Tampere University of Applied Sciences

Implementing Customer-Specific Test Automation

Sabita Manandhar

BACHELOR'S THESIS
June 2025

Bachelor's Degree Programme in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Software Engineering

MANANDHAR, SABITA:
Implementing Customer-Specific Test Automation

Bachelor's thesis 37 pages
June 2025

Continuous Integration (CI) pipelines are essential for modern software development in multiple ways. Specifically, when it comes to quality assurance of the software, CI pipelines have ability to run tests on integrated code changes. Issues that might have been missing during the manual testing can be detected in the test automation execution in the CI pipeline runs. This thesis explores and design to enhance CI pipelines within Azure Pipelines, aiming to make the software testing process more automated and efficient for the modified configuration according to the customer requirements. This will eventually reduce the time and effort required by software testers to execute repetitive testcases in every regression cycle.

The knowledge for this thesis was acquired through the personal experience and extensive research of the products and services in the Glaston company. Various tools were integrated along with Azure pipeline, such as Bitbucket, TwinCAT3, PyCharm, Inspect, Robot Framework, and more. The required tools and technologies were learnt and practised gradually. In addition to the tools, the agile way of working in the company, integration of different tools, code merge culture, etc, were experienced. Communication with respective team deepened an understanding about the research topic. Internal documentation was studied to get familiar with the subject. The existing testcases were also studied and understood in test management tool which provided detail information about the use cases, its test steps, test data and methods to achieve the results.

The development process started with creation of the testcases and merging code to main branch in Bitbucket. The existing Azure CI pipeline was updated to support the automated builds. After a running the pipeline, the necessary test results were successfully listed for review and debugging. This initial setup improved the development workflow and created a base for future improvements. It was discussed with the team that further advancements will be done and the setup will be implemented for the monthly regression testing cycle of the company.

Key words: software testing, test automation, continuous integration, robot framework, azure pipeline

CONTENTS

ABBREVIATIONS AND TERMS

CI/CD	Continuous Integration and Continuous Deployment
Clone	Downloading a remote repository to local machine to work offline.
CPU	Central Processing Unit
IDE	Integrated Development Environment
MCT	Machine Configuration Tool or a machine configuration (config) file that includes settings and parameters to configure a software application.
PAE	Project Automation Engineering
PLC	Programmable Logic Controller
PR	Pull Request
QA	Quality Assurance
RDBMS	Relational Database Management System
Regression testing	A testing process that verifies new updates and bug fixes in the code do not create issue in the existing functionality.
RPA	Robotic Process Automation
Smoke testing	A preliminary testing method that verifies the basic functionality of a software build before more detailed testing is conducted.
TwinCAT	The Windows Control and Automation Technology
UI	User Interface
VCS	Version Control System
WinAppDriver	Window Application Driver
XA	eXtended Automation

1 INTRODUCTION

Software development is the process of building software applications or products. It is a challenging process because developers are involved in the complete software development process and finally create innovative software, but also, they should ensure that the product meets quality adapting to the changing requirements (GeeksforGeeks, 2024). Creating a perfect software on the first time is nearly impossible, except in very simple cases. Most sophisticated software will require testing, adjustments, and improvements. Thus, software testing is a critical final step in the development process because it helps to find errors before the software is released and deployed in the customer's environment. It ensures the software works as expected and meets quality standards before the product is ready for operation (Woodward & Hennell, 2005). Also, Jia (2023) has stated that software testing is a systematic process of evaluation, inspection and verification of the software system. Through testing, defects introduced during the software development or maintenance phase can be detected and corrected. The goal is to ensure the software system is developed and delivered according to user requirements and specifications.

Software applications are growing and becoming more complex. This leads to increase in workload, consumes more time and gets repetitive for the software testing. It creates challenge in improving the efficiency and quality of software testing. To address this, test automation has been developed, integrating tools and standardized frameworks to enhance the software testing process. The main goal of development of software test automation is to create and run test cases using automated tools, which makes the testing process faster and more efficient (Jia, 2023).

Automation testing is a technique where software testers write scripts and use tools to test applications automatically. This process mimics manual testing but is done by machines where the repetitive tasks are executed without human intervention. Automation testing is particularly useful for tasks that are time-consuming to perform manually because it can efficiently run large quantity of test-cases in short period of time. Tests can be executed at any time using scripted

sequences to check the software's functionality. Automation tests can input test data, compare expected results with actual outcomes, and generate detailed reports. Additionally, test suites can be recorded and replayed as needed, making the testing process more efficient and reliable. (Mohammad, 2015; Berga, 2024). When integrated with specialized tools like Jenkins, CircleCI, Travis-CI, GitLab, GitHub Actions, Bamboo, or Azure Pipelines, it evolves from being a separate phase to a continuous, essential part of the development process. This integration means the application is consistently tested and validated automatically in the desired time, leading to faster releases and higher-quality products. This seamless workflow helps organizations deliver reliable software to users more rapidly and with greater confidence (GeeksforGeeks, 2025). It also helps teams to maintain quality standard and improve test coverage (Berga, 2024).

As DevOps becomes more popular with automation, continuous integration and continuous delivery (CI/CD) tools, software testing is starting to align more closely towards it. DevOps facilitates with testing by running tests and providing feedback on the quality of the code. It helps find defects and errors early, reducing the risk of problems in production and improving the overall quality of the software (Jia, 2023). Azure Pipelines are a key part of Microsoft’s Azure DevOps suite. They provide a powerful platform to automate and manage pipelines. These pipelines help to meet the growing need for efficiency and scalability in the software development process. It combines CI, CD, and continuous testing to build, test, and deliver code within the predefined time. This kind of automation helps team to find issues faster and release new features quicker (Soma, 2024).

This thesis focuses on developing testcases that can automate the feature validation before they are delivered to the end customers. Also, it focuses on developing the CI pipelines which can test the automation and provide the results. These improvements will help the team to work faster as well as effectively as the test automation will solve the problem of repetitive manual work with reliability. This helps to provide customer-specific test results prior project deployment and to maintain the capability of testing new releases with customer configurations in a highly automated environment.

2 BACKGROUND

2.1 Company Profile

Glaston Corporation is a global company that makes machines and technology for processing glass. The company started back in 1870 as Hammarén & Comp, a forest and paper industry. The company has continually evolved by developing new technologies and international expansion. After acquiring Tamglass in the 1980s the focus was shifted towards the glass industry. In June 2007, the company is officially renamed as Glaston Corporation, focusing fully on glass technology. In 2019, Glaston acquired Bystronic glass, a well-known company from Switzerland and Germany (Our Journey – Glaston, 2023). The vision of Glaston is to lead glass processing industry by using innovative technologies with lifecycle solutions (Annual Review, 2024). It focuses on two main business areas and four business lines which are described in Table 1.

TABLE 1: Business area of Glaston (Annual Review, 2024).

Glaston	Business Area	Business Line	Specialization
Business Area	Architecture	Tempering and Laminating	Advanced heat treatment machines, services for maintenance, up-grades, modernization, and spare parts for glass flat tempering and laminating.
		Insulating Glass	High-tech machines for manufacturing insulating glass. Maintenance, upgrades, modernization services, and spare parts.
	Mobility, Display & Solar (MDS)	Pre-processing	Pre-processing technologies for the MDS glass industries as well as the related services business MDS market. Digital services, such as glass processing machine remote monitoring and fault analysis services. Consulting and engineering services.
		MDS Heat Treatment	Heat treatment technologies and services for MDS glass market.

2.2 Objective

In Glaston, Quality Assurance (QA) team and Project Automation Engineering (PAE) team are two separate teams in Tempering and Laminating business line. The description of QA and PAE team is listed in Table 2. The QA team is responsible for testing the default Machine Configuration Tool (MCT) and conducts a regression testing for those MCTs. There is a good coverage of the test automation in this team. Additionally, manual, feature and explorative testing is conducted for the remaining testcases which are not automated. On the other hand, every customer has their special MCT developed based on customer's requirements. PAE team is testing such customized MCTs configuration manually and deploys the software version in the customer's environment. The team is manually testing the MCTs that not only involves the repetitive tasks but also possibility of overlooking some issues. These issues may remain unresolved and delivered to the customer's environment. To minimize manual testing in PAE team, the test automation development and Azure pipeline solution for testing customer specific MCT is planned. This solution will help PAE team to verify the functionality in the customer's customized configuration with less human effort. A CI pipeline can be created to run test automation in each custom MCT and identify issues in advance. This will reduce the time that is spent on the repetitive tasks that should be tested in multiple MCT. Additionally, the saved time can be used in some other explorative testing and further test automation development.

The goal of this research is to explore ways to improve and simplify the CI pipelines in Azure DevOps, making the software testing process automated and more effective. For achieving the goal, following questions are researched.

- What approach can reduce the inefficiencies while testing the customer-specific configuration?
- How can the test automation assist in simplifying the validation process and reduce the quality issues before customer delivery?

