An Experiment in Requirements Engineering and Testing using EARS Notation for PLC Systems

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

***Abstract*— Regulatory standards for engineering safety-critical systems often demand both traceable requirements and specification-based testing, during development. Requirements are often written in natural language, yet for specification purposes, this may be supplemented by formal or semi-formal descriptions, to increase clarity. However, the choice of notation of the latter is often constrained by the training, skills, and preferences of the designers.**

The Easy Approach to Requirements Syntax (EARS) addresses the inherent imprecision of natural language requirements with respect to potential ambiguity and lack of accuracy. This paper investigates requirement formalization using EARS and specification-based testing of embedded software written in the IEC 61131-3 language, a programming standard used for developing Programmable Logic Controllers (PLC). Further, we investigate, by means of an experiment, how human participants translate natural language requirements into EARS and how they use the latter to test PLC software. We report our observations during the experiments, including the type of EARS patterns participants use to structure natural language requirements and challenges during the specification phase, as well as present the results of testing based on EARS-formalized requirements.

***Index Terms*—EARS, PLC, Requirement Engineering, Testing**

I. Introduction

Programmable Logic Controllers (PLCs) are used in engineering embedded safety-critical software (e.g., in the railway and automation control domains) [1]. Engineering such systems commonly demands certification according to safety standards [2] that impose specific constraints on requirements engineering, implementation-based and specification-based testing. Several studies [3]–[6] have looked at how to generate test input data to achieve high implementation coverage for domain-specific PLC systems.

However, since requirements are often expressed in natural language, using them in such form to create test cases while keeping requirements and test cases aligned is a difficult task. While such an alignment requires extensive domain knowledge, a systematic process for requirements engineering – including their translation into a semi-formal, non-ambiguous form – combined with testing would facilitate linking requirements to tests. Generally, in industry, such translation is most often being carried out manually, so manual processes are used to model requirements by using structured notations, and automatically create a set of tests that systematically exercises the specification when fed to the system under test [7]. Given that there is little evidence on the extent to which humans can effectively model requirements using semi-formal notations, and how the modeling impacts the development and testing of reliable systems, in this paper, we investigate the implications of applying semi-structured requirements specification and test generation based on the latter, for PLC systems. In this context, we study the behaviour of human writing requirements in different modeling styles using the Easy Approach to Requirements Syntax (EARS) [8], a simple notation for writing textual requirements that were first published at the IEEE Requirements Engineering 2009 Conference.

For requirement modeling, we chose the *Easy Approach to Requirements Syntax*(EARS) and testing by experimenting with human subjects. Ten individuals took part as subjects in an experiment. The subjects were given three requirements and were asked to rewrite these using EARS notation based on a specification manually.

The results of this study show that humans create requirements using semi-formal notations in distinct ways and using different patterns. Completeness is the most common issue when rewriting and using such requirements for testing. Additionally, we found that test generation and execution using these EARS requirements for PLC systems is applicable. Our results highlight the need for more research into how different requirement formalization and test design techniques for PLC software can influence the efficiency and effectiveness of requirements engineering and requirements-based testing for this type of software.

II. PRELIMINARIES

A. Programmable Logic Controllers

Programmable Logic Controllers (PLC) are one of the most used logic controllers in today's automated industry [9]. PLCs are being widely used in different industrial applications such as supervisory systems in nuclear and power plants. Programming a PLC device is usually done via one or a combination of different programming languages that are proposed in the IEC 61131-3 standard [10]. The aforementioned PLC programming languages are Function Block Diagram (FBD), Structured Text (ST), Ladder Diagram (LD), Sequential Function Chart (SFC), and Continuous Function Chart (CFC). Among all introduced programming languages for PLCs, FBD and ST are our main focus in this study because of two reasons. Firstly, these two languages gained remarkable popularity in the industry during the last couple of years [11]. Secondly, the industrial case

study that is provided to us for this study is a supervisory PLC program that is developed in ST and FBD. ST is a text-based programming language with a similar syntax to high-level programming languages such as C, whereas, FBD is a visualized programming language that is easy to use because of its graphical interface. PLC programs are commonly developed in an Integrated Development Environment (IDE) and are executed cyclically. Based on the provided concept in IEC 61131-3, each cycle loop of PLC program execution consists of 3 main stages including read, execute, and write [10]. The first stage reads all available inputs and stores them in the memory while the second stage does the computation tasks without any interruptions. The final stage (write), updates the output values based on the completed computations in the previous stage.

