Towards a Reliability Prediction Model based on Internal Structure and Post-Release Defects Using Neural Networks

Andreea Vescan
Babes-Bolyai University
Cluj-Napoca, Cluj, Romania
avescan@cs.ubbcluj.ro

Camelia Şerban
Babes-Bolyai University
Cluj-Napoca, Cluj, Romania
camelia@cs.ubbcluj.ro

Alisa Budur
Babes-Bolyai University
Cluj-Napoca, Cluj, Romania
alisabudur@gmail.com

ABSTRACT

Reliability is one of the most important quality attributes of a software system, addressing the system's ability to perform the required functionalities under stated conditions, for a stated period of time. Nowadays, a system failure could threaten the safety of human life. Thus, assessing reliability became one of the software engineering‘s holy grails. Our approach wants to establish based on what project's characteristics we obtain the best bug-oriented reliability prediction model. The pillars on which we base our approach are the metric introduced to estimate one aspect of reliability using bugs, and the Chidamber and Kemerer (CK) metrics to assess reliability in the early stages of development. The methodology used for prediction is a feed-forward neural network with back-propagation learning. Five different projects are used to validate the proposed approach for reliability prediction. The results indicate that CK metrics are promising in predicting reliability using a neural network model. The experiments also analyze if the type of project used in the development of the prediction model influences the quality of the prediction. As a result of the operated experiments using both within-project and cross-project validation, the best prediction model was obtained using PDE (PlugIn characteristic) for MY project (Task characteristic).

CCS CONCEPTS

• **Software and its engineering → Software reliability**; **Software design tradeoffs**; *Software maintenance tools*; *Maintaining software*.

KEYWORDS

Reliability, Metrics, Assessment, Prediction, Neural network, Object-Oriented Design

ACM Reference Format:
Andreea Vescan, Camelia Şerban, and Alisa Budur . 2021. Towards a Reliability Prediction Model based on Internal Structure and Post-Release Defects Using Neural Networks. In *Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3463274.3463363

1 INTRODUCTION

One of the factors that affects the software development process resides in the quality pressure. DevOps help to increase the deployment speed with high software quality. Several studies investigate the links between DevOps and software quality [19], [24], some revealed that there is a need for further research in the area of measurement and proposing metrics for various development stages to assess software performance. Thus, there is a positive relation between DevOps's practices and software quality, i.e. quality is increased when practice DevOps by following CAMS (Culture, Automation, Measurement, Sharing) framework.

To assure a high quality of the software, one of the key factors resides in the reliability of software system. Reliability expresses “the degree to which a system is free of faults or can recover to a stable state after changes were applied” [20]. Also, reliability can be expressed as “the probability of failure-free operation of a system over a specified time within a specified environment for a specified purpose" [25].

The definition of reliability seems to be related only to the software system external behaviour. However, *failures* are usually derived from *faults* or *design flaws* within the system internal structure [30]. It is well known that a good internal design structure has a strong positive impact on external quality attributes such as *reliability*, *maintainability* and *reusability*. For instance, the complexity of a reliable system should be minimized as much as possible. Also, coupling is another important aspect: a highly coupled system is difficult to be tested, due to its dependencies between constituents components. Insufficient testing increases the probability of failures. Thus, assessing the internal structure of the software system, afford us to predict its external behaviour. One efficient way to assess a software system's internal structure is by means of software metrics.

The motivation of this work is to help in the DevOps practices regarding the measurement of reliability quality attribute and provide a prediction model for defect classes. Having into account the reliability definition and the above-mentioned reasoning regarding the vital influence of the system internal structure on its external behaviour, the pillars of this research investigation are *a metric for reliability definition using the aspect relating to bugs* and *the use of metrics that quantify the internal structure of the system to predict the system reliability*:

- Thus, firstly we define a metric for reliability assessment, based on the software external aspects that influence reliability, in this approach we consider software bugs. We study *one of the indicators of reliability* in the form of the bug count, advocating once again that even if reliability is an operational characteristic of the software (i.e., remaining faults

activation is environment context-dependent) post-release bugs influence its value. For example, the "high priority" bugs may be framed/enclosed as "every time the software is executed, a remaining fault is activated", thus relating to reliability estimation.
- Secondly, the paper employs metrics that quantify the internal structure of the system in predicting the system reliability.

