On the Correlation between Testing Effort and Software Complexity Metrics

Adnan Muslija, Eduard Enoiu,
Mälardalen University, Västerås, Sweden.

***Abstract*—Software complexity metrics, such as code size and cyclomatic complexity, have been used in the software engineering community for predicting quality metrics such as maintainability, bug proneness and robustness. However, not many studies have addressed the relationship between complexity metrics and software testing and there is little experimental evidence to support the use of these code metrics in the estimation of test effort. We have investigated and evaluated the relationship between test effort (i.e, number of test cases and test execution time) and software complexity metrics for industrial control software used in an embedded system. We show how to measure different software complexity metrics such as number of elements, cyclomatic complexity, and information flow for a popular programming language named FBD used in the safety critical domain. In addition, we use test data and test suites created by experienced test engineers working at Bombardier Transportation Sweden AB to evaluate the correlation between several complexity measures and the testing effort. We found that there is a moderate correlation between software complexity metrics and test effort. In addition, the results show that the software size (i.e., number of elements in the FBD program) provides the highest correlation level with the number of test cases created and test execution time. Our results suggest that software size and structure metrics, while useful for identifying parts of the system that are more complicated, should not be solely used for identifying parts of the system for which test engineers might need to create more test cases. A potential explanation of this result concerns the nature of testing, since other attributes such as the level of thorough testing required and the size of the specifications can influence the creation of test cases. In addition, we used a linear regression model to estimate the test effort using the software complexity measurement results.**

I. Introduction

Testing is a vital part of engineering industrial software. Its efficiency and effectiveness depends not only on the quality of the test suite, but also on the complexity of the software to be tested: some programs need more test cases than others. In practice, test managers and engineers want to know what parts of their software are more complex and need more testing effort. Software engineering studies and textbooks often used software complexity metrics to predict quality metrics such as faults proneness and maintainability (e.g., [1]–[3]). However, not many studies have looked at the relationship between code complexity and test effort. It is therefore not clear that software complexity is directly related to test efficiency.

This paper presents a study of the relationship between software complexity metrics and testing effort in an industrial embedded software project from Bombardier Transportation Sweden AB. In practice, we answer the following research questions:

- ***Research Question 1****: Is software complexity correlated with the number of test cases in a test suite?*
- ***Research Question 2****: Is software complexity correlated with the execution time of all test cases in a test suite?*

The paper makes the following contributions to this kind of investigations:

- A method and tool for measuring code complexity for embedded software written in IEC61131-3 Function Block Diagram (FBD), a popular programming language in the safety critical domain. There is a need to investigate how existing software complexity metrics can be tailored to FBD software.
- Empirical and industrial evidence showing that there is a moderate correlation between software complexity and the number of test cases created by experienced industrial engineers as well as the execution time of these test cases.
- A discussion of these results as well as the use of a linear regression model to estimate the test effort using software complexity are shown.

II. Background

In this section, we explain the concepts necessary for understanding the methodology used to obtain the results by covering software complexity metrics and industrial control software. The work presented in this paper focuses on the measurement of software complexity for industrial embedded software written in the FBD language.

A. Industrial Control Software

In domain-specific domains (e.g. transportation, nuclear, aerospace and automotive), embedded systems implemented using Programmable Logic Controllers (PLCs) are widely used to provide supervisory control [4]. For example, this supervisory control can be used for opening and closing doors or controlling the temperature in a furnace. PLC software differentiate from its general-purpose counterpart in several ways, including the way that they are written and tested. PLC programs are usually created using one of the IEC 61131-3 programming languages [5]. IEC 61131-3 is a international standard that describes the programming language rules and requirements used for creating PLC programs [5]. IEC 61131-3 has a number of programming language implementations: *Structure Text* (ST), *Instruction List* (IL), *Ladder Diagram*

```
if (a > b) {
    sum = a + c;
} else {
    sum = b + c;
}
```

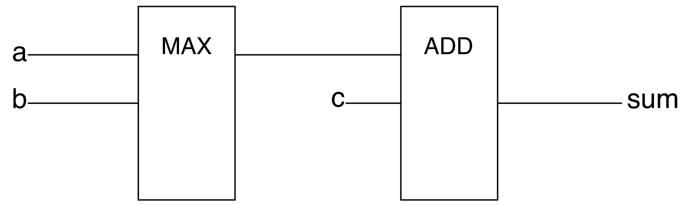


Fig. 1: Two equivalent simplified programs written in two different program languages: a Java example (left) and an FBD program (right).

(LD), *Function Block Diagram* (FBD). Two of these languages, FBD and LD, are graphical programming languages and do not use a textual source code notation. Since the IEC 61131-3 programming languages are used in domain-specific applications, the resulting software is organized and operates using Program Organization Units (POUs) [5] containing functions (i.e., procedure-like program code), function blocks (i.e., stateful functions) and a top-level program code that has access to the IO ports. FBDs contain variables, data types, functions. However, conditional statements and loops are implemented differently in FBDs. As shown in Figure 1, the IF statement is encapsulated in the MAX function. In this study we used programs developed in the IEC 61131-3 standard and FBD programming language by industrial engineers describing a safety-critical system used in the train domain.

