iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 7 **Issue:** I **Month of publication:** January 2019

**DOI:** http://doi.org/10.22214/ijraset.2019.1036

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 7 Issue I, Jan 2019- Available at www.ijraset.com*

# Spectrum Enhanced Fault Localization with Artificial Intelligence Paradigm for Software Diagnosis

P. Merlin[1], C. Saravanan[2]
[1]*M.Phil Research Scholar, Department of Computer Science, Kaamadhenu Arts and Science College, Sathyamangalam, Tamilnadu, India*
[2]*Assistant Professor, Department of Computer Science, Kaamadhenu Arts and Science College, Sathyamangalam, Tamilnadu, India*

***Abstract:*** ***Troubleshooting is an important part of software development. It starts when a bug is detected, for example, when testing the system and ends when the relevant source code is fixed. Among different researches, a paradigm has been presented to incorporate Artificial Intelligence (AI) in the modern software troubleshooting process. This paradigm have been used some methods named as Learn, Diagnose and Plan (LDP) and integrated three AI technologies. However, this approach is required for improving its performance of diagnosis accuracy. Hence, in this paper, LDP paradigm is incorporated the Spectrum-based Fault Localization (SFL)-based software diagnosis to improve the diagnosis accuracy. SFL is a statistical method which aims at guiding software developers to find faults quickly by providing a ranking of the most probable faulty components. In this approach, a training set is a set of Software Components (SCs) and a labeling that denotes which components are faulty and which are not. After that, the diagnoser is integrated with the Spectrum Enhanced Dynamic Slicing fault prediction model. The input to the diagnoser is the observed system behavior in the form of a set of tests that are executed and their result-pass or fail. Test planner accepts the set of diagnoses outputted by the diagnoser and if required, suggests additional tests that the tester should perform to discover the correct diagnosis. The experimental results show that the proposed approach provides better results. The experimental results show that the proposed approach provides better results.***
***Keywords: Spectrum-based Fault Localization, Diagnoser, Software developers, Test planner, Artificial Intelligence.***

## I. INTRODUCTION

This Troubleshooting is an important part of software development. It starts once a bug is detected, for example, once testing the system and ends when the relevant source code is fixed. Key actors in this process are the tester, who runs the tests and the developer, who writes the program and is able for fixing bugs (that is, to debug them). In modern software engineering [1-2], the interaction between tester and developer during troubleshooting is usually as follows. First, the tester executes a suite of tests and finds a bug. The tester then files a bug report, usually in some issue tracking system such as Bugzilla [3].

A new paradigm [4] has been presented for integrating AI with the modern software troubleshooting process. This paradigm was used LDP, incorporate three AI approaches: (1) machine learning (2) automated diagnosis (3) automated planning. These AI methods were integrated in LDP in a synergistic manner: the diagnosis approach was altered for assuming the learned fault predictions and the planner was changed the possible diagnoses outputted via the diagnosis approach. But, this approach has low diagnosis accuracy.

In this paper, LDP paradigm is integrated the SFL-based software diagnosis to improve the diagnosis accuracy. SFL is a statistical technique which aims at guiding software developers to discover faults by providing a ranking of the most probable faulty components. Initially, a training set is a set of SCs and a labeling that denotes which components are faulty and which are not. Then, the diagnoser is included with the Spectrum Enhanced Dynamic Slicing fault prediction model. The input to the diagnoser is the observed system behavior in the form of a set of tests that are executed and their result-pass or fail. Finally, test planner accepts the set of diagnoses outputted by the diagnoser and if needed, suggests additional tests that the tester must perform to find the correct diagnosis.

The remainder of the article is organized as follows: Section 2 describes about the software fault prediction. Section 3 describes about the proposed methods. Section 4 illustrates the performance evaluation of the proposed techniques. Section 5 concludes the research work.

## II. RELATED WORK

A combination of Artificial Intelligence (AI) techniques [5] was presented to improve software testing. Once a test fails, a model-based diagnosis (MBD) technique was utilized to propose a group of possible explanations and it was called diagnoses. A planning technique was used to suggest further tests for discovering the correct diagnosis. A tester was performed these tests and reports their result back to the MBD technique that was used this information to prune incorrect diagnoses. The iterative process was continued until the correct diagnosis was returned. Automated software fault detection [6] was developed using semi-supervised hybrid self-organizing map for identifying defect proneness of modules with software projects with high accuracy and enhancing detection model generalization ability. The benefit of this approach was the ability for predicting the label of the modules with semi-supervised manner via software measurement threshold values with the absence of quality data. The task of expert to recognize fault prone modules becomes less critical and more supportive.