Thus, based on those two pillars, our approach aims to uncover associations between the project 's characteristics and reliability, using a neural network prediction model and five open source projects, conducting both within and cross-project validation. This investigation is a challenging problem due to the fact that reliability is a dynamic emerging aspect of software, which can be hardly captured by observing static metrics alone. This prediction is a necessity due to the fact that assessing the reliability later in the development life cycle of the project (when failures are known) may result in increasing the cost of modifications for improvement. *The novelty of this paper refers therefore at proposing a neural network - based prediction model for reliability defined by one aspect (i.e. bugs) using CK metrics, reducing the cost of improvements.*

The paper is organized as follows: Section 2 contains related work. Section 3 describes the research design: the research question, the Goal-Question-Metric (GQM) approach used for reliability assessment and prediction along with the neural network approach, the used dataset and the description of experiments. Section 4 outlines the obtained results, followed by a discussion. Section 5 portrays our long-term objectives and planned work. Section 6 discuses threats to validity that can affect the results of our study. The conclusions of our paper and further research directions are outlined in Section 7.

2 RELATED WORK

Reliability is one of the most important measurements when we describe safety-critical systems. It is so important because a fail in such a system could produce life losses. This subject was of major interest in the last years and several research works studied its impact on software safety, as well as methods through which we can predict and accomplish a high reliability value from the earliest development stages.

How reliability predictions can increase trust in the reliability of safety-critical systems was studied in paper [27]. The author determines a prediction model for different reliability measures (remaining failure, maximum failures, total test time required to attain a given fraction of remaining failures, time to next failure), concluding that they are useful for assuring that the software is safe and for determining how long to test a piece of software.

Another approach [4] defined a classifier (with 37 software metrics) and used it to classify the software modules as fault-none or fault-prone. They compared their work with others and concluded that their model has the best performance. The approach in [14] proposes a new tool named *Automated Reliability Prediction System* for predicting the reliability of safety-critical software. An experiment was conducted where some students used this tool. The result was that they made fewer mistakes in their analysis.

The work described in [18] tries 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. The paper [15] proposes an approach for predicting software reliability using Relevance vector machines as kernel-based learning methods that have been adopted for regression problems.

In relation to existing approaches, ours investigates how we can use Chidamber and Kemerer (CK) [3] metrics to predict reliability and relates to approach [4], 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 design classes in two categories – faulty or healthy.

3 RESEARCH DESIGN OR METHODOLOGY

This section presents our approach for reliability prediction, presenting first our research question, the ingredients of our investigated problem (the proposed reliability metric definition, and the metrics used to define the prediction model), and then the determination of the neural network model for reliability prediction.

3.1 Research Question

In this paper, we empirically try to uncover, using five open source projects, the associations between the project's characteristics and reliability using a prediction model based on Neural Networks. Therefore, we use both within and cross-project approaches to discover the relations between the characteristics of the projects used as training and those used to validate the obtained reliability prediction model. The details about each project's characteristics are provided in Section 3.3.

More specifically, the study aims at addressing the following research question:

RQ: What is the association between the project's characteristics and reliability expressed by bugs when using a CK-based metrics neural network prediction model?

3.2 Goal-Question Metric approach to quantify reliability

The studies on how to quantify reliability highlight two aspects: *reliability assessment*, respectively *reliability prediction*. Reliability assessment relies on collecting data during testing, operation, and maintenance in order to conclude about the overall reliability of the system. It can be seen as a "black box" approach, and very few information can be inferred about which part of the code has a negative impact on reliability. Also, this analysis is performed late in the development cycle, when making modifications for improvement are costly. Reliability prediction is collecting information from the code (quantified by means of software metrics) and uses different approaches, such as simulations, statistical analysis, and others, to infer a prediction about reliability. Thus, the results of reliability assessment can be used in other future projects to predict reliability based on the internal structure of the analyzed system.

To approach the above-mentioned aspects on how to quantify reliability, we are led by Fenton's theory of measurement: "any measurement activity must have clear objectives". The Goal-Question-Metric (GQM) model [1] spells out the necessary obligations for setting objectives before embarking on any software measurement activity [17]. Thus, our proposed model to quantify reliability is driven by a GQM approach applied for both aspects mentioned above regarding on how to quantify reliability:

- First of all, the measurements objectives are linked to *reliability assessment* defining a new metric to quantify reliability,
- Then, we connect the measurements objectives to *reliability prediction*, by using the metric defined at the previous step and the characteristics of the software system internal structure.

In what follows, we present the two steps involved in our approach to predict reliability, using a GQM approach. A graphical view of our approach is presented in Figure 1.

