YRKESHÖGSKOLAN NOVIA

Version Control of PLC Blocks and Code

Jesper Nikander

Degree Thesis

Thesis for a Bachelor of Engineering Degree

Electrical Engineering and Automation

Vaasa 2024

BACHELOR'S THESIS

Author:	Jesper Nikander
Degree	Electrical Engineering and Automation, Vaasa
Specialization:	Automation Engineering
Supervisor:	Mats Warg, Mikko Valli, VEO
	Jan Berglund, Novia University of Applied Sciences

Title: Version Control of PLC Blocks and Code

Date: 01.05.2024 Number of pages 31

Abstract

This thesis is conducted for VEO OY in Vaasa as a groundwork for their plans to improve the backup and version control infrastructure for their Programmable Logic Controller (PLC) software. The goal of the thesis is to evaluate different version control systems and implement solutions that are tailored to meet the specific needs of VEO's PLC programming environment.

While different version control systems have been used for traditional text-based coding for decades, the advancement of effective version control systems for PLCs has not kept pace. The development of such a system to version control PLC code and blocks is recognized as an important step to guaranteeing the reliability and longevity of their automation solutions.

This project addresses the different challenges VEO faces in managing and troubleshooting changes to PLC code in their final applications, focusing on the extraction of historical function block data in a standardized form. Additionally, conducting detailed research and testing of both proprietary and open-source solutions with the cost-effectiveness and practicality of different systems in mind.

Language: English

Key words: version control, PLC, function block, backup

EXAMENSARBETE

Författare:	Jesper Nikander
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktning:	Automation
Handledare:	Mats Warg, Mikko Valli, VEO
	Jan Berglund, Yrkeshögskolan Novia

Titel: Versionshantering av PLC-block och Kod

Datum: 1.05.2024 Sidantal: 31

Abstrakt

Detta examensarbete genomförs för VEO OY i Vasa som en grund för deras planer på att förbättra infrastrukturen för säkerhetskopieringen och versionshantering för deras mjukvara för programmerbara logiska styrsystem (PLC). Målet med arbetet var att utvärdera olika versionshanteringssystem och implementera lösningar som är skräddarsydda för att möta de specifika behoven hos VEO:s PLC-programmeringsmiljö.

Medan olika versionshanteringar har använts för traditionell textbaserad kodning i årtionden har utvecklingen av effektiva versionshanteringssystem för PLC:er inte hållit samma takt. Utvecklingen av ett sådant system för PLC-kod och block anses vara ett nödvändigt steg för att säkerställa tillförlitligheten och livslängden hos deras automationslösningar.

Detta projekt tar upp de olika utmaningarna som VEO står inför när det gäller att hantera ändringar och felsöka PLC-kod i deras slutliga applikationer. Detta har gjorts med fokus på utvinning av historiska funktionsblocksdata i standardiserad form. Samtidigt har detaljerad forskning och testning av både licenserade och öppna källkodslösningar med kostnadseffektiviteten och funktionalitet för olika system varit i åtanke.

Språk: svenska
Nyckelord: versionshantering, funktions block, säkerhetskopiering, PLC

OPINNÄYTETYÖ

Tekijä:	Jesper Nikander
Koulutus ja paikkakunta:	Sähkö- ja automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto:	Automaatiotekniikka
Ohjaaja:	Mats Warg, Mikko Valli, VEO
	Jan Berglund, Yrkeshögskolan Novia

Nimike: PLC-lohkojen ja koodin versionhallinta

Päivämäärä: 1.05.2024   Sivumäärä: 31

Tiivistelmä

Tämä opinnäytetyö on tehty VEO OY:lle Vaasassa pohjana heidän suunnitelmilleen kehittää ohjelmoitavien logiikkasäädinten (PLC) varmuuskopiointi- ja versiohallintainfrastruktuuria. Opinnäytetyön tavoitteena oli arvioida erilaisia versionhallintajärjestelmiä ja toteuttaa ratkaisuja, jotka on räätälöity vastaamaan VEO:n PLC-ohjelmointiympäristön erityistarpeita.

Vaikka erilaisia versionhallintajärjestelmiä on käytetty perinteiseen tekstipohjaiseen koodaukseen jo vuosikymmeniä, PLC-ohjelmointiin tarkoitettujen versiohallintajärjestelmien kehitys ei ole pysynyt samassa tahdissa. Tällaisen järjestelmän kehittäminen PLC-koodin ja lohkojen versiohallintaan on todettu tärkeäksi askeleeksi heidän automaatioratkaisujen luotettavuuden sekä pitkäikäisyyden takaamiseen

Tämä projekti käsittelee erilaisia haasteita, joita VEO kohtaa PLC-ohjelmiston muutoksissa ja vianmäärityksessä heidän lopullisissa sovelluksissaan. Painopisteenä on historiallisten lohkotietojen manipulointi ja poimiminen standardoidussa muodossa, sekä lisensoitujen että avoimen lähdekoodin ratkaisujen tutkimus ja testaus eri järjestelmien kustannustehokkuutta ja käytännöllisyyttä ajatellen.

Kieli: suomi
Avainsanat: versionhallinta, lohko, varmuuskopiointi, PLC

Table of contents

Table of figures

List of abbreviations

PLC	-	Programmable logic controller
DVC	-	Distributed version control
CVC	-	Centralized version control
XML	-	Extensible Markup Language
XBD	-	XBus Device
CSV	-	Comma-separated value
SVN	-	Subversion
SSH	-	Secure shell protocol
IEC	-	International Electrotechnical Commission
GUI	-	Graphical user interface
TIA	-	(siemens) Totally integrated Automation
API	-	Application programming interface

1 Introduction

Programmable Logic Controllers (PLCs) are essential for the operation of modern industrial processes. As industrial automation projects grow more complex, the need for a powerful version control system becomes crucial to ensure the smooth development and maintenance of PLC programs. The effectiveness and reliability of PLC-based systems depend significantly on proper management of their programming code. Version control systems (VCS) have been a crucial part of software development for decades. Adapting these systems for PLC programming proves quite a challenge with its unique needs and graphic elements.

1.1 Purpose and challenges

The primary goal of this bachelor's thesis is to explore and implement methods to enhance the traceability of modifications made to final PLC projects. This will be achieved by comparing these projects with VEO's backups. This study will assess how the adoption of version control systems such as GIT, SVN, or other PLC-specific solutions can improve troubleshooting efforts for problems that arise during the operational runtime of the PLC. The focus will be specifically on projects that use Siemens and Schneiders PLCs but methods that work with all PLC brands are seen as advantageous.

The main challenges stem from the PLC programming languages and environments. Unlike traditional software development that uses text-based coding languages. PLCs are programmed with specialized languages with graphical elements and binary files. This complicates access to historical versions of code in a format that is human-readable and meaningful. Another complexity that is introduced is the physical processes that PLC programs are tightly integrated with. This demands documentation of not only code changes but also their impact on the overall system.

Maintaining a detailed version history not only helps development but also provides legal documentation. Especially in industries with tight regulations where the need for correct documentation and detailed records is mandatory. Detailed documentation ensures that version histories of software and hardware components provide a complete audit trail, fulfilling regulatory requirements.

2 PLC

Since the focus of this thesis revolves around version control systems tailored for PLCs, this section provides a brief overview of programmable logic controllers (PLCs). A PLC is a specialized type of microprocessor-based controller that utilizes programmable memory to execute instructions controlling different machines and processes. PLCs are engineered for simplicity and longevity which allows easy accessibility to engineers with varying levels of expertise.

The operation of a PLC can be broken down into three fundamental components: inputs, program execution, and outputs. PLCs interact with processes by gathering data from interconnected devices, monitoring inputs, and subsequently comparing this data against the programmed instructions. Following this comparison, the PLC adjusts the outputs accordingly to regulate the connected devices' operation. A special feature of PLCs is their ability to connect a single device such as a valve position sensor to both inputs and outputs of the same controller. This configuration enables the PLC to capture the real-time status of the device through inputs and make control adjustments based on programmed instructions through outputs, thereby guaranteeing undisturbed process regulation (inductiveautomation.com, 2020).

Compared to conventional computers, PLCs are more expensive relative to their processing power and flexibility. While conventional computers might cost less and are more versatile. PLCs offer several advantages that make them essential within industrial automation settings. Including reliability, real-time control, safety features, and low maintenance. They are designed to work in harsh environments and withstand extreme temperatures, vibrations, and electrical interference. These attributes contribute to the widespread adoption and reliance on PLCs across several industrial sectors (plctable, 2024).

2.1 Programming languages

There are five types of programming languages for PLCs that comply with the IEC 61131-3 standard. This standard works both as a rulebook and a guide for automation and control systems. Each language caters to different programming needs and preferences. This presents unique features and challenges when it comes to version control.

Graphical Languages:

- Ladder logic (LAD): Evolved from electrical wiring diagrams ladder logic's graphical nature represents circuits and logic in a format familiar to electrical engineers. However, this will complicate version control as changes in diagrams may not be as straightforward to track as text-based modifications.
- Function Block Diagram (FBD): Utilizes blocks to represent functions such as logic operations and data processing. The graphical interface allows for the visual assembly of complex logic but like ladder logic its graphical basis complicates the version control.
- Sequential Function Charts (SFC): is a visual programming language that organizes program execution flow in a graphical format. It excels in outlining the overall structure of the program by providing a high-level view of the operational sequences. Although, it is often used in conjunction with other languages, which introduces its complications with version controlling combined with the graphical nature.

Text Languages:

- Structured text (ST): Resembles high-level languages like Pascal and C. It enables complex data processing and algorithm implementation. While structured text is more compatible with conventional version control systems. The complexity can make tracking changes and understanding the program's flow more difficult without proper documentation.
- Instruction List (IL): Resembles low-level languages like assembly. It offers fine-dialed control over PLC operations at the cost of increased complexity and difficulty of the language. Its text-based code is version control friendly, but the lower-level operations can be challenging to manage and understand over time.

Version control for PLC programming compliant with IEC61131-3, demands a diversified but standardized approach to accommodate all the different PLC languages. Managing different revisions effectively in this context often requires different tools and practices that can handle both graphical and textual elements, while also supporting customary file formats that integrate closely with PLC programming environments (Lamb, 2023).

2.2 Function blocks

Function blocks are defined by the IEC61131-3 standard and encapsulate specific functions ranging from simple mathematical operations to custom complex control algorithms. These become crucial in the automation of industrial tasks. These blocks are mostly used with FBD (Function Block Diagram) programming language. They are an important part of the entire PLC programming environment due to their modularity and reusability. Their graphical nature allows for clear visualization of inputs and outputs, which makes them user-friendly and straightforward to integrate into larger systems.

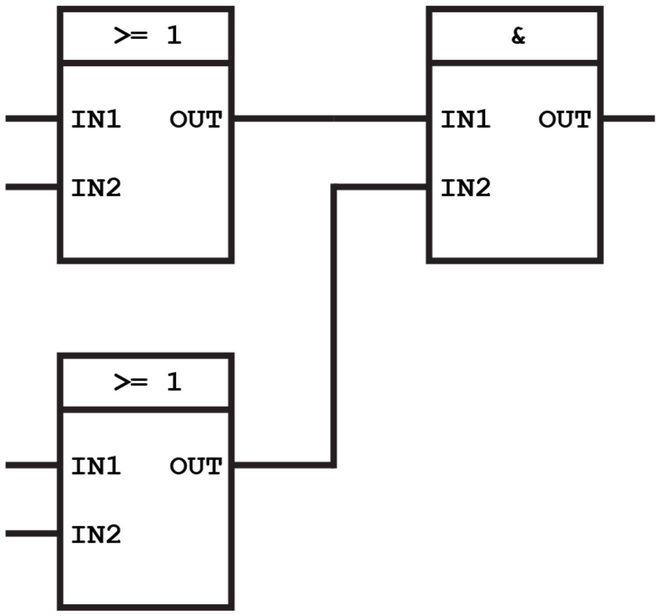


Figure 1 Combination of simple function blocks to make a basic function block diagram (Peter, 2018).