B. CODESYS Development Environment

Developing a PLC program and simulating its behavior needs to be done in an IDE. Several different PLC IDEs have been proposed by different vendors so far. One of the most popular IDEs in the market is CODESYS [1] which was initially developed by CODESYS Group in 1994. CODESYS is a manufacturer-independent IDE that has matured by releasing numerous updates and the latest version at this moment is V3.5 SP18. Among all available PLC IDEs in the market, We have chosen CODESYS as our preferred IDE because of several reasons. Firstly, CODESYS is very popular among practitioners and has almost full compatibility with the IEC61131-3 standard and supports all proposed standard programming languages of this standard [11]. Secondly, CODESYS is free to use for personal use and is equipped with good support through releasing different versions. Last but not least, CODESYS can execute Python scripts directly inside the IDE and it is also equipped with numerous automation add-ons such as test automation tools.

C. EARS Semi-Structured Requirement Engineering Syntax

Writing the stakeholder requirements in unconstrained Natural Language (NL) is not accurate and can raise critical problems in lower levels of system development [8]. Aiming at mitigating the ambiguity problems and increasing the accuracy in the process of requirements engineering, some practitioners stand up for using other textual and non-textual notations [8]. Using non-textual notations demands translation of the original requirement, which can be faulty sometimes. Training overhead is another drawback of proposing a new type of notation. EARS is a semi-structured requirement engineering syntax that was proposed by Alistair et al. in 2009 [8]. EARS provides a syntax for transforming all-natural language requirements in one of the proposed five Generic requirements syntax simple templates. The aforementioned five simple templates of EARS are ubiquitous requirements, event-driven requirements, unwanted behaviours, state-driven requirements, and optional features. Moreover, EARS supports writing complex requirements using a combination of considered conditional keywords, including *Where*, *While*, and *When*.

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

III. EXPERIMENTAL DESIGN

In this section, we report the description of the performed experiment, including the details of the instruction material and the artefacts used.

A. Research Questions

The main goal of this study is to investigate the process of requirements creation when constraining the use of NL. The EARS modeling notation has been adopted by other organizations in different sectors and countries, so it is a realistic model for requirements engineering and test creation. Since these are intellectual activities in which humans allocate a variety of cognitive resources (such as attention and effort) that one needs to use when confronted with challenges as they perform such tasks, our first step is understanding how human practitioners write such requirements and how these can be used for test creation.

The main goal of this study is to investigate the applicability of the EARS semi-structured requirement engineering syntax in the context of PLC programs. Aiming at achieving this goal, we formulated the following research questions.

- RQ1: How is the EARS semi-structured requirement engineering syntax and test creation applied in the context of PLC programs?
- RQ2: What EARS patterns are used during the writing of requirements?
- RQ3: What challenges are perceived during the specification of requirements and test creation using EARS?

B. Experimental Setup Overview

Aiming at achieving the goal of this study, we conduct a controlled experiment that asks the participants to write 3 given requirements using EARS syntax. The participants are free to choose their preferred EARS syntax template based on their personal interpretation of the given requirements. The *subjects* of this experiment are a group of 10 individuals as follows: four experienced engineers at a large automation company in Sweden and Spain and six researchers and managers from different universities and research institutions across Europe.

C. Object Selection

The objects of study were chosen manually based on the following criteria:

- The requirements should have a natural language specification that is understandable and sufficiently rich in detail for an engineer to write executable tests.
- The requirements should represent different types of real testing scenarios in different areas where the IEC 61131-3 standard is used.
- The requirements should be simple to understand without any domain knowledge.

TABLE I
THE NATURAL LANGUAGE REQUIREMENTS USED DURING THE EXPERIMENT.

Requirement ID	Requirement Text
RI1	User account should be uniquely identified to a user.
RI2	The software shall warn the user of malware detection.
RI3	Only authorised devices are allowed to connect into the ICS network

- The resulting test cases should be executed in the CODESYS environment.

We investigated the industrial libraries provided by a large-scale company focusing on the development and manufacturing of control systems. We identified three candidate requirements matching our criteria, shown in Table I. The requirements should not be trivial, yet fully manageable to use within 60 minutes and no domain-specific knowledge should be needed to understand the requirements. We then assessed the relative difficulty of the identified requirements by manually writing and creating tests.

