STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 2, 2019
DOI: 10.24193/subbi.2019.2.05

PREDICTING RELIABILITY OF OBJECT-ORIENTED SYSTEMS USING A NEURAL NETWORK

ALISA BUDUR, CAMELIA ŞERBAN, AND ANDREEA VESCAN

Abstract. One of the most important quality attributes of computer systems is reliability, which addresses the ability of the software to perform its required function under stated conditions for a stated period of time.

The paper aim is twofold. Firstly, the proposed approach explores how to define a metric to qualify the sub-aspects comprised in ISO 25010 regarding reliability as maturity and availability. Secondly, we investigate to what extent the internal structure of the system quantified by the Chidamber and Kemerer (CK) metrics may be used to predict reliability.

The approach for prediction is a feed-forward neural network with back-propagation learning.

The results indicate that CK metrics are promising in predicting reliability using a neural network method.

1. Introduction

Quality of a system can be described by different attributes such as reliability, maintainability, usability, etc. Among these attributes, reliability has an important role because it reveals how stable a system is or, in other words, how often it fails.

The definition of reliability is based exclusively on the software external behaviour, although it is well known that the internal structure has an important impact on a quality attribute such as reliability. For example, a reliable system has a complexity minimized as much as possible. Also, coupling in a reliable system is reduced at maximum because it facilitates testing. Considering this, we can say that the better we assess the internal structure of the system, the more accurate will be the prediction of its external behavior. Several studies [4], [2], [12], [11] reveal that Chidamber and Kemerer (CK) [5]

Received by the editors: October 24, 2019.

2010 *Mathematics Subject Classification.* 68T05,68M15.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing methodologies**]: Artificial Intelligence – *Learning*; D.2.8 [**Software engineering**]: Metrics – *Complexity measures*.

Key words and phrases. Reliability, prediction, neural network.

metrics have a strong impact on software reliability. These metrics are briefly presented in what follows [5].

Definition 1. Depth of Inheritance Tree (DIT) [5] is defined as the length of the longest path of inheritance from a given class to the root of the tree.

Definition 2. Weighted Methods per Class (WMC) [5]metric defined as the sum of the complexity of all methods of a given class. The complexity of a method is the cyclomatic complexity.

Definition 3. Coupling Between Objects *(CBO) [5] for a class c is the number of other classes that are coupled to the class c, namely that Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.*

Definition 4. Response for a Class *(RFC) [5] metric is defined as the total number of methods that can be invoked from that class.*

Definition 5. Lack of Cohesion in Methods *(LCOM) [5] is defined by the difference between the number of method pairs using common instance variables and the number of method pairs that do not use any common variables.*

Definition 6. Number of children of a class (NOC) [5] *is defined as the number of all direct sub-classes of a given class.*

Considering the reliability definition and the fact that the internal structure is very important for a reliable system, the goal of this paper is twofold. Firstly, to compute the reliability attribute based on the software external behavior, i.e number of faults occurred during the testing phase, as well as the number of faults discovered during the usage of the software - for this step, the Quality Model, ISO25010 [10] was considered because it addresses two of the four sub-characteristics related to reliability: *Availability*, *Maturity*, *Fault Tolerance* and *Recoverability*. Secondly, to investigate whether the neural networks can predict reliability attribute based on the previously defined attribute and having as predictors object-oriented design metrics.

The second step is very useful because it allows us to know the reliability quality attribute as early as possible in the development life cycle. This helps because it can suggest what classes are more likely to have bugs and the testing team can focus on testing features that use those classes. In this way, we identify bugs earlier and the cost of fixing them is lower.

The structure of the paper is the following: Section 2 presents the software reliability, Goal Question Metric (GQM) approach and how it is used to determine reliability, neural networks and a short related work section. Section 3 presents in more details how to achieve the two objectives of this paper:

how to compute reliability attribute and then how to predict it using a neural network. Section 4 describes the data sets used for validation, the conducted experiments and obtained results. Finally, the conclusions and future directions are emphasized in Section 5.

2. Setting the context

This section presents the theoretical aspects used in this research investigation, i.e. reliability, the GQM approach to quantify reliability, neural networks and a short related work.

2.1. **Reliability as an important aspect of safety-critical systems.** The official definitions of reliability are: "The ability of the software to perform its required function under stated conditions for a stated period of time" (*IEEE Standard Glossary of Software Engineering Terminology* [17]) and "The probability of failure-free operation of a computer program for a specified period of time in a specified environment" (ANSI [14]).

