SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

# SIEMENS PLC-OHJELMOINNIN INTEGROINTI OHJELMISTO-KEHITYKSEEN

TEKIJÄ: Niko Illikainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

| | |
|---|---|
| Koulutusala<br>Tekniikan ja liikenteen ala | |
| Koulutusohjelma<br>Sähkötekniikan koulutusohjelma | |
| Työn tekijä<br>Niko Illikainen | |
| Työn nimi<br>Siemens PLC ohjelmoinnin integrointi ohjelmistokehitykseen | |
| Päiväys 14.5.2019 | Sivumäärä/Liitteet 51+0 |
| Ohjaaja(t)<br>Pasi Lepistö, lehtori ja Jari Ijäs, lehtori | |
| Toimeksiantaja/Yhteistyökumppani<br>YSP Oy, Tatu Mikkonen, projektipäällikkö | |


Tiivistelmä

Insinöörityön aiheena oli Siemens logiikkaohjelmoinnin liittäminen osaksi yrityksen ohjelmistokehitystä. Työn tavoitteena oli perehtyä Siemensin tarjoamiin kirjastomuotoihin, simulaattoreihin ja Openness-rajapintaan, sekä kehittää niistä versionhallintajärjestelmään ja jatkuvaan integraatioprosessiin soveltuvat ratkaisut. Versionhallintaa tullaan käyttämään PLC-lohkojen säilytyspaikkana ja niiden kirjanpitojärjestelmänä. Jatkuva integraatioprosessi hoitaa PLC-lohkojen automaattisen testauksen automaatiopalvelimien simulaattoreilla.

Työ aloitettiin teoriaosalla perehtymällä yleisesti versionhallintajärjestelmiin, jatkuvaan integraatioprosessiin ja Siemens TIA Portal -ohjelmistoon sekä sen eri kirjastomuotoihin. Tämän jälkeen siirryttiin tutkimaan tarkemmin TIA Portalin eri kirjastotyyppejä ja niiden potentiaalia versionhallintajärjestelmän kanssa työskentelyssä. Kun asianmukainen ratkaisu löytyi, siirryttiin kirjaston käytön suunnitteluun. Suunnitellun kokonaisuuden rakentamisen jälkeen liitettiin kirjasto versionhallintajärjestelmään ja testattiin toimivuus. Kirjastojen valmistuttua tutkittiin TIA Portalin logiikkasimulaattoreita ja Opennes-rajapintaa sekä C#-ohjelmointikieltä, jotta voidaan tehdä automaattitestejä edeltävät toimenpiteet C#-sovelluksella Openness-rajapinnan kautta automaatiopalvelimella. Tarpeellisen tiedonhaun jälkeen aloitettiin sovelluksen rakentaminen, joka automaattisesti avaa ja konfiguroi annetun TIA-projektin sekä lataa logiikkaohjelman konfiguroituun simulaattoriin automaatioympäristössä. Lopuksi tuotettiin yrityksen sisäiset TIA Portal -kirjastojen ja simulaattorien käyttöohjeet, sisältäen myös ohjeen muutoksien viemisestä versionhallintaan ja sieltä muutoksien tuomisesta omaan säilytyskansioon.

Työn lopputuloksena saatiin TIA Portal -kirjastot versionhallintaan ja niiden dokumentoinnin seurauksesta syntyneet käyttöohjeet sekä sovellus, joka hyödyntää Openness-rajapintaa ohjelmien latauksissa automaatiopalvelimen logiikkasimulaattoriin. Työn tuotoksia tullaan hyödyntämään tulevaisuudessa kehittäjien luomien PLC-sovelluksien laitteiden ja lohkojen organisoinnissa, versioinnissa sekä automaatiopalvelimilla suoritettavissa automaattitesteissä.



Avainsanat
PLC, TIA Portal, Siemens, ohjelmoitava logiikka, ohjelmistokehitys, versionhallinta, SVN, PLC-kirjasto, Openness

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

| | | | |
|---|---|---|---|
| Field of Study<br>Technology, Communication and Transport | | | |
| Degree Programme<br>Degree Programme in Electrical Engineering | | | |
| Author(s)<br>Niko Illikainen | | | |
| Title of Thesis<br>Integration of Siemens PLC Programming into Software Development | | | |
| Date | 14 May 2019 | Pages/Appendices | 51+0 |
| Supervisor(s)<br>Mr. Pasi Lepistö, Senior Lecturer and Mr. Jari Ijäs, Senior Lecturer | | | |
| Client Organisation /Partners<br>YSP Oy, Mr. Tatu Mikkonen, project manager | | | |

Abstract

The main subject of this thesis was to incorporate Siemens PLC programming to be a part of the company's software development. The goal of this thesis was to study library formats, simulators and Openness API provided by Siemens and to develop suitable solutions for version control systems and continuous integration concerning PLC libraries and actions in automation servers.

The thesis was started by studying the basics of version control, continuous integration and the TIA Portal.
After the basics, the work continued with a deeper investigation of different library possibilities of the TIA Portal and how they would work with version control. When a suitable solution was found, the planning of the library started. After the desired library was created it was transported to version control and its functionality was tested. Upon the completion of the libraries, the work continued by studying the PLC simulators and Openness API of the TIA Portal and also C# programming language in order to program an appropriate application for the automation environment. The application will perform a necessary number of steps via C# software that utilizes Openness API before automatic tests can be ran on the automation environment. After necessary search of information, the programming of the application started. It would automatically do a certain number of actions required to download the program to a simulated PLC instance. At the end of the work, instructions of TIA Portal libraries and simulators for the company's internal use were created, including a guide on how to commit and update changes in and from the version control system.

As a result, libraries in version control were created and by documentating them and simulators, a manual for other developers was produced. Also, software which uses Openness API in order to download a PLC program into an instance in an automation server was created. These results will be exploited in the future in managing and versioning the PLC workpieces of the developers and in automatic tests which take place in automation servers.

Keywords
PLC, Programmable Logic Controller, Siemens, TIA Portal, subversion, SVN, function block library, software development, Openness

SISÄLLYS

LYHENTEET JA KÄSITTEET ..... 6
1 JOHDANTO ..... 7
2 YSP OY ..... 7
2.1 YSP Oy:n historia ..... 8
2.2 Dynniq ..... 9
3 OPINNÄYTETYÖN KUVAUS, TAVOITE JA RAJAUS ..... 10
4 KEHITYSYMPÄRISTÖT JA TYÖKALUT ..... 11
4.1 TIA Portal ..... 11
4.2 Simulaatiotyökalut ..... 13
4.2.1 S7-PLCSIM ..... 14
4.2.2 S7-PLCSIM Advanced ..... 16
4.3 Openness-rajapinta ..... 19
4.4 Visual Studio ..... 20
4.5 Versionhallintajärjestelmä ..... 21
4.5.1 Perustoiminnot ..... 23
4.5.2 Versionhallintajärjestelmien sukupolvet ..... 25
4.5.3 Subversion ..... 28
4.5.4 TortoiseSVN ..... 28
4.6 Automaatiopalvelin ..... 30
4.6.1 Jenkins ..... 30
5 JATKUVA INTEGRAATIO ..... 31
5.1 Historia ..... 32
5.2 Yleisimmät käytänteet ..... 33
6 YSP OY:N OHJELMISTOKEHITYSPROSESSI ..... 35
7 SUUNNITTELU JA TOTEUTUS ..... 36
7.1 Kirjastot ..... 36
7.1.1 Kirjastotyypit ja niiden rakenne TIA Portalissa ..... 36
7.1.2 Rakenne levyllä ..... 37
7.1.3 Kirjastojen suunnittelu ..... 39
7.1.4 Kirjastojen toteutus ja vieminen versionhallintaan ..... 40
7.2 PLC-sovelluksen automaattinen lataaminen simulaattorille ..... 43

7.2.1 Openness-rajapintasovelluksen rakentaminen ..... 43
7.3 Käyttöohjeiden laatiminen ..... 46
8 JATKOTOIMENPITEET ..... 48
9 YHTEENVETO ..... 48
LÄHTEET ..... 49

# LYHENTEET JA KÄSITTEET

API (Application Programming Interface) = Ohjelmointirajapinta. Julkinen rajapinta, jonka kanssa muut ohjelmat voivat vaihtaa tietoja.

Build = Koontiversio, työkappaleista rakennettu kokonaisuus.

C# (C Sharp) = .NET-alustalle kehitetty oliopohjainen ohjelmointikieli.

CI (Continuous Integration) = Jatkuva integraatio. Kehitysprosessissa hyödynnettävä käytäntö, joka auttaa kehittäjien luomien tuotoksien automaattista testausta ja julkaisua.

FB (Function Block) = PLC-ohjelmoinnissa käytettävä funktiolohko.

FC (Function) = PLC-ohjelmoinnissa käytettävä funktio.

LD (Ladder Diagram) = Tikapuudiagrammi, PLC ohjelmointikieli.

PC (Personal Computer) = Tietokone.

PLC (Programmable Logic Controller) = Ohjelmoitava logiikka.

SCADA (Supervisory Control and Data Acquisition) = Valvonta- ja käytönohjausjärjestelmä, PC-valvomo.

STEP 7 = Siemensin kehittämä logiikkaohjelmointisovellus.

SVN (Subversion) = Keskitetyn palvelimen versionhallintajärjestelmä.

TIA Portal (Totally Integrated Automation - Portal) = Siemensin kehittämä automaatiosuunnitteluun ja -ohjelmointiin tarkoitettu työkalu.

UDT (User Defined dataType) = Käyttäjän määrittelemä datatyyppi TIA Portalissa.

VCS (Version Control System) = Versionhallintajärjestelmä.

WinCC = Siemensin SCADA-sovellus, jolla voidaan luoda valvomo- ja käyttöliittymäsovelluksia.

# 1 JOHDANTO

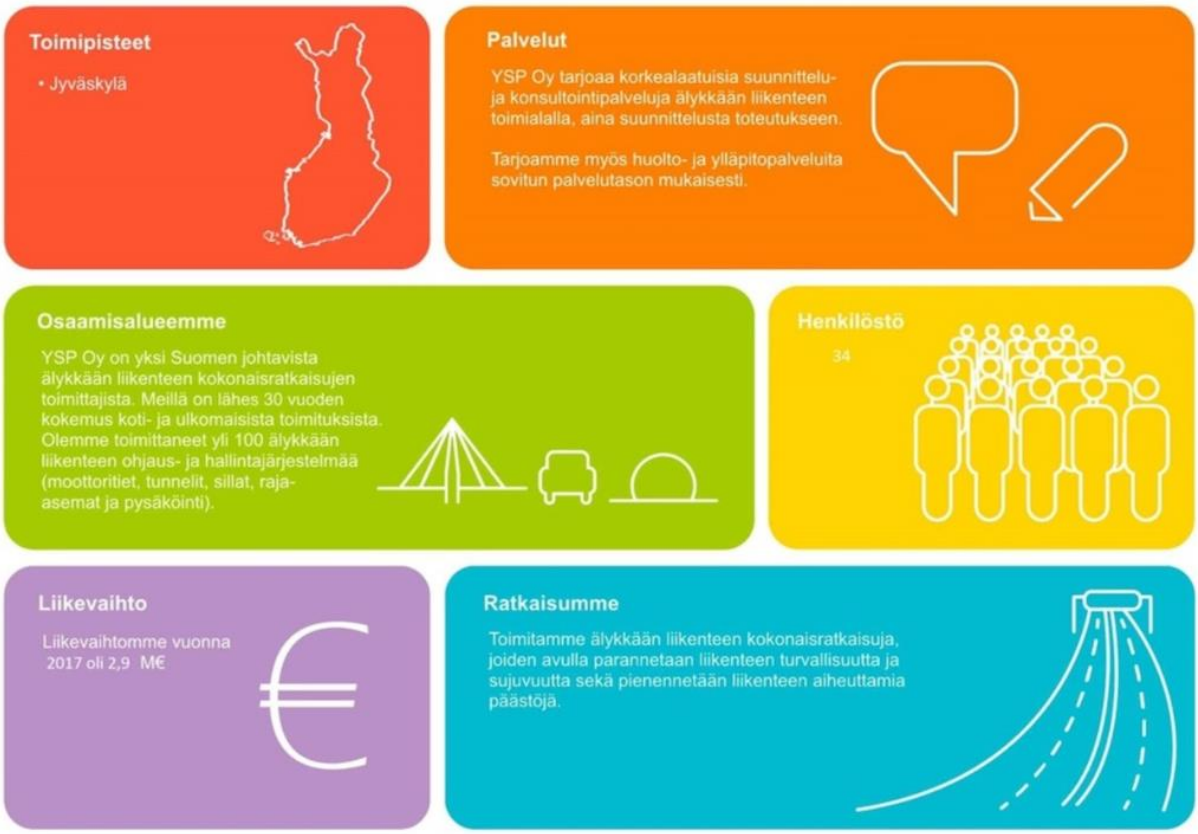
Tämä opinnäytetyö käsittelee jatkuvan integraatioprosessin ja versionhallinnan lisäksi TIA Portalin PLC-kirjastoja, Openness-rajapintaa ja simulaattoreita sekä näiden soveltamista edellä mainittuihin kokonaisuuksiin. Toimeksiantajana toimi jyväskyläläinen insinööritoimisto YSP Oy. Jatkuva integraatio on ohjelmistokehityksessä sovellettava käytäntö, jolla voidaan tehostaa kehittäjien tuottamien työkappaleiden automaattista koostamista, testausta ja julkaisua. Versionhallintaa käytetään kirjanpitojärjestelminä kappaleiden eri versioiden muutoksien jäljitettävyyden ja palautettavuuden takia. Versionhallintajärjestelmää käytetään usein myös jatkuvan integraatioprosessin läpi menevien tiedostojen säilytyspaikkana. Lisäämällä logiikkaohjelmoinnista syntyneet lohkot ja muut työkappaleet tähän prosessiin, tuloksena on edellä mainittujen hyötyjen lisäksi useita positiivisia puolia, kuten kehittäjien välinen selkeys, käytössä olevan sovelluksen jäljitettävyys ja koodien testausten nopeutus. PLC-ohjelmoinnissa käytetään tavallisesti lohkojen organisoimiseen lohko- ja PLC-kirjastoja. Kyseisien kirjastojen vieminen versionhallintaan edesauttaa lohkojen kehittämistä, ja virhetapauksissa edelliset versiot ovat helposti saatavilla. YSP:llä logiikkaohjelmointi tehdään lähtökohtaisesti tekstipohjaisella ohjelmointikielellä sen luettavuuden ja selkeyden kannalta. Tämän lisäksi tekstimuotoisen koodin uudelleenkäytettävyys eri laitetoimittajien ja sovellusten välillä onnistuu helpommin verrattuna esimerkiksi LD-kieleen. Lähdekoodin vieminen erilaisiin järjestelmiin helpottuu myös huomattavasti. Asiantuntemuksen laajentaminen Siemensin TIA Portaliin mahdollistaa osallistumisen hankkeisiin, joissa asiakasvaatimuksina ovat Siemensin logiikkalaitteet.

# 2 YSP OY

"YSP Oy on suomalainen älykkään liikenteen ja erilaisten älykkäiden ohjausjärjestelmien asiantuntijayritys." (YSP lyhyesti, ei pvm.) Älyliikenteen toteutuksiin kuuluu muun muassa maantieliikenne ja moottoritiet, tunnelit, avattavat sillat, kanavien sulut, raja-asemat ja pysäköinnin opastus. Yrityksen logo on esitetty kuvassa 1. (Älyliikenne, ei pvm.)

KUVA 1. YSP Oy:n logo (YSP Oy, ei pvm).

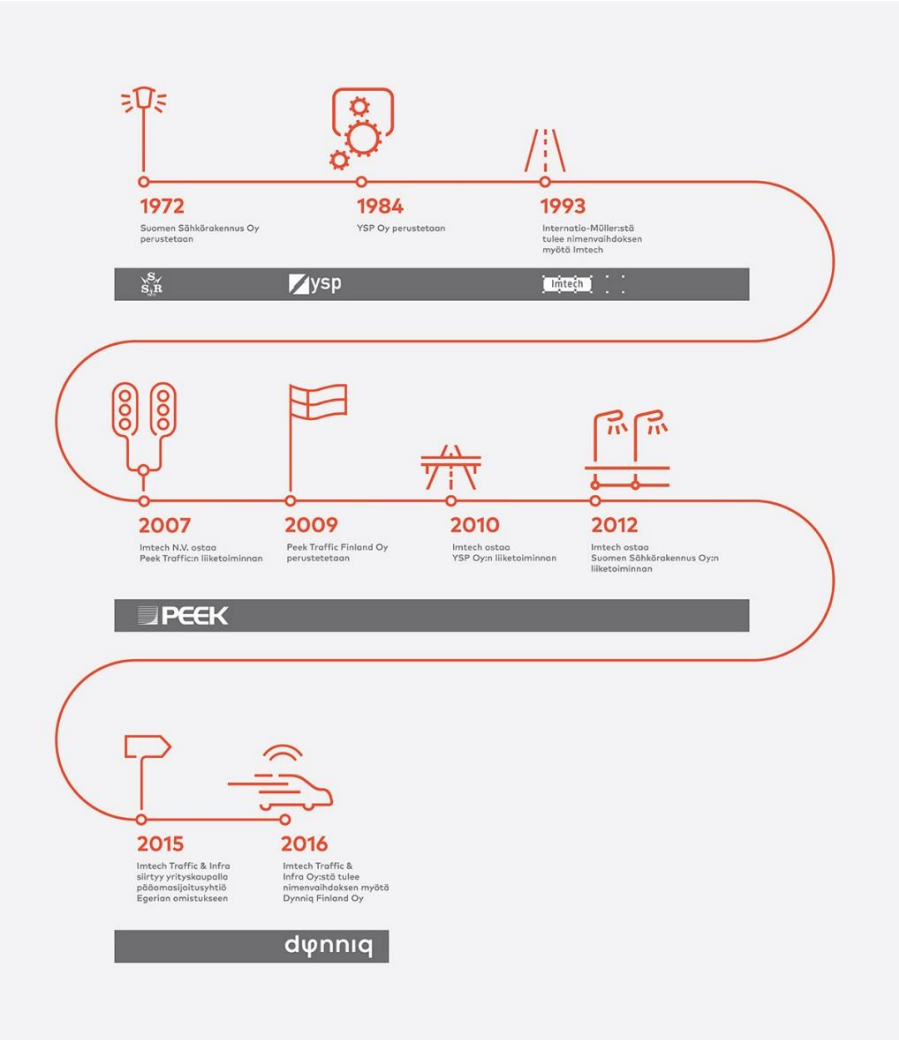
YSP Oy on Jyväskylässä sijaitseva insinööritoimisto, jonka henkilöstöön kuuluu n. 40 henkilöä. Palveluihin kuuluvat automaatio-, sähkö- ja ICT-suunnittelun lisäksi ylläpito- ja huoltotoiminta.
Laadun varmistamiseksi yrityksen laatujärjestelmä seuraa SFS-EN ISO 9001:2015 -standardin mukaisia vaatimuksia (YSP Laatujärjestelmä, ei pvm). YSP on osa Dynniq-konsernia, joka on yksi Euroopan johtavista älykkään liikenteen ratkaisutoimittajista (YSP lyhyesti, ei pvm). Kuvassa 2 on yrityksen lyhyt esittely.




KUVA 2. YSP lyhyesti (YSP Oy, ei pvm).

## 2.1 YSP Oy:n historia

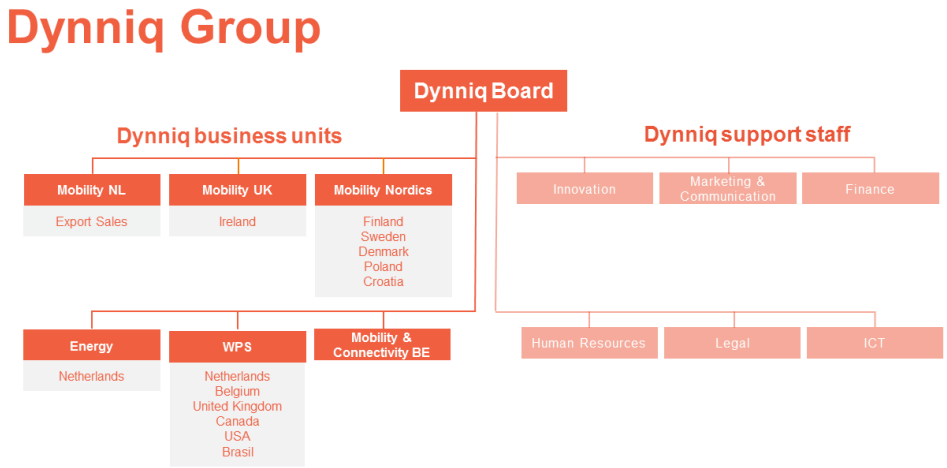
Yrityksen liiketoiminta alkoi vuonna 1984 jolloin toimialana oli sähkö- ja automaatiosuunnittelu. 90-luvun puolivälissä yritys laajensi toimintaa ICT-puolelle, johtuen teknisten järjestelmien ja vaatimusten lisääntymisestä. Vuonna 2010 hollantilainen pörssiyhtiö Imtech osti YSP:n ja liittää osaksi Traffic & Infra liiketoimintayksikköä. Kuusi vuotta myöhemmin Imtech-konserni myi Traffic & Infra liiketoimintayksikön pääomasijoittajille, jonka jälkeen liiketoiminta brändättiin nimelle Dynniq, katso kuva 3. (YSP Historia, ei pvm.)




