Received 6 August 2023, accepted 3 September 2023, date of publication 6 September 2023, date of current version 14 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3312556

RESEARCH ARTICLE

Development of Testability Prediction Models Considering Complexity Diversity for C Programs

HYUN-JAE CHOI AND HEUNG-SEOK CHAE
Department of Electrical and Computer Engineering, Pusan National University, Busan 46241, South Korea

Corresponding author: Heung-Seok Chae (hschae@pusan.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1F1A1060056).

ABSTRACT Testability prediction can help developers identify software components that require significant effort to ensure software quality, plan test activities, and recognize the need for refactoring to reduce the test effort. Previous studies have predicted code coverage as a measure of testability based on software metrics. However, these studies have primarily used object-oriented software with simple code structures. Industrial software developed using C is often more complex than the object-oriented software used in these studies. Models trained primarily on low-complexity training data may have insufficient training for the testability of high-complexity software. In this study, we developed a testability prediction model for C programs by considering the complexity diversity. We analyzed the impact of the complexity of the training/test data on the testability prediction model for C programs. The results showed that the model with the best performance achieves an MAE of 7.436 and an R^2 of 0.813. Moreover, the results demonstrated that as the complexity diversity of the training data decreased, MAE increased from 5.203 to 6.361, and R^2 decreased from 0.809 to 0.725. Furthermore, the performance of the model trained with low complexity-diversity deteriorated as the complexity level of the test data increased, with MAE increasing from 3.498 to 6.631, and R^2 decreasing from 0.841 to 0.687. Additionally, in the correlation analysis between the model performance and the difference in the complexity of the training and test data, a strong correlation was observed, with MAE of 0.898 and R^2 of -0.848.

INDEX TERMS Coverage prediction, metrics, regression, software testability, testing.

I. INTRODUCTION

Testing is essential to ensuring the quality of software during the software development process. Testing is a costly activity in the software development industry, accounting for approximately 50% of total software development costs [1]. In other words, improving the effectiveness and efficiency of testing can lower software development costs. Testability is a measure of testing effectiveness and efficiency [2]. High testability means that the software can be tested effectively and efficiently.

Testability prediction can help developers identify software components that require significant effort to ensure software quality, plan test activities, and recognize the need for refactoring to reduce the test effort. Previous studies predicted testability through regression analysis using metrics such as cyclomatic complexity, which measures the structural characteristics of software [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13].

Some testability prediction studies have used testing information, such as test case metrics or testing time, as measures of testability in terms of test effort [3], [4], [5], [6], [7], [8], [9], [10]. However, accurately predicting testability based on testing information is challenging because testing depends on the capabilities of the test team or individual testers.

Among the recent testability prediction methods, existing studies use code coverage as a measure of testability in terms of test effectiveness [11], [12], [13]. Code coverage is one of the representative indicators that can measure test effectiveness as a test criterion required by ISO 26262 [14],

The associate editor coordinating the review of this manuscript and approving it for publication was Claudia Raibulet.

IEC 60601 [15], EN 50129 [16], IEC 61508 [17], and DO 178C [18] industry standards.

Previous studies mainly constructed prediction models for object-oriented open-source software [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. The software used in previous studies often does not have a complex code structure; therefore, these studies were conducted using a high proportion of low-complexity source code. For example, in SF110 [19], which consists of 110 open-source Java projects used in existing studies [12], [13], only 2.9% of the non-abstract methods have a cyclomatic complexity (CC) exceeding eight or a number of structuring levels (SL) exceeding four.

However, the upper limitations of the complexity metrics in C/C++ industry standards such as MISRA [20], JPL [21], and JSF [22] indicate that a substantially larger proportion of high-complexity source code may be present in industrial C/C++ software than in previous studies. In GCC 13.1.0, a representative C compiler, functions that exceeded a CC of eight or an SL of four were observed in 7.9% of the functions. Furthermore, in our analysis of a large-scale C source code, which consists of 6,873 functions from a real-world automotive industry domain, we found that 9.6% of the functions exceeded a CC of eight or an SL of four.

If the proportion of high-complexity training data is low, training for the testability of high-complexity software may be insufficient. In other words, to enhance the testability-prediction performance of high-complexity software, it is necessary to use high-complexity training data for model construction. Low-complexity functions, with fewer decision statements and lower nesting levels, have relatively simple conditions. This simplicity of the conditions makes it easier to achieve high code coverage. However, the conditions for high-complexity functions are relatively complex, making it difficult to achieve high code coverage. Therefore, predictions for not only low-complexity functions but also high-complexity functions are important.

We developed a testability prediction model using training data with various complexities for C programs. C language is widely used in embedded systems since it offers efficient memory management and hardware access. C language is being used in popular microcontroller platforms such as the AVR and ARM Cortex-M, as well as in automotive systems. Moreover, many legacy systems have been written in C, and these systems still require maintenance and testing. The testability prediction model for C programs can be usefully utilized by test engineers who want to develop testability models, and by developers who aim to evaluate the quality of C programs.

We then analyzed the differences in prediction performance based on complexity diversity, which refers to the diversity in program complexity. First, we confirmed that a model with excellent prediction performance can be constructed using training data under various complex conditions. In addition, we analyzed whether the prediction performance of the model was affected as the complexity of the training data became biased and diversity decreased. We also examined whether prediction performance decreased when high-complexity test data were input into a model trained with data that exhibited low complexity-diversity. Finally, we verified whether the difference in complexity levels between the test and training data affected the prediction performance of the model. Analysis of the impact of complexity diversity on testability prediction can be useful for test engineers who want to improve the performance of testability prediction models.

We developed a methodology and conducted experiments to address these research questions.

- RQ1: Does a model that considers complexity diversity for C programs exhibit high prediction performance? Which model exhibited the best performance?
- RQ2: Do training data with low complexity-diversity degrade testability prediction performance? Does a lower complexity diversity of the training data lower the prediction performance?
- RQ3: Does the model trained with low complexity-diversity data present lower prediction performance depending on the complexity level of the test data? Does a higher complexity level in the test data result in lower prediction performance?
- RQ4: Does a larger complexity-level difference between the training and test data result in a greater performance difference? Is there a correlation between the complexity level difference in the training and test data and the prediction performance?

To answer these questions, we developed a testability prediction model based on highly diverse training data. In addition, to analyze the impact of the complexity diversity of the training data on the performance of the model, we constructed models based on training data with limited complexity diversity.

To obtain high-complexity-diversity data, we determined the maximum/minimum value of the metric based on the maximum allowed complexity metrics of the MISRA [20], JPL [21], and JSF [22] C/C++ industry standards and automatically generated software under test (SUT) that satisfied various metric combinations. In addition, test data were generated and branch coverage was measured using search-based test data generation, which is a representative test data generation method that effectively generates test data.

A testability prediction model was constructed using a regression analysis based on the training data of various metric combinations. Metrics applicable to C programs affecting the search and solution spaces were selected and used as independent variables for training. When the structure of the code becomes complicated, the conditions required to achieve coverage also become complicated, thus reducing the solution space and making it difficult to find test data that improve the coverage. Branch coverage, a testability measure of test effectiveness, was used as the dependent variable.

The main contributions of this study are as follows:

- Unlike previous studies that have primarily focused on object-oriented software, we develop a high-performance testability prediction model specifically for C programs, taking into account complexity diversity.

- We empirically investigate that low complexity-diversity in training data can reduce testability prediction performance. This demonstrates the importance of complexity diversity in training data for testability prediction, a problem not thoroughly examined in previous studies.
- We explore the relationship between training and test data complexity, proving that testability prediction performance decreases with less complex training data and more complex test data. This finding highlights the need for high complexity-diversity training data to improve testability prediction performance for complex C programs.

The structure of the remainder of this paper is as follows: Section II describes the related work. Section III describes the methodology used to construct a testability prediction model with a high complexity-diversity for C programs. Section IV analyzes the experiments for each research question. Section V discusses the results and limitations. Finally, Section VI concludes the study.

II. RELATED WORK

Several standards have defined testability, none of which specifies concrete measurement methods, leading to the study of various approaches. According to Garousi et al. [23], there are more than 30 definitions of testability. Representative definitions of testability are as follows:

- ISO 9126 [24]: "attributes of software that bear on the effort needed to validate the software product."
- ISO 25010 [2]: "degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met."

Existing studies on metric-based testability prediction have focused mainly on testability in terms of test effort, as defined by ISO 9126, and test effectiveness, as defined by ISO 25010. Studies have been conducted on testing efforts using test information [3], [4], [5], [6], [7], [8], [9], [10] and on testing effectiveness using code coverage [11], [12], [13].

Gupta et al. [3] proposed a testability index defined using a fuzzy approach to CC and object-oriented metrics for 25 Java classes. They divided the values into preferred, acceptable, and not acceptable for each metric and built a fuzzy model to define the testability index. In their analysis of the correlation between the testability index and unit testing time, they found that the testability index was strongly correlated with testability. However, generalizing the results is challenging because the testing time depends on the capabilities of the tester and SUT features. Bruntink and Deursen [4] analyzed the correlation between object-oriented metrics and testability using the lines of code for test classes and the number of test cases as measures of testability in five Java systems.

Singh and Saha [5] and Badri and Toure [6] constructed prediction models for object-oriented metrics and testability using the lines of code for test classes and the number of assertions. Singh and Saha [5] used linear regression and regression trees on three open-source Java systems, and Badri and Toure [6] used logistic regression analysis on three open-source Java systems.

Toure and Badri [7] built a testability prediction model based on LOC and object-oriented metrics using four regression algorithms on ten open-source Java systems. They classified a class as tested or untested as a measure of testability, assuming that the classes selected by the testers required test effort. They predicted whether a class needed to be tested on the basis of these metrics.

Albattah [8] built a testability prediction model using package-level cohesion metrics and logistic regression on five open-source Java systems. They measured testability by the lines of code for test classes.

Terragni et al. [9] conducted a correlation analysis between 28 object-oriented metrics and testability using normalizing test effort with test quality on open-source Java systems. Six test-case metrics were used as test effort, while statement coverage, branch coverage, and mutation score were used as test quality. They showed that normalizing test effort with test quality increases the correlation between object-oriented metrics and test effort.

Testing depends on the capabilities of the test team or the individual testers. Therefore, it is difficult to predict the test-case metrics or testing times using only these metrics. According to Bajeh [10], there is a significant relationship between the metrics and test-case metrics, but the magnitude of the relationship is low. This implies that the metrics alone do not accurately measure the task of developing test cases.

Wang et al. [25] proposed a state-based testability model for testing systems with multi-state characteristics. They used fault detection rate, fault isolation rate, and state detection rate as measures of testability. However, the performance of the model can vary depending on the domain knowledge and capabilities of the test engineer constructing the state-based testability model.

Recent studies have used code coverage as a measure of testability effectiveness. Grano et al. [11] constructed a prediction model for package, object-oriented, CK, Halstead metrics, and branch coverage by using four regression algorithms on seven open-source Java systems. They measured branch coverage using search-based test data generation with GA and random approaches.

Zakeri-Nasrabadi and Parsa [12], [13] constructed a testability prediction model based on object-oriented metrics for 110 open-source Java systems. In [12], five regression algorithms were used to build a prediction model using the product of the average branch coverage, statement coverage, and minimum test case ratio to improve the coverage as a measure of testability. In [13], seven regression algorithms were used to build a prediction model using the average branch coverage and statement coverage divided by the average time to improve coverage as a measure of testability.