D. Operationalization of Constructs

Requirements Templates. In this experiment, we investigate the effect of using the EARS approach for requirements engineering and test creation. The proposed generic requirements syntax of EARS we used in this experiment works as follows:

<optional preconditions><optional trigger> the <system name> shall <system response>

This simple syntax template forces the requirement engineer to emphasise preconditions, triggers, and system responses in their developed requirements. In EARS syntax, preconditions, and triggers are both optional, and the order of the used clauses is very important. The following briefly describes each template of EARS.

1) Ubiquitous requirements (U): A ubiquitous requirement is a type of requirement that is not bonded to any preconditions or triggers and is always enabled in the system. The generic structure of this template is as follows:

The <system name> shall <system response>

2) Event-driven requirements (ED): The event-driven requirement is used only when an event is identified in the system. This type of requirement uses *When* keyword. The generic structure of this template is as follows:

WHEN <optional preconditions> <trigger> the <system name> shall <system response>

3) Unwanted behaviours (UB): Requirements that are related to Unwanted behaviors are defined using a structure that is extracted from Event-driven requirements. *Unwanted behavior* refers to covering all possible situations that are not desirable and are usually a big source of omissions in preliminary requirements. The reserved keywords for this type of requirement in EARS are *If* and *Then*. The generic structure of this template is as follows:

IF <optional preconditions> <trigger>, THEN the <system name> shall <system response>

4) State-driven requirements (SD): The State-driven requirement is only active if the system is in a specific status. The reserved keyword for defining State-driven requirements in EARS is *While*. The generic structure of this template is as follows:

WHERE <feature is included> the <system name> shall <system response>

5) Optional features (OF): The *Optional feature* requirement is designed to be used when the author of the requirement wants to include a specific feature in the system. The keyword *Where* is considered for defining this type of feature in EARS. The generic structure of this template is as follows:

WHERE <feature is included> the <system name> shall <system response>

Process Challenges. We are interested in two types of challenges encountered during the use of EARS templates and their use for testing: challenges encountered during the specification of requirements and problems when designing test cases for PLC systems. We performed thematic analysis [12] for qualitative data analysis to extract the main themes as reflected by the input given by each participant.

E. Instrumentation

One session was organized for the sake of the experiment. The subjects were given the task to use the three requirements and rewrite these in EARS (to the extent they consider sufficient based on the given specifications). They were instructed to read the specification, create these templates and think out loud. The subjects were not grouped and the document needed for this experiment was provided digitally and in written form. Before commencing the session, a short tutorial of approximately 10 minutes on EARS syntax was provided to the subjects in order to avoid further problems with the subjects' unfamiliarity with the concepts used. The tutorial included screencasts demonstrating EARS requirements. Detailed information about the problem and instructions were provided in the experiment session.

F. Data Collection Procedure

As part of the instructions, subjects submitted their solutions in the form of a record documenting their work. Data from

this experiment session was then used for quantitative and qualitative analysis.

IV. EXPERIMENT CONDUCT

Once the experiment design was defined, the requirements for executing the experiment were in place. The session was held for one hour and preceded by a lesson on EARS notation. The requirements specification and testing process used during the conduct of this experiment corresponds to the methodology in Figure 1. The first step corresponds to the transformation of the requirement specified initially in Natural Language (NL) into an EARS requirement using the EARS syntax (Step 1 in Figure 1. In the next step, we are using the resulting requirement to generate test cases that cover the specified behaviour (Step 2 in Figure 1). The final steps in this methodology are to execute these test cases (Step 3 in Figure 1) and to compare the actual behaviour with the expected result to monitor whether the program works as expected (Step 4 in Figure 1).

In total, *ten individuals* participated in our experiment. Before starting the experiment, the participants were informed that their work would be used for experimental purposes. The participants had the option of not participating in the experiment and not allowing their data to be used this way.

The subjects worked individually during the experiment; we briefly interacted with the participants to ensure that everybody had a sufficient understanding of the involved notations without getting involved in the writing of the solution. All subjects used the provided documents and their machines. The experiment was fixed to one hour. To complete the assignment, the subjects were given the same time to work on writing these requirements according to the given instructions. For collecting data, we provided a template to enforce the usage of the same reporting interface. By having a common template for reporting, we eased the data collection and analysis process.

