WJCMS

Wasit Journal of Computer and Mathematics Science

Journal Homepage: https://wjcm.uowasit.edu.iq/index.php/WJCM

e-ISSN: 2788-5879 p-ISSN: 2788-5879

Machine Learning Empowered Software Prediction System

Asst.Prof.Dr. Abdul Syukor Mohamad*

Faculty of Computer Science, Universiti Teknikal Malaysia Melaka (UTEM)/Malaysia

*Corresponding Author: Asst.Prof.Dr. Abdul Syukor Mohamad

DOI: https://doi.org/10.31185/wjcm.61
Received: May 2022; Accepted: July 2022; Available online: September 2022

ABSTRACT: Prediction of software defects is one of the most active study fields in software engineering today. Using a defect prediction model, a list of code prone to defects may be compiled. Using a defect prediction model, software may be made more reliable by identifying and discovering faults before or during the software enhancement process. Defect prediction will play an increasingly important role in the design process as the scope of software projects grows. Bugs or the number of bugs used to measure the performance of a defect prediction procedure are referred to as "bugs" in this context. Defect prediction models can incorporate a wide range of metrics, including source code and process measurements. Defects are determined using a variety of models. Using machine learning, the defect prediction model may be developed. Machine inclining in the second and third levels is dependent on the preparation and assessment of data (to break down model execution). Defect prediction models typically use 90 percent preparation information and 10 percent testing information. Improve prediction performance with the use of dynamic/semi-directed taking in, a machine learning approach. So that the results and conclusion may be sharply defined under many circumstances and factors, it is possible to establish a recreated domain to house the entire method. Computer-aided engineering (CAE) is being used to identify software defects in the context of neural networks. Neural network-based software fault prediction is compared to fuzzy logic fundamental results in this research paper. On numerous parameters, neural network training provides better and more effective outcomes, according to the recommended findings and outputs.

Keywords: Analysis of Software Modules, Fuzzy Systems and Software Defects, Software Defects Analytics

1. INTRODUCTION

An impartial, unbiased source of information regarding a product's quality and failure risk, software testing can be provided to users and/or sponsors. It is possible to undertake software testing as soon as executable software is available (even if it is only half developed). Software testing is typically influenced by the overall approach taken to software development.

To put it another way, the majority of testing occurs after the system requirements are developed and implemented in testable programs. Contrast this with an Agile methodology, where requirements are frequently written, coded, and tested simultaneously.

This cycle, testing and cursing, can occur regardless of how well-defined a Software Testing Life Cycle (STL) is implemented in your project or company.

To put it another way, if you don't finish the previous stage before moving on, then the following step isn't possible. While theoretically feasible, this isn't usually the case in the real world.

In the Software Testing Life Cycle, the initial phase is Requirement Analysis (STLC). What we're going to test and how we're going to do it are the focus of this phase of the quality assurance process (QA). It is the responsibility of the

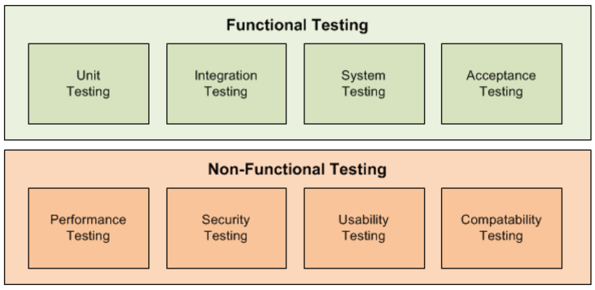


FIGURE 1. **Assorted Testing Strategies**

QA team to follow up with various stakeholders, such as Business Analysts, System Architects and Clients as well as technical managers and leaders in order to have a deeper understanding of the specifics of the requirements.