These studies were conducted on object-oriented systems with a high proportion of simple methods without analyzing the diversity of complexity in the training data. According to an analysis using Understand [26], a commercial

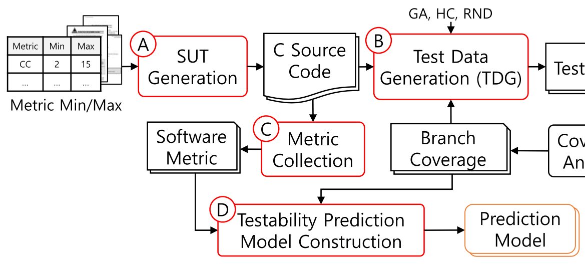


FIGURE 1. Testability prediction model construction process.

metric analysis tool, the proportions of non-abstract methods (excluding the test suite) with a CC of eight or less and an SL of four or less in the systems used by Grano and Zakeri-Nasrabadi were both 97.1%. However, a methodology for constructing a testability prediction model using software metrics for C programs has not yet been established, and there has been no detailed analysis of the impact of complexity on testability prediction.

III. METHODOLOGY

This section describes the construction of a testability prediction model for C programs based on high-complexity-diversity training data. It describes a method for generating software under test (SUT) to obtain training data with high complexity-diversity, a method for generating test data to measure coverage, a metric collection used as an independent variable, and a method for predicting testability using regression analysis. Fig. 1 illustrates the procedure for constructing the testability prediction model.

We generated SUTs with high complexity-diversity implemented in C using the metric maximum/minimum range determined using the upper limit presented in C/C++ industry standards. We performed search-based test data generation on the generated SUT for the branch coverage measurements. We analyzed the metrics of SUT and collected those to be used as independent variables in the construction of a testability prediction model. We built a testability prediction model using the measured branch coverage as a dependent variable, collected metrics as independent variables, and predicted testability using the constructed model.

A. SUT GENERATION

SUT was automatically generated to obtain training data with a high complexity-diversity. Complexity and size metrics that can affect the performance of search-based test data generation, such as cyclomatic complexity (CC), number of structuring levels (SL), number of equality operators (NOEO), and number of parameters (NOP), were selected as the criteria for SUT generation. The higher the CC, the higher the number of branches that must be executed to increase branch coverage. As SL increases, the maximum nesting depth of the decision statement also increases, complicating the conditions that must be satisfied to execute the decision statement. The larger the NOEO, the more difficult it is to determine a solution that satisfies the branch conditions. This is because the proportion of solutions that satisfy the equality comparison in the domain space is limited. Therefore, the search for a solution to the conditions in which the equality operator exists is more difficult than the search for conditions in which the relational operator exists. As NOP increases, the size of the domain space that must be explored for test data generation becomes wider.

TABLE 1. Metric min/max range.

Metric	Basis	Min value	Max value
CC	MISRA	2	15
SL	MISRA	1	min(6, CC - 1)
NOEO	-	0	CC - 1
NOP	JPL, JSF	2	6

The range of each metric value is determined based on the upper limit presented in the MISRA [20], JPL [21], and JSF [22] C/C++ industry standards. Table 1 lists the ranges of the metric values used to generate SUT.

The standard only provides the upper limit of the metrics and not the minimum value. In this study, the minimum value is established by assuming at least one decision statement and two or more parameters. Because the nesting depth cannot exceed the number of decision statements, we used a smaller value between CC-1 and 6 (six being the upper limit in the standard) as the maximum value for SL. As the standard does not reference NOEO, we assumed that NOEO would be used less frequently than the decision statements and used CC – 1 as its maximum value. Based on these constraints, 3,320 feasible metric combinations were identified.

We generated ten programs for each metric combination, for a total of 33,200 programs. Because automatically generated programs may contain infeasible branches, we use Joggie [27], a tool for detecting infeasible codes, to remove these infeasible branches. Joggie assessed its feasibility by using the Princess solver [28]. The Princess solver, a tool that checks whether the conditions are satisfiable, serves as an essential feature in infeasible code detection. The Princess solver is utilized in JavaSMT [29], a unifying Java interface for SMT solvers, and Eldarica [30], a predicate abstraction-based model checker.

B. TEST DATA GENERATION

Search-based test data generation is an automatic generation method of test data that uses search algorithms to explore test data that meet test goals. We generated test data using branch coverage as a test goal and the genetic algorithm (GA), hill climbing (HC), and random (RND) methods as search algorithms. The GA is a global search algorithm that finds values that improve branch coverage by changing values through crossover and mutation operations. Values that satisfy relational comparisons are found quickly; however, more searches are required to identify the values that satisfy equality comparisons. HC is a local search method used to determine a value that improves branch coverage by changing the value to a relatively small value, which is the neighboring

TABLE 2. Test data generation parameters.

Algorithm	Parameter	Value
HC	Search method	Alternating variable method
GA	Selection operator	Ranking
	Selection strategy	Elitism
	Crossover operator	One point
	Crossover probability	1
	Mutation operator	Bit flip
	Mutation probability	1 / len
	Gene encoding	Binary
GA, RND	Population size	50

value of the current value, as the next value. The HC tends to find values that satisfy the equilibrium comparison more quickly than the GA. Because RND is a method for generating arbitrary values without a separate search technique, the branch coverage is determined by the ratio, the proportion of the solution space within the domain space. We used the value from [31] for the crossover probability, and the values from [32] for the rest of the search algorithm parameters. Table 2 lists the search algorithm parameters used to generate test data.

For the range of each variable, [-2^{31}, 2^{31}-1], which is the range of values that can be expressed as a variable of 4 bytes, was used. The results were derived by repeating each program ten times to reduce the randomness of the search-based testing method. According to an existing study [33], search-based testing results require at least ten repetitions to achieve minimal statistical power.

C. METRIC COLLECTION

We selected complexity metrics that affected the search for test data to improve branch coverage and collected the independent variables. All the selected metrics were applicable to C programs. CC, SL, NOP, and NOEO were used to generate the SUT, and the number of paths (NPath) [34], which are metrics that express software complexity and Halstead complexity [35], program length, program vocabulary, volume, and difficulty, were used.

The CC, SL, NOP, and NPath metrics were measured using Understand [26], a commercial metric measurement tool, and the Halstead complexity metrics were measured using PC Lint Plus [36], another commercial metric measurement tool. NOEO was measured by developing a custom tool to count the number of == and != operators within the function.

D. TESTABILITY PREDICTION MODEL CONSTRUCTION

The histogram-based gradient boosting regressor (HGBR), random forest regressor (RFR), multilayer perceptron regressor (MLPR), decision tree regressor (DTR), linear regressor (LR), Huber regressor (HR), and stochastic gradient descent regressor (SGDR) are representative regressors used to build testability prediction models in existing studies [11], [12], [13]. HGBR, RFR, and DTR are tree-based methods; MLPR is a neural network-based method; and LR, HR, and SGDR are linear-based methods.

HGBR [37] is a representative ensemble machine-learning method. The training data were separated into bins for each feature based on feature percentiles. The gradient boosting algorithm trains a sequence of decision-tree sizes to minimize the global loss function. Gradient boosting algorithms are used to train a sequence of decision tree sizes to minimize the global loss function. This allows the algorithm to leverage histograms instead of relying on sorted continuous values when building the trees.

RFR [38] is another representative ensemble machine-learning method. It combines multiple decision trees, each built using subsamples of the dataset, and uses averaging. By averaging these predictions, some errors can be canceled out because of the tendency of individual decision trees to overfit.

DTR [39] predicts the value of a target variable by learning the decision rules inferred from data features in a tree structure. A decision tree was constructed by dividing the dataset into smaller subsets. DTR is easy to understand and interpret. However, it has disadvantages, such as overfitting and generating biased trees when some classes are dominant.

MLPR [40] is a fully connected class of feedforward artificial neural networks consisting of multiple layers of numerous computational neurons. An MLPR consists of at least three layers of nodes: an input layer, an output layer, and one or more nonlinear hidden layers. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. Neural networks are difficult to tune in terms of hyperparameter variables but have the advantage of training nonlinear interactions between features.

LR [41] is the first type of regression analysis used to model the relationship between a target variable and one or more input variables. The model parameters were estimated by minimizing the error between the predicted and actual values. LR has the advantages of simplicity and ease of interpretation.

HR [42] is a linear regression method that is robust to outliers. It uses the Huber loss function, which is less sensitive to outliers in the data than the mean squared error.

SGDR [43] is a linear regression method that uses stochastic gradient descent as an optimization algorithm to fit the model. The model is particularly useful for large-scale sparse datasets.

Scikit-learn [44] was used to build the testability prediction models. Scikit-learn is a representative Python machine-learning library that provides various classification, regression, and clustering algorithms. We built testability prediction models using scikit-learn's HistGradientBoostingRegressor, RandomForestRegressor, DecisionTreeRegressor, MLPRegressor, LinearRegression, HuberRegressor, and SGDRegressor. For hyperparameter tuning of the regression model, GridSearchCV, a search method that investigates all parameter combinations, was used.

Hyperparameter tuning was performed for each regression algorithm to identify optimal parameters. The

TABLE 3. Hyperparameter search range.

Model	Hyperparameter name	Search range
HGBR	loss	['squared_error', 'absolute_error']
	max_depth	range(3, 50, 5)
	min_samples_split	range(5, 50, 10)
	max_iter	range(100, 500, 100)
RFR	n_estimators	range(50, 200, 50)
	criterion	['squared_error', 'absolute_error']
	max_depth	range(3, 50, 5)
	min_samples_split	range(2, 30, 2)
DTR	criterion	['squared_error', 'absolute_error']
	max_depth	range(3, 50, 5)
	min_samples_split	range(2, 30, 2)
MLPR	hidden_layer_sizes	[(128, 64), (256, 100), (512, 256, 100)]
	activation	['relu', 'tanh', 'logistic']
	learning_rate	['constant', 'adaptive']
	max_iter	range(100, 500, 50)
HR	alpha	pow(2, np.linspace(-30, 20, num=15))
	max_iter	range(50, 1000, 50)
SGDR	loss	['squared_error', 'huber']
	penalty	['l2', 'l1', 'elasticnet']
	learning_rate	['invscaling', 'optimal', 'constant', 'adaptive']
	max_iter	range(50, 1000, 50)

hyperparameter is an adjustable parameter whose value is used to control the model training process. The model performance significantly depends on the hyperparameters.

We used the values from [13] for HGBR, RFR, DTR, SGDR, [12] for MLPR, and [11] for HR to tune the parameters and search ranges. However, for HR, we added the max_iter parameter because the regression analysis library reported a warning message to increase max_iter. Table 3 presents the search scope for hyperparameter determination.

We constructed testability prediction models for the GA, HC, and RND datasets. Each dataset comprised 33,200 data points derived from test data generation on SUTs of all complexity combinations.

The testability prediction models were evaluated using performance metrics such as the coefficient of determination (R^2), mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). R^2 represents the extent to which an independent variable explains a dependent variable. The MAE is an easy-to-interpret error metric because it has the same units as the actual value. MSE is an error metric that employs squared operations to sensitively reflect errors. In other words, if an outlier exists, the value fluctuates significantly compared with the MAE. RMSE is an error metric that takes the square root of MSE and converts it into units similar to the actual values. R^2 implies that the larger the value, the better the performance of the model, whereas MAE, MSE, and RMSE, as error metrics, indicate that the smaller the value, the better the performance of the model.

IV. EXPERIMENTS

This section describes the experiments conducted to answer the research questions. First, to verify whether a high-performance testability prediction model can be constructed using C programs with complexity diversity, we built prediction models using complexity diversity training data and various regression algorithms. The performance of the models was analyzed using statistical methods. To determine whether training data with low complexity-diversity reduces testability prediction performance, we constructed various prediction models using training data that exhibited different levels of low complexity-diversity and compared their performances. Furthermore, we investigated whether the prediction performance of models trained with low complexity-diversity data was degraded as the complexity level of the test data increased by sampling test data with various complexity levels and comparing their performances. Finally, we analyze the correlation between the complexity-level differences between the training and test data and the performance metrics to determine whether the difference in the complexity levels between the training and test data leads to performance degradation.