The popularity of function blocks stems from their flexibility in being able to be programmed to do whatever the user desires, along with the ability to be used in different PLC systems and brands without compatibility issues. Thanks to the IEC61131-3 standard's requirements, these blocks can operate with other programming languages in the same environment and expect to perform identically in another environment regardless of the manufacturing brand differences (Peter, 2018).

Given their crucial role in PLC programming as a whole and especially in VEO's practices of coding PLC projects. Custom-made function blocks are being used modularly in the projects at hand. These function blocks become a focus point when it comes to version control.

Managing different versions of function blocks efficiently becomes important as it impacts the ability to update, debug, and maintain a clear history of modifications. Which is invaluable for troubleshooting and refining system performance over time.

3 Version control

Version control systems are essential tools in software development for tracking and managing changes to source code throughout the software development lifecycle. These systems record every change including details like the author's name, time of when the change was made, and other important information. This documentation becomes especially useful for solving conflicts that happen when team members make changes at the same time. With version control, developers can revert to earlier revisions to fix errors with little trouble.

The transition from centralized version control systems (CVCS), which rely on a single repository to distributed version control systems (DVCS) represents a noteworthy advancement. DVCS allows for a more collaborative and flexible approach as each copy of the project is a complete repository. This structure supports better collaboration and allows teams to work on different parts of the project simultaneously without compromising the code's integrity. This also enables branching and merging strategies which let teams develop new features, fix bugs, and make enhancements all at once (BasuMallick, 2022).

3.1 Version control tools

With several options available, the selection of a version control tool typically depends on the specific needs and preferences of the development team. Most GNU General Public License (GPL) tools are designed for handling text-based files and code. Meanwhile, proprietary version control software developed by specialized companies in the field often offers advantages in the management of a wider range of file formats and code along with better user interfaces.

Among well-known version control tools are Git, IBM Configuration Management Version Control (CMVC), and Apache Subversion (SVN). Git is regarded as one of the most powerful distributed version control systems. It is known for its flexibility in managing various file formats and accommodating diverse automation needs. In contrast, SVN represents a centralized version control tool. While it is compatible with binary files like PLC code it primarily operates with text file formats. Both GIT and SVN can integrate with different programming environments simplifying the versioning process without requiring users to open a separate program to commit changes (BasuMallick, 2022).

3.2 Terms

Version control, which is also known as "source control" or "revision control," includes various terms that enable collaboration by coordinating changes among multiple users. This encourages independent work while guaranteeing a united result. For effective versioning in day-to-day use only a few key terms are essential. These include commit, repo, clone, pull, and branch (Wright, 2020).

Important Version control terms:

Repo (repository): Storage space for files and tracks changes made to a project. Allowing navigation through the project’s history.

Commit: The action of adding changes to the repository with a commit message explaining the modifications.

Diff (difference): A tool for comparing changes between commits. Meant to visually track modifications over time.

Conflict: A situation that happens when two parties modify the same file and manual resolution is needed.

Clone: The process of creating a local copy of a remote repository by duplicating its contents at the time of cloning.

Push: The action of Updating the remote repository with changes made in the local repository.

Pull: the action of updating the local copy with changes made by others in the remote repository.

Forking: Creation of a new repository from an existing one. Often on platforms like GitHub or a virtual network to start a new development.

Branch: Creation of an alternate version of the project to work on new features or ideas without affecting the main development.

Check out: The action of moving between branches to work on specific parts of the project.

3.3 Repositories

Centralized version control systems (CVCS) represent an older yet simpler approach to version control methodology. These systems centralize all version control activities within a single repository. This singular repository model simplifies setup and usage which makes it an appealing choice for teams seeking straightforward version management solutions. The linear progression of file revisions in CVCS allows for direct and uncomplicated collaboration among team members. The latest version of the project is always accessible on the central server. However, this centralized approach ties the efficiency and availability of version control operations to the server's reliability, potentially becoming a critical point of failure (Ernst, 2012).

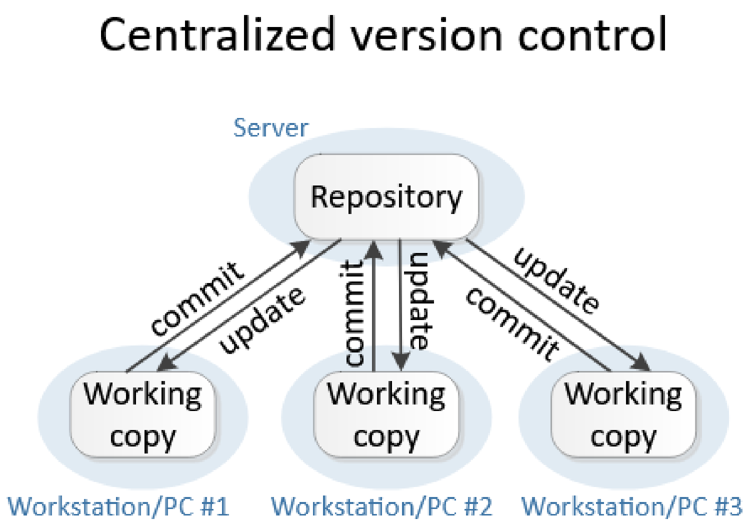


Figure 2 Centralized version control (CVC) (Ernst, 2012).

On the other hand, distributed version control systems (DVCS) represent a more modern approach to version control defined by its use of multiple repositories. This setup allows each contributor to have a complete copy of the repository including the full history of changes on their workstation. This decentralized model offers several advantages including faster operations, reduced sensitivity to errors, and a broader set of features. The ability to commit changes, branch, and merge locally. Without immediate reliance on a central server supports a more dynamic and resilient workflow. However, the complexity of managing multiple repositories and understanding distributed histories introduces a steeper learning curve. Making DVCS somewhat more challenging to learn and understand (Ernst, 2012).

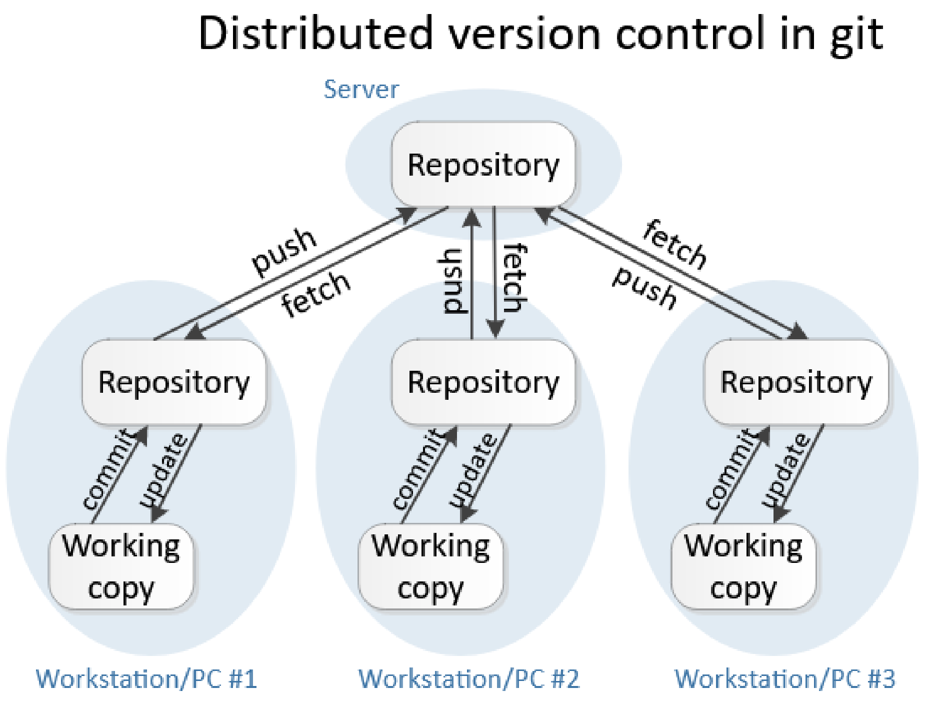


Figure 3 Distributed version control (DVC) (Ernst, 2012).

4 GIT and SVN

The major difference between GIT and other version control systems lies in the way GIT understands its data. Most VCSs such as Concurrent Version System (CVS), Subversion, Perforce, and others store information as a list of file-based changes (delta-based). Git operates differently. GIT views its data as a series of snapshots of a miniature filesystem. With each commit or saved state of the project GIT captures a snapshot of what all the files look like at that moment and stores a reference to that snapshot (Henry, 2022).

Apache Subversion (SVN) came into being in 2000 by CollabNet to replace the Concurrent versions system (CVS) due to its design flaws, such as not accepting atomic commits which are a set of modifications that may be constituted by several lines of code as a single operation. SVN became self-hosting in August 2001 with the entire CVS team migrating to SVN. Adopting the centralized architecture SVN allows a single server to store all project data. Allowing programmers to interact with the data on their local machines.

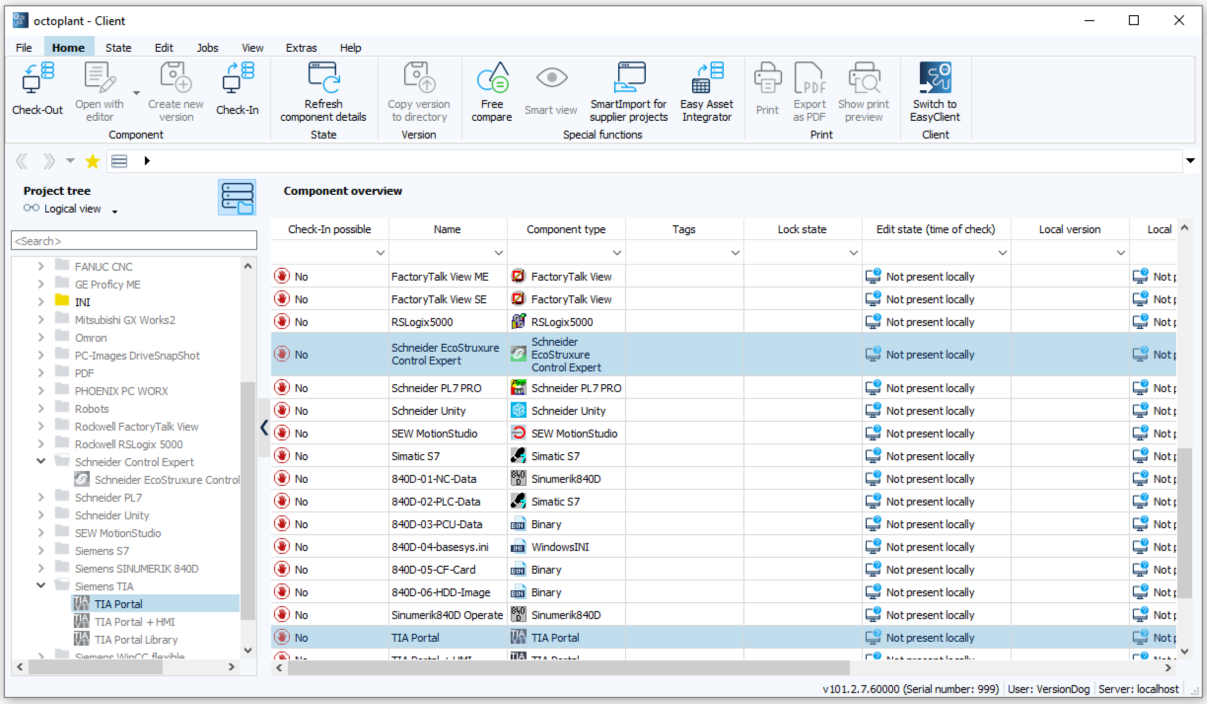
A noteworthy feature of SVN is the use of Skip-deltas which optimizes data storage. Commits in SVN require thorough review before they reach the repository and become visible to all users. This guarantees that most activities remain invisible in case of a committed failure. The server reverts to the last confirmed good state and allows the programmer to resolve conflicts in their files without affecting others (Varela, 2022).