The application of hybrid artificial neural network (ANN) and Quantum Particle Swarm Optimization (QPSO) [7] were investigated in software fault-proneness prediction. ANN was employed for classifying software modules into fault-proneness or non fault-proneness categories and QPSO was applied for decreasing dimensionality. Bayesian Regularization technique [8] was used to find the software faults before the testing process. This technique was used to decrease the cost of software testing and reduce a combination of squared errors and weights. The proposed approach based neural network was used to find the results on the given public dataset.

Random forest and support vector machines [9] were used for regression making use of a rule extraction approach ALPA. The proposed technique was built trees based on C4.5 and REPTree that mimic the black-box model as closely as possible. This technique was applied for publicly available data sets, completed and new datasets that this paper was put together by the Android repository. The usefulness of package-modularization metrics (PMMs) [10] were examined for predicting the fault-proneness of package in object-oriented software systems. Initially, this paper was used principal component analysis for analyzing whether PMMs capture additional information compared with traditional package-level metrics, integrating source code size and Martin's metrics suite. Then, univariate prediction models were used for investigating how PMMs were associated with package fault-proneness. Multivariate prediction models were built for examining the ability of PMMs to predict fault-proneness.

Software fault prediction techniques [11] were used to improve software diagnosis. The resulting data-augmented diagnosis technique overcomes key issues with software diagnosis techniques: ranking diagnosis and distinguishing between diagnoses with high and low probability. The experimental result was demonstrated the effectiveness of this approach empirically on three open sources domain and shown significant increase with accuracy of diagnosis and efficiency of troubleshooting. A quality assurance activity named software fault prediction [12] was discussed to decrease development cost and improve software quality. The aim of this paper was to investigate change metrics in conjunction with code metrics for improving performance of fault prediction models. Machine learning techniques were applied in conjunction with the change and source code metrics to built fault prediction models. The classification technique and new change metrics was performed efficiently.

## III.PROPOSED METHODOLOGY

In this section, LDP paradigm is integrated the spectrum-based fault localization (SFL)-based software diagnosis for enhancing the diagnosis accuracy.

### *A. Learn, Diagnose and Plan (LDP)*

The LDP is a paradigm that involves both human and AI elements. The humans in the loop have one of the tester or developer roles.

1) *Tester:* The tester is the one which observed and reported the abnormal system behavior. Based on this definition, a tester is a human quality assurance (QA) professional, a user of the system or even an automated testing script.
2) *Developer:* This is the person(s) in charge of developing the software. Its role in LDP is for fixing given software components (methods/ line of code/ file) which were discovered as faulty. The AI elements of LDP are,
3) *Fault Predictor:* This is a model which estimates the probability of every software component to be faulty. Significantly, the fault predictor does not assume the observed abnormal behavior of the system. This is significant as the probabilities created using the fault predictor will be employed as priors for the diagnoser. It can assume the entire data which does not integrate the current observations (past software versions and bugs, static code analysis and historical data), for example, various code complexity measures.
4) *Diagnoser:* It is an algorithm which accepts as input the observed abnormal behavior of the system as reported through the tester and outputs one or more possible explanations for that behavior. Each of these explanations, called as diagnoses, can be

an assumption which a particular group of software components is faulty. It is expected to output for every diagnosis an estimate of the probability which it is correct.

5) *Test Planner:* It is an approach which accepts the group of diagnoses outputted through the diagnoser and if required, recommends additional tests which the tester must execute to discover the correct diagnosis, that is, the software components which caused the abnormal system behavior.

### *B. Software Fault Prediction*

Fault prediction in software is a classification issue. Given a SCs, the objective is for discovering its class – faulty or healthy. Supervised Machine Learning (ML) methods are utilized for solving classification issues. As input, they are given a group of labeled instances that are pairs of instances and their correct labeling, that is, the correct class for every instance. In this case, instances are SCs and the labels are that SC is healthy and that is not. They output a classification model that maps an (unlabeled) instance to a class. This group of labeled instances is named the training set and the act of creating a classification model from the training set if represented as learning. Learning algorithms extract features from a given instance and try to learn from the training set the relation among the features of an instance and its class. A key for the success of ML methods is the choice of features utilized.