A. RQ1: CONSTRUCTION OF PREDICTION MODELS CONSIDERING COMPLEXITY DIVERSITY FOR C PROGRAMS

Previous studies constructed testability prediction models using data with low complexity-diversity. In other words, both the training and test data had a higher proportion of low complexity. However, low-complexity source codes tend to have fewer branches and simpler conditions, making it easier to achieve high code coverage. We investigated whether we could build high-performance models for C programs using training data with high complexity-diversity by increasing the proportion of high-complexity data.

We built models using seven learning algorithms – HGBR, RFR, DTR, MLPR, LR, HR, and SGDR – with high complexity-diversity training data generated using a combination of metrics within the maximum and minimum ranges determined based on C/C++ industry standards. Because the coverage achievement varied for each TDG algorithm, the datasets were used separately for the three TDG algorithms: GA, HC, and RND. In other words, 21 models were constructed by combining the seven regression algorithms with the three TDG datasets.

We validated the performance of the models using k-fold cross-validation with randomly shuffled datasets comprising 33,200 data points for each TDG algorithm. K-fold cross-validation divides the dataset into k parts, using k-1 parts as training data and the remaining one as validation data. While there is no formal rule for the choice of K, we used five which is usually selected [45].

The experiment was repeated 30 times to reduce the data randomness caused by shuffling. Performance metrics

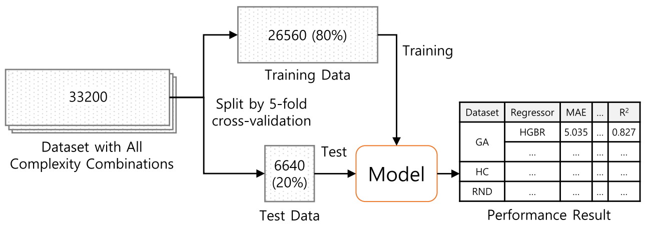


FIGURE 2. **Experimental process of RQ1.**

used were MAE, MSE, RMSE, and R^2. Fig. 2 shows the experimental process of RQ1.

We calculated the mean MAE and R^2 to verify whether the models trained with high complexity-diverse data exhibited a high prediction performance. Fig. 3 shows the mean MAE and R^2 results for RQ1.

We observed a mean MAE between 4.502 and 7.734, and a mean R^2 between 0.579 and 0.866. Tree-based HGBR, RFR, and DTR showed better mean performances than neural-network-based MLPR and linear-based LR, HR, and SGDR. Considering the R^2 values of 0.525 [11], 0.680 [12], and 0.678 [13] in previous testability prediction studies, the results indicate that the model has moderately high performance. The mean MAE and R^2 performance of the models improved in the order of HGBR, RFR, DTR, MLPR, LR, HR, and SGDR for all datasets.

We compared the performance of the models built using each regression algorithm to determine which model exhibited a statistically significant difference. We performed an ANOVA to analyze the differences among the means for statistical analysis. Prior to the ANOVA, we performed Levene's test [46] to assess the homogeneity of variances for the measured performance metrics. As Levene's test did not pass, we used Welch's ANOVA, an analysis method applicable when the homogeneity of variances assumption was not met, and the Games-Howell post-hoc test [47] for the analysis.

Table 4 presents the experimental results for RQ1 separated by the dataset and regression algorithms. The experimental results across Tables 4 through 8 are presented as mean ± standard deviation (SD), and the letters to the right of the SD represent the groups with statistically significant differences according to the Games-Howell test results with a p-value below 0.05. Different letters indicate statistically significant differences between the groups.

In all cases, M^{HGBR} exhibited the best performance, followed by M^{RFR}. In some cases, no statistically significant differences were observed between M^{DTR}, M^{MLPR}, and M^{LR}; however, the order of model performance remained consistent: M^{DTR}, M^{MLPR}, and M^{LR}. In all cases, M^{HR} and M^{SGDR} exhibited the lowest performances.

B. RQ2: PERFORMANCE ANALYSIS BASED ON THE COMPLEXITY DIVERSITY OF TRAINING DATA

To analyze whether the prediction performance decreased with lower complexity diversity in the training data, we compared the performances of the models trained using data with low complexity-diversity. We separated the training data into low-complexity and high-complexity data and generated training data with low complexity-diversity by combining them such that the proportions of low complexity-data were 90%, 95%, 98%, and 99%, respectively. As the criteria for distinguishing between low and high complexity, we used a CC of eight or lower and an SL of four or lower. This criterion was determined based on the observation that SF110, a representative SUT used in search-based test data generation [19] and employed in related studies [12], [13], had 97.1% non-abstract methods with a CC of eight or lower and an SL of four or lower. Fig. 4 illustrates the experimental process for RQ2.

We used 4,800 training data points and 1,200 test data points for RQ2. M^{All} was trained using 4,800 training data samples from the dataset. M^{LC90} was trained using the training data sampled from 4,320 low-complexity training data points ($CC \leq 8$ and $SL \leq 4$) and 480 high-complexity training data points ($CC > 8$ or $SL > 4$). In other words, 90% (4,320 out of 4,800) of the training data comprised low-complexity data, and 10% (480 out of 4,800) consisted of high-complexity data. M^{LC95}, M^{LC98}, and M^{LC99} were trained using the same method, with the training data consisting of low-complexity data in proportions of 95, 98, and 99%, respectively. The four models, M^{LC90}, M^{LC95}, M^{LC98}, and M^{LC99}, are collectively referred to as LC models.

We used 1,200 samples of test data from the dataset, excluding the training data. To reduce the randomness of the training and test data due to sampling, the experiment was repeated 30 times.

We used HGBR, which exhibited the best performance for RQ1, as the regression algorithm to build the prediction model. The experimental results were measured and analyzed using MAE, MSE, RMSE, and R^2, and an ANOVA test was performed. Fig. 5 shows the mean MAE and R^2 results for RQ2.

The mean MAE and R^2 performances of the models were consistently better in the order of M^{All}, M^{LC90}, M^{LC95}, M^{LC98}, and M^{LC99} across all datasets. In other words, we confirmed that the mean prediction performance of the models decreased as the proportion of low-complexity data increased and the complexity diversity of the training data decreased. The differences in the mean MAE and R^2 of M^{All} compared with those of M^{HGBR} in RQ1 were owing to the use of sampled training data to limit the complexity diversity of the training data. Table 5 presents the experimental results for RQ2 separated by the dataset and the proportion of low-complexity data in the training data.

The mean performance consistently improved in the order of M^{All}, M^{LC90}, M^{LC95}, M^{LC98}, and M^{LC99} across all results. The ANOVA results do not always indicate statistically significant performance improvements. However, M^{All} always demonstrated significantly better performance than the LC models. Moreover, M^{LC90} always exhibited a significantly better performance than M^{LC98} and M^{LC99}, and M^{LC95} always exhibited a significantly better performance than M^{LC99}. Therefore, the results indicate that the prediction performance tends to decrease as the complexity

TABLE 4. RQ1. Performance of regression models in each dataset.

Dataset	Regressor	Model	MAE	MSE	RMSE	R^2
GA	HGBR	M^{HGBR}_{GA}	5.035±0.053[a]	49.235±1.295[a]	7.016±0.092[a]	0.827±0.004[a]
	RFR	M^{RFR}_{GA}	5.132±0.051[b]	51.502±1.232[b]	7.176±0.086[b]	0.819±0.004[b]
	DTR	M^{DTR}_{GA}	5.422±0.055[c]	57.128±1.251[c]	7.558±0.083[c]	0.799±0.004[c]
	MLPR	M^{MLPR}_{GA}	5.660±0.454[d]	58.156±10.948[c]	7.601±0.619[c]	0.795±0.038[c]
	LR	M^{LR}_{GA}	6.658±0.057[e]	75.460±1.587[d]	8.686±0.091[d]	0.734±0.005[d]
	HR	M^{HR}_{GA}	6.985±0.157[f]	84.668±4.510[e]	9.199±0.236[e]	0.702±0.016[e]
	SGDR	M^{SGDR}_{GA}	7.067±0.168[g]	95.373±5.470[f]	9.762±0.273[f]	0.664±0.019[f]
HC	HGBR	M^{HGBR}_{HC}	5.837±0.063[a]	64.145±1.660[a]	8.008±0.104[a]	0.745±0.006[a]
	RFR	M^{RFR}_{HC}	5.901±0.058[b]	65.893±1.462[b]	8.117±0.090[b]	0.738±0.006[b]
	DTR	M^{DTR}_{HC}	6.121±0.065[c]	70.484±1.792[c]	8.395±0.107[c]	0.720±0.007[c]
	MLPR	M^{MLPR}_{HC}	6.518±0.585[d]	75.41±11.903[d]	8.662±0.621[d]	0.700±0.047[d]
	LR	M^{LR}_{HC}	6.634±0.064[d]	78.484±1.793[e]	8.859±0.101[e]	0.688±0.007[e]
	HR	M^{HR}_{HC}	6.798±0.154[e]	84.896±3.474[f]	9.212±0.181[f]	0.663±0.013[f]
	SGDR	M^{SGDR}_{HC}	7.734±0.169[f]	105.986±4.081[g]	10.293±0.197[g]	0.579±0.017[g]
RND	HGBR	M^{HGBR}_{RND}	4.502±0.050[a]	44.744±1.059[a]	6.689±0.079[a]	0.866±0.003[a]
	RFR	M^{RFR}_{RND}	4.532±0.057[b]	46.943±1.229[b]	6.851±0.090[b]	0.860±0.004[b]
	DTR	M^{DTR}_{RND}	5.022±0.061[c]	52.570±1.475[c]	7.250±0.102[c]	0.843±0.005[c]
	MLPR	M^{MLPR}_{RND}	5.206±0.421[d]	52.597±6.720[c]	7.239±0.439[c]	0.843±0.020[c]
	LR	M^{LR}_{RND}	7.158±0.065[e]	86.313±1.956[d]	9.290±0.105[d]	0.742±0.006[d]
	HR	M^{HR}_{RND}	7.526±0.134[f]	96.181±4.836[e]	9.804±0.239[e]	0.713±0.014[e]
	SGDR	M^{SGDR}_{RND}	7.509±0.163[f]	107.509±4.996[f]	10.366±0.238[f]	0.679±0.015[f]

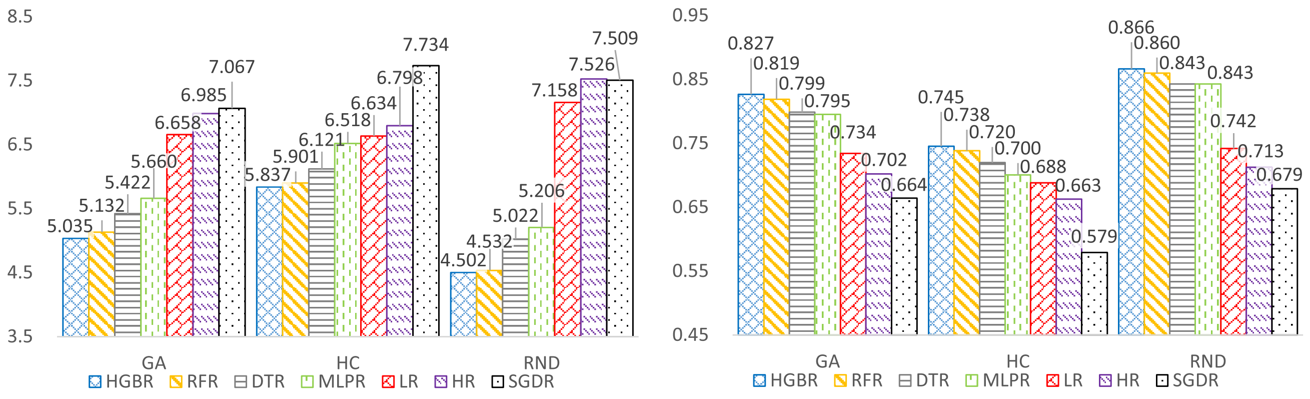
TABLE 5. RQ2. Performance of models with low-complexity of training data in each dataset.