B. Software Complexity

A software complexity metric is a quantitative value that describes a certain dimension of the software and depends on the type of the artifact used for measurement [6]. Even if multiple software dimensions can be used, it is not easy to use such measures on multiple software artifacts (e.g., program source code and the software architecture). Nevertheless, there are a number of software complexity metrics that have been successfully used in software engineering domain [7]. Source Lines of Code (SLoC) is a simple size metric that measures the logical and the physical size of a source file. It is a size metric, since it can only describe the size dimension of a software artifact (e.g. the program source code). Weyuker et al. [8] have shown how to formalize, evaluate and compare different complexity metrics including SLoC. Although the motivation for measuring a specific dimension of a software varies in practice [9], several studies [10], [11] have indicated that complexity metrics may be good at predicting quality and development effort. For example, software complexity measurements [12] can be used to indicate the number of test cases needed to cover the logic of a particular artifact or can be used to show that a software architecture has a high levels of coupling [13]. Even if the literature on measuring software complexity for industrial control software written in IEC 61131-3 FBD programming language is scarce, other graphical programming languages have been the focus of research on complexity measurements. Olszewska et al. [14] tailored software complexity metrics to the component-based syntax of Simulink models. This study also performed a correlation analysis between complexity and fault data obtained from a car fuel program created using Simulink and found a positive correlation between components with high complexity and the number of faults found. The data was validated using three domain experts. At the time of writing, no other study considered any correlation evaluation between test effort and software complexity metrics in the industrial control software domain.

III. Methodology

This section shows the experimental roadmap used in this paper including the subject software, how we measured software complexity on such software as well as the measurement of test effort.

A. Subject Programs

The safety-critical industrial control software we used in this paper is part of the Train Control Management System (TCMS). TCMS is a system developed and used by Bombardier Transportation Sweden AB for high speed trains. TCMS is an embedded software running on PLCs and used for handling a wide variety of operation-critical and safety-critical functions of a train. TCMS is written in IEC 61131-3 FBD programming language using a combination of IEC 61131-3 function and function blocks and in-house built function blocks. We used 82 different FBD control programs part of TCMS that perform supervisory operations and are developed independently of each other.

B. Measuring Software Complexity

Since there is no approach and tool support for measuring software complexity on FBD programs, we developed a tool called TIQVA [15] in an effort to create a complexity measurement method for FBD software. Practically, we create an FBD data structure model based on the Abstract Syntax Trees (AST) used for parsing and processing source code. The data model is organized hierarchically and works in several iterations. We use the FBD XDS Schema rules in creating the data model (as shown in Figure 2) that contains all relevant program information from the FBD representation and maintains the hierarchy and relationships between different FBD elements.

A significant number of software complexity metrics have been proposed in the literature [12], [13], [16]–[18]. We chose to adapt and implement the following complexity metrics, since these are among the most popular and well-researched metrics [6] in the software engineering literature: Source

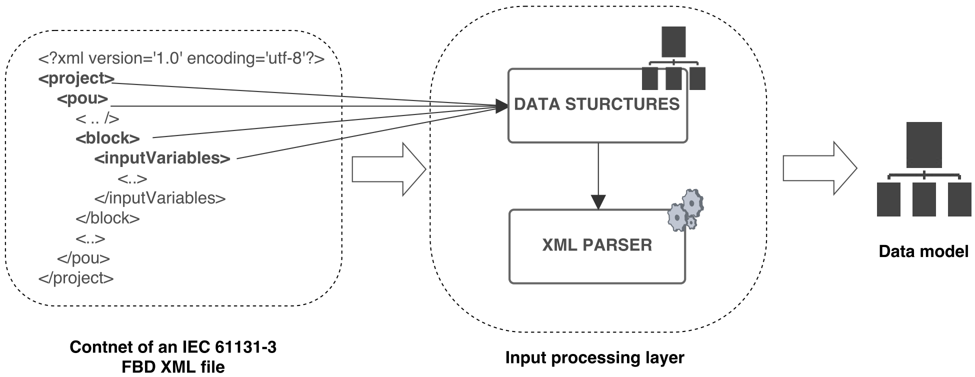


Fig. 2: A high-level view of the input processing layer of TIQVA for parsing the FBD programs.

Lines of Code (SLoC), Cyclomatic complexity (CC), Halstead complexity (HC) and Information Flow complexity (IFC). Both SLoC and HC are code size metrics and can abstract the size or the length of a program artifact. The HC metric is suggested to be good at providing information about software maintenance [16]. CC is a direct measure of the amount of decisions programmed in the software [12] and is used for determining the number of tests achieving basic path coverage [19]. The IFC metric was proposed by Henry and Kafura [13] and is mainly used for measuring the complexity of software architecture designs, since the metric computes the amount of coupling and cohesion between different software modules. We use IFC for FBD programs since some of these architecture models and FBD programs are both using basic component-based modelling concepts [5].