A safety-critical system is a system whose failure might lead to life loss, financial loss, and/or environmental damage. Many everyday systems can be dangerous for us and therefore, the software architects and developers should design and create systems that are very safe. An important question that raises here is *"How can we test that the system is safe?"*.

The first approach is to prove that there are no faults in it. This can be accomplished using formal mathematical methods in the design and proofs of the design correctness. The disadvantage of this approach is that it works well only for small systems.

The second approach is to accept that mistakes can appear and to consider error prediction methods. This is more generally adopted and can be done by quantifying reliability quality attribute of the system.

2.2. **Goal-Question-Metric approach.** Software metrics are very important in understanding, controlling and improving software quality. Fenton's theory of measurement [9] explains that any measurement must have a well defined goal, but in practice, a lot of measurements are not goal oriented and therefore, the data collected is not useful at all. Goal Question Metric approach [3] was introduced to ensure that the measurements are goal oriented. It has three steps:

(1) Define goals with respect to various points of view: quality, time etc.
(2) Define questions to characterize how the goals defined above could be achieved.
(3) Determine which metrics must be collected in order to answer the above questions.

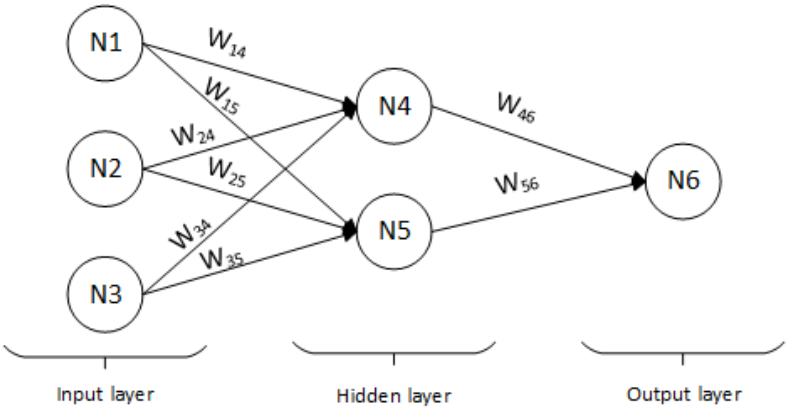

FIGURE 1. Structure of a feed-forward neural network.

2.3. **Neural networks.** A neural network is a supervised learning algorithm [13] that is inspired by biological neural networks. It learns how to perform tasks by taking into consideration already labeled data. For example, if we supply a neural network with weather data from the last month, it will learn how to predict the weather for the next days.

A neural network [13] consists of multiple nodes connected by links. A numeric weight is associated with each link. The neural network communicates with the environment through input and output nodes. A layered feed-forward network is a neural network in which every node is linked only to nodes in the next layer, for example in Figure 1 node $N5$ is linked to node $N6$ but not back to nodes in the previous layers.

Each node performs the following computation: it takes the input values from its input links (for example, input values for node N4 in Figure 1 are N1, n2, N3) and computes a new value (activation value), sending it along each of its output links. The computation consists of two parts. Firstly, it computes the weighted sum of the input values of the node. The weighted sum of a node is the sum of all its input values times their respective weights. Secondly, it computes the activation value of the node by applying the activation function to the weighted sum previously computed.

All the above steps are graphically presented in Figure 2.

Usually, learning in a feed-forward neural network is done using the back-propagation algorithm. Back-propagation learning works in the following way: the network is supplied with inputs and if it computes an output vector that matches the expected output, the algorithm terminates. Otherwise, an error is computed (the difference between the expected output and the actual output),