To finish the assignment, we required the participants to provide the produced results as soon as they finished writing their responses. During the experiment, the subjects do not directly communicate with others to avoid introducing bias. After each individual finished their assignment, a complete solution was saved containing the answers for each solution. In addition, we separated the data provided by the participants from their names.

V. EXPERIMENT ANALYSIS

This section provides an analysis of the data collected in this experiment. In analyzing the qualitative data, we followed the guidelines on qualitative analysis procedures provided by Braun and Clarke [12]. For each requirement, each subject in our study provided a set of EARS expressions. These expressions were used to conduct the experimental analysis and testing. For each set of tests produced, we provide evidence for their generation and execution in CODESYS. These metrics form the basis for our analysis toward answering the research questions.

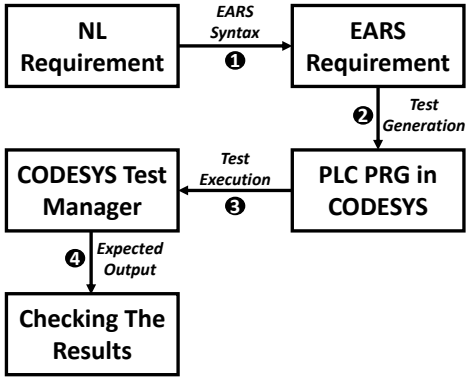

Fig. 1. An overview of the EARS-based requirement specification and PLC testing methodology used in this experiment.

TABLE II
RESULTS OF THE TEMPLATES USED FOR EACH REQUIREMENT USED IN THE EXPERIMENT.

RI1	RI2	RI3	Requirement ID/EARS Template
10	1	1	**Ubiquitous (U)**
0	5	4	**Event-Driven (ED)**
1	5	6	**Unwanted Behaviours (UB)**
0	0	3	**State-Driven (SD)**
0	0	0	**Optional Features (OF)**

A. Requirement Engineering Results

For each requirement, we have collected data about the type of EARS template used by each participant, the approaches, and the challenges participants experienced during requirement representation using the EARS notation. The results are shown in Table II, Table III, and Table IV.

Participants strictly adhered to one or multiple EARS templates. It seems that the ubiquitous template has been used by all participants to model requirement RI1 and just in one case when representing requirements RI2 and RI3 (as shown in Table II). Participants explained that the "shall" statement is clearly indicated and should be used to describe the required behavior. Nevertheless, one participant decided to use the unwanted behaviour template for RI1 to indicate the prohibited behavior in such a form that can be used for testing.

TABLE III
RESULTS OF THE REQUIREMENTS WRITING IN TERMS OF THE TEMPLATES USED BY EACH PARTICIPANT FOR EACH REQUIREMENT. EARS TEMPLATE TYPES ARE SHOWN USING THEIR SPECIFIC ACRONYMS AS STATED IN SECTION III-D AND TABLE II.

RI1	RI2	RI3	Requirement ID/Participants
U, UB	U, UB, ED	U, SD, ED	**P1**
U	ED	UB	**P2**
U	ED	UB	**P3**
U	UB	SD	**P4**
U	ED	UB	**P5**
U	ED	UB	**P6**
U	SD	UB	**P7**
U	UB	ED, UB, SD	**P8**
U	UB	ED	**P9**
U	UB	ED	**P10**

TABLE IV
RESULTS SHOWING THE MAIN THEMES IDENTIFIED RELATED TO APPROACHES AND CHALLENGES ENCOUNTERED DURING THE TRANSLATION PROCESS.

Main Themes	Theme Descriptions
Requirements are not complete and clear enough for EARS translation.	When starting with the translation, requirements in NL are not complete enough to decide precisely which EARS template to use.
Using single or multiple EARS templates is not clear enough, especially when using these for testing.	There is a need, when using these patterns for testing, to use multiple and separate templates for each requirement to cover both positive and negative cases arising.
The system perspective is not easily identifiable from the requirements.	It is difficult to decide which perspective to use when translating the EARS requirement (e.g., system, subsystem level).
The optional feature template is not applicable for the selected requirements	Even if the Option requirement is used for systems that include a particular element and variants, this modeling form was not used during requirement transformation using the EARS notation since the participants did not need to handle system or product variation.