1) Number of Elements: In the IEC 61131-3 FBD programming language the notion of a program statement is very different compared to other general-purpose programming languages. While in Java, a line of code can be a function call, in FBDs functions are encapsulated inside block components. Therefore, we mapped the SLoC metric to FBDs. If the function calls and other program statements are abstracted via blocks, and the order of their execution is controlled via connections, then we assumed that the SLoC metric for FBDs would measure the number of elements including the blocks and connections in an FBD program [5]. We propose the use of *Number of Elements (NOE)* in the context of the FBD programming language by counting the number of all declarations, blocks and connections. When initialized in the graphical programming environment, FBD variables and their data types are represented as component blocks (e.g., input, output and local component blocks).

2) Cyclomatic Complexity: In the original paper [12], Thomas McCabe proposed a software measurement technique for computing the number of linearly independent paths through a program code. This metric is based on graph theory and can be applied to a wide range of software artifacts (from simple program functions to architectures [20]). Artifacts measured for CC have to be abstracted via control flow graphs. Since CC is influenced by the decision points of a program, CC can directly be used for the FBD programming language.

3) Halstead Complexity: HC metric [16] is computing multiple software dimensions based on the measurement of operands and operators. We assume that a set of operators are represented using different mathematical and logical operations and programming language functions and syntax, while the set of operands are variables and values used in the operations. HC metric defines the following measurements: program vocabulary, program length, calculated program length, volume, difficulty, effort, time, and delivered bugs. These measurements are computed based on both the unique and total number of operators and operands with the rest of the measurements being built upon them. In FBDs, program variables and their definition are separated from the logic itself, so operators and operands are created in a different fashion compared to a Java programs. Functions and function blocks are representing operations such as comparison and multiplication. In addition, FBD connections are used for the flow of data through the FBD algorithm using connections and thus we used them to calculate the number of operands in an FBD program.

4) Information Flow Complexity: Henry and Kafura proposed the use of a software complexity metric [13] that could be applied at earlier stages of software development (e.g., during the software architecture modelling). IFC can be used also to measure the information flow between procedures or functions of a single program unit. Although IEC 61131-3 POUs (i.e., programs, functions and function blocks) are used as independent program units, these can be represented as software modules in the overall software architecture of a PLC software system. IFC can be tailored by measuring the number of defined inputs and outputs of an FBD functions or function blocks. This provides a baseline IFC score of an FBD POU, and that value can only increase when the POU is used in other FBD programs. In addition, we compute fan-in using the number of output parameters and fan-out using the number of input parameters. The SLoC value was measured using the NOE metric already defined in this paper.

C. Manual Testing and Test Effort

In this paper we use test cases for the individual TCMS FBD programs, which have been manually created and used for thorough testing performed by experienced industrial engineers. Practically, we used 82 test suites created by industrial engineers in Bombardier Transportation from a TCMS project delivered already to customers. A test case created for an FBD program contains a set of test cases containing inputs, expected and actual outputs and timing information. Data about these test cases was collected by using a post-mortem analysis [21]. In testing FBD programs in TCMS, the testing processes of software assurance are performed according to safety standards and regulations. Requirement-based testing is mandated by the EN 50128 standard [22] to be used to design test cases with each test case contributing to the requirement satisfaction. In addition, testers are required to create test cases based on multiple goals such as their experience, negative test cases as well as coverage-based test cases. Executing test cases on TCMS is supported by a test automation framework. The test cases collected in this study were based on functional requirements expressed in a natural language.

Many factors affect the effort needed to test an FBD program. According to Leung and White [23], testing involves direct and indirect costs. A direct cost includes the time needed for testing activities and the machine resources such as the test infrastructure used. Indirect costs could include the management of the testing process and the test tool development. Ideally, the test effort is captured by measuring the time required for performing all the different testing activities. Since this is a post-mortem study of a deployed TCMS system and the testing process was performed a few years back, we used proxy measures capturing the context that directly affects testing effort.

We note here that the number of test cases depends on the testing strategy used but also on other contextual factors. A test strategy that requires that every branch in the program to be executed generally needs more tests than one which only requires all statements of the program to be executed. In this paper we consider that the test effort is related to the number of test cases and the execution time needed to execute such test cases. The higher the number of tests cases and the test execution time, the higher is the respective test effort. Practically, this is a measure of the test effort of industrial engineers (working at Bombardier Transportation Sweden AB testing the programs used in our study) to perform thorough testing. The intuition is that a complex program will require more effort for testing, and also more tests than a simple program. Thus, the investigated hypothesis is that the test effort is related to the same factor— the complexity of the software which will influence the number of test cases and test execution time.

IV. Results

In this section, we quantitatively answer the two research questions posed in Section I. As Section III explained, we collected the data to answer these questions by computing the complexity measures for the FBD programs considered; and collecting the number of test cases in each test suite as well as the test execution time.

