Can Pairwise Testing Perform Comparably to Manually Handcrafted Testing Carried Out by Industrial Engineers?

Peter Charbachi*, Linus Eklund*, and Eduard Enoiu*
*Mälardalen University, Västerås, Sweden.

***Abstract*—Testing is an important activity in engineering of industrial software. For such software, testing is usually performed manually by handcrafting test suites based on specific design techniques and domain-specific experience. To support developers in testing, different approaches for producing good test suites have been proposed. In the last couple of years combinatorial testing has been explored with the goal of automatically combining the input values of the software based on a certain strategy. Pairwise testing is a combinatorial technique used to generate test suites by varying the values of each pair of input parameters to a system until all possible combinations of those parameters are created. There is some evidence suggesting that these kinds of techniques are efficient and relatively good at detecting software faults. Unfortunately, there is little experimental evidence on the comparison of these combinatorial testing techniques with, what is perceived as, rigorous manually handcrafted testing. In this study we compare pairwise test suites with test suites created manually by engineers for 45 industrial programs. The test suites were evaluated in terms of fault detection, code coverage and number of tests. The results of this study show that pairwise testing, while useful for achieving high code coverage and fault detection for the majority of the programs, is almost as effective in terms of fault detection as manual testing. The results also suggest that pairwise testing is just as good as manual testing at fault detection for 64% of the programs.**

***Index Terms*—combinatorial testing, manual testing, industrial control software, fault detection, code coverage, PLC.**

I. Introduction

Software testing [1] is an important activity used for verification and validation by observing the software, executed using a set of test inputs. In practice, engineers are creating these inputs based on different test goals and test design techniques (e.g., specification-based, random, combinatorial, code coverage-based). These techniques have so far been performed manually or semi-automatically with respect to distinct software development activities (i.e, unit and integration testing). With the emerging use of large complex software products, the traditional way of testing software has changed; engineers need to deliver high-quality software while devoting less time for properly testing the software.

In practice, test suites are still created manually by handcrafting them using specific test design techniques and domain-specific experience. Although the automatic or semi-automatic creation of test suites has been the focus of a great deal of research, manual testing is still widely used [2], [3] in the software development industry. However, over the past few decades, several test design techniques [1] have been proposed for the creation of test suites with less effort. Combinatorial testing [4] is a technique that creates test inputs based on combinations among the input values. Pairwise testing is an approach to combinatorial testing that generates a test suite which covers each combination of value pairs at least once. There is some evidence [5], [6] suggesting that pairwise testing is efficient and effective at detecting software faults. However, even if pairwise techniques have been found useful and applicable in industrial applications, the experimental evidence regarding its effectiveness in practice is still limited.

In this paper we compare pairwise testing and manual testing performed by industrial engineers on industrial software created using the IEC 61131-3 programming language [7] that runs on Programmable Logic Controllers (PLCs). The paper makes the following contributions:

- Empirical evidence showing that pairwise testing achieves marginally lower levels of code coverage while in the same time using more tests cases on average than manual testing performed by industrial engineers.
- Results showing that manual testing is not significantly better at finding faults than pairwise testing. Our paper suggests that pairwise testing is just as good in fault detection as manual testing for 64% of the programs considered.
- A discussion of the implications of these results for test engineers and researchers.

II. Background

This paper describes a case study evaluating pairwise testing when used on PLC industrial programs implemented in the IEC 61131-3 FBD language. In this section, we provide a background on PLC industrial software and pairwise testing. According to Ammann and Offutt [1], a test case is a set of inputs, expected outputs and actual outputs executed on the specified program. A test suite is a set of ordered test cases. Throughout the paper, we will use the terms test case and test suite in this way.

A. Programmable Logic Controllers

A Programmable Logic Controller (PLC) is a computer system containing a processor, a memory, and a communication bus. PLCs [8] have a programmable memory for storing the software used for expressing logical behaviour, timing and

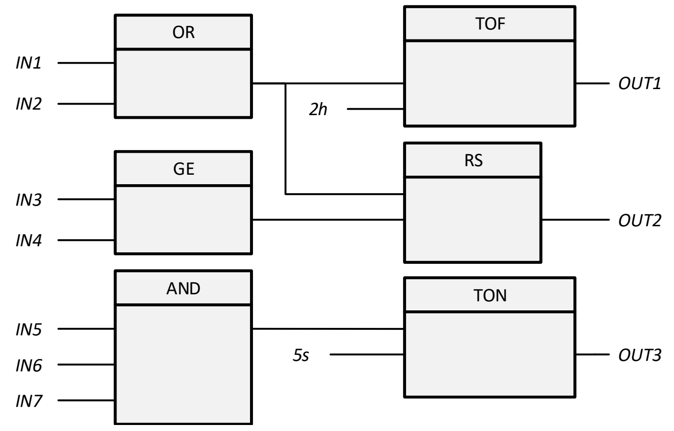

Fig. 1. A PLC program with seven inputs and three outputs written using the FBD IEC 61131-3 programming language.

input/output control, networking and data processing. Safety-critical industrial systems implemented using PLCs are used in many applications [9] such as transportation, robotics, nuclear and pharmaceutical. Software running on a PLC execute in a loop called scan cycle, in which the iteration follows the "read-execute-write" semantics. The PLC reads the input signals, computes the logical behaviour without interruption and updates its output signals [10].