The event-driven and unwanted behaviour templates have been used by participants to represent requirement RI2, while some participants used the state-driven pattern (as shown in Table III). Participants chose to do this since they drafted requirements in several increments. Firstly, they considered how the system behaves typically (also called sunny-day behaviour). For some participants using EARS, this results in requirements in the state-driven and event-driven patterns. Secondly, some participants decided to specify what the system must do in response to the unwanted behaviour, which produced requirements in the unwanted behaviour pattern.

In addition, the thematic analysis of the notes taken by participants when performing these steps in requirement representation resulted in several main themes related to approaches and challenges experienced during the translation process. Several participants mentioned that the initial NL requirements are not complete and clear such that these can be used directly for testing. One participant mentioned the following: *"What happens if the device is not authorized, missing failure models, startup/default/safe state...?"*. This resulted in issues when starting with the translation process, especially when deciding which templates to use. Several participants had issues in deciding when to use single or multiple EARS templates to cover both positive and negative behaviours that need to be tested. One participant stated the following: *"We could possibly use event-driven type requirement. At the same time, it is unwanted we could use, this one is quite complicated"*. Some participants preferred the use of the "shall not" form, which has been observed by some participants as having an impact on the test case created since only a set o test cases involving the unwanted behaviour would need to be created to show satisfaction with the requirement. Another observation relates to the use of an optional feature template, which for the given requirements was not used by any of the participants since there was no need to specify any product variation or specific features.

B. PLC Testing Results

Aiming at evaluating the applicability of using EARS semi-structured syntax when creating test cases for PLC programs, we used three programs that implement the behaviour stated in the three provided natural language requirements used in this experiment. All these three PLC programs are developed in CODESYS IDE using the Structure Text (ST) programming language. In this paper, we refer to these programs as *PRG1*, *PRG2*, and *PRG3*.

```
PROGRAM UniqueUserAccount
VAR
   user : ARRAY[1..10] OF WSTRING;;
   user_account : ARRAY[1..10] OF DINT;
   i,j : INT;
   K : INT;
   UniqueID : BOOL;(*Non-Unique ID counter*)
   Result_Unique: BOOL := FALSE;
END_VAR
```

Fig. 2. PRG1 PLC interface program written in the ST language in CODESYS IDE corresponding to the evaluation of the RI1 requirement.

After generating the EARS-based test cases for each program, we execute these automatically using the CODESYS test automation framework named CODESYS Test Manager[2]. The final step in this methodology is to compare the actual output with the expected output to observe whether the program works as expected.

We used the concretization steps of the EARS expressions as stated by Flemstrom et al. [13]. This happens by mapping the system response, condition, and events to the actual implementation in PLC. This contains information about the implementation elements of a system and its interfaces. An engineer needs to consider this information and identify the given signals and their characteristics. In this way, we define a set of signals related to the feature under test. In these cases, the next step for the selected requirements would need to design test cases to show that requirement has been met. In our experiment, we could directly use a subset of positive and negative cases by randomly choosing values from an equivalence class. Nevertheless, in a general case, the translation and concretization steps are not easy and one would need to decide how to automate such steps and if we are to use exhaustive testing, equivalence class testing, combinatorial testing, or any other test selection technique for designing test cases.

1) Test Results of PRG1: PRG1 is the PLC program we considered for testing the RI1 requirement in the PLC environment. This program is using the values of the *user account* and *user* lists. Then it checks for unique IDs and returns an

[2]https://store.codesys.com/en/codesys-test-manager.html

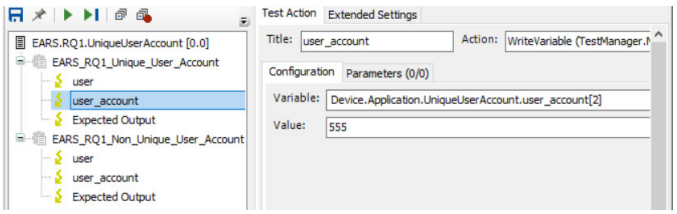

Fig. 3. The generated test cases for PRG1 based on the EARS syntax for RI1 as shown in CODESYS IDE