KUVA 3. YSP Oy:n Historia (YSP Oy, ei pvm).

## 2.2 Dynniq

Dynniq:n pääkonttori sijaitsee Hollannissa, sen lisäksi sillä on useita tytäryhtiöitä, jotka tarjoavat useita asiantuntevia palveluja ympäri maailmaa, katso kuva 4. YSP on osa Dynniq Nordics:ia ja toimii usein hankkeissa tiiviissä yhteystyössä Dynniq Finland Oy:n kanssa. (Dynniq, ei pvm; YSP Oy, ei pvm.)




KUVA 4. Dynniq-konsernin rakenne (Dynniq, ei pvm; YSP Oy, ei pvm).

# 3 OPINNÄYTETYÖN KUVAUS, TAVOITE JA RAJAUS

Työn tavoitteena oli saada YSP:lle tarvittavat, alustavat Siemensin työkalut, ns. perusta ja niiden käyttöohjeet, jotta niitä voisi hyödyntää tulevaisuudessa projektien suunnittelussa, PLC-kirjastojen organisoinnissa sekä PLC-sovelluksien testausten automatisoinnissa. Yrityksellä ei ollut aiempaa perustuntemusta kummempaa osaamista Siemensin TIA Portalista, tämän simulaattoreista, niiden toiminnoista ja operoinnista, sekä mahdollisesta potentiaalista osana automaatiojärjestelmiä. Yrityksen automaatiototeutukset oli suunniteltu ja toteutettu pääosin Schneiderin sovelluksilla ja laitteilla. Asiantuntemuksen laajentaminen Siemensin tuotteisiin parantaa myös yrityksen kilpailukykyä markkinoilla ja monipuolisuutta tulevaisuudessa sekä mahdollistaa paremman vastaamisen asiakkaiden vaatimuksiin.

Opinnäytetyö rajattiin kolmeen pääkohtaan;

- PLC-kirjaston suunnittelu.
  Tutkitaan TIA Portal -kirjaston rakennetta niin tiedostollisella kuin ohjelmallisella tasolla. Suunnitellaan miten TIA Portalin PLC-kirjastoja käytetään versionhallintajärjestelmän kanssa, jotta lopputulos olisi mahdollisimman vaivaton käyttö TIA Portalista käsin. Lisäksi tutkitaan, miten lohkoja saadaan tekstimuotoon TIA Portalin ja Opennessin import & export -toiminnoilla, sekä miten niitä voidaan muokata TIA Portalin ulkopuolella. Luodaan suunniteltu toteutuskokonaisuus ja viedään kirjasto SVN:ään.

- Siemens PLC-sovelluksien lisääminen jatkuvaan integraatioprosessiin.
  Tutkitaan Siemensin tarjoamia logiikkasimulaattoreita. Suunnitellaan TIA Portal -projektin automaattinen lataus automaatiopalvelimella olevalle simulaattorin logiikkainstanssille. Tutkitaan mahdollisuuksia muokata ja päivittää projektissa olevia muuttujia paremman ohjelman version jäljittämisen vuoksi.

- TIA Portal -kirjastojen ja simulaattorien dokumentointi.
  Luodaan yrityksen sisäiset käyttöohjeet TIA Portalin ja versionhallinnan yhteiseen käyttöön, sekä dokumentoidaan eri simulaattorien toiminnat, niiden erot ja operointi.
  Käyttöohjeista täytyy tulla selväksi kirjaston rakenne, sisäisen rakenteen muokkaus, lohkojen luominen ja vieminen kirjastoon, niiden muokkaaminen, päivitys, kommentointi ja versiointi. Lisäksi ohjeisiin lisätään, miten versionhallintajärjestelmästä haetaan ja miten sinne viedään päivitetty kirjasto.

# 4 KEHITYSYMPÄRISTÖT JA TYÖKALUT

## 4.1 TIA Portal

Totally Integrated Automation (TIA) Portal on Siemensin kehittämä automaatio-ohjelmointityökalu, johon on integroitu aiemmat erilliset automaatiosuunnitteluun ja -ohjelmointiin tarkoitetut ohjelmat. Näihin kuuluvat Siemensin STEP 7 -logiikkaohjelmointi, WinCC-käyttöliittymäsuunnittelu ja StartDrive Sinamics -sähkökäyttösuunnittelu. (Siemens AG, ei pvm.)

Simatic Step 7 on Siemensin vuonna 1995 julkaisema logiikkaohjelmointityökalu, joilla voidaan ohjelmoida Siemensin tuoteperheiden logiikkalaitteita, katso kuva 5 (Siemens AG, ei pvm). Step 7 tukee logiikkaohjelmointia 5:llä eri kielellä, jotka noudattavat logiikkaohjelmointikielille suunnattua IEC 61131-3 standardia;

- STL, Statement List vastaa standardin Instruction List -kieltä. Tekstipohjainen ohjelmointikieli.

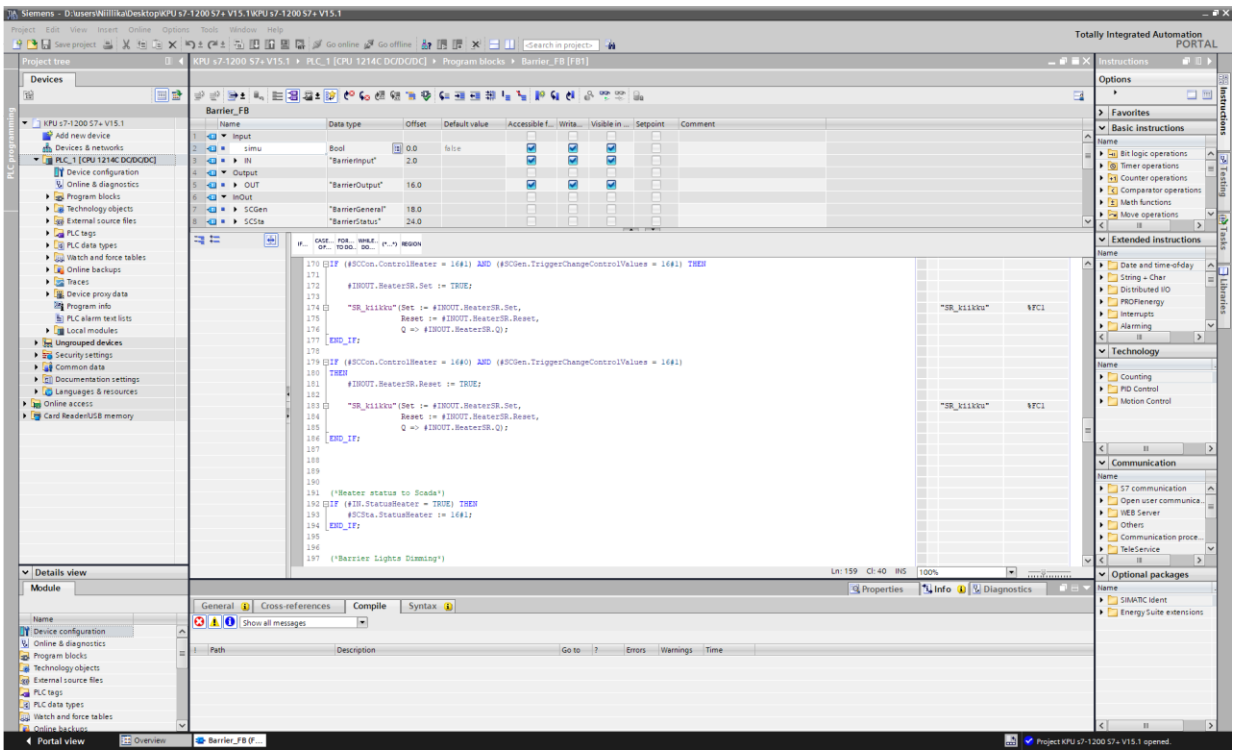
- SCL, Structured Control Language vastaa standardin Structured Text -kieltä. PASCAL:iin perustuva korkeamman luokan tekstipohjainen ohjelmointikieli.

- S7 Graph/SFC, Sequential Function Chart. Graafinen sekvenssien ohjelmointiin tarkoitettu logiikkaohjelmointikieli. Esitystapa muistuttaa ylhäältä alaspäin virtaavaa vuokaaviota.

- FBD, Function Block Diagram. Graafinen logiikkaohjelmointikieli. Esitystapa muistuttaa elektroniikan ja sähkötekniikan piirikaavioita.

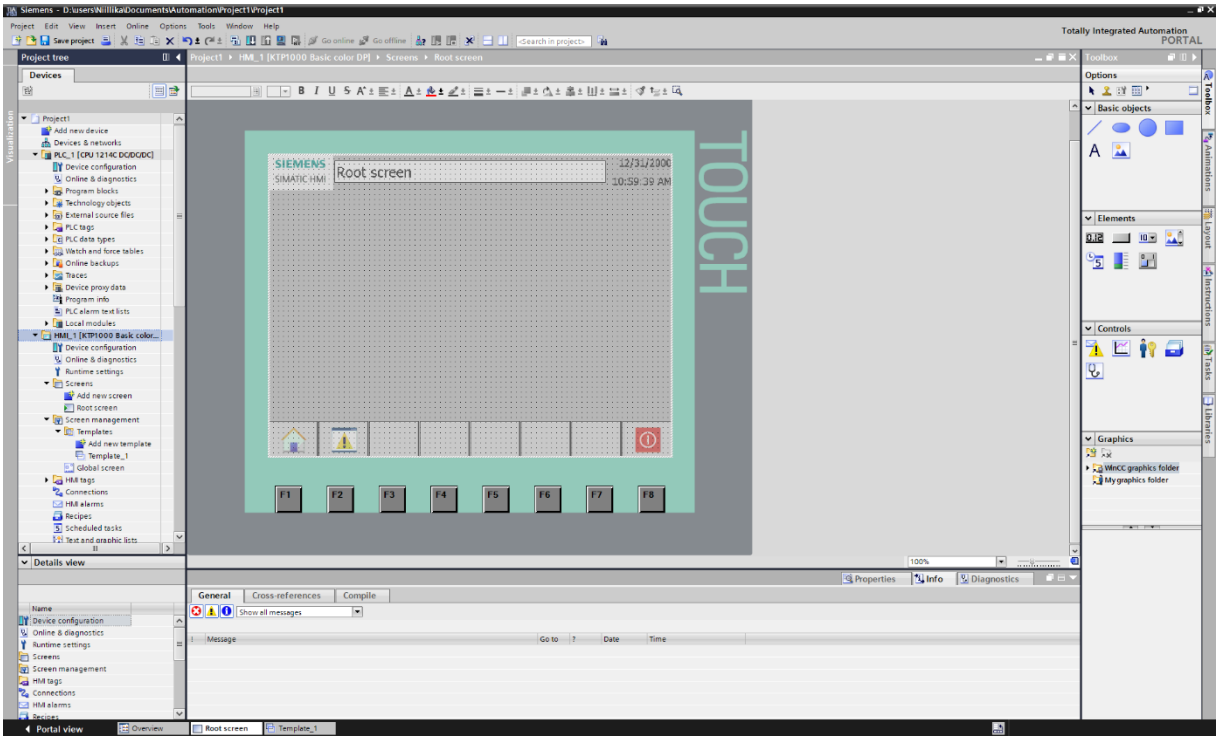
- LD, Ladder Diagram. Graafinen logiikkaohjelmointikieli. Esitystapa perustuu piirikaavioihin. Komponentit esitetään releinä, keloina sekä ohjelmalohkoina.

  (Siemens AG, 2015; Siemens AG, 2018a.)



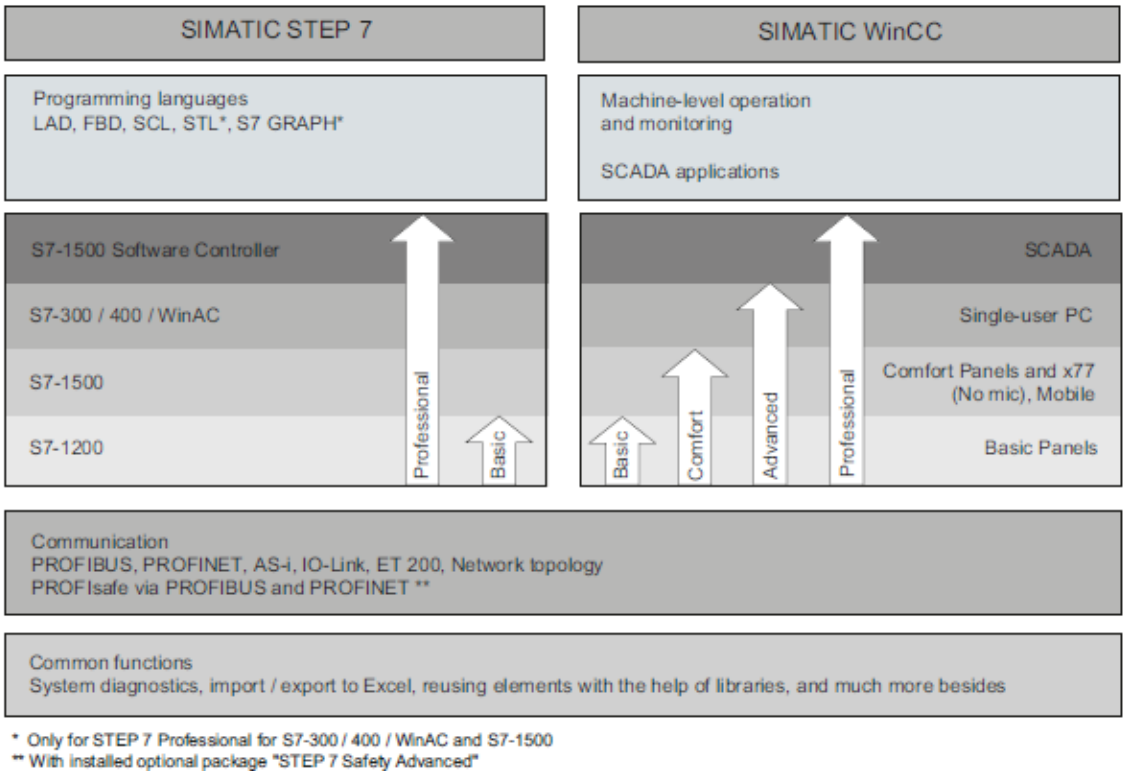
Kuva 5. Step 7 ja SCL-ohjelmointikieli TIA Portalissa (Siemens AG TIA Portal Step 7 V15.1, 2019).

Simatic WinCC on Siemensin vuonna 1996 julkaisema valvomo- ja käyttöliittymäohjelmointiin tarkoitettu työkalu, katso kuva 6. WinCC:llä voidaan ohjelmoida eri tuoteperheiden HMI-paneeleita ja suurempia SCADA-toteutuksia. (Siemens AG, ei pvm).



Kuva 6. WinCC TIA Portalissa (Siemens AG TIA Portal WinCC V15.1, 2019).

Step 7 ja WinCC -ohjelmointiin tarvitaan erillinen lisenssi, jotka tarjoavat erilaisia ulottuvuuksia automaatiosuunnittelussa ja -ohjelmoinnissa tasojensa perusteella. Professional-lisenssi on korkeimman luokan lisenssi ja kattaa kaikki alemman luokan lisenssit, katso kuva 7. (Siemens AG 2018b, 5.)




Kuva 7. Step 7 ja WinCC tuotteiden ohjelmointialueet sekä lisenssivaatimukset (Siemens AG, 2018b)

## 4.2 Simulaatiotyökalut

Siemensillä on tarjolla kahta eri logiikkasimulaattoria S7-PLCSIM ja erikseen hankittava S7-PLCSIM Advanced. Simulaattoreilla voidaan simuloida virtuaalista logiikkalaitetta ja niitä käytetään ohjelmoitujen PLC-sovelluksien testaukseen. Simulaattorin Advanced-versio on tarkoitettu vaativamman ja kehittyneemmän ympäristön testaustyökaluksi. Yhdellä tietokoneella voi olla molemmat simulaatiotyökalut asennettuna, pelkästään silloin kun PLCSIM versio on V15 tai uudempi. Advancedin ja PLCSIM V15.1 välisen kommunikoinnin simulointi ei ole tällä hetkellä mahdollista, kun taas PLCSIM V5.4 SP8:n kanssa kommunikointi on mahdollista. Simulaattorin ja projektin välinen S7-kommunikointi tapahtuu Softbus-rajapinnan kautta. (Siemens AG, 2017a.)

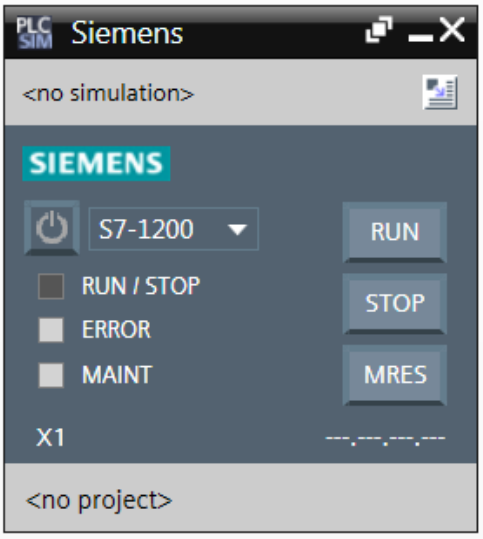
### 4.2.1 S7-PLCSIM

S7-PLCSIM logiikkasimulaattori on Siemensin perustyökalu logiikkasovelluksien testauksiin. Sovelluksen vanhempi versio V5.4 tukee Simatic-tuoteperheitä S7-300, S7-300F, S7-400 ja S7-400F. Versio V14 ja uudemmat tukevat tuoteperheen tuotteita S7-1200, S7-1200F, S7-1500, S7-1500F, ET200SP, ET200SPF. Tiettyjen tuoteperheiden simuloimiseen tarvitaan siihen verrattavissa oleva TIA Portal -lisenssi. Esimerkiksi S7-1500 logiikkalaitteen simulointi vaatii Professional-lisenssin. Samalla koneella voi olla simulaattorien kaksi eri pääversiota asennettuna ja niitä voidaan käyttää yhtäaikaisesti, esimerkiksi S7-PLCISM V5.4.8 ja V15.1. Katso kuva 8. (Siemens AG, 2017a.)

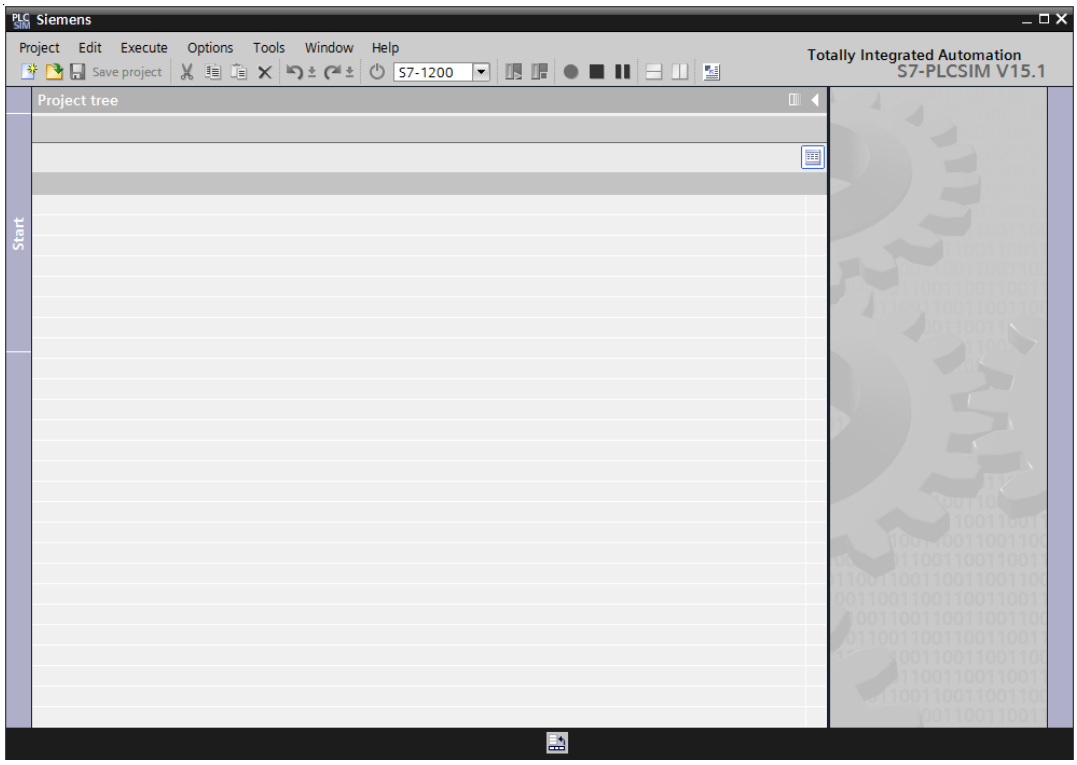
| Function | PLCSIM Advanced V2.0 | PLCSIM V15 | PLCSIM V5.x |
|---|---|---|---|
| Runtime | Independent | Programming with STEP 7 | Programming with STEP 7 |
| User interface | Control Panel | Look&Feel of TIA Portal | Look&Feel of STEP 7 V5.x |
| Communication | Softbus, TCP/IP | Softbus only | Softbus only |
| Supported CPU families | S7-1500 (C, T, F),<br>ET 200SP, ET 200SP F | S7-1200 (F),<br>S7-1500 (C, T, F),<br>ET 200SP, ET 200SP F | S7-300, S7-300F<br>S7-400, S7-400F |
| API for co-simulation[1] | ✓ | - | - |
| Web server | ✓ | - | - |
| OPC UA | ✓ | - | - |
| Process diagnostics | ✓ | ✓ | - |
| S7 communication | ✓ | Softbus | Softbus |
| Open user communication | ✓, UDP only via TCP/IP | Softbus | - |
| Traces[2] | ✓ | (✓) | - |
| Motion[3] | ✓ | (✓) | - |
| Protected blocks (KHP) | ✓ | ✓, for S7-1500 CPUs only | - |
| Multiple instances | Up to 16 | Up to 2 | - |
| Support of distributed instances | ✓ | - | - |
| Virtual time | ✓ | - | - |
| Connection of real CPUs/HMIs | ✓ | - | - |
| DNS usage | ✓ | - | - |
| Virtual memory card | ✓ | - | - |
| Communication between the instances | - | PLCSIM as of V12 and PLCSIM V5.x can be installed and operated on the same PC or the same virtual machine. | |
| | - | Instances of PLCSIM as of V12 can communicate via Softbus with PLCSIM V5.x. | |
| | PLCSIM Advanced 2.0 and PLCSIM V15 and higher can be installed and operated on the same PC or the same virtual machine. The communication between the two applications **cannot** be simulated. | | - |
| | PLCSIM V5.4 SP8 is automatically installed with PLCSIM Advanced. The communication between the two applications can be simulated.<br>Instances of PLCSIM Advanced can communicate via Softbus with PLCSIM V5.4 SP8. | | |

Kuva 8. Simulaatiotyökalujen erot (Siemens AG 2017a, 29).