Table 2: Team description

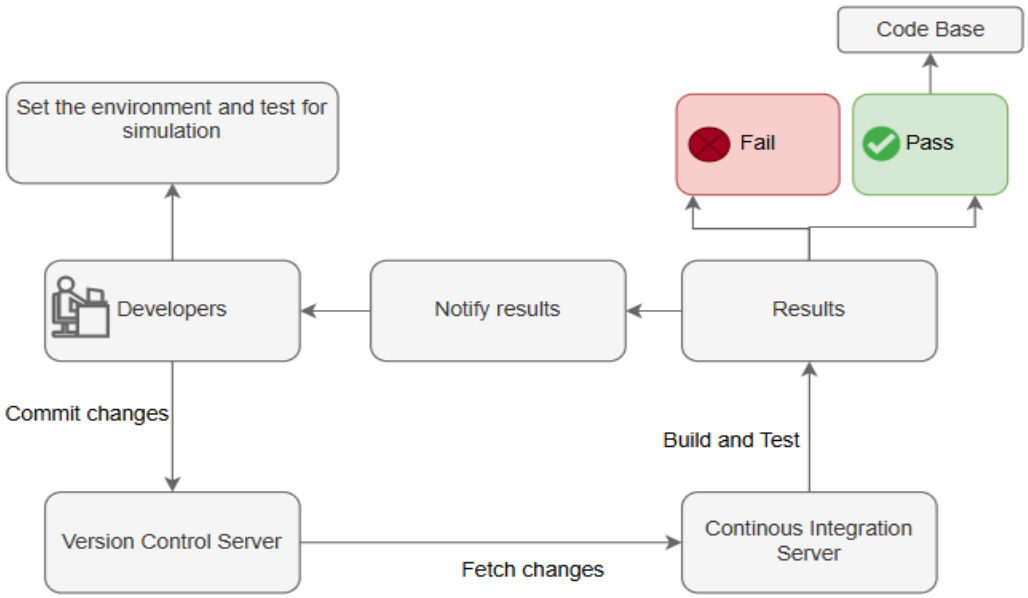
	QA Team	PAE Team
MCT	The team conducts testing in default MCT. Default MCT is a configuration of machine which is provided with .NET artifact of a particular version by the development and not intended for a specific customer or project.	The team performs testing in customized MCT. Customized MCT is a specific machine configuration designed based on customers' requirements and features provided.
Testing	Test automation is executed. The results are analysed. Failed testcases are investigated. Following testing are done for non-automated testcases: Regression and manual testing - Feature testing - Explorative testing	Following testing are done for customized MCT: - Smoke Testing - Manual
Responsibilities	After completion of testing, release the version. Then the team members work on test automation development until the next version is provided for test.	After the release from QA team, the default MCT is modified by adding or removing the machine features according to the customer requirements. After testing new version, the team deploy in the customer's environment

3 TEST AUTOMATION REQUIREMENTS

This chapter discusses the requirements and tools that are necessary to implement for test automation and CI tool. It begins with a general overview supported by a planning diagram and after that required tools are explained in detail in next sub-heading.

3.1 Overview of Requirements

For this project, a wide range of tools needs to be integrated. The overview of requirements as shown in Picture 1, provides a framework of necessary tools and steps to be implemented. The key part includes Windows environment setup attached with required applications, code editor for code development, version control systems for code management, and CI server for automation.



PICTURE 1: Test automation requirements (CircleCI, n.d., edited).

3.2 Tools And Technologies Used

3.2.1 Integrated Development Environment (IDE)

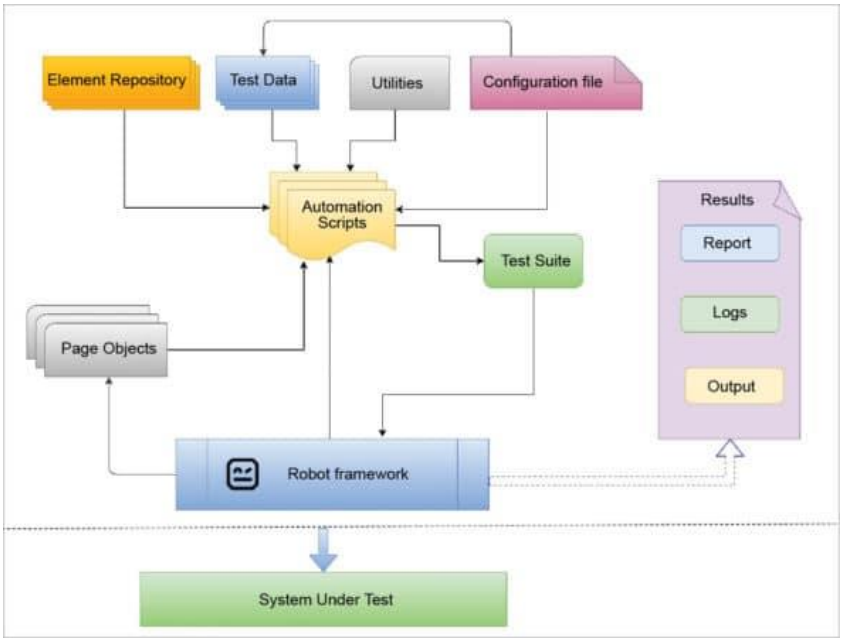
An IDE allows developers to start coding efficiently without needing extensive setup. They offer built-in support for building, debugging, editing, testing, and terminal use. IDE also provide reliability for large projects by integrating tools that work well together. Additionally, many IDEs are tailored for specific technologies,

optimizing workflows, and reducing the complexity of managing multiple languages and frameworks. Popular IDEs include JetBrains products like IntelliJ IDEA, PyCharm and WebStorm, as well as Visual Studio and Eclipse (Watson, 2024).

3.2.2 Robot Framework

Robot Framework is an open-source framework used for test automation and robotic process automation (RPA). It is widely used in many industries and supported by a Robot Framework Foundation. The framework uses simple, easy-to-understand and set of sensible rules to write Robot Framework code. It supports libraries in Python, Java and multiple other languages. It integrates well with other tools to help automate without any paid license. Thus, the tests can be run automatically after the build and deployment on a server (Robot Framework, 2019). A test runner can also execute test cases using the tags name. A well-structured report is generated that has a results summary, timestamps, recordings showing how and what was executed, search results and automatically embedded screenshots if there is a failure. Azure DevOps has the capability for scheduling tests like daily, weekly or monthly. It can integrate with a CI system so that the tests can be run automatically after the build and deployment on a server too. (Singh, 2020)

Picture 2 shows the automated testing framework using the Robot Framework that operates through a structured process. It begins with preparation, where elements, test data, utilities, and configuration settings are organized. Page objects are created to represent the application's pages. Automation scripts are then developed, defining test cases using these page objects, test data, and utilities. These test cases are grouped into a test suite for organized execution. The Robot Framework executes the test suite, interacting with the system under test to perform actions and validations. Finally, the framework generates detailed reports, logs, and outputs, documenting the test results and any issues encountered. This comprehensive process ensures thorough evaluation and documentation of the system.



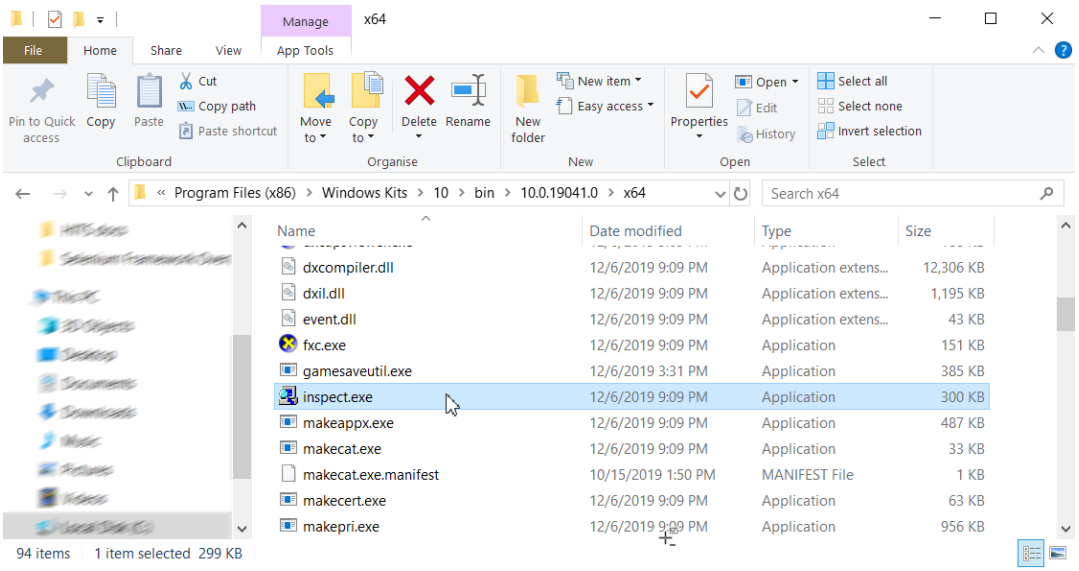
PICTURE 2: Test automation framework architecture (Singh, 2020).

3.2.3 Window Application Driver (WinAppDriver)

A Microsoft creation, WinAppDriver, is one of the most popular tools for desktop UI automation in Window applications. It helps for the test libraries API to interact with desktop application user interface that is necessary during the automated testing. It supports all kind of Windows desktop applications. (Uraizee, 2019)

3.2.4 Element Inspector

Element Inspector is a tool to locate and examine Windows UI elements in desktop applications. “Inspect.exe” or the Inspect tool can be used to find UI elements and view their properties. To inspect elements effectively, the Windows Kit or Windows SDK is required. The newest version of the Microsoft Visual Studio contains the Windows SDK and Inspect tool by default. However, it can also be manually installed from the Windows SDK site. This tool helps to find every UI element that can be queried using the WinAppDriver (Uraizee, 2019). The Inspect tool can be found under the Windows SDK folder (Picture 3) which is typically under path. *C:\Program Files (x86)\WindowsKits\10\bin\10.0.19041.0\x64*



PICTURE 3: Location of Inspect.exe tool (Uraizee, 2019).

With inspect tool, there are several strategies to access the locators for UI interaction like accessibility ID, class name, runtime ID, name attribute, tag name, and XPath expressions as listed in Picture 4. With the use of these locators and keywords, desired interactions can be performed in the UI.