1) *Obtaining A Training Set:* We need a training set for learning a fault prediction model. A training set is a set of SCs and a labeling which represents that components are faulty and that are not. Manually tracking past bugs and tracing back their root cause is not a scalable solution. Being able to automatically create a training set highlights one of the main benefits of this work: it is applied for any software project which utilizes version control and problem tracking systems. So, this technique is deployed in such cases. The majority projects these days utilize a version control system and a problem tracking system. Version control systems, like Mercurial and Git, track modifications done to the source files. Problem tracking systems, like Trac and Bugzilla, record the entire reported bugs and track changes within their status, integrating once a bug gets fixed. A key feature within modern problem tracking and version control systems is which enable tracking that modifications to the source were completed for fixing a particular bug.

### *C. Diagnoser*

The diagnoser elements are described in LDP, and in particular how it integrates with the fault prediction model described in the previous section. The input to the diagnoser is the observed system behavior in the form of a set of tests that were executed and their outcome—pass or fail. The output is one or more explanations, which are sets of software components (e.g., class or method) that, if faulty, explain the observed failed and passes tests. The diagnoser we implemented is an extension of the Barinel software diagnosis algorithm. We provide here a brief description of Barinel. Barinel is a combination of model-based diagnosis (MBD) and spectrum-based fault localization (SFL). In MBD, we are given a tuple $\langle SD,,\rangle$, where $SD$ is a formal description of the diagnosed system's behavior, $COMPS$ is the set of components in the system that may be faulty, and $OBS$ is a set of observations. A diagnosis problem arises when $SD$ and $OBS$ are inconsistent with the assumption that all the components in $COMPS$ are healthy.

### *D. Test Planer*

In this section, we explain the test planner element of LDP. The test planner runs after the diagnoser, for cases where the diagnose outputted several plausible diagnoses. Indeed, a known limitation of software diagnosis algorithms is that they may output a large set of diagnoses. The task of the test planner is to plan additional tests for the tester to perform. The purpose of these tests is to provide additional information for the diagnoser, so that it will be able to find more accurate diagnoses. Test generation is a well-developed topic in software engineering. In LDP, we focus test planning by considering the diagnoses outputted by the diagnoser, as the purpose of these test is not to find new bugs but to improve the accuracy of the returned diagnoses (possibly pruning diagnoses that are found to be incorrect).

1) *Integrating the AI components in LDP:* A key strength of LDP is in its effective integration of its AI components. We illustrate this integration. The fault prediction algorithm generates a fault prediction model from the data in the issue tracking and source control systems. This fault prediction model is used to generate priors for the diagnoser. The diagnoser uses these priors, along with the knowledge it has about the analyzed software system, to output a set of diagnoses. Each of these diagnoses is associated with a likelihood score. These diagnoses are then passed to the test planner along with their likelihood scores. If needed, the test planner plans additional test, taking into consideration the given set of diagnoses. After performing these tests, new observations will be obtained and considered by the diagnoser.

*E. Combine SFL-Based Software Diagnosis In The Ldp Paradigm*

In this section, SFL-based software diagnosis (Spectrum Enhanced Dynamic Slicing) is incorporated in the LDP paradigm. This technique is used for enhancing the diagnosis accuracy.

1) *Spectrum-Based Software Diagnosis:* SFL is a statistical technique that aims at guiding software developers for identifying faults quickly by providing a ranking of the most probable faulty components. This ranking is generated by abstractions of program traces
2) *Program Spectra:* It is a collection of data that denotes a specific view on dynamic behavior of software. This data, gathered at run-time, consists of a number of flags or counters to the different components (statements) of a program. A lot of different forms of program spectra exist. Then, hit spectra (statements) are the most common for program debugging.

In diagnosis process, the hit spectra of N runs constitute a binary coverage matrix, whose columns correspond to M various statements of the program.

$$N\ spectra \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & \cdots & x_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \quad (1)$$

M statements errors

The information in that runs an error is detected constitutes a novel column vector, the error vector. This vector is used to represent a hypothetical statement of the program that is responsible for the entire observed errors. SFL basically consists in discovering the statements whose column vector resembles the error vector most.