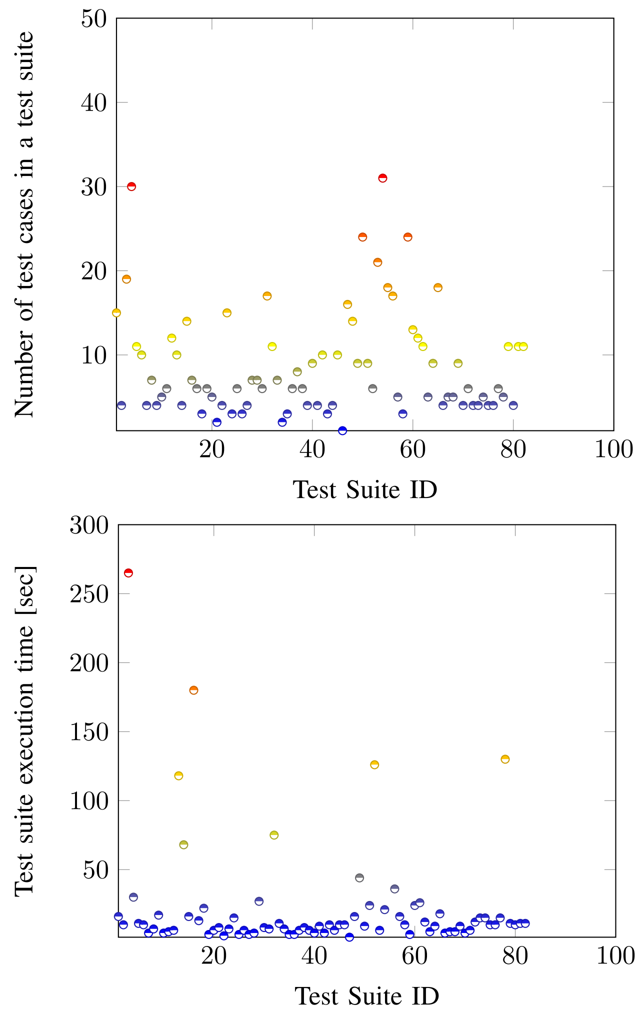


Fig. 3: The number of test cases and the test execution time plotted for each individual test suite created for all programs.

A. Complexity Measurements

Figure 3 shows some of the data collected for all FBD programs, with each data point representing the number of test cases in each test suite as well as the test case execution time. Table I gives the descriptive statistics of the test data. We can observe that the average test execution time is 32 seconds while the test suite with the largest execution time takes 900 seconds. In addition, the average number of test cases in a test suite is 8.5 with the largest test suite containing 31 test cases.

In addition, we measured software complexity of each FBD program using the TIQVA tool. This resulted in a total of 15 measurements for all software complexity measures used in this paper (NoE, CC, HC, IFC). TIQVA consisted of these 15 measurements for the selected 82 programs.

In Table II we show the results of measuring the complexity of all FBD industrial control software. The different software complexity measures cannot be directly compared with each other. However, one program and its IFC score stands out with a high value of IFC complexity. Actually, four FBD programs

Measure of Test Effort	Min	Max	Average	Median	Standard Deviation
Number of test cases	1	31	8.5	6	6.2
Test execution time (sec.)	1	900	32.2	10	105.2

TABLE I: Results for both test effort metrics: the number of test cases in each test suite and the test execution time.

Software complexity measure	Min	Max	Average	Median	Standard Deviation
Variables	4	85	22	18.5	14.7
Connections	3	216	33.8	25.5	32.5
Blocks	5	228	39	29.5	35.1
Number of elements (NoE)	*12*	*483*	*94.8*	*74.5*	*80*
Cyclomatic Complexity (CC)	*1*	*133*	*18.9*	*13*	*21.8*
Information flow complexity (IFC)	*12*	*57065472*	*2690651*	*68506*	*8478907.8*
Unique operators	8	19	11.4	11	2.6
Unique operands	9	262	59.5	47.5	47
Total operators	12	229	59.1	49.5	39.8
Total operands	12	402	86.3	69	71.1
Halstead Program vocabulary	17	278	71	60	48.5
Halstead Program length	24	599	145.5	117	110.1
Halstead Calculated Program Length	52.5	2168.7	413.2	309.1	385.6
Halstead Volume	98.1	4844.3	942.5	691.5	886.8
Halstead Difficulty	5.3	14.1	8.1	7.9	1.8
Halstead Effort	523.2	59747.4	8756.5	5470.9	10925.8
Halstead Time	29	3319.3	486.4	303.9	606.9
Halstead Delivered Bugs	0.02	0.5	0.1	0.1	0.09

TABLE II: Results for all software complexity metrics (i.e., Number of Elements (NoE), Cyclomatic Complexity (CC), Information Flow Complexity (IFC) and Halstead) together with the basic measures used to calculated these metrics.

from TCMS showed high software complexity scores (i.e., Program 9, Program 60, Program 32 and Program 55). In particular, Program 32 achieved high complexity scores for NoE, CC, and Halstead (HC). Program 55 achieved the highest NoE score, Program 60 the highest IFC score and Program 9 the highest Halstead Difficulty score. Upon closer inspection, Program 32 has a very high number of input and outputs parameters as well as elements and can be considered based on all complexity scores the most complex program considered in this paper.