Simulaattoria voidaan käyttää joko kompaktissa näkymässä, katso kuva 9, tai laajemmassa projektinäkymässä, katso kuva 10. Kompaktissa näkymässä voidaan suorittaa toimenpiteet instanssin simuloimiseen kuten logiikan käynnistys, pysäytys ja tarvittaessa nollaus. PLCSIM:llä voidaan luoda simulointitaulukkoja (SIM table), joihin voidaan viedä ja tarkastella Step 7 -projektissa olleita muuttujia. Työkalulla voidaan manipuloida osoitteita ja symbolisia muuttujia sekä luoda ja nauhoittaa omia sekvenssejä esimerkiksi tietyin aikavälein. Sekvenssissä voidaan esimerkiksi muokata tiettyjen muuttujien arvoja 10 minuutin päästä instanssin käynnistyksen jälkeen. (Siemens AG, 2017b.)




Kuva 9. S7-PLCSIM V15.1 kompaktissa näkymässä (Siemens AG S7-PLCSIM V15.1, 2019).

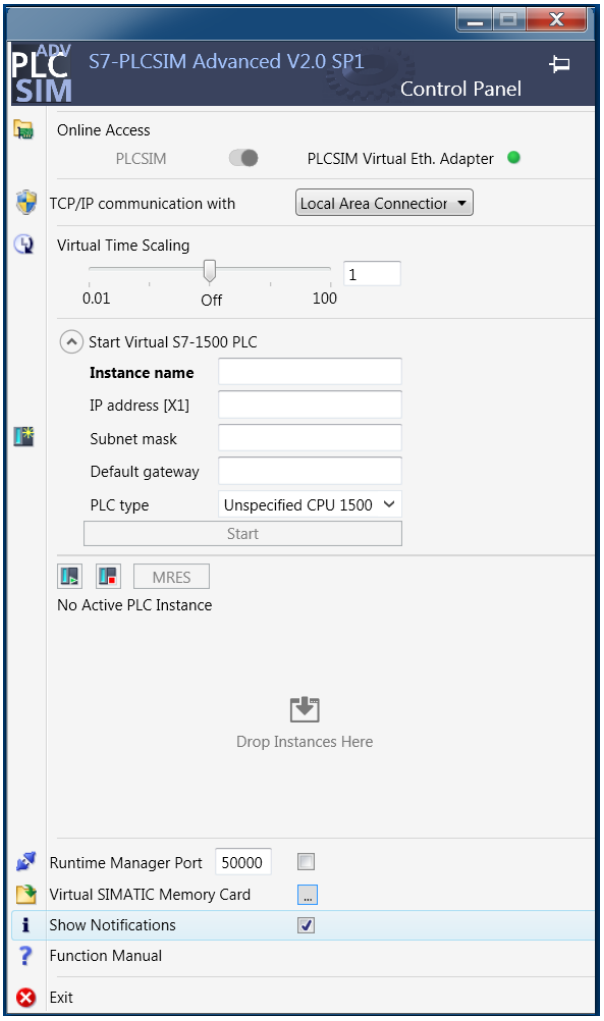



Kuva 10. S7-PLCSIM V15.1 kokoruututilassa (Siemens AG S7-PLCSIM V15.1, 2019).

### 4.2.2 S7-PLCSIM Advanced

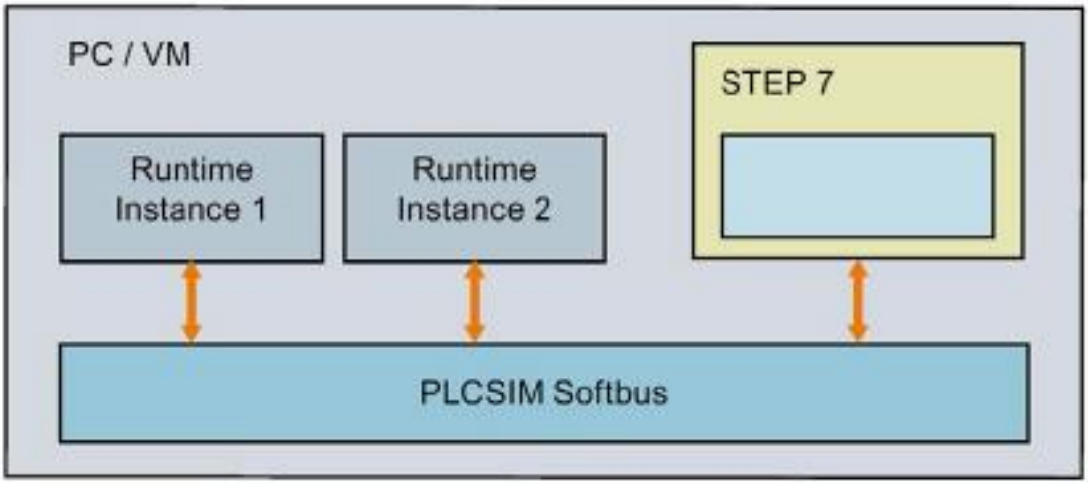
S7-PLCSIM Advanced tarvitsee erillisen lisenssin toimiakseen ja toimii edistyneempänä simulointityökaluna S7-1500 ja ET200 tuoteperheille. Simuloituja instansseja voi olla 16 kappaletta. Advanced ei tule TIA Portalin mukana, vaan on erikseen ladattavissa Siemensin verkkosivuilta. Advancedin asennuspaketin mukana tulee myös PLCSIM V5.4.8. (Kuva 8).
Suurimpina eroina aiemmin mainittuihin työkaluihin, Advancedista löytyy ohjelmointirajapinta, TCP/IP kommunikoinnin vaihtoehto, logiikan virtuaalinen ajan skaalaus ja mahdollisuus käyttää virtuaalista muistikorttia. Verkkosovittimen paikallinen kommunikointi tapahtuu softbussin yli. TCP/IP kommunikoinnin lisäksi voidaan simuloida hajautettuja järjestelmiä TCP/IP:n avulla, katso kuva 14 ja 15. (Siemens AG, 2017a.)

S7-PLCSIM Advanced eroaa muista simulointityökaluista myös ulkoasun kannalta. Sillä on erillinen ohjauspaneeli, josta instansseja luodaan ja hallitaan, katso kuva 11.
Sekvenssien luomiseen ja muiden haluttujen toimenpiteiden, kuten muuttujien manipulointiin täytyy käyttää avuksi sen rajapintaa C# tai C++ -sovelluksessa. Hyödyntääkseen rajapintaa, rajapintakirjasto täytyy referoida valitussa C# tai C++ -ohjelmankehitysympäristössä. Simulaattorin rajapintakirjaston tiedostomuoto on dll (Dynamic-link library). (Siemens AG, 2017a)



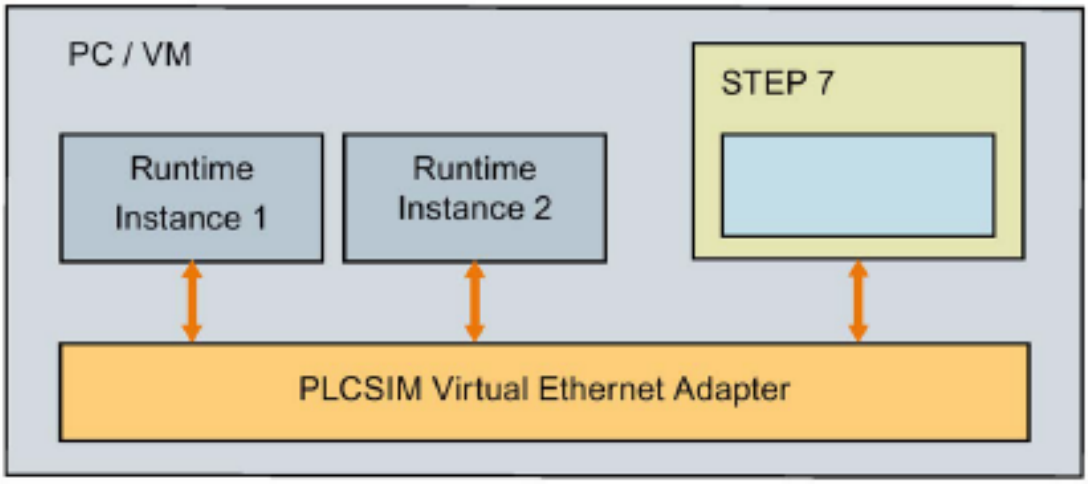

Kuva 11. S7-PLCSIM Advanced V2.0 SP1 ohjauspaneeli (Siemens AG S7 PLCSIM Advanced V2.0 SP1, 2019).

Kuvan 12 esimerkissä on kuvattu paikallisen tietokoneen tai virtuaalikoneen sisäinen S7-kommunikointi kahden logiikkainstanssin ja Step7:n/TIA Portalin välillä.



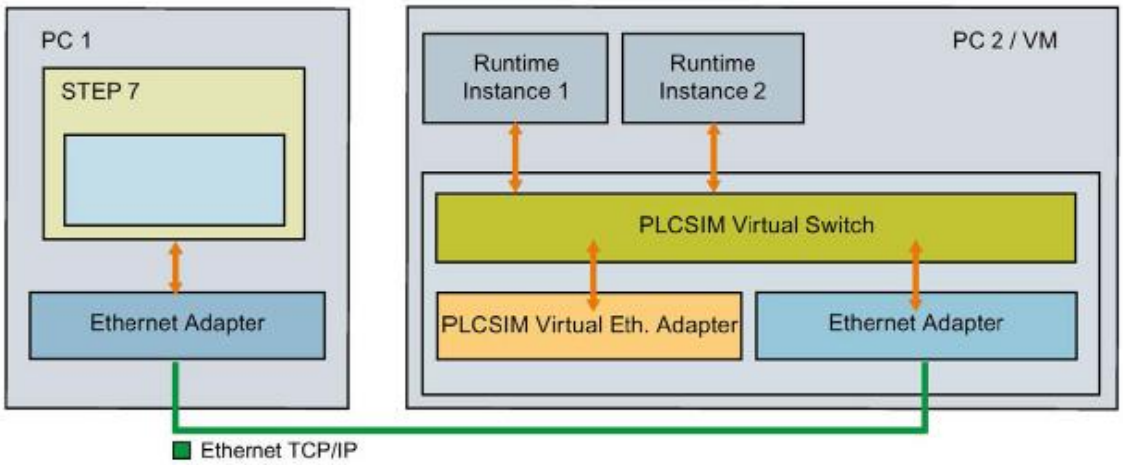

Kuva 12. Paikallinen S7-kommunikointi Softbussilla toimintakaavio (Siemens AG 2017a, 53).

Kuvan 13 esimerkissä on kuvattu paikallisen TCP/IP kommunikoinnin toiminta tietokoneella instanssien ja Step 7:n välillä. TCP/IP kommunikointi tapahtuu PLCSIM Virtual Ethernet -sovittimen avulla, joka käyttäytyy kuten oikea verkkoliitäntä.



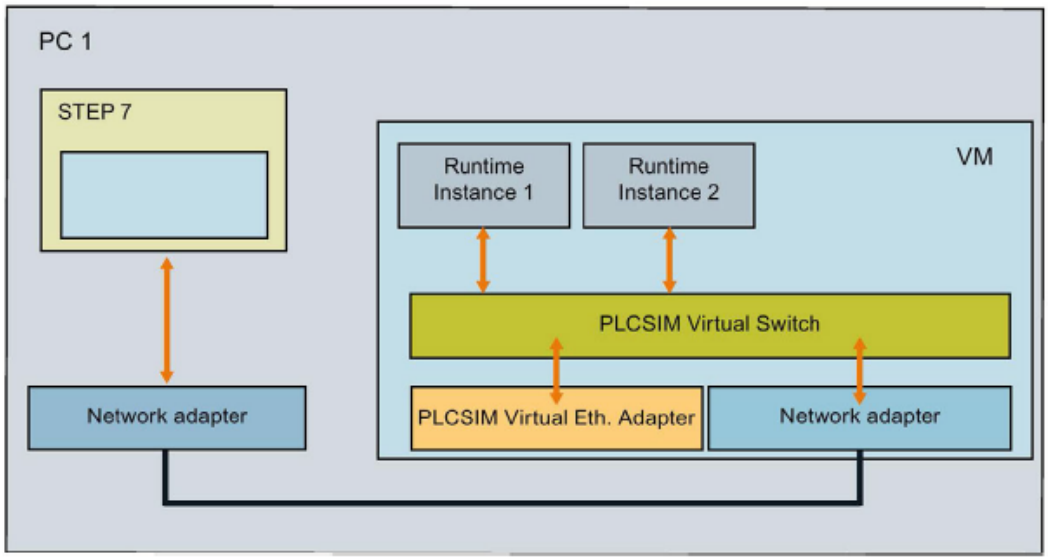

Kuva 13. Paikallinen S7-kommunikointi TCP/IP toimintakaavio (Siemens AG 2017a, 53).

Kuvan 14 esimerkissä kuvataan kahden eri tietokoneen välistä hajautettua TCP/IP -yhteyttä simulaattorin avulla. Toisessa tietokoneessa on Step 7 -sovellus/TIA Portal ja toisessa on PLCSIM Advanced instanssineen. Tietokoneiden välinen yhteys on kuvan tapauksessa ethernet-kaapeli.




Kuva 14. Hajautuksen simuloiminen kahden tietokoneen välillä TCP/IP toimintakaavio (Siemens AG 2017a, 54).

Kuvan 15 mukainen viimeinen esimerkki kuvaa yhden tietokoneen hajautetun simuloinnin toimintaa. Tietokoneella on virtuaalikone, joka on varustettu PLCSIM Advancedillä. Kommunikointi koneiden välillä tapahtuu verkkosovittimien avulla.



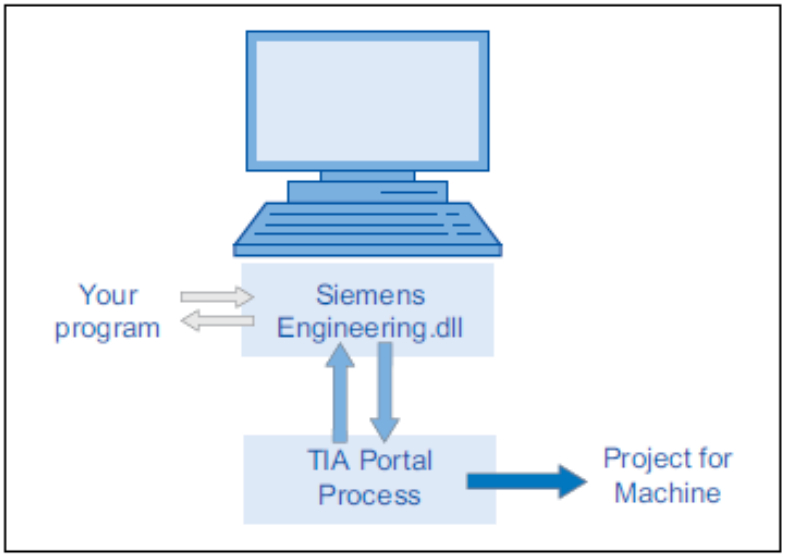

Kuva 15. Hajautetun yhteyden simuloiminen virtuaalikoneen ja verkkosovittimien avulla (Siemens AG 2017a, 55).

(Siemens AG 2017a, 53-55.)

## 4.3 Openness-rajapinta

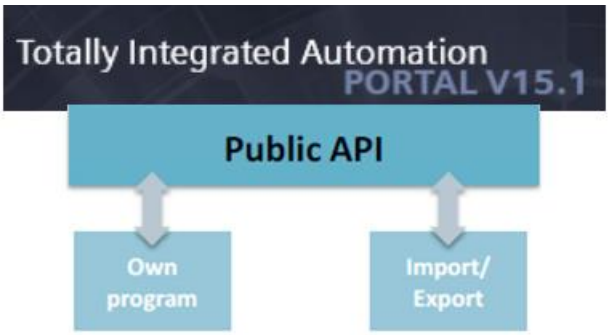
Openness API on TIA Portalin ohjelmointirajapinta, jota voidaan käyttää hyväksi logiikkaohjelmoinnissa esimerkiksi automatisoimalla toistuvia tehtäviä. Openness-rajapinta on tullut WinCC:n ja Step 7:n mukana versiosta V13 SP1 asti. (Siemens AG, 2018c.)

Rajapinta mahdollistaa TIA Portal -projektin eri komponenttien ohjelmoinnin ja interaktiivisen tiedonvaihdon kehittäjän oman sovelluksen kautta. Rajapinnan avulla kehittäjä pääsee omalla sovelluksellaan käsiksi TIA-projektiin, logiikkaan ja käyttöliittymän tietoihin.
Openness API:lle sopivat ohjelmointikielet ovat .NET-alustan ohjelmointikielet C# ja Visual Basic.net. Kuten PLCSIM Advancedin rajapinta, Opennessin rajapintakirjasto on myös muotoa dll (Dynamic-link library) ja tarvitsee referenssin ohjelmankehitysympäristössä, katso kuva 16. (Siemens AG, 2018d.)



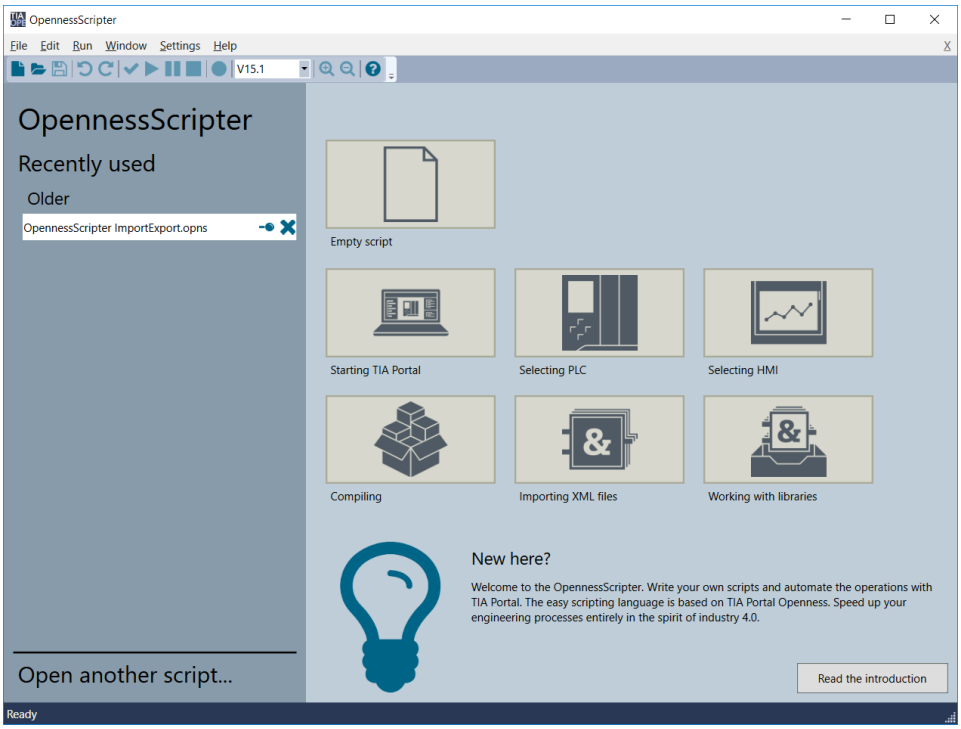

Kuva 16. TIA projektin sisäiseen dataan pääsy rajapintakirjaston ja oman ohjelman avulla (Siemens AG 2018e, 33).

