Choosing a Test Automation Framework for Programmable Logic Controllers in CODESYS Development Environment

Mikael Ebrahimi Salari*, Eduard Paul Enoiu*, Wasif Afzal*, Cristina Seceleanu*
*Mälardalen University, Sweden
{mikael.salari, eduard.enoiu, wasif.afzal, cristina.seleceleanu}@mdu.se

***Abstract*— Programmable Logic Controllers are computer devices often used in industrial control systems as primary components that provide operational control and monitoring. The software running on these controllers is usually programmed in an Integrated Development Environment using a graphical or textual language defined in the IEC 61131-3 standard. Although traditionally, engineers have tested programmable logic controllers' software manually, test automation is being adopted during development in various compliant development environments. However, recent studies indicate that choosing a suitable test automation framework is not trivial and hinders industrial applicability. In this paper, we tackle the problem of choosing a test automation framework for testing programmable logic controllers, by focusing on the COntroller DEvelopment SYStem (CODESYS) development environment. CODESYS is deemed popular for device-independent programming according to IEC 61131-3. We explore the CODESYS-supported test automation frameworks through a grey literature review and identify the essential criteria for choosing such a test automation framework. We validate these criteria with an industry practitioner and compare the resulting test automation frameworks in an industrial case study. Next, we summarize the steps for selecting a test automation framework and the identification of 29 different criteria for test automation framework evaluation. This study shows that CODESYS Test Manager and CoUnit are mentioned the most in the grey literature review results. The industrial case study aims to increase the know-how in automated testing of programmable logic controllers and help other researchers and practitioners identify the right framework for test automation in an industrial context.**

***Index Terms*—PLC, test automation framework, CODESYS, Grey Literature Review, Automation framework Comparison.**

I. Introduction

Testing is an important activity in the engineering of industrial control software. In certain application domains (e.g., automation industry), programmable logic controllers (PLCs) provide management and monitoring for control software [1]. Even if test execution on PLCs is usually performed manually, test automation is emerging during PLC development at different stages of integration. Different PLC vendors and PLC software manufacturers have proposed several Integrated Development Environments (IDEs). One of the most frequently-used PLC IDEs in the industry is CODESYS, a manufacturer-independent software that is free to use. It supports all the PLC programming languages of IEC 61131-3 standard and is widely used by many industrial companies.

Test automation can be defined as the process of automating software testing tasks such as test script development, test execution, and requirements verification using an automation test framework [2] [3]. Choosing the right test automation tool received significant attention from both academia and industry in recent years [4]. Furthermore, recent observations of collaborations between industry and academia emphasize the importance of selecting the right test automation tool since it is a non-trivial task for many practitioners [5]. This could stem from at least two reasons; the misunderstanding of what important criteria to use for choosing the right tool and the lack of knowledge of the pros and cons of using test automation frameworks in practice.

In this paper, we address the problem of choosing the right test automation tool for PLC programs in CODESYS IDE by leveraging a Grey Literature Review (GLR) followed by a comparative study on the discovered tools. Aiming at conducting an effective comparison between the detected tools, we discover the most important features of the test automation tools through a literature review. We evaluate the validity of the discovered features by asking a group of engineers of a large automation company to review them. Based on the goal of this study, we formulate the following research questions:

1) What are the reported test automation frameworks for CODESYS in the grey literature?
2) What are the reported features that should be considered when choosing a test automation framework for CODESYS?
3) How do different test automation frameworks for CODESYS compare in terms of different features?
4) How do different test automation frameworks for CODESYS compare in terms of their applicability to an industrial use case?

We aim to answer these research questions using a GLR and a case study in which we compare the identified test automation frameworks.

II. Background and Related Work

A. PLC Programming, IEC 61131-3 Standard and CODESYS

In recent years the IEC 61131-3 programming standard for the automation industry has been proposed, and today it is widely accepted and used by a variety of well-known PLC

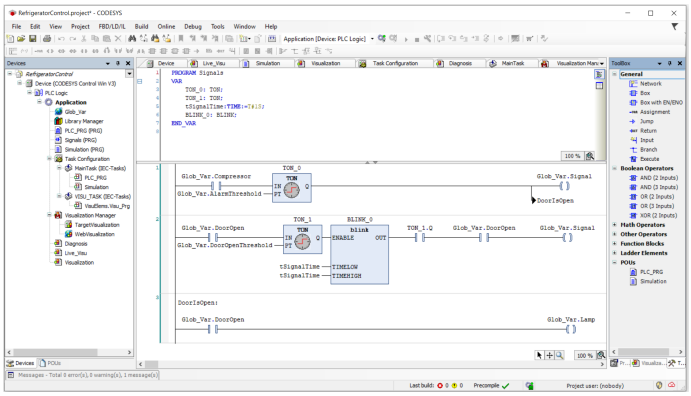

Fig. 1: CODESYS Integrated Development Environment

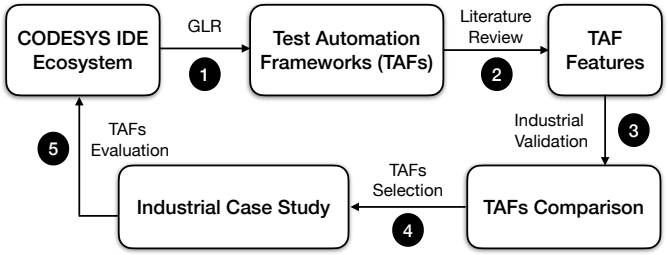

Fig. 2: An Overview of The Methodology used for Choosing Test Automation Frameworks (TAFs) for PLCs.