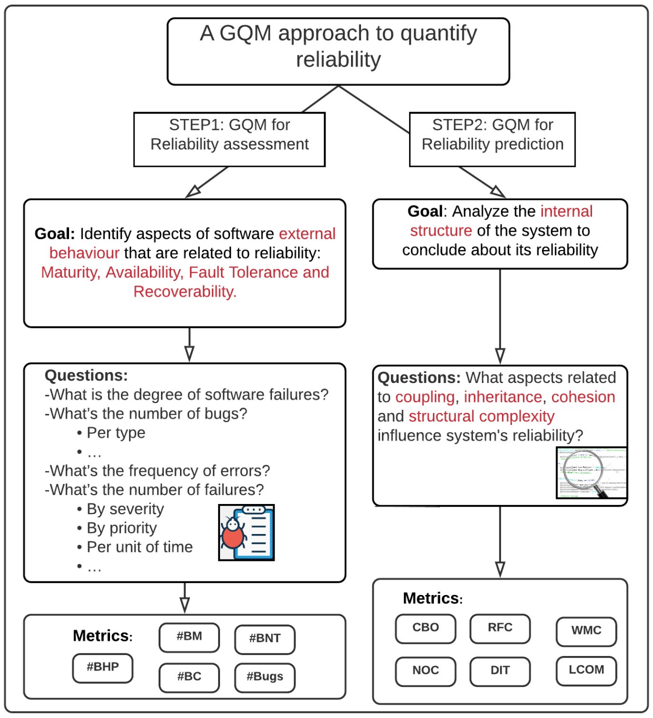
A GQM approach to quantify reliability
STEP1: GQM for Reliability assessment
STEP2: GQM for Reliability prediction
Goal: Identify aspects of software external behaviour that are related to reliability: Maturity, Availability, Fault Tolerance and Recoverability.
Goal: Analyze the internal structure of the system to conclude about its reliability
Questions:
-What is the degree of software failures?
-What's the number of bugs?
• Per type
• …
-What's the frequency of errors?
-What's the number of failures?
• By severity
• By priority
• Per unit of time
• …
Questions: What aspects related to coupling, inheritance, cohesion and structural complexity influence system's reliability?
Metrics:
#BM
#BNT
#BHP
#BC
#Bugs
Metrics:
CBO
RFC
WMC
NOC
DIT
LCOM

Figure 1: Detailed Goal-Question-Metric Model [1] applied in two steps in our investigation/approach

3.2.1 GQM approach for reliability assessment. Reliability is generally accepted as the major factor in software quality since it quantifies *software failures*, which can make a powerful system inoperative. Having into account the reliability definition of a software system as being *the probability of failure-free operation for a specified period of time* we aim to measure its complementary value, its degree of failures.

In order to assess reliability, we follow the steps implied in the GQM approach. The *first step* consists in establishing *The Goal of Measurement*. In this respect, "Identify aspects of software external behaviour that are related to reliability". The current approach considers these aspects as being reliability sub-characteristics from the ISO25010 Quality Model [9]: *Maturity*, *Availability*, *Fault Tolerance* and *Recoverability*. *The second step* is to derive *Questions* that must be answered to determine if the Goals are met. For each of the questions highlighted in the previous step, identify metrics that quantify the aspects involved in questions. This is the *third step* of GQM approach. This process is detailed in Figure 1 left side.

Following the GQM approach described before, we claim that the four subcharacteristics of reliability are related in a great extent to the post-release faults/bugs found in the analyzed system. These bugs are grouped in four categories considering their severity and priority. Thus, the following types of bugs were reported by a bug tracking system: #HighPriorityBugs (HPB) - number of bugs considered to be a priority, #NonTrivialBugs (NTB) - number of bugs being non trivial, #MajorBugs (MB)- number of bugs having a major importance, #CriticalBugs (CB) - number of bugs considered to be critical, and #Bugs - number of bugs that were not categorized.

We establish weights for each of the above four categories considering an order relation that establishes a priority in solving these faults/bugs, thus assigning a greater impact for high priority bugs HPB, major bugs BM and for critical ones, CB, 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. This order relation, together with the corresponding weight values is defined as a set of pairs: {(0.25,HPB), (0.25,MB), (0.25,CB), (0.15,NTB), (0.10,Bugs) }. The reliability of a class is defined as an aggregate measure by means of Equation 1 that linearly combines the number of different types of bugs:

$$Reliability = (0.25 * HPB + 0.15 * NTB + 0.25 * MB + 0.25 * CB + 0.10 * Bugs). \quad (1)$$

Thus, the *Metric* for the reliability assessment is that stated above in Equation (1).

3.2.2 GQM approach for reliability prediction. The process of designing software is error prone and object-oriented design makes no exception. We have argued earlier that the flaws of the software system internal structure have a highly negative impact on quality attributes such as reliability, reusability, or maintainability. Thus, the system internal structure affords us to predict its external behaviour.

