ISSN 2047-3338

# An Improved Model for Software Defect Prediction in Concurrent Software Systems

FARINRE, J. Moyosore[1], ABIOLA, O. Arowolo[2] and AKINOLA, S. Olalekan[3]
[1,2,3]University of Ibadan, Ibadan, Nigeria

***Abstract*– Defects found after the development of software has made researchers develop different types of software defect prediction model. Most concurrent software has defects in them that makes them to give incorrect results. This study provides a model that predict defect in concurrent software system so as to make the software system to produce a correct and expected results. Deep learning classifier Convolutional Neural Network (CNN) was employed to build the model and its performance was compared with an existing model that also predicts defect in concurrent software systems. The model combined Static and Dynamic metric and Convolutional Neural Network (CNN) for predicting defects in concurrent software system.**

***Index Terms*– Concurrent Programs, Feature Selection, Convolutional Neural Network and Defect Prediction**

## I. INTRODUCTION

A software defect is an error, flaw, failure or fault in a computer program or system that produces an incorrect or unexpected result, or to behave in unintended ways (Parameswari, 2015). A defect indicates the unexpected behavior of system for some given requirements. The unexpected behavior is identified during software testing and marked as a defect. Timely identification of software bugs facilitates the testing resources allocation in an efficient manner and enables developers to improve the architectural design of a system by identifying the high-risk segments of the system (Menzies *et. al*., 2017).

Most defects arise from mistakes and errors made in either a program's source code or its design, or in components and operating systems used by such programs. A few are caused by compilers producing incorrect code. A program that contains a large number of defects, and/or defects that seriously interfere with its functionality, is said to be buggy (defective). Defects can trigger errors that may have ripple effects (Bug Archived March 23, 2017, at the Wayback Machine).

This work is aimed at improving on predicting defects in concurrent software systems using deep learning technique. Static and dynamic metrics using Yu *et. al*., (2018) mathematical formula will be generated with an improved model for the prediction of software defects with deep learning techniques. The result of the research was evaluated against the existing model using accuracy, precision, recall and F1-Score.

## II. LITERATURE REVIEW

Developing fault-free reliable software is a daunting task in the current context, when software is being developed for problems with increasing difficulty with more and more complex problem domains (Vashisht *et. al*., 2015). Many Software Defection Prediction models have been developed to fix the problem of defects in software but none of these models have proven to give accurate results. Machine learning, deep learning and data mining algorithms have been used to develop a Software Defect Prediction Model.

Software defect prediction is a process of building classifiers to predict sections of codes that potentially contain defects, using information such as code complexity and change history. The prediction results (i.e., buggy code areas) can place warnings for code reviewers and allocate their efforts. The code areas could be files, changes or methods (Moser *et. al*., 2008). Machine Learning and Deep learning Algorithms that have been used for Software. Defect Prediction are Random Forest, Convolutional Neural Network (CNN), Logistic Regression, Naives Bayes, K-means, Decision Tree, Support Vector Machine (SVM), Linear Regression, K-Nearest Neighbors (KNN) etc. Deep Learning Algorithm is used because it automatically extracts features for users instead of having to select features one by one.

A Neural Network is a Deep Learning Technique and the term "deep" refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction. Example of the most popular Neural Networks are Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) (Pingel, 2017).

An Artificial Neural Network (ANN) is an information processing system which contains a large number of highly interconnected processing neurons. These neurons work together in a distributed manner to learn from the input information, to coordinate internal processing, and to optimize its final output. Fig. 1 shows an Artificial Neural Network, which is organized in layers.