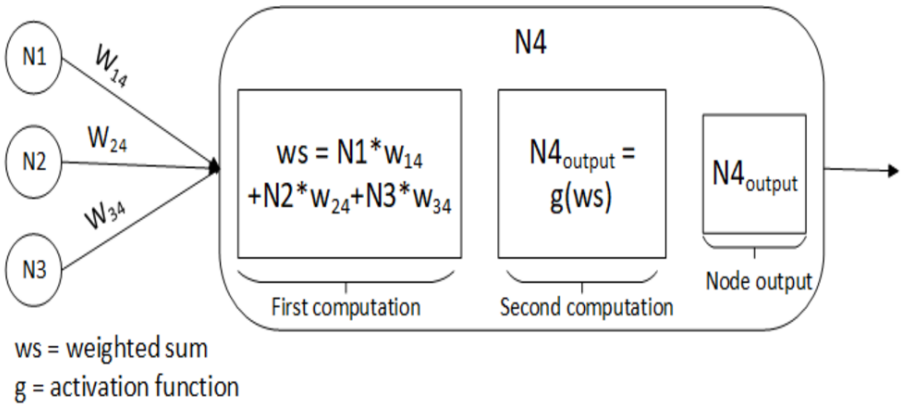

FIGURE 2. Node computation.

```
1  Network NeuralNetworkLearning(trainingData)
2
3    network <- a network with random assigned
4                weights;
5
6    while(prediction is incorrect or
7    termination condition not reached)
8      foreach item in trainingData
9        networkOutput =
10           network.ComputeOutput(item);
11       expectedOutput =
12           item.GetExpectedOutput();
13       UpdateWeightsBasedOn(networkOutput,
14                expectedOutput);
15     end foreach
16   end while
17
18   return network
```

LISTING 1. Generic learning in a neural network

then this error is used to adjust each weight in the network such that the next error to be smaller than the current one. The back-propagation uses gradient-descendant for dividing the error among all weights. The generic learning in a neural network is presented in Listing 1.

2.4. **Related work.** Reliability is one of the most important quality attributes when we describe safety-critical systems. It is so important because a fail in such a system could produce significant losses. This subject was of major interest in last years and several studies investigated its impact on software safety, as well as searched for methods through which we can predict and accomplish a reliability value from the earliest development stages.

How reliability prediction can increase trust in reliability of safety-critical systems was studied in paper [15]. The author determines a prediction model for different reliability measures (remaining failure, maximum failures, total test time required to reach a given number of remaining failures, time to next failure), concluding that they are useful for assuring that software is safe and for determining how long to test a piece of software.

Another approach [6] defined a classifier (with 37 software metrics) and use it to classify the software modules as fault-none or fault-prone. They compared their works with others and concluded that their model has the best performance.

The work described in [8] investigates how to solve the problem of determining the error rate of the electronic parts of a track circuit system (which is a safety critical system) by using Markov chains in order to predict the reliability of the fault-tolerant system.

An approach for assessing and predicting reliability of an object oriented system, taking a statistical approach by using multiple linear regression was proposed in [16].

In relation to existing approaches, ours investigates how we can use CK metrics to predict reliability and relates to approach [6], with the difference that we use CK metrics instead of cyclomatic complexity, decision count, decision density, etc. and we predict a reliability value for each class in the project, instead of classifying the modules in two categories.

3. Proposed approach for predicting reliability

This section presents our approach for reducing the time necessary to compute reliability of a software system. The approach is based on GQM and has the following structure:

- **Goal**: To reduce the time necessary to compute reliability of a software system.
- **Question**: Can the internal structure of a software affect the reliability?
- **Metrics**:
  (1) Collect CK metrics.

(2) Collect bugs (with severity and priority) from testing, operation and maintenance phases for a period of time.
(3) Predict reliability using CK metrics.

The data collected in the measurement phase (i.e. CK metrics and bugs data) are processed in two steps: *Reliability Assessment* and *Reliability prediction*. These two steps are graphicaly detailed in Figure 3.

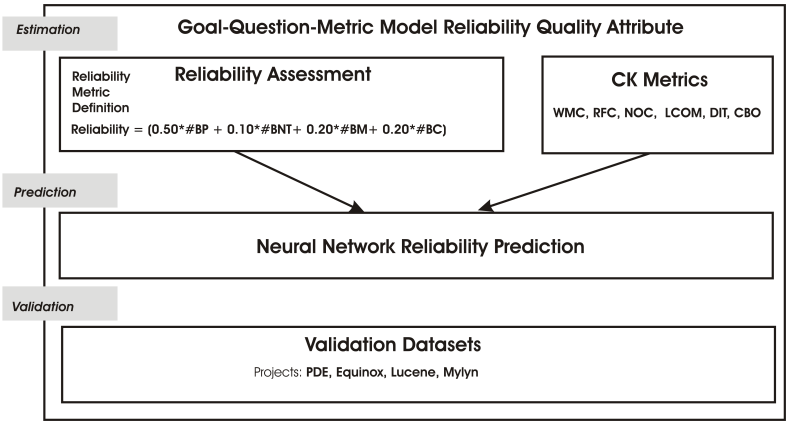

FIGURE 3. The two steps applied in processing the data collected using the GQM approach