Dataset	Training Data	Model	MAE	MSE	RMSE	R^2
GA	All	M^{All}_{GA}	5.073±0.154[a]	49.499±3.473[a]	7.031±0.246[a]	0.824±0.012[a]
	LC90	M^{LC90}_{GA}	5.330±0.165[b]	56.606±3.192[b]	7.521±0.211[b]	0.799±0.013[b]
	LC95	M^{LC95}_{GA}	5.598±0.171[c]	61.043±3.996[c]	7.809±0.254[c]	0.785±0.014[c]
	LC98	M^{LC98}_{GA}	5.829±0.259[d]	65.027±5.304[d]	8.058±0.318[d]	0.771±0.021[d]
	LC99	M^{LC99}_{GA}	6.041±0.327[d]	69.231±7.125[d]	8.310±0.431[d]	0.754±0.026[d]
HC	All	M^{All}_{HC}	5.907±0.113[a]	65.910±3.421[a]	8.116±0.212[a]	0.739±0.013[a]
	LC90	M^{LC90}_{HC}	6.169±0.197[b]	72.361±4.853[b]	8.502±0.284[b]	0.711±0.018[b]
	LC95	M^{LC95}_{HC}	6.362±0.212[c]	76.929±4.849[c]	8.767±0.275[c]	0.698±0.016[c]
	LC98	M^{LC98}_{HC}	6.645±0.287[d]	83.629±7.502[d]	9.136±0.407[d]	0.663±0.030[d]
	LC99	M^{LC99}_{HC}	6.960±0.372[e]	90.472±9.629[e]	9.499±0.495[e]	0.642±0.036[d]
RND	All	M^{All}_{RND}	4.630±0.139[a]	46.166±2.764[a]	6.792±0.204[a]	0.863±0.009[a]
	LC90	M^{LC90}_{RND}	5.015±0.145[b]	54.831±3.041[b]	7.402±0.205[b]	0.836±0.009[b]
	LC95	M^{LC95}_{RND}	5.216±0.242[c]	57.724±4.879[b]	7.591±0.319[b]	0.827±0.015[b]
	LC98	M^{LC98}_{RND}	5.674±0.324[d]	66.350±6.868[c]	8.135±0.419[c]	0.802±0.023[c]
	LC99	M^{LC99}_{RND}	6.083±0.347[e]	74.541±6.888[d]	8.624±0.408[d]	0.779±0.022[d]

diversity of the training data decreases, following the order of M^{All}, M^{LC90}, M^{LC95}, M^{LC98}, and M^{LC99}.

C. RQ3: PERFORMANCE ANALYSIS BASED ON THE COMPLEXITY LEVEL OF TEST DATA

We investigated whether the prediction performance of models trained on data with low complexity-diversity decreased as the complexity level of the test data increased. If the difference in the complexity levels of the test data reduces the prediction performance of LC models, it could be challenging to accurately predict the testability of complex source codes. To distinguish the complexity levels of the test data, we sorted them by the sum of the CC and SL values. Even if the CC is high, a low SL indicates fewer nested conditions, which



(a) MAE results (b) R^2 results

FIGURE 3. RQ1. Mean MAE and R^2 of regression models in each dataset.

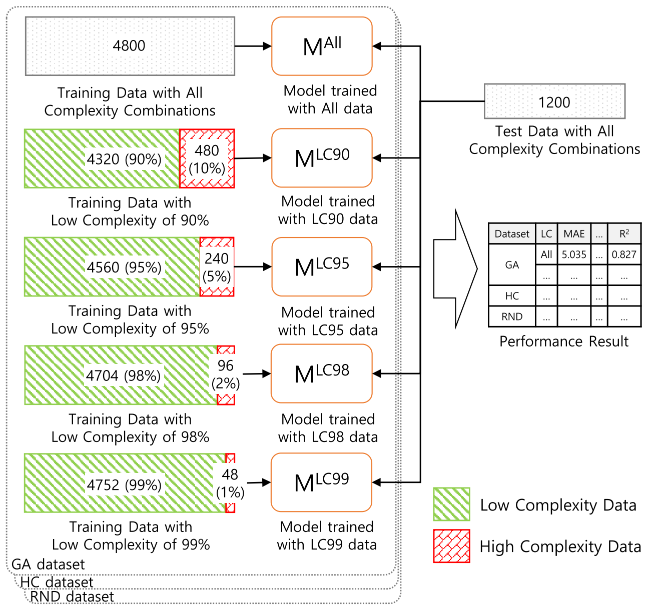

4800
Training Data with All Complexity Combinations
M^{All}
Model trained with All data
4320 (90%)
480 (10%)
Training Data with Low Complexity of 90%
M^{LC90}
Model trained with LC90 data
4560 (95%)
240 (5%)
Training Data with Low Complexity of 95%
M^{LC95}
Model trained with LC95 data
4704 (98%)
96 (2%)
Training Data with Low Complexity of 98%
M^{LC98}
Model trained with LC98 data
4752 (99%)
48 (1%)
Training Data with Low Complexity of 99%
M^{LC99}
Model trained with LC99 data
GA dataset
HC dataset
RND dataset
1200
Test Data with All Complexity Combinations
Dataset LC MAE ... R²
GA All 5.035 ... 0.827
HC
RND
Performance Result
Low Complexity Data
High Complexity Data

FIGURE 4. Experimental process of RQ2.

means that the conditions are relatively easy to cover. Conversely, even if SL is high, a low CC implies that there are fewer conditions to cover. Therefore, we identified a combination of CC and SL as features that influence code coverage, which is a measure of testability effectiveness. The sorted test data are divided into five parts, each representing a different level of complexity. LC models were constructed using RQ2. Fig. 6 illustrates the experimental process for RQ3.

We analyzed the performance using test data with divided complexity levels on models with low complexity-diversity, which were constructed using the same method as in RQ2. To identify test data with different levels of complexity, we sampled 6,000 data points, excluding the training data. The sampled data were then sorted based on the CC+SL values and subsequently divided into five parts. We classified the 1,200 test data points differentiated by complexity level as CL1 for data with the lowest complexity level and CL5 for data with the highest complexity level. The experiment was repeated 30 times because of the randomness of the training and test data caused by sampling.

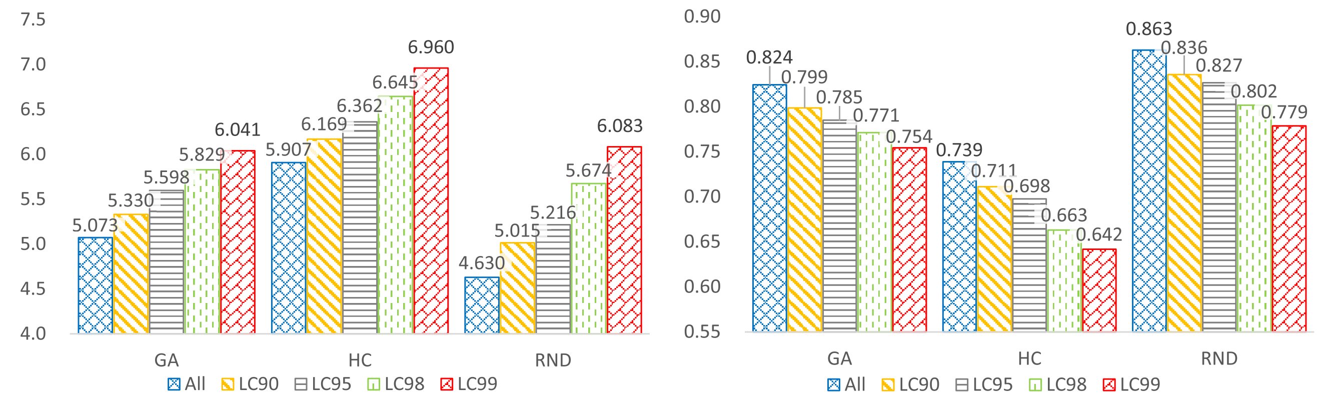
We obtained 25 performance results using five test datasets from five models. The experimental results were analyzed using the MAE, MSE, RMSE, and R^2, and an ANOVA test was performed. The experiment was conducted separately for each dataset, and we subsequently analyzed whether there was a similar tendency across all the datasets. Fig. 7 presents the mean MAE and R^2 results for the GA dataset for RQ3.

The experimental results indicate that M^{All}, which was built using training data without complexity limitations, demonstrated a mean MAE of 4.579 or less and a mean R^2 of 0.866 or higher at all complexity levels of the test data. For the CL1 test data, all models exhibited a mean MAE of 3.916 or less and a mean R^2 of 0.871 or higher. In the CL1 test data, the LC models showed high performance similar to that of M^{All}. This was because the LC models were built using a high proportion of low-complexity training data, similar to the CL1 test data.

All LC models showed a decrease in prediction performance in terms of the mean MAE and R^2 as the complexity level of the test data increased from CL1 to CL4. However, despite the increase in the complexity level of the test data from CL4 to CL5, the MAE decreased slightly for M^{LC98}, and R^2 increased slightly for M^{LC90}, M^{LC95}, and M^{LC98}. Table 6 presents the experimental results for RQ3 in the GA dataset, separated by the proportion of low-complexity data in the training data and the complexity level of the test data.

M^{All} did not exhibit a consistent performance difference based on the complexity level of the test data in all cases. The mean performance consistently decreased from CL1 to CL4 across all LC models. In some LC model results, a reversal of mean performance was observed between CL4 and CL5.

However, in all LC model results, CL2-CL5 always showed a statistically significant lower performance than CL1, and CL4 always showed a statistically significant lower performance than CL2. Furthermore, except for the MSE and RMSE between CL4 and CL5 for M^{LC90} and M^{LC95},



(a) MAE results

(b) R^2 results

FIGURE 5. RQ2. Mean MAE and R^2 of models with low-complexity training data in each dataset.

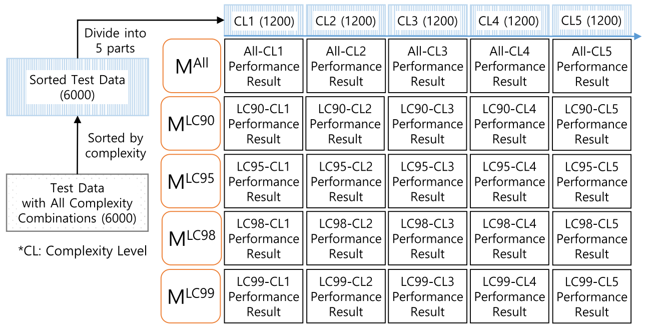

Divide into 5 parts
Sorted Test Data (6000)
Sorted by complexity
Test Data with All Complexity Combinations (6000)
*CL: Complexity Level
CL1 (1200)
CL2 (1200)
CL3 (1200)
CL4 (1200)
CL5 (1200)
M^All
M^LC90
M^LC95
M^LC98
M^LC99
All-CL1 Performance Result
All-CL2 Performance Result
All-CL3 Performance Result
All-CL4 Performance Result
All-CL5 Performance Result
LC90-CL1 Performance Result
LC90-CL2 Performance Result
LC90-CL3 Performance Result
LC90-CL4 Performance Result
LC90-CL5 Performance Result
LC95-CL1 Performance Result
LC95-CL2 Performance Result
LC95-CL3 Performance Result
LC95-CL4 Performance Result
LC95-CL5 Performance Result
LC98-CL1 Performance Result
LC98-CL2 Performance Result
LC98-CL3 Performance Result
LC98-CL4 Performance Result
LC98-CL5 Performance Result
LC99-CL1 Performance Result
LC99-CL2 Performance Result
LC99-CL3 Performance Result
LC99-CL4 Performance Result
LC99-CL5 Performance Result

FIGURE 6. Experimental process of RQ3.

no statistically significant performance improvement was observed between CL4 and CL5. Therefore, despite some exceptions, the prediction performance tends to decrease as the complexity level of the test data increases in the GA dataset. Fig. 8 shows the mean MAE and R^2 results for the HC dataset of RQ3.