Programming a PLC differs from general-purpose computers; the PLC software follows a standardized programming paradigm: the IEC (International Electronical Commission) 61131-3 standard [7]. IEC 61131-3 is a popular programming language standard for PLCs used in industrial practice. As shown in Figure 1, computational blocks in an IEC 61131-3 program can be represented in a Function Block Diagram (FBD). This program contains predefined logical and/or stateful blocks (i.e., OR, RS, TOF, GE, AND and TON in Figure 1) and signals (i.e., connections) between blocks representing the whole behavior of an FBD program. PLC software contains a particular type of blocks named timers that are used to activate or deactivate an output signal after a specific time interval [8]. A timer block (e.g., TON and TOF in Figure 1) keeps track of the number of times its input is either true or false and outputs different signals. The IEC 61131-3 standard contains four other programming languages: Instruction List (IL), Structured Text (ST), Ladder Diagram (LD) and Sequential Function Chart (SFC) [11]. For more details on PLC programming and FBDs we refer the reader to the work of John et al. [12].

B. Pairwise Testing

The process of test generation is that of finding suitable test inputs using a certain test goal that guides the search in an algorithmic way [1]. Many algorithms and techniques [13] for test generation have been proposed. One such technique is combinatorial testing which is used to reveal faults caused by interactions between input parameters inside a software program. Such techniques design test cases by combining different input parameters based on a combinatorial strategy. Grindal et al. [14] surveyed several strategies used for combinatorial testing (e.g., each-used, pair-wise, t-wise, base choice). One of the most commonly used strategy is pairwise (also known as two-way) testing in which each combination for all possible pairs of input parameters are covered by at least one test case. Several empirical studies [5], [6], [14], [15] on the use of combinatorial testing for industrial software have been reported and showed that pairwise testing is a very effective technique. In this paper we seek to investigate the use of pairwise testing for industrial control software and compare this technique with manual testing performed by industrial engineers.

III. Related Work

Most studies concerning pairwise testing and related to the work included in this paper have focused on how to generate tests as quickly as possible, measure the code coverage score and/or compare with other combinatorial criteria [14] or with random tests [16], [17]. For example, Cohen et al. [4] found that pairwise generated test suites can achieve 90% block code coverage. These test suites where generated by the AETG tool. The same tool was used by Burr and Young [18] in a different study. In this paper, pairwise testing achieved 93% block coverage on average. In addition, Vilkomir and Anderson [19] showed that pairwise test suites could achieve 77% MC/DC code coverage.

Other studies [20]–[22] have reported the use of pairwise testing on real systems and how it can help in the detection of additional bugs when compared to standard test techniques. On the other hand, a few other studies compared manual with pairwise testing [23], [24] and the results suggest that pairwise testing is not able to detect more faults than manually created tests. These results encouraged our interest in investigating on a larger case study how manual testing performed by industrial engineers compares to pairwise testing for industrial software systems. Is there any compelling evidence on how pairwise test suites compare with rigorously handcrafted test suites in terms of test effectiveness?

IV. Method

The goal of this paper is to study the comparison between manual test suites created by industrial engineers and automatically generated test suites using a generation tool for pairwise testing in terms of efficiency and effectiveness of testing. To achieve this goal, we designed a case study (mirrored in Figure 2) using industrial software programs from an already developed train control management system to answer the following research questions:

- *RQ1: Are pairwise generated test suites able to cover more code than test suites manually created by industrial engineers?*
- *RQ2: Are pairwise generated test suites able to detect more faults than test suites manually created by industrial engineers?*
- *RQ3: Is the size of pairwise generated test suites smaller than those manually created by industrial engineers?*

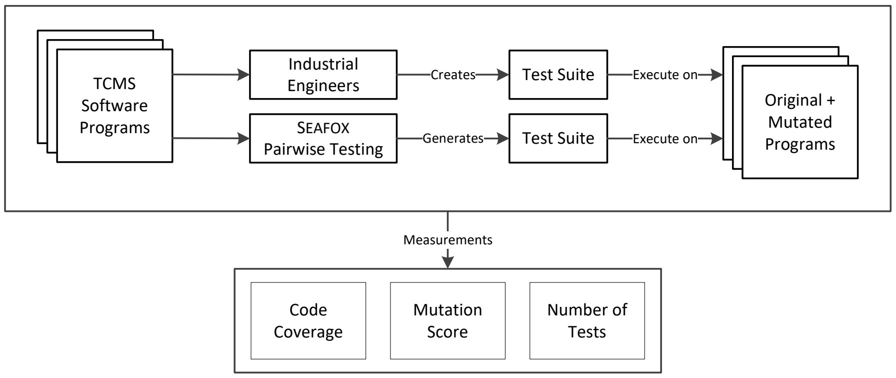

Fig. 2. Overview of the experimental method used to perform the case study. For each program in the Train Control Management System (TCMS), test suites are created manually by industrial engineers, generated by SEAFOX for pairwise testing, and executed on both the original and the mutated programs in order to collect scores for code coverage, mutation and number of tests.

