RESEARCH ARTICLE

SAMRIDDHI Volume 12, Special Issue 3, 2020 Print ISSN : 2229-7111 Online ISSN : 2454-5767

# Compiler-Based Approach to Predict Reliability of Object-Oriented Programs

Vipin Kumar K S[1], Haritha Madhav C[2], Sheena Mathew[3],

[1,2] Dept of CSE, Govt. Engineering College, Thrissur, PIN-680009,India; e-mail : vipin.kumar.k.s@gectcr.ac.in

[3] Dept. Of CSE, School of Engineering,Cochin University of Science and Technology PIN-682022, India.

## ABSTRACT

In this work, we use structural analysis to estimate reliability of developed program. The compiler-based approach to reliability prediction first builds a System Dependence Graph (SDG) model for the program. The model is used for analysis, which output metric values representing structural complexity of the program. The metric values are then used to predict whether the program is reliable or not. Processing of the program is achieved by building a compiler based analysis tool using ANTLR [1].

**Keywords:** OOP, Metric, Software Reliability, Programming, SDG, Software, Testing.

*SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology, (2020); DOI : 10.18090/samriddhi.v12iS3.17*

## INTRODUCTION

Reliable software is expected to work free of errors for a specific period of time. It is difficult to express software reliability in terms of time. But many researchers have been successful in representing reliability as a function of time. In this work our aim is to broadly categorize a software as being reliable or not. Testing is done to improve the reliability aspect of the developed program. When considering object-oriented programs, testing is complicated by the features like dynamic polymorphism and inheritance [2]. Different phases of software development employ different approaches to testing [3] [4]. The testing phase often consumes 30 to 50 percent of total development time and cost. State based [5] and structural analysis based test case generation often helps in generating test cases that may not be anticipated by testers. Compiler based approach for preprocessing OOP is discussed in [6]. Compiler based model construction for distributed regression testing is discussed in [7]. In this work the compiler based model construction produces a System Dependence Graph (SDG) which is analyzed to evaluate different metrics suitable to predicting error proneness of the program. Since reliability is directly related to probability of error, more the likelihood of error lesser will be the reliability of the program and vice versa.

The work can be partitioned into

1. Software metrics selection for estimating reliability.

2. Compiler-based model construction and metric evaluation.

**Corresponding Author :** Vipin Kumar K S, Dept of CSE, Govt. Engineering College, Thrissur, PIN-680009,India; e-mail : vipin.kumar.k.s@gectcr.ac.in

**How to cite this article :**

Vipin Kumar K.S, V.P., Mathew, S., Madhav, H. (2020). Compiler-Based Approach to Predict Reliability of Object-Oriented Programs. *SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology, Volume 12, Special Issue (3), 74-79.*

**Source of support :** Nil

**Conflict of interest :** None

Estimating reliability based on metric evaluated. For metric selection we have used PROMISE DATA SET developed by NASA under the NASA Metrics Data Program. Now available under PROMISE SOFTWARE ENGINEERING REPOSITORY as PROMISE DATA SET, PROMISE DATA SET is maintained by University of Ottawa.

## SOFTWARE METRIC SELECTION

Metric selection is an important step in reliability based modeling. Metric selection increases efficiency by not compromising on accuracy. Selecting the right metric, from the data set is of at most importance to achieve higher performance level with minimum effort. The following steps were performed to arrive at a manageable set of metrics:

1. Cleaning the data set
2. Eliminate redundant features
3. Elimination of features by Recursive Feature Elimination (RFE)
4. Rank features by importance

The cleaning of data set comprised of removal of the features/metrics that showed no variation in values. Following this *findCorrelation* function in Cart R package was used to remove redundant metrics. Recursive feature elimination was then carried out on the resulting data set using the *rfe* function in R tool which resulted in eleven metrics. The accuracy when computed showed that further reduction reduced the accuracy as shown in Fig 1.