manufacturers worldwide. The smallest independent software unit in a PLC program is called a POU (Program Organisation Unit), also known as a block. There are three types of POU: *function*, *function block* (FB), and *program*, which can call each other with or without parameters. Based on IEC 61131-3 standard, a POU can be programmed using several programming languages [6] (i.e., *structured text* (ST), *instruction list* (IL), *ladder diagram* (LD), *Sequential Function Chart* (SFC) and *function block diagram* (FBD)).

CODESYS stands for COntroller DEvelopment SYStem, and it is an integrated development environment (IDE) for programming controller applications according to the international industrial standard IEC 61131-3 [7]. The framework is developed by Smart Software Solutions GmbH In this work, we choose CODESYS as our IDE for two reasons. First, CODESYS is free to use and is popular in industry [7]. Second, CODESYS is a device-manufacturer-independent IDE [1] that can be used to develop PLC programs for a wide range of PLC devices from various vendors.

B. Related Work

In recent years, researchers have made efforts in developing test automation frameworks for PLC software. Jamro introduces a method for POU-oriented unit testing for IEC 61131-3 languages [8]. In this approach, test cases are defined in CPTest+, a dedicated test definition language. The proposed approach is introduced in the CPDev engineering environment. Recently, Hofer and Russo [9] presented a unit-testing framework named APTest (Advanced Program Organization Unit Testing) for CODESYS IDE. The framework is developed based on the IEC61131-3 standard and CPTest+. APTest is a POU-based framework equipped with a test library supporting different types of assertions and compatible with CODESYS (version 2.3). Even if these academic tools have a wide range of capabilities such as test parallelization, simulating analog signals, and supporting time-dependent behaviors, there is limited evidence of how industrially-useful these frameworks are. In addition, these tools are only compatible with older versions of CODESYS.

[1]https://www.codesys.com/

Selecting a test automation framework is an essential part of software testing, and recent studies have looked at different challenges to implementing automation support. Raulamo-Jurvane et al. [5] perfomed a GLR to identify the practitioners' criteria for choosing the right test automation tools. The study showed that practitioners select and embrace the widely known and utilized tools. Garousi et al. [4] compared visual GUI testing frameworks (i.e., Sikuli and JAutomate) using several relevant features and perfomed an industrial case study. In 2019, Raulamo-Jurvane et al. investigated the practitioners' opinions on evaluating testing tools by conducting an online survey [10]. They found that evaluations in which one uses a tool seem to be more favorable than those based on opinions and considering the opinions of seven experts provides a reasonable level of reliability.

These results kindled our interest in studying how to tackle the problem of choosing a test automation frameworks for programmable logic controllers in CODESYS, especially when these tools are used to test safety-critical industrial control systems.

III. METHOD

In this study, we leverage a hybrid methodology that combines conducting a GLR and an industrial case study. Our aim of performing GLR is to find the most-discussed available test automation frameworks of CODESYS IDE systematically and reasonably, reflecting the practitioners' point of view. In contrast, the case study is performed to represent and compare the functionality of the proposed frameworks on a real-world industrial case study. The overall conceptual architecture of the proposed hybrid methodology can be observed in Figure 2, and it will be discussed in more detail in the rest of this section.

A. Grey Literature Review

We find the GLR as one of the most suitable approaches to conduct our study, since the available information about test automation frameworks of CODESYS is more accessible online than in academic papers. Consequently, to explore the available automation frameworks targeting CODESYS, we need to gather information across the web, and prioritize it based on features and popularity. We base our approach on the GLR approach proposed by Garousi e al. [11]. The conducted GLR approach (Step 1 in Figure III) consists of several steps.

TABLE I: GLR Search Strings

A	B	C	D
CODESYS	Test	Automation	framework
	Testing	Automated	Framework
		Automatic	

1) Search Process and Framework Selection: First, we aim to detect the test automation frameworks available for CODESYS, and consequently we perform a GLR by searching for a combination of topic-related keywords on Google. We carry out several exploratory search queries using strings such as "CODESYS test automation framework" and "CODESYS test automation tool". Moreover, we also consider the related search strings suggested to us by Google. However, we remove the suggestions that include a specific framework name. We present the final search strings in Table 1. We consider the keywords of each column as synonyms and combine them via the OR operator (e.g., framework OR tool). Then, we produce the whole search string using the AND operator between the existing keywords of columns A to D.