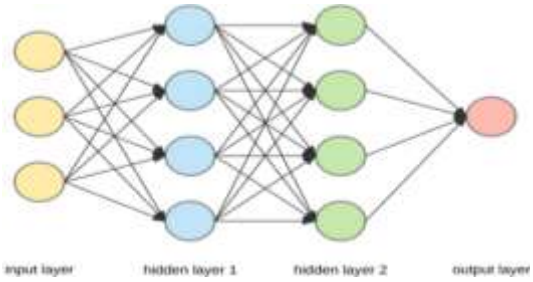


Fig. 1: A Neural network organized in layers (Sorokina, 2017)

A Convolutional Neural Network (CNN) is an example of a deep learning technique. It is a special kind of neural network for processing data that have a known grid-like topology (Goodfellow *et. al.,* , 2016), such as time-series data in 1-D grid and image data in 2-D grid. CNN have been proven to be successful in many areas such as image classification, speech recognition, etc. Fig. 2 shows an image of a Convolutional Neural Network (CNN).

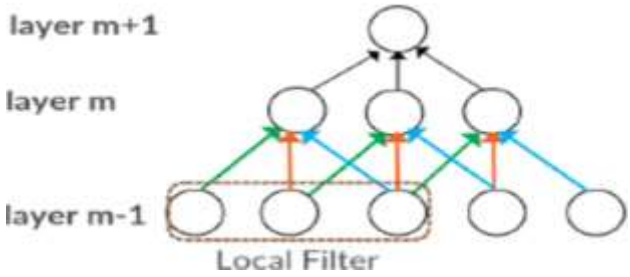


Fig. 2: An image of a Convolutional Neural Network (CNN) (Jianli *et. al*., 2017)

Concurrent computation makes programming much more complex (Cartwright, 2000). In a concurrent program, several streams of operations may execute concurrently. Each stream of operations executes as it would in a sequential program except for the fact that streams can communicate and interfere with one another. Each such sequence of instructions is called a thread. For this reason, sequential programs are often called single-threaded programs.

When a multi-threaded program executes, the operations in its various threads are interleaved in an unpredictable order subject to the constraints imposed by explicit synchronization operations that may be embedded in the code. The operations for each stream are strictly ordered, but the interleaving of operations from a collection of streams is undetermined and depends on the vagaries of a particular execution of the program. One stream may run very fast while another does not run at all, a given thread can starve unless it is the only "runnable" thread (Cartwright, 2000). Fig. 3 shows how a concurrent program works, with three threads running.

*A) Related Works*

Software defect prediction is the process of locating defective modules in software. To produce high quality software, the final product should have as few defects as

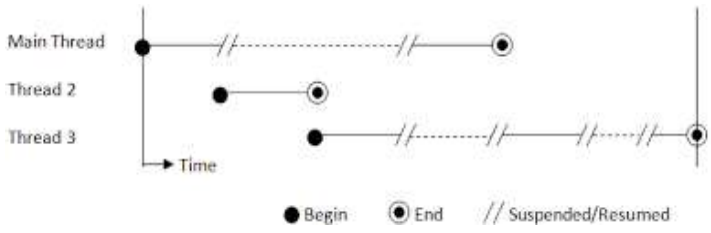


Fig. 3: Image that shows how a concurrent program works (Java Prog. Tutorial, 2020)

possible. Early detection of software defects could lead to reduced development costs and rework effort and more reliable software. So, the study of the defect prediction is important to achieve software quality. The most discussed problem is software defect prediction in the field of software quality and software reliability. Boehm and Basili (2001) observed, finding and fixing a problem after delivery is 100 times more expensive than fixing it during requirement and design phase. Additionally software projects spend 40 to 50 percent of their efforts in avoidable rework. Therefore, defect prediction is extremely essential in the field of software quality and software reliability. Defect prediction is comparatively a novel research area of software quality engineering.

There has been much research on developing various software metrics and prediction algorithms to assess software quality. For example, Lee *et. al*., (2011) proposed a set of micro-interaction metrics (MIMs) that leverage developers' interaction information combined with source code metrics to predict defects. Meneely *et. al.*, (2008) examine structure of developer collaboration and use developer network derived from code information to predict defects. Basili *et. al.*, (2008) used Chidamber and Kemerer metrics, and Ohlsson and Alberg (1996) used McCabe's cyclomatic complexity for defect prediction. Menzies *et. al.*, (2010) conclude that static code metrics are useful in predicting defects under specific learning methods. These techniques, however, focus on sequential programs while ignoring code attributes and testability for concurrent programs.