In the data clustering, resemblances among vectors of binary, nominally scaled data, such as the columns within the matrix of program spectra, are quantified through similarity coefficients.

A lot of coefficients are learned in the past within the fault localization domain. Ochiai, generally utilized within the molecular biology domain, was determined to be amongst the most efficient ones for software fault localization. For every component $j$, it is explained,

$$OCHIAI(j) = \frac{n_{11}(j)}{\sqrt{(n_{11}(j)+n_{01}(j))\times(n_{11}(j)+n_{10}(j))}} \quad (2)$$

In this equation, $n_{11}(j)$ represents the number of failed runs in that component $j$ is involved, $n_{10}(j)$ denotes the number of passed runs in that component $j$ is involved, $n_{01}(j)$ indicates the number of failed runs in that component $j$ is not involved and $n_{00}(j)$ represents the number of passed runs in that component $j$ is not involved. Then,

$$n_{00}(j) = \left|\{i|x_{ij} = 0 \wedge e_i = 0\}\right| \quad (3)$$

$$n_{01}(j) = \left|\{i|x_{ij} = 0 \wedge e_i = 1\}\right| \quad (4)$$

$$n_{10}(j) = \left|\{i|x_{ij} = 1 \wedge e_i = 0\}\right| \quad (5)$$

$$n_{11}(j) = \left|\{i|x_{ij} = 1 \wedge e_i = 1\}\right| \quad (6)$$

It is assumed that a high similarity to the error vector represents a high probability which the corresponding component of the software causes the detected error. In this assumption, the calculated similarity coefficients rank the parts of the program with respect to their likelihood of being faulty.

*F. Refining Spectrum-Based Rankings*

Although SFL has been shown to be efficient and effective, its diagnostic accuracy is inherently limited, since the semantics of statements is not taken into account.

In particular, greatly due to the statistical nature of the technique, statements that exhibit identical execution patterns cannot be distinguished amongst themselves. To enhance its diagnostic quality, in SFL was combined with a model-based debugging approach (MBSD) within a framework coined Deputo.

This approach has its roots in AI and is based on abstract interpretation. This method is used to refine the ranking obtained from the spectrum-based method.

This algorithm implements in three steps, with the SFL technique employed in the setup phase, feeding into the subsequent model-based filtering phase, followed by an optional best-first search phase. This combination has importantly lower re-source requirements than applying MBSD on the entire program and by SFL only for ranking results. This algorithm is detailed description in (Hofer, B., et al. 2012).

*G. Spectrum-based Reasoning*

Model-based diagnosis techniques deduce component failure through logic reasoning using propositional models of component behavior. An inherent, strong point of model-based diagnosis is that it reasons in terms of multiple faults. In addition, BARINEL models program behavior in terms of program spectra. It uses a Bayesian technique for deducing multiple-fault candidates and their associated probabilities. So, it yields a probabilistic, information-rich diagnostic report. This algorithm is contained three main phases. In the initial phase, a list of candidates D is calculated from an activity matrix A and error vector e by STACCATO, a ultra-low cost approach to measure diagnosis candidates. The required performance is obtained at the cost of completeness, due to solutions are truncated at 100 candidates. In the next phase, $Pr(d_k)$ (probability which a given candidate $d_k$ is faulty) is measured for every candidate in D. In the last phase, for every $d_k$ the diagnoses are ranked along with $Pr(d_k)$ that is calculated through the EVALUATE function using the usual Bayesian update for every row. This approach is independent of test case ordering.