The first step, named *Reliability Assessment*, aims to find a formula for computing the reliability quality attribute taking into account the information collected about bugs. It is known that the bugs found the in system are classified by severity and priority in the following way: bugs considered to be priority, bugs being non trivial, bugs considered to be critical and bugs that have a major importance. So, it is known the number in each category of bugs, for each class. For this, the ISO25010 Quality Model [10] was used. Four reliability sub-characteristics are related to this model: *Maturity*, *Availability*, *Fault Tolerance* and *Recoverability*. The main aspects for *Maturity* and *Availability* are related to the post release faults/bugs found in the analyzed system. We claim that these sub-characteristics of reliability could be correlated to bugs discovered during testing and maintenance phase of the development, considering their severity and priority. number of bugs found in the source code of a class. An important remark that should be emphasized here, is that various aspects should be considered to assess the reliability of a class, not only those aspects related to bugs. Finding a perfect metric is a very difficult problem, thus the proposed metric does not claim an equality

relation between bugs and reliability. Others aspects related to reliability can be added to improve the metric effectiveness.

Having this in mind, we establish weights for each of the above four categories of bugs having into account the priority in treating these faults/bugs. Thus, we considered assigning a greater impact for high priority bugs #BHP, major bugs #BM and for critical ones, #BC, with weights of 0.25. Common bugs are the lowest priority and we consider the weights of 0.15 for non-trivial bugs and 0.10 for common bugs. The pairs of (weight, bug category) are the following: {(0.50,#BP), (0.20,#BM), (0.20,#BC), (0.10,#BNT)}, where (#BP) represents the number of bugs viewed as being priority, (#BNT) is the number of bugs considered to be non trivial, (#BM) denotes the number of bugs with a major importance, and (#BC) is number of bugs treated as being critical. The reliability of a class is defined as an aggregate measure by means of Equation 1 that linearly combines the number of different categories of bugs.

$$Reliability = 0.5 * \#BP + 0.10 * \#BNT + 0.2 * \#BM + 0.2 * \#BC \tag{1}$$

The reliability measured in above described way can be only computed in the latest stages of development, when we have a functional piece of software. The goal of the GQM approach is to find a way to compute reliability as early as possible, so in the second step of processing the data collected, named *Reliability Prediction*, we investigate the potential of system's internal structure, expressed by CK metrics, to predict reliability.

To predict reliability, a feed-forward neural network with back-propagation learning is used, with the following structure (see Figure 4): six nodes on the input layer, one node on the output layer and two hidden layers, each of them having four nodes. Each node uses the Bipolar Sigmoid activation function, given by the following formula:

$$g(h_i) = \frac{1 - e^{-h_i}}{1 + e^{-h_i}} \tag{2}$$

This investigation uses a neural network for the following reasons: it has the ability to learn non-linear and complex relationships, after learning it can generalize [13], which means that it shows good results even for unseen data and the way that it works is simple and understandable.

When the algorithm training terminates, we want to know how well the algorithm will work on an unseen dataset. In some cases, it is quite difficult to achieve this because of insufficient data. The concept of cross-validation [7] is used in order to help us to solve this problem.

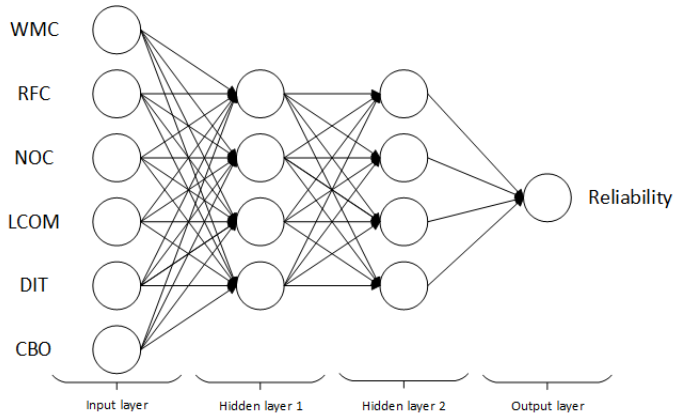

FIGURE 4. Structure of the feed-forward neural network used.