5 Octoplant and Copia

Octoplant and Copia are proprietary version control systems specifically designed for managing PLC code. These systems offer features that align well with the requirements of VEO. Especially for comparing PLC code between projects and extraction of data or code. However, the integration of these systems with VEOs network of PLCs and projects means that the cost of these proprietary licenses quickly adds, making them a hefty investment.

Given the scale of VEO's operations and the potential rapid escalation in licensing costs, a thorough evaluation of the cost-to-benefit ratio of implementing these version control solutions is necessary. Accurate pricing information requires direct discussions between VEO's personnel and representatives from proprietary version control providers. The pricing of these systems is known to vary significantly depending on several factors such as scale of operations, features included, and number of projects that need to be managed.

Understandably these system providers are cautious about disclosing pricing details to students conducting research to prevent misrepresentation. However, discussions on various forums and articles can provide a rough estimate of such systems. Unofficial sources suggest that proprietary systems with basic PLC version control functionality are estimated to cost around ten thousand dollars per year. Enterprise systems with substantial amounts of features for larger operations can start from twenty thousand dollars or more. It should be noted that these figures are indicative and should not be taken at face value.



Check-In possible	Name	Component type	Tags	Lock state	Edit state (time of check)	Local version	Local
No	FactoryTalk View ME	FactoryTalk View			Not present locally		Not
No	FactoryTalk View SE	FactoryTalk View			Not present locally		Not
No	RSLogix5000	RSLogix5000			Not present locally		Not
No	Schneider EcoStruxure Control Expert	Schneider EcoStruxure Control Expert			Not present locally		Not
No	Schneider PL7 PRO	Schneider PL7 PRO			Not present locally		Not
No	Schneider Unity	Schneider Unity			Not present locally		Not
No	SEW MotionStudio	SEW MotionStudio			Not present locally		Not
No	Simatic S7	Simatic S7			Not present locally		Not
No	840D-01-NC-Data	Sinumerik840D			Not present locally		Not
No	840D-02-PLC-Data	Simatic S7			Not present locally		Not
No	840D-03-PCU-Data	Binary			Not present locally		Not
No	840D-04-basesys.ini	WindowsINI			Not present locally		Not
No	840D-05-CF-Card	Binary			Not present locally		Not
No	840D-06-HDD-Image	Binary			Not present locally		Not
No	Sinumerik840D Operate	Sinumerik840D			Not present locally		Not
No	TIA Portal	TIA Portal			Not present locally		Not

v101.2.7.60000 (Serial number: 999) | User: VersionDog | Server: localhost

Figure 4 Octoplant DEMO.

Octoplant as a proprietary version control system is widely recognized across various industries such as Automotive, pharmaceuticals, chemical manufacturing, and energy. Several well-known corporations have incorporated Octoplant into their operations and is also suitable for smaller businesses. This extensive software combines AUVESY's versiondog and MDT's autosave into a single system, which emerged when AUVESY and MDT Software unified in 2021. Most PLC brands and programming environments can be directly implemented into the system and support easy comparison between revisions of the PLC code on both Schneider and Siemens systems. Octoplant also supports the extraction and analysis of specific project components. Including the listing of all function blocks used in the project to enable swapping blocks between both the backup and current version of the project (Ayvesy-mdt, 2024).

If testing of proprietary version control systems is desired before discussing pricing and full-scale demonstrations by these providers, Octoplant offers a free 30-day trial for a demo environment with all of the features. This way the user is able to understand better what to expect from such systems when starting the discussion process.

6 GIT - setup

For this thesis, GIT is chosen as the main version control system to test the implementation. It is the most popular open-source solution and is very modifiable to meet all requirements set by both developers and users. Depending on VEO's preferences for features and workflow either "GIT extensions" or "GIT Gui" as the graphical user interface can be used for this system. Although GITs configurations and operations can be managed without a graphical user interface by using tools like Windows Terminal or GIT BASH, this thesis employs a GUI for its simplicity in teaching new users the workflow. Especially in the development of new PLC projects. (Scott Chacon, 2014)

The best way to get started with GIT is by consulting the official guidebook "Pro Git" available at https://git-scm.com. This comprehensive resource covers all aspects of GIT. But for a quick start, a brief overview of the setup process is provided here.

6.1 Client side

Installation

For Windows and macOS there are a few ways to install git. The most official build is available for download on GIT's own website. https://git-scm.com/download/win
Simply following the installation instructions provided sets the user up with the basic tools necessary for everyday use.
For Linux users, the basic tools can be installed using the package manager specific to the user's distribution. This method guarantees that git integrates smoothly with the user's system.

On Red Hat/Fedora-based systems dnf can be used:

```
$ sudo dnf install git-all
```

On Debian/Ubuntu-based systems apt can be used:

```
$ sudo apt install git-all
```

Configuration:

When using GIT for the first time some settings need to be configured that will stick around between upgrades and updates. This can be done through the Windows terminal or GIT's own command line tool "GIT BASH". Username and email address are configured to know logging who committed certain changes with the time of modifications. Alternatively, this can be done on the chosen Graphical user interfaces if the command line approach is not preferred.

```
§ git config -- global user.name USERNAME
§ git config -- global user.email THESIS.TEST@EMAIL.COM
```

New repository

To initialize a new repository using the command line the user will need to navigate to the desired directory. For example, a folder located on the user's desktop called "REPOSITORY" desired to be used as a GIT repository. The following steps need to be taken.

1. Navigation to the folder: `§ cd ~/desktop/"REPOSITORY"`
2. Initializing the repository: `§ git init`

For a simpler approach the graphical user interface can be used since is included with the installation of GIT. The use of the included GUI simplifies the creation of new repositories by selecting the folder directly in the GUI and configuring it to the users' requirements before choosing the primary GUI for the development workflow.

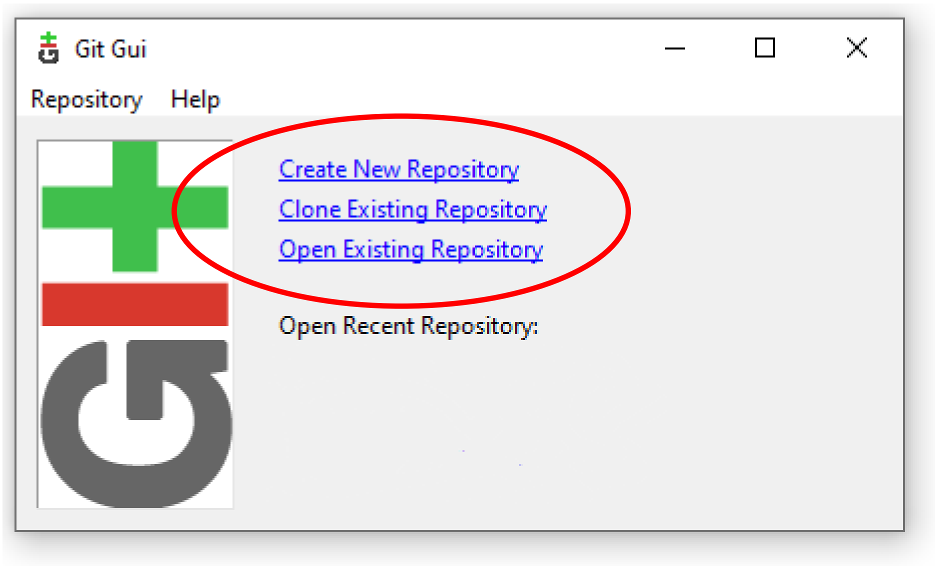


Figure 5 GIT GUI interface.

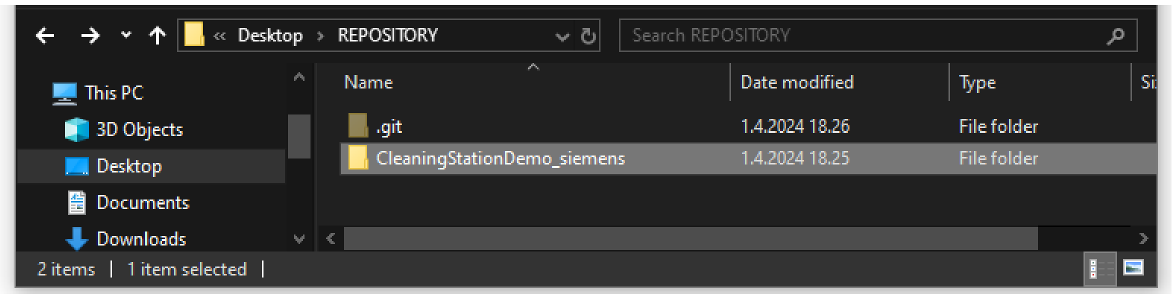
Add and commit

When navigating within the repository a hidden '.git' file is present. This folder contains all the information and data that Git uses to manage the repository's version control system and should be left alone.

Files can be added to the repository by first navigating to the repository's directory and using the 'git add' command to stage files for the next commit.

1. Navigation to directory: `§ cd ~/desktop/"REPOSITORY"`
2. Adding specific file: `§ .add 'SPECIFIC_FILE'`

Another way of adding files to the repository is simply by dragging and dropping the files into the repository.