Having into account the above-mentioned motivation to predict reliability, we collect information from the code by means of software metrics and we use various approaches, such as simulation, statistical analysis, and others, to infer a prediction about reliability. Our proposal will follow this approach using the information provided by Chidamber and Kemerer (CK) [3] object-oriented metrics.

The reason for choosing CK metrics is that several studies [2], [13], [31], [28] [12], reveal them having a strong impact on software reliability by predicting fault-prone classes. In addition, these metrics measure four *internal characteristics* that are essential to object orientation, i.e., coupling, inheritance, cohesion, and structural complexity [17]. All these metrics are defined at the class level, thus we predict the reliability of a class.

Following a GQM approach to predict reliability, our *Goal* is to identify those aspects of the software system internal structure that are related to reliability, and then to derive questions that suggest us metrics to quantify these internal structure characteristics. This process is detailed in Figure 1 right side.

Thus, the goal of this study is to explore the relationship between object-oriented metrics and reliability by predicting reliability at the class level. For this, we use the value of reliability, computed with Equation 1, as target value, and the values of CK metrics as independent variables in the prediction model.

In order to predict the reliability, a feed-forward neural network with back-propagation learning is used, with the following structure: six nodes on the input layer (one for each considered metric), one node on the output layer and two hidden layers, each of them having four nodes. Each node uses the bipolar sigmoid activation function [26]. The termination condition of training is either the error to be less or equal then 0.001 or the number of epochs to be at most 10000. After training this neural network, we obtained a neural network model for reliability prediction.

3.3 Dataset

The dataset used in our investigation is described in [5]: JDT [10], PDE [23], Equinox (EQ) [6], Lucene (LU) [16], and Mylyn (MY) [21] projects. The values for the metrics WMC, RFC, NOC, LCOM, DIT, CBO are used for the last version of the projects before release [5], thus the faults/bugs were found in the analyzed system during a period o six months.

More information about the number of classes in each project and the number of bugs may be visualized in Table 1 (#C=number of total classes, #CB=number of classes with Bugs, #B=number of Bugs, #NTB=number of Non Trivial Bugs , #MB =number of Major Bugs, #C=number of Critical Bugs, #HPB=number of High Priority Bugs), and #Bugs - number of bugs that were not categorized.

Table 1: Dataset description

	#C	*#CB*	*#B*	*#NTB*	*#MB*	*#CB*	*#HPB*
JDT	**997**	**206**	**374**	**17**	**35**	**10**	**3**
PDE	**1497**	**209**	**341**	**14**	**57**	**6**	**0**
EQ	**324**	**129**	**244**	**3**	**4**	**1**	**0**
LU	**691**	**64**	**97**	**0**	**0**	**0**	**0**
MY	**1862**	**245**	**340**	**187**	**18**	**3**	**36**

Our study used data from 5 different software projects. For example, the experiment that considered the JDT project for training, the prediction model, used over 1000 classes for training, and the projects used for validation had classes as follows: 2 projects with a number of classes between 400 and 1000 classes, and 2 projects around 2000 classes. Thus, over 1000 instances/classes were used in the determination of the reliability prediction neural network model and for the validation phase over 5000 instances/classes were used (cumulative over the 4 projects).

The characteristics used in our investigation related to the project used are: UI, Framework, Indexing and search technology, Plug-in management and Task management. We mention next for each project the characteristic that is present: JDT (UI, IndexSearch), PDE (UI, PlugIn), EQ (UI, Framework), LU c(UI, IndexSearch), MY (UI, Task). Table 2 details for each project if the characteristic is present (Y) or not (N).

Table 2: Characteristics of investigated Projects

Projects	**Characteristics of projects**				
	UI	***Framework***	***Search***	***Plug-in***	***Task***
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**

3.4 Experiments description

Our conducted investigation used various experiments in five projects. Each experiment builds a neural network bug-oriented reliability prediction model considering one of the five projects as training. A graphical view of our experiments descriptions are provided in Figure 2.

In each experiment, we trained a neural network-based prediction model using 9/10 data from a single dataset (each experiment used a different dataset for training). Each prediction model was then validated in two steps. The first step was to validate it using the cross-validation technique [7]), 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 from the dataset.

Thus, the within-project validation was first applied; in Figure 2 Project A is considered both for training and testing. As a second part of each experiment, the cross-project validation was performed, i.e., using the other four projects from the dataset for validating the model, for example, for JDT-training PDE, EQ, LU and MY projects are used in turn as Project B in Figure 2.

As marked in Figure 2, Project A was both used as training and validating, instantiated in turn with JDT, PDE, EQ, LU and MY. In the same experiment, for the same Project A considered as training, the other four projects were considered for testing, thus Project B being initialized in turn with the other projects.