Another work found is the work by Mathews and Tu, (2010) based on Ada programs. This work considers only the number of synchronizations and conditional branches that contain synchronization points without utilizing them to perform defect prediction. However, this work does not aim to predict concurrency faults. In addition, the work does not consider dynamic metrics.

Jianli *et. al.* (2017) worked on using Convolutional Neural Network to predict defects in software. This work focused only on file level prediction. They claim that programs have well-defined syntaxes and rich semantics hidden in the Abstract Syntax Trees (ASTs), which traditional features often fail to capture, and the traditional methods does not really give satisfactory results. Jianli *et. al.* used Convolutional Neural Network combined with Traditional Defect prediction for predicting defects because it has been proven to automatically generate features since it can effectively capture highly complicated non-linear features.

Yu *et. al*., (2018) developed a model with a combination of derived metric sets to improve prediction of defects in

concurrent software programs. Yu *et. al*., (2018) named their model the ConPredictor. The ConPredictor prediction model was built upon all 24 static and dynamic concurrency metrics. They empirically compare the performance of ConPredictor to those of prediction models built using traditional metrics that have been widely used in previous fault prediction work. They also investigate whether the combined use of mutation metrics and source code metrics improve the accuracy of the resulting prediction model. However, there were few cases where an instance (i.e., function) is mislabeled as non-faulty. This is because a concurrency fault may involve more than one function because the conflicting shared variables may exist in different functions. A function labeled as non-faulty may have conflicting accesses with other functions that are labeled as faulty.

This present study proposes an improved ConPredictor model for software defect prediction in concurrent program using deep learning technique. This research work improves on the existing model of Yu *et. al.,* (2018).

## III. METHODOLOGY

The research work was designed in such a way as to allow for the collection and analysis of defect data from a standard data source. The data was collected with the aim of identifying defects using deep learning techniques (Convolutional Neural Network). The dataset used for the analysis was the one used by the ConPredictor model. Tyu *et. al.,* (2018) provide a link (http://cs.uky.edu/_tyu/ConPredictor) to where we can access the dataset so has to use for further research work.

The data used in this study was gotten from the link where four large concurrent software projects (Apache, MySQL, Mozilla, and OpenOffice) were used to build the model. The module that had concurrent program properties were selected to identify which particular module has concurrent type of defects. The dataset was divided into train and test sets and the model was trained using Convolutional Neural Network (CNN). The number of defects found in the dataset were counted and recorded. These data sets were selected because with millions of lines of publicly accessible code and well-maintained bug repositories, they have been widely used by existing bug characteristic studies (Jin *et. al.,* 2012), and concurrency fault detection and testing techniques (Deng *et. al.,* 2013).

The proposed methodology for the study is shown in Fig. 4. The proposed methodology classifies the selected software modules into defective and non-defective categories. Software defect prediction helps to identify defective modules in software, and it reduces the cost of having to fix defects after a software has been delivered to customer.

### *A) Nature/Type of Defect Found in the Software Project*

The types of defects found in the software projects were the usual concurrency defects found in concurrent software. These defects are Live lock, Deadlocks, Hold and wait (Resource Holding), Starvation. In this study, only these four defects were considered. The model predicts functions that are likely to contain concurrency defects in real-world applications.

Specifically, six novel code metrics specific to concurrent programs were proposed. Concurrency control flow graph (CCFG) was adapted to generate code metrics involving: Access Point of Public Thread Variables (APPTV), Number of Concurrent Conditionals (NCC), Edge Communication value in CCFG (EC-CCFG), Concurrent Cyclomatic Number (CCN), Number of Public Variables (NPV) and Number of Interleaved Operations (NIO) (Yu *et. al.,*2018).