Desktop > REPOSITORY

Name	Date modified	Type
.git	1.4.2024 18.26	File folder
CleaningStationDemo_siemens	1.4.2024 18.25	File folder

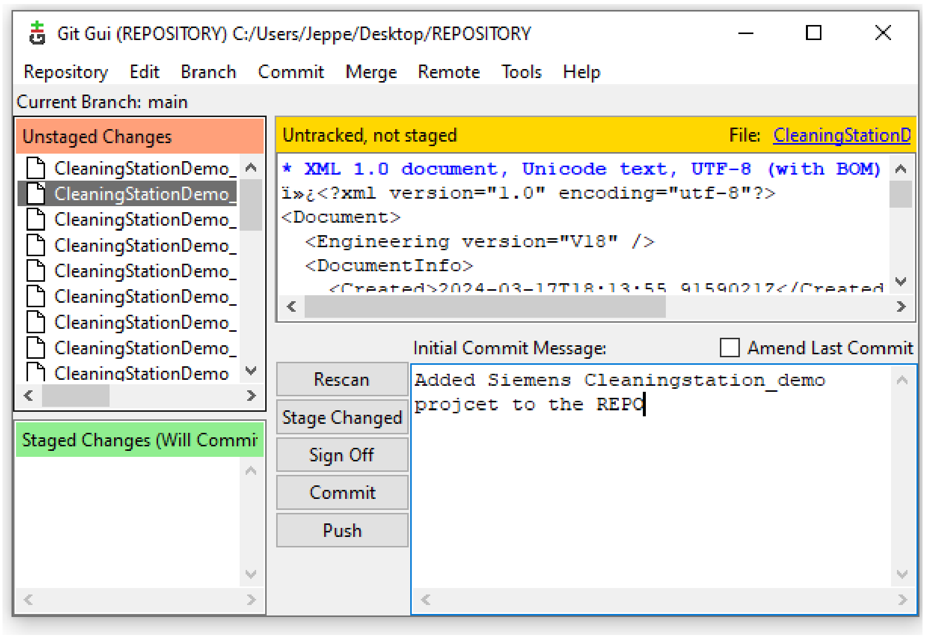
2 items | 1 item selected |

Figure 6 File explorer view of GIT repository.

The files enter a staged area where they must be committed before they are officially under version control. This step records a snapshot of the current state of the repository. If the command line is used the user is most certainly already located inside the repository since the commit often takes place directly after adding the file. Using the command line tool the command 'git commit' accompanied by a brief explanation of changes is used.

1. Committing changes: `§ git commit -m "brief explanation"`

GUIs provide an alternative simpler method where the changes are first staged accompanied by a brief explanation of changes before, they are finally committed to the repository. This gives the user a better visualization of what changes are about to be made to the repository.



```
* XML 1.0 document, Unicode text, UTF-8 (with BOM)
ï»¿<?xml version="1.0" encoding="utf-8"?>
<Document>
  <Engineering version="V18" />
  <DocumentInfo>
    <Created>2024-03-17T18:13:55.9159021Z</Created
```

```
Added Siemens Cleaningstation_demo
projcet to the REPO
```

Figure 7 GIT GUI repository changes view.

6.2 Server side

Both the server and client sides utilize the same underlying Git software. Which results in many of the configurations being similar. The server side of Git is designed for hosting remote repositories that enable collaboration between users. This setup allows the Individual users to interact with the remote repositories by “pushing” their local changes to the server.

Git provides direct integration with GitHub through SSH keys for cloud service-based hosting of remote repositories. This solution works flawlessly with excellent security that suits smaller teams or independent developers well. However, opting for an on-site hosting approach is possible and preferred by VEO to meet strict security requirements. This solution avoids storing data on external cloud servers like GitHub to better control access to the data.

Internal server setup guarantees that access to and manipulation of the repository content is strictly governed by correct security credentials. By hosting GIT repositories internally organizations can implement additional layers of security such as network firewalls and encrypted data transfers and automate certain tasks easier (Scott Chacon, 2014).

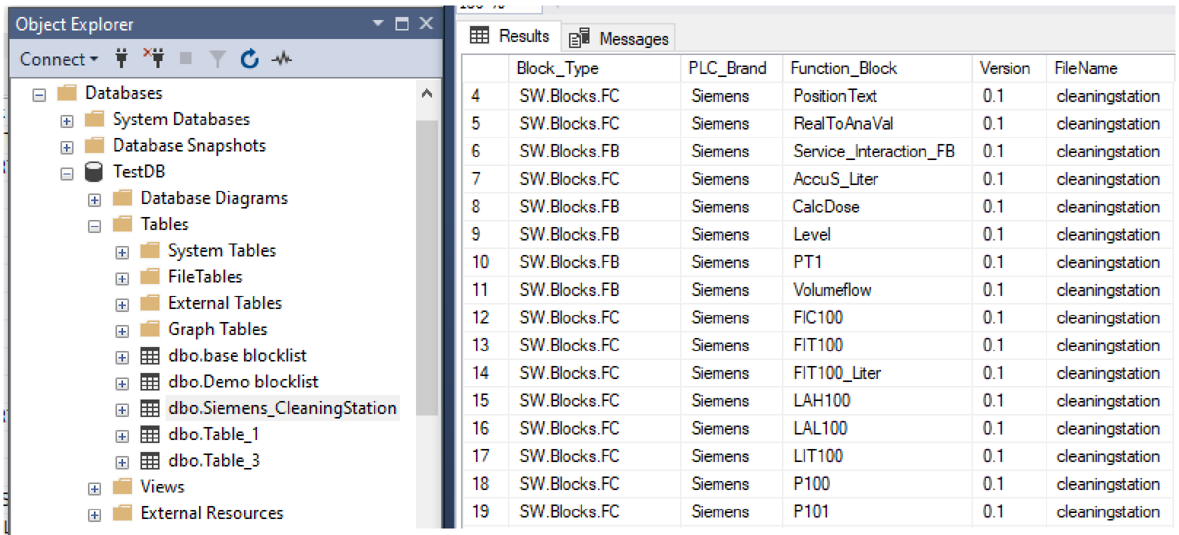
7 Database or a VEO Library

When comparing and swapping function blocks is the primary requirement without all the version control features, a database or library environment can be a practical alternative. By storing blocks and code in database-friendly formats like comma-separated value, querying and data manipulation capabilities offered by modern database systems can be utilized.

7.1 Database selection and usage

Databases like PostgreSQL and Microsoft SQL Server both support XML format for handling structured data. Which is ideal for managing different attributes of function blocks directly from XML data. However, querying through XML requires deep knowledge of every element and attribute. The names of attributes do not match between different plc brands and the user would need to learn exactly how each plc brand structures their XML code. This becomes counterproductive and time-consuming, which would defeat the whole purpose of utilizing a database.

This can be mitigated by using XML parsing tools that can generate a list of function blocks with comma-separated values. A standardized structure for data extraction and analysis is



	Block_Type	PLC_Brand	Function_Block	Version	FileName
4	SW.Blocks.FC	Siemens	PositionText	0.1	cleaningstation
5	SW.Blocks.FC	Siemens	RealToAnaVal	0.1	cleaningstation
6	SW.Blocks.FB	Siemens	Service_Interaction_FB	0.1	cleaningstation
7	SW.Blocks.FC	Siemens	AccuS_Liter	0.1	cleaningstation
8	SW.Blocks.FB	Siemens	CalcDose	0.1	cleaningstation
9	SW.Blocks.FB	Siemens	Level	0.1	cleaningstation
10	SW.Blocks.FB	Siemens	PT1	0.1	cleaningstation
11	SW.Blocks.FB	Siemens	Volumeflow	0.1	cleaningstation
12	SW.Blocks.FC	Siemens	FIC100	0.1	cleaningstation
13	SW.Blocks.FC	Siemens	FIT100	0.1	cleaningstation
14	SW.Blocks.FC	Siemens	FIT100_Liter	0.1	cleaningstation
15	SW.Blocks.FC	Siemens	LAH100	0.1	cleaningstation
16	SW.Blocks.FC	Siemens	LAL100	0.1	cleaningstation
17	SW.Blocks.FC	Siemens	LIT100	0.1	cleaningstation
18	SW.Blocks.FC	Siemens	P100	0.1	cleaningstation
19	SW.Blocks.FC	Siemens	P101	0.1	cleaningstation

Figure 8 Table of function blocks in Microsoft SQL server.

simplified over every PLC brand. Full Utilization and benefits of these tools will be discussed in more detail in Chapter 12.

7.2 RDBMS

Relational database management systems (RDBMS) like Microsoft SQL are especially good for storing and providing access to data points that are related to one another. Making utilization of these databases to manage PLC function blocks and code is surprisingly advantageous. Structuring the storage of function blocks to include attributes such as version number, file of origin and associated metadata allows users to perform thorough queries and analyses. This approach also supports tracking block usage, detection of changes, and management of versions through the database.

These databases can be deployed locally on-site If cloud service storage of sensitive data is not preferred. This local deployment enables the management of the database on its own physical servers or within a private cloud for powerful data security. By utilizing all the advanced features provided by RDBMS, organizations can even eliminate the need for separate version control software in certain situations (Rahul Awati, 2024).

7.3 NoSQL

NoSQL databases differ from other RDBMS by not relying on standard tabular schema of rows and columns. These databases can store data in various formats such as documents like JSON, key-value pairs, Wide columns, or graphs. Making them especially suitable for large sets of unstructured data. This design offers better scalability and faster access to data. RDBMS struggles with horizontal scaling due to the tabular database structure. Meanwhile, NoSQL allows for new data types and fields to be added without disturbing the existing database structure (IBM, 2024).

PLC-projects with complex and differing data structures may not always fit into a functional schema of traditional RDBMS. While NoSQL databases will not be necessarily useful in the use case of this thesis. The management of function blocks as documents may be more beneficial if large amounts of non-standardized PLC projects are developed. Due to the flexibility of NoSQL a whole restructuring and maintenance of the database is not required.

8 Binary file

In version control systems like Git the handling of binary and graphical files requires specific configurations to accommodate their unique properties. Graphical files in particular present challenges due to the way they are generated and stored. Each time a modification is made the entire file gets restructured rather than only updating the altered sections, leading to redundancy and inefficiency in version control. Since traditional version control systems lack the ability to track and manage these files efficiently, they resort to creating duplicate copies for each version.

Even if the PLC program's configuration is stored in XML format for manipulation and comparison. The original files must still be managed within the version control environment. To address this limitation specialized tools or extensions such as Git Large File Storage (Git LFS) are necessary. Git LFS allows large binary files such as Schneider's encrypted file compression formats like '.sta' and '.stu' to be stored externally from the Git repository. Instead of storing the actual files. GIT LFS stores a pointer in the repository that references the actual binary file stored on the LFS server. This approach supports efficient version control without compromising on storage or performance.

In practice, setting up GIT LFS involves installing the extension and configuring the repository to track specific file formats such as '.sta' and '.stu'. This guarantees that these files are managed efficiently using Git LFS thus preventing unnecessary duplication and ensuring smooth version control of PLC program files."

Command line tool:

1. Installation of GIT LFS: **§ git lfs install**
2. Navigation to repository: **§ cd ~/desktop/"REPOSITORY"**
3. Track desired file format: **§ git lfs track "*.stu"**
4. Track desired file format: **§ git lfs track "*.sta"**

9 XML file and automation

PLC projects are stored in a format specific to their original programming environment. That is why different file formats need to be used for the manipulation of project data outside the original programming environment. Exporting XML files from PLC projects offers advantages for managing and documenting specific elements such as function blocks and other graphical coding outside their original programming environment. While proprietary version control systems may not require this process, it remains advantageous for all open-source systems and database solutions. This section details the manual process of exporting XML files and discusses potential automation approaches for both Siemens and Schneider Electric platforms.

9.1 Schneider Control expert

Schneider Electric's Control Expert is a programming environment for developing Schneider PLC applications. Regarding version control, it presents certain challenges due to the absence of an end-user-accessible API for task automation. This requires a manual approach to exporting XML files for version control and project archiving.

In terms of versioning function blocks, which is the highlight of this thesis. Developers can export the .XBD format, which compiles data from all the project's blocks into an XML structure. While this method is functional, the .XEF file is often the preferred choice. It not only contains the complete source code but is also executable. Which allows for the direct opening of the PLC program within Schneider's Control Expert.