The idea of cross-validation is to put aside a part of the training data and use it later to test the trained model. In this way, the model is validated on an unseen dataset. This technique causes another problem: by removing a part of the training data, we may lose some patterns. In order to solve the second problem, the concept of k-fold cross-validation is used. K-fold cross-validation involves that the entire dataset to be split into k subsets. We will train the model k times and each time we will use a different subset for testing and the rest k-1 subsets for training. The total error of the model will be the average error of all k trials. Our investigation considered splitting the entire data set into ten subsets: nine of them are used for training and the remaining one is used for testing. The model is trained ten times and each time the testing set is changed. That means that each subset is used once for testing and nine times (k-1) for training. Figure 5 presents how cross-validation was applied in our experiments. Black rectangles represent the testing subsets and white rectangles represent the training subsets.

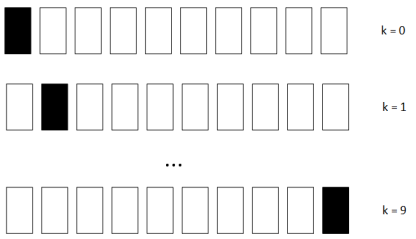

FIGURE 5. Cross-validation process used.

4. Experiments description

This section expose the datasets used to develop the reliability neural network prediction model. In order to validate our model, we used the Root Mean Square Error.

4.1. **Datasets.** The data set used is "Bug prediction dataset" and is described in [1]. For this research, the chosen data are collected from the last version of five different software systems: JDT (Java development tool - release 3.4, version 91), PDE (Plug-in Development Environment - release 3.4.1, version 97), Equinox (release 3.4, version 91), Lucene (release 2.4.0, version 99), and Mylyn (release 3.1, version 98). Table 1 compares the characteristics of each project: JDT includes an index-based search infrastructure used for refactoring, PDE yields solutions for Eclipse plug-ins, Equinox is an implementation of the OSGi R6 framework, Lucene implement an indexing and search technology, and Mylyn is a task management tool for developers.

Table 1. Characteristics of investigated Projects

Metrics	**Characteristics of projects**				
	UI	***Framework***	***Indexing and search***	***Plug-in manag.***	***Task manag.***
JDT	**Y**	**N**	**Y**	**N**	**N**
PDE	**Y**	**N**	**N**	**Y**	**N**
Equinox	**Y**	**Y**	**N**	**N**	**N**
Lucene	**Y**	**N**	**Y**	**N**	**N**
Mylyn	**Y**	**N**	**N**	**N**	**Y**

These data contain CK metrics and number of bugs categorized (with severity and priority) for each class of the system. Data are collected during the testing, operational and maintenance phases.

More information about the number of classes in each project and number of bugs may be visualized in Table 2 (C=number of total classes with no bugs, CB=number of classes with bugs, #B=number of bugs, #BNT=number of bugs Non Trivial, #BM=number of bugs Major, #BC=number of bugs Critical, #BHP=number of bugs High Priority)

4.2. **Experiments methodology.** This investigation used five experiments, using the five projects/datasets. In each experiment, a neural network-based prediction model was trained using 9/10 data from a single dataset (each experiment used a different dataset for training). Each prediction model was

TABLE 2. Data sets information

Metrics	Data sets information						
	$\#C$	$\#CB$	$\#B$	$\#BNT$	$\#BM$	$\#BC$	$\#BHP$
JDT	**44**	**997**	**11605**	**10119**	**1135**	**432**	**459**
PDE	**426**	**1497**	**5803**	**4191**	**362**	**100**	**96**
Equinox	**120**	**324**	**1496**	**1393**	**156**	**71**	**14**
Lucene	**197**	**691**	**1714**	**1714**	**0**	**0**	**0**
Mylyn	**701**	**1862**	**14577**	**6806**	**592**	**235**	**8004**

then validated in two steps. The first step was to validate it using the cross-validation technique, which means that we used for validation the remaining 1/10 data from the dataset that was used for training. The second step was to validate the model using data from the other four projects/datasets. More information about each experiment is listed in Table 3.