4 RESULTS

After conducted the above-mentioned experiments, we scrutinized the results to find out if the type of project used for the training of the prediction model has an impact on the obtained RMSE values.

The experiments investigated to what extend the obtained neural network model for reliability prediction is different in terms of the Root Mean Squared Error (RMSE) value (thus better) when a different type of project is used for training varies. The lower this RMSE value is, the better the model is in its predictions. If you have a smaller value, this means that the predicted values are closer to the observed values.

Table 3 contains the RMSE values for all experiments, considering each project as a training project, for both within-project and cross-project validation. Thus, for each project used as training project, the RMSE values for the withing-project validation is listed (values

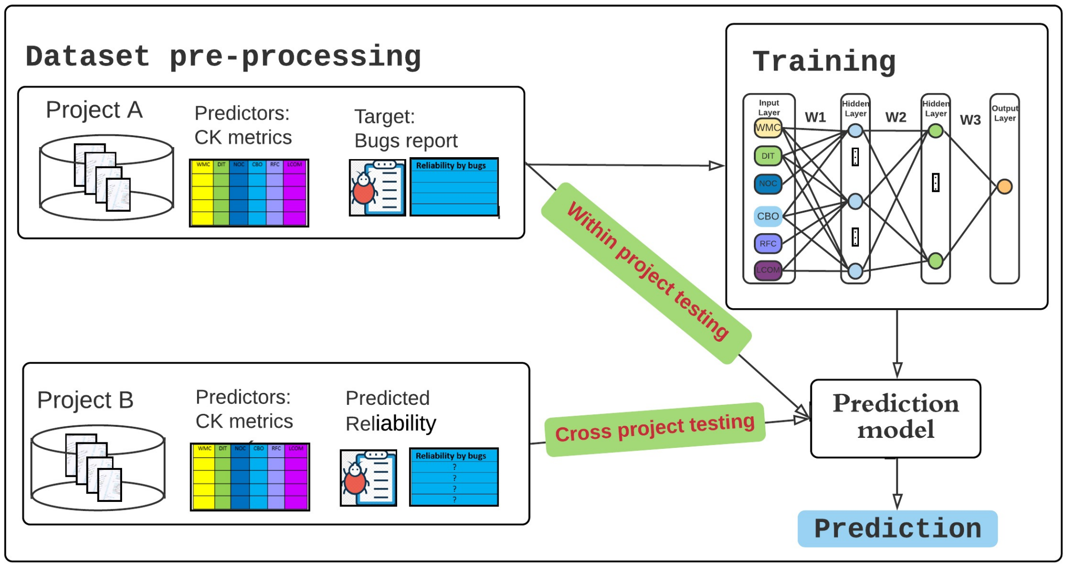

Figure 2: Our approach overview

on the principal diagonal in the table) and also the RMSE values for the cross-project validation (for the JDT-training the values in the second column, one value for each PDE, EQ, LU, My, except JDT).

Scrutinizing the result in Table 3, we can observe that:

- For within-project – the PDE-training obtained the minimum value (bold in the table, among all other each training project).
- For cross-project – the minimum RMSE values for each validation set for the cross-project experiments is marked with gray background: for all projects (except LU-training and MY) the minimum RMSE obtained is for MY. For LU-training the minimum RMSE is obtained for EQ and for MY-training the minimum RMSE is obtained for PDE project.

Scanning the values for both within and cross-project, we note that for all projects (except LU) the value for the within-project is smallest than the cross-project validation values. Thus, in what follows we will analyze thoroughly the obtained cross-project values.

The beanplot, an enhancement of the boxplot that adds a rug showing every value and a line showing the mean, was introduced by Kampstra [11]. The name is inspired by the appearance of the plot: the shape of the density looks like the outside of a bean pod and the rug plot looks like the seeds within.

Figure 3 presents the beanplot for the cross-project validation experiment. Examining the figure, we discern that the RMSE mean for EQ is the highest among all projects. The JDT project had the largest number of similar RMSE values for all its validations projects, whereas PDE and MY have the smallest RMSE values.