Client API	Locator Strategy	Matched Attribute in inspect.exe	Example
FindElementByAccessibilityId	accessibility id	AutomationId	AppNameTitle
FindElementByClassName	class name	ClassName	TextBlock
FindElementById	id	RuntimeId (decimal)	42.333896.3.1
FindElementByName	name	Name	Calculator
FindElementByTagName	tag name	LocalizedControlType (upper camel case)	Text
FindElementByXPath	xpath	Any	//Button[0]

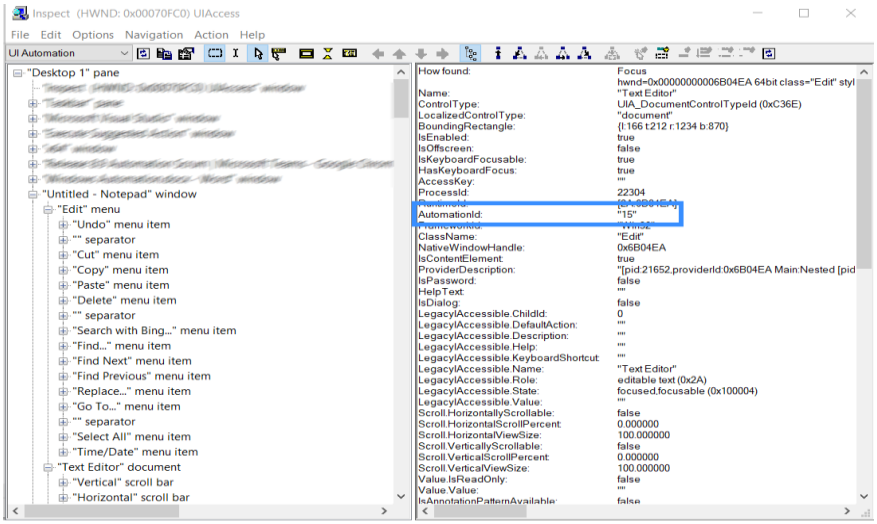
PICTURE 4: WinAppDriver supportive locators (Uraizee, 2019).

How does "inspect.exe" relate to WinAppDriver?

Inspect tool allows users to find locators and elements inside an application window. Once the attribute and its respective locators (at least one as shown in Picture 4) is accessed, users can take note of its attribute data and refer to it in testing scripts for WinAppDriver to interact with Desktop UI. For example, in the Calculator Test;

session.FindElementByXPath("//Button[@AutomationId=\"equalButton\"]").Click();

The AutomationID of the "=" can easily be found using Inspect tool with the Calculator application and applied to WinAppDriver script. (Uraizee, 2019).



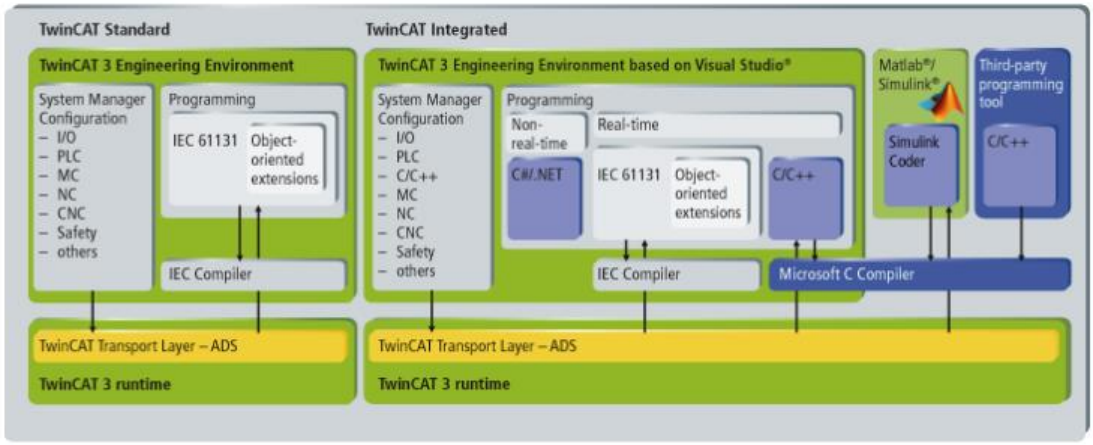
PICTURE 5: Inspect tool and the locator (Uraizee, 2019).

3.2.5 TwinCAT 3 PLC

Beckhoff revolutionized automation with its PC-based control technology, setting a global standard. Since 1996, The Windows Control and Automation Technology (TwinCAT) automation suite has been the core of this system, combining software and hardware for highly scalable and open control solutions. TwinCAT turns almost any PC into a real-time control system, supporting PLC, NC, CNC, and robotics functions. Its modular design allows easy upgrades and functional changes. The system can also integrate third-party components (Beckhoff, n.d.).

TwinCAT 3 is the latest version of the software or the updated version from TwinCAT 2 logic. It enhances Beckhoff's automation solutions fulfilling the basic principles of an open and highly scalable control system (Beckhoff, n.d.). To handle the complexity of modern machines and reduce engineering effort, there is a shift towards modular control software. This approach treats individual functions, components, or machine units as separate modules. Each module performs a specific function or controls a specific part of the machine. These modules are designed to be as independent as possible and are organized in a hierarchy. The simplest modules, which are reusable building blocks, form the base of this structure and at the bottom of the hierarchy. With standardized interfaces, these basic modules

can be combined to create more complex machine units, all the way up to a complete machine. Ideally, each module can be operated, updated, scaled, and reused on its own being independent from each other. TwinCAT 3 uses this kind of modular design, making programming easier and more flexible. It is also called eXtended Automation (XA) as it combines the newest IT-technologies, scientific software-tools and automation technology. This XA concept applies not only during the development phase but also during runtime operation. Picture 6 presents the different applications for the TwinCAT standard and integrated runtime and its compatibility with third party programming languages and the tools (Beckhoff, 2024).



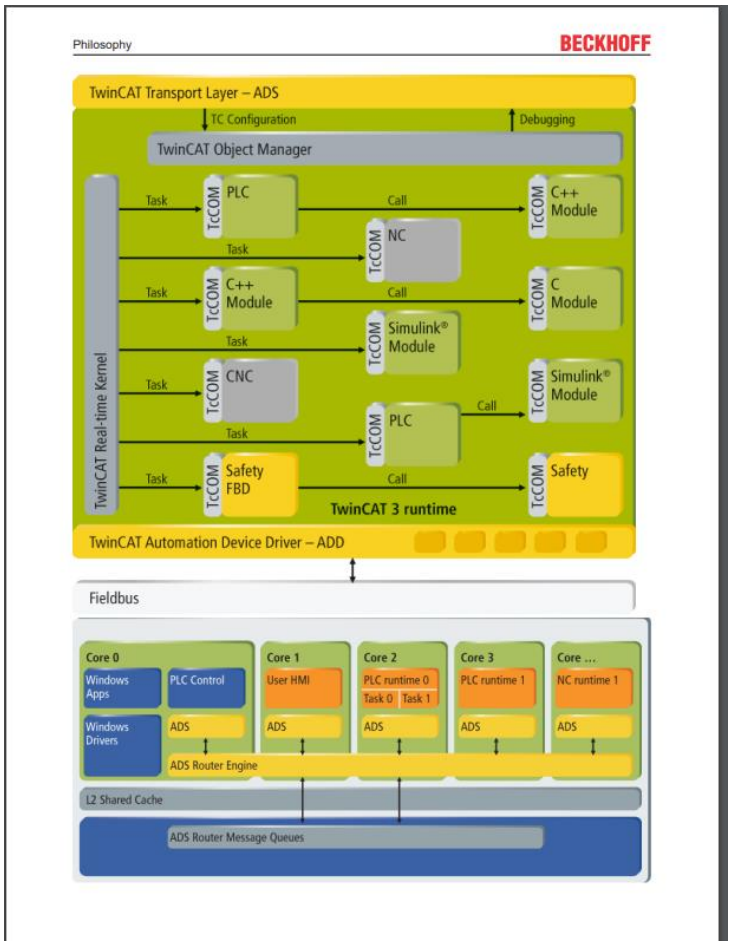
PICTURE 6: TwinCAT 3 automation framework (Beckhoff, 2024).

The TwinCAT 3 Runtime provides a real-time environment where TwinCAT modules can be loaded, run, and managed. The best part is that these modules do not have to be created using the same programming tools, so they can be developed by different developers or companies. The modules can be written in different languages or tools, like PLC programs, NC, CNC, or C/C++ code.

These modules are run in a cycle by Tasks. Multiple Tasks can run on a single control PC. Since different types of modules (like PLC, C/C++, or MATLAB) can call each other in TwinCAT 3 Runtime, this offers many possibilities for designing the software. For example, it can combine various modules with different functions to create a complete machine application. There is no limit to the number of modules that can be called by a Task, although if a module takes too long to execute, the system will make an alert.

TwinCAT 3 can handle up to 65,000 Tasks, but this depends on the available system resources of the device running the software.

Another key feature of TwinCAT 3 is its support for multicore Central Processing Unit (CPU) as highlighted in Picture 7. Individual Tasks can be assigned to different cores of the CPU. TwinCAT 3 allows to take full advantage of modern multicore Industrial and Embedded PCs for better performance.



PICTURE 7: TwinCAT 3 support for multicore CPU (Beckhoff, 2024).

3.2.6 .NET Desktop Application

.NET is a free, open-source, cross-platform software development framework that supports Linux, macOS, and Windows. It is a development platform as it provides programming languages and libraries for building applications. It supports many languages including C#, F# and tools like Visual Studio or Visual Studio Code. It provides the tools and flexibility to create a wide range of applications such as web applications and services, mobile and desktop apps, cloud-based

solutions, Internet of Things (IoT) projects, and APIs. Essentially, it supports almost any type of software development to pursue. (Regio, 2023).