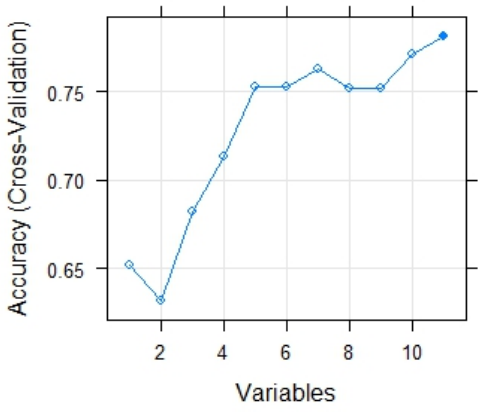


**Figure 1:** Accuracy with respect to number of features

The importance was calculated for these metrics using Learning Vector Quantization (LVQ). All metrics were seen to have an importance higher that 5 as shown in Fig 2.

```
ROC curve variable importance

                                Importance
sumLOC_EXECUTABLE                   0.8111
sumCYCLOMATIC_COMPLEXITY            0.7841
COUPLING_BETWEEN_OBJECTS            0.7697
WEIGHTED_METHODS_PER_CLASS          0.6991
RESPONSE_FOR_CLASS                  0.5872
NUM_OF_CHILDREN                     0.5382
DEPTH                               0.5332
PERCENT_PUB_DATA                    0.5230
DEP_ON_CHILD                        0.5204
LACK_OF_COHESION_OF_METHODS         0.5168
FAN_IN                              0.5002
```

**Figure 2:** Features ranked by importance

We were able to reduce from the initial set of ninety four metrics to eleven metric. The final list of metrics is:

PERCENT_PUB_DATA, COUPLING_BETWEEN_OBJECTS, DEPTH, FAN_IN, LACK_OF_COHESION_OF_METHODS, NUM_OF_CHILDREN, DEP_ON_CHILD, RESPONSE_FOR_CLASS, WEIGHTED_METHODS_PER_CLASS, sumCYCLOMATIC_COMPLEXITY and sumLOC_EXECUTABLE

## COMPILER-BASED MODEL CONSTRUCTION AND METRIC EVALUATION

The intermediate representations for procedural programs were extended by several researchers to be useful for representing object-oriented programs. Kung et al. proposed Object Relation Diagram(ORD) and Block Branch Diagram(BBD) for capturing essential object oriented features[8][9]. The Call Graph representation was introduced by Harrold and Rothermel[10]. It has been shown in the paper [11] that inheritance from the view point of depth of inheritance, maintenance and maintainability is an important feature of object-oriented programs. It is shown that it is 20% faster to make modifications to a program employing inheritance compared to programs without inheritance. Inheritance has been incorporated in many program models. In the work by Najumudheen et al. [12], they extended the SDG for object-oriented programs with a representation of inheritance. SDG has gained wide acceptance as a program representation. An Ext-SDG, where SDG is augmented with control flow is used in [13]. Numerous models pertaining to Object-Oriented Programs have been proposed in literature. One of the most popular model which was based on System Dependence Graph (SDG)

[14]. It is not possible to find a single model that is best suited for all purposes. One model may capture certain features of the program better than other models.

ANTLR is a powerful parser generator for understanding, preparing, executing and interpreting code. By taking input, which is a Context Free Grammar augmented with actions, ANTLR produces a parser that can create and traverse parse trees. ANTLR accepts EBNF grammar, which also includes lexer rules in addition to normal parser rules. The working of ANTLR tool can be summarized as shown in the Fig 3. Fig 3 also shows how the SDG Model based evaluation of metric is carried out.

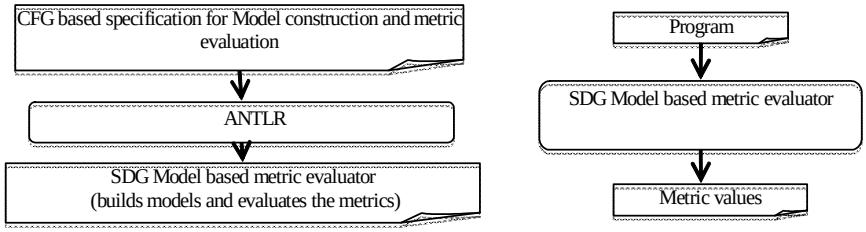


**Figure 3**: SDG Model based metric evaluation

```
public class Rectangle {
          private int length;
          private int width;
          private double result;
          public Rectangle(int length, int width) {
                    this.length = length;
                    this.width = width;
    }
          public double getArea()
    {
        Area area = new Area();
          result = area.rectangleArea(length,breadth);
          return result;
    }
}
```

```
public class Area {
          public int a;
          public int b;
          public double rectangleArea(int a, int b) {
                    if(a==0 || b==0)
                              return 0.0;
                    else return a*b;
          }
}

public class Execute {
          public static void main (String args[]) {
                    Scanner scn = new
Scanner(System.in);
                    Rectangle r = new Rectangle(10,15);
                    int value = r.getArea();
     }
}
```

**Figure 4**: Sample program

A sample program is given in Fig. 4 and its graphical representation is depicted in Fig. 5 for clarity.