In order to get a better view on the distribution of individual metrics, we normalized the reported values (i.e., with 0 representing the lowest complexity score while 1 showing the highest complexity score. The plots in Figure 4 show the distribution of NoE and CC metrics. We can observe that two outliers (Programs 32 and Program 55) show high NoE and CC scores. The rest of the programs are scattered below the 0.5 threshold score. A similar result can also be seen when considering IFC and Halstead Difficulty metrics. In Figure 5 we observe that IFC scores are very much polarized with quite low scores for most of the programs. In Figure 6 we show an area plot for two Halstead complexity measures (i.e., Halstead Difficulty and Halstead Volume) which are used to construct the rest of the other Halstead metrics (i.e., Effort, Testing Time and Delivered Bugs). Both of the metrics shown in Figure 6 have a similar distribution for all the programs considered in this study.

For all programs we used the following complexity measurements: Number of Elements and Halstead Metric for measuring the size, Cyclomatic Complexity for measuring the structure and Information Flow Metric as an architectural metric.

B. Is software complexity correlated with the number of test cases and test suite execution time?

Research questions 1 and 2 asked if the test effort (i.e, number of test cases and test execution time) is influenced by the software complexity of the programs considered in this study. Table III shows the Kendall correlation coefficients [24] we computed to answer this question. Kendall rank correlation is used as a measure of correlation between software complexity scores and the test effort proxy scores. Since the data is not normally distributed we use Kendall correlation to not introduce unnecessary assumptions about the collected data. We try to determine the possible statistical relationship between software complexity and the test effort scores. We used Kendall's rank correlation coefficient to calculate the statistical relationship between the scores with a significance level of *0.05* since a statistically significant correlation does not necessarily mean that the strength of the correlation is strong. Here we use the Cohen scale [25], in which correlations with absolute value less than 0.3 are described as weak, 0.3 to 0.5 as moderate, 0.5 to 0.9 as strong and very strong.

The two test effort proxy measures required the computation of the correlation coefficients using R [26]. Table III shows τ coefficients and p-values for the two proxy measures (i.e., E stands for test suite execution time and N stands for the number of test cases in a test suite). A positive correlation can be observed for four software complexity metrics (i.e., Halstead is shown as three separate complexity measures:

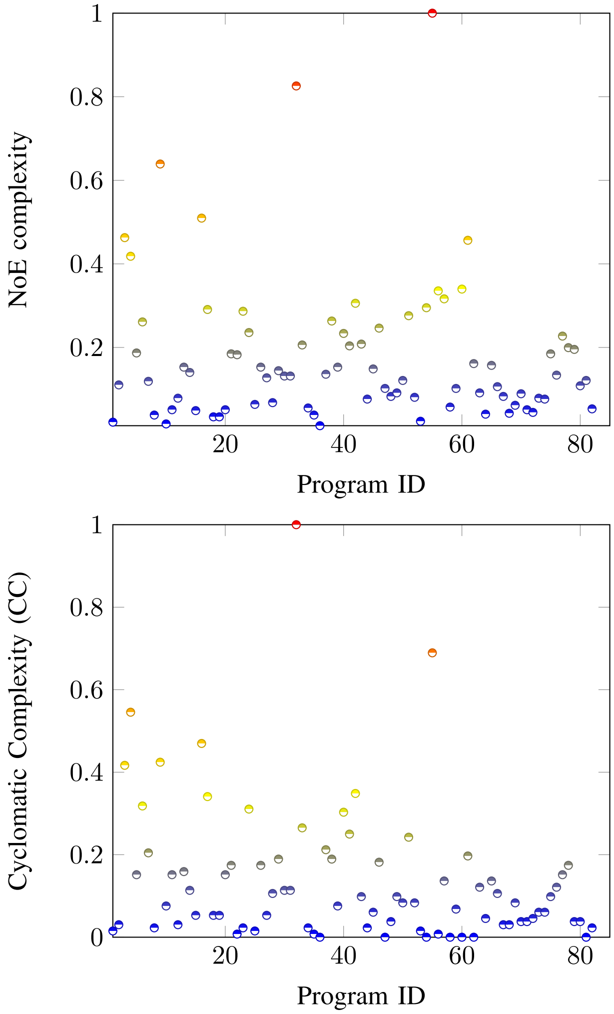


Fig. 4: Normalized scores for the Number of Elements (NoE) complexity and Cyclomatic Complexity (CC) for all considered programs.