For each selected program, we executed the test suites produced by both manual testing and pairwise testing and collected the following measures: branch coverage in terms of achieved code coverage, the number of generated test cases and the mutation score as a proxy for fault detection. In order to calculate the mutation score, each test suite was executed on the mutated versions of the original program to determine whether it detects the injected fault. This section describes the design of our case study, including the subject programs, the evaluation metrics and the test generation and selection.

A. Subject Programs

Our case study uses an industrial safety-critical system developed by Bombardier Transportation Sweden AB, a large-scale company developing and manufacturing railway equipment. The system is a train control management software (TCMS) that has been in development for several years and is tested according to safety standards and regulations. TCMS is a control system containing both software and hardware components in charge of the safety-related functionality of the train and is used by Bombardier Transportation Sweden AB for the control and communication functions in high speed trains. These functions are developed as software programs for PLCs using the Function Block Diagram (FBD) IEC 61131-3 graphical programming language [7]. Programs in TCMS are developed in a graphical development environment, compiled into PLC code and saved in standardized PLCOpen XML [1] containing structural and behavioural declarations.

We selected the subject programs for our case study by investigating the TCMS programs provided by Bombardier Transportation Sweden AB. We identified 53 programs and excluded eight programs due to the following reasons: one program contained only one input parameter, for another program the test generation got a memory exception, while for the six remaining programs the test execution failed due to wrong parameter ranges that resulted in an execution exception. Our final set of subjects contains 45 programs. These programs contain nine input parameters and 1076 Lines of XML Source Code (LOC) on average per program. The studied programs were already thoroughly manually tested and are currently used in an operational train.

B. Test Case Creation

We used manual test suites created by industrial engineers working at Bombardier Transportation Sweden AB. These manual test suites were obtained by using a post-mortem analysis of the data provided. In testing these programs, engineers perform testing according to specific safety standards and regulations. Specification-based testing is used by engineers to manually create test suites as this is mandated by the EN50128 standard [25]. The test suites collected in this study were based on functional requirement specifications written in a natural language.

In addition, we generate pairwise test suites using SEAFOX [26]. SEAFOX is the only available combinatorial test suite generation tool for IEC 61131-3 control software. SEAFOX is open source software and is available at https://github.com/CharByte/SEAFOX [2].

SEAFOX supports the generation of test suites using pairwise, base choice and random strategies. For pairwise generation, SEAFOX uses the IPOG algorithm as well as a first pick tie breaker [28]. SEAFOX was used in this study as it supports as input a standard PLCOpen XML implementation of the programs. A developer using SEAFOX can automatically generate test suites needed for a given IEC 61131-3 program after manually providing the input parameter range information based on the defined behaviour written in the specification.

In order to collect realistic data, we asked one test engineer from Bombardier Transportation, responsible for testing IEC 61131-3 software used in this study, to identify the range values for each input parameters and constraints. We used these predetermined input parameter ranges for each program variable for generating pairwise test suites using SEAFOX in order to maintain the same input model as the one used to create manual test suites.

[1]http://www.plcopen.org/

[2]For more details on the SEAFOX tool we refer the reader to the work of Charbachi and Eklund [27].

C. Evaluation Measurements

In this section, we present how the case study is conducted with respect to each research question. We first discuss the evaluation measurements used for efficiency and effectiveness of testing.

Code Coverage: We use code coverage criteria to assess the test suites thoroughness [1] and answer RQ1. These coverage criteria are used to evaluate the extent to which the program has been exercised by a certain test suite. In this study, code coverage is directly measured using the branch coverage criterion. For the programs selected in this study the EN50128 safety standard [25] involves achieving high branch coverage. A test suite achieves 100% branch coverage if executing the program causes each branch in the IEC 61131-3 program to have the value *true* and *false* at least once. A branch coverage score was obtained for each generated test suite using our own tool implementation based on the PLC execution framework provided by Bombardier Transportation Sweden AB.