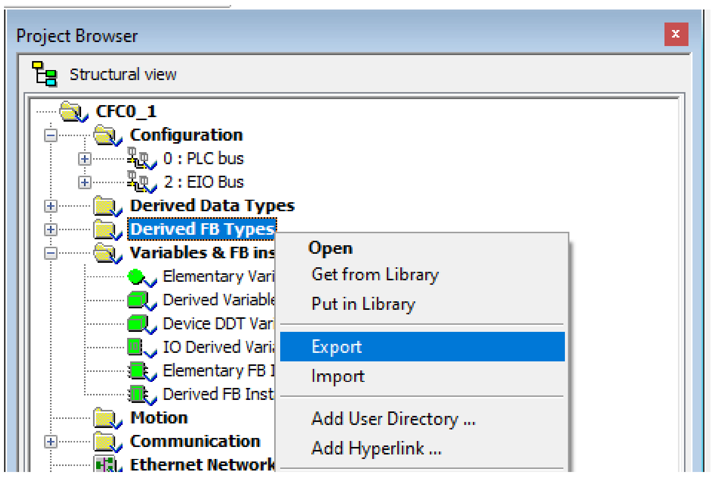
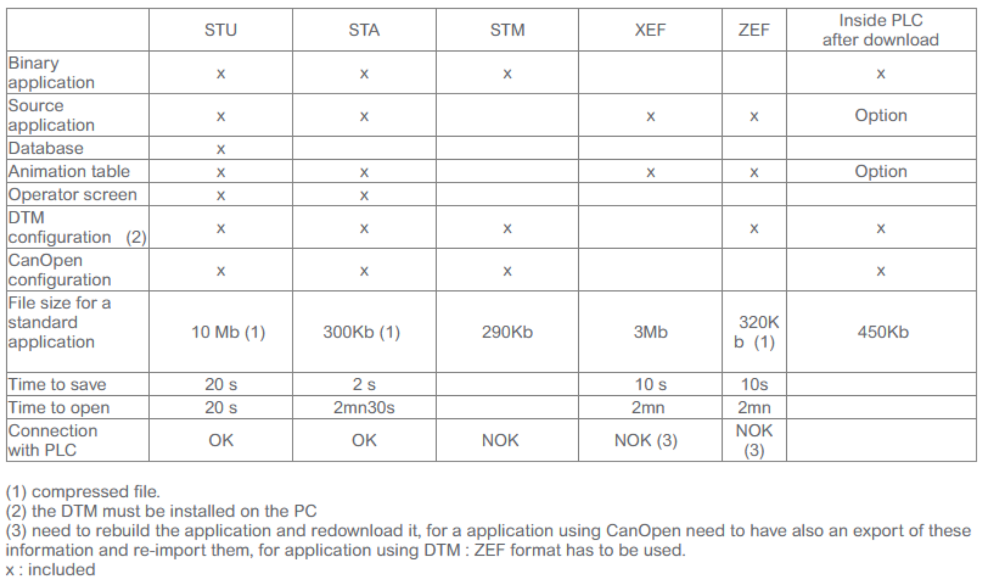


Figure 9 Schneider .XBD extraction.

However, as Control Expert continues to advance. The latest versions have shifted towards generating the .ZEF file as the standard export option. This format acts as a container that includes the desired .XEF file within it. Developers are now tasked with an additional step extracting the .XEF file from its container. Which introduces more intricacies to their process. Adapting to this change requires developers to navigate the added complexity, striking a balance between leveraging the advanced features of the environment and managing an increased workload. With the implementation of a version control system to the workflow we want to automate as much as possible and the manual labor should be as simple as possible (Schneider Electric India, 2022).



	STU	STA	STM	XEF	ZEF	Inside PLC after download
Binary application	x	x	x			x
Source application	x	x		x	x	Option
Database	x					
Animation table	x	x		x	x	Option
Operator screen	x	x				
DTM configuration (2)	x	x	x		x	x
CanOpen configuration	x	x	x			x
File size for a standard application	10 Mb (1)	300Kb (1)	290Kb	3Mb	320Kb (1)	450Kb
Time to save	20 s	2 s		10 s	10s	
Time to open	20 s	2mn30s		2mn	2mn	
Connection with PLC	OK	OK	NOK	NOK (3)	NOK (3)	

(1) compressed file.
(2) the DTM must be installed on the PC
(3) need to rebuild the application and redownload it, for a application using CanOpen need to have also an export of these information and re-import them, for application using DTM : ZEF format has to be used.
x : included

Figure 10 Schneider file formats (Schneider Electric India, 2022).

9.2 Siemens TIA PORTAL

Siemens TIA portal includes a built-in feature known as Version Control Interface (VCI), which offers a solid way to integrate projects and function blocks with external version control systems such as GIT. This integration allows versioning and management of all project files directly within the TIA Portal environment without the need to switch between different software tools during development or troubleshooting.

A unified solution to version control or extract XML code would be preferable and VCI supports extraction of various project components to XML files such as program blocks, PLC tag tables, and data types. Which would be more than enough for the purpose of version control in the context of this thesis. By standardizing the version control process, all PLC brands can be managed by the same VCS. Therefore, simplifying version control of PLC code and encouraging the use of different PLC brands as the version control process stays relatively the same across brands (Smith, 2022).

Siemens API allows for the automation of various tasks within the VCI and the broader project environment through TIA Portal's Openness. This includes scripting repetitive actions or automating processes like extracting XML code and committing changes to a GIT repository, thereby smoothing the version control process and minimizes the chance of human errors occurring (Siemens, 2021).

By exporting the program blocks as "SIMATIC Markup Language" either with API or manually, we get an XML-based format that is tailored to SIMATIC programming environments. With the SimaticML format users can manipulate and extract configurations and components outside the TIA portal. This can then be imported to other projects or TIA Portal instances. And can be especially useful with Veos practices in a "common way of programming" where standardization across multiple systems is required.

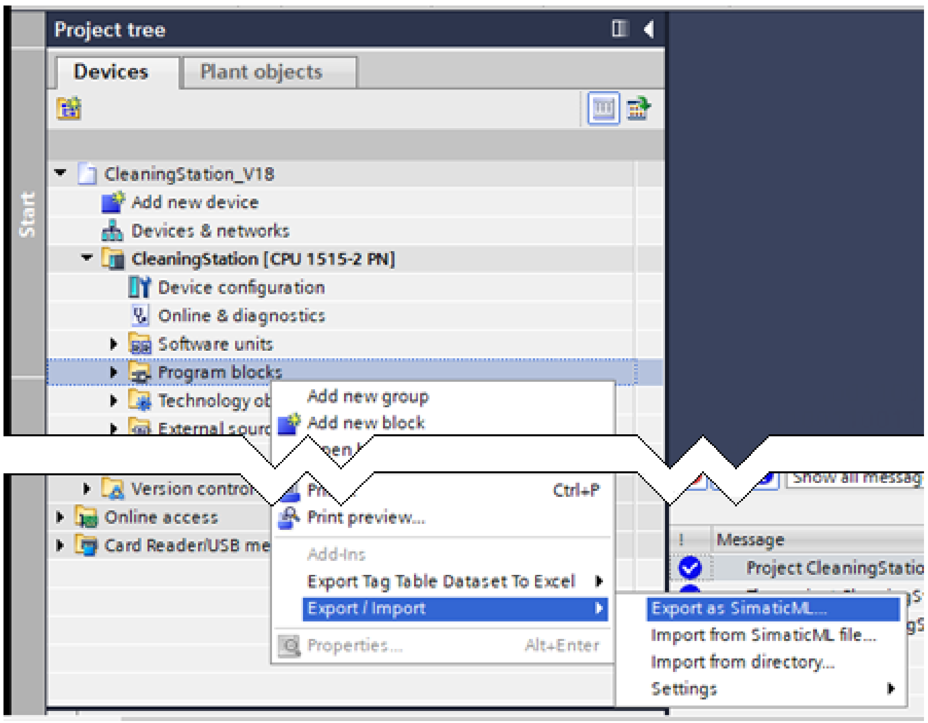
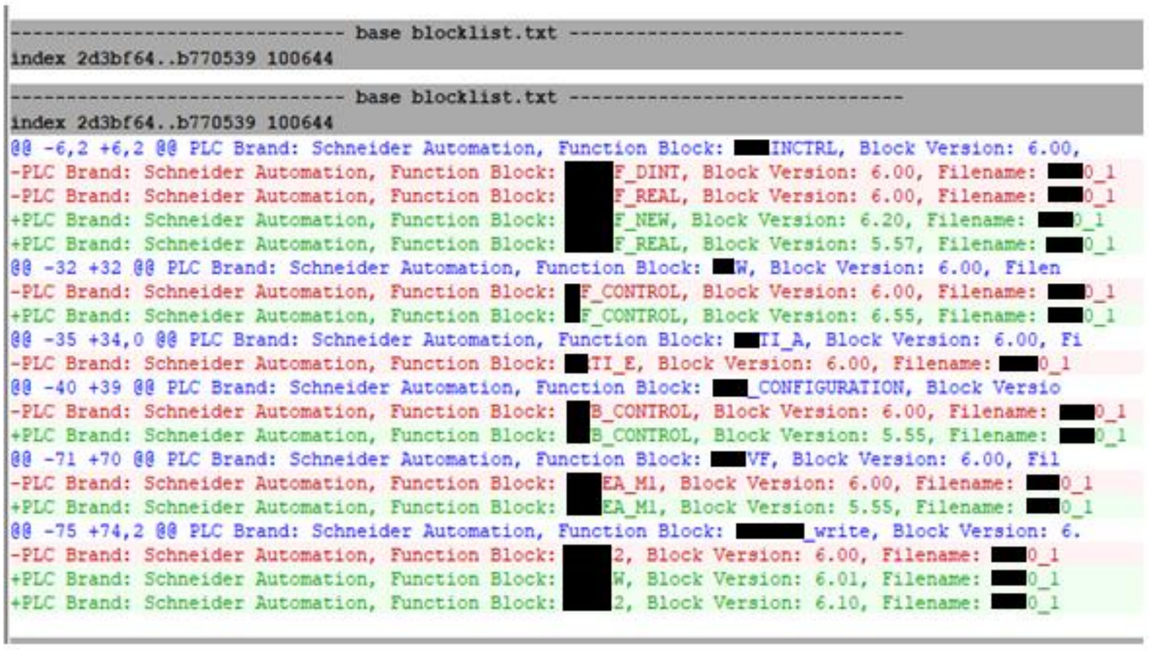


Figure 11 Export/import addin.