We follow the guidelines used by Raulamo et al. [5] to identify the available test automation frameworks targeting CODESYS. Firstly, the pool of contents is revised by the first author of this paper to remove any irrelevant results from it. Secondly, the new version of the pool is reviewed by at least one of the authors of this paper. We only include the results related to CODESYS-compatible test automation frameworks. Furthermore, all the academic papers are excluded from the contents pool. It should be noticed that in case of any conflicting views among authors, we leverage a voting system to choose the results for the final pool, based on the opinion of the majority of the authors.

2) Pool of Objects: Every result and its corresponding information is called an "Object" in the rest of this study. We consider several properties for every object The considered properties are shown in Table II. The final pool of objects and defined criteria are accessible online on GitHub[2].

3) Data Extraction Method: To classify the objects of the pool of contents and specify the required criteria to detect the efficient test automation frameworks of CODESYS, we follow a systematic qualitative method of specific related work [12]. All authors of this paper have reviewed the qualitative analysis of this work. Since coding in qualitative analysis is not just a preliminary step of analyzing the data, but it also includes "deep analysis and interpretation of the data meanings" [12], we select the relevant results and analyze them.

4) Selection Criteria: The final version of considered criteria to classify the objects of the pool is shown in Table III. Since the goal is to discover the valid test automation frameworks that are compatible with CODESYS IDE, we checked the official CODESYS documentation. The next criterion is the credibility of the object's author, for which two parameters are evaluated: the number of referral sites to an object, and the Alexa rank of the object's website. Moreover, as we are looking for up-to-date frameworks compatible with the CODESYS v3.x family, the publishing date is another criterion used in filtering the results. The first version of the CODESYS V3.x family was released in 2016[3].

B. Discovery and Validation of Features

To perform an effective comparison between the discovered test automation frameworks of CODESYS, first, we need to identify the most important reported features of test automation frameworks. To this end, we conduct an informal literature review on the related work in this area to gather a list of features that should be considered during the comparison process (Step 2 in Figure III).

In this paper, we reviewed the literature and identified five existing works as our sources of information for test automation framework feature extraction: (1) The study of Ferrari et al. [13] on the selection and adoption of formal tools in the railway domain, (2) Umar et al. [14] which is an overview of popular existing test automation tools, (3) the study of Gamido et al. [15] performing a comparative review based on the user's needs, (4) a study [16] focusing on some well-known test automation frameworks including Selenium, Quick Test Professional and Testcomplete, and (5) a comparative study [17] on two different automated visual GUI testing tools including CommercialTool and Sikuli.

Aiming at making this work more aligned with industry, we asked a group of engineers who are working at a large industrial automation company in Sweden to evaluate the validity of the discovered features from their point of view (Step 3 in Figure III). These engineers are experts in developing and testing the supervisory PLC programs. Besides feature validation, the engineer also added two new features that are important from the company perspective in choosing the right test automation tool. We only include the industrial validated features in our tool comparison study, since our priority is to help practitioners in their choice, based on their needs.

C. Industrial Case Study

To practically use the results of this comparison, we evaluate the applicability of the discovered test automation frameworks for CODESYS in a real-world scenario, using an exploratory case study using an industrial system (Step 4 & 5 in Figure III). The case is provided by a large industrial automation company in Sweden.

IV. Results

A. RQ1 - Discovered Test Automation Frameworks

As a result of the GLR, we obtain 120000 search results which are all written in English. We only stored the first 100 results locally to build the pool of contents since we discovered that these contain relevant sources to our topic. Most of the objects in the final version in the pool of objects have been published by industry individuals, including IDE developers

[2]https://github.com/MikaelSalari/CODESYS-Tool-Comparison

[3]https://store.codesys.com/en/codesys.html

TABLE II: Considered Object Properties

Object Properties												
Title	Link	Extracted framework	Object Type					Literature Type		Demographic info		
			Documentation	Tutorial	Description	Discussion	other	Grey	Formal	Year	Author(s)	Author's Organisation

TABLE III: Selection Criteria for GLR

Selection Criteria			
Author's Credibility		CODESYS Verified	v3.x Support
Alexa Rank	Referral Sites		

and PLC vendors. Aiming to establish a trade-off between the preferences of companies and independent framework developers in our results, we included valid third-party developers and Github topics in the final pool. After reviewing the content of the pool, we ended up with a pool consisting of 13 sources. After analyzing the final objects based on the defined criteria, we discovered three test automation frameworks as the most prevalent automation frameworks targeting CODESYS. Out of all the collected results, 62% of the objects in the pool are pointing towards CODESYS Test Manager[4] (the largest share of the discovered objects). Two other frameworks, CoUnit[5] and TcUnit are revealed in 15% of the objects each. Other frameworks were mentioned in 8% of the objects. Our results suggest that most of the discovered objects of our GLR after screening and applying the selection criteria point towards CODESYS Test Manager, CoUnit (formerly known as CfUnit), and TcUnit as the predominant test automation frameworks targeting CODESYS. We note here that CoUnit is developed based on TcUnit and both frameworks have similar functionality. Since CODESYS IDE officially supports only the former, we include CoUnit in the final list of discovered automation frameworks.