TABLE 3. Training and testing data for each experiment

Experiments	Training data	Validation data
Experiment 1	9/10 of JDT	1/10 of JDT, PDE, Equinox, Lucene, Mylyn
Experiment 2	9/10 of PDE	1/10 of PDE, JDT, Equinox, Lucene, Mylyn
Experiment 3	9/10 of Equinox	1/10 of Equinox, JDT, PDE, Lucene, Mylyn
Experiment 4	9/10 of Lucene	1/10 of Lucene, JDT, PDE, Equinox, Mylyn
Experiment 5	9/10 of Mylyn	1/10 of Mylyn, JDT, PDE, Equinox, Lucene

4.3. **Results.** The mean reliability values computed using the bugs-based formula for each project, are listed in Table 4. The mean reliability values computed using the neural network based prediction model for each project and for each experiment are listed in Table 5. The bolded values are obtained in

the cross-validation step, while the others are obtained in the second step of validation. A visual representation of the results is listed in Figure 6.

The experiments explored how the obtained neural network model differs in terms of performance when using projects with different characteristics.

TABLE 4. Mean reliability value for each project computed with the bugs based formula

Projects	**Reliability values by bugs**
JDT	0.048582
PDE	0.023806
Equinox	0.062020
Lucene	0.030623
Mylyn	0.049389

TABLE 5. Mean reliability value for each project predicted by neural network prediction model

Projects	**Reliability values**				
Experiment	***1***	***2***	***3***	***4***	***5***
JDT	**0.046973**	0.026686	0.029756	0.037569	0.068397
PDE	0.054409	**0.020266**	0.046260	0.049615	0.061642
Equinox	0.070716	0.043553	**0.061428**	0.073883	0.118110
Lucene	0.043030	0.0022160	0.031426	**0.025880**	0.066846
Mylyn	0.023523	0.018856	0.015811	0.031991	**0.038243**

To validate our model we use the Root Mean Squared Error (RMSE) metric. It is computed as the square root of the average of squared differences between prediction and actual observation. The metric represents the standard deviation of prediction errors (the residuals).

The RMSE formula is:

$$RMSE = \sqrt{\overline{(f-o)^2}}, \tag{3}$$

Where:
f = forecasts (expected values or unknown results),
o = observed values (known results).

The model is better in its predictions when RMSE is lower, thus the predicted values are close to the observed values.

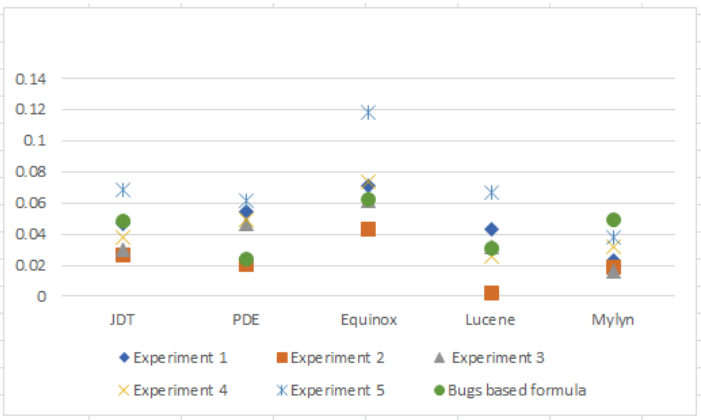

FIGURE 6. Mean reliability value for each project predicted by neural network prediction model.

The RMSE analysis for each experiment is listed in Table 6. Also, a visual representation of the results is listed in Figure 7.

TABLE 6. RMSE analysis for each experiment

Projects	**RMSE**				
Experiment	***1***	***2***	***3***	***4***	***5***
JDT	0.047940	0.075931	0.096312	0.075865	0.090505
PDE	0.060582	0.043147	0.155933	0.063116	0.091675
Equinox	0.113011	0.110130	0.109097	0.097998	0.174369
Lucene	0.068518	0.051839	0.102805	0.043320	0.112893
Mylyn	0.069323	0.072348	0.103535	0.070011	0.063391

Our findings on predicting reliability using CK metrics considering various projects with different characteristics as a basis for the neural network model construction identified best RMSE for the PDE project, thus with UI and plug in management characteristics. Worst value is obtained with Equinox project, thus with UI and Framework characteristics. Overall these findings are in accordance with findings reported in [16] where a statistical approach by using multiple linear regression was used for the same set of data.