Opennessin avulla kehittäjä voi esimerkiksi tuoda projektista lohkoja tekstimuodossa levyllä ja myös viedä haluttuja lohkoja projektiin, katso kuva 17 (Siemens AG 2018c, 7–8).




Kuva 17. Openness-rajapinta TIA Portal -projektin lohkojen välitykseen kehittäjän sovelluksen avulla (Siemens AG 2018c, muokattu).

Siemens on kehittänyt oman Openness-sovelluksen nimeltä Openness Scripter, jolla voidaan tehdä projektiin liittyviä tehtäviä, kuten esimerkiksi projektin avaaminen, sulkeminen ja lohkojen välitys levyaseman ja projektin välillä, katso kuva 18. Openness Scripterillä luodaan ohjelmia yksinkertaisten tekstipohjaisten komentojen avulla. (Siemens AG, 2018e; Siemens AG, 2018f.) Kuvassa 18 näkyy Openness Scripterin käyttöliittymä. Jos halutaan tehdä kehittyneempiä toimenpiteitä projektille tai sen sisällölle, kuten projektiin laitteiden lisääminen tai PLC-sovelluksen lataaminen logiikkaan, täytyy kehittäjän luoda oma .NET-pohjainen sovellus esimerkiksi Visual Studiolla.




Kuva 18. Siemensin Openness Scripterin käyttöliittymä (Siemens AG Openness Scripter, 2019).

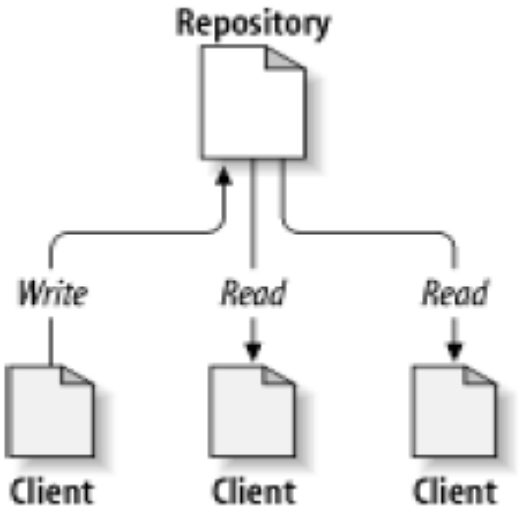
## 4.4 Visual Studio

Microsoft Visual Studio on ohjelmankehitysympäristö, jolla voidaan kehittää Windows-, mobiili- ja web-sovelluksia. Visual Studion ensimmäinen versio julkaistiin vuonna 1997 ja uusimpia versioita kehitetään edelleen. Seuraavan Visual Studion julkaisu on tulossa elokuussa 2019. Visual Studio 2017 tukee useita eri ohjelmointikieliä kuten C#, Visual Basic, C++, HTML ja tarvittaessa ohjelmistoon saadaan liitäntä myös muille ohjelmointikielille. (Microsoft Visual Studio, 2017.)

## 4.5 Versionhallintajärjestelmä

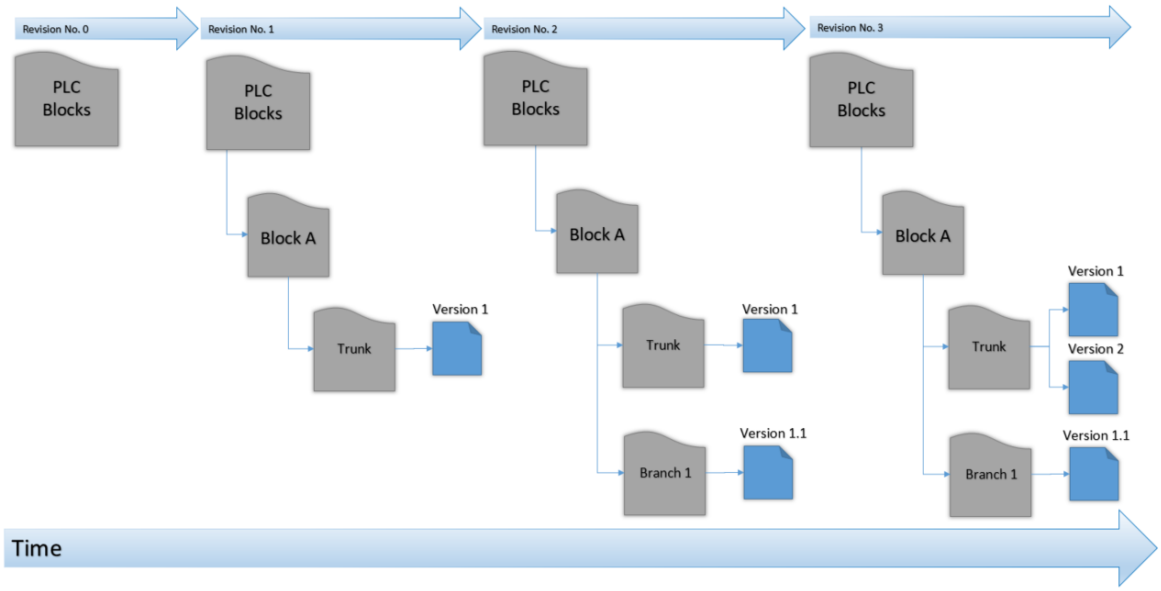
Versionhallinta on tapa hallita mitä tahansa tiedostoa ja sen versioita sekä tiedoston ylläpitoa ja muokkaamista. Versionhallintaa käytetään usein ohjelmistokehityksessä työkappaleiden hallinnoimisessa, jatkuvassa kehittämisessä ja eri versioiden välisien erojen tunnistamisessa. Versionhallinnan tavoitteet ovat yhtäaikaisen muokkauksen mahdollistamisen ja versioinnin lisäksi myös aiempien versioiden arkistointi ja palauttaminen (Sink 2011, 1). Modernit versionhallintajärjestelmät mahdollistavat useamman käyttäjän yhtäaikaisen yhteyden järjestelmän tiedostojen säilytyspaikkaan sekä tehdä muutoksia siellä olleisiin tiedostoihin (Collins-Sussman, Fitzpatrick & Pilato 2011; Git, ei pvm). Toisin sanoen versionhallintajärjestelmä on tavallaan tiedostojen kirjanpito- ja varastointijärjestelmä.

Järjestelmät koostuvat nykyään pääasiassa palvelimella olevasta arkistosta (engl. repository), käyttäjistä ja työkappaleista (Collins-Sussman ym. 2011, 1–2). Järjestelmän käyttäjät hakevat ja vievät päivitetyt tiedostot palvelimen säilytyspaikasta toisten käyttäjien ulottuville, katso kuva 19. Eri järjestelmien välillä on eroja, mutta pääpiirteet pysyvät samankaltaisina.



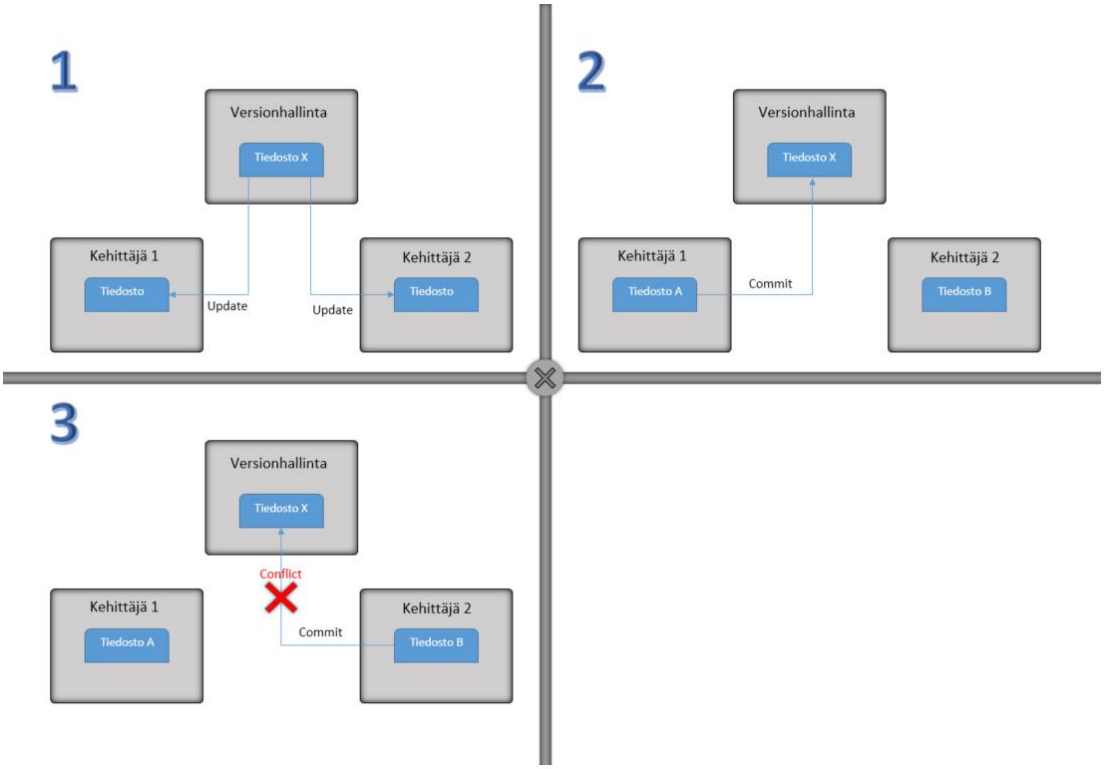

Kuva 19 Tyypillinen versionhallintajärjestelmä (Collins-Sussman ym. 2011, 1).

Järjestelmät seuraavat muutoksia tarkkailemalla tiedoston uniikkia binäärijonoa, joka muuttuu, kun esimerkiksi lähdekoodia tiedoston sisällä muokataan. Huomatessaan tiedostossa muutoksen, järjestelmä ilmoittaa siitä käyttäjälle, jolloin käyttäjä voi viedä tehdyt muutokset versionhallintajärjestelmän arkistoon. Kun tiedosto viedään versionhallintajärjestelmään, sille määrätään tietty versio-/revisionumero, jota toiset käyttäjät voivat hyödyntää selvittääkseen versiokohtaisesti historiatiedot, muutokset, kommentit, aikaleimat ja muokkaajat. Muiden käyttäjien muokatessa tiedostoa ja päivitetyn version vietyään järjestelmään, järjestelmä luo tiedostolle taas uuden revisionumeron. Tiedoston aiemmat versiot voidaan halutessa palauttaa. (TortoiseSVN, ei pvm; Git, ei pvm.)
Tyypillisen versionhallinnan arkisto voidaan esittää puukaavioina, jonka haarat kasvavat uusien revisioiden tullessa (Collins-Sussman ym. 2011, 7–8), katso kuva 20.



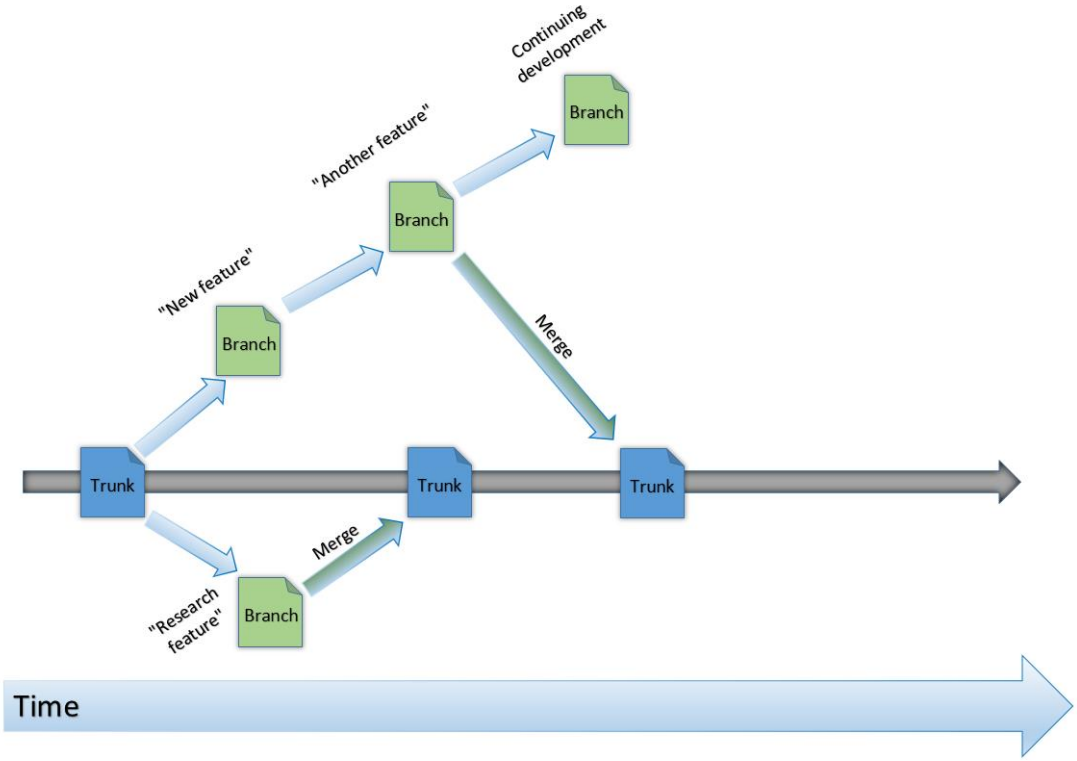

Kuva 20. Havainnollistava puukaavio palvelimen arkiston kasvusta uusien revisioiden johdosta (Collins-Sussman ym. 2011, muokattu).

Järjestelmissä voi esiintyä ongelmia tapauksissa, jossa kaksi kehittäjää hakee järjestelmästä saman version ja toinen heistä päivittää oman version palvelimelle. Kun jälkimmäinen kehittäjä on viemässä omaa versiota järjestelmään, nykyiset järjestelmät ilmoittavat työkappaleen vanhentuneesta lähtötilanteesta palvelimella olevan versioon verrattuna. Jos kehittäjien työkappaleeseen tehdyt muutokset menevät päällekkäin, kutsutaan tilannetta konfliktiksi, katso kuva 21. (Collins-Sussman ym. 2011, 7–8; Ernst, 2012.) Eri versionhallintajärjestelmät ovat kehittäneet tapoja ratkaista nämä ongelmat automaattisesti, mutta joskus jälkimmäisen kehittäjän on ladattava palvelimelta toisen kehittäjän päivitetty versio ja yhdistää manuaalisesti nämä kaksi työkappaletta uudeksi versioksi.




Kuva 21. Konflikti versionhallintajärjestelmissä (Collins-Sussman ym. 2011, muokattu).

Versionhallinnassa työkappaleen kehitysprosessin alussa kappale koostuu rungosta, jota aletaan työstämään. Kehitysprosessin aikana kappaleen rungosta jakautuu useampi haara, jotka kuvaavat uusia versioita ja niiden ominaisuuksia. Nämä haarat voidaan yhdistää eri haarojen ja rungon kanssa, joka luo taas uudenlaisen version kappaleesta, katso kuva 22.




Kuva 22. Työkappaleen kehitysprosessi/elämänkaari versionhallinnassa.

### 4.5.1 Perustoiminnot

Versionhallintajärjestelmät sisältävät 18 kappaletta perustoimintoja, jotka perustuvat tiedostojen siirtämiseen, muokkaamiseen, käsittelyyn ja muihin versioinnille tarpeellisiin ominaisuuksiin:
Create, checkout, commit, update, add, edit, delete, rename, move, status, diff, revert, log, tag, branch, merge, resolve ja lock. Komennot voivat hieman poiketa toiminnaltaan eri versionhallintajärjestelmissä mutta pääpiirteet ovat samat. (Sink, 2011.)

Useat eri versionhallintajärjestelmät sisältävät perustoimintojen lisäksi useita kymmeniä, jopa yli sata eri käskyjä, jotka helpottavat järjestelmien kanssa työskentelyssä. Niiden tapauskohtaisuuden ja epäoleellisuuden takia tässä kontekstissa nämä jäävät mainitsematta. (Collins-Sussman ym. 2011, 248–414; Git, 2019.)

**Create-**toiminto luo uuden, tyhjän versionhallinnan tiedostojen säilytyspaikan omalle levylle, johon kaikki versionhallintaan menevät tiedostot talletetaan. Versionhallinta pitää säilytyspaikan tiedostojen muutoksista kirjaa.

**Checkout-**käsky luo määritetyt työkappaleet omalle tiedostojen säilytyspaikalle versionhallinnasta.

**Commit** tarkoittaa uuden työkappaleen version viemistä versionhallintaan, joka korvaa vanhemman version (kommitointi). Muutoksien viemisen yhteydessä yleensä järjestelmä pakottaa myös sen kommentoinnin, johon yleensä kirjataan mitä muutoksia kappaleeseen on tehty. Esimerkiksi poistettiinko lohkosta bugi, lisättiinkö uusi ominaisuus vai parannettiinko sen toimintaa yleisesti. Järjestelmät pitävät kirjaa jokaisesta kommitista ja niiden mukana tulleista kommenteista.

**Update** tarkoittaa kommitin vastakohtaa, kappaleen uusimman version tuomista versionhallinnasta omalle levylle. Update päivittää vanhentuneet tiedostot levyllä. Jos tiedostoon on tehty muutoksia ja sitä ei ole viety versionhallintaan, update toiminto yrittää joissain tapauksissa yhdistää tuodun version levyllä olevan muokatun version kanssa.

**Add-**toiminto lisää tiedoston tai polun tiedostojen säilytyspaikkaan, joka seuraavassa kommitissa viedään versionhallintaan.

**Edit** aloittaa tiedoston muokkauksen halutussa muokkausympäristössä, jos mahdollista.

**Delete** poistaa tiedoston säilytyspaikasta tai versionhallinnasta. Tiedoston historia jää talteen poistamisenkin jälkeen ja sen voi halutessa palauttaa.

**Rename** uudelleennimeää valitun kohteen halutuksi. Muutos tulee voimaan heti paikalliseen säilytyspaikkaan ja versionhallintaan vasta kommitin jälkeen.

**Move** siirtää tiedoston tai tiedostopolun.

**Status** kertoo työkappaleen tilan. Jos työkappaleelle on tehty muutoksia, komento esittää myös listan tehdyistä muutoksista.

**Diff** tulee englannin sanasta "differences", erot. Kertoo tarkemmin kappaleeseen tehdyt muutokset.

**Revert**-toiminto palauttaa tehdyt muutokset tekemättömiksi.

**Log** esittää muutoslokin, josta selviää kappaleeseen tehdyt muutokset niiden kanssa tulleiden kommenttien kanssa aikaleimoineen ja tekijöineen.

**Tag** lisää kappaleen versiolle "tagin" eli tunnisteen, jolla sitä voi hallita.

**Branch** haaroittaa tietyn version kahteen tai useampaan, jotta esimerkiksi kaksi eri ominaisuutta voidaan tehdä samankaltaisiin tiedostoihin.

**Merge** yhdistää haarojen muutokset työkappaleeseen.

**Resolve** antaa kehittäjän selvittää mergestä eli yhdistyksestä johtuneet konfliktit.

**Lock** laittaa tiedoston lukkoon, jolloin muut kehittäjät eivät voi sitä muokata.

Näiden lisäksi hajautetut versionhallintajärjestelmät tarvitsevat kolme toiminnan kannalta tarpeellista lisätoimintoa, clone, push ja pull.

**Clone** kopio/kloonaa pääpalvelimen versionhallinnan omalle levylle sisältöineen.

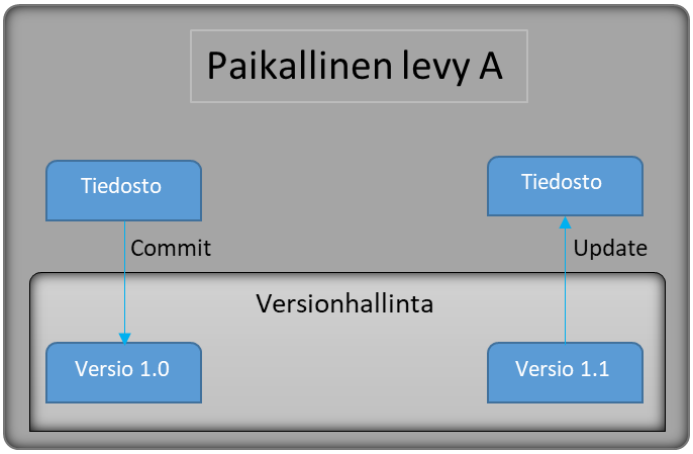
**Push** toimii samankaltaisesti kuten aiemmin mainittu kommit, mutta työntää omalla levyllä olevan versionhallinnan säilytyspaikan muutokset pääpalvelimelle.

**Pull** taas toimii samankaltaisesti kuten aiemmin mainittu update, mutta synkronoi pääpalvelimella olevat muutokset omaan paikalliseen versionhallintaan.
(Sink 2011, 5–14; 45–47.)