Fault Detection: Ideally, in order to measure fault detection, real faulty versions of the programs are required. In our case, the data provided did not contain any information about what faults occurred during the testing of these programs. To overcome this issue and answer RQ2, we used mutation analysis by generating faulty versions of the original programs. Mutation analysis is a method of automatically creating artificial faulty versions of a program in order to examine the fault detection ability of a test suite [1]. A *mutant* is a different version of the original program containing a small syntactical change. For example, in an IEC 61131-3 program, a mutant is created by replacing a constant value with another one, negating a signal or changing the type of a computational block. If the execution of the resulting mutant on a test is producing a different output as the execution of the original program, the test suite *detects* the mutant. The mutation score is computed using an output-only verdict (i.e., using the expected values for all of the program outputs) against the set of mutants. The fault detection capability of each test suite was calculated as the ratio of mutants detected to the total number of mutants. Just et al. [29] provided compelling experimental evidence that the mutation score is a proxy for real fault detection.

In the creation of mutants we used common type of faults in IEC 61131-3 software [30] as a basis for establishing the following mutation operators:

- *Logic Block Replacement*. Replacing a logical block with another block from the same category (e.g., an OR block is replaced by an AND block).
- *Comparison Block Replacement*. Replacing a comparison block with another block from the same category (e.g., a Greater-Or-Equal (GE) block is replaced by a Greater-Than (GT) block).
- *Arithmetic Block Replacement*. Replacing an arithmetic block with another block from the same category (e.g., replacing a maximum calculation block (MAX) with a minimum calculation block (MIN)).
- *Negation Insertion*. Negating an input or output connection (e.g., an output boolean connection is negated).
- *Value Replacement*. Replacing a value of a constant variable connected to a block (e.g., a constant variable is replaced by its boundary values).
- *Timer Block Replacement*. Replacing a timer block with another block from the same function category (e.g., a Timer-On (TON) block is replaced by a Timer-Off (TOF) block).

Each of the mutation operators were applied to each program element. In total, for all of the selected programs, 1597 mutants were generated (i.e., 35 mutants on average per program). A mutant was considered detected by a test suite if the output from the mutated program differed from that of the original program. A mutation score was obtained for each generated test using our own tool implementation.

Number of Tests: In an ideal situation, the cost of testing is measured by taking into account direct and indirect type of cost, by measuring directly the test suite creation, the test suite execution and the checking of test suite results. However, since this is a case study using programs for which the development was performed a few years back, this kind of cost data was not available. To answer RQ3, we used the number of created test cases as a proxy for efficiency as we assume that all human costs are depended on the number of tests. The higher the number of tests, the higher are the respective testing costs. For example, a complex program will require more effort for test creation, execution and checking of the results.

V. Experimental Results

In this section, we quantitatively answer the three research questions posed in Section IV. We collected the data to answer these research questions by generating test suites for pairwise testing using SEAFOX; collecting manual test suites created by experienced industrial engineers; measuring their code coverage; and measuring their effectiveness in terms of mutation score. The overall results of this study are summarized in the form of boxplots in Figure 3. In Tables I, II and III we present the code coverage scores, mutation scores and the number of tests in each test suite by listing the standard deviation, mean, median, minimum and maximum values. In addition, statistical analysis was performed using the R statistical software [3]. We assume that the collected data is drawn from an unknown distribution. In order to evaluate if there is any statistical difference between manual and pairwise testing we use a Wilcoxon-Man-Whitney U-test [31], a non-parametric hypothesis test used for checking if two data samples are randomly obtained from identical populations. We also use the Vargha-Delaney test (also known as the standardized effect size) to calculate the statistical significance. The Vargha-Delaney $\hat{A}$-measure is also "*a measure of stochastic superiority*" [32] and is used to measure the difference between two populations. The test result is denoted as $\hat{A}$, and simply specifies the amount of times population A is expected to be better than population

[3]https://www.r-project.org/

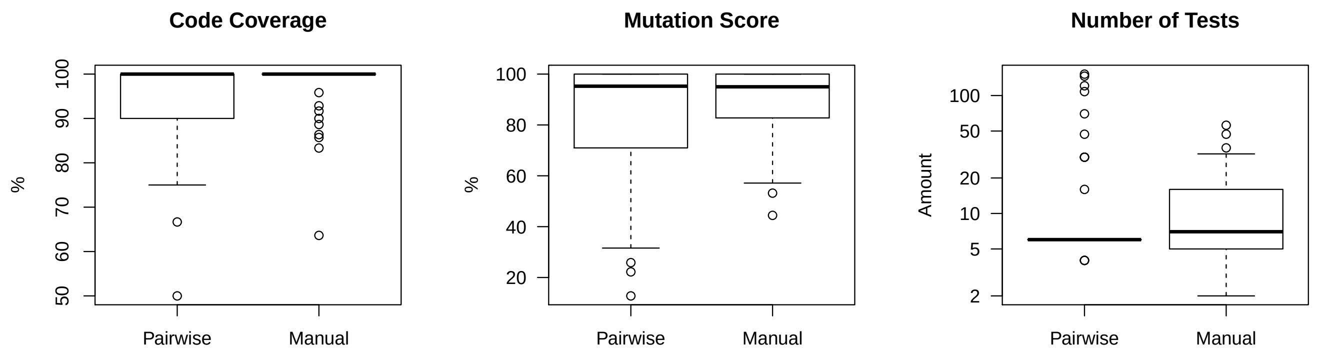

Fig. 3. Code Coverage, mutation score and number of test cases comparison between manually handcrafted test suites (Manual) and test suites generated using pairwise testing (Pairwise); boxes span from 1st to 3rd quartile with black middle lines marking the median and the whiskers extending up to 1.5x the inter-quartile range; the circle symbols represent the outliers.