5. CONCLUSION

We have proposed in the current paper an approach to measure and investigate reliability of an object oriented system, employing two steps: estimating

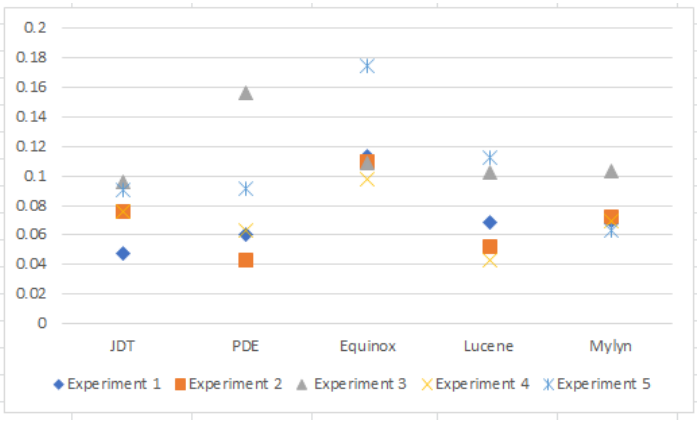

FIGURE 7. RMSE analysis for each experiment.

reliability using the numbers of bugs and predicting reliability using a neural network model based on CK metrics values. The present study confirmed the findings about the relevance and impact of CK metrics to quantify the reliability quality attribute.

The neural network model obtained to predict reliability is validated using a date set containing over 5000 instances/classes, grouped in 5 projects. The experiments revealed that a UI and indexing and search (JDT) project obtain the "best" neural network reliability prediction model.

Future work will investigate the reliability prediction problem by aggregating the results obtained using regression equation and neural networks prediction model. Another future direction refers to apply the equation prediction for other quality attributes.

REFERENCES

[1] M. D'Ambros, M. Lanza, R. Robbes,"An Extensive Comparison of Bug Prediction Approaches",Proceedings of MSR, 2010, pp. 31–41.

[2] V.R. Basili, L.C. Briand, W.L. Melo, *A Validation of Object-Oriented Design Metrics as Quality Indicators.* Technical Report, Univ. of Maryland, 1995. p. 1-24.

[3] V. Basili, D. Rombach. The TAME project: Towards Improvement-Oriented Software Environments. IEEE Transactions on Softw. Engineering, 14(6), jun 1988.

[4] F. Brito e Abreu and W. Melo, *Evaluating the impact of object-oriented design on software quality*, Proceedings Third Int. Software Metrics Symposium, 1996., 90–99

[5] S. R. Chidamber, C. F. Kemerer, *A Metric Suite for Object- Oriented Design*, IEEE Transactions on Software Engineering. 20 (6), 476–493 (1994)

[6] S. Chitra, K. Thiagarajan, M. Rajaram: Data collection and Analysis for the Reliability Prediction and Estimation of a Safety Critical System Using AIRS. International Conference on Computing, Communication and Networking, (2008)

[7] Cross-Validation in Machine Learning, https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f. Last accessed 17 Feb 2019
[8] M. Danhel, Prediction and Analysis of Mission Critical Systems Dependability, PhD Thesis, Faculty of Information Technology, Czech Technical University (2018)
[9] N. Fenton. *Software Measurement: A Necessary Scientific Base.* IEEE Transactions on Softw. Engineering, 20(3), 1994.
[10] ISO25010 description information, https://www.iso.org/standard/35733.html, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[11] B. Kitchenham, S. L. Pfleeger, N. E. Fenton, *Towards a Framework for Software Measurement Validation*, IEEE Trans. on Software Engineering, 21(12), 929–944 (1995)
[12] W. Li, S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, 23(2):111–122, 1993
[13] S. Russel, P. Norvig, : Artificial intelligence: a modern approach. Alan Apt, Englewood Cliffs, New Jersey 07632 (1995)
[14] A. Quyoum, UdM. Din Dar, S.M.K. Quadr: Improving software reliability using software engineering approach—a review. I.J. Comput. Appl. 10(5), 0975– 8887 (2010).
[15] N. Schneidewind: Reliability Modeling for Safety-Critical Software. IEEE Transactions on Reliability 46(1), 88–98 (1997)
[16] C. Serban, A. Vescan, "Predicting Reliability by Severity and Priority of Defects", Proceedings of the 2Nd ACM SIGSOFT International Workshop on Software Qualities and Their Dependencies, 2019, pp. 27–34.
[17] Standards Coordinating Committee of the IEEE Computer Society, IEEE Standard Glossary of Software Engineering Terminology, IEEE-STD-610.12-1990 (1991)

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania

Email address: {camelia, avescan}@cs.ubbcluj.ro, abudur@riasolutionsgroup.com