The experimental results indicate that M^{All} demonstrates a mean MAE of 5.229 or less and a mean R^2 of 0.736 or higher across all complexity levels of the test data. For the CL1 test data, all models exhibited a mean MAE of 4.458 or less and a mean R^2 of 0.702 or higher, with LC models exhibiting a slightly lower performance in terms of both MAE and R^2 than M^{All}.

The mean MAE consistently increased from CL1 to CL5 across all LC models. The mean R^2 was lower in the order of CL1, CL2, CL4, and CL5 across all LC models. However, for M^{LC90}, M^{LC95}, and M^{LC98}, there were instances in which R^2 for CL3 was slightly higher than that for CL2. Table 7 presents the experimental results for RQ3 for the HC dataset.

Similar to the GA dataset, M^{All} did not exhibit a consistent performance difference based on the complexity level of the test data in all cases. The mean performance consistently decreased in the order of CL1, CL2, CL4, and CL5 across all LC models. In some LC model results, a reversal of mean performance was observed between CL2 and CL3.

However, in all LC model results, CL2-CL5 always showed a statistically significant lower performance than CL1, and CL5 always showed a statistically significant lower performance than CL1-CL4. Furthermore, in all LC models, although there were instances where the performance of the model significantly decreased as the complexity level of the test data increased, no instances of a statistically significant increase were observed. Therefore, for the HC dataset, the prediction performance decreased as the complexity level of the test data increased. Fig. 9 presents the mean MAE and R^2 results for the RND dataset for RQ3.

The mean MAE consistently increased from CL1 to CL5 across all LC models. Similarly, the mean R^2 consistently decreased from CL1 to CL5 across all LC models. The RND dataset consistently demonstrated a decrease in mean performance in terms of MAE and R^2 as the complexity level of the test data increased. Table 8 presents the experimental results for RQ3 for the RND dataset.

In all cases, M^{All} shows statistically significant differences between CL1 and CL2-CL4. However, there was no consistent relationship between CL2-CL4. Across all LC models, the mean performance consistently decreased from CL1-CL5, except for M^{LC90} and M^{LC98}, and between CL4 and CL5 in terms of MSE and RMSE.

Furthermore, except for the MAE, MSE, and RMSE between CL4 and CL5 for all LC models, statistically significant performance increases were observed in all cases as the complexity level of the test data increased. Therefore, for the RND dataset, the prediction performance decreases as the complexity level of the test data increases.

D. RQ4: PERFORMANCE ANALYSIS BASED ON THE DIFFERENCE IN COMPLEXITY LEVELS OF TRAINING AND TEST DATA

We investigated whether the larger the difference in the complexity levels between the training and test data, the greater the difference in the prediction performance of the model. If the difference in complexity levels between the training and test data linearly decreases the prediction performance, it can be expected that the lower the complexity diversity of the training data, the greater the decrease in the prediction performance for high-complexity source code.

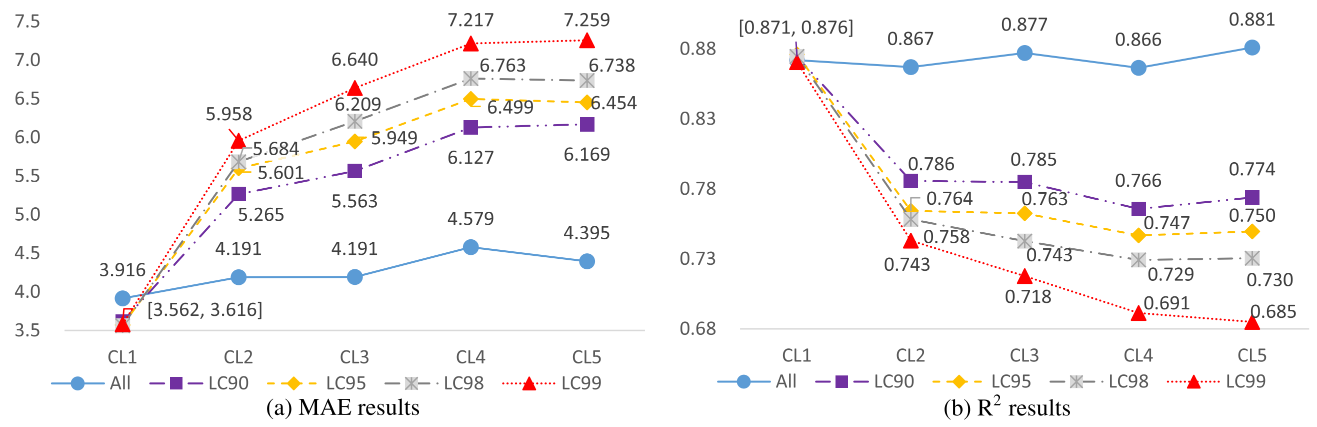


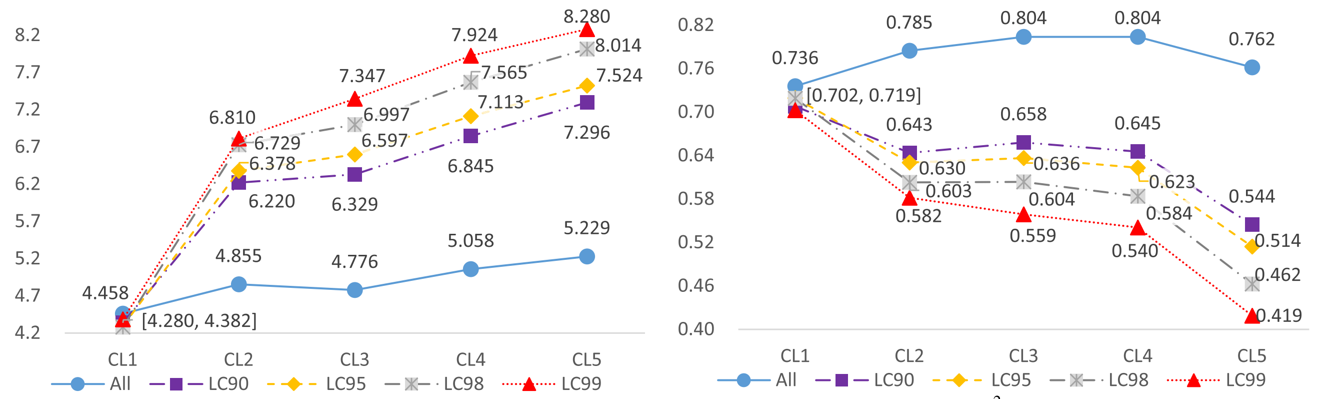
FIGURE 7. RQ3. Mean MAE and R^2 of models with the different complexity levels of test data in GA dataset.

TABLE 6. RQ3. Performance of models with the different complexity levels of test data in GA dataset.

Training Data	Model	Test Data	MAE	MSE	RMSE	R^2
All	M_{GA}^{All}	CL1	3.916±0.360^{a}	33.109±5.679^{a}	5.734±0.490^{a}	0.872±0.023^{a}
		CL2	4.191±0.393^{b}	35.884±6.250^{ab}	5.969±0.513^{ab}	0.867±0.023^{a}
		CL3	4.191±0.333^{b}	33.703±4.814^{a}	5.791±0.412^{a}	0.877±0.017^{a}
		CL4	4.579±0.403^{c}	39.559±6.438^{b}	6.270±0.504^{b}	0.866±0.022^{a}
		CL5	4.395±0.383^{bc}	33.256±5.590^{a}	5.748±0.478^{a}	0.881±0.022^{a}
LC90	M_{GA}^{LC90}	CL1	3.616±0.117^{a}	32.760±2.112^{a}	5.721±0.184^{a}	0.874±0.009^{a}
		CL2	5.265±0.182^{b}	57.330±4.340^{b}	7.566±0.286^{b}	0.786±0.016^{b}
		CL3	5.563±0.199^{c}	58.930±4.286^{b}	7.672±0.277^{b}	0.785±0.015^{b}
		CL4	6.127±0.207^{d}	69.071±4.725^{d}	8.306±0.284^{d}	0.766±0.017^{c}
		CL5	6.169±0.291^{d}	64.076±5.318^{c}	7.998±0.326^{c}	0.774±0.021^{bc}
LC95	M_{GA}^{LC95}	CL1	3.571±0.152^{a}	32.315±3.385^{a}	5.677±0.292^{a}	0.876±0.011^{a}
		CL2	5.601±0.258^{b}	63.051±5.585^{b}	7.933±0.350^{b}	0.764±0.019^{b}
		CL3	5.949±0.280^{c}	64.964±4.545^{b}	8.055±0.280^{b}	0.763±0.016^{b}
		CL4	6.499±0.196^{d}	74.673±4.615^{d}	8.637±0.268^{d}	0.747±0.018^{c}
		CL5	6.454±0.248^{d}	69.507±5.194^{c}	8.332±0.308^{c}	0.750±0.019^{c}
LC98	M_{GA}^{LC98}	CL1	3.562±0.133^{a}	32.411±2.924^{a}	5.688±0.252^{a}	0.875±0.011^{a}
		CL2	5.684±0.235^{b}	64.256±5.016^{b}	8.010±0.313^{b}	0.758±0.020^{b}
		CL3	6.209±0.355^{c}	70.384±6.938^{c}	8.380±0.415^{c}	0.743±0.027^{bc}
		CL4	6.763±0.386^{d}	79.973±8.613^{d}	8.931±0.464^{d}	0.729±0.027^{c}
		CL5	6.738±0.478^{d}	75.415±10.38^{cd}	8.666±0.567^{cd}	0.730±0.036^{c}
LC99	M_{GA}^{LC99}	CL1	3.582±0.173^{a}	33.574±3.864^{a}	5.785±0.332^{a}	0.871±0.014^{a}
		CL2	5.958±0.341^{b}	69.301±6.808^{b}	8.315±0.402^{b}	0.743±0.026^{b}
		CL3	6.640±0.561^{c}	77.492±10.783^{c}	8.783±0.606^{c}	0.718±0.038^{c}
		CL4	7.217±0.644^{d}	91.385±14.300^{d}	9.532±0.739^{d}	0.691±0.050^{c}
		CL5	7.259±0.826^{d}	88.062±19.532^{cd}	9.333±0.991^{cd}	0.685±0.071^{c}

To measure the difference in complexity levels between the training and test data, we employed the concept of effect size, which measures the difference between the means of the two groups relative to the standard deviation of the data. Specifically, we use Cohen's d [48], a representative method for computing the effect size, which is calculated as the difference between the means divided by the pooled standard deviation of the data.

To calculate the difference in complexity levels, we used the model and test data from RQ3. The experiments were conducted using 25 combinations of five distinct models and five CL test data. Each combination was repeated 30 times to reduce the random sampling effects. Thus, the experiment was conducted 750 times, and the difference in complexity levels between the training and test data was calculated each time. We use the CC+SL complexity level indicator, which is



(a) MAE results (b) R^2 results

FIGURE 8. RQ3. Mean MAE and R^2 of models with the different complexity levels of test data in HC dataset.