### 4.5.2 Versionhallintajärjestelmien sukupolvet

Versionhallintajärjestelmät voidaan jakaa kolmeen eri sukupolveen, joiden pääperiaate on samankaltainen. Suurimmat erot eri järjestelmissä ovat rakenne, jotkin komennot, ohjelmalliset toiminnot ja tiedostojen käsittelytavat. Sukupolvet jakautuvat järjestelmien suurien muutoksien merkkipaaluilla, jolloin niihin on lisätty uusi elementti tai toimintamallia on muutettu.

1. Sukupolvi on ”vanhanaikainen” **paikallinen** versionhallintajärjestelmä. Järjestelmä, johon voidaan viedä muutoksia yksittäisinä tiedostoina, joka ei ole millään tavoin verkossa. Tiedoston muokkaus ja julkaisu tapahtuu paikallisesti yhdellä asemalla. Tiedosto menee muokatessaan lukkoon, jolloin sitä ei voi käyttää muualla samanaikaisesti, katso kuva 23. Esimerkkinä SCCS, Source Code Control System. (Raymond 2008; Sink 2011, 1.)

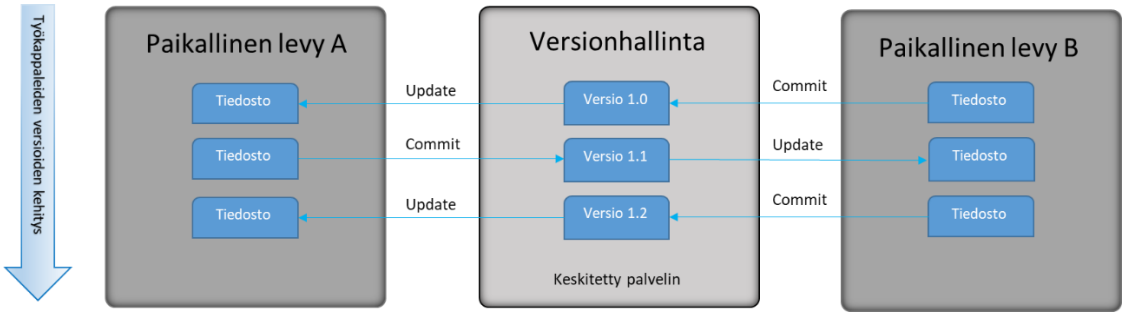



Kuva 23. Paikallinen versionhallintajärjestelmä.

Paikallinen versionhallinta sisältää tiedostojen versioinnin lisäksi niiden kommentoinnin ja aikaleimat. Ensimmäiset versionhallintajärjestelmät eivät olleet verkossa ja työkappaleiden uusien versioiden jako kehittäjien välillä tapahtui esimerkiksi levykkeiden välityksellä.

Paikallisten versionhallintajärjestelmien suurimmat ongelmat olivat verkottomuus ja tiedostojen lukittautuminen sekä se että, pelkästään yhden tiedoston vieminen kerralla järjestelmään oli mahdollista. (Raymond 2008; Sink 2011, 1.)

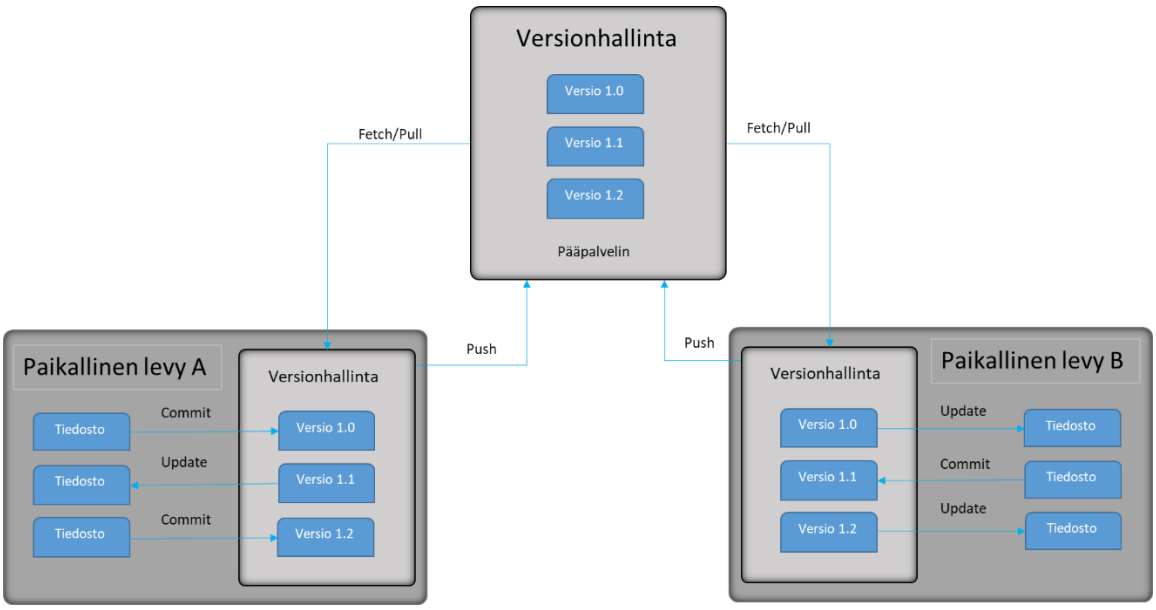
2. Sukupolvi, **keskitetyn palvelimen** versionhallintajärjestelmä, joka on verkossa. Useimmissa järjestelmissä voidaan viedä monia muutoksia kerralla. Tiedoston muokkaus ja julkaisu samanaikaisesti useammalla asemalla on mahdollista. Konfliktien välttämiseksi ennen version julkaisua voidaan versionhallinnasta tuoda omalle levylle kyseisen työkappaleen viimeisin versio, jotta tehdyt muutokset voidaan yhdistää ja täten luoda uusi yhdistetty versio, katso kuva 24. Esimerkiksi Subversion, SVN. (Sink 2011, 1; Raymond, 2008.)




Kuva 24. Keskitetyn palvelimen versionhallintajärjestelmä.

Erona aiempaan paikalliseen järjestelmään keskitetty järjestelmä esitti versionhallinnalle tärkeän elementin, verkostoitumisen. Verkossa oleva palvelin helpottaa käyttäjien välistä työkappaleiden siirtoa ja selkeyttää toisten ajankohtaisia kehittämiskohteiden tunnistamista. (Raymond 2008; Sink 2011, 1.)

3. Sukupolvi, **hajautetun palvelimen** versionhallintajärjestelmä, joka on verkossa. Muutoksia ja julkaisuja voidaan tehdä samanaikaisesti eri asemilla. Jokaisella käyttäjällä on oma kopio pääpalvelimen versionhallinnasta, josta päivitetyt versiot viedään pääpalvelimelle. Muutokset viedään järjestelmään yksittäisinä muutoksina tai joukkona. Konfliktien välttämiseksi pääpalvelimella käytetään samankaltaisia käytäntöjä kuin keskitetyissä järjestelmissä. Ainoana erona on, että uudempi versio tuodaan omaan paikalliseen versionhallintaan. Eri haarat tarvittaessa integroidaan yhdeksi ennen version pääpalvelimelle viemistä, katso kuva 25. Esimerkiksi Git versionhallinta. (Raymond 2008; Sink 2011, 1.)




Kuva 25. Hajautetun palvelimen versionhallintajärjestelmä.

Suurin ero hajautetun ja keskitetyn versionhallintajärjestelmän välillä on se, että jokaisella käyttäjällä on oma erillinen kopio pääpalvelimen versionhallinnasta sisältöineen. Toimiakseen, hajautettu järjestelmä tarvitsee kolme uutta komentoa, copy, pull/fetch ja push. Käyttääkseen järjestelmää, täytyy kehittäjän kopioida ensin pääpalvelimen versionhallinta omalle paikalliselle asemalle. Työkappaleiden muutokset tehdään tälle paikalliselle versionhallinnalle, jonka jälkeen ne ”työnnetään” pääpalvelimelle. Muutoksien tuominen pääpalvelimelta tapahtuu pull/fetch toiminnolla, joka tuo päivitetyt työkappaleet oman aseman versionhallinnalle.

Yksi suuri etu hajautetuissa järjestelmissä on sen käytön mahdollisuus ilman internet-yhteyttä. Käyttäjä voi jatkaa normaalia versiointia offline-tilassa ja myöhemmin, kun yhteys on saatavilla, viedä muutokset pääpalvelimelle. Hajautetun järjestelmän huonoina puolena ovat binääritiedostojen käsittelykyvyt ja pitkään jatkuneiden kehityskohteiden tapauksissa arkiston massiivinen koko, joka vie huomattavasti tilaa omalta levyltä. (Raymond, 2008; Sink 2011, 4–60.)

Yleisimmät käytössä olevat versionhallintajärjestelmät ovat Apache Subversion ja Git. Yllä mainittujen 2. ja 3. sukupolven järjestelmien erojen lisäksi niiden peruskomennot eroavat hieman toiminnaltaan, joten samat termit voivat tarkoittaa hieman eri tavalla toimivaa toimintoa (Git, ei pmv; TortoiseSVN, ei pvm). Kohteiden uudelleennimeäminen ja siirtäminen voi tuottaa hankaluuksia esimerkiksi SVN:ssä. Järjestelmä luulee, että uusi tiedosto tai polku on lisätty, joten tiedosto täytyy erikseen linkittää aiempaan kohteeseen saadakseen oikean versiohistorian (TortoiseSVN, ei pvm). Git toisaalta hoitaa uudelleennimeämiset useimmissa tapauksissa ongelmitta. Tämä siksi, koska Git seuraa tiedoston sisältöä pikemminkin kuin pelkästään tiedostonimeä. (Git, ei pvm.) 3. sukupolven kanssa perustoimintojen suoritusajat ovat luonnollisesti nopeampia, sillä niiden täytyy päästä käsiksi pelkästään paikalliseen versionhallintaan palvelimen sijasta.

Yleisimmät versionhallintajärjestelmät vievät muutokset ”atomisina”, tarkoittaen sitä, että jos kappaleeseen halutaan tehdä muutoksia, ne viedään järjestelmään joko kokonaan, tai ei ollenkaan. Esimerkiksi muutosjoukkoa vietäessä versionhallintaan kyseisen joukon jotkin tiedostot tai tiedoston osat menevät läpi ja muut epäonnistuvat, mitään muutoksia ei viedä arkistoon. (Raymond 2008; Collins-Sussman ym. 2011, 7.)

### 4.5.3 Subversion

Apache Subversion eli SVN, on vuonna 2000 julkaistu avoimen lähdekoodin **keskitetty** versionhallintajärjestelmä. Subversion kehitettiin korvaamaan aiempi, myös keskitettyyn palvelimeen kohdistettu versionhallinta CVS (Concurrent Versions System) (Collins-Sussman ym. 2011, xiii).

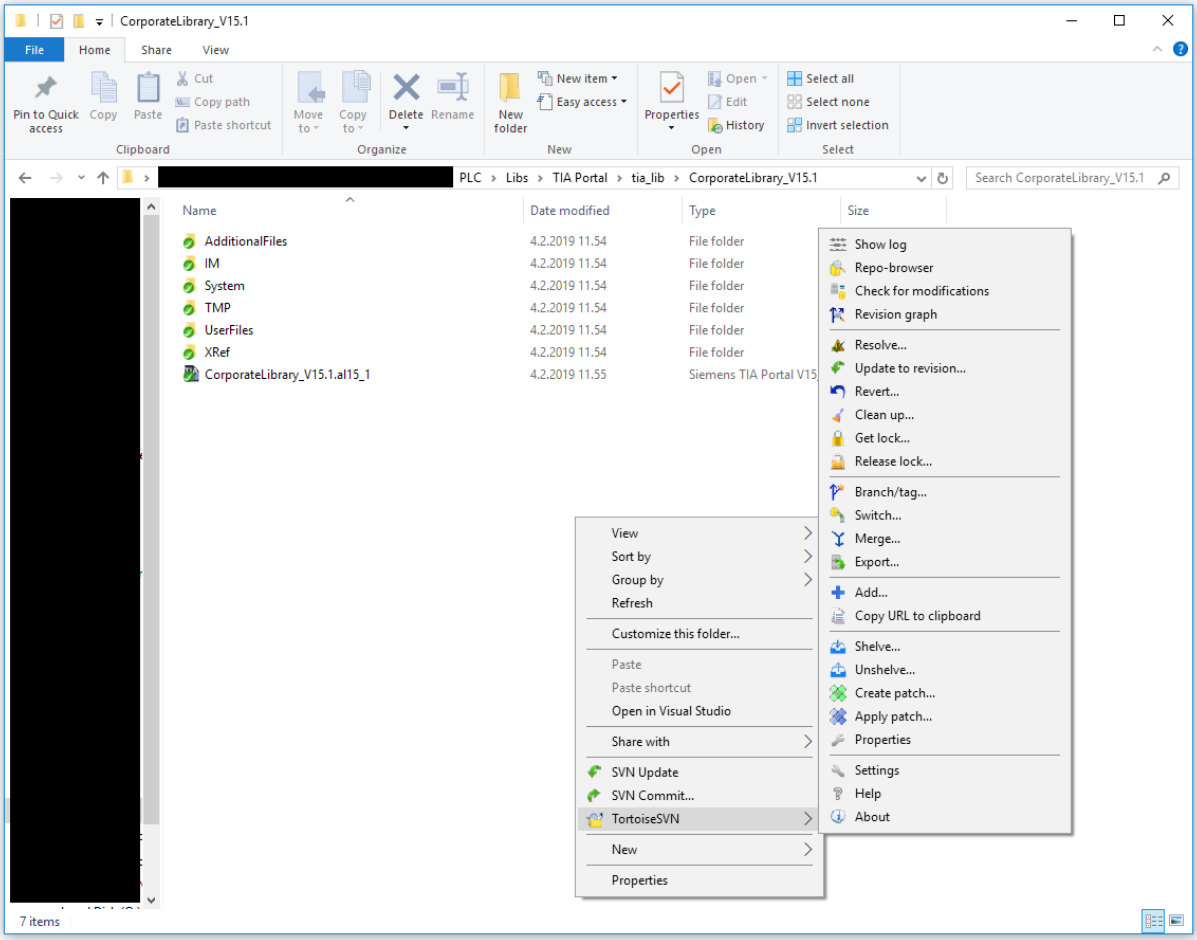
SVN oli ensimmäisiä versionhallintoja, joka hyödynsi tiedostojoukkoja. Tiedostojoukot, ”filesets”, tarkoittavat useamman tiedoston sisältävää joukkoa, joka käsitellään yhtenä ja sen voi kerralla kommitoida arkistoon (Raymond, 2008). Tällä tavoin toisista riippuvien työkappaleiden versiointi helpottui ja siirto paikallisen levyn ja versionhallinnan välillä nopeutui ja yksinkertaistui. SVN:ää voidaan käyttää joko komentoriviltä tai valitun SVN-sovelluksen käyttöliittymästä käyttäen samoja toimintoja. Yleisesti ottaen perustoimintojen eteen laitetaan ”svn” ja toiminnon jälkeen kohde. Esimerkiksi muutoksien vieminen versionhallintaan tapahtuu komennolla ”svn commit [Tiedoston polku]” (Collins-Sussman ym. 2011).

SVN:ssä muutokset viedään versionhallintaan atomisina ja revisionumerot ovat jatkuvia koko järjestelmässä, ei pelkästään yksissä tiedostoissa. Tämä tarkoittaa sitä, että jos muokataan tiedostoa, jolla on revisionumero ”12”, ja viedään järjestelmään, seuraava revisionumero ei ole välttämättä ”13”. Tämä johtuu siitä että, joku toinen on vienyt muutoksia versionhallintaan muokkauksen aikana. (Collins-Sussman ym. 2011; Raymond, 2008.)

Käyttäjät voivat hyödyntää SVN:n rajapintaa luodakseen omaan käyttöön sopivia sovelluksia ja liitännäisiä (plugin) työskentelyn sujuvoittamisen vuoksi. Rajapintaa ei ole rajoitettu pelkästään C-kielellä kirjoitettuihin sovelluksiin, vaan sitä voidaan hyödyntää myös eri kielillä esimerkiksi Pythonilla tai Javalla. (Collins-Sussman ym. 2011, 232.)

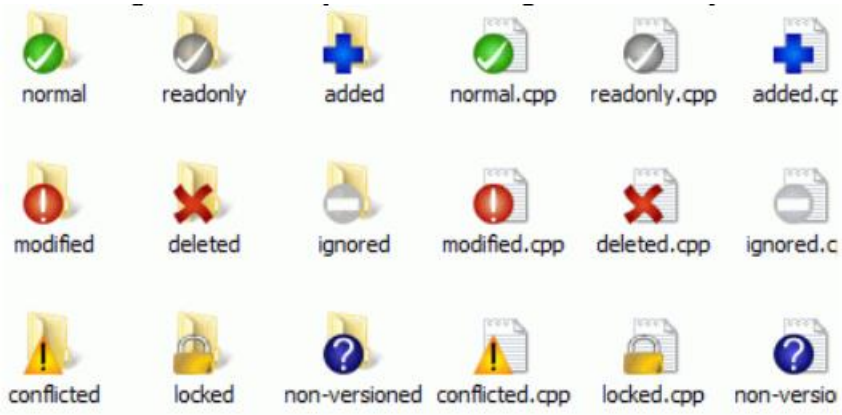
### 4.5.4 TortoiseSVN

TortoiseSVN on vuonna 2002 julkaistu Apache Subversion -versionhallintaohjelmisto, joka sisältää myös Windows shell -laajennuksen, katso kuva 26. Sovellus tarjoaa keskitetyn palvelimen versionhallintajärjestelmään hyvät työkalut tiedostojen kanssa työskentelyyn sekä niiden seuraamiseen. Shell-laajennuksen ansiosta pääsy versionhallintakomentoihin on vaivatonta ja käyttöliittymästä saadaan helposti selville kohteiden tarkemmat tiedot, kuten esimerkiksi revisionumero, versiohistoria kommenteineen ja muokkaajat. (TortoiseSVN, ei pvm.)




Kuva 26. TortoiseSVN:n valinnat resurssienhallinnassa.

Tiedostojen ikoneissa olevat versionhallinnan tilatietojen indikoinnit helpottavat tiedoston tilan ymmärtämistä ilman erikoisempia toimenpiteitä, katso kuva 27 (TortoiseSVN, ei pvm).




Kuva 27. TortoiseSVN versionhallinnassa olevien kohteiden ikonit (TortoiseSVN, 2018).

## 4.6 Automaatiopalvelin

Automaatiopalvelimilla tarkoitetaan yleensä jatkuvassa integraatioprosessissa käytettyjä palvelimia, joiden sisällä lähdekoodi muunnetaan konekielelle eli binäärinumeroiksi. Lisäksi koodi mahdollisesti testataan sille määritetyllä tavalla ja testien onnistuttua, työkappale julkaistaan päälinjaan. Yleensä palvelimien arkistona on versionhallinta, jonka sisäisiä työkappaleita testataan käskystä, jokaisen kommitin jälkeen tai aika ajoin, kuten esimerkiksi yöllä tiettyyn kellonaikaan.

### 4.6.1 Jenkins

Jenkins on Java-pohjainen avoimen lähdekoodin automaatio-/integraatiopalvelin, joka julkaistiin Hudsonin haarautumisen/nimenvaihdon jälkeen vuonna 2011. Integraatiotyökalu Hudson julkaistiin alun perin vuonna 2005. Hudsonin sisäisten ongelmien takia avainkehittäjät järjestivät äänestyksen Hudson-projektin nimeämisestä/haarautumisesta Jenkinsiksi. (Bayer, 2011.) Haarautumisen jälkeen Hudsonia kehitettiin vuoteen 2016 saakka (Hudson, ei pvm).

Jenkinsiin saatavat erilliset liitännät tarjoavat mahdollisuuden muokata ja laajentaa sen tarjoamia toimintoja kehittäjien mukaisiin tarpeisiin kuten versionhallinnan suhteen. Jenkinsin versionhallintaliitännöiksi sopivat useat eri järjestelmät kuten SVN, Git ja CVS. (Jenkins, ei pvm.)

Huomatessaan arkistossa muutoksen, määritetyin aikavälein tai käskyn saadessaan Jenkins käynnistää kohteelle määritetyt tehtävät, joita seurataan ja tuloksista raportoidaan käyttäjille. Jenkinsin käyttöliittymässä näkyy tehtävien lopputulokset, katso kuva 28 ja kohteiden statukset, katso kuva 29. (Jenkins, ei pvm.)

Jobs statistics

| Job health | Description | Number of jobs with this health status |
|---|---|---|
| | No recent builds failed | 3 |
| | 20-40% of recent builds failed | 0 |
| | 40-60% of recent builds failed | 0 |
| | 60-80% of recent builds failed | 0 |
| | All recent builds failed | 0 |
| | Unknown status | 0 |
| **Total jobs** | All jobs | 3 |

Kuva 28. Tehtävien suoriutumisen indikoinnit (Jenkins, 2018).

**Build statistics**

| Status of the build | Description | Number of builds with status |
|---|---|---|
| | Failed | 18 |
| | Unstable | 1 |
| | Success | 92 |
| | Pending | 0 |
| | Disabled | 0 |
| | Aborted | 3 |
| **Total builds** | All builds | 114 |

Kuva 29. Kohteiden buildin tilatiedot (Jenkins, 2018).