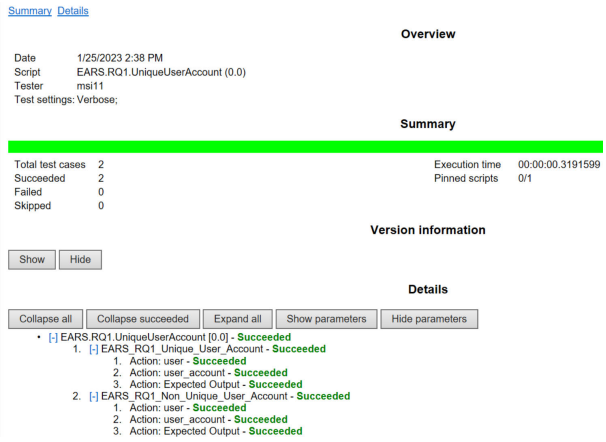

Fig. 4. Test execution results for PRG1 PLC program based on the EARS-based generated test cases for RI1

indication of whether each user account is uniquely identified to a user or not. A snippet of the PRG1 PLC program is shown in Figure 2.

To design and execute the required test cases to test the RI1 Requirement in PRG1, we use the transformed requirement from the NL requirement shown in Table V.

Based on the EARS requirement we use two test cases to cover the identification of the user and the case when the user is not identified. Each test case includes the following three test actions: two *WriteVariable* test actions to alter the *user* and *user account* inputs and one *CompareVariable* test action that compares the actual output with the expected one. The generated test cases for PRG1 used to test the adherence of the program to RI1 requirement are shown in Figure 3.

After designing the required test cases, we execute them automatically on PRG1 to investigate the adherence of the mentioned PLC program to the RI1 requirement. As can be observed in Figure 4, all test cases have been executed in 0.3 seconds. All executed test cases have successfully passed on the PRG1 program.

2) Test Results of PRG2: The PLC program we use for executing the generated test cases for *RI3* in Table I is named PRG2. This program is shown as a black-box malware detection system in the PLC environment that can be used for investigating the context of RI2. PRG2 consists of the following interfaces: two input signals named *MaliciousActivity* and *NormalActivity* as well as one output signal named

```
1  PROGRAM MalwareDetection
2  VAR
3      MalwareDetected: BOOL;
4      MaliciousActivity: BOOL;
5      NormalActivity: BOOL;
6  END_VAR
7
```

Fig. 5. A snapshot showing the PRG2 PLC interface program written in the ST language in CODESYS IDE corresponding to the evaluation of the RI2 requirement.

```
1  PROGRAM SearchID
2  VAR
3    id_to_find : INT := 111;
4    found : BOOL;
5    array_of_ids : ARRAY[0..9] OF INT :=
6  [000,111,222,333,444,555,666,777,888,999];
7    i : INT;
8  END_VAR
```

Fig. 6. A snapshot showing the PRG3 PLC program written in the ST language in CODESYS IDE corresponding to the evaluation of the RI3 requirement.

MalwareDetected. When *MaliciousActivity* and *NormalActivity* signals have divergent information, the Malware Detection system is triggered, and the value of the *MalwareDetected* signal becomes True. An interface snippet of PRG2 is shown in Figure 5.

Considering the results of the experiment we use the resulting EARS *Event-driven requirement* pattern as the most suited type of template for transforming the requirement from NL to EARS in the form shown in Table V.

Based on the developed EARS requirement for RI2 requirement, we generate two test cases for PRG2. Each test case consists of two test actions(*MaliciousActivity* and *NormalActivity*) that alter the value of the inputs, as well as one test action (*Expected Output* that compares the actual behaviour with the expected one. The first test case checks if a(*Malware is Detected*) while the second test case checks if a (*Malware is Not Detected*)

The generated test cases for PRG2 based on the RI2 requirement are then automatically executed using CODESYS Test Manager in 1.71 seconds. All developed test cases have successfully passed.

3) Test Results of PRG3: PRG3 is the PLC program used to execute the generated test cases for *RI3* in Table I (*"Only authorised devices are allowed to connect into the ICS network"*. This program consists of the following units: *1)* a database of authorised device IDs, which is implemented using an array of IDs, *2)* an input signal corresponding to the device ID that needs to be authorised, and *3)* a boolean output signal (i.e., *found*) which returns True in the case of the authorised device being allowed to connect given the ID is known. We show a snapshot of this PLC program in Figure 6.

As discussed in Section V-A, different individuals transformed the NL requirement into the EARS requirement in

TABLE V
EARS REQUIREMENTS EXAMPLES OBTAINED FROM THE EXPERIMENT AND THE RESULTING CONCRETIZED EARS REQUIREMENTS.