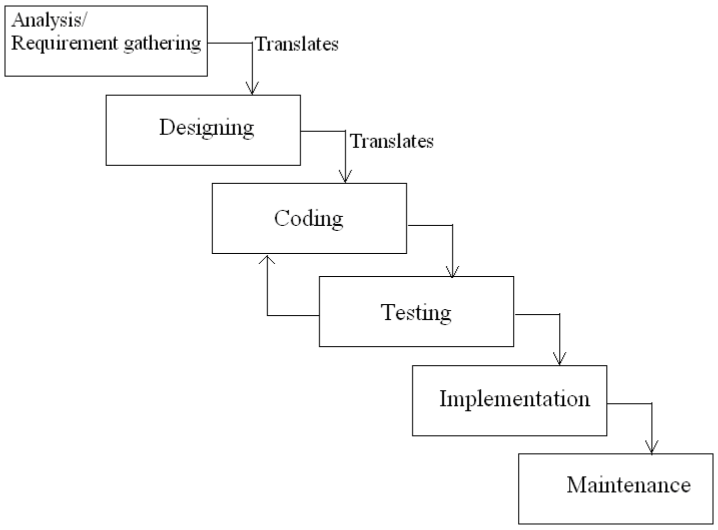


FIGURE 2. **Software Testing Life Cycle**

QA is involved in the STLC from the very beginning, helping to keep the software under test free of faults. Performance and security testing are examples of non-functional requirements. At this point, the project's requirements and automation feasibility may both be evaluated (if applicable)

The most critical part of the software testing life cycle is Test Planning, when every testing strategy is established. This stage is also known as the development of the test strategy. Typically, at this phase, the Test Manager (or Test Lead, depending on the firm) is engaged in determining the effort and cost estimates for the entire project. Once the requirement collection phase is complete and the requirement analysis has begun, the Test Plan preparation phase will be launched. The output of the test planning phase is a test strategy and plan, as well as an estimate of testing effort. The QA team can begin developing test cases once the design portion of the tests has been finished.

Once the test design phase is complete, the next step is to generate test cases. These thorough test cases are written out in this step of STLC. As well as preparing test cases, the testing team also gathers and organises the necessary test data for testing purposes. Peer members or the QA lead review the test cases after they are complete.

Another thing that's been completed is the Requirement Traceability Matrix (RTM). Each test case in the Need Traceability Matrix is mapped to a specific requirement, and it is widely acknowledged in the industry. Tracking backward and forward traceability is possible with this RTM.

The STLC would be incomplete without a discussion of how to properly set up a testing environment. In essence, the test environment dictates the conditions in which software is put to the test. This is a stand-alone task that may be begun concurrently with the development of test cases. No members of the testing team are involved in the process of creating a testing

environment. The testing environment may be created by the developer or the client, depending on the firm. Testing should begin with smoke tests to ensure the test environment is ready for use.

Execution of Tests - The test execution phase can begin when the preparations for Test Case Development and Test Environment setup have been finished. Based on the preceding step's prepared test planning and prepared test cases, in this phase the testing team begins running test cases.

2. THE STATEMENT OF PURPOSE AND RELATED PATTERNS OF RESEARCH

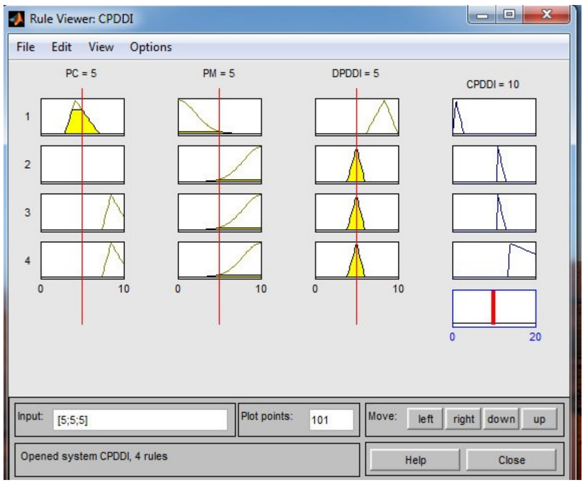
Imperfection is what is meant by the word "defect," and that is exactly what it is. It is a flaw if the actual results differ from the intended results or if the requirements are met incorrectly. Defect and bug, for example, are synonyms, but their meanings are distinct. When developing software, you'll hear a lot of these phrases. Bugs can be detected during the testing process, but if they are discovered after the program has been developed and rectified by the developers, they are referred to as defects.

3. IMPLEMENTATION AND SIMULATION PATTERNS

It's time to test the model's output for acceptability once the dataset has been properly trained and adequate learning epochs have been produced.