TABLE I
RESULTS FOR CODE COVERAGE MEASUREMENTS.

Code Coverage	SD	Mean	Median	MIN	MAX
Pairwise	10.35	93.95	100.00	50.00	100.00
Manual	6.71	97.29	100.00	63.64	100.00

TABLE II
RESULTS FOR FAULT DETECTION (MUTATION ANALYSIS) MEASUREMENTS.

Mutation Score	SD	Mean	Median	MIN	MAX
Pairwise	25.69	81.58	95.24	12.77	100.00
Manual	14.22	88.90	95.00	44.44	100.00

B [33]. Its significance is determined when the effect size is above 0.7 or below 0.2.

A. Code Coverage

RQ1 asked if pairwise testing achieves better code coverage scores than manual testing. The coverage scores achieved by pairwise testing are ranging between 50% and 100% while for manual testing these are varying between 63% and 100%. As shown in Table I, the use of manual testing achieves on average 97% branch coverage (3% on average higher than pairwise testing). Results for all programs (in Table IV) show that differences in code coverage between manual and pairwise testing are statistically significant with a p-value of 0.04 but their effect is not strong (i.e., an effect size of 0.6).

As seen in Figure 4, for 62% of the programs considered, pairwise performs equally good as manual testing; for 29% of the programs manual testing performed better in terms of achieved code coverage while for 9% of the programs pairwise testing covers more code than manual testing.

The results for all programs were surprising: test suites created using pairwise testing achieved relatively high code coverage (94% on average). This shows that, for the programs studied in this experiment, pairwise testing achieves high branch coverage. This is likely due to the complexity of the studied programs. It is possible that more complex software would yield a greater code coverage difference between manual and pairwise test suites.

Overall, as shown in Figure 4, we confirm that pairwise test suites achieve just as good or better code coverage scores as manual testing for 71% of programs considered in this study. This can be explained by the fact that pairwise testing if properly used is quite good at covering the logical behaviour of the code.

Answer RQ1: Code coverage scores achieved by pairwise test suites are slightly lower than the ones created manually by industrial engineers.

B. Fault Detection

To answer RQ2, we first computed the mutation score of each manual and pairwise test suites. Figure 3 shows box plots of our results for fault detection in terms of mutation score. Table II summarizes statistics for these test suites. For all programs the fault detection scores obtained by manually written test suites are higher on average with 7% than those achieved by pairwise testing. However, there is no statistically significant difference at 0.05; as the p-value is 0.67 and the effect size is 0.53 in Table IV. A larger sample size would be needed to obtain more confidence in our results. Interestingly, as show in Figure 4, our results suggest that fault detection scores achieved by manual testing are not significantly better at finding faults than pairwise testing. It seems that test suites generated using pairwise testing are just as good in terms of fault detection as manual test suites for 64% of the cases considered in this study. For 42% of the programs, pairwise testing performs as well as manual testing while for 36% of the programs manual testing performed better in terms of fault detection.

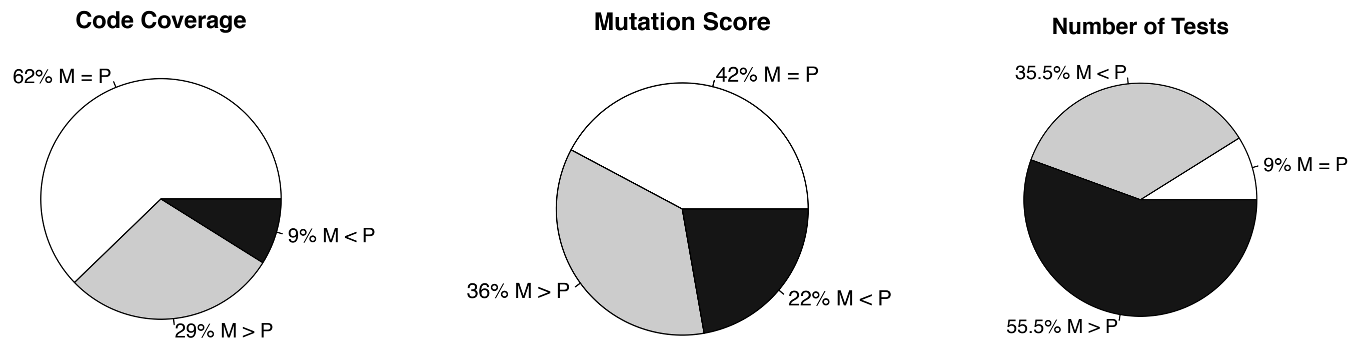

Fig. 4. Charts of number of tests, code coverage and mutation scores depicting how each technique (M stands for manual testing and P is short for pairwise testing) compared to the other for each measure. M>P indicates that manual testing achieved a higher measurement score, M<P shows how many times manual testing achieved a lower measurement score and M=P shows when both techniques achieve the same measurement score.