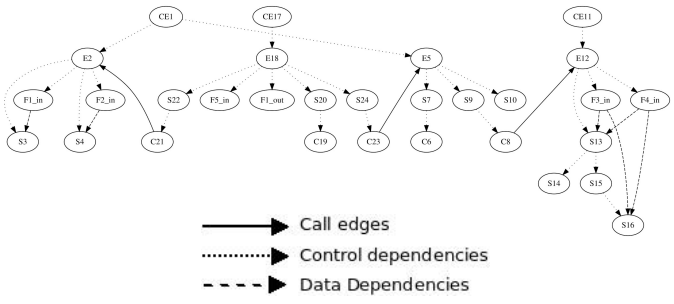


**Figure 5**: SDG for sample program

The nodes and edges of the SDG in Fig. 5 is constructed and represented internally as a result of the actions being carried during parsing. The Fig 6 shows the metrics evaluated from the SDG model for a sample java program in Fig. 4. Fig. 7 shows the SDG for the same, both of which are generated by the compiler-based tool.

```
Class Metrics
--------------
ClassName                CBO   DIT   NOC   FAN_IN RFC   DOC   WMC   S_CC  S_LOCEX   PPD      LCOM
------------------------ ----- ----- ----- ------ ----- ----- ----- ----- -------  -----      -----
Area                     0     0     0     0      1     0     1     3     4        100.00     0.00
Execute                  1     0     0     0      4     0     1     1     3        0.00     0.00
Rectangle                1     0     0     0      4     0     2     2     5        0.00     0.50
```

**Figure 6**: SDG Model based metric evaluator output

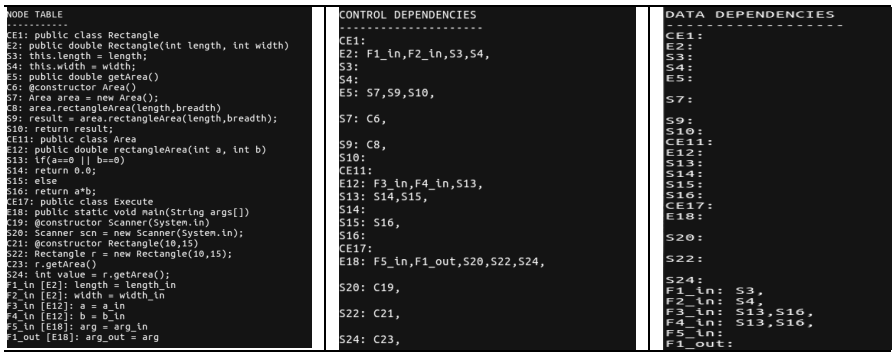


**Figure 7**: SDG Model for the program

## ESTIMATING RELIABILITY BASED ON METRIC EVALUATED

Inputs given for reliability prediction are java programs. Data set needed for training is created by using these java files for obtaining the target object oriented metric values. The java program is parsed to obtain these measures. The ANTLR based compiler tool is used for this purpose. The data for training comprised of metric values corresponding to the java programs, and target is the reliability value. This data is used to model the relationship between metrics and reliability. The functioning of the model is described in Fig 8.

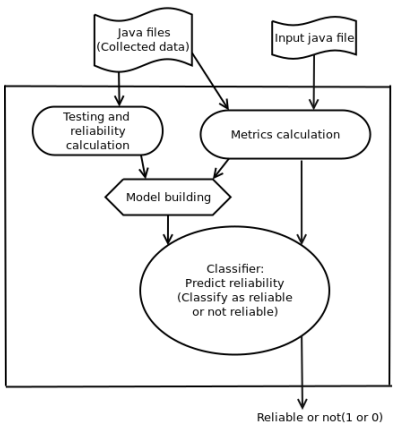


**Figure 8**: Reliability Prediction

The overall functionality of the model in Fig. 8 is implemented as different modules as listed below:

- Parsing and metrics generation module: The input program is parsed and metrics are generated. This forms the input value used as training data.
- Reliability determination module: The target values of the training data are reliability values. The reliability of the program is evaluated.
- Modeling using neural network: This module implements training of the neural network with the generated data to model the relationship between metrics and reliability.

According to Eldred Nelson's work [15], a program's reliability, means that execution failure does not occur in a sequence of execution of a program. So let us assume the total number of executions done is 'n'. Let the set of test cases be T, be:

$T = \{ t_1 , t_2 ,\ldots,t_i \}$, such that i ranges from 1 to N.