A .NET desktop application can serve as an effective UI for integrating with a PLC in a machine (Abto Software, 2012). There are various types of PLCs available, such as MELSEC, Siemens, and others. Each PLC uses different communication protocols for integration. .NET supports interoperability with various PLCs through different communication protocols like TCP/IP, OPC UA, and Modbus. For example, MELSEC PLC uses TCP/IP communication. This flexibility allows seamless integration with different types of PLCs which enhances the efficiency and versatility of industrial automation systems (Syed, 2013). It promotes smooth communication between the machine's control system and the user, enabling real-time monitoring, control, and data acquisition. It allows multiple versions of the .NET runtime to run simultaneously, ensuring application stability and reducing conflicts. .NET comes with built-in security features like data encryption and role-based access, reducing the risk of data breaches and unauthorized access. Microsoft backs the .NET framework, ensuring it undergoes extensive testing and receives frequent updates and patches. It includes built-in tools for handling exceptions, managing errors, garbage collection, and memory management, which help .NET applications deal with unexpected issues effectively. (Abto Software, 2012)

3.2.7 Microsoft SQL Server 2014

Microsoft SQL Server is a relational database management system (RDBMS). That is, it is a database system which organizes data points with predefined relationships for easy access. Applications connect to a SQL Server instance or database and communicate using Transact-SQL or database engine. It is the core service for storing, processing, and securing data. The database server facilitates to migrate and data management (Microsoft, n.d.).

3.2.8 MCT

MCT is a software application used for configuring and managing machine devices, including creating XML schemas, integrating components, and linking to PLC variables. The iControL (Glaston machine software) configuration file is also

referred as the MCT. After the PLC is activated and database is attached, UI can be started integrating the configuration, MCT file.

3.2.9 Version Control System (VCS)

Version control has different names, such as revision control, source control, and source code management. Software team members can work together efficiently and faster, even if they make changes to the same codebase. VCS records every change made to the code as a database. Thus, in case of a mistake, developers can revert to a previous version of the code without any issues or merge conflicts. It is especially helpful for teams working on complex projects where multiple developers might be updating the same code at the same time from different locations. For software teams, the code is their asset. VCS makes possible to protect this code from errors, accidents, or conflicts between developers' changes. Without this system, teams might accidentally overwrite each other's work or struggle to figure out which version of the code is the most up to date. Git is one of the most widely used VCS today. Hosting services such as GitHub, GitLab, and Bitbucket are built around Git. Main benefits of VCS include:

Comprehensive History of Changes

Version control keeps a detailed record of all changes made to files, including additions, deletions, and edits, over time. This history includes information about who made the change, when it was made, and the purpose behind it. Some version control tools handle renaming and moving files better than others, but all maintain a complete log. This detailed history is invaluable for identifying and fixing bugs, especially in older versions of the software. Since software is constantly evolving, nearly every version eventually becomes an older version that may need revisiting.

Branching and Merging

Branching allows developers to work on separate features, fixes, or updates without interfering with each other. Even solo developers can benefit from this approach. By creating a branch, each stream of work remains isolated, and once complete, the changes can be merged back into the main codebase. This process ensures that conflicting changes are identified and resolved before integration.

Teams often use branching for new features or releases, and different workflows can be tailored to suit a project’s specific needs.

Tracking and Traceability

Version control tracks every change, which can be linked to project management or bug tracking tools like Jira. Developers can annotate changes with descriptions of their purpose, making it easier to understand the reasoning behind them. This level of transparency is critical for debugging, maintaining legacy code, and planning future changes. With an annotated history, developers can confidently make adjustments that align with the software's long-term goals and design. (Atlassian, n.d.).

3.2.10 Continuous Integration in Azure DevOps

Marking as the turning point in the late 20th century, programming start to shift towards modular and object oriented and allow developers to create reusable components, reuse code and simplify workflows. Modern engineering projects demand and capable of a software solutions that can adapt to rapidly changing requirements, integrate with diverse tools, and support real-time collaboration among multidisciplinary teams. Scalability, adaptability, and efficiency have become key principles in engineering software development. Scalability ensures that applications can handle increasing workloads without compromising performance. Adaptability allows software to adjust to new challenges or technological advancements. Efficiency, both in terms of computational performance and resource utilization, makes smoother workflows. The integration of these principles has led to creation of DevOps which align with modern engineering demands. It makes software development and delivery faster as well as efficient. DevOps, combining "development" and "operations," is a set of practices designed to close the gap between software development and IT operations. It promotes a culture of teamwork, shared responsibility, and accountability (Kolawole & Fakokunde, 2024). Picture 8: depicts DevOps process in a nutshell integrating development, operations, and QA with continuous feedback loops.

PICTURE 8: DevOps process (DevOps University, 2024).

In modern software development, CI/CD pipelines are essential for delivering high-quality software efficiently. Azure Pipeline, a core part of Microsoft's Azure DevOps suite, provides a powerful platform to automate and manage these pipelines. These pipelines assist to meet the growing demand for effective and scalable software development environments by combining CI, delivery, and testing. The pipelines make the lifecycle of developing software easier and more efficient by automating build, test, and deploy process. The pipelines identify the issues or bugs quickly and release new features faster. Thus, release of new features is faster with less manual effort. (Soma, 2024)

CI is often used with agile software development workflow. An organization creates a list of tasks for their product called as the tasks in a backlog assigned to different developers. These tasks can be worked independently and simultaneously on different features at the same time without depending on other team members. When the code is completed, team members can independently merge the work into master branch depending on the code review practice of the team or company. This makes CI a key practice in modern, high-performing software teams.

Without CI, developers must manually coordinate and communicate their code changes, that can be complicated and time-consuming. This involves a lot of back-and-forth communication and planning. This practice slows down not just the development team process but also operations and other parts of the organization. Product teams need to plan when to release new features, patches and decide who will handle them.

After merging, the code is automatically built and run to ensure it works correctly. The core of CI is a version control system that tracks all changes, supplemented by automated tests for code quality and style. This helps catch errors early and keeps the project running smoothly. (Rehkopf, n.d.)

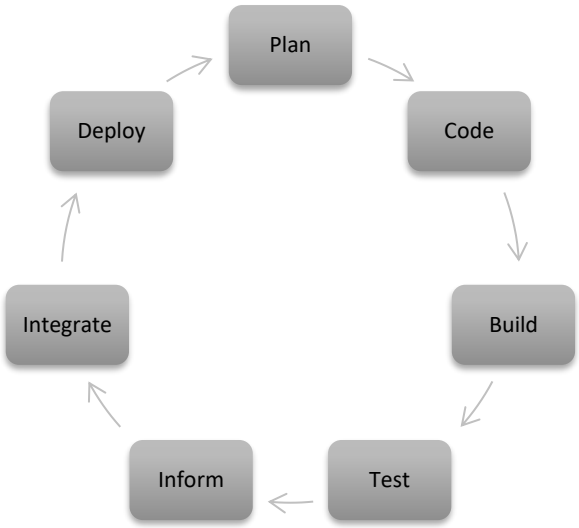
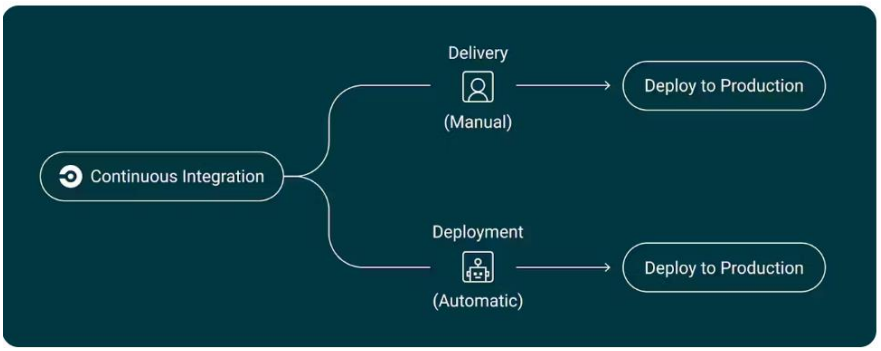


FIGURE 1: Framework of Continuous Integration (CircleCI, n.d.).

Figure 1 illustrates the CI process, key practices in software development with multiple stage. Stages are interconnected, forming a continuous loop that promotes ongoing development, integration, and eventually a deployment activity. It starts with a proper planning of what and how to implement. Developers frequently save their code changes in their shared common repository which ensures change is integrated with the existing codebase. As the code changes are committed and saved, CI systems automatically build a new version of the application. This process checks if the new code works well with the existing code and the application is in a state where it can be tested or released without issues.

After the build process, the CI server automatically runs a series of tests to check the new version’s impact on the application's functionality, security, and compliances to organizational policies. These tests include unit tests, integration tests, security scans, regression tests and code quality checks. This process helps ensure the new code does not introduce problems and keeps the application reliable and secure. After script execution is completed, CI informs about the health of the application by informing whether the code changes were successful or if there were any issues. CI gives a detailed reports and mentions which part did not work if there is any issue. By knowing the status of their code, developers can iterate more effectively and maintain a healthy, reliable application. If the tests are successful, the changes can be merged into the main branch of the code repository. Then, team members have access to the latest working version of the application, and it helps to maintain everyone up to date with the most recent updates and improvements. Finally, CI is often paired with CD to create an automated deployment pipeline. This means that once the code successfully passes all tests, it can be manually or automatically sent to a staging environment for further evaluation or directly to production for users, based on the organization's system as shown in Picture 9.