> *Answer RQ1: Our results suggest that the most prevalent test automation frameworks targeting CODESYS IDE for PLC testing are the CODESYS Test Manager and CoUnit.*

B. *RQ2 - Test Automation Frameworks Features*

Conducting a comparison between the discovered test automation frameworks of CODESYS, first, we need to identify the essential features of these test automation frameworks. To this end, we followed a hybrid approach which consisted of a literature review of related works followed by an industrial feature validation. Based on three sources of information used (academic related-works, industrial input, and official documentation), we discovered 29 industry-reported essential features that should be considered when choosing a test automation framework for PLCs. We acknowledge that many of these features are generic. Still, the instantiation of these features is specific to PLCs. Since our aim of conducting this work is to address the needs of industrial practitioners, we evaluated the validity of the discovered features by checking them with a group of engineers working with CODESYS and PLC testing in an industrial automation company in Sweden. These engineers validated these features of a test automation framework by marking the ones a tester would use to choose such a framework (i.e., 15 out of 29 features were considered important by these engineers). The list of the discovered and validated test automation framework features and non-validated ones as well as their category and source of extraction, are shown in Table IV and V respectively. It should be noted that the gathered data does not need any further processing (e.g., open coding).

[4]https://store.codesys.com/codesys-test-manager.html?

[5]https://forge.codesys.com/lib/counit/home/Home/

TABLE IV: Extracted and Validated Framework Features.

Industry-validated Features		
Category	**Feature**	**Extraction Source**
Company Constraints	Cost	[13], [14], [15], [16], [17]
	Supported Platforms	[13], [14], [15]
Maturity	Industrial Usage	[13]
	Stage of Development	[13]
Testing Functionalities	Documentation and Report Generation	[13]
	Playback Record	[15]
	Test Suite Support	[17]
	Test Suite Extension	Industry
Tool Flexibility	Teamwork Support	[13]
Usability	DevOps/ALM Integration Support	[14]
	Continuous Integration (CI) Support	[14]
	Script Language	[14], [15], [17]
	Availability of Customer Support	[13], [14]
	Quality of Documentation	[13]
	Maintenance Support	Industry

TABLE V: Other Extracted Framework Features.

Other Features		
Category	**Feature**	**Extraction Source**
Company Constraints	Ease of Installation	[13], [14]
	License Type	Tool Documentation
Testing Functionalities	Test Script Specification	[13]
	Supported Testable Objects	Tool Documentation
	Requirements Traceability	[13]
	Script Creation Time	[14]
	Import Support	[17]
Tool Flexibility	Backward Compatibility	[13], [17]
	Standard Input Format	[13]
	Modularity of The Tool	[13]
	Framework Development Language	[17]
Usability	Programming skills	[14], [15]
	Report Format	[15]
	Graphical User Interface (GUI)	[13]

We divided the discovered features into five categories based on their focus, including Company Constraints, Maturity, Testing Functionalities, Framework Flexibility, and Usability.

*1) **Company Constraints**:* **Cost** indicates the cost model used (FREE, MIX, PAY) [13], **Supported Platforms** specifies the platforms supported by the framework (Windows, Mac, Linux) [13], **Ease of Installation** indicates whether the framework installation requires installing other additional components or it covers all the installation requirements. (YES, NO, PARTIAL) [13], and **License Type** implies the type of license used for the test automation framework (e.g. Apache, MIT).

*2) **Maturity**:* **Industrial Usage** specifies the level of reported industrial usage in academic papers and reports (HIGH, MEDIUM, LOW) [13] and **Stage of Development** indicates whether the framework is evolved through releasing different versions (MATURE), it is an academic or early version (PROTOTYPE) or it is new but has a strong fundamental roots (PARTIAL) [13].

*3) **Testing Functionalities**:* **Test Script Specification** determines how the test script is represented by the framework: Graphical User Interface (GUI), Textual Representation (TEXT), imported textual file (IMPORT) [13]. **Supported Testable Objects** feature was discovered by reviewing the tools documentation. This feature indicates the object types that are supported for testing in a PLC program (APPLICATION, IEC LIBRARIES, COMMUNICATION). **Documentation and Report Generation** characterizes whether the automatically generated reports and documentation of a tool contain well-detailed technical details (COMPLETE) or only some summarized technical details are available (SUMMARIZED) [13]. **Requirements Traceability** specifies if the framework can provide traceability between the generated test cases to other related artifacts (YES, NO) [13]. **Script Creation Time** relates to the time required to produce test scripts (QUICK, SLOW) [14]. **Playback Record** indicates whether the framework can record testing sessions and playback these as test scripts (YES, NO) [15]. **Import Support** specifies if the framework can import test cases and test scripts using external files (Python, Java, NO) [17]. **Test Suite Support** relates to the framework's ability to support the user in the creation and execution of test suites (YES, NO) [17]. **Test Suite Extension** indicates the ease of extending test suites using provided features of a certain test automation tool. Developing new test suites in a PLC program is a crucial and sensitive task because all the connections between the different test suites (test counterparts of a POU under test) should be updated after any new modifications. (EASY, MEDIUM, HARD).