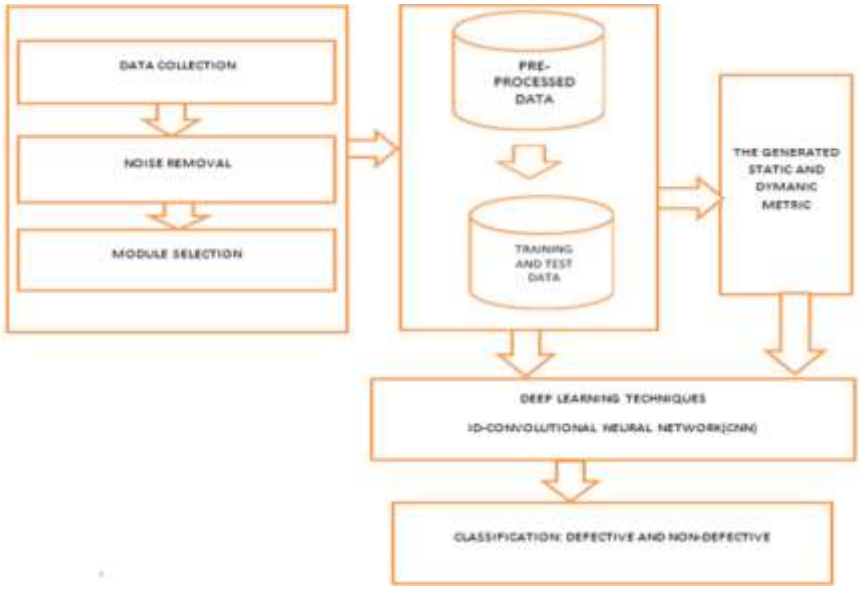


Fig. 4: The proposed methodology

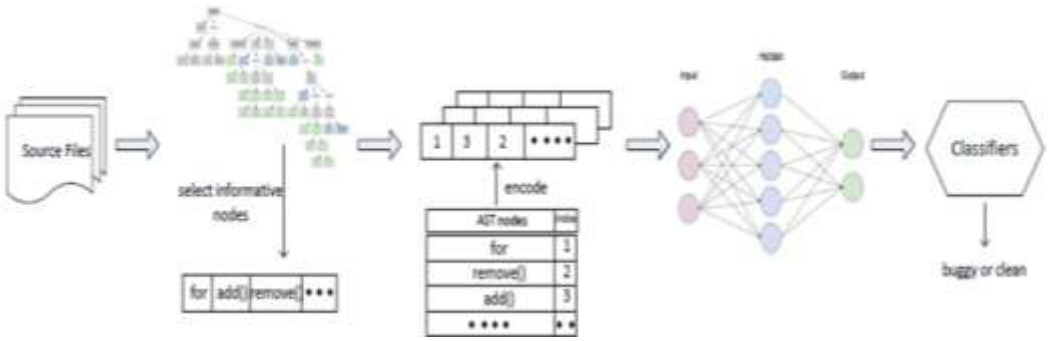


Fig. 5. The overall workflow of CNN (Li *et. al.*, 2017)

Eighteen mutation metrics were defined and computed by applying a variety of mutation operators specific to concurrent programs. These metrics include six static metrics (e.g., the number of times a mutation operator is applied) and 12 dynamic metrics from dynamic mutation analysis. The proposed model was built on these metrics i.e., the six derived and eighteen mutation metrics.

*B) Using Convolutional Neural Network (CNN) to Build the Improved Model*

Convolutional Neural Network (CNN) is primarily used to classify images. So, there is need to convert our datasets to vectors so that the CNN can take as input. The Source code (datasets) is parsed into Abstract Syntax Trees (ASTs) to form token vectors. Thus, each source file is represented by a token vector. Then conduct mapping and word embedding converts the token vectors into numerical vectors and input the numerical vectors to CNN. CNN will automatically generate features and classify the dataset into defective and non-defective. The overall workflow of CNN (Li *et. al.*, 2017) is depicted in Fig. 5.