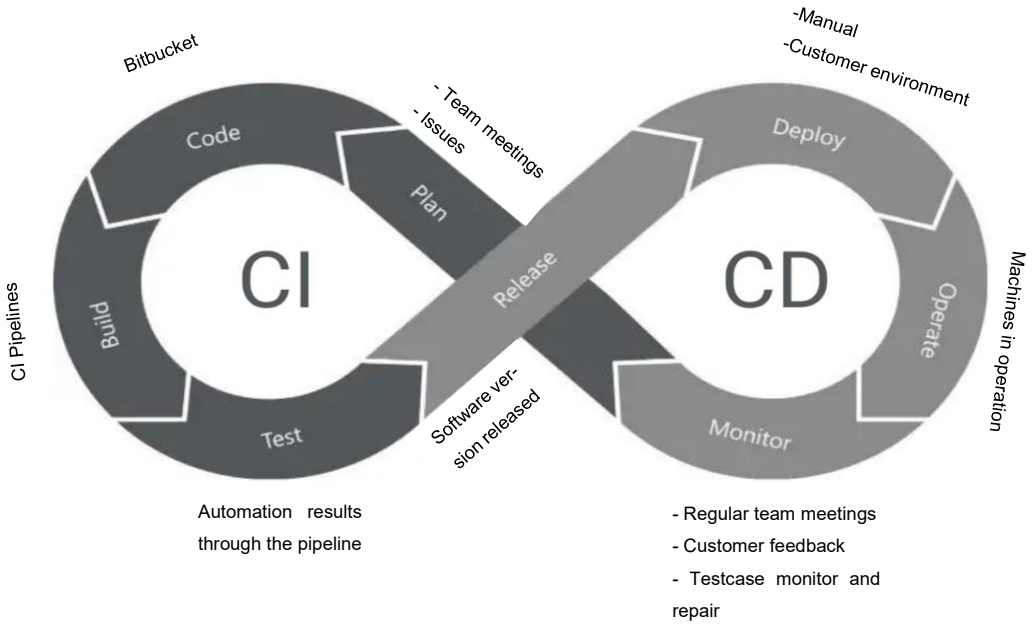


PICTURE 9: Continuous integration and deployment workflow (CircleCI, n.d.).

CI streamlines the process of moving code from development to production by automating key steps like integration, building, and testing. This automation reduces the time and effort needed to deploy new features and updates, allowing organizations to deliver quality products quickly and consistently. As the code changes are continuously integrated and tested, CI helps maintain a smooth and efficient workflow, making teams easier to release reliable software. (CircleCI, n.d.)

4 ROADMAP FOR CI IMPLEMENTATION

Starting with a plan in the CI automation is an important first step that shapes the whole process. This phase requires careful planning to make sure the pipeline fits the project's needs, goals, workflows and success criteria. After a discussion with both QA and PAE teams, the goal and process was set to develop the test automation that can verify the basic functionality in the customized MCT file. To understand the basic requirements, internal documentations, test cases and release notes were researched. Another plan was to study the operations in Azure pipelines. Microsoft Teams was the platform for the prompt communication in company. Decided to communicate with PAE team through it for further communication when required. The tools that will be implemented was discussed in the meeting. Also, agreed with PAE team to provide the MCT files in the repository. The detail list of discussed plans is presented in Figure 3. The comprehensive process structured for software testing in the company and the description in each stage is illustrated in Picture 10.



PICTURE 10: DevOps process (Azeri, 2020, edited).

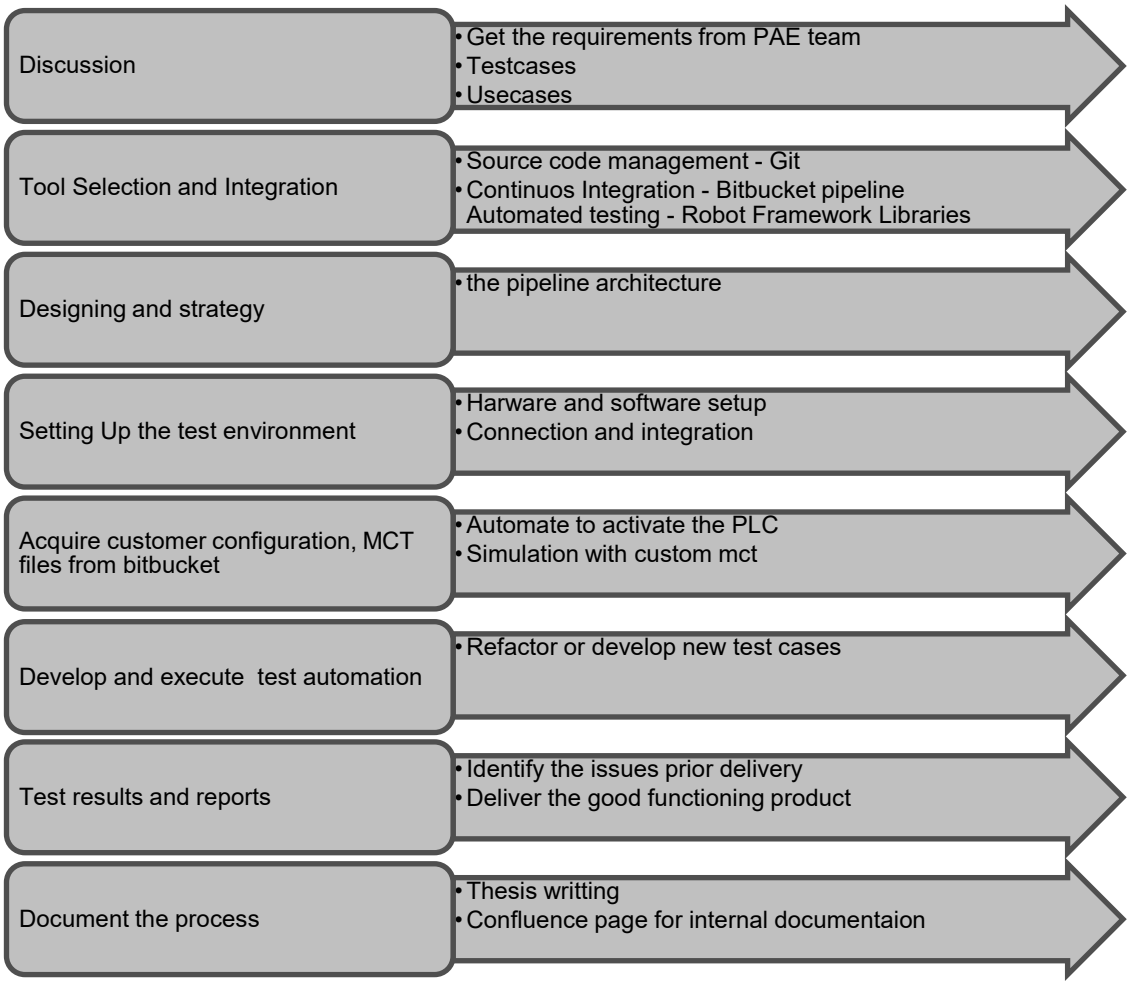
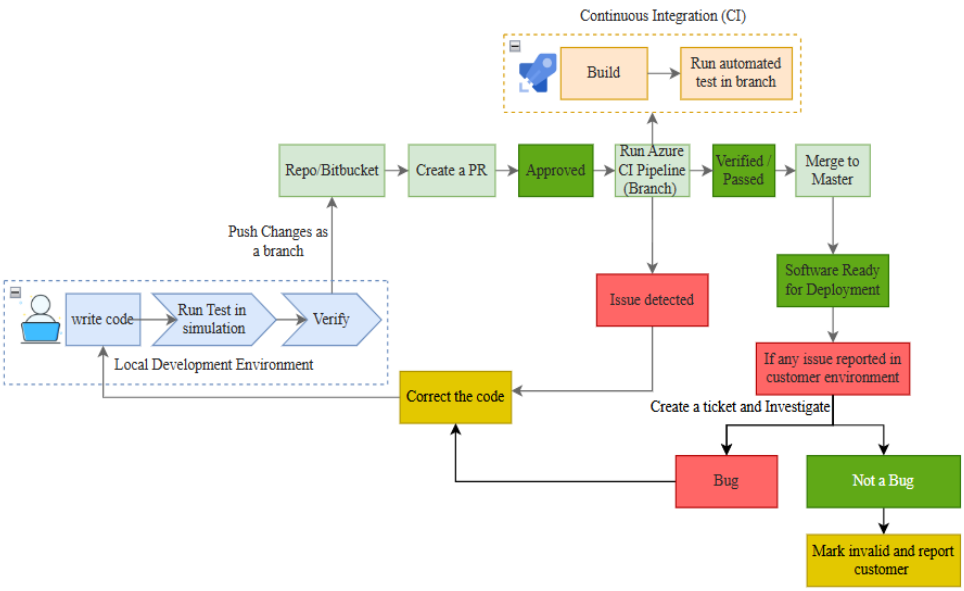


FIGURE 3: Initial plan



PICTURE 11: Framework for CI workflow

The implementation of the project has been aligned based on the Test automation requirements as presented in Picture 1. The description of used tools and implementation process is shown in Picture 11. In the local environment, employee in QA team was free to choose any IDE as per their preference for developing the codes. PyCharm was used as it is a powerful IDE designed specifically for Python programming. Developed by JetBrains, it offers a range of features to enhance productivity, including intelligent code completion, real-time code analysis, and debugging tools. When paired with Robot Framework, a test automation framework, PyCharm becomes an ideal tool for developing and running automated acceptance tests. In PyCharm, the needed libraries were installed for e.g. python, robot framework, selenium library, pyads and many more. Pyads library is a Python wrapper for TwinCATs ADS library. It provides a pythonic way to communicate with TwinCAT3 runtime devices by using the Python programming language. It uses the C API provided by *TcAdsDll.dll* on Windows.