*4) **Framework Flexibility**:* **Backward Compatibility** indicates to which extent test scripts developed with previous versions of the CODESYS framework can be used in the current version (YES, NO, UNCERTAIN) [13]. **Standard Input Format** specifies whether the language that is used for developing test cases is based on a standardized programming language or not (YES, NO) [13]. **Modularity of The Tool** specifies if the framework supports a wide range of different modules and add-ons that can be used to extend its functionality or not (YES, NO) [13]. **Teamwork Support** indicates whether the framework supports multi-user development and collaboration (YES, NO) [13]. **Framework Development Language** details the programming language that is used to develop the framework (e.g., Python, Java, C, Jython) [17].

*5) **Usability**:* **Programming Skills** specifies what level of programming skills is needed to work with the framework. Available options are advanced needed programming skills (ADVANCED), no programming skills required (NOT REQUIRED) or it only required for advanced test scripts (PARTIAL) [14]. **DevOps/ALM Integration Support** relates to the framework's ability to support integration with DevOps or ALM environments (YES, NO) [14]. **Continuous Integration (CI) Support** indicates if the framework supports CI frameworks (YES, NO) [14]. **Script Language** specifies the programming language(s) required for creating test scripts (e.g. Python, Structured Text, Function Block Diagram, Ladder) [14]. **Report Format** indicates how the test reports are represented in a framework (HTML, XML, CSV) [15]. **Availability of Customer Support** evaluates the level of support and tutorials provided for the users of a certain tool (HIGH, MEDIUM, LOW) [13]. **Graphical User Interface (GUI)** investigates the suitability of the designed graphical user interface of a framework. Available options are; The GUI is designed properly and is powerful enough to cover almost all the available functionalities of a framework (YES); The provided GUI is user-friendly, but does not provide a graphical representation of all functionalities of a framework (PARTIAL); The GUI exists but it is limited in its functionality (LIMITED); the framework has no GUI (NO) [13]. **Quality of Documentation** Specifies the framework's level of the documentation and tutorial which is provided by the framework developers. Available options are; The framework is well-documented and a wide range of updated tutorials and framework specifications are easily accessible online (VERY GOOD); The framework is documented properly but the provided documentation is not easily accessible or it is only available offline (GOOD); The framework is not documented sufficiently or it can not be accessed easily (INSUFFICIENT) [13]. **Maintenance Support** implies the level of maintenance support that is provided and whether testing new functions in a POU under test is easy or not (EASY/HARD).

> *Results RQ2: We discovered several features that should be considered when choosing a test automation framework for PLC testing: cost, supported platforms, industrial use, stage of development, documentation and report generation, record playback, test suite support, test suite extension, team support, DevOps/ALM support, continuous integration support, scripting language, import support, availability of customer support, quality of documentation, and maintenance support.*

C. RQ3 - Test Automation Frameworks

We conducted an initial comparative examination given the features identified in the previous section. We focus on Test Manager and CoUnit as our chosen test automation frameworks in CODESYS IDE. The results of this comparative examination are shown in Table VI. Even if both frameworks support only Windows platforms and have continuous integration support, we can observe significant differences. In terms of cost, CODESYS Test Manager is a commercial product but available for academics to use in their research. CoUnit is an open-source software freely available. Industrial usage of Test Manager is considered HIGH since its use has been

reported in several industry-related reports [18] [19] [20] [21] [22]. Regarding the framework's maturity, CODESYS Test Manager seems to be more mature and has evolved through eight different versions so far, compared to CoUnit (i.e., in 3 versions). One of the main advantages of CODESYS Test Manager is the ability to record and playback that is not supported by the counterpart framework. Both frameworks support test suites in .xml file format. In addition, CODESYS Test Manager has the advantage of supporting .tsd (Tamino Schema) extension (used as a container of elements that a Tamino XML Server document contains). CODESYS Test Manager supports test suites to be extended by using specific predefined test commands, but the extension of test suites in CoUnit demands ST programming knowledge, and the user needs to instantiate the code for each single test case. CODESYS Test Manager supports Python and all IEC 61131-3 programming languages for developing the test scripts, while CoUnit only supports the ST programming language. Availability of customer support is another essential factor from an industry point of view in this comparison, and CODESYS Test Manager seems to be superior in this respect. The quality of the documentation provided by the CODESYS Test Manager is excellent since comprehensive educational material and good video tutorials are available. On the other hand, CoUnit provides less documentation and tutorials. Maintenance support is another important feature proposed. CODESYS Test Manager supports direct main PLC program testing and one instantiation of the code under test can be used in all related test suites but these features are not available in CoUnit.