Requirements	EARS Requirements	Concretized EARS Requirements
RI1	The <user account system> shall <identify the user> If <the user is not identified> then <user account system> shall <alert>	if <uniqueID=FALSE> then <UniqueUserAccount> shall <Result_Unique=FALSE>
RI2	When <malware is detected> the <system> shall <warn the user>	When <NormalActivity $\neq$ MaliciousActivity> the <MalwareDetection> shall <MalwareDetected=TRUE>
RI3	When <the device is authorised> the <system> shall <grant access to the device>	When <found=TRUE> the <SearchID> shall <ConnectionAllowed=TRUE>

different forms. We use the most common form developed by the participants to transform RI3 to an EARS *Event-Driven* syntax pattern in the following form shown in Table V.

Based on the aforementioned EARS requirement for RI3, we develop 2 test cases for *Successful Authorization* and *Unsuccessful Authorization*. Each developed test case consists of two actions, including the provision of a *new Input ID* and *Comparing the actual output with the expected output*. The generated test cases have been automatically executed on PRG3 using CODESYS Test Manager in 1.14 seconds. Both test cases have successfully passed after being executed on the PRG3 PLC program.

C. Limitations of the Study and Threats to Validity

External Validity. All of our subjects are individuals that have limited experience with EARS. Furthermore, because these practitioners have experience in requirements engineering, we see no reason the use of professionals with deep knowledge of EARS in our study would yield a completely different result. Professionals with experience in EARS would intuitively write better requirements than the ones written by our subjects. Our study has focused on three relatively brief with reduced complexity, but these requirements represent relevant samples they would encounter in practice. We have used the CODESYS tool for automated test creation and execution. There are many tools for developing and executing tests, and these may give different results. Nevertheless, CODESYS is one of the most used development environments for PLCs, and its output in tests is similar to the output produced by other tools.

Internal Validity. All subjects were assigned to perform the experiment at the same time. This was dictated by the way the experiment was organized, with a presentation followed by practical work. Subjects without sufficient knowledge of EARS may affect the final result. To avoid this problem, the session was structured to follow the corresponding EARS lesson. Another threat to internal Validity could arise from using unclear objectives given to the subjects. To address this, we tested the material ourselves.

Construct Validity. Capturing the challenges of requirements engineering and testing is a difficult problem. We rely on human feedback by using a think-out-loud method that gives a rough measure of the challenges encountered.

Conclusion Validity. The results of this study are based on an experiment using 10 participants and three requirements. For each requirement, all participants performed the study, which is a relatively small number of subjects. Nevertheless, this was sufficient to obtain various results showing an effect between the modeling of these different types of requirements.

VI. RELATED WORK

Mavin and Wilkinson [14] reflected on the ten years of EARS [8] and shared some lessons learned in their review paper. For example, they have discovered that users of EARS manage to author more useful draft requirements as they incrementally work to find the appropriate EARS pattern. They recommend that new engineers write several requirements and seek expert review with the application of EARS being more useful if one can apply the following activities: training, thinking, semantics, syntax, and review. In our study, we confirm some of these results even if we do not cover all these activities stated.

Mavin et al. [15] report on the understanding of four experienced EARS practitioners and their reflections on their experiences of applying EARS in different projects and domains over six years. They report the following EARS-specific lesson learned: training should be short, use EARS with or without a tool, use coaching to embed learning, challenge the EARS Patterns, and question if the EARS clauses are necessary and sufficient.

Mäntylä et al. [16] performed a controlled experiment on test case development and requirement review and the effects of time pressure. They saw no statistically significant evidence that time pressure would lower effectiveness or provoke negative influences on motivation, frustration, or performance.

Dalpiaz et al. [17] investigated the adequateness, completeness, and correctness of use cases and user stories for the manual creation of a static conceptual model. They performed a controlled experiment with 118 subjects, and their results show that for creating conceptual models, user stories work better than use cases. Furthermore, user story repetitions and conciseness contribute to these results. However, as we aim with our study, more evidence needs to be provided regarding the aspects that must be considered when selecting and using a modeling and requirement notation.