The deep learning classifier (Convolutional Neural Network) was used to make the model learn from the Training set. The model is tested on the Test set and the performance measures were calculated. Hence, the dimensionality of the data was first reduced to a set of 6 cumulated features using 4 different techniques and then trained the model using Deep learning classifier. A detailed comparison was then made based on the performance metrics that include Accuracy, Precision, Recall and F1-Scores. Anaconda was used as the development environment alongside Jupyter which is a launch tool for scientific analysis using Python in the Anaconda environment.

Multiple network architectures were implemented with different configurations for the auto encoder and the neural network. Once we carried out the tests, it was found that the best configuration for the model was the following: 64 hidden neurons, 0.9 of learning rate, 0.1 for corruption level, and 5000 iterations of training. In the case of the deep leaning configuration, the best result was adjusted with the following combination of values: 4 hidden layers, 0.3 of learning rate, 0.9 of momentum factor and 1000 iterations of training.

Our model was applied on the four datasets that were collected, defective instances were predicted and instances useful in the dataset were selected.

## IV. RESULTS AND PERFORMANCE EVALUATIONS

Table I shows the defective instances found in the four datasets. In the first dataset, 76 instances were found to be defective. In the second dataset, 300 instances were found to be defective. In the third dataset, 138 instances were found to be defective and 115 instances were found to be defective in the last dataset. The results of the Accuracy, Precision, F – Score and Recall of the Model is presented in Table II. From the Table, it is shown that that the average accuracy for the four datasets was 75.4% while the precision of the model was 73.9%.

*A) Result Comparison*

Comparing the performance of our Improved Model with the ConPredictor model proposed by Tyu *et. al.*, , (2018), Table 3 and Fig. 6 show the results obtained. Tyu *et. al.*, , (2018). ConPredictor model analysis was done using WEKA as the tool for analysis. We adopted the same data and metrics used by Tyru *et. al.* (2018) but our model was implemented in Python development environment. From Table 3, it can be deduced that our Improved Model outperforms the ConPredictor in all the four-performance metrics used.

Table I: Numbers of defects found in each software projects

| Software Project | Total Instances | Selected Instances | Defective Instances |
|---|---|---|---|
| Dataset 1 | 35761 | 2000 | 76 |
| Dataset 2 | 71665 | 2145 | 300 |
| Dataset 3 | 144382 | 4700 | 158 |
| Dataset 4 | 110509 | 6139 | 120 |

Table II: Result of the Accuracy, Precision, F – Score and Recall for the Model

| DATASET | ACCURACY | PRECISION | RECALL | F-SCORE |
|---|---|---|---|---|
| Dataset 1 | 75.2% | 72.5% | 74.3% | 73.4% |
| Dataset 2 | 74.5% | 73.4% | 72.5% | 73.0% |
| Dataset 3 | 76.6% | 75.7% | 73.2% | 74.4% |
| Dataset 4 | 75.4% | 73.9% | 73.3% | 73.6% |
| AVERAGE | 75.4% | 73.8% | 73.3% | 73.6% |

Table III: Comparison Results of ConPredictor against the Improved Model

| | ConPredictor | Improved Model |
|---|---|---|
| Accuracy | 72.0% | 75.4% |
| Precision | 68.0% | 73.8% |
| Recall | 64.0% | 73.3% |
| F1-Score | 66.0% | 73.6% |