TABLE 7. RQ3. Performance of models with the different complexity levels of test data in HC dataset.

Training Data	Model	Test Data	MAE	MSE	RMSE	R^2
All	M_{HC}^{All}	CL1	4.458 ± 0.400^a	46.050 ± 7.422^a	6.766 ± 0.533^a	0.736 ± 0.042^a
		CL2	4.855 ± 0.382^{bc}	45.577 ± 7.625^a	6.730 ± 0.547^a	0.785 ± 0.037^{bc}
		CL3	4.776 ± 0.329^b	41.939 ± 5.854^a	6.462 ± 0.439^a	0.804 ± 0.027^c
		CL4	5.058 ± 0.388^{cd}	45.429 ± 6.936^a	6.722 ± 0.497^a	0.804 ± 0.030^c
		CL5	5.229 ± 0.420^d	45.731 ± 7.077^a	6.744 ± 0.508^a	0.762 ± 0.038^{ab}
LC90	M_{HC}^{LC90}	CL1	4.336 ± 0.204^a	50.550 ± 5.940^a	7.098 ± 0.417^a	0.708 ± 0.032^a
		CL2	6.220 ± 0.249^b	75.253 ± 5.781^b	8.669 ± 0.330^b	0.643 ± 0.025^b
		CL3	6.329 ± 0.228^b	73.338 ± 4.736^b	8.559 ± 0.275^b	0.658 ± 0.020^b
		CL4	6.845 ± 0.231^c	81.196 ± 4.785^c	9.007 ± 0.265^c	0.645 ± 0.022^b
		CL5	7.296 ± 0.203^d	87.041 ± 4.416^d	9.327 ± 0.237^d	0.544 ± 0.029^c
LC95	M_{HC}^{LC95}	CL1	4.293 ± 0.163^a	49.633 ± 4.781^a	7.037 ± 0.343^a	0.719 ± 0.023^a
		CL2	6.378 ± 0.208^b	78.917 ± 5.251^b	8.879 ± 0.297^b	0.630 ± 0.023^b
		CL3	6.597 ± 0.266^c	78.372 ± 6.111^b	8.846 ± 0.344^b	0.636 ± 0.029^b
		CL4	7.113 ± 0.289^d	86.879 ± 6.699^c	9.314 ± 0.359^c	0.623 ± 0.027^b
		CL5	7.524 ± 0.315^e	92.267 ± 6.812^d	9.599 ± 0.355^d	0.514 ± 0.035^c
LC98	M_{HC}^{LC98}	CL1	4.280 ± 0.172^a	49.367 ± 4.616^a	7.019 ± 0.332^a	0.719 ± 0.026^a
		CL2	6.729 ± 0.340^b	85.491 ± 6.412^b	9.240 ± 0.343^b	0.603 ± 0.026^b
		CL3	6.997 ± 0.376^c	85.257 ± 6.856^b	9.226 ± 0.369^b	0.604 ± 0.030^b
		CL4	7.565 ± 0.436^d	96.137 ± 9.532^c	9.793 ± 0.486^c	0.584 ± 0.040^b
		CL5	8.014 ± 0.424^e	104.147 ± 10.122^d	10.193 ± 0.498^d	0.462 ± 0.047^c
LC99	M_{HC}^{LC99}	CL1	4.382 ± 0.254^a	51.786 ± 5.074^a	7.188 ± 0.351^a	0.702 ± 0.031^a
		CL2	6.810 ± 0.329^b	88.595 ± 7.552^b	9.404 ± 0.401^b	0.582 ± 0.037^b
		CL3	7.347 ± 0.329^c	94.305 ± 7.629^c	9.704 ± 0.390^c	0.559 ± 0.037^{bc}
		CL4	7.924 ± 0.487^d	106.488 ± 13.642^d	10.300 ± 0.649^d	0.540 ± 0.052^c
		CL5	8.280 ± 0.587^d	111.479 ± 15.435^d	10.535 ± 0.709^d	0.419 ± 0.082^d

consistent with RQ3. Scatter plots were drawn to analyze the relationship between the complexity-level difference and R^2. Fig. 10 presents a scatter plot of the differences in complexity levels between the training and test data and R^2.

It can be observed that in all the datasets, there is a tendency for R^2 to decrease as Cohen's d increases. To quantitatively measure the relationship between Cohen's d and the model performance, we conducted a Pearson correlation analysis. Table 9 presents the Pearson correlation coefficient between Cohen's d and the model performance. All correlation coefficients had p-values < 0.05.

The experimental results showed that the absolute value of the correlation coefficient in all cases was 0.804 or higher. Assuming a reasonably sized dataset, a correlation

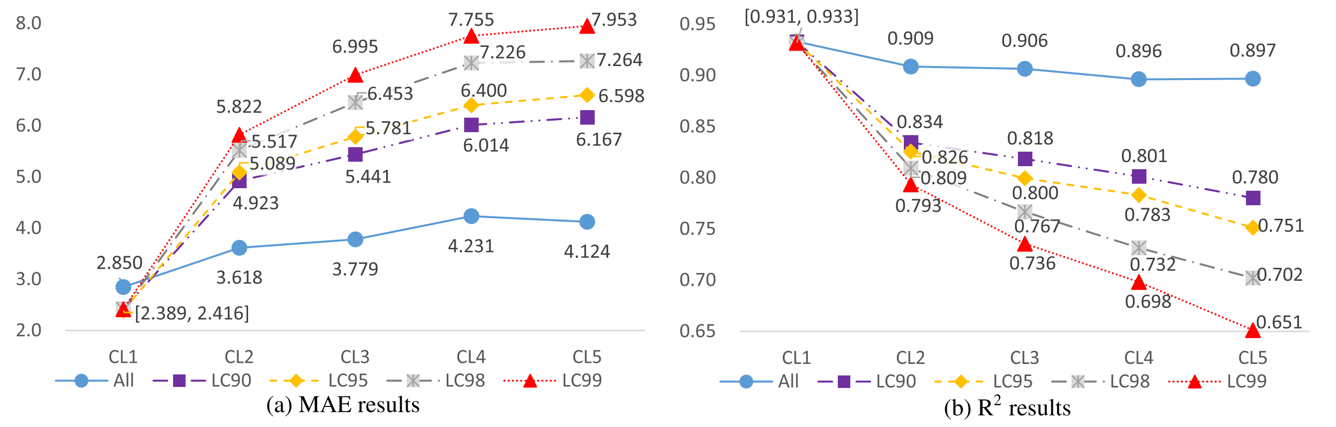


FIGURE 9. RQ3. Mean MAE and R^2 of models with the different complexity levels of test data in RND dataset.

TABLE 8. RQ3. Performance of models with the different complexity levels of test data in RND dataset.

Training Data	Model	Test Data	MAE	MSE	RMSE	R^2
		CL1	2.850 ± 0.304^{a}	21.215 ± 3.677^{a}	4.589 ± 0.407^{a}	0.933 ± 0.011^{a}
		CL2	3.618 ± 0.313^{b}	29.232 ± 4.890^{b}	5.388 ± 0.453^{b}	0.909 ± 0.016^{b}
All	M_{RND}^{All}	CL3	3.779 ± 0.312^{b}	30.920 ± 4.721^{b}	5.545 ± 0.421^{b}	0.906 ± 0.014^{bc}
		CL4	4.231 ± 0.359^{c}	35.966 ± 5.732^{c}	5.979 ± 0.476^{c}	0.896 ± 0.016^{c}
		CL5	4.124 ± 0.400^{c}	31.306 ± 5.753^{b}	5.572 ± 0.521^{b}	0.897 ± 0.019^{bc}
		CL1	2.412 ± 0.124^{a}	21.281 ± 2.721^{a}	4.604 ± 0.294^{a}	0.933 ± 0.009^{a}
		CL2	4.923 ± 0.221^{b}	53.079 ± 4.870^{b}	7.278 ± 0.331^{b}	0.834 ± 0.014^{b}
LC90	M_{RND}^{LC90}	CL3	5.441 ± 0.188^{c}	59.951 ± 4.293^{c}	7.738 ± 0.277^{c}	0.818 ± 0.014^{c}
		CL4	6.014 ± 0.183^{d}	69.055 ± 4.499^{d}	8.306 ± 0.271^{d}	0.801 ± 0.013^{d}
		CL5	6.167 ± 0.225^{e}	67.668 ± 4.843^{d}	8.221 ± 0.294^{d}	0.780 ± 0.017^{e}
		CL1	2.389 ± 0.156^{a}	21.367 ± 3.251^{a}	4.609 ± 0.354^{a}	0.933 ± 0.010^{a}
		CL2	5.089 ± 0.270^{b}	55.812 ± 4.408^{b}	7.465 ± 0.290^{b}	0.826 ± 0.016^{b}
LC95	M_{RND}^{LC95}	CL3	5.781 ± 0.229^{c}	65.555 ± 4.418^{c}	8.092 ± 0.275^{c}	0.800 ± 0.014^{c}
		CL4	6.400 ± 0.210^{d}	75.292 ± 4.716^{d}	8.673 ± 0.273^{d}	0.783 ± 0.015^{d}
		CL5	6.598 ± 0.209^{e}	75.454 ± 4.519^{d}	8.683 ± 0.261^{d}	0.751 ± 0.018^{e}
		CL1	2.409 ± 0.135^{a}	21.589 ± 1.914^{a}	4.642 ± 0.205^{a}	0.931 ± 0.007^{a}
		CL2	5.517 ± 0.283^{b}	61.162 ± 5.071^{b}	7.814 ± 0.321^{b}	0.809 ± 0.014^{b}
LC98	M_{RND}^{LC98}	CL3	6.453 ± 0.409^{c}	76.155 ± 8.339^{c}	8.714 ± 0.476^{c}	0.767 ± 0.028^{c}
		CL4	7.226 ± 0.451^{d}	92.053 ± 10.298^{d}	9.580 ± 0.529^{d}	0.732 ± 0.029^{d}
		CL5	7.264 ± 0.494^{d}	90.319 ± 11.262^{d}	9.486 ± 0.594^{d}	0.702 ± 0.040^{e}
		CL1	2.416 ± 0.163^{a}	21.573 ± 2.508^{a}	4.637 ± 0.271^{a}	0.932 ± 0.008^{a}
		CL2	5.822 ± 0.349^{b}	66.140 ± 6.360^{b}	8.124 ± 0.386^{b}	0.793 ± 0.019^{b}
LC99	M_{RND}^{LC99}	CL3	6.995 ± 0.596^{c}	87.267 ± 12.195^{c}	9.321 ± 0.638^{c}	0.736 ± 0.037^{c}
		CL4	7.755 ± 0.776^{d}	104.458 ± 19.271^{d}	10.182 ± 0.896^{d}	0.698 ± 0.056^{d}
		CL5	7.953 ± 0.897^{d}	106.012 ± 21.239^{d}	10.249 ± 1.000^{d}	0.651 ± 0.068^{e}

value of less than 0.1 is trivial, 0.1-0.3 is minor, 0.3-0.5 is moderate, 0.5-0.7 is large, 0.7-0.9 is very large, and 0.9-1 is almost perfect [49]. MAE, MSE, and RMSE are error metrics; the larger their values, the lower the prediction performance. Therefore, a positive correlation implies that the greater the difference in complexity, the lower the prediction performance. R^2 denotes the model's explanatory power. The smaller the value, the lower the prediction performance. Hence, a negative correlation indicates that the larger the difference in complexity, the lower the prediction performance. Based on the experimental results, we discovered a strong relationship between the prediction performance of the model

TABLE 9. RQ4. Correlation between Cohen'd and performance.