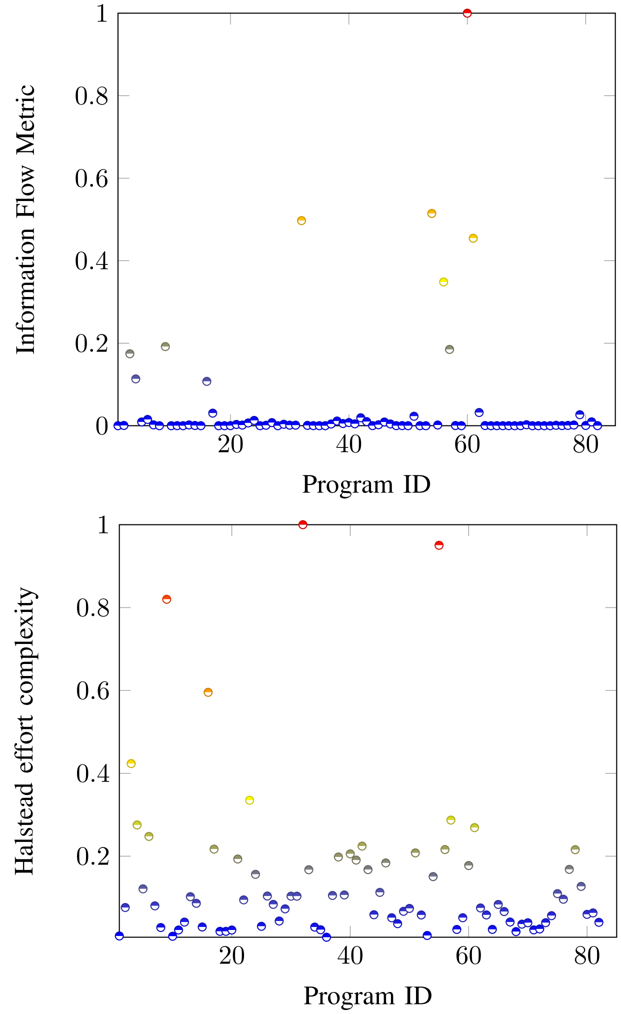


Fig. 5: Normalized scores for the Information Flow Metric (IFC) and Halstead Effort Complexity for all considered programs.

Software complexity metrics	τ_E	p-value$_E$	τ_N	p-value$_N$
Number of Elements	0.342	$8.192e^{-6}$	0.368	$2.315e^{-6}$
Cyclomatic Complexity	0.225	0.003	0.252	0.001358
Information Flow Metric	0.264	0.0005	0.345	$9.116e^{-06}$
Halstead Volume	0.328	$1.878e^{-5}$	0.351	$6.25e^{-6}$
Halstead Difficulty	0.208	0.006	0.125	0.1061
Halstead Effort	0.320	$2.882e^{-5}$	0.320	$3.876e^{-5}$

TABLE III: Kendall correlation coefficient (τ) and p-value between software complexity metrics and the test effort. The test effort was expressed by two proxy scores: test suite execution time (E) and the number of test cases in a test suite (N).

Difficulty, Volume and Effort). We should note here that the p-value$_N$ for Halstead Difficulty is 0.1, thus showing that for this measure the correlation is not strong. Overall, the results show that all coefficients, except for Halstead difficulty measure, are significant. Table III gives the correlation between the different complexity scores and the test suite execution time and the number of test cases measures of test effort. For all programs, we see a low to moderate correlation between software complexity and the number of test cases created as well as for the test suite execution time (with a highest correlation coefficient of 0.368 for the Number of Elements (NoE) metric). These results show that there is a statistical relationship between software complexity measures and test effort measures for FBD programs and test data for a real industrial system engineered by Bombardier Transportation Sweden AB. The NoE metric achieved the highest correlation score. Interestingly enough, the cyclomatic complexity metric, a structure metric, obtained a lower correlation than NoE size metric. This can be taken as an argument in favor of not measuring the structure of FBD programs in this way.

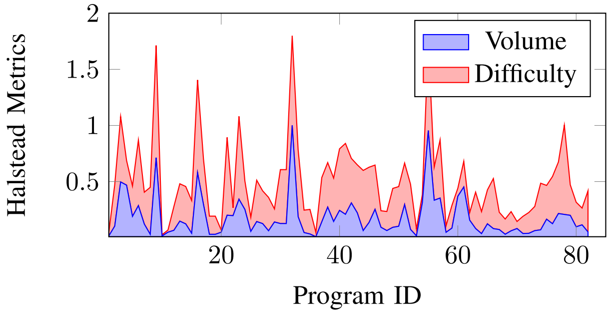


Fig. 6: A normalized area plot showing two Halstead metrics (i.e., Volume and Difficulty) with a similar distribution of scores across all programs in TCMS.

> Our results suggest that, for 82 industrial programs, there is a low to moderate correlation between the test effort (i.e., the number of created test cases and the test execution time) and the software complexity of a program. The size of the software (i.e., number of elements measure) provides the highest correlation with the test effort.

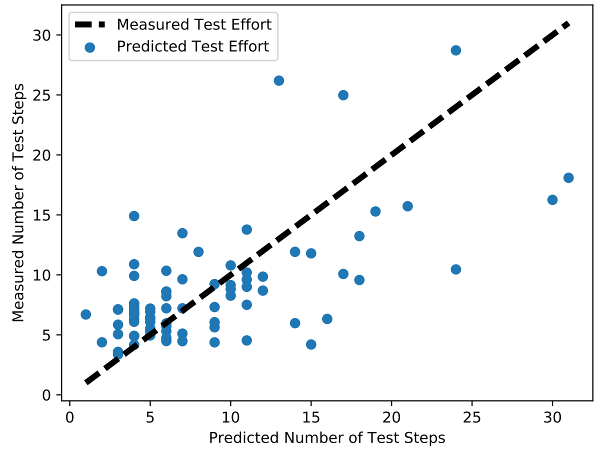
C. Discussions

We examined a train control software, which is a valid and representative case for industrial software and IEC 61131-3 FBD software used in the embedded system domain. By using the complexity measurement results and the test effort dedicated for testing the software (i.e., the number of tests cases of a test suite and the execution time of a test suite), we indicated that there is a statistical relationship between software complexity and test effort. However, the correlation is low to moderate (i.e, 0.368 Kendall's τ coefficient is the highest correlation score). There are indications that higher FBD software complexity does not imply a higher test effort.