- Gradient and regression analysis parameters are used to assess the quality of the result.
- A figure that is as close to zero as possible while yet providing the lowest possible error rate

Prediction of software defects is an important area of software engineering that necessitates the use of cutting-edge techniques. Using proposed artificial neural networks, fuzzy modelling techniques can be enhanced.



Input: [5;5;5]
Plot points: 101
Opened system CPDDI, 4 rules

FIGURE 3. **Fuzzy Logic Rule Viewer for CPDDI**

The test case can be marked as Passed once it has been successfully completed. An error in any test case can be reported to the development team using a bug tracking system, which can then be associated with an error in the relevant test case for further investigation. There should be at least one problem in every test case that fails.

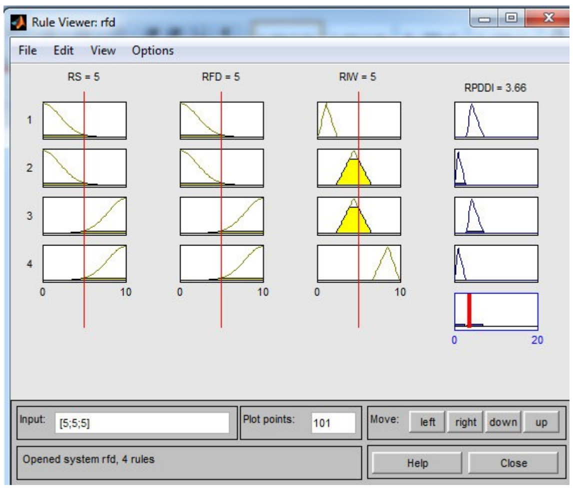
We can retrieve the failed test case and the problem that was connected with it thanks to this linkage. It is possible to use the same test case after a bug has been repaired by the development team.

Algorithms must be well-structured in order to be useful, and this is the primary purpose of ML. A defect prediction model is built using these. ML is a data-driven approach since it takes into account not just the temporal and spatial dimensions of data, but also the sheer volume of information. An important part of machine learning is determining whether a given dataset is utilised for training or testing.

Using MLT in SDP dramatically increases the quality of the program. SDP does not favour one machine approach over another for enhancing quality.

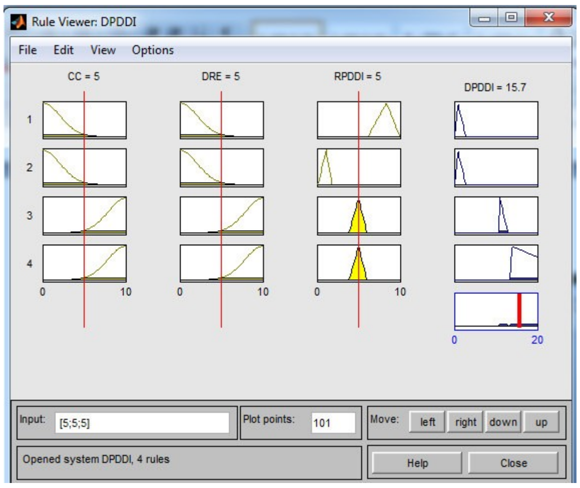
An artificial neural network (ANN) toolbox (nntool) and fuzzy logic toolbox (fuzzy) are used to activate and initialise the dataset. Predictive reports are developed using the models trained using these toolkits.

The simulation's mean squared error, as shown in the figure above, is in line with predictions. Elapsed and future epochs are producing data with a very low error rate because the error factor is decreasing and approaching zero. The error factor is higher at the top, where it should be decreasing to zero.



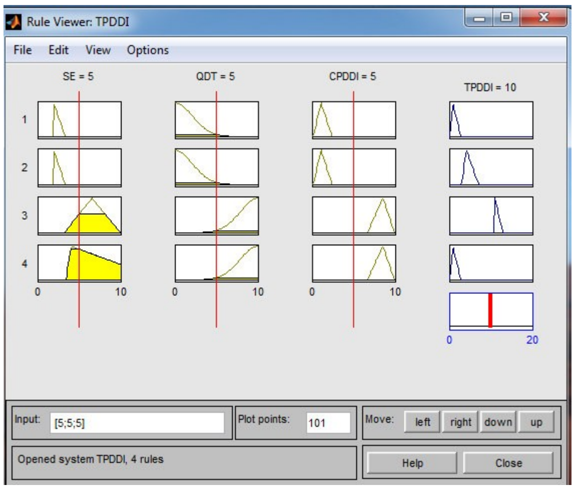
Input: [5;5;5]
Plot points: 101
Opened system rfd, 4 rules

FIGURE 4. **Fuzzy Rule Viewer for RFD**



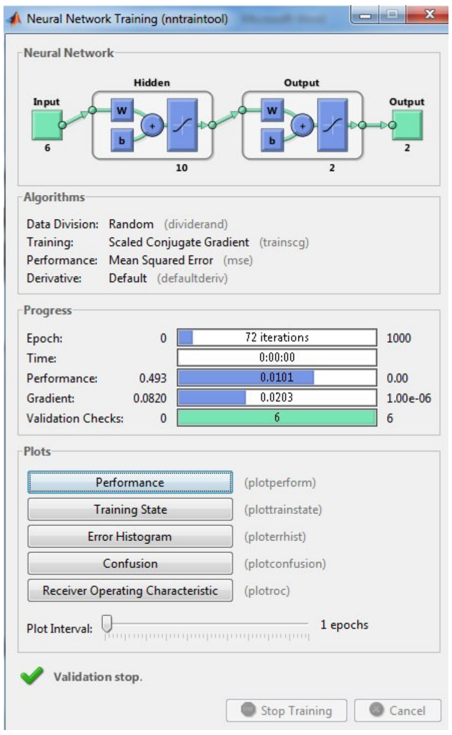
Input: [5;5;5]
Plot points: 101
Opened system DPDDI, 4 rules

FIGURE 5. **Fuzzy Rule Viewer for DPDDI**



Input: [5;5;5]
Plot points: 101
Opened system TPDDI, 4 rules

FIGURE 6. **Fuzzy Rule Viewer for TPDDI**



Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

```
Epoch:                 0    72 iterations    1000
Time:                       0:00:00
Performance:       0.493    0.0101           0.00
Gradient:         0.0820    0.0203           1.00e-06
Validation Checks:     0    6                6
```

Plots

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Confusion (plotconfusion)
Receiver Operating Characteristic (plotroc)

Plot Interval: 1 epochs

Validation stop.

FIGURE 7. **Implementation Patterns**

4. CONCLUSION

There has been a long history of software defect prediction and associated testing methods in use since the beginning of software development. As a result of the programming, execution, and logic activities, software programs are notoriously prone to a wide range of errors. In order to prevent new flaws from arising, it is imperative that systems for assessing the limitations and downsides of present technology be devised. Algorithms based on artificial neural networks (ANNs) have been implemented in this study, and the findings show that the ANN-based strategy outperforms the fuzzy logic approach. Implementation of fuzzy logic in the present work includes a set of fuzzy rules, and the software defect prediction metrics may be assessed based on these fuzzy rules. More accurate and precise results are obtained using the ANN-based technique as opposed to the traditional fuzzy one, making it more useful and performance-oriented.

FUNDING

None

ACKNOWLEDGEMENT

None

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

[1] M. F. Adak, “Software defect detection by using data mining based fuzzy logic,” *Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*, pp. 65–69, 2018.
[2] T. A. Khalid and E. T. Yeoh, “Early cost estimation of software reworks using fuzzy requirement-based model,” *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pp. 1–5, 2017.
[3] S. S. Maddipati, G. Pradeepini, and A. Yesubabu, “Software defect prediction using adaptive neuro fuzzy inference system,” *International Journal of Applied Engineering Research*, vol. 13, no. 1, pp. 394–397, 2018.
[4] M. Eftekhari and M. Khamar *Proposing an evolutionary-fuzzy method for software defects detection. Signal and Data Processing*, vol. 15, pp. 3–16, 2019.

[5] S. Chatterjee, B. Maji, and H. Pham, "A fuzzy rule-based generation algorithm in interval type-2 fuzzy logic system for fault prediction in the early phase of software development," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 31, no. 3, pp. 369–391, 2019.

[6] P. Sharma and A. L. Sangal, "Building and testing a fuzzy linguistic assessment framework for defect prediction in asd environment using process-based software metrics," *Arabian Journal for Science and Engineering*, vol. 45, no. 12, pp. 10327–10351, 2020.