Dataset	MAE	MSE	RMSE	R^2
GA	0.884	0.827	0.830	-0.804
HC	0.907	0.854	0.846	-0.839
RND	0.902	0.884	0.882	-0.900

and the difference in complexity levels between the training and test data.

V. DISCUSSION AND LIMITATIONS

This section describes the discussion on the experimental results and the threats to validity. In the discussion, we summarize the experimental results for each RQ, analyze the implications of these results, and suggest possibilities for expanding research on testability prediction. In the threats to validity, we analyze the possible threats that might affect the validity of the results.

A. DISCUSSION

We summarize the experimental results by averaging the outcomes across all datasets. Fig. 11 presents the mean results across all datasets for each RQ. The RQ1 charts depict the average MAE and R^2 for each regression algorithm. For RQ2, the charts display the average MAE and R^2 based on the complexity diversity of the training data. The RQ3 charts show the average MAE and R^2 across all LC models, categorized by the complexity level of the test data. Lastly, the RQ4 chart exhibits the absolute value of the correlations between the difference in complexity levels of the training and test data and the MAE and R^2.

In RQ1, the regression algorithm with the highest mean performance was HGBR, which showed an MAE of 5.125 and an R^2 of 0.813. Tree-based HGBR, RFR, and DTR exhibited better mean performances than the neural-network-based MLPR and linear-based LR, HR, and SGDR. HGBR showed an MAE that was 11.6% lower and an R^2 that was 4.3% higher than MLPR. Compared to linear-based algorithms, HGBR showed an MAE that was 24.8% to 31.1% lower and an R^2 that was 12.6% to 26.9% higher.

In RQ2, the performance of the models consistently decreased as the complexity diversity of the training data decreased. Compared to All, the MAE of LC90-LC99 was respectively 5.8%, 10.0%, 16.3%, and 22.3% higher, and the R^2 was respectively 3.3%, 4.8%, 7.8%, and 10.4% lower.

In RQ3, the performance of the models consistently decreased as the complexity level of the test data increased. CL1 showed significantly higher performance than CL2-CL5, as it has a similar proportion of low-complexity training data used in the construction of the LC models. Compared to CL2-CL5, the MAE of CL1 was 57.5% to 89.6% lower, and the R^2 was 10.2% to 18.3% higher. There was a smaller performance difference between CL2-CL5, which have a dissimilar proportion of low-complexity data compared to LC models, than with CL1. Compared to CL2, the MAE of CL3-CL5 was respectively 7.7%, 17.7%, and 20.3% higher, and the R^2 was respectively 1.4%, 3.8%, and 9.1% lower.

In RQ4, the correlation between the difference in complexity between training and test data and the MAE and R^2 were 0.898 and −0.848, respectively. The positive correlation in MAE and the negative correlation in R^2 both imply the model's performance decreases as the complexity difference increases. In the chart, we used the absolute values of the correlation coefficients to focus on their magnitudes.

The experimental results indicate the following implications. The high performance of the HGBR model provides accurate predictions of testability, suggesting that it can help developers proactively identify software components that require significant testing effort, plan testing activities, and recognize the need for refactoring to reduce the test effort. The findings of this study highlight the importance of considering complexity diversity in the testability prediction for C programs. The decrease in model performance as the complexity diversity of the training data decreases, the variation in model performance based on the complexity level of the test data, and the correlation between model performance and complexity difference demonstrate the impact of complexity diversity in testability prediction. These findings enhance the understanding of the impact of complexity diversity on testability prediction, an aspect not thoroughly examined in previous studies. These insights can assist test engineers in improving the performance of testability prediction models.

Testability is a multifaceted issue that depends on source code, design patterns, software architecture, process complexity, domain characteristics, programming language, and even the programmer's experience. However, source code is the most concrete object for testability evaluation, and collecting and measuring source code-based metrics is relatively easy. Therefore, we focused our research on source code, but the study can be expanded by considering other factors that affect testability.

Firstly, the study can be expanded by using test patterns that consider domain characteristics. By utilizing test patterns that consider domain-specific behavior and types of defects, a testing process can be made more systematic and effective. Therefore, test pattern-based metrics such as the number of applied test patterns and the complexity of applied test patterns can be used as variables for evaluating testability in terms of testing effectiveness.

For example, Siddiqui and Khan [50] proposed test patterns for cloud applications. They proposed a structure of test patterns and methods for identifying applicable patterns through feature analysis of the test patterns. The test patterns include the test situation, test target, and sequence of actions needed to perform the test. Based on this study, the number of applicable test patterns, the number of applied test patterns, and the complexity of applied test patterns can be used in testability prediction research. And Górski [51] proposed a test pattern for smart contracts. He proposed a test pattern that considers symmetric characteristics based on verification rules for smart contracts. Based on this study, the number of

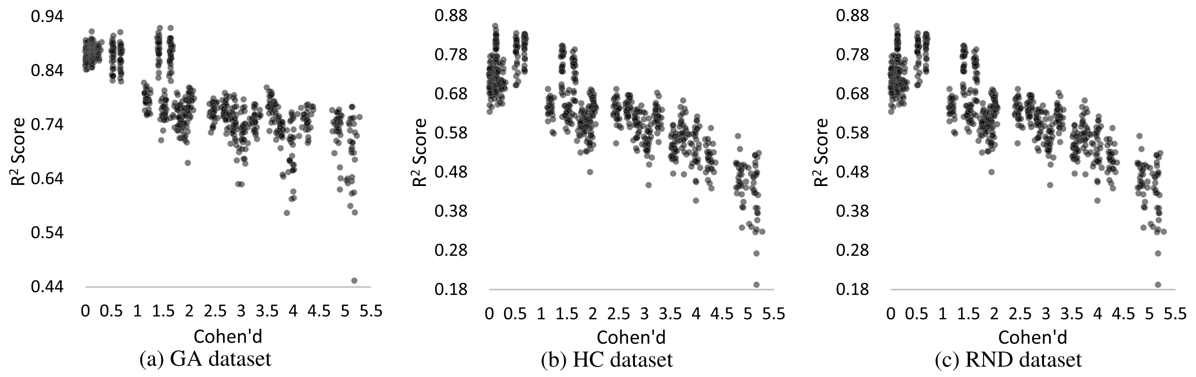


FIGURE 10. RQ4. Scatter plot of the difference in complexity levels between training and test data and R^2.

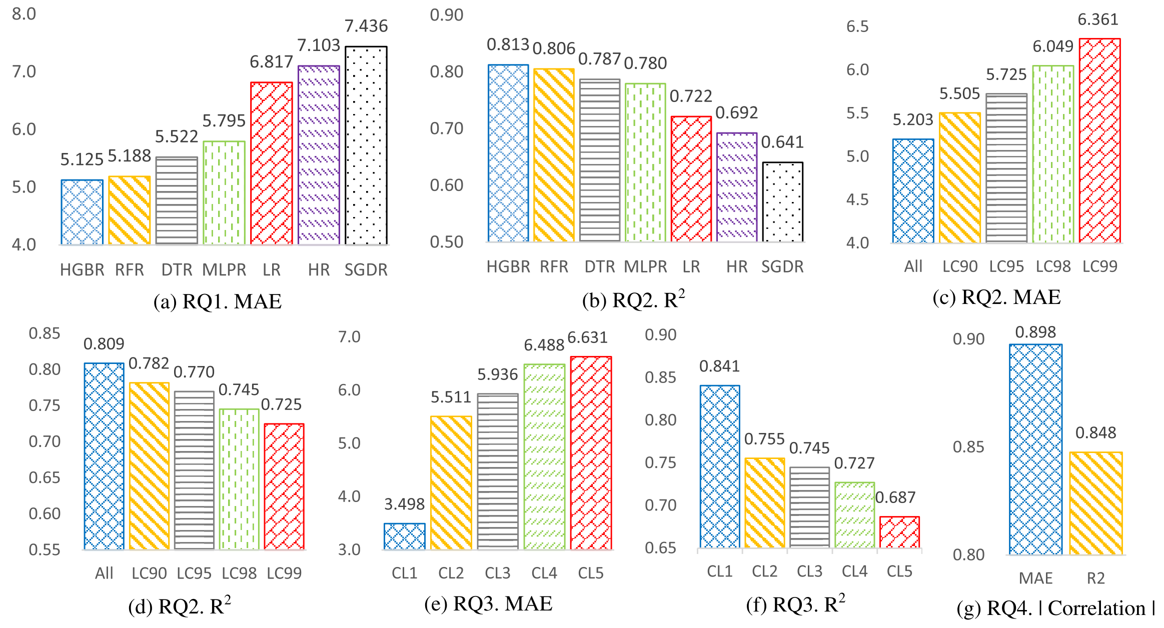


FIGURE 11. Summary of experimental results.

necessary test cases and the number of performed test cases can be used in testability prediction research.

Secondly, the study can be expanded to other languages. By considering the structural characteristics and types of defects according to the programming language, the effectiveness of testing can be enhanced. We discuss expanding the study to Rust and JavaScript.

Rust is a systems programming language focused on performance, type safety, and concurrency. Rust ensures safety by providing ownership and borrowing, memory management methods that check for memory leaks or invalid references at compile time and prevent race conditions. Considering these language-specific characteristics, metrics such as the number of ownership transfers and the number of borrow checks can be used for testability prediction in Rust programs.

JavaScript is a scripting language for web development. JavaScript supports implicit global, making it easy to use global variables, and callbacks are widely used for programming in asynchronous web environments. Considering these language-specific characteristics, metrics such as the number used of global variables, the number of callbacks, and the nested callback depth can be used for testability prediction in JavaScript programs.

B. THREATS TO VALIDITY

Threats to internal validity include the selection of parameters used in both the generation of test data and the construction of the model, as well as the reduction of randomness in these processes. We used parameter values from previous studies [31], [32] to generate the test data. To obtain reasonable statistical power for the test data generation, we repeated the generation ten times [33]. For the model construction, we tuned the optimal parameters using GridSearchCV based on the hyperparameter range of existing studies [11], [12], [13]. The experiments were repeated 30 times to reduce the randomness caused by sampling.

Threats to construct validity include obtaining the appropriate tools for the experiment. We developed tools for test data generation and SUT generation to collect training data with high complexity-diversity. Both tools were developed in Java, and we used the JavaCC C parser available on Java.net for the C source code analysis. The test data generation tool developed is available from Zenodo.[1] We automatically generate SUTs with feasible combinations of metric ranges based on C/C++ industry standards. Because the generated SUTs may contain infeasible branches, we use Joggie [27], a tool for detecting infeasible code, to remove infeasible branches. Joggie used a Principle solver [28] to assess its feasibility. The Princess solver, a tool that checks whether the conditions are satisfiable, is an essential feature in infeasible code detection. The Princess solver won the TFA division (arithmetic problems) in the 2012 CADE ATP System Competition, a yearly competition for fully automated theorem provers, and the TFI category (integer problems), and was runner-up in the TFA division in 2013 and 2014. Furthermore, the Princess solver is used in JavaSMT [29], a unifying Java interface for SMT solvers, and Eldarica [30], a predicate-abstraction-based model checker.

Other threats include the methods of metric collection, model construction, and statistical analysis. For the metric collection, we used the commercial metric analysis tools Understand 6.2 [26] and PC-lint Plus 2.0 [36]. One of the metrics used, NOEO, was measured using a tool developed to count the numbers of '==' and '!=' operators within a function. We verified the correctness of the measured NOEO by inspecting a subset of the C source codes. We used scikit-learn [44], a Python machine-learning library, for model construction. For statistical analyses, such as ANOVA and correlation analysis, we used IBM SPSS Statistics 27.