As an effort to implement a test effort prediction model, we designed a linear regression model of the test effort using several software complexity metrics. The idea is to predict a dependent variable using correlated independent variables. We use a linear regression [27] (i.e., multiple linear regression (multiple independent variable)) to show how the results and the TIQVA tool can be used predict the test effort required for adequate testing. In practice, we used a linear regression model of the test effort measure using the following measures:

$$\beta_1 C_1 + \beta_2 C_2 + \cdots + \beta_{n-1} C_{n-1} + \beta_n C_n = T_{\text{measure}}, \quad (1)$$

where the test effort measure T_{measure} is a linear function of different weighted software complexity scores C_ns. The linear regression model shown in equation 1 could be used to predict the test effort for an industrial control software after determining the weight values (βs), and it requires an existing set of software complexity measurements and a test effort measure to be solved.



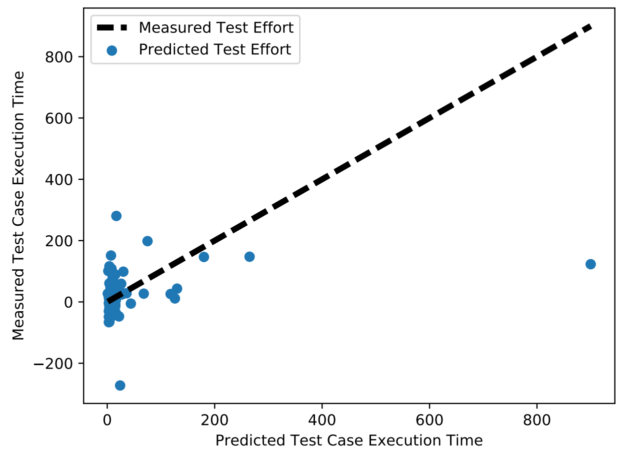


Fig. 7: Graphs showing the predictions of the trained linear regression model for the test effort proxy measures (i.e., number of test cases and test suite execution time).

Using the previously measured software complexity scores for the 82 FBD programs as the input data set and the two test effort proxy measures as the output data set, we determined the weights using a Python machine learning library [28]. We used the *Linear Regression* module to determine the weights as well as to assign a variance score (i.e., the amount of correct predictions of the model with 1.0 being the highest score). Two regression models have been developed based on the previous equation (i.e., Equation 1) and several complexity metrics for FBD programs:

$$\beta_{\text{NoE}} C_{\text{NoE}} + \beta_{\text{CC}} C_{\text{CC}} + \beta_{\text{HEC}} C_{\text{HEC}} + \beta_{\text{IFM}} C_{\text{IFC}} = T_{\text{N}} \quad (2)$$

$$\beta_{\text{NoE}} C_{\text{NoE}} + \beta_{\text{CC}} C_{\text{CC}} + \beta_{\text{HEC}} C_{\text{HEC}} + \beta_{\text{IFM}} C_{\text{IFC}} = T_{\text{E}}, \quad (3)$$

where NoE is the Number of Elements, CC is the Cyclomatic Complexity, HEC is the Halstead Effort Complexity,

Test Effort Measures	β_{NoE}	β_{CC}	β_{HEC}	β_{IFM}	MSE	Score
Number of Test Cases	$1.118e^{-7}$	$4.69e^{-2}$	$-6.528e^{-4}$	$1.36e^{-7}$	17.31	0.55
Test Suite Execution Time	1.92	-1.4	$-3.82e^{-3}$	$-5.56e^{-6}$	5077.68	0.54

TABLE IV: The complexity weights, mean square error of predictions based on the weights and the variance score of these predictions.

IFC is the Information Flow Complexity, N and T stand for the number of test cases and test suite execution time respectively. Only Effort has been taken into account from the different HC metrics.

Based on our results, we assumed that the linear regression model will not have a high prediction accuracy considering the low to moderate correlation between software complexity metrics and test effort. After examining the trained linear regression model using the full data set of 82 programs for training and for testing the model, we report the results in Table IV. These results show that only half of the test effort predictions were accurate. The mean squared value was low when the model tried to predict the number of test cases and significantly higher for the test suite execution time. This can be explained by the non-linear and irregular values of the execution time test effort shown in Figure 3. Another characteristic of the model is the achieved higher β weight value for the Number of Elements (NoE) messure in both models.

Figure 7 shows the predictions (in blue scatter points) of the test effort in contrast to the measured test effort (black line). Although the predictions of the test execution time are similar to the predictions for the number of test cases, these are clustered around one area (i.e., lower test execution time), while the number of test cases is distributed across the complete test effort scores spectrum. This shows that a test effort estimation can be made using software complexity measurement scores. Since test cases in industry are designed using different information sources (e.g., in the safety critical domain using functional specifications and human domain knowledge), one would need to include other metrics for predicting the test effort. The results are promising, but but we only achieved a rough estimator. However, researchers and practitioners should fine tune this kind of estimations by taking into account other metrics for software artifacts (e.g., specification, test knowledge) which are heavily influencing the overall test effort.