Weninger et al. [18] report the results of a controlled experiment in which they compared two approaches for defining restricted use case requirements from multiple perspectives, including misuse, understandability, and restrictiveness. Their results indicate the usefulness of the restricted use case modeling approach.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have conducted an experiment in requirements engineering and testing using EARS notation for PLC systems. In the requirement engineering part of our experiment, we found out that most participants preferred the EARS ubiquitous pattern for transforming the RI1 requirement from NL to the EARS syntax, whereas the unwanted behaviour and event-driven patterns were the most popular types for RI2 and RI3 requirement transformations. It was observed that different individuals used different EARS patterns for transforming the same requirement based on their personal interpretation, which shows an acceptable level of flexibility in EARS syntax. In the testing part of our experiment, we investigated the applicability of using the EARS patterns in terms of PLC testing, which has been done by using three PLC programs in the ST language and testing them by executing the test cases. The gathered test execution results show that using EARS in creating requirement-based test cases for PLC programs is promising and can benefit the PLC testers by establishing an easy-to-understand way of expressing test specifications.

In future work, we want to investigate the applicability of using EARS in PLC requirement engineering on other levels of testing and by including more PLC programs. Inspection of the impact of choosing different EARS templates for describing the requirements over the quality of the generated test cases can be another future direction of our work. Moreover, we want to automate our solution and generate test cases from the created EARS requirements based on existing functional and non-functional requirements.

ACKNOWLEDGMENT

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

REFERENCES

[1] Moses D Schwartz, John Mulder, Jason Trent, and William D Atkins. Control System Devices: Architectures and Supply Channels Overview. In *Sandia Report SAND2010-5183*. Sandia National Laboratories, 2010.

[2] CENELEC. 50128: Railway Application–Communications, Signaling and Processing Systems–Software for Railway Control and Protection Systems. In *Standard Report*. 2001.

[3] Eduard P Enoiu, Adnan Čaušević, Thomas J Ostrand, Elaine J Weyuker, Daniel Sundmark, and Paul Pettersson. Automated Test Generation using Model Checking: an Industrial Evaluation. In *International Journal on Software Tools for Technology Transfer*, volume 18, pages 335–353. Springer, 2014.

[4] Yi-Chen Wu and Chin-Feng Fan. Automatic Test Case Generation for Structural Testing of Function Block Diagrams. In *Information and Software Technology*, volume 56. Elsevier, 2014.

[5] E. Jee, J. Yoo, S. Cha, and D. Bae. A data flow-based structural testing technique for FBD programs. In *Information and Software Technology*, volume 51, pages 1131–1139. Elsevier, 2009.

[6] Kivanc Doganay, Markus Bohlin, and Ola Sellin. Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study. In *International Conference on Software Testing, Verification and Validation Workshops*, pages 425–432. IEEE, 2013.

[7] Vahid Garousi and Junji Zhi. A Survey of Software Testing Practices in Canada. In *Journal of Systems and Software*, volume 86, pages 1354–1376. Elsevier, (2013).

[8] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. Easy approach to requirements syntax (ears). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322. IEEE, 2009.

[9] David M Auslander, Christopher Pawlowski, and John Ridgely. Reconciling programmable logic controllers (plcs) with mechatronics control software. In *Proceeding of the 1996 IEEE International Conference on Control Applications*, pages 415–420. IEEE, 1996.

[10] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*, volume 166. Springer, 2010.

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

[12] Virginia Braun and Victoria Clarke. *Thematic analysis*. American Psychological Association, 2012.

[13] Flemström Daniel, Enoiu Eduard, Azal Wasif, Sundmark Daniel, Gustafsson Thomas, and Kobetski Avenir. From natural language requirements to passive test cases using guarded assertions. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 470–481. IEEE, 2018.

[14] Alistair Mavin Mav and Philip Wilkinson. Ten years of ears. *IEEE Software*, 36(5):10–14, 2019.

[15] Alistair Mavin, Philip Wilksinson, Sarah Gregory, and Eero Uusitalo. Listens learned (8 lessons learned applying ears). In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 276–282. IEEE, 2016.

[16] Mika V Mäntylä, Kai Petersen, Timo OA Lehtinen, and Casper Lassenius. Time pressure: a controlled experiment of test case development and requirements review. In *Proceedings of the 36th International Conference on Software Engineering*, pages 83–94, 2014.

[17] Fabiano Dalpiaz and Arnon Sturm. Conceptualizing requirements using user stories and use cases: a controlled experiment. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 221–238. Springer, 2020.

[18] Markus Weninger, Paul Grünbacher, Huihui Zhang, Tao Yue, and Shaukat Ali. Tool support for restricted use case specification: Findings from a controlled experiment. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 21–30. IEEE, 2018.