N is the number of test cases. There is a functionality that maps any input to output during execution. $F(t_I)'$, is the intended output when input for test case $t_I$ is given. The actual output obtained will be $F(t_i)$. The software failure is when one of the conditions occur:

- $F(t_i) \neq F(t_i)'$
- Failure to terminate
- Premature termination

If $P_i$ is the probability that $t_i$ is the test case being executed, then R(1) is said to be the probability that a single execution becomes successful which is represented as:

$$R(1) = \Sigma_{i=1}^{N} P_i X_i \qquad (1)$$

where $X_i$ is the probability of test case $t_i$ being successful. Then the probability that n executions will produce correct output,

$$R(n) = R_1(1)R_2(1)...R_n(1). \qquad (2)$$

Equation (1) applies to a single execution of a random test case, which is one from the N test cases available for the software. A considerable number of executions have to be carried out so as to obtain the reliability of the software. That is, the value of n is taken as 200 for the work. When we apply this to equation (2), the reliability value is obtained.

**Modeling Using Neural Network**

Prepared data is modeled using the neural network library of MATLAB. The model called patternnet[16] is used, using which we can identify a pattern in the input vector, rather than the dimension and easily classify the input into output classes. 80% of data were used for training, 10% for validation and 10% for testing. There have been many researches on metrics, that shows that different sets of metrics serves in determining different domain and different aspects like the complexity of software, productivity impact variables, object oriented properties, assessment of quality, flexibility, functionality etc. [17]. Different approaches to reliability estimations have been proposed like reliability assessment using various measures like Mean Time To Failure, failure intensity function, mean value function, etc. [18]. Also models based on a Mixed Poisson process where the failure rate follows an Inverse Gaussian distribution is discussed in [19]. For evaluation of bugs in the code, the bug statistics at the initial stages of testing is used to associate reliability issues[20]. But above all these, in order to predict the reliability of software accurately, it is important to apply models that both characterize the observed failure data well and make accurate predictions of the future [21]. In this work, the given program is classified as reliable or unreliable based on the structural complexity which is represented through various metrics reflecting the error proneness of the program. A sample of input data used in this work is shown in the Fig 9.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | PPD | 0 | 0 | 0 | 0 | 0 | 75 | 50 | 0 | 0 | 60 | 80 | 0 | 50 |
| 2 | CBO | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 1 | 2 | 1 | 21 | 8 |
| 3 | DIT | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | LCOM1 | 2 | 95 | 13 | 0 | 0 | 3 | 7 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | NOC | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| 6 | FAN_IN | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 0 | 3 | 1 | 2 | 1 |
| 7 | RFC | 33 | 12 | 9 | 11 | 9 | 3 | 12 | 3 | 4 | 11 | 1 | 21 | 8 |
| 8 | WMC | 5 | 12 | 8 | 10 | 1 | 3 | 5 | 3 | 4 | 10 | 1 | 2 | 5 |
| 9 | sumCC | 10 | 15 | 10 | 12 | 1 | 8 | 7 | 3 | 5 | 14 | 1 | 16 | 2 |
| 10 | sumLOC_EX | 47 | 22 | 36 | 44 | 19 | 20 | 25 | 16 | 17 | 29 | 4 | 99 | 9 |
| 11 | DOC | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 12 | | | | | | | | | | | | | | |
| 13 | Rel | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 14 | | | | | | | | | | | | | | |

**Figure 9:** Sample data for training

The patternnet we have used have one input layer, one hidden layer, and one output layer. The data that we have evaluated needs to be converted into relevant format that can be fed into neural network. From equations (1) and (2), we obtain reliability value of the software. To proceed to further with the processing, the obtained result is divided into two classes as reliable and not reliable. The reliability value obtained from equation(2) is considered as 1 if R(n) is greater than 0.5 and 0 if less than 0.5, for software being reliable and not reliable respectively. This is the target for the patternnet. The resultant confusion matrix is shown in Fig 10.

## CONCLUSION

In current trend of software engineering, object oriented design is extensively preferred in software engineering scenarios. All the metrics described are crucial in determining the quality and reliability of the developed software in such an environment. Object oriented programming has many useful features, the properties like inheritance, polymorphism, abstraction, and encapsulation.

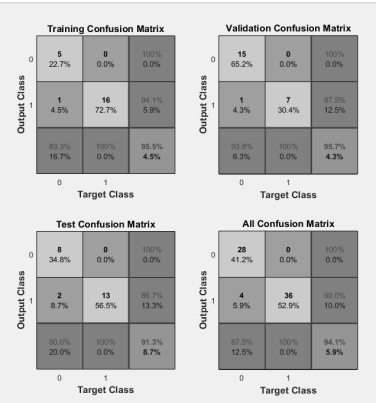


**Figure 10 :** Confusion Matrix

Object oriented metrics are guaranteeing factors to reduce maintenance effort and the cost by serving as early predictors to estimate software fault proneness. The work has improved accuracy in prediction of reliability of object oriented software from around 70% in previous works to more than 90%. This is the prime justification for the selected metrics to be most influencing factors in predicting reliability of software. Metrics observation is a proactive measure that can be taken to ensure reliability in object oriented software development.