10 File comparison tools

In fulfilling one of Veo's requirements for rapid comparison between a backup project and the finalized project, file comparison tools become essential. While version control systems often include built-in DIFF tools for text-based code, the use of XML necessitates a distinct comparator capable of handling XML formats and filtering irrelevant data. Especially with the focus on extracting function blocks and their respective versions.



```
------------------------------ base blocklist.txt ------------------------------
index 2d3bf64..b770539 100644
------------------------------ base blocklist.txt ------------------------------
index 2d3bf64..b770539 100644
@@ -6,2 +6,2 @@ PLC Brand: Schneider Automation, Function Block: [redacted]INCTRL, Block Version: 6.00,
-PLC Brand: Schneider Automation, Function Block: [redacted]F_DINT, Block Version: 6.00, Filename: [redacted]0_1
-PLC Brand: Schneider Automation, Function Block: [redacted]F_REAL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_NEW, Block Version: 6.20, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_REAL, Block Version: 5.57, Filename: [redacted]0_1
@@ -32 +32 @@ PLC Brand: Schneider Automation, Function Block: [redacted]W, Block Version: 6.00, Filen
-PLC Brand: Schneider Automation, Function Block: [redacted]F_CONTROL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_CONTROL, Block Version: 6.55, Filename: [redacted]0_1
@@ -35 +34,0 @@ PLC Brand: Schneider Automation, Function Block: [redacted]TI_A, Block Version: 6.00, Fi
-PLC Brand: Schneider Automation, Function Block: [redacted]TI_E, Block Version: 6.00, Filename: [redacted]0_1
@@ -40 +39 @@ PLC Brand: Schneider Automation, Function Block: [redacted]_CONFIGURATION, Block Versio
-PLC Brand: Schneider Automation, Function Block: [redacted]B_CONTROL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]B_CONTROL, Block Version: 5.55, Filename: [redacted]0_1
@@ -71 +70 @@ PLC Brand: Schneider Automation, Function Block: [redacted]VF, Block Version: 6.00, Fil
-PLC Brand: Schneider Automation, Function Block: [redacted]EA_M1, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]EA_M1, Block Version: 5.55, Filename: [redacted]0_1
@@ -75 +74,2 @@ PLC Brand: Schneider Automation, Function Block: [redacted]_write, Block Version: 6.
-PLC Brand: Schneider Automation, Function Block: [redacted]2, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]W, Block Version: 6.01, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]2, Block Version: 6.10, Filename: [redacted]0_1
```

Figure 12 Git – changes to .CSV file shown in Gits diff tool.

GIT's own diff tool is good for comparing changes between commits in text-based files and offers a quick overview of changes. More detailed comparison requires the use of a third-party file comparison tool that supports more file formats and has more features. Extensive testing on both proprietary and open-source tools reveals that most proprietary tools can compare XML files based on attributes and elements rather than performing a simple line-by-line comparison. This would simplify and enable the process of extracting relevant function block data from the exported XML files.

Open-source comparison tools often operate on a line-by-line basis. This approach makes comparison of XML files challenging, as it does not consider the hierarchical attribute nature of XML data. Despite its line-by-line comparison limitation Winmerge was chosen for testing as a viable option due to its compatibility to integrate with version control systems. Challenges arose instantly during the comparison of exported XML files from PLC programs. Highlighting the need for further investigation and potential solutions with the use of open-source file comparison tools.

11 Problems

The problems arise when comparing exported XML files due to the significant restructuring that occurs with minor changes. Even with these complications, XML is chosen as the preferred file format for version control because it can be manipulated more smoothly by tools like GIT and XML parsers. This becomes especially essential when dealing with Programmable Logic Controller (PLC) code which is graphical or binary and cannot be easily manipulated outside the specialized programming environments that both Siemens and Schneider use.

The figure below presents the changes when comparing two plc project files where only a couple of function blocks were changed in some minor way. The whole XML code gets restructured which traditional line-by-line comparison tools struggle to handle.

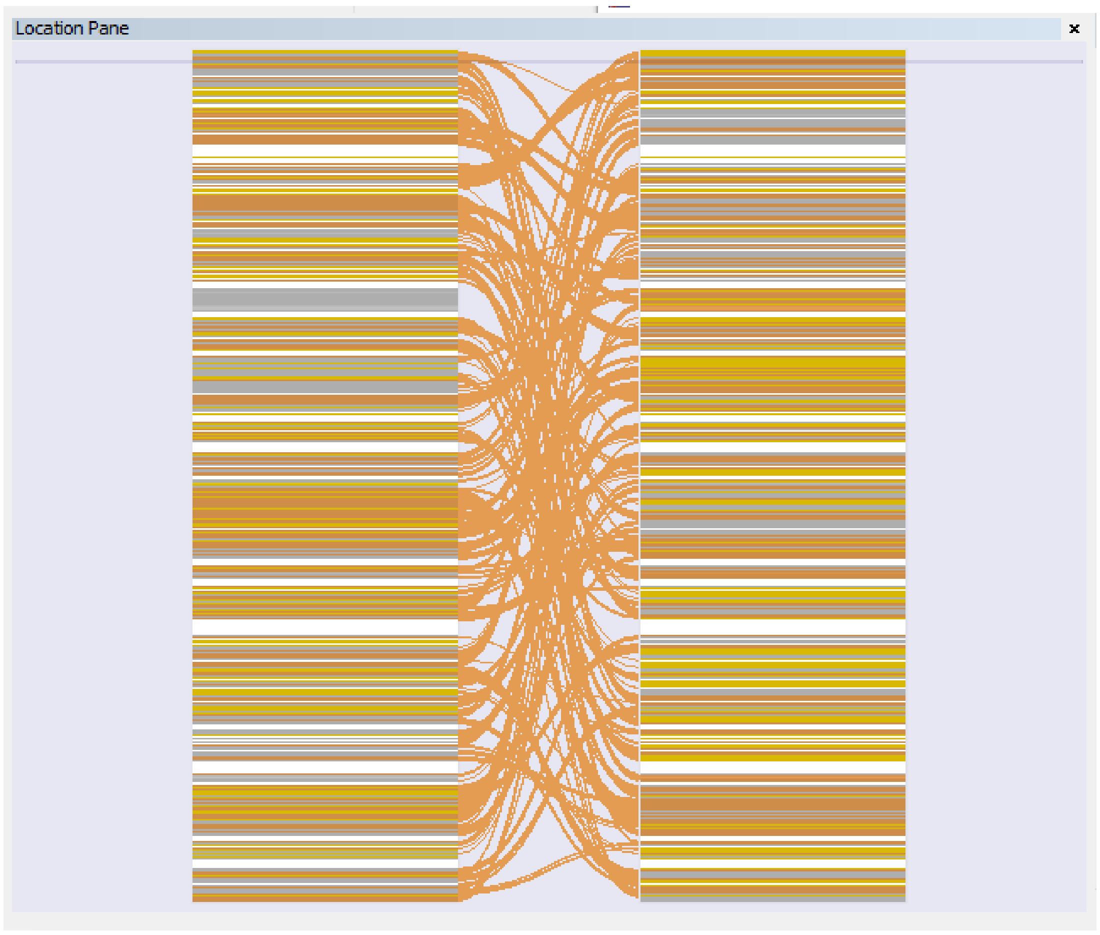


Figure 13 Winmerge location panel

To address these limitations a practical solution involves using the exported XML as a common format and using specialized XML comparison tools that can compare elements and attributes by their name and code instead of line-by-line. There are proprietary licensed tools that can parse XML code based on attributes and elements like XMLspy, OxygenXMLdiff, and BeyondCompare.

A specialized XML parsing tool is also developed for testing purposes that can create a comprehensive table list from the parsed XML code. This list is aimed to be used for easy visual comparison and manual manipulation of function block attributes. Integration with databases and version control systems becomes easier with a text-based list of blocks.

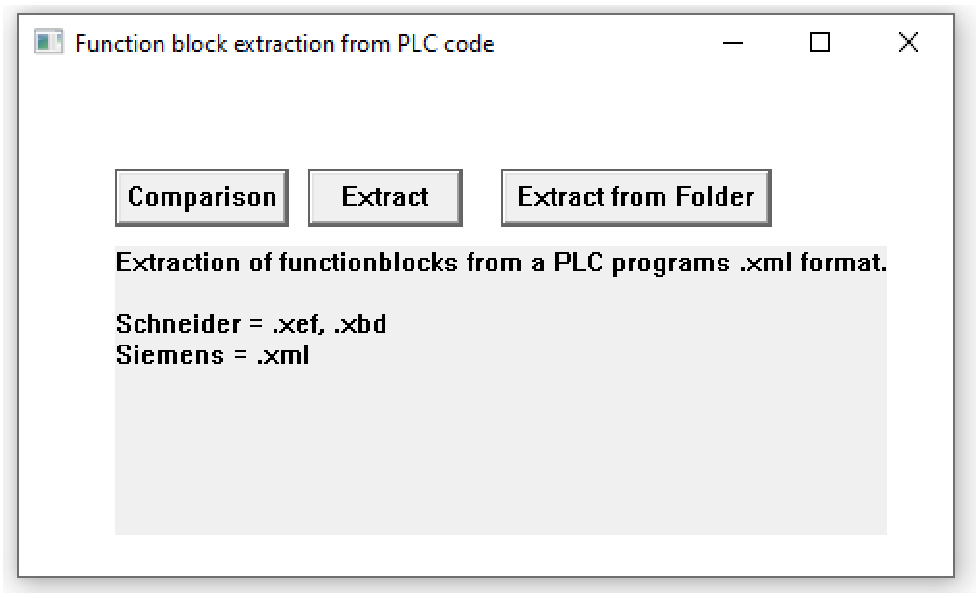
12 Custom XML parsing tool

Traditional file comparison tools struggle to effectively handle XML code generated from a PLC program. due to their line-by-line approach. This limitation becomes apparent when minor changes in the PLC code like moving an FBD by a small amount result in significant restructuring in the exported XML format.

To address this issue and to prove the concept of this solution, a custom XML parsing tool has been developed using C++. This tool is specifically designed to compare PLC blocks and code. Capable of parsing the XML code and extracting the specific elements and attributes. Mitigating the limitations and enabling tracking of modifications at the function block level. Offering an easier visualization of changes and improved importing capabilities of the blocklists. The resulting list is presented in .csv format which allows for easy integration with software like Excel or databases for future applications.

The development of this tool was significantly aided by solutions and insights found on websites like Stack Overflow and GitHub forums. These platforms have been invaluable in offering an overabundant amount of code snippets, tips, and troubleshooting advice. The tool is pieced together with solutions and tips found on these forums. However, due to the vast collection of contributions and the iterative nature of coding. Referring to individual pieces of code within this thesis becomes impractical.

It is important to note that this tool has been developed primarily as a proof of concept. The aim was to demonstrate the feasibility of a more intuitive approach to comparing PLC code changes by focusing on function blocks within XML files.



Extraction of functionblocks from a PLC programs .xml format.