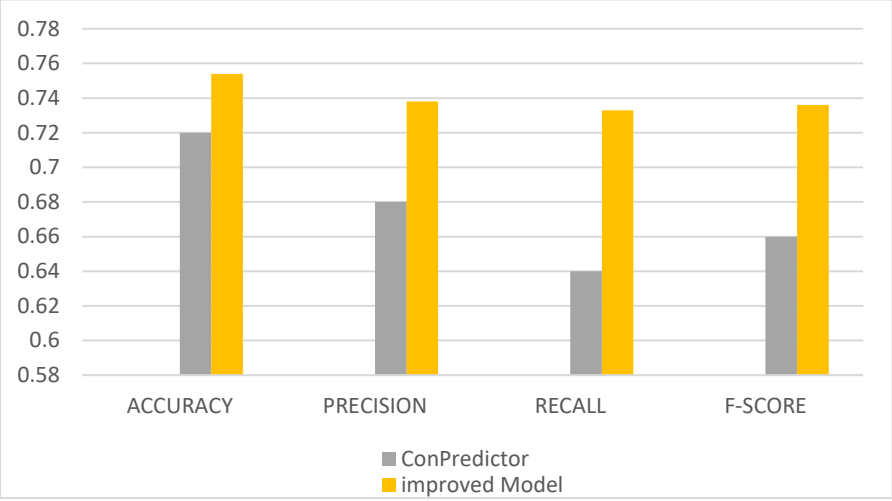


Fig. 6: Graph of Result Comparison

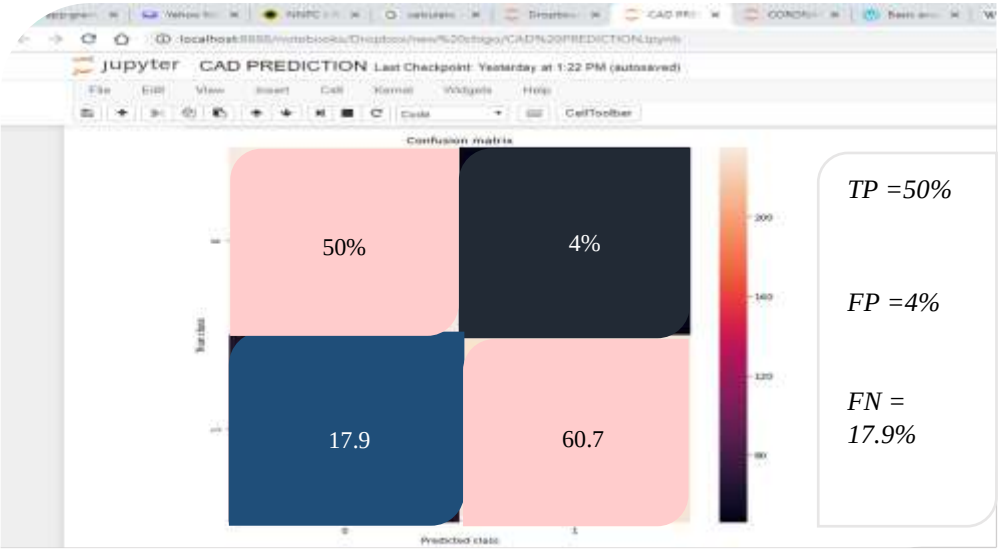


Fig. 7: Confusion matrix

*B) Confusion Matrix*

A confusion matrix, also known as an error matrix, is a two by two table layout that contains four outcomes produced by a binary classifier to visualize the performance of the applied algorithm. It reports the number of *false positives (FP)*, *false negatives (FN)*, *true positives (TP)*, and *true negatives (TN)*. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.

Fig. 7 shows the confusion matrix of the performance of our improved model. It shows that the true positive (TP) "correct positive prediction" covers 50% of the entire dataset while the false positive (FP) "incorrect positive prediction" is covers only 4% of the entire dataset. However, the true positive (TN) "correct negative prediction" covers 17.9% of the entire dataset while the false negative (FN) "incorrect negative prediction" covers 60.7%.

*C) Discussion of Results*

From the results presented, it was demonstrated that the improved model reduced the number of false positive in the datasets compared to the ConPredictor model. For the improved model, the accuracy was higher because it predicted more defects in each dataset than the ConPredictor model predicted. In the first dataset, the improved model predicted 6 defective instances more than the ConPredictor model. In the second dataset the improved model predicted 70 defective instances more than the ConPredictor model. In the third dataset, the improved model predicted 16 defective instances more than the ConPredictor model. In the fourth dataset the improved model predicted 4 defective instances more than the ConPredictor model.

The improved model accuracy was 3.4% higher than the ConPredictor model, the improved model precision was 5.8% higher than the ConPredictor model. The improved model recall was 9.3% higher than the ConPredictor model, the improved model f1-score was 7.6% higher than the ConPredictor model. Therefore, the improved model gives a slightly better performance in all the performance measures than the ConPredictor model.