In Figure 4 we present a radar plot to emphasize the relationships between the project used as training and the projects used as validation. Analyzing the results, it can be seen that the obtained RMSE values may differ depending on which project is used as a basis to train the model. Best RMSE is obtained for PDE-training and MY-training, thus with PlugIn and Task characteristics. The worst value

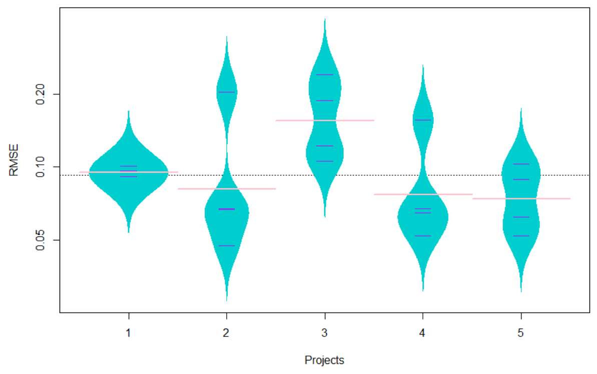

Figure 3: RMSE - beanplot for all projects (1-JDT, 2-PDE, 3-EQ, 4-LU, 5-MY) for cross-project experiment

is obtained with EQ project, thus using a project with framework characteristics as training. In conclusion, PlugIn-training finds best RMSE for a Task-oriented project and a Framework-training project finds the worst RMSE for a Task-oriented project.

The experiments conducted in this paper enabled us to state the following observations:

- One aspect for reliability prediction based on code metrics is considered in this investigation: bugs with various flavors (high priority bugs, non-trivial bugs, major bugs, critical bugs); other aspects for reliability need to be considered for a deeper understanding, aspects such as changes (number of revisions, number of authors, number of fixies, etc);

Table 3: RMSE values for within-project and cross-project experiments

Projects	RMSE values				
	JDT-training	***PDE-training***	***EQ-training***	***LU-training***	***MY-training***
JDT	0.087380777	0.204252533	0.188846308	0.156155316	0.089221831
PDE	0.100760247	**0.043892742**	0.122337023	0.064645116	0.051891154
EQ	0.091224966	0.066800726	0.071841441	0.052064477	0.062254302
LU	0.096635928	0.067367806	0.240844976	0.072722731	0.103173017
MY	0.091181044	0.047469571	0.105609546	0.067368627	0.044976185

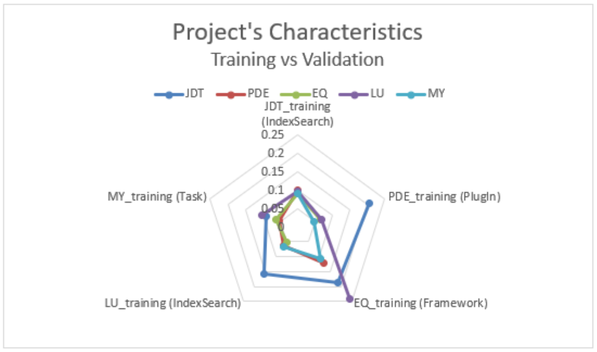

Figure 4: Projects's Characteristics - relation between training and validation for cross-project validation.

- Various prediction models were compared using cross-project validation and only one model per trained project for within-project validation; it would be interesting to investigate various prediction models for the same project using various releases.

In summary, with respect to our *RQ*, namely *What project (i.e characteristics) should be used as training for better bugs-oriented reliability prediction?*, we elaborate the following response:

The experiments using within-project and cross-project prediction models identified that the best reliability by bugs prediction is obtained using PlugIn-based characteristic training project for Task-based characteristic project.

Comparison: Multiple regression model versus Neural Network model

In what follows, we compare a multiple regression approach [29] with the current proposed neural network approach, both using the same dataset.

The approach in [29] investigated the same reliability prediction using CK metrics with the same bug-based definition of reliability as in this current paper approach, but considering the multiple regression approach. Three models encapsulating various CK metrics were investigated and the best one revealed that the equation with CBO and WMC metrics predicts better.

The same type of experiments were conducted, i.e., considering one of the five projects as training and the others four as validation. The RMSE values were also reported for the multiple regression approach. We should mention that in the [29] approach only four initial CK values were used after conducting the preliminary analysis (Pearson correlation between independent variables and dependent variable and multicoliniarity - correlation between any two metrics). Thus, in what follows we present the analysis between the two approaches, however we should keep in mind this difference: the [29] approach used only four metrics (reduced to two CK metrics by experiments) and the current approach uses all six CK metrics.

The obtained results in the multiple regression approach [29] showed that using MY as training provides the best prediction (for the Lucene project as validation). In the current paper, i.e. using Neural Network approach, the best training project was the PDE with the best RMSE value obtained for the MY validation project.

An important remark regarding both approaches is that for all training-based projects (except for the current approach with LU project) the project that obtained the minimum RMSE values is the MY project. Thus, having as a validation a project with *Task management* characteristic, we obtained the best prediction approach no matter what characteristics had the training project.

The current approach also conducted experiments within project testing in contrast to the multiple regression approach. The within-project testing results showed that the PDE project obtained the best prediction model, as showed in Table 3.