Schneider = .xef, .xbd
Siemens = .xml

Figure 14 Custom XML parsing tool

12.1 XML parsing libraries

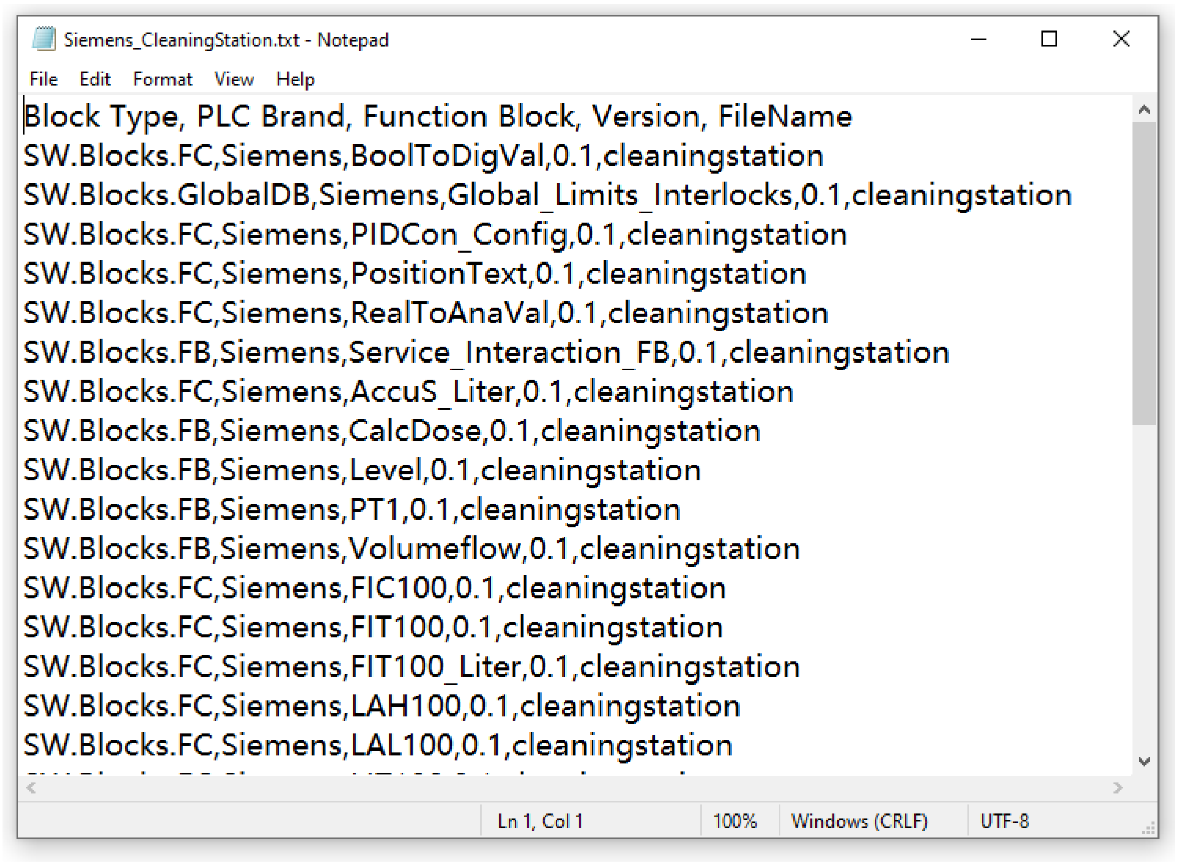
The use of custom XML parsing tools offers a range of functionalities aimed at enhancing PLC code version control and enabling other data manipulation tasks. Especially when exporting and managing configurations in XML from different environments like Schneider and Siemens. There are many XML parsing libraries available, and the choice of a library often depends on the programming environment and language used for the specific project.

Among all the libraries available for parsing XML code tinyXML2 is chosen for its compatibility with C++ and simplicity. It requires minimal overhead for quick and easy integration into existing systems without sacrificing performance.

The primary function of this tool involves parsing XML formats of PLC code to extract all relevant types of blocks. By including all attributes connected to these blocks such as version, PLC brand, and origin, all critical information gets captured and maintained throughout the version control process. This becomes essential for maintaining the integrity of PLC programs when manipulating data outside their own programming environment.

12.2 CSV output

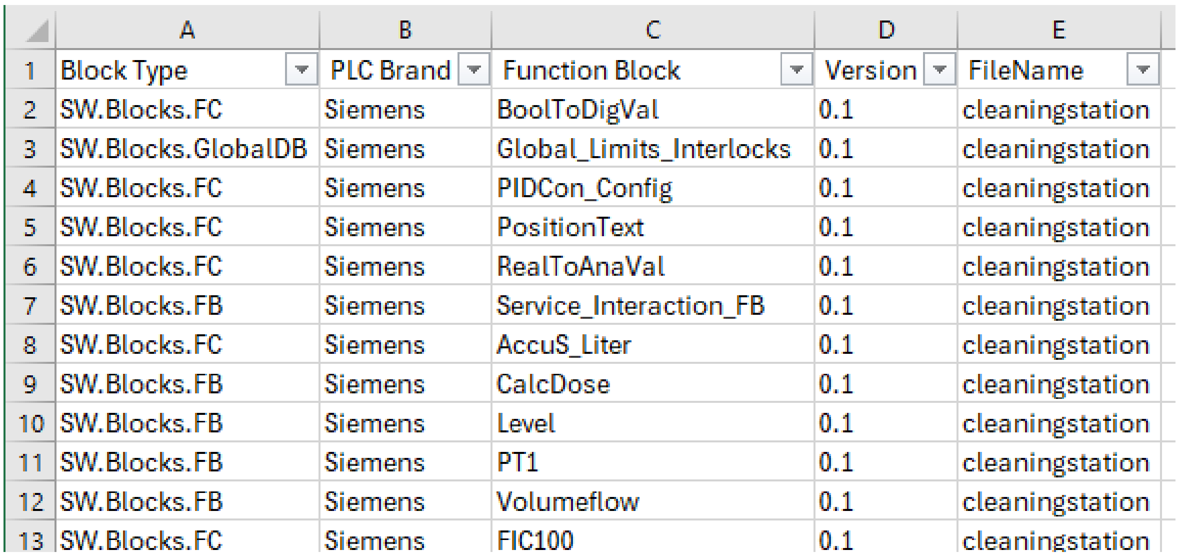
The tool generates a .CSV file listing all function blocks and other block types with the help of the TinyXML2 library. By iterating through all attributes and elements we can extract the relevant data for version controlling the function blocks. This simplifies the manipulation and automation with databases or version control systems straightforward.


Siemens_CleaningStation.txt - Notepad
File Edit Format View Help
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

```
Block Type, PLC Brand, Function Block, Version, FileName
SW.Blocks.FC,Siemens,BoolToDigVal,0.1,cleaningstation
SW.Blocks.GlobalDB,Siemens,Global_Limits_Interlocks,0.1,cleaningstation
SW.Blocks.FC,Siemens,PIDCon_Config,0.1,cleaningstation
SW.Blocks.FC,Siemens,PositionText,0.1,cleaningstation
SW.Blocks.FC,Siemens,RealToAnaVal,0.1,cleaningstation
SW.Blocks.FB,Siemens,Service_Interaction_FB,0.1,cleaningstation
SW.Blocks.FC,Siemens,AccuS_Liter,0.1,cleaningstation
SW.Blocks.FB,Siemens,CalcDose,0.1,cleaningstation
SW.Blocks.FB,Siemens,Level,0.1,cleaningstation
SW.Blocks.FB,Siemens,PT1,0.1,cleaningstation
SW.Blocks.FB,Siemens,Volumeflow,0.1,cleaningstation
SW.Blocks.FC,Siemens,FIC100,0.1,cleaningstation
SW.Blocks.FC,Siemens,FIT100,0.1,cleaningstation
SW.Blocks.FC,Siemens,FIT100_Liter,0.1,cleaningstation
SW.Blocks.FC,Siemens,LAH100,0.1,cleaningstation
SW.Blocks.FC,Siemens,LAL100,0.1,cleaningstation
```

Figure 15 Blocklist in .CSV format

By Leveraging the .CSV list of function blocks, the user can implement them in Excel and databases for rapid information extraction. This also makes the automation of library environments easier, allowing for quick identification and extraction of XML code for specific function blocks along with their versions. Therefore, smoothing the updating process of outdated PLC projects or troubleshooting the PLC code by comparing it to the backups.

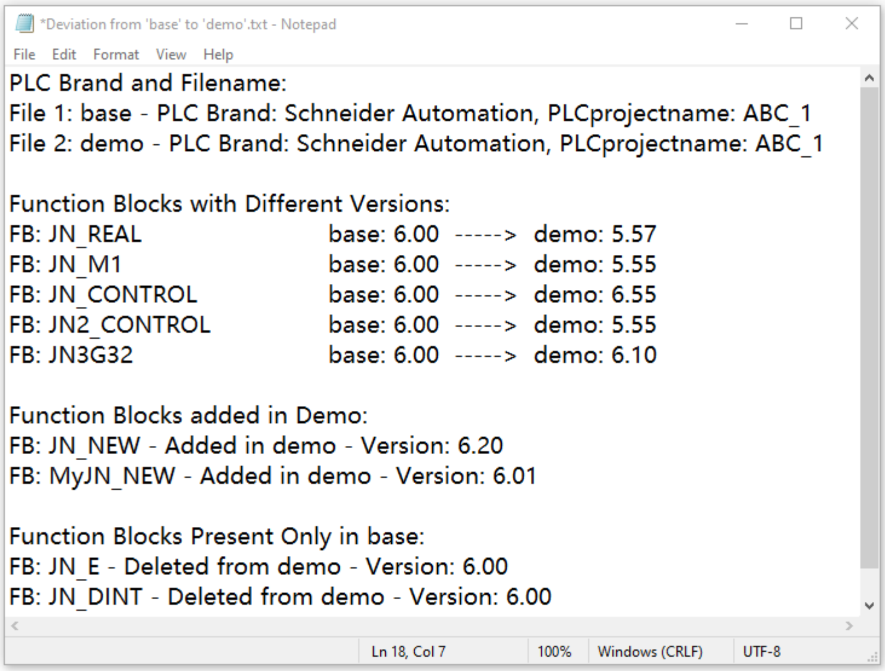


	A	B	C	D	E
1	Block Type	PLC Brand	Function Block	Version	FileName
2	SW.Blocks.FC	Siemens	BoolToDigVal	0.1	cleaningstation
3	SW.Blocks.GlobalDB	Siemens	Global_Limits_Interlocks	0.1	cleaningstation
4	SW.Blocks.FC	Siemens	PIDCon_Config	0.1	cleaningstation
5	SW.Blocks.FC	Siemens	PositionText	0.1	cleaningstation
6	SW.Blocks.FC	Siemens	RealToAnaVal	0.1	cleaningstation
7	SW.Blocks.FB	Siemens	Service_Interaction_FB	0.1	cleaningstation
8	SW.Blocks.FC	Siemens	AccuS_Liter	0.1	cleaningstation
9	SW.Blocks.FB	Siemens	CalcDose	0.1	cleaningstation
10	SW.Blocks.FB	Siemens	Level	0.1	cleaningstation
11	SW.Blocks.FB	Siemens	PT1	0.1	cleaningstation
12	SW.Blocks.FB	Siemens	Volumeflow	0.1	cleaningstation
13	SW.Blocks.FC	Siemens	FIC100	0.1	cleaningstation

Figure 16 Excel view of the .CSV file.

12.3 Block Comparison

An additional feature of the tool is the ability to compare two XML files directly, highlighting differences at the function block level. It detects changes such as additions, deletions, and modifications, including alterations in comments or version numbers associated with function blocks. A small demonstration file was created where versions of the function blocks were changed, and a couple of blocks were deleted and added from the base file.