TABLE III
RESULTS FOR CODE NUMBER OF TESTS (TEST SUITE SIZE) MEASUREMENTS.

Number of Tests	SD	Mean	Median	MIN	MAX
Pairwise	37.05	21.20	6.00	4.00	152.00
Manual	12.26	12.98	7.00	2.00	56.00

The difference in effectiveness between manual and pairwise could be due to other factors such as the number of test cases and the test design techniques used to manually create test suites (e.g., testing the timed behavior of the PLC software).

Answer RQ2: Pairwise testing is able to produce comparable results to manual testing in terms of fault detection. However, manual testing produced better mutation scores on average.

C. Number of Tests

This section aims to answer RQ3 regarding the relative cost of performing manual testing versus pairwise testing. Analysing the cost in this study is directly related to the number of test cases giving a picture of the effort needed per created test suite. Based on the results highlighted in Figure 3, the use of pairwise testing results in very inconsistent number of tests created, compared to manual testing which seems to create tests with more diverse number of steps than pairwise testing. Examining Table III, we see a different pattern: less number of tests on average are manually created by industrial engineers (13 test cases on average in a test suite) than when using pairwise testing (21 test cases on average in a test suite). As seen in Figure 4, for 44% of the programs considered, pairwise test suites produced equally or larger test suites than the manual ones, leading to 55% where pairwise produced fewer. Table IV shows an interesting pattern in the statistical analysis: the standardized effect size being 0.53, with p-value being higher than the traditional statistical significance limit of 0.05. Results are not strong in terms of effect size and we did not obtain any statistical difference for the number of tests.

TABLE IV
RESULTS OF THE STATISTICAL ANALYSIS. FOR EACH METRIC WE CALCULATED THE EFFECT SIZE OF EACH TEST CREATION METHOD COMPARED TO EACH OTHER. WE ALSO REPORT THE P-VALUES OF A WILCOXON-MANN-WHITNEY U-TESTS.

Statistics	Values	Code Coverage	Mutation Score	Test Suite Size
M vs P	$\hat{A}$	0.60	0.53	0.53
	$p - value$	0.04	0.67	0.64

The results for all programs matched our expectations: manual tests are handcrafted by experienced industrial engineers that can create very diverse tests. It is possible that more complex software would yield greater number of tests differences between tests written manually and pairwise testing.

Answer RQ3: The use of pairwise testing results in more number of tests created on average than the use of manual testing. Even so, pairwise testing produced more cases with less tests created compared to manual testing, resulting in very inconsistent results.

VI. DISCUSSION

The goal of this work was to compare pairwise testing with manual testing performed by industrial engineers in terms of code coverage, fault detection and the number of created test cases. We found out that pairwise testing achieves high code coverage, but slightly lower scores than manual testing. In addition, we found the fault detection scores for pairwise testing to be lower on average than the ones written manually by industrial engineers. Interestingly, pairwise testing achieves equally good or better fault detection scores than manual testing for 64% of the programs considered, which might indicate that for more than half of the programs the fault detection scores for pairwise testing are a good predictor of test effectiveness. This is reinforced by the mutation score box plot in Figure 3: the median mutation score for pairwise testing is 95%. This in combination with the achieved high code coverage, suggests that pairwise testing can cover as much code as manual testing performed by industrial engineers, but this is not entirely reflected in its fault detection capability. The fault detection rate between manual and pairwise testing was found, in some of the published studies [23], [24] to be similar to our results. Interestingly enough, our results indicate that pairwise test suites might be even better in some cases in terms of fault detection than manual test suites. However, a larger pool of programs and tests is needed to statistically confirm this hypothesis.

The mean value for fault detection of 81% for pairwise testing is right in line with the proportion of 2-way faults seen in other domains [34]. It is interesting to note here that the fault distribution for PLC industrial control software is similar to other types of software.

As part of our study, we used the number of tests to estimate the test efficiency in terms of creation, execution and result checking. While the cost of creating and executing a test for pairwise testing can be low compared to manual testing, the cost of checking the result is usually human intensive. Practically, the higher the number of test cases, the higher the cost of checking the test result. Our study suggests that pairwise test suites, while inconsistent, are longer on average in terms of created test steps (number of tests) than manual test suites. By considering generating optimized or shorter test suites, one could improve the cost of performing pairwise testing.