5 LONGER-TERM OBJECTIVES AND PLANNED WORK

Emerging from the above two observations, we plan to further refine the work done in this paper to firstly consider other aspects of reliability prediction, secondly to experiment with various releases of the same project, and thirdly to investigate with further experiments different weights for the considered bugs.

Considering other reliability aspects. The used dataset [5] contains also historical data, such as versions, fixes and authors, refactorings made data that could be used further in the reliability estimation model. Thus, future experiments will investigate an aggregated metric for assessing reliability, taking into account others aspects that influence reliability such as changes made during the entire system lifecycle.

Considering various releases of the same project. Conducting within-project experiments for multiple releases should provide a better model than considering training and testing for projects with different characteristics.

Considering various weights for various types of bugs. It is worth mentioning that establishing the weights based on priority in solving the faults and then considering their severity, it is our first attempt to propose a metric for reliability estimation. Further investigation will follow, conducting various experiments, employing different weights for diverse types of bugs for a better reliability estimation metric, such as, for example, the design method of Taguchi [8], [22].

6 THREATS TO VALIDITY

The proposed approach for reliability prediction, as every experimental analysis, may suffer from some threats to validity and biases that can affect the results of our study. There are several issues which may have influenced the obtained results and their analysis are pointed out in the following.

Threats to internal validity refer to the subjectivity introduced in setting the weights for the reliability estimation equation. To minimize threats to internal validity, we reason the weight establishment by the fact that bugs that are categorized with high severity and high priority, influence in a greater extent the system reliability.

Threats to external validity are related to the generalization of the obtained results. Only five open-source projects were considered for evaluation, written in the same programming language (Java) and considering a single version. Another threat to external validity refers to the bias between the set of bugs categorized by priority and severity and the set of bugs, as the results obtained on a biased dataset are less generalizable. We tried to reduce threats to external validity by choosing projects of different sizes and types and using information extracted from a bug tracking system. We plan to extend the experimental evaluation to others large-scale software systems, analyzing several versions of the project, and also using others information regarding the software changes that could influence reliability.

Construct validity refers to check if the proposed construct is real and if the proposed indicators reflect its target construct. Intentional validity, if the constructs we chose adequately represent what we intend to study: we wanted to study the prediction of defect classes that are represented by the bugs it may contain. Regarding representation validity, i.e. how well do the constructs or abstractions translate into observable measures, we argue that the different categories of bugs were considered with various weights in the proposed reliability metric. Thus, the sub-constructs define the construct. Observation Validity, i.e. how good are the measures themselves, the values of the metrics used in our investigation come from the last version of the projects, before release, thus the bugs were found in the analysed systems during a period of six months.

Threats to conclusion validity are related to the relationship between treatment and outcome and are mitigated through ensuring that experimental assumptions are valid. In this investigation, the conclusion threats are mitigated by classifying the bugs based on their quality impact degree on the software system being investigated and using the same bug categories used in software development and studies (high priority bugs, non trivial bugs, major bugs, critical bugs, and those bugs that are not categorized). Further more, we only used the measurements that come from well-known sources, other research investigations that provide the metrics values.

7 CONCLUSIONS AND FUTURE WORK

The paper proposes an approach for predicting bug-oriented reliability using as pillars two perspectives: first, estimating the reliability using the number of bugs in the system and, second, using CK metrics for prediction. The obtained model is validated using a dateset containing over 5000 instances/classes, grouped in 5 projects. The experiments revealed that a PlugIn based characteristic project obtains the “best” neural network reliability prediction model.

Future work will investigate: applying the neural network prediction model for other quality attributes, overcoming the limitation of the subjectively established weights of the reliability equation by using supervised learning methods, and extending the experimental evaluation considering other open source projects.

REFERENCES

[1] Victor Basili and D. Rombach. 1988. The TAME project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14, 6 (June 1988), 758–773.

[2] Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22, 10 (Oct 1996), 751–761. https://doi.org/10.1109/32.544352

[3] S.R. Chidamber and Chris F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (June 1994), 476–493. https://doi.org/10.1109/32.295895

[4] S. Chitra, K. Thiagarajan, and M. Rajaram. 2008. Data collection and analysis for the reliability prediction and estimation of a safety critical system using AIRS. In *2008 International Conference on Computing, Communication and Networking*. 1–7. https://doi.org/10.1109/ICCCNET.2008.4787675

[5] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 31–41. https://doi.org/10.1109/MSR.2010.5463279

[6] Equinox. Online; accessed 29 Feb 2019. Eclipse Equinox. https://projects.eclipse.org/projects/eclipse.equinox

[7] Prashant Gupta. Online; accessed 17 Feb 2019. Cross-Validation in Machine Learning. https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f

[8] Mei-Ling Huang, Yung-Hsiang Hung, and Zih-Sian Yang. 2016. Validation of a method using Taguchi, response surface, neural network, and genetic algorithm. *Measurement* 94 (2016), 284–294.