*Deviation from 'base' to 'demo'.txt - Notepad
File Edit Format View Help
Ln 18, Col 7
100%
Windows (CRLF)
UTF-8

```
PLC Brand and Filename:
File 1: base - PLC Brand: Schneider Automation, PLCprojectname: ABC_1
File 2: demo - PLC Brand: Schneider Automation, PLCprojectname: ABC_1

Function Blocks with Different Versions:
FB: JN_REAL                  base: 6.00  ----->  demo: 5.57
FB: JN_M1                    base: 6.00  ----->  demo: 5.55
FB: JN_CONTROL               base: 6.00  ----->  demo: 6.55
FB: JN2_CONTROL              base: 6.00  ----->  demo: 5.55
FB: JN3G32                   base: 6.00  ----->  demo: 6.10

Function Blocks added in Demo:
FB: JN_NEW - Added in demo - Version: 6.20
FB: MyJN_NEW - Added in demo - Version: 6.01

Function Blocks Present Only in base:
FB: JN_E - Deleted from demo - Version: 6.00
FB: JN_DINT - Deleted from demo - Version: 6.00
```

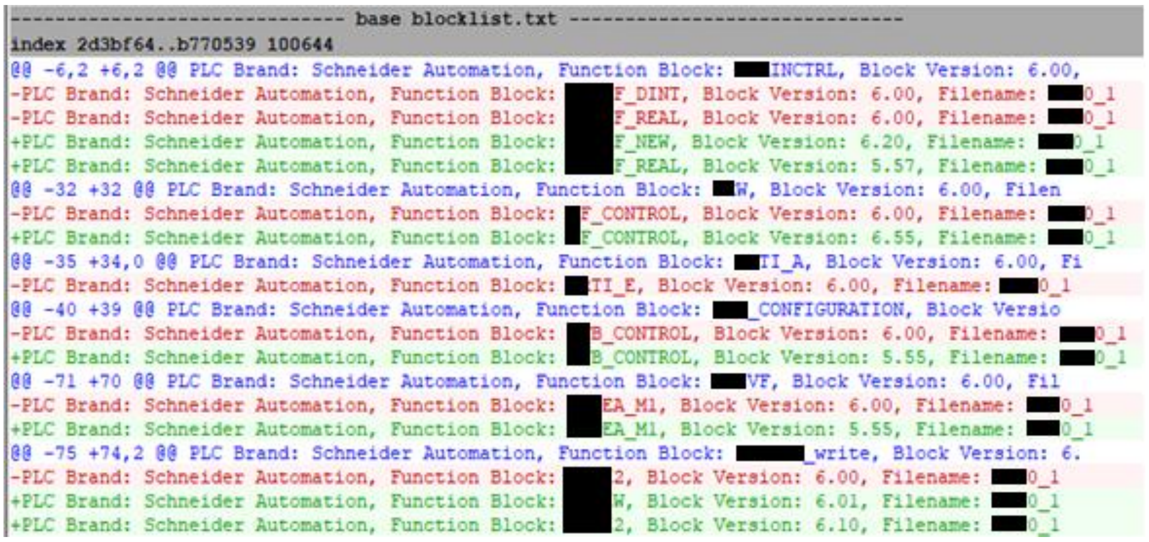
Figure 17 Block comparison.

The primary objective of this comparison functionality is to aid in the troubleshooting of customer code by comparing the finalized product against Veos archived backups. Through this comparison, changes are documented in a .txt format. Providing a clear and immediate understanding of any manipulations or modifications to function blocks. Preventing the need to check every function block individually and compare what has changed.

12.4 Integration of Blocklist for version controlling

In the pursuit of a powerful version control system for PLC code with open-source solution or the use of a database for storing and versioning blocks, the integration of a blocklist derived from a custom XML parsing tool proves effective. The generated .CSV file storing comprehensive information about function blocks and other block types serves as an important component for effective versioning.

While open-source version control systems can handle PLC code in one way or another, the ability to manipulate or extract data from the PLC files is not possible without their own programming environment. By generating a blocklist in .CSV format, the user can directly integrate the list with databases or libraries for easy querying and importing. This integration allows diff tools in GIT and other version control systems to quickly show changes visually to the user without the need for a third-party file comparator, while XML code can also be compared directly. The .CSV provides quicker visualization thanks to its compatibility with built-in diff tools that work with line-by-line comparison.



```
----------------------------- base blocklist.txt ------------------------------
index 2d3bf64..b770539 100644
@@ -6,2 +6,2 @@ PLC Brand: Schneider Automation, Function Block: [redacted]INCTRL, Block Version: 6.00,
-PLC Brand: Schneider Automation, Function Block: [redacted]F_DINT, Block Version: 6.00, Filename: [redacted]0_1
-PLC Brand: Schneider Automation, Function Block: [redacted]F_REAL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_NEW, Block Version: 6.20, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_REAL, Block Version: 5.57, Filename: [redacted]0_1
@@ -32 +32 @@ PLC Brand: Schneider Automation, Function Block: [redacted]W, Block Version: 6.00, Filen
-PLC Brand: Schneider Automation, Function Block: [redacted]F_CONTROL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]F_CONTROL, Block Version: 6.55, Filename: [redacted]0_1
@@ -35 +34,0 @@ PLC Brand: Schneider Automation, Function Block: [redacted]TI_A, Block Version: 6.00, Fi
-PLC Brand: Schneider Automation, Function Block: [redacted]TI_E, Block Version: 6.00, Filename: [redacted]0_1
@@ -40 +39 @@ PLC Brand: Schneider Automation, Function Block: [redacted]_CONFIGURATION, Block Versio
-PLC Brand: Schneider Automation, Function Block: [redacted]B_CONTROL, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]B_CONTROL, Block Version: 5.55, Filename: [redacted]0_1
@@ -71 +70 @@ PLC Brand: Schneider Automation, Function Block: [redacted]VF, Block Version: 6.00, Fil
-PLC Brand: Schneider Automation, Function Block: [redacted]EA_M1, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]EA_M1, Block Version: 5.55, Filename: [redacted]0_1
@@ -75 +74,2 @@ PLC Brand: Schneider Automation, Function Block: [redacted]_write, Block Version: 6.
-PLC Brand: Schneider Automation, Function Block: [redacted]2, Block Version: 6.00, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]W, Block Version: 6.01, Filename: [redacted]0_1
+PLC Brand: Schneider Automation, Function Block: [redacted]2, Block Version: 6.10, Filename: [redacted]0_1
```

Figure 18 GIT diff tool - changes done to an .CSV file.

13 Conclusion

A combination of GIT as version controlling system and a database for managing function blocks proves the most optimal solution in the thesis. While a database would suffice for the purposes of handling and manipulating function blocks between different projects, integrating GIT guarantees correct versioning and easier troubleshooting processes. By utilizing XML code both systems can manipulate the PLC data without the need for the respective PLC brands‘ own programming environment.

Despite all PLCS brands adhering to the IEC61131-3 standard. The way exported XML code is structured varies significantly between manufacturers and programming environments. Even if databases and GIT can handle XML formats from the get-go. A blocklist in .CSV format that is generated by a proprietary or self-developed XML parsing tool provides a standardized structure for easy comparison and swapping of blocks between projects.

GIT provides an accessible entry point into version control with substantial scalability benefits. Even though full-scale project version control may not be mandated by industry standards or explicitly required by VEO. Considering the crucial role of PLCs in managing essential infrastructure critical to human safety. It is important to anticipate potential future legal requirements for firm version control. This provides verifiable documentation that could be vital in legal scenarios.

Proprietary licensed systems such as Copia and Octoplant prove best suited for version control and management of PLC components outside PLC brands’ own programming environment. However, when the focus narrows down specifically to function blocks then the extensive features offered by these systems become unnecessary and may not justify the cost. The advantages of using these systems do not outweigh their expenses in this case. Making GIT with database management a more cost-effective option in this situation even with some unconventional solution needed for manipulation of PLC code.

14 Discussion

Proprietary version control systems like Copia and Octoplant had impressive feature sets that align closely with the functional requirements for function block version comparison and other data integrations with libraries. However, the escalating licensing costs associated with the inclusion of multiple developers and PLCs present a significant problem. Prompting a shift towards more cost-effective, albeit less conventional solutions utilizing open-source tools. While these open-source alternatives significantly mitigate the financial burden by offering low to no-cost options. They introduce complexities in terms of setup and integration. The trade-off here lies in balancing the financial advantages against the potential reduction in system simplicity and flexibility.

This proves a broader issue that development for PLC systems is lagging compared to traditional computers, especially in version control. Traditional software development benefits from the mature ecosystem of version control tools such as GIT which are user-friendly, mostly free, and well-integrated. Meanwhile, specialized, and effective version control for PLC systems depends on expensive proprietary software that smaller teams and hobbyists may find costly. This discourages smaller entities from adopting correct version control practices and often resort to rudimentary methods like manually saving project versions with new dates and storing them in folders on a computer. This not only increases the risk of error but also discourages collaboration, traceability, and scalability.

While versioning of PLC code is possible with open-source solutions Like GIT, direct manipulation and visualization become impractical outside the PLC programming environments. Unconventional solutions need to be implemented when comparing files which includes exporting and importing different formats back and forth. While these solutions work it does not mean manipulation of plc code for versioning needs to be this tedious and time-consuming.

The integration of the blocklist into version control systems and the use of XML circumvents the limitations of versioning plc code by providing a human-readable representation. While this surrogate representation enables version control operations within both open-source and proprietary environments, multiple steps and liabilities are introduced. Version control of PLC code should be as straightforward and simple as possible.

15 List of references

Ayvesy-mdt. (3 3 2024). *Full control for production plants and industrial automation with octoplant*. Retrieved from https://auvesy-mdt.com/en/octoplant

BasuMallick, C. (6 10 2022). *What Is Version Control? Meaning, Tools, and Advantages*. Retrieved from https://www.spiceworks.com/tech/devops/articles/what-is-version-control/

Dietrich, S. (13 10 2022). *Managing Multi-user PLC Programming: Git Version Control* . Retrieved from https://control.com/technical-articles/managing-multi-user-plc-programming-git-version-control/

Ernst, M. (1 9 2012). *Version control concepts and best practices*. Retrieved from https://homes.cs.washington.edu/~mernst/advice/version-control.html

Henry, D. (12 7 2022). *Understanding Git-Based Version Control for Industrial Automation*. Retrieved from https://www.automation.com/en-us/articles/july-2022/git-based-version-control-industial-automation

IBM. (1 5 2024). *What is a NoSQL database?* Retrieved from https://www.ibm.com/topics/nosql-databases

inductiveautomation.com. (24 02 2020). Retrieved from https://inductiveautomation.com/resources/article/what-is-a-PLC

Lamb, F. (7 2023). *ControlEngineering.* Retrieved from https://www.controleng.com/articles/plc-programming-language-fundamentals-for-improved-operations-maintenance/

N Deepa, K. L. (2020). An Analysis on Version Control Systems. *International conference on emerging trends in information Technology* (s. 9). Researchgate.

Peter. (13 3 2018). *plcacademy.com*. Retrieved from https://www.plcacademy.com/function-block-diagram-programming/

plctable. (4 3 2024). *PLC vs. PC*. Retrieved from https://www.plctable.com/plc-vs-pc/

Rahul Awati, A. H. (3 2024). *Microsoft SQL Server*. Retrieved from https://www.techtarget.com/searchdatamanagement/definition/SQL-Server

Schneider Electric India. (21 7 2022). *www.se.com.* Retrieved from https://www.se.com/in/en/faqs/FA225557/

Scott Chacon, B. S. (2014). *Pro Git.* USA: Apress. Retrieved from https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Siemens. (05 2021). *Industry Support Siemens.* Retrieved from https://cache.industry.siemens.com/dl/files/533/109798533/att_1069908/v1/TIAPortalOpennessenUS_en-US.pdf

Smith, P. (11 10 2022). *Using the Version Control Interface (VCI) in TIA Portal V16.* Retrieved from https://www.dmcinfo.com/latest-thinking/blog/id/10380/using-the-version-control-interface-vci-in-tia-portal-v16

Varela, L. R. (1 2022). *A Study of Version Control System in Software Development Management Concerning PLC Environments.* Portugal: Researchgate.net.

Wright, M. (7 October 2020). *Bloom Institute of Technology*. Retrieved from https://www.bloomtech.com/article/version-control-vocabulary