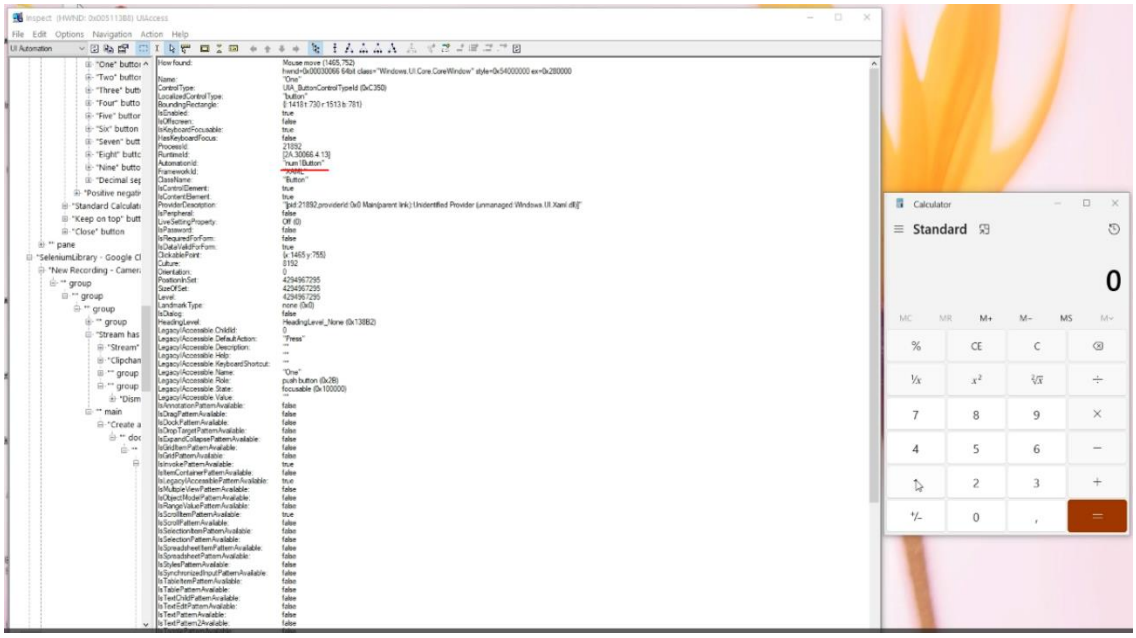
Installation of the library can be done with the command “pip install” followed by library name. Example ***pip install pyads.*** All the libraries installed, and its versions used in this project can be seen in Picture 12.

```
PS C:\Glaston\testautomation> pip list
Package                       Version
----------------------------- -----------
attrs                         23.2.0
certifi                       2024.2.2
cffi                          1.16.0
charset-normalizer            3.3.2
comtypes                      1.4.8
et-xmlfile                    1.1.0
exceptiongroup                1.2.1
h11                           0.14.0
idna                          3.7
MouseInfo                     0.1.3
openpyxl                      3.1.2
outcome                       1.3.0.post0
pillow                        10.3.0
pip                           23.3.1
psutil                        5.9.8
pyads                         3.4.0
PyAutoGUI                     0.9.54
pycparser                     2.22
PyGetWindow                   0.0.9
PyMsgBox                      1.0.9
pymssql                       2.2.7
pyodbc                        5.2.0
pyperclip                     1.8.2
PyRect                        0.2.0
PyScreeze                     0.1.30
PySocks                       1.7.1
pytweening                    1.2.0
PyYAML                        6.0.1
requests                      2.32.2
robotframework                6.1.1
robotframework-assertion-engine 3.0.3
robotframework-databaselibrary 1.3.1
robotframework-excellib       2.0.1
robotframework-pythonlibcore  4.4.1
```

PICTURE 12: List of libraries installed and respective version

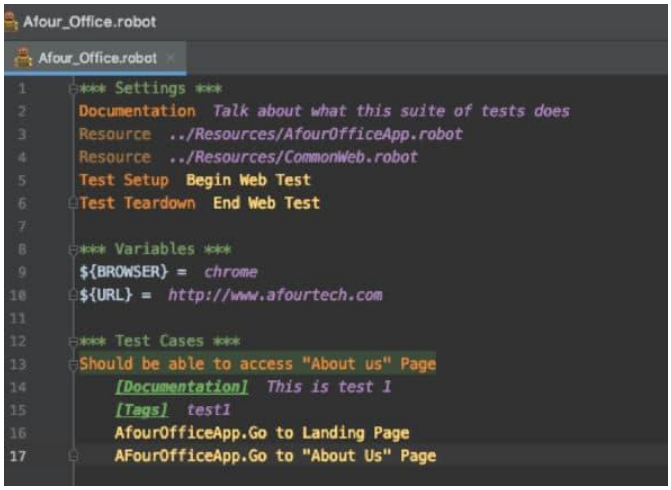
Glaston has already its repository in bitbucket. First, the exist code is cloned in the local environment. The team is using Python as coding language and using Robot Framework to automate the default machine's configuration. A branch is created and in it a separate folder which includes just the test cases for testing the customized machine configuration is created. The GIT branch also helps to keep testcases organized and run the specific test suite. Mostly, Glaston's internal library is used which provides keywords for interacting with Windows desktop application using WinApp Driver. It handles starting up WinApp Driver and interacts with it using HTTP requests via Python's Requests library. In addition to that, SeleniumLibrary is also used as a testing library for Robot Framework. All Robot Framework test keywords in these libraries that interact with an element on a UI interface use an argument typically named as locator.

It is investigated what kind of locator are available using the inspect tool. After getting the locator and the attributes, it is explored how user interface elements can be interacted with using the locators. For that, the present keywords by the team are explored. For an example, to click number 1 in the calculator tool (Picture 13), it can be commanded as "click element num1Button". Here, "num1Button" is one of its locators with attribute "AutomationId" and it interacts as the keyword provided. Thus, button number 1 will be clicked.



PICTURE 13: Locator and Attribute (Singh, 2020).

Step by step, test cases were explored chronologically based on the requirements and use cases from the PAE team. Picture 14 demonstrates how robot framework test suite should be structured. An example of a test script under the robot framework test suite.



PICTURE 14: An example of a Test script (Singh, 2020).

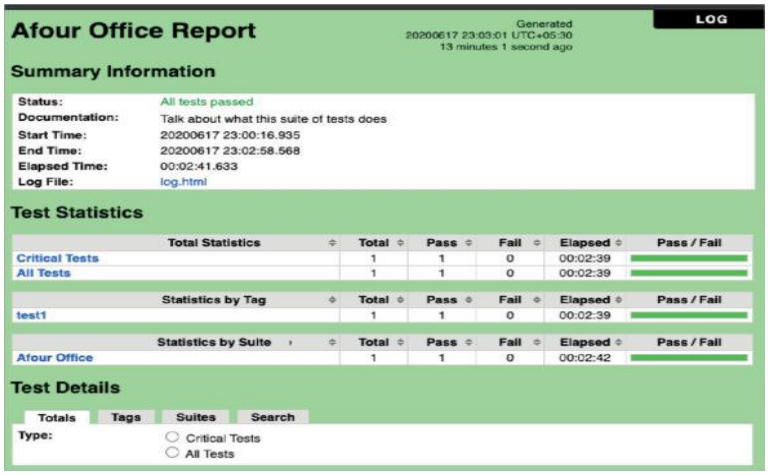
First in the Settings part, documentation is prepared with the introduction about the test suite. Resource part links the file with the respective keyword's main robot file where necessary libraries are imported, and the keywords are defined. It helps in clean coding. Suite setup and suite tear-down mechanism opens the window application and closes it after every test case is executed if there are multiple testcases. It makes the code clean as it avoids repeating setup and teardown scripts each time the testcase is executed. Robot Framework allows to group and execute the test cases from different suites using the test tag name. Tags can be also used for whole suite by providing as a test tags in settings part. The test tags are selected as option to run the tests from the CI pipeline. There can be multiple testcases and all the testcases in the suite should run through the tag name. Tags can be also added to individual test cases that need to be executed as in Picture 14.

After setting up IDE, PLC runtime is activated to start the machine software simulation. Old data tables are detached and new data table for a specific or separate configuration is attached. Desktop UI is started using the commands from YAML file in azure pipeline. Azure pipeline has list of stages and tasks that executes based on the command defined in YAML configuration file to first start the testing environment with necessary libraries and run the tests against the environment.

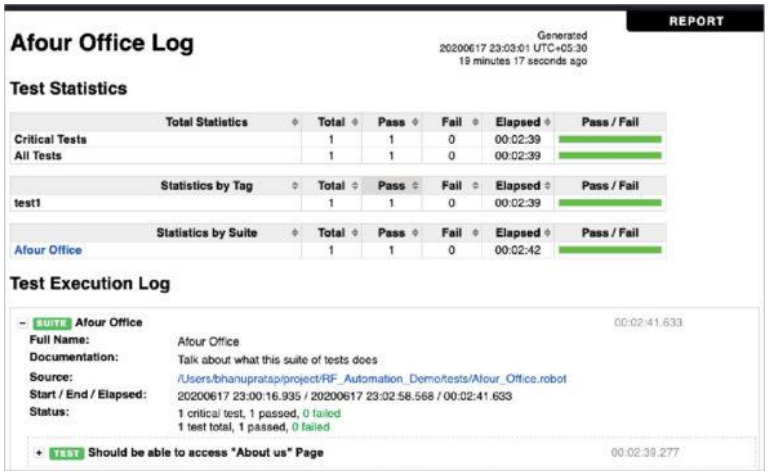
The testcases are run chronologically and after completion of the command, it gives the results of testcases with status and logs like in Picture 15. With the provided log link, detail reports can be visualized in browser as in Picture 16 and 17.

```
(venv1) Bhanuprataps-MacBook-Pro:RF_Automation_Demo bhanuprataps$ robot -d results tests/Afour_Office.robot
==============================================================================
Afour Office :: Talk about what this suite of tests does
==============================================================================
Should be able to access "About us" Page :: This is test 1            | PASS |
------------------------------------------------------------------------------
Afour Office :: Talk about what this suite of tests does              | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /Users/bhanupratap/project/RF_Automation_Demo/results/output.xml
Log:     /Users/bhanupratap/project/RF_Automation_Demo/results/log.html
```

PICTURE 15: Execution output (Singh, 2020).