The idea of using pairwise testing in practice stands as a significant progress in the development of automatic test generation approaches. This progress implies, to some extent, that pairwise testing should be at least as effective and more efficient than manual testing for it to be considered ready to be used as a replacement to the manual effort of creating tests. The overall result, from this case study, is that pairwise testing alone is not better than manual testing. However, pairwise testing or stronger combinatorial criteria are capable of at least aiding an engineer in testing of industrial software. Our observations showed that experienced engineers are very effective at generating the right choice of values and considering the timing of the input parameters. Industrial PLC software typically have a complex and time-dependent behaviour. This behaviour require inputs to retain and change a sequence of inputs for some time in order to trigger a certain logical event. Several manual test suites collected in this case study contained that kind of test cases. The engineers creating these test suites had years of experience in developing and testing this type of software, including good knowledge of what combinatorial interactions are needed to cover the code and detect faults. As it turns out, pairwise testing is not particularly useful for some of the programs considered in this study. By considering generating more complex (stronger t-wise) and time-depended tests, one could improve both the achieved code coverage as well as the fault detection capability.

VII. Threats to Validity

There are many tools (e.g., ACTS [35], AETG [36], TCG [37]) for automatically generating tests using pairwise testing and these may produce different results. The use of these tools in this study is complicated by the modelling of the input space for a PLC program. Hence, we choose a tool specifically developed for testing PLC industrial programs. For more details on the comparison between SEAFOX and ACTS, we refer the reader to the work of Charbachi and Eklund [27]. SEAFOX is using the IPOG algorithm. This algorithm continually expands the test suite to fit the input parameters with the use of vertical and horizontal extension. SEAFOX currently uses a first element tie breaker which was chosen when a clean-cut best choice for tie breaking was absent [38]. This choice of a tie breaker might affect the effectiveness of the generated tests.

Another threat to the validity of this study is also related to the use of the SEAFOX tool. Vertical extension [39] is a step which adds new tests, if needed, to the test set produced by horizontal growth when there are no modifiable test cases to cover a specific pair. The IPOG algorithm, in this case, creates a new test case, thus expanding the size of the test suite by making the test case to cover the specific pair and keeping all the other parameters modifiable. This is done to reduce the need of further vertical extensions and can be used to avoid creating a new test case. However, when trying to execute a test suite, these values are syntactically illegal and need to be changed. SEAFOX currently handles this by randomising the modifiable values to a default option for each parameter. A more accurate option for each parameter would be needed to obtain more confidence in the test suite effectiveness.

The data collected is based on a study in one company using one industrial system containing 45 programs and manual test suites created by industrial engineers. We argue that even if the number of programs can be considered relatively small, reporting a case study using industrial artefacts can be representative.

Since this case study was performed post-mortem, the cost information was not available. We used the number of test cases as a valid proxy measure and a more detailed cost model should be used to obtain more accurate results.

VIII. Conclusion

In this paper, we studied the comparison between pairwise testing and manual testing in terms of branch coverage, mutation score and number of created test cases. From the 45 PLC industrial programs we studied, we drew the following conclusions:

- The use of pairwise testing results in high branch coverage and mutation scores for the majority of the programs considered.
- The results of this paper support the claim that pairwise testing is not quite as effective (i.e., achieved branch coverage and fault detection) and efficient (in terms of number of tests created) as manual testing.

- The use of pairwise testing results in similar or better fault detection scores than the use of manual testing for 64% of the programs.

The results imply that pairwise testing can achieve high branch coverage, but slightly lower scores than manual testing. In summary, our results suggests that pairwise testing can perform in some cases comparably with manual testing performed by industrial engineers. This is a significant experimental evidence on the progress of pairwise testing that needs to be further studied; we need to consider the cost of using pairwise testing in practice. In addition, pairwise testing is only one type of combination strategy and we would need to evaluate the use of stronger criteria such as t-wise (with $t > 2$) testing.

Acknowledgment

This research was supported by the Swedish Research Council (VR) through the "Adequacy-based testing of extra-functional properties of embedded systems" project and by VINNOVA and ECSEL (EU's Horizon 2020) under grant agreement No 737494. The authors would like to thank Bombardier Transportation AB for the valuable assistance in the planning and execution of this work.

References

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] C. Andersson and P. Runeson, "Verification and validation in industry-a qualitative survey on the state of practice," in *International Symposium on Empirical Software Engineering*. IEEE, 2002, pp. 37–47.

[3] A. Beer and R. Ramler, "The role of experience in software testing practice," in *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2008, pp. 258–265.

[4] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE software*, vol. 13, no. 5, p. 83, 1996.

[5] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of acts: A case study," in *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2012, pp. 591–600.

[6] J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker, "Introducing combinatorial testing in a large organization," *Computer*, vol. 48, no. 4, pp. 64–72, 2015.

[7] IEC, "International Standard on 61131-3 Programming Languages," in *Programmable Controllers*. IEC Library, 2014.

[8] R. W. Lewis, *Programming industrial control systems using IEC 1131-3*. IET, 1998, no. 50.