Threats to external validity include obtaining large-scale data with high complexity-diversity to generalize the results. Based on the metric upper limits of the MISRA [20], JPL [21], and JSF [22] standards, we generated ten SUTs for each of the 3,320 feasible metric combinations, totaling 33,200 SUTs.

We performed a statistical analysis of the experimental results to ensure the validity of our conclusions. We analyzed not only the mean and SD but also whether there was a statistically significant difference using the ANOVA test.

VI. CONCLUSION

In this study, we developed a testability prediction model for C programs and investigated the impact of the complexity diversity of training and test data on testability prediction performance. We built a model to predict branch coverage, which is a measure of testability effectiveness, using training data with high complexity-diversity and analyzed the prediction performance. To confirm the importance of complexity diversity in testability prediction models, we observed performance differences according to the complexity levels of the training and test data. For the experiment, we generated 33,200 SUTs with high complexity-diversity and measured their branch coverage using search-based test data generation. We collected nine metrics affecting branch coverage achievement. We built a testability prediction model through regression analysis using branch coverage as the dependent variable and these metrics as independent variables.

The regression algorithm HGBR, which had the best performance metric value, showed a mean R^2 of 0.813. Through ANOVA, we observed a statistically significant decrease in the performance of the models across all datasets when the complexity diversity of the training data was low. Compared to All, the mean R^2 of LC90-LC99 was respectively 3.3%, 4.8%, 7.8%, and 10.4% lower. In addition, we confirmed that the performance of the model trained with low complexity-diversity data decreased as the complexity of the test data increased. Compared to CL2, the mean R^2 of CL3-CL5 was respectively 1.4%, 3.8%, and 9.1% lower. Finally, we observed a strong mean correlation of 0.848 or higher between the difference in the complexity levels of the training and test data and the performance of the prediction model. Our research assists developers in focusing their testing efforts efficiently and highlights the necessity for test engineers to use training data with complexity diversity. This approach can improve the training process of the testability prediction model for C programs.

In future work, we plan to expand our study by using methods that reflect domain characteristics such as test patterns, and by using language-specific metrics tailored to other languages such as Rust and JavaScript. By using test patterns that consider domain-specific behavior and types of defects in testability predictions, the impact of domain characteristics can be reflected. In addition, by considering language-specific features such as memory management and asynchronous processing methods in testability predictions, the impact of language characteristics can be reflected.

[1]https://zenodo.org/record/7935356#.ZGGjvnZByUl

REFERENCES

[1] B. Beizer, *Software Testing Techniques*, 2nd ed. New York, NY, USA: International Thomson Computer Press, 1990.

[2] *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, Standard ISO/IEC 25010:2011, 2011.

[3] V. Gupta, K. K. Aggarwal, and Y. Singh, "A fuzzy approach for integrated measure of object-oriented software testability," *J. Comput. Sci.*, vol. 1, no. 2, pp. 276–282, Feb. 2005, doi: 10.3844/jcssp.2005.276.282.

[4] M. Bruntink and A. van Deursen, "An empirical study into class testability," *J. Syst. Softw.*, vol. 79, no. 9, pp. 1219–1232, Sep. 2006, doi: 10.1016/j.jss.2006.02.036.

[5] Y. Singh and A. Saha, "Prediction of testability using the design metrics for object–oriented software," *Int. J. Comput. Appl. Technol.*, vol. 44, no. 1, pp. 12–22, 2012, doi: 10.1504/IJCAT.2012.048204.

[6] M. Badri and F. Toure, "Empirical analysis of object-oriented design metrics for predicting unit testing effort of classes," *J. Softw. Eng. Appl.*, vol. 5, no. 7, pp. 513–526, 2012, doi: 10.4236/jsea.2012.57060.

[7] F. Toure and M. Badri, "Prioritizing unit testing effort using software metrics and machine learning classifiers (S)," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2018, pp. 653–658, doi: 10.18293/SEKE2018-146.

[8] W. Albattah, "Software package testability prediction using object-oriented cohesion metrics," in *Proc. 13th Int. Conf. Inf. Commun. Syst. (ICICS)*, Jun. 2022, pp. 155–161, doi: 10.1109/ICICS55353.2022.9811192.

[9] V. Terragni, P. Salza, and M. Pezzè, "Measuring software testability modulo test quality," in *Proc. 28th Int. Conf. Program Comprehension*, Jul. 2020, pp. 241–251, doi: 10.1145/3387904.3389273.

[10] A. O. Bajeh, O. J. Oluwatosin, S. Basri, A. G. Akintola, and A. O. Balogun, "Object-oriented measures as testability indicators: An empirical study," *J. Eng. Sci. Technol.*, vol. 15, no. 2, pp. 1092–1108, 2020.

[11] G. Grano, T. V. Titov, S. Panichella, and H. C. Gall, "Branch coverage prediction in automated testing," *J. Softw., Evol. Process*, vol. 31, no. 9, pp. 1–18, Sep. 2019, doi: 10.1002/smr.2158.

[12] M. Zakeri-Nasrabadi and S. Parsa, "Learning to predict test effectiveness," *Int. J. Intell. Syst.*, vol. 37, no. 8, pp. 4363–4392, Aug. 2022, doi: 10.1002/int.22722.
[13] M. Zakeri-Nasrabadi and S. Parsa, "An ensemble meta-estimator to predict source code testability," *Appl. Soft Comput.*, vol. 129, Nov. 2022, Art. no. 109562, doi: 10.1016/j.asoc.2022.109562.
[14] *Road vehicles—Functional safety*, Standard ISO 26262, 2009.
[15] *Medical Electrical Equipment*, Standard IEC 60601, 1977.
[16] *Railway Applications—Communications, Signalling and Processing Systems—Safety Related Electronic Systems for Signalling*, Standard EN 50129, 2003.
[17] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems*, Standard IEC 61508, 2010.
[18] *Software Considerations in Airborne Systems and Equipment Certification*, Standard DO-178C, 2011.
[19] G. Fraser and A. Arcuri, "A large-scale evaluation of automated unit test generation using EvoSuite," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 2, pp. 1–42, Dec. 2014, doi: 10.1145/2685612.
[20] *The Motor Industry Software Reliability Association, Report 5: Software Metrics*, MISRA, 1995.
[21] B. Kernighan, *JPL Institutional Coding Standard for the C Programming Language*, document JPL DOCID D-60411, California Insititute of Technology, 2009.
[22] *Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program*, Lockheed Martin, 2005.
[23] V. Garousi, M. Felderer, and F. N. Kılıçaslan, "A survey on software testability," *Inf. Softw. Technol.*, vol. 108, pp. 35–64, Apr. 2019, doi: 10.1016/j.infsof.2018.12.003.
[24] *Information Technology—Software Product Evaluation—Quality Characteristics and Guidelines for Their Use*, Standard ISO/IEC 9126, 1991.
[25] P. Wang, Y. Yu, and X. Li, "Testability modeling and test point optimization method of multi-state system," *IEEE Access*, vol. 8, pp. 155011–155019, 2020, doi: 10.1109/ACCESS.2020.3018693.
[26] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution," in *Software Evolution*. Berlin, Germany: Springer, 2008, pp. 37–67.
[27] S. Arlt and M. Schäf, "Joogie: Infeasible code detection for Java," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2012, pp. 767–773, doi: 10.1007/978-3-642-31424-7_62.
[28] P. Rümmer, "E-matching with free variables," in *Proc. Int. Conf. Logic Program. Artif. Intell. Reasoning (LPAR)*, 2012, pp. 359–374, doi: 10.1007/978-3-642-28717-6_28.
[29] D. Baier, D. Beyer, and K. Friedberger, "JavaSMT 3: Interacting with SMT solvers in Java," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2021, pp. 195–208, doi: 10.1007/978-3-030-81688-9_9.
[30] J. Leroux, P. Rümmer, and P. Subotić, "Guiding Craig interpolation with domain-specific abstractions," *Acta Inf.*, vol. 53, no. 4, pp. 387–424, Jun. 2016, doi: 10.1007/s00236-015-0236-z.
[31] K. Lakhotia, M. Harman, and H. Gross, "AUSTIN: An open source tool for search based software testing of c programs," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 112–125, Jan. 2013, doi: 10.1016/j.infsof.2012.03.009.
[32] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 453–477, Mar. 2012, doi: 10.1109/TSE.2011.18.
[33] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 742–762, Nov. 2010, doi: 10.1109/TSE.2009.52.
[34] B. A. Nejmeh, "NPATH: A measure of execution path complexity and its applications," *Commun. ACM*, vol. 31, no. 2, pp. 188–200, Feb. 1988, doi: 10.1145/42372.42379.
[35] M. H. Halstead, *Elements of Software Science* (Operating and Programming Systems Series). Amsterdam, The Netherlands: Elsevier, 1977.
[36] J. Zyzyck, "A report generator for PC-lint," *Dobb's J. Softw. Tools Prof. Program.*, vol. 28, no. 2, pp. 52–55, 2003.
[37] A. Guryanov, "Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees," in *Proc. Int. Conf. Anal. Images, Social Netw. Texts (AIST)*, 2019, pp. 39–50, doi: 10.1007/978-3-030-37334-4_4.
[38] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, "Random forest: A classification and regression tool for compound classification and QSAR modeling," *J. Chem. Inf. Comput. Sci.*, vol. 43, no. 6, pp. 1947–1958, Nov. 2003, doi: 10.1021/ci034160g.
[39] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Tree*. Boca Raton, FL, USA: CRC Press, 1984, doi: 10.1201/9781315139470.
[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362, doi: 10.7551/mitpress/5236.001.0001.
[41] X. Yan and X. G. Su, *Linear Regression Analysis: Theory and Computing*. Singapore: World Scientific, 2009, doi: 10.1142/6986.
[42] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemp. Math.*, vol. 443, no. 7, pp. 59–72, 2007, doi: 10.1090/conm/443/08555.
[43] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2004, p. 116, doi: 10.1145/1015330.1015332.
[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/1953048.2078195
[45] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. New York, NY, USA: Springer, 2013, doi: 10.1007/978-1-4614-6849-3.
[46] H. Levene, "Robust tests for equality of variances," in *Contributions to Probability and Statistics*. Stanford, CA, USA: Stanford Univ. Press, 1960, pp. 278–292, doi: 10.1007/978-1-4612-3678-8.
[47] P. A. Games and J. F. Howell, "Pairwise multiple comparison procedures with unequal N's and/or variances: A Monte Carlo study," *J. Educ. Statist.*, vol. 1, no. 2, pp. 113–125, Jun. 1976, doi: 10.3102/10769986001002113.
[48] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Evanston, IL, USA: Routledge, 1988, doi: 10.4324/9780203771587.
[49] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *J. Syst. Softw.*, vol. 80, no. 8, pp. 1349–1361, Aug. 2007, doi: 10.1016/j.jss.2006.10.049.
[50] S. Siddiqui and T. A. Khan, "Test patterns for cloud applications," *IEEE Access*, vol. 7, pp. 147060–147080, 2019, doi: 10.1109/ACCESS.2019.2946315.
[51] T. Górski, "The k + 1 symmetric test pattern for smart contracts," *Symmetry*, vol. 14, no. 8, p. 1686, Aug. 2022, doi: 10.3390/sym14081686.

HYUN-JAE CHOI received the B.S. degree in computer science and engineering and the M.S. degree in computer science from Pusan National University, Busan, South Korea, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests include software engineering, validation, and test automation.

HEUNG-SEOK CHAE received the B.S. degree in nuclear engineering from Seoul National University, Seoul, South Korea, in 1994, and the M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1996 and 2000, respectively. He is currently a Professor with the Department of Computer Science and Engineering, Pusan National University, Busan, South Korea. His research interests include object-oriented methodologies, software testing, and software verification.