*H. Spectrum Enhanced Dynamic Slicing*

SFL with AI is Spectrum Enhanced Dynamic Slicing (SENDYS) that is used the similarity coefficients computed through spectrum-based fault localization as a-prior fault probabilities in model-based debugging. A lightweight model-based-debugging approach using dynamic slicing, called as slicing-hitting-set-approach (SHSC), is used in SENDYS. SHSC is calculated the dynamic slices of the entire faulty variable in the entire failed test cases, derives the minimal diagnoses from the slices and calculates the fault probabilities of the single statements using the size and the amount of the diagnoses that have the statement. SENDYS is a simple approach for combining SHSC with program spectra. This approach is measured the similarity coefficients from the program spectra, normalizes them and passes them to the SHSC technique as a-prior probabilities. This algorithm is explained in following paragraph. Initially, the program P is divided into components C. The coverage matrix M is calculated through implementing the entire test cases T on program P. From M the similarity coefficients R are measured and normalized. The normalized similarity coefficients are assigned to $\hat{R}$. The function ALLDIAGNOSES (P, T) is calculated the slices for the entire faulty variables in whole failing test cases. The resultant slices are combined to diagnoses by means of the corrected Reiter algorithm and assigned to D. The diagnosis fault probabilities $pd$ represent for the entire diagnoses $d$ the probability which $d$ contains a fault. The value of $pd[d]$ arises from two products: the product of the fault probabilities $\hat{R}$ of the statements contained in d and second, the product of the counter probabilities $(1-\hat{R})$ of the statements which are not contained in $d$. The statement fault probabilities are calculated through mapping back the diagnosis fault probabilities $pd$ to the statements. This is completed via building the sum of the entire diagnosis fault probabilities that have the statement. The statement fault probabilities $ps$ are normalized, sorted after their size and assigned to $\overline{ps}$. Then, the normalized statement fault probabilities $\overline{ps}$ are returned.

*I. Algorithm For Proposed Approach*

*1)* *Input:* Program P, set of test cases T
*2)* *Output:* List of possible fault locations sorted after their fault likelihood
*a)* Obtain a training set
*b)* Compute coverage matrix M for program P and test cases T
*c)* Compute similarity coefficients R for all statements
*d)* Compute the normalized values of the similarity coefficients $\hat{R}$ for whole statements
*e)* Compute the minimal diagnoses using ALLDIAGNOSES (P, T)
*f)* Perform the sorted statement fault probabilities $\overline{ps}$. Again, start with step 3. Use $\hat{R}$ instead of $pd[d]$. Test planner runs after the diagnose.
*g)* Return $\overline{ps}$

## IV. EXPERIMENTAL RESULTS

In this section, the performance of the proposed approach is analyzed with the existing technique. The comparison is made between proposed and existing approaches in terms of Precision, Recall and Accuracy.

*A. Precision*

Precision value is evaluated according to the feature classification at true positive prediction; false positive.It is expressed as follows:

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive + False positive}}$$

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 7 Issue I, Jan 2019- Available at www.ijraset.com*

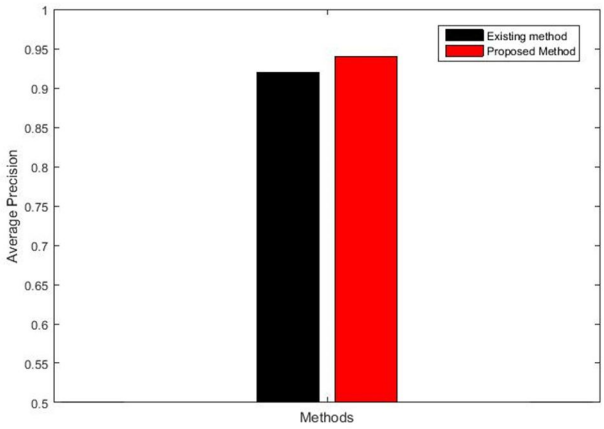



Fig. 1 Comparison of Average Precision

Fig. 1 shows that the comparison of proposed and existing techniques in terms of average precision. From this graph, methods are represented in X-axis and average precision values are denoted in Y-axis. In this analysis, the average precision value is increased for proposed method compared to existing method.

*B. Recall*

Recall value is evaluated according to the feature classification at true positive prediction, false negative. It is given as,

$$\text{Recall} = \frac{\text{Truepositive}}{(\text{Truepositive} + \text{Falsenegative})}$$