# 5 JATKUVA INTEGRAATIO

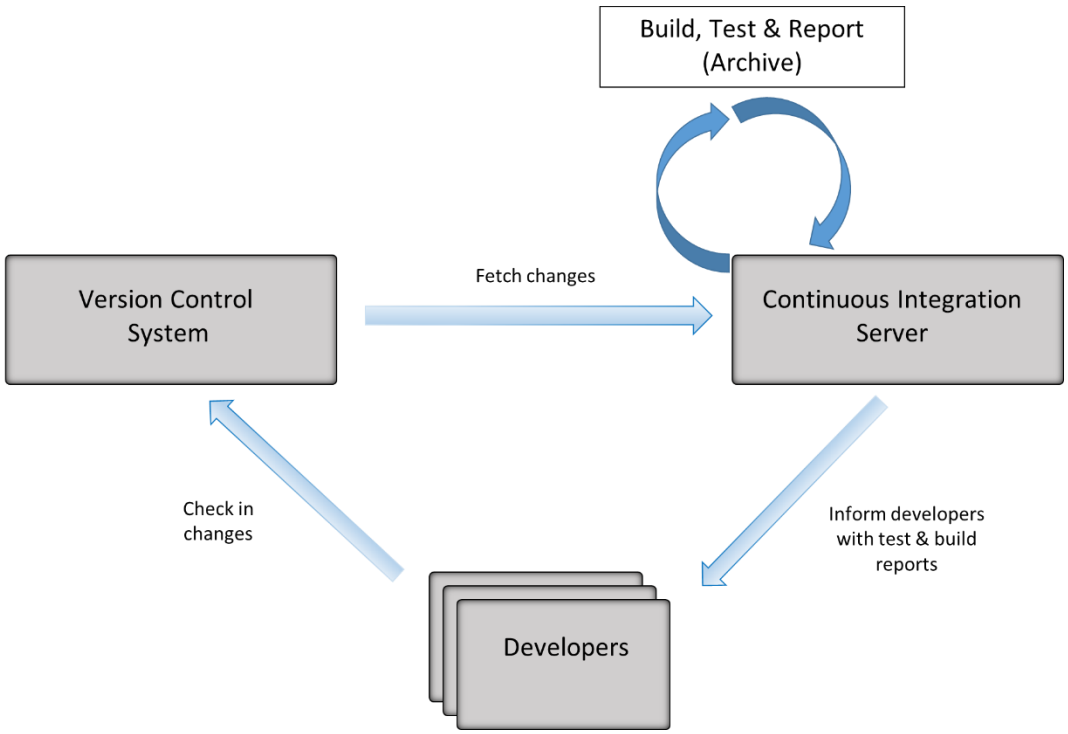
”Jatkuva integraatio on ohjelmistokehityksessä käytetty käytäntö, jossa kehittäjät liittävät/integroivat työkappaleet useasti projektin integraatiohaaraan ja luovat buildin”. (Pathania 2016, 13.)

Ohjelmistokehitykseen liittyy usein samojen toimenpiteiden toistaminen jatkuvasti. Näihin kuuluvat sovelluksien buildaaminen, kääntäminen ja työkappaletyyppien mukaisten testien suorittaminen ja niistä raportoiminen. Buildaamisella tarkoitetaan lähdekoodin syntaksin tarkistamista, lähdekoodin koostamista ja pakkaamista tietokoneelle ymmärrettävään binäärimuotoon.

”Jatkuva integrointi on välttämätöntä, jotta voidaan tuoda esiin integraation aikana ilmenneet ongelmat mahdollisimman nopeasti”. (Pathania 2016, 13.)

Integraatioprosessia hyödynnetään edellä mainittujen toimenpiteiden automaattisessa suorittamisessa ja työkappaleiden yhdistämisessä toimivaksi kokonaisuudeksi. Tällä prosessilla varmistetaan tehtyjen muutoksien oikeaoppinen toimivuus, integraatiovirheet ja vältetään suurin osa useista muutoksista johtuva integraatiokaaos.

Tyypillinen jatkuva integraatioprosessi vaatii tiedostojen keskityksen, automaatiopalvelimen ja määritetyt tehtävät työkappaleille testipalvelimilla, katso kuva 30.




Kuva 30. Tyypillinen jatkuva integraatioprosessi.

Kuvan 30 mukaisessa integraatioprosessissa tiedostot on keskitetty versionhallintaan, josta automaatiopalvelin seuraa työkappaleiden muutoksia. Havaitessaan muutoksen, palvelin aloittaa työkappaleelle määritellyt toimenpiteet, kuten lähdekoodin buildaamisen ja onnistuttuaan jatkaa automaattitesteillä. Näiden suorittamisen jälkeen luodut artefaktit arkistoidaan ja tuloksista raportoidaan automaattisesti kehittäjille. Artefakti on testattu, toimiva, valmis sovellus.

”Jatkuva integraatio ei suoranaisesti poista virheitä, vaan helpottaa niiden löytämistä ja eliminointia.” (Fowler 2006.)

## 5.1 Historia

Termi ”jatkuva integraatio” perustuu Grady Boochin vuonna 1991 kehittämään Boochin menetelmään, joka on olio-ohjelmoinnin suunnittelu- ja analysointitapa (Booch 1991, 209). Boochin menetelmässä ohjelmistoa kehittäessä ja päivittäessä tavoitteena on iterattivinen ja inkrementaalinen elämänkaari hyödyntäen nk. mikro- ja makroprosesseja. Mikroprosessi tarkoittaa ohjelmistoon tehtäviä pieniä muutoksia, ja makroprosessi taas suurempia muutoksia, jotka toimivat aiemmin mainitun viitekehyksenä. (Booch 1994, 227–262.) Menetelmässä esiintyy jatkuvan integraation piirteitä kuten muutoksien toistuva päivittäminen päälinjaan, joskin ei niin tiheästi kuin nykyään integraatioprossessissa on tapana.

Vuonna 1997 Kent Beck & Ron Jeffries keksivät ohjelmistokehitysmetodin Extreme Programming (XP), jossa yksi käytäntö oli soveltaa jatkuvaa integraatiota siten, että työkappaleet integroitiin päälinjaan vähintään kerran päivässä, mielellään useammin suosien pieniä muutoksia. (Wells, 1997; Jeffries, 2011.)

Ensimmäinen integraatiotyökalu CruiseControl julkaistiin vuonna 2001. Kyseisellä työkalulla pystyi suorittamaan automaattisesti lähdekoodin koostamisen, artefaktin luomisen ja raportoinnin. Työkalussa oli myös käyttöliittymä eli nk. dashboard, josta kehittäjät pystyivät seuraamaan työkappaleiden tehtäviä ja tuloksia. (CruiceControl, ei pvm.)

## 5.2 Yleisimmät käytänteet

Fowlerin (2006) mukaan tehokkaan jatkuvan integraationprosessin saavuttamiseksi täytyy soveltaa seuraavanlaisia avainkäytäntöjä:

”**Tiedostojen keskitys.** Ohjelmistoprojekteissa on paljon tiedostoja, jotka täytyy organisoida/järjestää yhteen kootakseen itse tuotteen. Kaikkien muutoksien ja osien jäljittäminen vaatii suuren ponnistelun varsinkin, kun kyseessä on useampi samanaikainen kehittäjä.” (Fowler 2006.)

Versionhallintajärjestelmän arkisto on mainio tapa keskittää tiedostot jatkuvalle integraatioprosessille. Versionhallinnan tuomien etujen vuoksi työkohteet pääsevät samalla integraatioprosessin kierteeseen. Automaatiopalvelin tuo testattujen kappaleiden artefaktit kaikkien kehittäjien saataville esimerkiksi Jenkinsin web-käyttöliittymän kautta.

**Automaattinen buildaaminen ja testaus.** ”Lähdekoodien koostaminen toimivaksi kokonaisuudeksi voi usein olla monimutkainen prosessi johtuen tiedostojen siirtelystä, koostamisesta ja muista tarvittavista toimenpiteistä, mutta kuten monet tehtävät ohjelmistokehityksessä, ne voidaan automatisoida – ja niin pitäisikin.” (Fowler 2006.)

”Perinteisesti ohjelmiston buildaaminen toimivaksi kokonaisuudeksi tarkoittaa osien koostamista, linkittämistä ja muita tarvittavia toimenpiteitä. Ohjelmisto saattaa toimia, mutta siinä voi olla useita bugeja, eli virheitä. Hyvä tapa napata nämä ovat lisätä prosessiin ohjelmiston automaattinen testaus.” (Fowler 2006.)

Automaattinen sovelluksen buildaaminen ja testaus nopeuttaa sovelluksien kehitysprosessia. Kaikki automaattisesti tehdyt toistuvat toimenpiteet kuten buildaaminen, testaus ja niiden tuloksien raportointi säästävät aikaa kehittäjiltä.

”**Päivittäiset kommitit ja siitä johtuvat integroinnit**. Kehittäjän päälinjaan kohdistuvan kommitin edellytyksenä on työkappaleen buildin ja testien läpäisy. Tekemällä näin, kehittäjien tekemien muutoksien konfliktit havaitaan ja korjataan hyvissä ajoin. Konfliktit, jotka ovat olleet huomaamattomia jopa viikkoja, on hankala selvittää.” (Fowler 2006.)

Muutosjoukkojen pitäminen pienenä edesauttaa virheiden minimoimista ja niiden identifioimista sekä nopeiden jatkotoimenpiteiden suorittamista. Joskus virheet täytyy kehittäjän selvittää manuaalisesti ja pienen muutosjoukon virheen kartoittaminen on huomattavasti helpompaa kuin massiivisen.

”**Virheellisten buildien korjaus.** Tärkeä elementti integraatioprosessissa on päälinjan nopea korjaus, jos buildaaminen tai testaus epäonnistuu. Koko prosessin tarkoitus on, että kehitys ja päälinja pysyvät vakaassa tilassa.” (Fowler 2006.)

Epäkunnossa oleva päälinja on korjattava mahdollisimman nopeasti, jotta sen oikeaoppinen kehitys voi jatkua ongelmitta. Päälinjan rikkoutuminen korjataan usein palauttamalla viimeisin toimiva versio.

”**Nopeus ohjelmiston buildauksessa ja testeissä.** Jatkuvan integraation tarkoitus on myös tarjota nopeaa palautetta. Useille projekteille määritetyt ohjelmiston maksimi buildiajan linjaus on XP:stä tuttu 10 minuuttia.” (Fowler 2006.)

”**Testit suoritetaan tuotantoympäristön kopiossa/kloonissa/mallinnuksessa.** Testausten tarkoitus on poistaa mahdolliset ongelmat hallituissa olosuhteissa, jotka voisivat tulla esiin tuotannossa. Jokainen havaittu ja korjattu ongelma testiympäristössä, on poissa tuotannosta.” (Fowler 2006.)

”**Viimeisimmän version helppo saatavuus kehittäjille.** Yksi haaste ohjelmistokehityksessä on oikeanlaisen sovelluksen rakentaminen. Tämän helpottamiseksi jokaisen, joka on osana kyseisen ohjelman kehityksessä, pitäisi löytää sovelluksen viimeisin versio ja pystyä ajamaan sitä.” (Fowler 2006.)

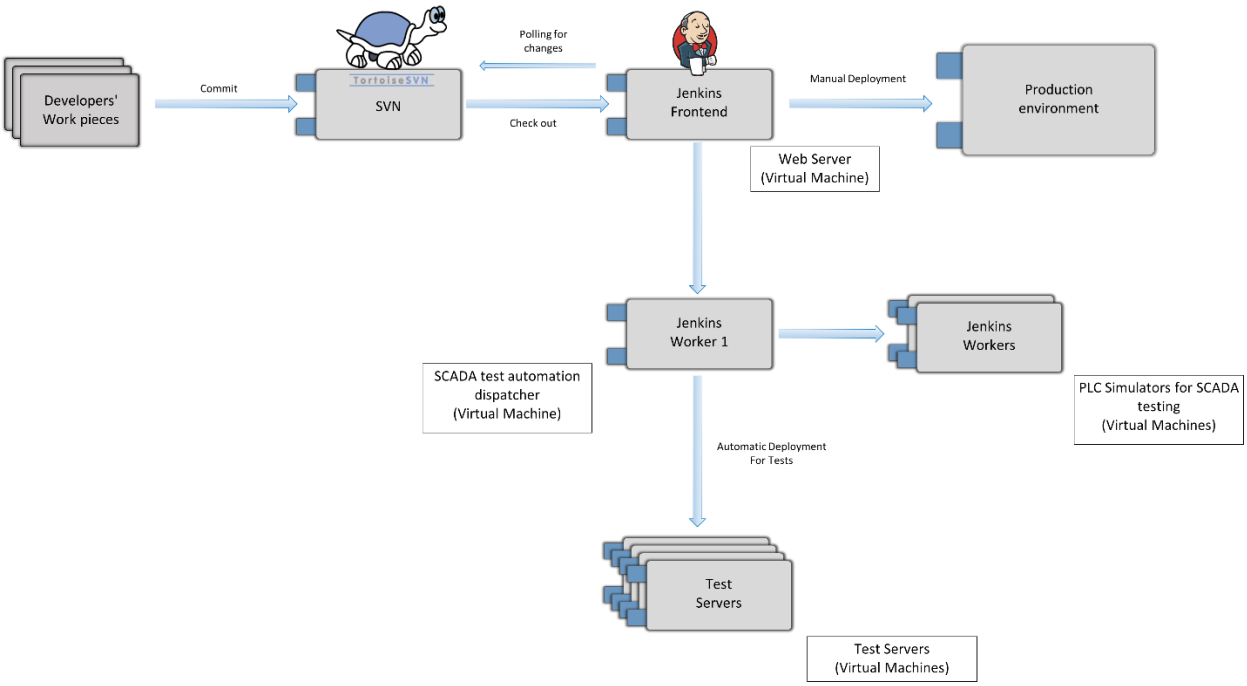
Tällä Fowler luultavasti tarkoittaa sitä, että kaikilla kehittäjillä on sovelluksen viimeisin versio ja he pystyvät sitä kokeilemaan. Kokeilemalla viimeisintä buildia, kehittäjät voivat helposti identifioida mitä työkappaleita ja muutoksia heidän täytyy tehdä, jotta kokonaisuus on toimiva.

”**Selkeys päälinjan tapahtumille ja kehittäjien välillä.**
Integraatioprosessin tärkeänä elementtinä on kommunikoiminen. Selkeys kehittäjien välillä ja päälinjan tilalla auttaa kehittäjät nopeasti ymmärtämään tehdyt muutokset sekä sen, mitä muut kehittäjät ovat sillä hetkellä työstämässä.” (Fowler 2006.)

”**Automaattinen käyttöönotto.** Jatkuva integraatioprosessi vaatii useampia ympäristöjä/työkaluja. Yksi ympäristö suorittaa koostamisen, toinen testit jne. Näiden välillä työkappaleiden siirto tapahtuu jatkuvasti, joten siirto ja sijoitus eri ympäristöihin kannattaa automatisoida.” (Fowler 2006.)

# 6 YSP OY:N OHJELMISTOKEHITYSPROSESSI

YSP:n ohjelmistokehitysprosessissa sovelletaan jatkuvaa integraatioprosessia hyödyntäen Apache Subversion -versionhallintaa ja Jenkins-automaatiopalvelinta. Jenkinsiin integroidut liitännät tekevät eri työkappaletyypeille määritetyt tehtävät kuten buildaukset, automaattitestit ja jatkotoimenpiteet mahdollisiksi. Jenkinsin pääpalvelimella sijaitsee web-käyttöliittymä, jonka kautta kehittäjät voivat muokata ja tarkastella projekteihin liittyviä yksityiskohtia. Toteutuksessa on hyödynnetty Jenkinsin hajautettavuutta sijoittamalla "työntekijät" eri virtuaalikoneille, jotka tekevät työkappaleiden koostamiset ja buildaamiset sekä joissain tapauksissa automaattitestit, katso kuva 31. (YSP ohjelmistokehitys, ei pvm.)




Kuva 31. YSP Oy:n jatkuva integraatioprosessi ja -ympäristö.

Tiedostot ovat keskitetty Apache Subversion -versionhallintaan. Jenkins toimii automaatiopalvelimena ja testipalvelimina toimivat Jenkins workerit, joihin pääsee tarvittaessa käsiksi RemoteDesktopin avulla.

Kehittäjät kommittoivat uusimman version muutoksineen versionhallintaan. Jenkinsin työhakemisto tulee SVN:stä ja pollaa, eli kiertokyselee tehtyjä muutoksia kappaleisiin. Huomatessaan muutoksen, Jenkins aloittaa työkappaleen tyypille määritellyn tehtäväjonon. Tehtävien aloittaminen on myös joissain tapauksissa toteutettu ajastimella. Tehtäviin kuuluu tapauskohtaisesti esimerkiksi lähdekoodien koostaminen, buildaaminen, automaattitestaus ja muut mahdolliset toimenpiteet. Onnistuneiden koostamisten ja testien jälkeen määritetyistä työkappaleista luodaan artefaktit, jotka on mahdollista

ladata Jenkinsin web-käyttöliittymästä esimerkiksi zip-muodossa. Kehittäjät saavat buildien lopputuloksen raportin myös pikaviestintäsovellukseen. Tarkemmat buildin tiedot saadaan tarvittaessa Jenkinsin web-käyttöliittymästä, josta voidaan myös selata projektien aiempien buildien tuloksia, työhakemistoa, tehtäviää, muutoksia ja aiemmin mainittuja artefakteja. (YSP ohjelmistokehitys, ei pvm.)

# 7 SUUNNITTELU JA TOTEUTUS

## 7.1 Kirjastot

### 7.1.1 Kirjastotyypit ja niiden rakenne TIA Portalissa

**Project library** (Projektikirjasto) on projektin oma kirjasto, jota käytetään siinä paikallisessa projektissa, missä se on määritelty. Tähän kirjastoon ei voida erikseen tallettaa lokeja eikä kieliasetuksia, koska ne sijaitsevat projektipuussa.

**Global library** (Yleinen kirjasto) on julkinen kirjasto, jonka voi hakea ja avata TIA Portalissa levyltä. TIA Portal sisältää myös valmiita yleisiä kirjastoja, joissa erilaisia funktioita, dokumenttipohjia ja WinCC elementtejä.

**Corporate library** (Yrityskirjasto) on yleinen kirjasto, jonka TIA Portal hakee automaattisesti ennalta määritellystä polusta sovelluksen avautuessa. Yrityskirjaston käyttöönotto edellyttää TIA Portalin asetustiedoston (CorporateSettings.xml) muokkaamista sen ProgramData-kansiossa

Yleistä ja yrityskirjastoa muokatessa kirjasto voi olla auki pelkästään muokkaajalla. Niiden sisältö ei poikkea TIA Portalin sisällä päärakenteeltaan millään tavalla toisistaan, katso kuva 32.

Kuva 32. Kirjaston rakenne TIA Portalissa (Siemens AG TIA Portal V15.1, 2019).

Kuvassa 32 näkyvät automaattisesti generoidut pääkansiot ovat valmiina jokaisessa yritys- ja global kirjastossa.

**Types,**

TIA Portalin oma versionhallinta lohkoille ja käyttäjän määrittelemille datatyypeille (UDT), sekä muille kohteille joille versiointi on mahdollista, kuten esimerkiksi HMI-laitteen näytöille. Aiemmat versiot voidaan säilyttää tai vaihtoehtoisesti poistaa kirjastoista.

**Master Copies,**

Master copies eli ”alkuperäisversio” tai ”pääkopio” -kansion alle voidaan tallentaa lohkojen ja datatyyppien valmiiden versioiden lisäksi esimerkiksi kokonaisia määriteltyjä logiikkalaitteita sisältöineen.

**Common Data,**

Common data -kansio sisältää kirjaston muutoslokin. Lokiin kirjataan tiedot kirjaston muutoksista, kun sinne päivitetään uusi versio (Type) tai kirjasto siivotaan clean up -työkalulla. Esimerkiksi lohkon version päivittämisen jälkeen lokiin ilmestyy tiedot muutoksen ajankohdasta, sekä tiedot, että mitä lohkoa on muokattu, mitkä versiot ovat ajan tasalla ja mikä versio kirjastoon lisättiin.