## FUTURE WORK

The observations can be used to further enhance the prediction model for detecting the code portion where defects or improvements can be made. Suggestions for improvement in the code can also be a future scope.

## REFERENCES


[1] Terence, Parr. Definitive ANTLR reference, Building domain specific languages. 2007.

[2] A Survey of Testing Techniques for Object-Oriented Systems,. Johnson, Jr., Morris S. 1996. Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative research.

[3] Object-Oriented Integration Testing. Jorgensen, Paul C. and Erickson, Carl. 1994, Communications of ACM, pp. 30-38.

[4] An UML Statechart Diagram-Based MM-Path Generation Approach for Object-Oriented Integration Testing. Zhao, Ruilian and Lin, Ling. s.l.: World Academy of Science, Engineering and Technology, 2008, International Journal of Applied Mathematics and Computer Sciences.

[5] Object state testing for object-oriented programs. Gao, J.Z., et al., et al. 1995. Computer Software and Applications Conference.

[6] Model Based Distributed Testing of Object Oriented Programs. Vipin Kumar, K,S and Mathew, Sheena. s.l. : Procedia Computer Science, Elsevier, 2014. International Conference on Information and Communication Technologies.

[7] An Efficient Approach for Distributed Regression Testing of Object Oriented Programs. Vipin Kumar, K, S, Lallu, A and Sheena, Mathew. s.l. : ACM Digital Library, 2014. International Conference on Interdisciplinary Advances in Applied Computing.

[8] Class Firewall, Test Order, and Regression Testing of Object-Oriented Programs. Kung, David C., et al., et al. 1993, JOOP.

[9] Design Recovery for Software Testing of Object Oriented Programs. Kung, David C., et al., et al. 1993. Proceedings of Working Conferenceon on Reverse Enginering.

[10] Performing Data Flow Testing on Classes. Harrold, Mary Jean and Rothermel, Gregg. s.l. : ACM SIGSOFT Software Engineering Notes, 1994. SIGSOFT '94 Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering. pp. 154-163.

[11] The effect of inheritance on the maintainability of object-oriented software: an empirical study,IEEE. Daly, J., et al., et al. s.l. : IEEE, 1995. International Conference on Software Maintenance.

[12] A Dependence Representation for Coverage Testing of Object-Oriented Programs. Najumudheen, E,S,F., Mall, Rajib and Samanta, Debasis. 2010, Journal of Object Technology.

[13] Test case prioritization and distributed testing of object-oriented program,VIPIN KUMAR K S, SHEENA MATHEW,TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES,DOI: 10.3906/elk-1806-177

[14] Sayyad Shirabad, J. and Menzies, T.J. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada . Available: http://promise.site.uottawa.ca/SERepository.

[15] ESTIMATING SOFTWARE RELIABILITY FROM TEST DATA Eldred Nelson TRW Defense and Space Systems Group Redondo Beach, California USA, Elsevier, doi.org/10.1016/0026-2714(78)91139-3

[16] https://in.mathworks.com/help/deeplearning/ref/patternnet.html

[17] Yasser Ali Alshehri, Katerina Goseva-Popstojanova, Dale G. Dzielski and Thomas Devine "Applying machine learning to predict software fault proneness using change metrics, static code metrics, and a combination of them" DOI:978-1-5386-6133-8/18/$31.00, 2018 IEEE

[18] B. R. Tantri and N. N. Murulidhar, "Software reliability estimation of gamma failure time models," 2016 International Conference on System Reliability and Science (ICSRS), Paris, 2016, pp. 105-109. doi: 10.1109/ICSRS.2016.7815847

[19] N. R. Barraza, "A Mixed Poisson Process and Empirical Bayes Estimation Based Software Reliability Growth Model and Simulation," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, 2017, pp. 612-613.doi: 10.1109/QRS-C.2017.114

[20] Z. Bluvband, S. Porotsky and M. Talmor, "Advanced models for software reliability prediction," 2011 Proceedings - Annual Reliability and Maintainability Symposium, Lake Buena Vista, FL, 2011, pp. 1-5. doi: 10.1109/RAMS.2011.5754487

[21] V. Nagaraju, "Software Reliability Assessment: Modeling and Algorithms," 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Memphis, TN, 2018, pp. 166-169. doi: 10.1109/ISSREW.2018.