V. Threats to Validity

Industrial control software used in PLCs can be programmed in a wide variety of programming languages, such as IEC 61131-3 FBD and ST languages. In this paper, we examined how software complexity metrics can be applied on industrial control software developed using FBDs in one company (i.e., Bombardier Transportation Sweden AB), thus narrowing the scope of the study. However, we argue that the examined software (TCMS) shows general characteristics of the safety-critical industrial domain.

The set of software complexity metrics chosen to be used on FBD programs is not complete by any means. We did not used other complexity measures such as entropy [29], Kolmogorov complexity [30]. The purpose of this paper is to explore the test effort relation with software complexity. This is a first step in an effort in this endeavour.

The test effort is not straightforward to measure and requires knowledge of the multiple phases performed during software testing including test creation. In this study we focus on two proxy measures for test effort. More studies are needed to generalize the results of this paper.

VI. Conclusion

This paper presents an initial exploration on how software complexity can be applied on industrial domain-specific software written in the FBD graphical programming language. We used four, well known, software complexity metrics. We studied the relationship between test effort (i.e., number of test cases and test execution time of a program's test suite) and the program complexity scores From the 82 industrial FBD programs we studied, we drew the following conclusions:

- There is a low to moderate correlation between the effort needed to test a program and its complexity.
- The size of the software in terms of the number of elements provides the highest correlation with the test effort.

The results from the study also indicate that other aspects than code complexity might be required to better capture the relationship between the implemented and specified software artifacts and test effort. Also, other complexity dimensions of the FBD programs (e.g., function block relationships, block coupling and timing) could be used to improve the measurement of complexity for an FBD program.

Acknowledgment

This work is partially funded from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737494 and The Swedish Innovation Agency, Vinnova (MegaM@Rt2).

References

[1] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigation0," *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, 1984.

[2] J. E. Gaffney, "Estimating the number of faults in code," *IEEE Transactions on Software Engineering*, no. 4, pp. 459–464, 1984.

[3] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 367–377.

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

[5] 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 Science & Business Media, 2010.

[6] T. Mens, "Research trends in structural software complexity," *arXiv preprint arXiv:1608.01533*, 2016.

[7] A. Abran, *Software metrics and software metrology*. John Wiley & Sons, 2010.

[8] E. J. Weyuker, "Evaluating software complexity measures," *IEEE transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[9] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Software Engineering Journal*, vol. 3, no. 2, pp. 30–36, 1988.

[10] D. Kafura and G. R. Reddy, "The use of software complexity metrics in software maintenance," *IEEE Transactions on Software Engineering*, no. 3, pp. 335–343, 1987.

[11] K. Kevrekidis, S. Albers, P. J. Sonnemans, and G. M. Stollman, "Software complexity and testing effectiveness: An empirical study," in *Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual*. IEEE, 2009, pp. 539–543.

[12] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[13] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE transactions on Software Engineering*, no. 5, pp. 510–518, 1981.

[14] M. Olszewska, Y. Dajsuren, H. Altinger, A. Serebrenik, M. Waldén, and M. G. van den Brand, "Tailoring complexity metrics for simulink models," in *Proccedings of the 10th European Conference on Software Architecture Workshops*. ACM, 2016, p. 5.

[15] A. Muslija, "amuslija/fbd-complexity-tool 0.0.2," May 2017. [Online]. Available: https://doi.org/10.5281/zenodo.580762

[16] M. H. Halstead, *Elements of software science*. Elsevier New York, 1977, vol. 7.

[17] C. L. McClure, "A model for program complexity analysis," in *Proceedings of the 3rd international conference on Software engineering*. IEEE Press, 1978, pp. 149–157.

[18] D. N. Card and R. L. Glass, *Measuring software design quality*. Prentice-Hall, Inc., 1990.

[19] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured testing: A testing methodology using the cyclomatic complexity metric*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996, vol. 500, no. 235.

[20] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.

[21] R. Conradi and A. I. Wang, *Empirical methods and studies in software engineering: experiences from ESERNET*. Springer, 2003, vol. 2765.

[22] 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.

[23] H. K. Leung and L. White, "A cost model to compare regression test strategies," in *Software Maintenance, 1991., Proceedings. Conference on*. IEEE, 1991, pp. 201–208.

[24] M. G. Kendall, "Rank correlation methods." 1948.

[25] J. Cohen, "Statistical power analysis for the behavioral sciences. 2nd," 1988.

[26] "Kendall rank coefficient — r tutorial," http://www.r-tutor.com/gpu-computing/correlation/kendall-rank-coefficient, (Accessed on 04/27/2017).

[27] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2015.

[28] "scikit-learn: machine learning in python scikit-learn 0.18.1 documentation," http://scikit-learn.org/stable/index.html, (Accessed on 04/29/2017).

[29] C. E. Shannon, "Prediction and entropy of printed english," *Bell Labs Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[30] A. N. Kolmogorov, "On tables of random numbers," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 369–376, 1963.