[9] D. L. Parnas, A. J. van Schouwen, and S. P. Kwan, "Evaluation of safety-critical software," *Communications of the ACM*, vol. 33, no. 6, pp. 636–648, 1990.

[10] J. Donandt, "Improving response time of programmable logic controllers by use of a boolean coprocessor," in *VLSI and Microelectronic Applications in Intelligent Peripherals and their Interconnection Networks*. IEEE, 1989, pp. 4–167.

[11] W. Bolton, *Programmable logic controllers*. Newnes Books, 2015.

[12] K.-H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. Springer, 2010.

[13] A. Orso and G. Rothermel, "Software Testing: a Research Travelogue (2000–2014)," *Proceedings of the International conference on Software Engineering (ICSE), Future of Software Engineering*, pp. 117–132, 2014.

[14] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler, "An evaluation of combination strategies for test case selection," *Empirical Software Engineering*, vol. 11, no. 4, pp. 583–611, 2006.

[15] X. Li, R. Gao, W. E. Wong, C. Yang, and D. Li, "Applying combinatorial testing in industrial settings," in *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016, pp. 53–60.

[16] L. S. Ghandehari, J. Czerwonka, Y. Lei, S. Shafiee, R. Kacker, and R. Kuhn, "An empirical comparison of combinatorial and random testing," in *International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2014, pp. 68–77.

[17] P. J. Schroeder, P. Bolaki, and V. Gopu, "Comparing the fault detection effectiveness of n-way and random test suites," in *International Symposium on Empirical Software Engineering*. IEEE, 2004, pp. 49–59.

[18] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation and code coverage," in *International Conference on Software Testing, Analysis & Review*. IEEE, 1998.

[19] S. Vilkomir and D. Anderson, "Relationship between pair-wise and mc/dc testing: initial experimental results," IEEE, pp. 1–4, 2015.

[20] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton, "The automatic efficient test generator (aetg) system," in *International Symposium on Software Reliability Engineering*. IEEE, 1994, pp. 303–309.

[21] S. R. Dalal, A. Jain, N. Karunanithi, J. Leaton, and C. M. Lott, "Model-based testing of a highly programmable system," in *International Symposium on Software Reliability Engineering*. IEEE, 1998, pp. 174–179.

[22] S. Sampath and R. C. Bryce, "Improving the Effectiveness of Test Suite Reduction for User-Session-based Testing of Web Applications," *Information and Software Technology*, vol. 54, no. 7, pp. 724–738, 2012.

[23] M. Ellims, D. Ince, and M. Petre, "The effectiveness of t-way test data generation," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2008, pp. 16–29.

[24] M. Ellims, D. Ince, M. Petre, W. Hall, and M. Ellims, "Aetg vs. man: an assessment of the effectiveness of combinatorial test data generation," *Technical Report, Department of Computer Science, Open University*, vol. 8, p. 2007, 2007.

[25] CENELEC, "50128: Railway Application: Communications, Signaling and Processing Systems, Software For Railway Control and Protection Systems," in *Standard Official Document*. European Committee for Electrotechnical Standardization, 2001.

[26] Peter, The-Luxs, D. of Springfield, and jfeljan, "Charbyte/seafox: Seafox," Mar. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.439253

[27] P. Charbachi and L. Eklund, "Pairwise testing for plc embedded software," in *Thesis for the Degree of Bachelor of Science in Computer Science*. Mälardalen University, 2016.

[28] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog/ipog-d: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.

[29] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are Mutants a Valid Substitute for Real Faults in Software Testing," in *International Symposium on the Foundations of Software Engineering*. ACM, 2014.

[30] Y. Oh, J. Yoo, S. Cha, and H. Seong Son, "Software Safety Analysis of Function Block Diagrams using Fault Trees," in *Reliability Engineering & System Safety*. Elsevier, 2005, vol. 88.

[31] D. C. Howell, *Statistical methods for psychology*. Cengage Learning, 2012.

[32] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[33] G. Neumann, M. Harman, and S. Poulding, "Transformed vargha-delaney effect size," in *International Symposium on Search Based Software Engineering*. Springer, 2015, pp. 318–324.

[34] D. Kuhn, R. Kacker, and Y. Lei, "Combinatorial testing," *Encyclopedia of Software Engineering*, 2012.

[35] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Acts: A combinatorial test generation tool," in *International Conference onSoftware Testing, Verification and Validation (ICST)*. IEEE, 2013, pp. 370–375.

[36] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The aetg system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.

[37] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Aerospace Conference*, vol. 1. IEEE, 2000, pp. 431–437.

[38] R. Huang, J. Chen, R. Wang, and D. Chen, "How to do tie-breaking in prioritization of interaction test suites?." in *SEKE*, 2014, pp. 121–125.

[39] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog: A general strategy for t-way software testing," in *International Conference and Workshops on the Engineering of Computer-Based Systems*. IEEE, 2007, pp. 549–556.