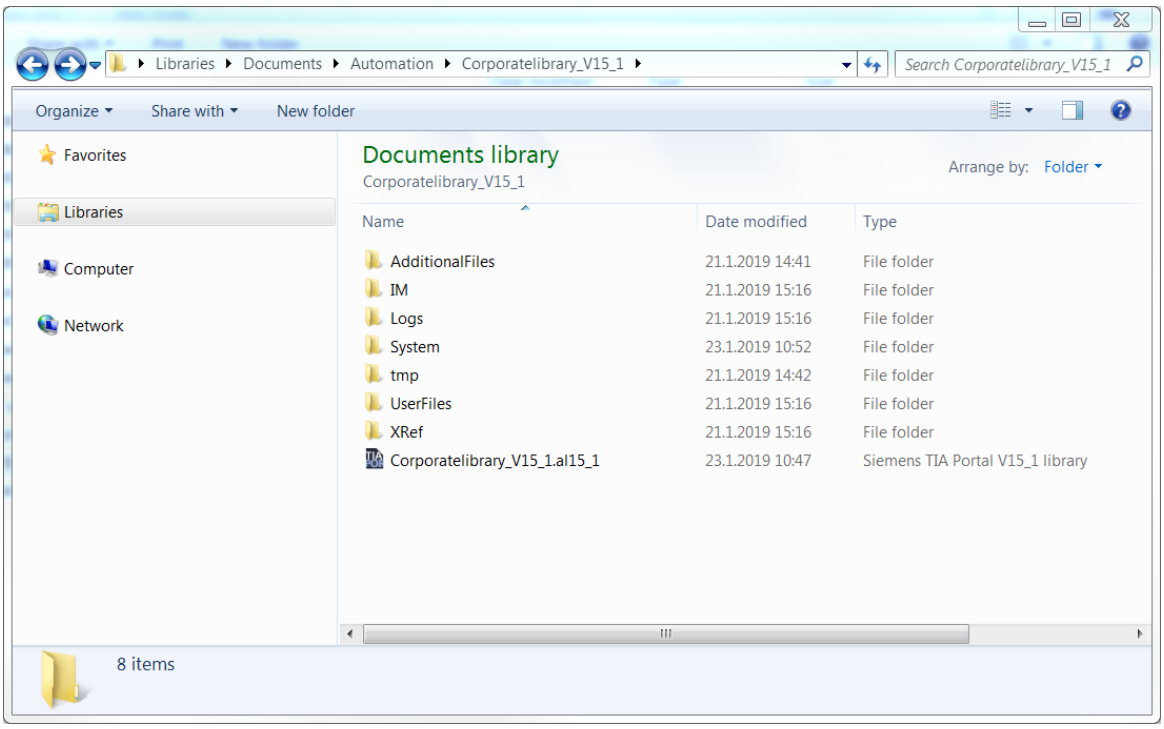
**Resources & Languages,**

Resurssit ja kielet -kansio sisältää kirjaston/projektin kieliasetukset, sen avattaessa voidaan muokata editointikieltä ja viitekieltä. Editointikieli kirjoittaa projektin tekstit valitun kielen alle.
Referenssikieltä käytetään tekstien kääntämisen ”alustana”, joka helpottaa esimerkiksi tekstien kääntämistä taulukkonäkymässä.

(Siemens AG TIA Portal V15.1, 2019)

### 7.1.2 Rakenne levyllä

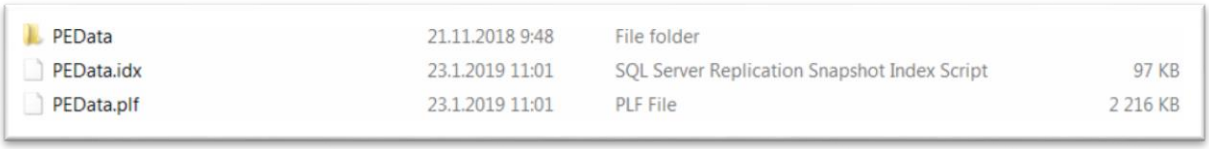
TIA Portalin yleisen ja yrityskirjaston rakenne levyllä on yksi kansio sisältöineen. Kirjaston avaukseen on erikseen referenssitiedosto, joka avataan TIA Portalilla tai levyltä. Referenssitiedoston tiedostopääte on al.***versio***. Esimerkiksi versiossa 15.1 luotu kirjasto on muodoltaan kirjasto.**al15_1** ja versiossa V14 taas **al14**. Tämä kirjastotiedosto ei sisällä kaikkea kirjastoon säilöttyä dataa, vaan se on viite TIA Portalille, katso kuva 33.




Kuva 33. Kirjaston rakenne levyllä.

Suurin osa kirjaston sisällöstä on **System-**kansion alla olevissa PEData-tiedostoissa, katso kuva 34. Kirjastoa muokatessa TIA Portalissa, PEData-tiedostot sekä kirjaston referenssitiedosto muuttuvat. Muokattaessa kirjastoa, kansioon syntyy infotiedosto, joka kertoo muokkaajan käyttäjänimen, tietokoneen nimen ja ajan.

PEData-tiedostojen sisältö on binäärimuodossa, joten niiden sisältöä ei saa vaivatta luettavaan muotoon TIA Portalin ulkopuolella.




Kuva 34. PEData-tiedostot levyllä.

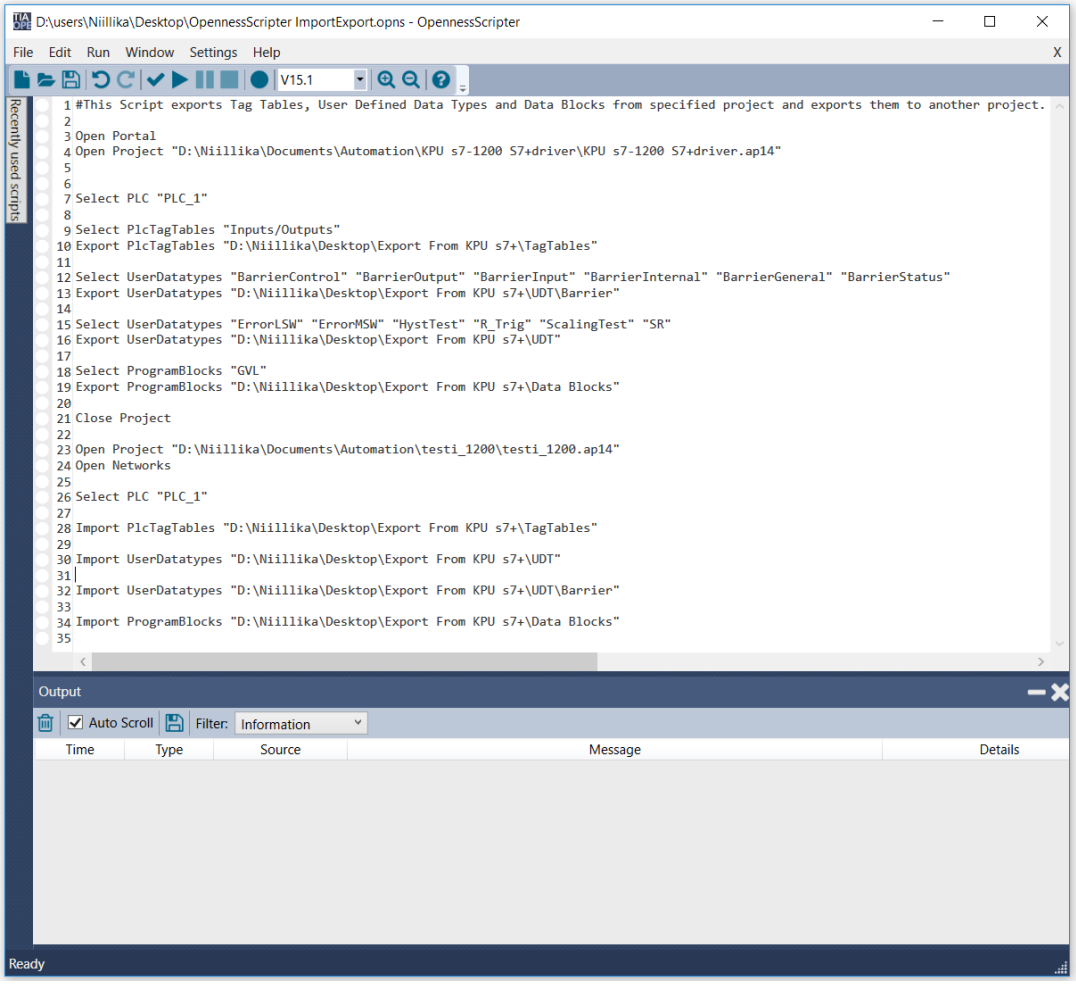
**UserFiles**-kansio sisältää käyttäjän omat tiedostot, esimerkiksi erikseen luotujen lohkojen help-tiedostot. Sinne voidaan myös tallentaa tiedostoja, joita halutaan siirtää kirjaston yhteydessä, kuten suunnitelmia.

**Logs**-kansio sisältää kirjaston muutoslokit zip-tiedostona.

### 7.1.3 Kirjastojen suunnittelu

Kirjastot versionhallintajärjestelmän kanssa olisi ollut mahdollista toteuttaa useammalla eri tavalla. Kaksi vaihtoehtoista tapaa TIA Portalin omien kirjastojen lisäksi olisi ollut hyödyntää export & import tai ”generate source from blocks” -toimintoja.

Export & import toimintoa käyttäessä täytyy hyödyntää Openness-rajapintaa. Export vie valitun lohkon paikalliselle levylle XML-muodossa. Import taas vie kyseisen lohkon TIA Portal -projektiin. Suunnittelun testausvaiheessa käytetty sovellus oli Siemensin oma Openness Scripter V14.1, jolla luotiin skripti, joka importtaa ja exporttaa lohkoja, datatyyppejä ja muuttujataulukkoja määritettyihin projekteihin katso kuva 35.




Kuva 35. Siemensin Openness Scripterillä luotu import/export ohjelma (Siemens AG Openness Scripter, 2019).

XML-muotoinen export-tiedosto ei ollut selkeästi luettavissa tekstieditorilla. Tiedoston selkokielisen koodin sisältämättömyyden takia tämä vaihtoehto poissuljettiin nopeasti.

TIA Portalin sisällä olevan ”generate source from blocks” -toiminnon avulla lohkon saa ulos muodossa, miltä se näyttää myös sovelluksen sisällä, eli se on luettavassa muodossa myös TIA Portalin ulkopuolella tekstieditorilla. Lohkot saadaan myös TIA-projektiin käyttämällä sen sisäistä ”external source files” -työkalua. Tämä tuo aiemmin ulos viedyt lohkot nykyiseen projektiin sisältöineen.

Aiemmin mainitut tavat helpottaisivat yksittäisten lohkojen jäljittämistä versionhallinnassa yksittäisten tiedostojen takia, mutta niiden käyttö ei ollut yhtä sujuvaa ja yhdenmukaista kuin TIA:n omia kirjastoja hyödyntäessä. Tästä syystä ja muutamasta yrityskirjastojen tuomista eduista päädyttiin hyödyntämään TIA:n omia yrityskirjastoja. Näihin etuihin kuuluu esimerkiksi muiden projektien osien kuin lohkojen vieminen kirjastoon (kappale 7.1.1), lohkojen ja datatyyppien versiointi ja kirjaston automaattinen avaus sovelluksen avautuessa.

Kirjastoon tullaan jatkossa viemään kaikki logiikkaohjelmoinnista syntyneet funktiolohkot, datalohkot, kehittäjien luomat datatyypit, muuttujat, tarvittavat tekstit kuten kieliasetukset ja valmiiksi konfiguroituja logiikkalaitteita.

Yksi negatiivinen puoli TIA:n omien kirjastojen käytössä oli se, että niiden sisältöä ei saatu helposti selville sovelluksen ulkopuolella. 7.1.1 kappaleessa mainittu yrityskirjaston muokkauksen yhtäaikaisen käytön rajoitus poistuu SVN-versiohallintaa käytettäessä, koska kaikilla kehittäjillä on oma paikallinen kopio tästä kirjastosta, joita he muokkaavat omilla työasemilla.
Tulevaisuudessa tullaan tarvittaessa hyödyntämään myös ”Generate source from blocks” -toimintoa, jotta voidaan hyödyntää TIA:n ulkopuolella lohkojen selkoluettavaa lähdekoodia eri järjestelmissä.

Yrityskirjastokansioon tullaan lisäksi tallettamaan kehittäjien luomien lohkojen help-tiedostot, jotka sisältävät tietoa lohkojen toiminnasta ja niiden käytöstä, jotta luotujen lohkojen dokumentointi on samassa paikassa kirjaston kanssa. Nämä tiedostot voidaan avata TIA Portalissa valitsemalla lohko ja painamalla näppäinyhdistelmää ”SHIFT+F1”. Tämä komento hakee määritetystä polusta lohkonnimeä vastaavan tiedoston (Siemens AG TIA Portal V15.1, 2019).

### 7.1.4 Kirjastojen toteutus ja vieminen versionhallintaan

Kirjastojen luominen TIA Portalissa oli yksinkertaista. Sovelluksen sisällä oikealla sijaitsevassa Global Libraries -välilehdellä valittiin ”create new library” jonka jälkeen kirjastolle annettiin nimi sekä polku ja tarvittaessa liitetään kirjastoa kuvaava kommentti.

Global-kirjaston muuntaminen yrityskirjastoksi vaati konfigurointitiedoston muokkaamista. Konfigurointitiedosto (CorporateSettings.xml) ProgramData-kansiossa (**C:\ProgramData\Siemens\Automation\Portal V15_1\CorporateSettings**).

Toimiakseen oikealla tavalla, asetustiedosto täytyi sisältää vähintään kuvassa 36 olevan tekstin:

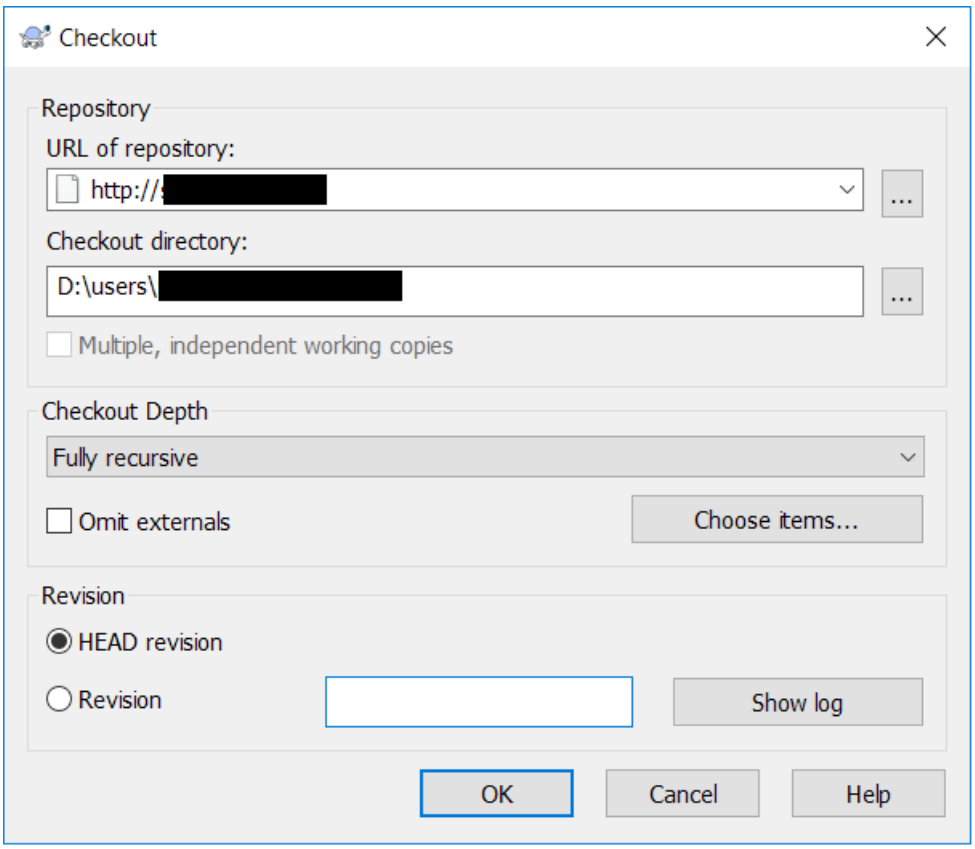
```
<?xml version="1.0" encoding="utf-8"?>
<Document>
 <Settings.Settings ID="0">
  <ObjectList>
   <Settings.General ID="1" AggregationName="General">
    <AttributeList>
     <CorporateLibraryPaths>
      <Item>KIRJASTON POLKU</Item>
      </CorporateLibraryPaths>
    </AttributeList>
   </Settings.General>
  </ObjectList>
 </Settings.Settings>
</Document>
```

Kuva 36. Yrityskirjaston asetustiedoston sisältö.

Kohdan ”KIRJASTON POLKU” tilalle tulee laittaa yrityskirjaston sijainti omalla koneella, eli tässä tapauksessa siihen laitettiin SVN:ssä olevan kirjaston sijainti. Kirjaston polkua muokattaessa TIA Portalin ollessa auki, saadaan ilmoitusikkuna, joka kertoo sen päivittämisestä. Jos tiedostoon lisätään uusia kirjastojen polkuja, ne tulevat näkyviin vasta TIA Portalin uudelleenkäynnistämisen jälkeen. Samaan tiedostoon voidaan myös liittää muita TIA Portalin asetuksia, kuten aiemmin mainitut lohkojen help--tiedostojen keskitetty haku versionhallinnasta. Koska tiedosto täytyy jokaisella käyttäjällä muokata oman kansiopolun mukaiseksi, päätettiin luoda skripti, jolla saadaan generoitua kyseinen tiedosto automaattisesti. Powershell-skripti sijoitettiin TIA Portal -kirjaston kansion sisälle, jotta se löytää yrityskirjaston referenssitiedoston samasta paikasta. Tämä kyseinen tiedostopolku kirjoitetaan asetustiedostoon pienin muokkauksineen. Ajettaessa, skripti luo asetustiedoston Programdatan alla olevaan kansioon ja muokkaa sen sisällön siten, että polut generoituvat oikealla tavalla tietokoneesta riippumatta. Skripti kirjoittaa tiedoston sisälle myös help-tiedostojen keskitetyn haun konfiguroinnin.

Kirjaston vieminen versionhallintaan oli yksinkertaista ja se tapahtui muutamalla SVN komennolla. Ensin täytyi tuoda versionhallinnan tiedostot paikalliselle levylle, jonka jälkeen päivitys oli mahdollista. Kyseiset toimenpiteet suoritettiin TortoiseSVN käyttöliittymän avulla sen käytettävyyden ja sujuvuuden takia. Samaiset komennot olisi ollut mahdollista suorittaa vaihtoehtoisesti komentoriviltä.

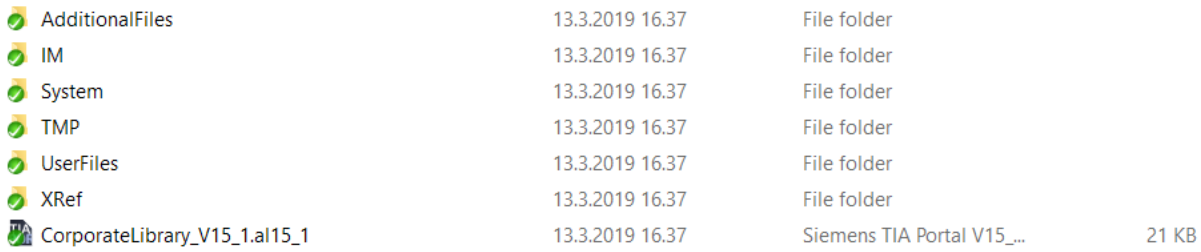
Komento svn **checkout** loi valittuun polkuun työkappaleet valituista versionhallinnan arkistosta kansioista ja tiedostoista, katso kuva 37 (TortoiseSVN, ei pvm).




Kuva 37. SVN Checkout TortoiseSVN käyttöliittymässä (TortoiseSVN, ei pvm).

Tämän jälkeen luotiin kirjastolle haluttu polku resurssienhallinnassa ja vietiin TIA Portalissa luotu kirjastokansio tiedostoineen kohteeseen.

Komento svn **commit** vie muutokset versionhallinnan arkistoon muutoksineen, katso kuva 38. Kommitoidessa, kehittäjän täytyy valita mitä tiedostoja halutaan viedä palvelimen arkistoon. (TortoiseSVN, ei pvm) Tässä tapauksessa valittiin luonnollisesti kaikki kirjastoon liittyvät tiedostot ja liitettiin asianmukainen kommentti.




Kuva 38. Kirjasto sisältöineen ajan tasalla versionhallinnassa.

## 7.2 PLC-sovelluksen automaattinen lataaminen simulaattorille

Integraatioympäristön simulaattoriksi valittiin PLCSIM Advanced V2.0 pääosin sen erillisen rajapinnan ja TCP/IP -kommunikoinnin takia. Advanced tarjoaa useita ominaisuuksia mitä perusversio PLCSIM ei sisällä (kts. kuva 8). Tulevaisuudessa myös kompleksien automaattitestien määrittäminen integraatiopalvelimella C#-sovelluksen kautta on mahdollista.
Automaatioympäristössä tapahtuva automaattinen logiikkasovelluksen lataus simulaattorille vaati Opennessin ja simulaattorin rajapintojen hyödyntämistä.
Ensin päädyttiin testaamaan taas Openness Scripter-sovelluksen potentiaalia kyseisessä tehtävässä. Sovelluksella luodaan suoraviivaista skriptikieltä, jotka tekevät komentoja vastaavia toimenpiteitä projektille. Hyvinkin nopeasti kävi ilmi, että sovelluksella ei voi tehdä kuin yksinkertaisimpia tehtäviä kuten esimerkiksi TIA Portalin ja määritetyn projektin avaus, lohkojen import- ja export toiminnot sekä projektien lohkojen päivitys ajan tasalle kirjastosta (Siemens AG, 2018e; Siemens AG, 2018f).

Kun Openness Scripter ei täyttänyt vaatimuksia integraatioympäristössä tapahtuville tehtäville, päädyttiin rakentamaan oma C#-kielinen rajapintasovellus.