PICTURE 16: Results Html (Singh, 2020).



PICTURE 17: Log file (Singh, 2020).

Finally, after the coding is completed and the testing is successful when conducted manually in local machine, it is ready to push to the bitbucket for a peer review. The QA team is using Bitbucket as source code repository and version control system because it has following advantages:

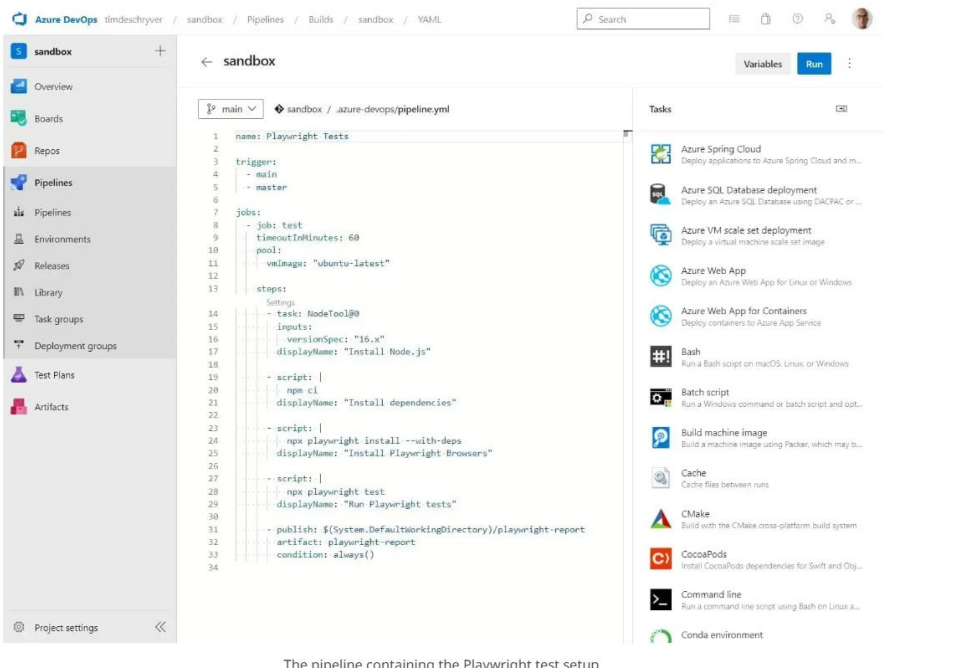
- easy to create and manage Git repositories
- collaborate on code and track every change in a secure and organized way.
- integrates with other Atlassian products such as Jira (for project management and issue tracking) and Confluence.
- allows developers to use pull requests to review, discuss, and approve code changes promoting team collaboration.
- work on branches to create features or fix bugs in isolation. Bitbucket supports Git workflows and makes merging changes easy while minimizing conflicts.

It ensures that the latest changes are securely stored and available for collaboration with team members. In Glaston, there is code review culture that each creates a pull request (PR) so that other team members and seniors can review and provide feedback. After PR is verified, it is integrated into the main branch. This process resolves code conflicts before merging into master branch. Also, the code can be tracked, merged, or reverted if necessary.

Glaston had already implemented Azure DevOps pipeline for a CI service. Only verified and respective employees have permission for accessing and using it. The needed permission to access the pipeline, Bitbucket and documents were provided. The beginning step of the process as per the planning was the PAE team to upload the customized MCT in the repository to the Bitbucket. The plan was to explore with five different MCT in the initial phase. The pipelines were defined in a YAML configuration file where developers can specify different steps for various tasks, such as running unit tests, building application or deploying to staging or production environment. The CI pipeline can be modified to run various tasks such as compiling code changes, running the tests and deploying to production based on predefined triggers or time schedule.

Company has other pipelines for various other purposes. Additional CI pipeline for the test automation results for PAE team was designed and developed. Also,

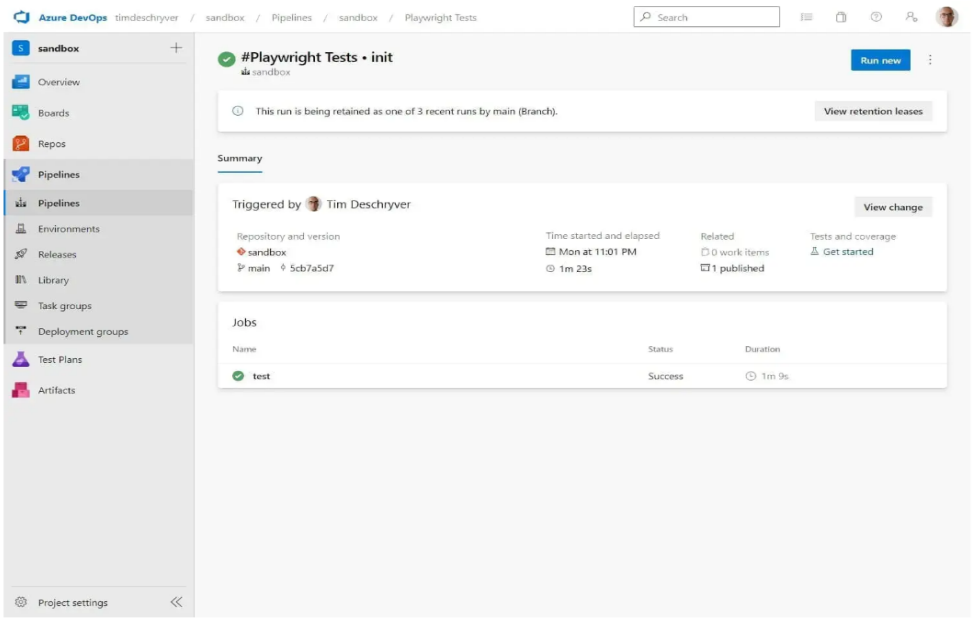
it was planned to run the pipeline once every month. The actual pipeline is not presented here because of data privacy but an example is provided in Picture 18.



The pipeline containing the Playwright test setup

PICTURE 18: An example of the pipeline (Deschryver, 2023).

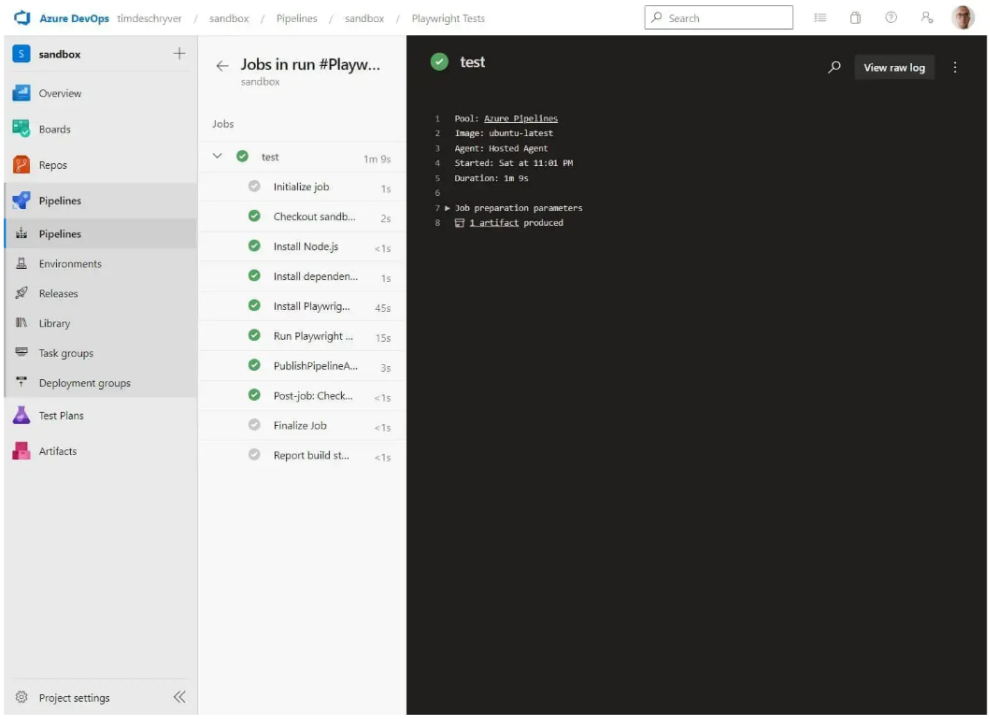
When CI pipeline is run selecting the correct MCT, version of PLC and UI, and branch, it will start to automate the testcases that have been scripted and stored in bitbucket. When the pipeline is completed and is successful, it should give the success page like in Picture 19.



A green pipeline build.

PICTURE 19: A green passed pipeline build (Deschryver, 2023).

A green coloured and tick icon in pipeline page represents that all testcases had been passed. If something went wrong the pipeline page will show a red and “x” icon. In both cases, the detail can be investigated by viewing the testcases clicking on the test job or “Artifact” which contains Test Suite Log (Picture 20).