## V. CONCLUSION

Different software prediction models have been developed but none has been accurate in predicting defects in concurrent software systems. Convolutional Neural Network (CNN) was used to build the improved model and static and dynamic metric was generated to help predict defects better in concurrent software system. The CNN was used because it has been established that it builds better prediction model and because of the unpredictable and non-deterministic properties of concurrent software system; Convolutional Neural Network (CNN) has been proven to work well for these kinds of trends.

In this study, Convolutional Neural Network (CNN) was used to build an improved model while static and dynamic metrics were generated to help predict defects better in concurrent software system. The performance of the model was evaluated against the existing model using performance measures like accuracy, precision, recall and f1-score. This model developed could be improved upon by using other deep learning algorithms to build a prediction model.

## REFERENCES


Basili V. R, Briand L. C, and Melo W. L (2008). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, Vol. 22, Number 10, pp: 751–761.

Boehm, B. W. and Basili, V. R. (2001). Software Defect Reduction Top 10 List, *IEEE Computer*, pp: 135-137.

Cartwright, R.C. (2000). Notes on Object Oriented Program Design. https://www.cs.rice.edu/~cork/book/newBook.html. Retrieved June 2019.

Deng D, Zhang W, and Lu S (2013). Efficient concurrency-bug detection across inputs. In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, pp:785–802.

Goodfellow I, Bengio Y, and Courville A (2016), Deep Learning. MIT Press, http://www.deeplearningbook.org.

Java Programming Tutorial, Multithreading & Concurrent Programming. https://www.ntu.edu.sg/home/ehchua/programming/java/J5e_multithreading.html, accessed June 2020.

Jian Li, Pinjia He, Jieming Zhu, and Michael R. Lyu (2017), Software Defect Prediction via Convolutional Neural Network. Department of Computer Science and Engineering, The Chinese University of Hong Kong, China. IEEE International Conference on Software Quality, Reliability and Security.

Jin G, Song L, Shi X, Scherpelz J, and Lu S (2012). Understanding and detecting real-world performance bugs. In PLDI, pp: 77–88.

Li J, He P, Zhu J, and Lyu M, (2017). Software Defect Prediction via Convolutional Neural Network. IEEE International Conference on Software Quality, Reliability and Security, pp: 318-328.

Mathews M. E and Tu S. (2010). Metrics measuring control flow complexity in concurrent programs. *IEEE Transactions on Computers*, Vol 3, Issue 5, Page 5

Meneely A, Williams L, Snipes W, and Osborne J (2008). Predicting failures with developer networks and social network analysis. *In Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 13–23.

Menzies T, Greenwald J, and Frank A (2010). "Data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, Vol 33, Number 1, Pages 2–13.

Menzies T, Greenwald J, and Frank A (2017). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13.

Moser R, Pedrycz W, and Succi G (2008), "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in ICSE'08: Proc. of the International Conference on Software Engineering.

Ohlsson N and Alberg H (1996). Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, Vol 22, Number 12, Pages 886–894.

Parameswari A, (2015). Comparing Data Mining Techniques for Software Defect Prediction, *International Journal of Science and Engineering Research (IJ0SER*), Vol 3 Issue 5 May -2015.

Pingel J (2017). Machine Learning vs Deep Learning: What's the Difference? https://itbriefs.com.au. Accessed May 2019.

Sorokina Ksenia (2017). Image Classification with Convolutional Neural Networks, https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8, Accessed May 2020.

Vashisht V, Lal M, Sureshchandar G. S (2015). A Framework for Software Defect Prediction Using Neural Networks. Journal of Software Engineering and Applications, Vol. 8, 384-394.

Yu T, Wen W, Han X, and Hayes J (2018). ConPredictor: Concurrency Defect Prediction in Real-World Applications. In IEEE International Conference on Software Testing, Verification and Validation, pp: 168–179, 2018.