> *Results RQ3: Based on our initial comparison between CODESYS Test Manager and CoUnit based on the 15 industry-validated features, the results show that CODESYS Test Manager is more mature and has several advantages over CoUnit, including user support, record and playback features, and easy test suite extension. Nevertheless, CoUnit, as an open-source counterpart, also provides testers with many key features used during PLC testing.*

D. RQ4 - Applicability in an Industrial Case Study

Aiming to answer this research question, we applied the two test automation frameworks we found through our GLR to an industrial case study by considering several possible test scenarios. Our case is a control system provided by a large automation company in Sweden consisting of several POUs. This system is developed in the FBD programming language. The Function Block (FB) in this POU consists of several computational blocks executed cyclically. The program executes in a cyclic loop where every cycle contains three phases: read (reading all inputs and storing the input values), execute (computation without interruption), and write (update the outputs). The FBD program is created as a composition of interconnected blocks with data flow communication. When activated, a program consumes one set of input data and then

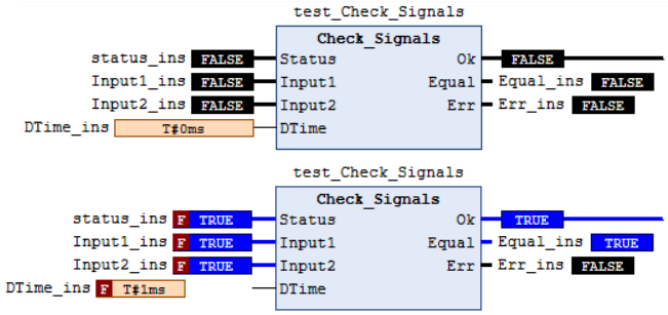

Fig. 3: An example of the POU under test before (top) and after execution (bottom) of the developed test scripts in CODESYS Test Manager

executes to completion. We considered functional scenarios for testing the POU. We evaluate this functionality and the applicability of *Test Manager* and *CoUnit* by automating the test execution for the provided case and all POUs. To this end, we generated several test suites consisting of manually created test cases.

To automate the test execution in CODESYS Test Manager, the first step needed is instantiating the POU in the main PLC program. Next, we create the required test suites containing 10 test cases. Each test case includes several test actions that are supposed to alter the values of the inputs and compare the output of the PLC program with the expected result. For example, five test cases target the functionality of the TON block. In these test cases, we provide just one active input signal with the expected output *Err* being true. The subsequent five test cases use two active signals at different random time slots. After executing the developed test scripts in CODESYS Test Manager, the results of running the PLC program can be observed during execution as shown in Figure 3. After running the test suite in CODESYS Test Manager, we observed all the test cases passing, as can be seen in Figure 4. The CODESYS Test Manager automatically generates a test report in HTML which includes information on e.g., test settings, test result and status, execution time, pinned scripts.

To evaluate the applicability of CoUnit, we developed ten different test cases. Since CoUnit does not contain any GUI interface facilitating the creation of test cases, we developed the test scripts in ST language. Moreover, unlike CODESYS Test Manager, CoUnit does not support the use of the entire Program as a testable object type, and it only supports this functionality at the POU level. We instantiate the POU for every defined test case. In addition, the framework needs to be added as a library into the target PLC program. Also, for each POU under test, the user needs to develop a dedicated function block as a test counterpart, which is responsible for four main tasks, including instantiating the POU under test, defining the inputs, describing the expected output, and finally, calling the CoUnit assertion methods to compare the expected output with the actual one. A snippet of the developed test suite for the CoUnit test automation framework implemented

TABLE VI: An Overview of the Comparison between CODESYS Test Manager and CoUnit based on the Validated Features.

Feature	Test Manager	CoUnit
Cost	MIX *Commercial license, but free to use for academic purposes	FREE *Open Source license
Supported Platforms	Microsoft Windows	Microsoft Windows
Industrial Use	HIGH	LOW
Stage of Development	MATURE *8 versions released so far	PARTIAL *3 versions released so far
Documentation and Report Generation	COMPLETE	SUMMARIZED
Playback Record	YES *Can be realized via the Test Progress feature	NO
Test Suite Support	YES *.tsd, .xml extensions are supported	YES *.xml extension is supported
Test Suite Extension	EASY *New test cases can easily be developed using the available graphical test commands. *One instantiation of the POU under test can be used in all new test suites and test cases *The number of test cases inside a test suite is not limited	HARD *New test cases need to be developed in ST language *For every new test case a new distinct instantiation of the POU under test is required *Every test suite can only contain 100 test cases
Teamwork Support	NO	NO
DevOps/ALM Integration Support	No Information Provided	No Information Provided
Continuous Integration (CI) Support	Yes	Yes
Script Language	Python, All IEC 61131-3 Supported Programming Languages	Structured Text (ST)
Availability of Customer Support:	YES(Official CODESYS costumer support is available)	NO
Quality of Documentation	VERY GOOD *Both tool documentations and official tutorial videos are available online	GOOD *Tool documentations and textual tutorial are available online
Maintanance Support	EASY *Direct testing of the PLC main program is supported *GUI and graphical test commands are available	HARD *Direct testing of the main PLC program is not supported *GUI and graphical test commands are not available