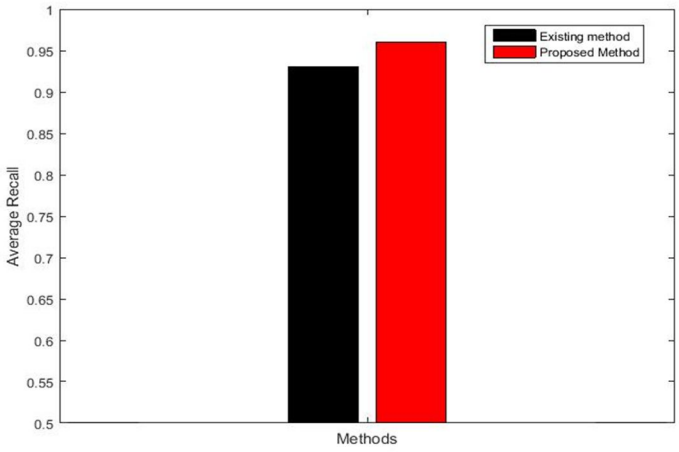



Fig. 2 Comparison of Average Recall

Fig. 2 shows that the comparisons of proposed and existing techniques in terms of average recall. In this analysis, methods are denoted in X-axis and average recall values are represented in Y-axis. From this graph, average recall value is increased for proposed method compared to existing method.

*C. Accuracy*

The accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined. Accuracy can be calculated from formula given as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

In the above equation, TP denotes true positive, TN represents true negative, FP denotes false positive and FN represents false negative.

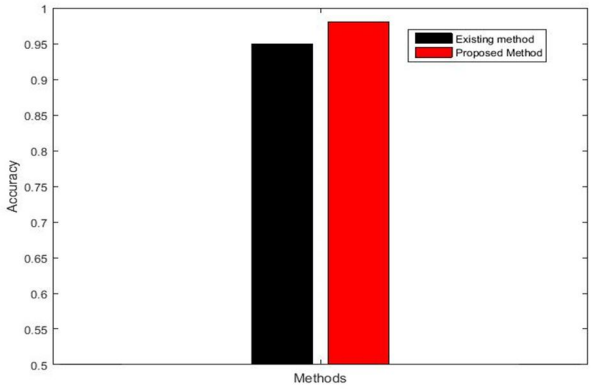



Fig. 3 Comparison of Accuracy

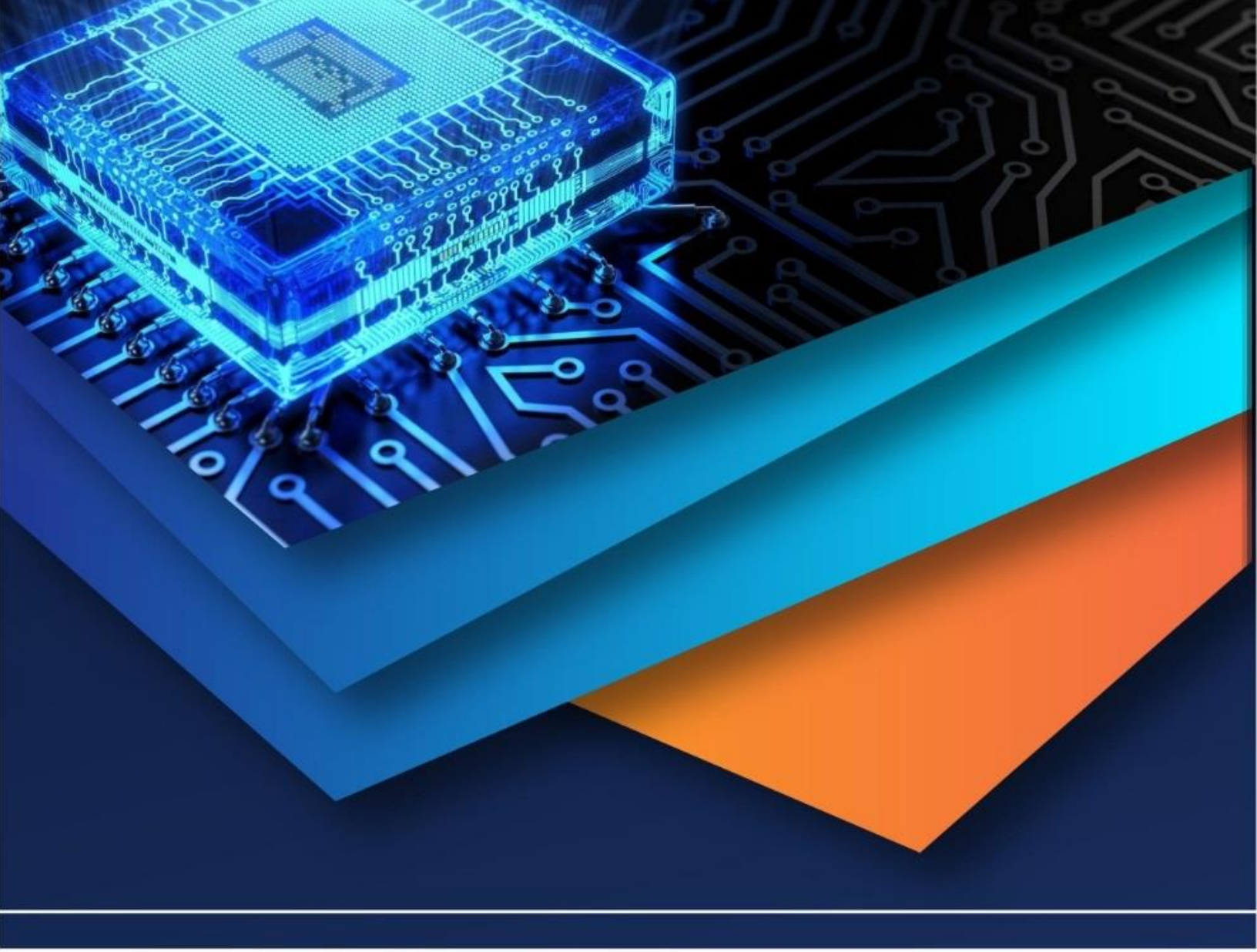
Fig. 3 shows that the overall performance comparison of proposed and existing methods in terms of Accuracy. In this graph, methods are represented in X-axis and accuracy values denoted in Y-axis. From this graph, the accuracy value is increased for proposed method compared to existing method.