[7] N. Iqbal and J. Sang, "Fuzzy logic testing approach for measuring software completeness," *Symmetry*, vol. 13, no. 4, pp. 604–604, 2021.

[8] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Software defect prediction based on fuzzy weighted extreme learning machine with relative density information. Scientific Programming," 2020.

[9] W. W. Agresti and W. M. Evanco, "Projecting software defects form analyzing Ada design," 1992.

[10] K. Y. Cai, C. Y. Wen, and M. L. Zhang, "A critical review on software reliability modeling," 1991.

[11] H. Can, X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang, and X. Liqiang, "A new model for software defect prediction using p swarm optimization and support vector machine," 2013.

[12] C. Catal, "Software fault Prediction: a literature review and current trends," *Exp Syst Appl*, vol. 38, pp. 4626–4636, 2011.

[13] C. Catal and B. Diri, "A systematic review of software fault predictions studies," *Exp Syst Appl*, vol. 36, no. 4, pp. 7346–7354, 2009.

[14] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Trans Softw Eng*, vol. 25, no. 4, pp. 573–583, 1999.

[15] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans Softw Eng*, vol. 25, no. 5, pp. 675–689, 1999.

[16] N. E. Fenton and M. Neil, "Predicting software defects in varying development lifecycles using bayesian nets," *Inf Softw Technol*, vol. 49, no. 1, pp. 32–43, 2007.

[17] N. E. Fenton and M. Neil, "On the effectiveness of early life cycle defect prediction with bayesian nets," *Empir Softw Eng*, vol. 13, pp. 499–537, 2008.

[18] M. A. Friedman, P. K. Tran, and P. L. Goddard, "Reliability techniques for combined hardware and software system," 1992.

[19] J. E. Gaffney and C. F. Davis, "An approach to estimating software errors and availability," *Proceedings of 11th Minnowbrook workshop on software reliability, SPC*, 1988.

[20] J. E. Gaffney and J. Pietrolewiez, "An automated model for software early error prediction (SWEEP)," in *Proceedings of 13th Minnowbrook workshop on software reliability*, 1990.

[21] "Guide for the use of IEEE standard dictionary of measures to produce reliable software," *IEEE Std*, vol. 982, pp. 2–1988, 1988.

[22] "Standard glossary of software engineering terminology," *IEEE Std*, vol. 610, pp. 12–1990, 1990.

[23] S. H. Kan, *Metrics and models in software quality engineering*. Boston: Addison wesley, 2002.

[24] M. Kaya and R. Alhajj, "A clustering algorithm with genetically optimized membership functions for fuzzy association rules mining," *The 12th IEEE international conference on Fuzzy systems, 2003, FUZZ'03*, vol. 2, pp. 881–886, 2003.

[25] A. B. Kitchenham, L. M. Pickard, S. G. Macdonell, and M. J. Sheppered, "What accuracy statistics really measure?," *IEEE Proc Softw*, vol. 148, no. 3, pp. 81–85, 2001.

[26] M. Li and C. Smidts, "Ranking software engineering measures related to reliability using expert opinion," in *Proceedings of the 11th international symposium on software reliability engineering (ISSRE)*, pp. 246–258, 2000.

[27] M. Li and C. Smidts, "A ranking of software engineering measures based on expert opinion," *IEEE Trans Softw Eng*, vol. 29, no. 9, pp. 811–824, 2003.

[28] M. R. Lyu, *Handbook of software Reliability Engineering*. Los Alamitos: IEEE Computer Society Press, 1996.

[29] Y. Maa, S. Zhua, K. Qinb, and G. Luob, "Combining the requirement information for software defect estimation in design time," *Inf Process Lett*, vol. 114, pp. 469–474, 2014.

[30] Mathscinet.

[31] J. A. Mccall, W. Randell, and J. Dunham, "Software reliability, measurement, and testing," 1992.

[32] S. Mohanta, G. Vinod, A. K. Ghosh, and R. Mall, "An approach for early prediction of software reliability," *ACM SIGSOFT Softw Eng Notes*, vol. 35, pp. 1–9, 2010.

[33] S. Mohanta, G. Vinod, and R. Mall, "A technique for early prediction of software reliability based on design metrics," *Int J Syst Assur Eng Manag*, vol. 2, pp. 261–281, 2011.