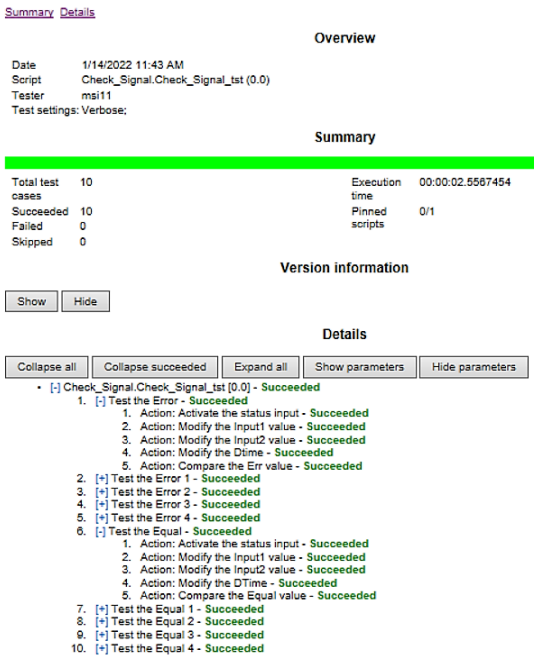

Fig. 4: A generated test report in CODESYS Test Manager

in the ST programming language is shown in Figure 5. After running CoUnit in the main program (PRG_Test in Figure 5), the framework automatically executes the defined test cases on the POUs under test. When the test execution ends, CoUnit generates a test report in the XML format that provides users with information about the test results, including test suite name, the number of test cases, test case name, test case status (PASS or FAIL), and the class name.

Finally, we report our overall experiences in using both test automation frameworks. The following results and features are PLC-specific. Regarding **installation and configuration**, we found out that setting-up CODESYS Test Manager seems to be more straightforward since it can be installed as a standard add-on package. On the other hand, CoUnit needs to be installed as a package and imported as a library in

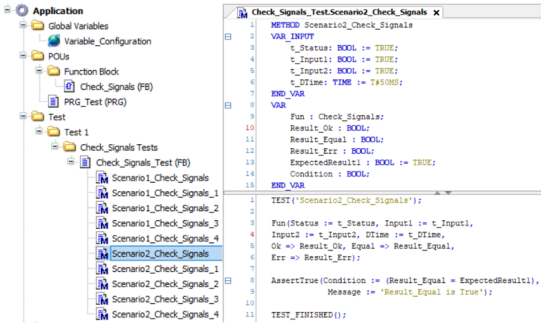

Fig. 5: A test suite developed in the CoUnit

every project under test. Regarding the **ease of use**, CODESYS Test Manager is more user-friendly and provides features for developing test scenarios using available test commands in the GUI integrated in CODESYS IDE. Moreover, developing test cases with this framework does not require the use of any of the IEC61131-3 programming languages. On the other hand, creating the same test cases in CoUnit is more time-consuming due to the use of ST scripts and instantions. When comparing the frameworks' capabilities related to **testable objects**, we found out that the Test Manager can create harnesses for PLC applications, IEC libraries, and communications. In contrast, CoUnit can only be used at the application level. Regarding **test assertion timeouts** we note here that PLC programs are executed cyclically in a loop, and one needs to set a test assertion timeout to make sure that the result comparison process ends after a certain amount of time. Only CODESYS Test Manager can be used to set a custom timeout, a useful feature when testing complex PLC programs. After executing test scripts on both frameworks, we discovered that test reports generated by CODESYS Test Manager provide the user with detailed information. On the other hand, CoUnit only reports scarce information.

Results RQ4: Using the discovered features as a basis, the application on an industrial PLC

program revealed that both frameworks provide proper automation functionality. However, CODESYS Test Manager seems to be more mature, provides more helpful test execution features, and is more user-friendly. In contrast, CoUnit seems limited in its usefulness, and working with it requires ST programming.

E. Threats to Validity

In this section, we discuss some of the threats to validity for this study. To address *internal validity*, we iterated on the search strings by conducting several initial searches. Aiming at minimizing the bias in the process of interpretation, analysis, and selection of the gathered sources, we made sure that at least two authors of this paper reviewed each source. The extraction method we used for filtering and categorizing the gathered data in the pool of contents is based on the systematic qualitative analysis approach proposed by Huberman et al. [12]. When considering the *construct validity* of our study, we employed already proposed methods [4], [5]. Consequently, the data has been examined and checked multiple times to realize an agreement on the obtained features in this study. Regarding *external validity*, since this research is conducted in a very specific domain and it only focused on test automation tools of a particular PLC IDE, more studies are needed to generalize the process of choosing a test automation framework.