The detailed view of the pipeline build.

PICTURE 20: Details of pipeline build (Deschryver, 2023).

Finally, it is important to monitor the pipeline and include ways to gather feedback. Monitoring helps spot problems early, and feedback ensures the pipeline and automation testcases keep getting better over time as per the changing requirements and software functionalities. After the software is deployed, Glaston takes the customers feedback, issues and comments. These issues are first recorded to salesforce as a quality issue ticket. Then identical bug ticket is created to Jira backlog. The feedback and comments are discussed in the team. Also, the test automation is constantly monitored and managed according to the issue and bug raised.

5 RESULTS

The goal of this research is to explore ways to improve and simplify the CI pipelines in Azure Pipelines, making the software testing process automated and more effective. This goal has been achieved through successfully implementing the test automation framework and its integration in the Azure pipeline as planned in the initial phase. The software components like PLC and .NET UI were successfully activated and database was attached. The Inspect tool was attached and used. The testcases were developed which runs as required by the PAE team. Generally, PAE team performs the smoke testing. There is a new suite designed for PAE testing, so the pipeline was updated accordingly.

During the process more than ten of new or refactored robot framework tests cases were added in the suite. These test cases utilize the Robot Framework libraries and Python functions integrated in the keywords. The test keywords and necessary libraries were also integrated to the Azure pipeline and can be executed using the Azure DevOps CI tool. This makes the process easier and easily accessible to all the team members involved in the testing of customer configurations. The test cases have been verified by the reviewers and merged into the master. The results and logs show that all the testcases that was planned with the PAE team had been passed. If a test fails, the Robot Framework test logs will show which test condition or case failed. The logs and test report file will include all the necessary details.

The development process and tests for the custom MCT have been carefully documented according to the team's guidelines. This documentation explains how to use the test suite and Azure build. It also includes important information for ongoing development and future updates, making it easier to maintain and improve the system over time.

6 CONCLUSIONS AND DISCUSSION

In this thesis work, various tools were integrated to automate the testing for the customized MCT. It was a bit challenging to document all the process in this thesis as not every information can be publicly disclosed because of the data privacy policies of the company. The pipeline logs and results show that the basic functionality of those MCT can be verified automatically. As per the initial planning, five customer's MCT were tested. It is planned to integrate additional MCT in future. Also, implementing continuous testing in a more advanced way was discussed. When the new MCT are uploaded to the PAE team's Bitbucket repository, CI pipeline runs the test automation in a predefined time. That means that manual effort will not be needed to run the pipeline. The results and log files are then delivered to the respective customers automatically after the test run. Customers will get the information on working functionalities with the help of such reports prior starting the production.

A demonstration of this project was presented internally with the team. In conclusion, the PAE team is satisfied and happy with test automation integrated in customer's MCT. It will save their valuable time as it minimizes the manual testing effort. During the planning and implementation process, same tools were utilized that were already in use by the QA team. This will make it easier to maintain and further improve the solution, since the team members are already familiar with all the tools and technologies. At the same time, this also adds a potential challenge that new libraries or tools alternative could have been better for the test automation implementation.

In conclusion, the development of test automation for the custom configuration was successful and completed on time. Working on the desktop application was interesting, but some challenges arise with missing or duplicate locators. These issues were resolved using alternative methods suggested by team colleagues.

As initial phase, current test suite is suitable to verify only smoke testing. The plan of PAE team is to add the feature testing also in the future. The challenge is that every customer has different feature requirement. So, the common test suites

cannot be generalized for all the customer's MCT because the irrelevant test suites will be failed and creates confusion in the test logs. Other options can be explored for running particular features. One of the good options might be using the tag for features and for the custom machine specific test cases. The current Azure pipelines now support running the test suites and cases using the tag name. In future the custom MCT features could be grouped using the tag name and just run the supported feature tests only. This will provide the efficient test reports with the supported features only.

During the test development phase, it was noticed that some parts of Windows UI and MCT tool were not suitable for automation. The automation Id or locator was not available to interact with the user interface. Using keyboard arrow keys and coordinate-based automation worked as a short-term workaround, but automation process might not be reliable every time. To support robust automation testing, the development team should maintain the UI with unique IDs or other supporting properties. Also, the robot framework has different libraries for automating desktop application testing. The popular ones like Desktop library from Robocorp could be also used as alternative since the library is continuously maintained and improved. There are also some libraries like PyWindowsGuiLibrary that uses pywinauto and pyautogui wrappers written in Python programming languages for interacting with Desktop user interface. This will support the cases where the earlier robot framework libraries don't have the keyword for doing some user interface actions.

REFERENCES

Abto Software. (2012, September 24). .NET Desktop Application Development - Abto Software. Abto Software. https://www.abtosoftware.com/blog/why-building-net-desktop-applications

Annual Review – Glaston. (2024). Glaston.net. https://glaston.net/wp-content/uploads/2025/03/Glaston_Annual_Review_2024.pdf

Atlassian. (n.d.). What is version control. Atlassian. https://www.atlassian.com/git/tutorials/what-is-version-control

Azeri, I. (2020, January 29). *An Introduction To CI/CD*. Mabl. https://www.mabl.com/blog/what-is-cicd

Beckhoff. (2024, October 28). *TwinCAT 3 Product overview* [Review of *TwinCAT 3 Product overview*]. https://download.beckhoff.com/download/document/automation/twincat3/Product_overview_EN.pdf

Beckhoff. (n.d.). TwinCAT | *Automation software*. Beckhoff Automation. Retrieved December 27, 2024, from https://www.beckhoff.com/en-en/products/automation/twincat/

Berga, K. (2024, July 3). *What Is Automated Testing and How Does It Work?* TestDevLab Blog. https://www.testdevlab.com/blog/automated-testing

CircleCI. (n.d.). *Continuous integration - CI*. CircleCI. Retrieved April 19, 2025, from https://circleci.com/continuous-integration/

Deschryver, T. (2023, April 12). Playwright in an Azure DevOps Pipeline. Timdeschryver.dev. https://timdeschryver.dev/blog/playwright-in-an-azure-devops-pipeline

DevOps University. (2024, December 4). *CICD and DevOps*. Devopsuniversity.org. https://devopsuniversity.org/cicd-and-devops/

GeeksforGeeks. (2024, April 24). *How to Start Automation Testing from Scratch?* GeeksforGeeks. https://www.geeksforgeeks.org/how-to-start-automation-testing-from-scratch/

GeeksforGeeks. (2025, April 7). *Automation Testing - Software Testing*. GeeksforGeeks. https://www.geeksforgeeks.org/automation-testing-software-testing/

Jia, X. (2023). *The Role and Importance of Software Testing in Software Quality Management*. *1*(4), 39–44. https://doi.org/10.62517/jiem.202303406

Kolawole, I., & Fakokunde, A. (2024). *Improving Software Development with Continuous Integration and Deployment for Agile DevOps in Engineering Practices*. International Journal of Computer Applications Technology and Research, 14(01), 25–39. ISSN:-2319-8656. https://doi.org/10.7753/IJCATR1401.1002

Microsoft. (n.d.). *What is SQL Server?* Https://Learn.microsoft.com/En-Us/Sql/Sql-Server/What-Is-Sql-Server?View=Sql-Server-Ver16. Retrieved December 27, 2024, from https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16

Mohammad, S. M. (2015). *Automation Testing in Information Technology*. IJCRT - International Journal of Creative Research Thoughts (IJCRT) IJCRT.ORG, 3(3), 118–125. https://doi.org/10.1729/journal.24200

Our Journey – Glaston. (2023). Glaston.net. https://glaston.net/fi/our-journey/

PLC Table. (n.d.). *Beckhoff PLC*. PLC Systems. https://www.plctable.com/beckhoff-plc/#google_vignette

Regio, C. (2023, November 10). *What is .NET? [Pt 1] | .NET for Beginners*. YouTube. https://www.youtube.com/watch?v=6BcPIvVfVAw&list=PLdo4fOcmZ0oUwBEC2bnwPtHqbU8Vmh_tj

Rehkopf, M. (n.d.). *What is Continuous Integration | Atlassian*. Atlassian. Retrieved April 14, 2025, from https://www.atlassian.com/continuous-delivery/continuous-integration

Robot Framework. (2019). Robot Framework. Robotframework.org. https://robotframework.org/

Singh, B. P. (2020, September 28). *Robot Framework: Test Automation Made Quick and Easy*. Open Source for You. https://www.opensourceforu.com/2020/09/robot-framework-test-automation-made-quick-and-easy/

Soma, V. (2024). *Enhancing CI/CD Pipelines with Azure Pipelines*. Journal of Engineering and Applied Sciences Technology, 6(8), 1–4. https://doi.org/10.47363/jeast/2024(6)e108

Syed, S. (2013, July 5). *PLC Communication Using .Net*. C-Sharpcorner.com. https://www.c-sharpcorner.com/uploadfile/asmabegam/plc-communication-using-net/

Uraizee, H. (2019, August 28). *Frequently Asked Questions*. GitHub. https://github.com/microsoft/WinAppDriver/blob/master/Docs/FAQ.md

Watson, D. (2024, March 11*). IDEs vs. Code Editors: Everything You Need to Know | The WebStorm Blog.* The JetBrains Blog. https://blog.jetbrains.com/webstorm/2024/03/ides-vs-code-editors/

Woodward, M. R., & Hennell, M. A. (2005). *Strategic benefits of software test management: a case study*. *Journal of Engineering and Technology Management*, *22*(1-2), 113–140. https://doi.org/10.1016/j.jengtecman.2004.11.006