia istuntoja. Muutokset tallennetaan paikallisesti ja synkronoidaan palvelimen ”Master-projektiin” (Kuva 9). Kun kaikkien ohjelmoijien työ on valmis, koko Master-projekti tallennetaan Teamcenteriin uutena revisiona [16].



Kuva 9. Projektipalvelimen periaate [16].

Master-projektin ja palvelimen käyttö mahdollistavat tiedon eheyden varmistamisen. Palvelin tarkistaa automaattisesti paikallisten istuntojen synkronoinnin yhteydessä, onko samoihin objekteihin kohdistunut rinnakkaisia muutoksia. Jos konflikteja löytyy, konfliktikohdat avautuvat editorissa ja ne on ratkaistava ennen projektin synkronointia. Projektissa eri ohjelmointiosioiden tilat visualisoidaan reaaliaikaisesti lippumerkintöjen avulla, mistä näkee mitä osioita muut ohjelmoijat muokkaavat (Kuva 10) [16].



Kuva 10. Havainnollistava kuva lippujärjestelmästä. Check-in jälkeen toinen muokkaaja näkee ”outdated” merkistä, että kyseiseen toimintalohkoon on tehty muutoksia [16].

6 Dokumentointi

Tilaajayrityksessä ensisijainen dokumentointitapa oli tehdä koodista selkeää ja luettavaa. Tämä sisältää ohjelmaverkkojen, muuttujien sekä tulojen ja lähtöjen täsmällisen nimeämisen. Muuttujilla käytetään etuliitteitä, josta näkee nopeasti muuttujan datatyypin ja koon. Etuliitteiden nimeämiskäytännöt löytyvät myös standardista ja ne on esitelty taulukossa 2. Näiden lisäksi yrityksillä voi olla muille datatyypeille omia etuliitteitä.

Taulukko 2. Datatyyppien etuliitteet [7].

Ensimmäinen etuliite	Toinen etuliite	Datatyyppi
I		Tulo (input)
Q		Lähtö (output)
M		Muistipaikka
	X	Bitti (BOOL): 1 bitti
	B	Tavu (BYTE): 8 bittiä
	W	Sana (WORD): 16 bittiä
	D	Double: 32 bittiä
	L	Long: 64 bittiä

Esimerkiksi merkintä %IX200.6 tarkoittaisi tulobittiä osoitteesta 200.6. Merkintä %MW8 tarkoittaa muistitavua osoitteessa 8. Ensimmäinen etuliite viittaa minkä tyyppinen tallennettu arvo on: tulo, lähtö vai tallennettu muistipaikka. Näiden lisäksi arvot voivat olla lokaaleja muuttujia, väliaikaisia muuttujia (temp), InOut-muuttujia ja niin edelleen.

Toinen etuliite viittaa datatyypin kokoon. Nämä merkitään X, B, W, D, L, jotka tarkoittavat vastaavasti 1, 8, 16, 32 ja 64 bittiä.

Muuttujien nimeämisen lisäksi ohjelmointiympäristö mahdollistaa kommenttikenttien luomisen ohjelmaverkkojen alkuun. Kenttään on hyvä kirjoittaa lyhyt kuvaus ohjelmaverkon toiminnasta. Ohjelmointiympäristö mahdollistaa myös nuolilla varustettujen lisäkommenttikenttien luomisen keskelle ohjelmaverkkoja. SCL-kielellä kirjoitettuna perinteisten kommenttien luonti tekstipohjaisen koodin sekaan onnistuu myös kahdella kauttaviivalla.

7 Yhteenveto

Työn tavoitteena oli selvittää, miten standardit, globaalit kirjastot ja hallintatyökalut, kuten Teamcenter, voivat parantaa automaatio-ohjelmistojen laatua, ylläpidettävyyttä ja suunnitteluprosessin tehokkuutta.

Tutkielmassa havaittiin, että kansainvälinen standardisointi on ollut merkittävässä asemassa siirtymässä vanhasta relelogiikasta kohti modernien, instanssipohjaisten ja modulaaristen rakenteiden syntyä. IEC 61131 -standardi auttaa kehittäjää käsittelemään koodin hierarkiaa ja muistinhallintaa. Keskeiset tekijät ohjelmoinnin modulaarisointiin ovat instanssipohjaiset funktioblokit, uudelleenkäytettävät funktiot sekä käyttäjän määrittämät datatyypit. Näiden kirjastopohjaisuus mahdollistaa muutosten päivittymisen läpi projektin muuttamalla pelkkää keskitettyä määritelmää.