V. Conclusions and Future Work

This paper addresses the practical problem of choosing the right test automation tool for PLC programs in CODESYS IDE. First, we identified the most-discussed test automation tools of CODESYS by performing a GLR on existing test automation frameworks of CODESYS followed by a qualitative analysis based on several criteria. Aiming at performing an effective comparison between the discovered test automation frameworks of CODESYS, we identified 29 features as important features of test automation tools through conducting a literature review. Finally, to investigate the applicability of the discovered tools on a real-world case study, we performed an automated test execution on an industrial case study based on two different test scenarios. Our findings imply that both discovered test automation tools of CODESYS provide user with necessary automation functionalities but CODESYS Test Manager seems more mature, has more useful test execution features, and is more user-friendly. By contrast, CoUnit is not user-friendly, is limited in its automation features, and working with it demands ST programming knowledge. In future research, we plan to conduct a more comprehensive evaluation of CODESYS Test Manager when used in an industrial context as well as how to connect such a test automation framework for automated testing of timer and stateful blocks in PLCs.

VI. Acknowledgment

This work has received funding from EU's H2020 research and innovation program under grant agreement No 957212.

References

[1] Irfan Ahmed, Sebastian Obermeier, Sneha Sudhakaran, and Vassil Roussev. Programmable logic controller forensics. *IEEE Security & Privacy*, 15(6):18–24, 2017.

[2] Elfriede Dustin, Jeff Rashka, and John Paul. *Automated software testing: introduction, management, and performance*. Addison-Wesley Professional, 1999.

[3] Eliane Figueiredo Collins and Vicente Ferreira de Lucena. Software test automation practices in agile development environment: An industry experience report. In *International Workshop on Automation of Software Test (AST)*, pages 57–63. IEEE, 2012.

[4] Vahid Garousi, Wasif Afzal, Adem Çağlar, İhsan Berk Işık, Berker Baydan, Seçkin Çaylak, Ahmet Zeki Boyraz, Burak Yolaçan, and Kadir Herkiloğlu. Comparing automated visual gui testing tools: an industrial case study. In *International Workshop on Automated Software Testing*, pages 21–28, 2017.

[5] Päivi Raulamo-Jurvanen, Mika Mäntylä, and Vahid Garousi. Choosing the right test automation tool: a grey literature review of practitioner sources. In *International Conference on Evaluation and Assessment in Software Engineering*, pages 21–30, 2017.

[6] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Springer, 2010.

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

[8] Marcin Jamro. Pou-oriented unit testing of iec 61131-3 control software. *IEEE Transactions on Industrial Informatics*, 11(5):1119–1129, 2015.

[9] Florian Hofer and Barbara Russo. Iec 61131-3 software testing: A portable solution for native applications. *IEEE Transactions on Industrial Informatics*, 16(6):3942–3951, 2019.

[10] Päivi Raulamo-Jurvanen, Simo Hosio, and Mika V Mäntylä. Practitioner evaluations on software testing tools. In *Proceedings of the Evaluation and Assessment on Software Engineering*, pages 57–66. 2019.

[11] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In *International Conference on Evaluation and Assessment in Software Engineering*, pages 1–6, 2016.

[12] A Michael Huberman and Johnny Saldana Matthew B Miles. *Qualitative data analysis: A methods sourcebook*. 2019.

[13] Alessio Ferrari, Franco Mazzanti, Davide Basile, and Maurice Ter Beek. Systematic evaluation and usability analysis of formal methods tools for railway signaling system design. *IEEE Transactions on Software Engineering*, 2021.

[14] Mubarak Albarka Umar and Chen Zhanfang. A study of automated software testing: Automation tools and frameworks. *International Journal of Computer Science Engineering (IJCSE)*, 6:217–225, 2019.

[15] Heidilyn Veloso Gamido and Marlon Viray Gamido. Comparative review of the features of automated software testing tools. *International Journal of Electrical and Computer Engineering*, 9(5):4473, 2019.

[16] Harpreet Kaur and Gagan Gupta. Comparative study of automated testing tools: selenium, quick test professional and testcomplete. *Journal of Engineering Research and Applications*, 3(5):1739–1743, 2013.

[17] Emil Borjesson and Robert Feldt. Automated system testing using visual gui testing tools: A comparative study in industry. In *International Conference on Software Testing, Verification and Validation*, pages 350–359. IEEE, 2012.

[18] Sebastian Ulewicz and Birgit Vogel-Heuser. Increasing system test coverage in production automation systems. *Control Engineering Practice*, 73:171–185, 2018.

[19] Sebastian Ulewicz and Birgit Vogel-Heuser. Guided semi-automatic system testing in factory automation. In *International Conference on Industrial Informatics (INDIN)*, pages 142–147. IEEE, 2016.

[20] Sebastian Ulewicz and Birgit Vogel-Heuser. System regression test prioritization in factory automation: Relating functional system tests to the tested code using field data. In *Annual Conference of the IEEE Industrial Electronics Society*, pages 4619–4626. IEEE, 2016.

[21] Sebastian Ulewicz and Birgit Vogel-Heuser. Industrially applicable system regression test prioritization in production automation. *Transactions on Automation Science and Engineering*, 15(4):1839–1851, 2018.

[22] GIACOMO Barbieri, GABRIEL Quintero, OSCAR Cerrato, JULIAN Otero, DAVID Zanger, and ALEJANDRO Mejia. A mathematical model to enable the virtual commissioning simulation of wick soilless cultivations. *J. Eng. Sci. Technol*, 16:3325–3342, 2021.