Ennen sovelluksen rakentamista täytyi selvittää latausta edeltävät toimenpiteet, jotka mahdollistavat itse logiikkaohjelman oikeaoppisen latauksen integraatiopalvelimen simulaattorille ilman virheitä.
Tarvittavat toimenpiteet olivat:

- TIA Portalin instanssin käynnistys.
- Määritetyn projektin avaaminen.
- PLCSIM Advancedin käynnistys.
- Logiikkainstanssin yhteyden ja IP osoitteiden määrittäminen.
- Logiikkainstanssin luominen simulaattorissa.
- TIA Projektin latauksen yhteyskonfigurointien määrittäminen.
- Latausta edeltävien toimenpiteiden suorittaminen kuten logiikkainstanssin moduulin pysäyttäminen ja lataustyypin määrittäminen.
- Latauksen jälkeisten toimenpiteiden suorittaminen.

Vasta näiden toimenpiteiden jälkeen itse ohjelman lataus simulaattoriin on mahdollista.

### 7.2.1 Openness-rajapintasovelluksen rakentaminen

Ohjelma päätettiin rakentaa Visual Studio 2017 ohjelmankehitysympäristössä. Sovelluksesta tehtiin luonnollisesti konsolisovellus. .NET frameworkin versioksi laitettiin 4.6.1 johtuen rajapintakirjaston vaatimuksista (Siemens AG 2018c, 7). Simulaattorin ja Opennessin rajapintatoimintojen suorittaminen vaati rajapintakirjastojen referoimisen ja kutsumisen Visual Studiossa. (Siemens AG 2017a, 82; Siemens AG 2018c, 7.)

Ohjelman rakentamisen aikana hyödynnettiin Siemensin laatimia, laajasti dokumentoituja manuaaleja PLCSIM Advancedille ja Openness-rajapinnalle.

Suurimpaan osaan funktioista luotiin poikkeustapauksen tunnistus ja sen tulostus konsoliin niiden ilmaantuessa. Lisäksi funktioiden sisälle lisättiin yleinen toiminnankuvauksen tulostus, jotta käyttäjä näkee missä mennään. Nämä toiminnat helpottivat sovelluksen virheiden tunnistamista ja tulevat jatkossa hyödylliseksi, jos poikkeustapauksia alkaa esiintymään.

TIA Portal instanssin luominen:

TIA Portal voidaan käynnistää Openness API:n avulla ilman käyttöliittymänäkymää tai sen kanssa (Siemens AG 2018d, 76–77). Koodin toimintojen seuraamisen takia, kehitysvaiheessa valittiin, että TIA käynnistyy käyttöliittymän kanssa, katso kuva 39. Käyttöliittymää ei kuitenkaan tulla tarvitsemaan integraatioympäristössä tapahtuville tehtäville.

TIA Portal -projektin avaaminen:

Projektin avaaminen tehtiin kehityshetkellä ns. kovaan hakatusti, eli tietty TIA-projektin polku on kirjoitettu suoraan koodin sisään, katso kuva 39. Tämä tullaan jatkossa muuttamaan tulevaisuuden tarpeita vastaavaksi.

```
71          private static void RunTiaPortal()
72          {
73              // Käynnistä TIA Portal UI:n kanssa tai ilman, avaa määritetty projekti, poikkeustilanteiden tulostus
74
75              Console.WriteLine("Starting TIA Portal");
76              TiaPortal tiaPortal = new TiaPortal(TiaPortalMode.WithUserInterface);
77
78
79
80              Console.WriteLine("TIA Portal has started");
81              ProjectComposition projects = tiaPortal.Projects;
82
83              FileInfo projectPath = new FileInfo("D:\\users\\Niillika\\Documents\\Automation\\OpennessProjekti\\OpennessProjekti.ap15_1");
84              Project project = null;
85              try
86              {
87                  project = projects.Open(projectPath);
88              }
89              catch (Exception)
90              {
91                  Console.WriteLine(String.Format("Could not open project {0}", projectPath.FullName));
92                  Console.WriteLine("Hit enter to exit");
93                  Console.ReadLine();
94                  return;
95              }
96              Console.WriteLine(String.Format("Project {0} is open", project.Path.FullName));
```

Kuva 39. TIA Portalin instanssin luominen & projektin avaus.

Advancedin käynnistys:

Simulointityökalun käynnistys toteutettiin System.Diagnostics -nimiavaruuden Process.Start(); -metodia, katso kuva 40. (Microsoft .NET Framework, ei pvm.)

```
61
62          public void StartAdvanced()
63          {
64              try
65              {
66                  System.Diagnostics.Process.Start("C:\\Program Files (x86)\\Siemens\\Automation\\PLCSIMADV\\bin\\Siemens.Simatic.PlcSim.Advanced.UserInterface.exe");
67              }
68              catch (Exception eksku)
69              {
70                  Console.WriteLine(eksku.StackTrace.ToString());
71              }
72          }
```

Kuva 40. StartAdvanced-funktio.

Logiikkainstanssin yhteysmääritykset:

Logiikkainstanssin IP ja kommunikointimääritykset asetettiin kuvan 41, tavalla. Kuvan mukaiset asetukset luovat instanssin IP-osoitteen ja määrittävät kommunikointityypin, tässä tapauksessa kommunikointityypiksi valittiin TCP/IP.

```
50          private SIPSuite4 instanceIP = new SIPSuite4("192.168.0.102", "255.255.255.0", "0.0.0.0"); //IP:n asetus
51
52          public PLCInstance(string instanceName)
53          {
54
55              //Instanssin määritys
56
57              Instance = SimulationRuntimeManager.RegisterInstance(instanceName);
58              Instance.CommunicationInterface = ECommunicationInterface.TCPIP;
```

Kuva 41. Logiikkainstanssin yhteyskonfigurointi.

Logiikkainstanssin luominen:

Instanssin luominen tapahtui rajapinnan PowerOn-funktiolla (Siemens AG 2017a, 147). Luomisen yhteydessä instanssille annetaan myös kuvassa 41 näkyvät yhteysmääritykset ja timeout-arvo millisekunneissa, katso kuva 42.

```
77          public void PowerOnPLCInstance()
78          {
79
80              //Käynnistää instanssin, (timeout arvo ms)
81
82              try
83              {
84                  Console.WriteLine($"Powering on {Instance}");
85                  Instance.PowerOn(60000);
86                  Instance.SetIPSuite(0, instanceIP, true);
87
88              }
89              catch (Exception eksku)
90              {
91                  Console.WriteLine(eksku.StackTrace.ToString());
92              }
93
94
95          }
96
```

Kuva 42. Simulointityökalun logiikkainstanssin luominen.

TIA Projektin latauskonfiguraatiot:

TIA Projektin logiikkalaitteen latauskonfiguraatiot määriteltiin kuvan 43, tavalla. Kyseinen osio vastaa TIA Portalissa näkymää, kun PLC-sovellusta aletaan lataamaan tietokoneelta logiikkaan. Liitäntätyypiksi valittiin PN/IE ja sovittimeksi Siemens PLCSIM Virtual Ethernet adapter, jotta löydetään oikea logiikkainstanssi. Molemmat ovat toteutettu Find(); -funktiolla, joka etsii vastaavaa string-muotoista muuttujaa.

```
150 ConnectionConfiguration configuration = downloadProvider.Configuration;
151 ConfigurationMode configurationMode = configuration.Modes.Find("PN/IE");
152 ConfigurationPcInterface pcInterface = configurationMode.PcInterfaces.Find("Siemens PLCSIM Virtual Ethernet Adapter", 1);
153 //("Intel(R) Ethernet Connection I217 - LM", 1);
154 IConfiguration targetConfiguration = pcInterface.TargetInterfaces[0];
```

Kuva 43. Projektin yhteysmääritykset

Ohjelman lataus logiikkainstanssiin:

Kuvassa 44 määritetään lataukselle suoritettavaksi toimenpiteiksi latausta edeltävät ja jälkeen tapahtuvat tehtävät sekä se, että instanssiin ladataan projektin PLC-laitteelle tehdyt laitteistomuutokset ja itse logiikkasovellus. Latausta edeltävät toimenpiteet (PreConfigureDownload) olivat moduulin pysäytys ja eri latausasetuksien määrittäminen. Määrityksiin kuului sovelluksen lataustyypin asetuksen lisäksi hälytystekstien lataus ja latausnotifikaattimuuttujan nostaminen TRUE:ksi.

Latauksen jälkeen (PostConfigureDownload) tapahtuviin toimenpiteisiin kuului pelkästään instanssin moduulin käynnistys eli asetus RUN-tilaan.

```
156
157                     DownloadConfigurationDelegate preDownloadDelegate = PreConfigureDownload;
158                     DownloadConfigurationDelegate postDownloadDelegate = PostConfigureDownload;
159
160                     downloadProvider.Download(targetConfiguration, preDownloadDelegate, postDownloadDelegate, DownloadOptions.Hardware | DownloadOptions.Software);
161                     Console.WriteLine("Software downloaded to PLC successfully");
162                 }
163                 catch (Exception excep)
164                 {
165                     Console.WriteLine("Could not download program to PLC");
166                     Console.WriteLine(excep.StackTrace.ToString());
167                 }
168             break;
169         }
170     }
171 }
```

Kuva 44. PLC-sovelluksen lataus instanssiin.

Yllä mainittujen toimintojen lisäksi ohjelmaan luotiin jatkoa varten instanssien käsittelyyn muutama lisäfunktio, kuten instanssin moduulin pysäytys, itse instanssin sammuttaminen ja sen poistaminen, sekä projektien sisäisten laitteiden etsintää helpottava funktio. Ohjelmaan tuli yhteensä n. 300 riviä koodia lyhyehköt kommentit mukaan lukien.

## 7.3 Käyttöohjeiden laatiminen

Kirjastojen käyttöohjeet laadittiin yrityksen tarpeiden mukaan. Käyttöohjeet sisältävät ohjeiden lisäksi yleistä tietoa TIA Portalin kirjastoista ja simulaattoreista sekä sen, että miten niitä käytetään kuvien kanssa havainnollistaen. Näiden lisäksi ohjeisiin lisättiin kirjastojen käyttö TortoiseSVN-versionhallinnan kanssa sisältäen kirjaston päivityksen omalla asemalla ajantasalle ja sen päivittäminen versionhallintaan svn commit & update -toiminnoilla. Kuvasta 45 selviää mitä dokumentti pitää sisällään otsikkotasolla.

## Sisällysluettelo

1. Yleistä kirjastoista ..... 2
1.1. Kirjaston rakenne levyllä ..... 2
1.2. Kirjastotyypit ..... 3
1.3. Yrityskirjaston asetustiedosto CorporateSettings.xml ..... 3
2. Kirjaston rakenne TIA Portaalissa ..... 4
2.1. Types ..... 5
2.2. Master Copies ..... 5
2.3. Common Data ..... 5
2.4. Resources & Languages ..... 5
3. Kirjaston käyttö ..... 5
3.1. Kohteen tuominen projektiin ..... 6
3.2. Lohkon & datatyyppien luominen ja vieminen kirjastoon ..... 7
3.3. Lohkojen & datatyyppien versiointi, päivitys ja niiden kommentointi ..... 8
3.4. Export & Import ..... 10
4. Kirjaston hallinta ..... 10
4.1. Kirjaston rakenteen muuttaminen ..... 10
4.2. Kirjastojen "siivoaminen" ..... 11
4.3. Lohkojen dokumentointi ..... 12
4.4. Kirjaston arkistointi ..... 12
5. SVN ..... 13
5.1. Kirjaston päivitysten tuominen SVN:stä (SVN Update) ..... 13
5.2. Kirjaston päivitysten vieminen SVN:ään (SVN Commit) ..... 13
5. Simulaattorit ..... 14
5.1. S7-PLCSIM ..... 14
5.1. S7-PLCSIM Advanced ..... 15

Kuva 45. Käyttöohjeiden sisällysluettelo.

TIA Portalin kirjastojen ja simulaattorien käyttöohjeisiin tuli sisältöä yhteensä 17 sivun verran. Huomiona, että dokumenttia, varsinkin "simulaattorit" osiota tullaan laajentamaan ja tarkentamaan jatkossa. Käyttöohjeen lisäksi opinnäytetyön tiedonetsinnästä johtuneesta "tiedonkeräysdokumentista" luotiin kehittäjille kirjastoista ja simulaattoreista sekä niiden toiminnoista yleistä ja teknistä tietoa sisältävä dokumentti. Dokumentin lukeminen ei ole mikään välttämättömyys kehittäjille, vaan antaa pikemminkin teknistä nippelitietoa työkaluista, joka saattaa tulevaisuudessa olla hyödyksi.

# 8 JATKOTOIMENPITEET

Jatkotoimenpiteinä tulee olemaan kirjaston sisällön rakenteen muokkaaminen yritykselle sopivaksi ja rajapintasovelluksen toimintojen optimointi sekä muokkaus täysin automaatioympäristöön soveltuvaksi. Lisäksi rajapintasovellukseen pitää lisätä toiminto, joka kirjoittaa TIA-projektin logiikkalaitteen muuttujataulukon tiettyyn muuttujaan sen hetkisen sovelluksen versio- ja build tiedot. Esimerkkinä muuttujassa voisi olla version numero, tekijä ja buildin numero. Openness-rajapinta tekee tulevaisuudessa monipuolisienkin automaattitestien suorittamisen mahdolliseksi, tätä saatetaan myös hyödyntää jatkossa. TIA Portalin lohkoista saatuja SCL-muotoisia export-tekstitiedostoja saatetaan jatkossa käyttää hyväksi, koska tiedostot ovat täysin luettavissa ja muokattavissa TIA:n ulkopuolella. Niiden dokumentoinnit lohkosisältöineen voidaan esimerkiksi viedä yrityksen sisäisille verkkosivuille, josta ne olisivat helposti kehittäjien saatavilla. Yritykselle luotuja käyttöohjeita tullaan viilaamaan kehittäjien tarpeiden mukaisiksi palautteiden perusteella. Jatkossa tullaan pitämään erillinen yrityksen sisäinen koulutus liittyen kirjastojen, simulaattorien ja yleisesti TIA Portalin käyttöön. Uusille työntekijöille tullaan luomaan perehdytystehtäviä, jotka selventävät TIA Portalin käyttöä ja sitä, että miten logiikkaohjelmointia tehdään YSP:llä.

# 9 YHTEENVETO

Työn lopputuloksina saatiin luotua yritykselle alustavat aputyökalut ja pohjarakenne integraatioprosessin läpi meneville tulevaisuuden Siemen PLC-sovelluksille. Lisäksi saatiin dokumentoitua TIA Portalin simulaatiotyökalut ja kirjastot. PLC-kirjastot saatiin luotua ja onnistuneesti vietyä versionhallintajärjestelmään. Eri kirjastomahdollisuuksia tutkittaessa opittiin myös TIA Portalin omista ominaisuuksista ja mahdollisista lohkojen ulosvientitavoista. Openness-rajapinta oli uutta ja sen kanssa työskentely mielenkiintoista. Integraatioympäristössä tapahtuva PLC-sovelluksen automaattinen lataus simulaatiotyökalun instanssille saatiin toteutettua kyseisen rajapinnan kautta C#-sovelluksella. Sovelluksen rakennus tuotti hankaluuksia johtuen siitä, että allekirjoittaneella ei ollut aiempaa kokemusta ohjelmistokehityksestä, puhumattakaan C#-kielestä. Työn aikana ilmeni myös jonkin verran teknisiä ongelmia, mutta näistä päästiin ylitse ja lopputulos on halutunlainen. Työ niin sanotusti opetti tekijää ja täten TIA Portalista ja sen eri työkaluista löytyi uusia ominaisuuksia, joista en aiemmin tiennyt. C#-ohjelmointi oli uutta ja rajapintasovellusta rakentaessa hyvinkin opettavaista.

# LÄHTEET

BAYER. Andrew, 2011. Hudson's future. [Viitattu 2019 -3- 4]. Saatavissa: https://jenkins.io/blog/2011/01/11/hudsons-future/

BOOCH, Grady 1991. Object Oriented Design: With Applications. KALIFORNIA: BENJAMIN CUMMINGS PUB. CO.

COLLINS-SUSSMAN, Ben, FITZPATRICK, Brian W ja PILATO, C. Michael 2011. Version Control with Subversion. [Viitattu 2019 -2- 10]. Saatavissa: http://svnbook.red-bean.com/

Cruicecontrol. [Viitattu 2019 -3- 6]. Saatavissa: http://cruisecontrol.sourceforge.net/overview.html

Dynniq. Dynniq Group. Sisäinen dokumentti [Viitattu 2019 -1- 30].

ERNST, Michael 2012. Version control concepts and best practices. [Viitattu 2019 -2- 13]. Saatavissa: https://homes.cs.washington.edu/~mernst/advice/version-control.html

FOWLER, Martin 2006. Continuous Integration. [Viitattu 2019 -2- 8]. Saatavissa: https://martinfowler.com/articles/continuousIntegration.html

Git. [Viitattu 2019 -2- 16]. Saatavissa: https://git-scm.com/

Git. Documentation. [Viitattu 2019 -2- 16]. Saatavissa: https://git-scm.com/docs/git

Hudson. [Viitattu 2019 -3- 4]. Saatavissa: http://hudson-ci.org/

Jenkins, 2018. Dashboard view [Vittattu 2019 -3- 4]. Saatavissa: https://wiki.jenkins.io/display/JENKINS/Dashboard+View

Jenkins. Extend Jenkins. [Viitattu 2019 -3- 4]. Saatavissa: https://jenkins.io/doc/developer/

JEFFRIES, Ron 2011. What is Extreme Programming?. [Viitattu 2019 -3- 6]. Saatavissa: https://ronjeffries.com/xprog/what-is-extreme-programming/#small

Microsoft .NET 4.6.1. Process.Start Method. [Viitattu 2019 -3- 23]. Saatavissa: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.start?view=netframework-4.6.1

Microsoft Visual Studio 2017. Visual Studio IDE Documentation. [Viitattu 2019 -2- 10]. Saatavissa: https://docs.microsoft.com/en-us/visualstudio/ide/?view=vs-2017

PATHANIA, Nikhil 2016. Learning Continuous Integration with Jenkins. BIRMINGHAM: PACKT PUBLISHING LTD.

RAYMOND, Eric 2008. Understanding Version-Control Systems. [Viitattu 2019 -2- 22] Saatavissa: http://www.catb.org/esr/writings/version-control/version-control.html

Siemens AG. TIA Portal – teollisuusautomaation ohjelmistoalusta. [Viitattu 2019 -2- 17]. Saatavissa: http://www.siemens.fi/fi/industry/teollisuuden_tuotteet_ja_ratkaisut/tuotesivut/tia_portal.php

Siemens AG. Visualization – Simatic WinCC (TIA PORTAL). [Viitattu 2019 -2- 18]. Saatavissa: https://w3.siemens.com/mcms/automation-software/en/tia-portal-software/wincc-tia-portal/Pages/default.aspx

Siemens AG 2015. Step 7 Standards compliance according to IEC 61131-3 (3rd edition). [Viitattu 2019 -2- 20].

Siemens AG 2017a. S7-1500 S7-PLCSIM Advanced Function Manual. [Viitattu 2019 -2- 25].

Siemens AG 2017b. Engineering Tools S7-PLCSIM V15 online help Programming and Operating Manual. [Viitattu 2019 -3- 18]. Saatavissa: https://cache.industry.siemens.com/dl/files/511/109761511/att_964932/v1/S7-PLCSIMenUS_en-US.pdf

Siemens AG 2018a. Guide to Standardization. [Viitattu 2019 -2- 20].

Siemens AG 2018b. Step 7 and WinCC Engineering V15.1 System Manual. [Viitattu 2019 -2- 22].

Siemens AG 2018c. TIA Portal Openness: Introduction and Demo Application. [Viitattu 2019 -2- 27]. Saatavissa: https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness%3A-introduction-and-demo-application?dti=0&lc=en-WW

Siemens AG 2018d. Openness: Automating creation of projects System Manual. [Viitattu 2019 -3- 12].

Siemens AG 2018e. Openness Scripter: Introduction. [Viitattu 2019 -2- 24].

Siemens AG 2018f. Openness Scripter: Detailed Documentation. [Viitattu 2019 -2- 24].

SINK, Eric 2011. Version Control by Example. [Viitattu 2019 -2- 16]. Saatavissa: https://ericsink.com/vcbe/html/index.html

TortoiseSVN. [Viitattu 2019 -2- 16]. Saatavissa: https://tortoisesvn.net/

TortoiseSVN, 2018 A Subversion client for Windows. [Viitattu 2019 -2- 16]. Saatavissa: https://tortoisesvn.net/docs/nightly/TortoiseSVN_en/

WELLS, Don 1997. The Rules of Extreme Programming. [Viitattu 2019 -3- 6]. Saatavissa: http://www.extremeprogramming.org/rules.html

YSP Historia. [Viitattu 2018 -1- 30]. Saatavissa: https://www.ysp.fi/ysp/historia

YSP Laatujärjestelmä. [Viitattu 2019 -1- 30]. Saatavissa: https://www.ysp.fi/ysp/laatu/

YSP Lyhyesti. [Viitattu 2019 -1- 30]. Saatavissa: https://www.ysp.fi/ysp/