Standardi antaa myös selkeät ohjeet koodin luettavuudelle nimeämiskäytäntöjen kautta. Muuttujien nimeäminen standardin mukaisten etuliitteiden avulla varmistaa, että koodi on paremmin tulkittavissa käyttäjille. Standardin lisäksi yrityksen on hyvä jatkaa nimeämiskäytäntöjä muillekin kuin standardin nimeämille kohteille, kuten staattisille muuttujille. Muuttujien nimien on myös tärkeää olla kuvaavia, jotta koodi on luettavampaa muille.

Teamcenteriin siirtyminen tarjosi tilaajayritykselle selkeän edun vanhaan versioon verrattuna ja käyttö ei aiheuttanut haasteita useamman ohjelmoijan ympäristössä. Jäljitettävyys ja muutosten dokumentointi paranivat, eikä rinnakkaismuokkauksien aiheuttamia konflikteja ja ylikirjoituksia enää tapahtunut.

Jatkossa olisi mahdollista ottaa käyttöön Teamcenterin rinnakkaissuunnitteluominaisuus, jossa samaa projektia voisi reaaliajassa kehittää useampi ohjelmoija. Konfliktienhallinta hoituu Master-projektissa automaattisesti, joten rinnakkaissuunnittelu olisi selkeä parannus tilanteeseen, jossa useamman kehittäjän olisi tarpeellista muokata samaan aikaan samaa projektia.

8 Viittaukset

[1] D. H. Hanssen, Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS, John Wiley & Sons, 2015.

[2] I. E. Commission, Programmable Controllers - Part 1: General Information (IEC 61131-1:2003), 2nd ed., Geneva: International Electrotechnical Commission (IEC), 2003.

[3] A. Ingale, S. Patil, S. Jambhale ja S. Patil, ”Transforming Industrial Control: The Evolution and Future Trends of PLC Systems,” *International Journal of Creative Research Thoughts (IJCRT),* osa/vuosik. 12, nro 5, pp. k236-245, 2024.

[4] H. Jack, Automating Manufacturing Systems, 2008.

[5] M. Koondhar, G. Kaloi, A. Junejo, A. Hameed, S. Chandio ja M. Ali, ”The Role of PLC in Automation, Industry and Education Purpose: A Review,” *Pakistan Journal of Engineering, Technology & Science,* osa/vuosik. 11, nro 1, pp. 22-31, 2023.

[6] K.-H. John ja M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids, Springer-Verlag Berlin Heidelberg, 2010.

[7] I. E. Commission, Programmable controllers—Part 3: Programming languages (IEC 61131-3:2025), 4th ed., Geneva, Switzerland: International Electrotechnical Commission (IEC), 2025.

[8] PLCopen, ”Logic,” [Online]. Available: https://plcopen.org/technical-activities/logic. [Haettu 7 August 2025].

[9] T. Thayer, ”Speaking in Tongues: Understanding the IEC 61131-3 Programming Languages,” 30 January 2009. [Online]. Available: https://www.controleng.com/speaking-in-tongues-understanding-the-iec-61131-3-programming-languages/. [Haettu 8 September 2025].

[10] I. E. Comission, Programmable controllers—Part 3: Programming languages (IEC 61131-3:2013), 3rd ed., Geneva, Switzerland: International Electrotechnical Commission (IEC), 2013.

[11] ABB, ”Overview of the IEC 61131 Standard,” ABB Inc. Totalflow Products.

[12] Siemens, ”"Programming Concepts" SIMATIC S7-1200 Programmable controller,” 16 September 2016. [Online]. Available: https://support.industry.siemens.com/cs/mdm/109741593?c=88590555531&lc=en-FI. [Haettu 24 February 2026].

[13] Siemens Simatic, ”STEP 7 and WinCC Engineering V18 System Manual,” 17 November 2022. [Online]. Available:

https://support.industry.siemens.com/cs/mdm/109815056?c=162901105419&lc=pt-AO. [Haettu 5 March 2026].

[14] B. Vogel-Heuser, A. Fay, I. Schaefer ja M. Tichy, ”Evolution of software in automated production systems: Challenges and research directions,” *The Journal of Systems and Software,* pp. 54-84, 2015.

[15] Siemens, ”"TIA Portal Teamcenter Gateway",” Siemens Industry Online Support, November 2025. [Online]. Available: https://docs.tia.siemens.cloud/r/en-us/v21/tia-portal-teamcenter-gateway. [Haettu 15 February 2026].

[16] Siemens, ”"Using Team Engineering",” Siemens Industry Online Support, 11 2024. [Online]. Available: https://docs.tia.siemens.cloud/r/en-us/v20/using-team-engineering. [Haettu 15 February 2026].