[9] ISO25010. 2019. ISO25010 description information, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. https://www.iso.org/standard/35733.html. [Online; accessed 30-May-2019].

[10] JDT. Online; accessed 29 Feb 2019. JDT Core Component. https://www.eclipse.org/jdt/core/index.php

[11] Peter Kampstra. 2008. Beanplot: A Boxplot Alternative for Visual Comparison of Distributions. *Journal of Statistical Software, Code Snippets* 28, 1 (2008), 1–9. https://doi.org/10.18637/jss.v028.c01

[12] Barbara Kitchenham, Shari L. Pfleeger, and Norman E. Fenton. 1995. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering* 21, 12 (Dec 1995), 929–944. https://doi.org/10.1109/32.489070

[13] Wei Li. 1998. Another metric suite for object-oriented programming. *Journal of Systems and Software* 44, 2 (1998), 155 – 162. https://doi.org/10.1016/S0164-1212(98)10052-3

[14] Xiang Li, Chetan Mutha, and Carol S. Smidts. 2016. An Automated Software Reliability Prediction System for Safety Critical Software. *Empirical Softw. Engg.* 21, 6 (Dec. 2016), 2413–2455. https://doi.org/10.1007/s10664-015-9412-6

[15] Jungang Lou, Yunliang Jiang, Qing Shen, Zhangguo Shen, Zhen Wang, and Ruiqin Wang. 2016. Software Reliability Prediction via Relevance Vector Regression. *Neurocomput.* 186, C (April 2016), 66–73. https://doi.org/10.1016/j.neucom.2015.12.077

[16] Lucene. Online; accessed 29 Feb 2019. Lucene. http://lucene.apache.org/

[17] R. Marinescu. 2002. Measurement and Quality in Object Oriented Design. PhD Thesis, Faculty of Automatics and Computer Science, University of Timisoara.

[18] J. Merseguer. 2003. *Software Performance Engineering based on UML and Petri nets.* Ph.D. Dissertation. University of Zaragoza, Spain.

[19] Alok Mishra and Ziadoon Otaiwi. 2020. DevOps and software quality: A systematic mapping. *Computer Science Review* 38 (2020), 100308. https://doi.org/10.

1016/j.cosrev.2020.100308
[20] John D. Musa. 1998. *Software Reliability Engineering.* McGraw-Hill.
[21] Mylyn. Online; accessed 29 Feb 2019. Mylyn.
[22] Kwan Ouyang, Horng Wen Wu, Shun-Chieh Huang, and Sheng-Ju Wu. 2017. Optimum parameter design for performance of methanol steam reformer combining Taguchi method with artificial neural network and genetic algorithm. *Energy* 138 (2017), 446–458.
[23] PDE. Online; accessed 29 Feb 2019. PDE UI. https://www.eclipse.org/pde/pde-ui
[24] Pulasthi Perera, Roshali Silva, and Indika Perera. 2017. Improve software quality through practicing DevOps. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer).* 1–6. https://doi.org/10.1109/ICTER.2017.8257807
[25] A. Quyoum, UdM. Din Dar, and S.M.K. Quadr. 2010. Improving software reliability using software engineering approach—a review. *Int. J. Comput. Appl.* 10, 5 (2010), 41 – 47.
[26] S. Russel and P. Norvig. 1995. *Artificial intelligence: a modern approach.* Englewood Cliffs, N.J. : Prentice Hall.
[27] N. F. Schneidewind. 1997. Reliability modeling for safety-critical software. *IEEE Transactions on Reliability* 46, 1 (March 1997), 88–98. https://doi.org/10.1109/24.589933
[28] Camelia Serban and Horia F Pop. 2008. Software quality assessment using a fuzzy clustering approach. *Studia Universitas Babes-Bolyai, Seria Informatica* 53, 2 (2008), 27–38.
[29] Camelia Serban and Andreea Vescan. 2019. Predicting reliability by severity and priority of defects. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software Qualities and Their Dependencies.* 27–34. https://doi.org/10.1145/3340495.3342753
[30] Ian Sommerville. 2018. *Software Engineering.* Pearson India; 10th edition.
[31] Mei-Huei Tang, Ming-Hun Kao, and Mei-Hwa Chen. 1999. An empirical study on object-oriented metrics. In *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403).* 242–249.