## V. CONCLUSION

In this work, LDP paradigm is integrated the SFL-based software diagnosis for enhancing the diagnosis accuracy. In the proposed approach, a training set is a set of SCs and a labeling that represents which components are faulty and which are not. After that, the diagnoser is incorporated with the Spectrum Enhanced Dynamic Slicing fault prediction model. The input to the diagnoser is the observed system behavior in the form of a set of tests that are implemented and their result-pass or fail. Finally, test planner accepts the set of diagnoses outputted by the tester must perform to find the correct diagnosis. The experimental results show that the proposed approach provides better results.

## REFERENCES

[1] P. A. Ng and R. T. Yeh, Modern software engineering, foundations and current perspectives, Van Nostrand Reinhold Co., 1989.
[2] I. Jacobson, Object-oriented software engineering: a use case driven approach, Pearson Education India, 1993.
[3] N. Serrano and I. Ciordia, “Bugzilla, itracker, and other bug trackers,” IEEE software, vol. 22, no. 2, pp. 11-13, 2005.
[4] A. Elmishali, R. Stern, and M. Kalech, “An Artificial Intelligence paradigm for troubleshooting software bugs,” Engineering Applications of Artificial Intelligence, vol. 69, pp. 147-156, 2018.
[5] T. Zamir, R. T. Stern, and M. Kalech, “Using Model-Based Diagnosis to Improve Software Testing,” in AAAI, vol. 14, pp. 1135-1141, 2014.
[6] G. Abaei, A. Selamat, and H. Fujita, “An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction,” Knowledge-Based Systems, vol. 74, pp. 28-39, 2015.
[7] C. Jin and S. W. Jin, “Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization,” Applied Soft Computing, vol. 35, pp. 717-725, 2015.
[8] R. Mahajan, S. K. Gupta, and R. K. Bedi, “Design of software fault prediction model using BR technique,” Procedia Computer Science, vol. 46, pp. 849-858, 2015.
[9] J. Moeyersoms, E. J. de Fortuny, K. Dejaeger, B. Baesens, and D. Martens, “Comprehensible software fault and effort prediction: A data mining approach,” Journal of Systems and Software, vol. 100, pp. 80-90, 2015.
[10] Y. Zhao, Y. Yang, H. Lu, Y. Zhou, Q. Song, and B. Xu, “An empirical analysis of package-modularization metrics: Implications for software fault-proneness,” Information and Software Technology, vol. 57, pp. 186-203, 2015.
[11] A. Elmishali, R. Stern, and M. Kalech, “Data-Augmented Software Diagnosis,” in AAAI, pp. 4003-4009, 2016.
[12] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, “Empirical analysis of change metrics for software fault prediction,” Computers & Electrical Engineering, vol. 67, pp. 15-24, 2018.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY