ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

# Software Fault Prediction Using Single Linkage Clustering Method

K.C. Sujitha, S. Leninisha
PG Student, Dept. of IT, Easwari Engineering College, Chennai, Tamilnadu, India [1]
Assistant Professor, Dept. of IT, Easwari Engineering College, Chennai, Tamilnadu, India [2]

**ABSTRACT—**Until now, various techniques have been proposed for predicting fault prone modules based on prediction performance. Unfortunately quality improvement and cost reduction has been rarely assessed. The main motivation here is optimization of acceptance testing to provide high quality services to customers. From this perspective, the primary goal of this proposed methodology is reduction of acceptance test effort based on fault prediction results using single linkage clustering and simulation model. The prediction is conducted using test dataset and fault prone module are predicted by means of jaccard similarity measure. Simulation model estimates number of discoverable faults and results of simulation showed that the best strategy was to let the test effort be proportional to the number of expected faults in a module multiplied by log(module size).

**Index Terms**—Jaccard similarity measure, prediction model simulation model, test effort estimation.

## I. INTRODUCTION

With the rapid development in size and complexity of software systems, quality assurance activities such as testing and inspection have become more important for software developers and software purchasers who are responsible for acceptance testing. Fault prediction model have the potential to improve the quality of systems and reduce the costs associated with delivering those systems. Fault prediction modeling has become essential for the early identification of fault-prone code. These studies typically produce fault prediction models which allow software engineers to focus development activities on fault-prone code, thereby improving software quality and making better use of resources.

To prioritize quality assurance efforts, the techniques have been proposed for predicting fault prone modules by their probability of having a fault [8], the number of expected faults [6],[9] or the fault density [7]. Based on the prediction results, tester can allocate limited testing efforts to fault prone modules so as to find more faults with smaller effort. Our primary goal is to estimate the reduction of acceptance test effort that fault prediction can achieve. To achieve this goal, we need to allocate test effort with appropriate strategy to each module after prediction. We need to compute the expected number of discoverable faults with respect to test resources, resource allocation strategy, and set of modules to be tested. Tests currently conducted by the company are not complete, and most software systems contain faults after release [3]. The required test effort discovers as many faults as actual testing through simulation. To assess the cost effectiveness of prediction we need to measure the effort for metrics collection, data cleansing and modeling.

The next section explains related research ongoing in the fault prediction. In section 3 we explain architecture based analysis for our proposed methodology. Section 4 describes the results and conclusion.

## II. RELATED RESEARCH

Various studies in fault prediction to industry dataset have been reported previously [11], [12], [13], few studies have estimated the reduction of test effort or increase the quality of software achieved by fault prediction.

Arisholm et al. 2010 proposed systematic and comprehensive investigation of methods to build and evaluate fault prediction models [1]. To evaluate the fault proneness models, the three main aspects are (1) Data mining and machine learning techniques are compared, (2) Assess the impact of using different metric sets such as source code structural measures and change/fault history, (3) Performance of the models are compared in terms of accuracy, ranking ability, cost effectiveness measure.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

Bug prediction models are used to allocate software quality assurance efforts. Effort needed to review code or test code when the prediction models are evaluated. Revisiting two common findings in the bug prediction literature (1) process metrics outperform product metrics, (2) package level predictions outperform file level predictions [14], whereas Kamei and et.al showed that file level predictions outperform package level predictions [4].
Many researchers have discussion on probability of having a fault to distinguish fault prone modules from fault free modules [15]. On the other hand, some practitioners have discussion on number of faults to allocate quality assurance resources based on number of expected bugs that exist before testing [6],[9]. Some researchers have chosen the fault density rather than probability of having fault or number of faults since larger files have more defects [7].

Graves et al. 2000 proposed predicting fault incidence using software change history [2].The code to be aged or decayed if its structure makes it unnecessarily difficult to understand or change. Process measures based on the change history are more useful in predicting fault rates than product metrics of the code. We also compare the fault rates of code of various releases, if a module is a year older than similar module, the older module will have roughly a third fewer faults i.e., the existing version contains lesser faults than current version for the same software project.

## III. PROPOSED METHODOLOGY

As shown in fig.1 the most suitable dataset has been considered as input which has been taken from the historical dataset. By considering the dataset, prediction model is used to analyze the metrics in both source code and design document. Base metrics is the number of lines already exists in the existing version and change metrics is the number of lines added and deleted in the current version. Once the base metrics and change metrics is analyzed, single linkage clustering method is used to analyze the modules dependency information based on minimum distance. Jaccard similarity measure is used to calculate the distance. Based on the modules distance measure, simulation model is used to assign the test effort to each module. The allocated test effort for each module is computed based on the given test effort allocation strategy. After computing the test effort, fault discovery model discovers the fault with respect to the given test resources, resource allocation strategy and set of modules to be tested. Fault discovery model computes the discoverable faults in every module based on the test effort and module size. The number of initial faults before testing and the expected number of discoverable faults in each module are identified by the tester. So that testing time and testing costs are reduced by the tester to provide better services to the customer.

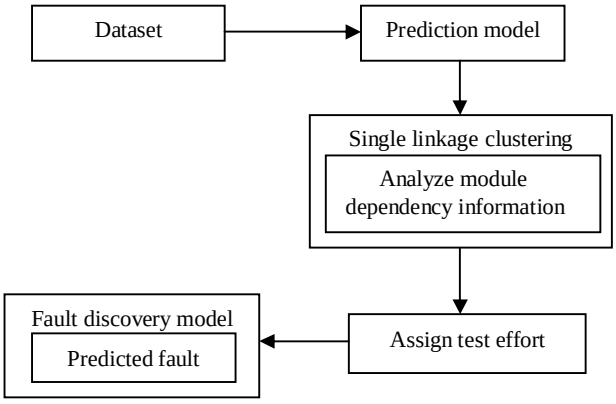


Fig.1. Proposed System Architecture

a. Prediction Model

The prediction model is conducted to analyze the base metrics and change metrics in both source code and design document. As shown in table 1, the base metrics for source code is the identification of number of lines in the existing version and change metrics indicates number of lines added and number of lines deleted in the current version. In this

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

prediction model, we need to predict both the number of faults and fault density and their prediction performance is compared.

TABLE 1

MEASURED METRICS

| Type | | Name | Definition |
|---|---|---|---|
| Source Code Metrics | Base Metrics | LOC<br>VG | Lines of Code<br>Cyclomatic complexity |
| | Change Metrics | ADD<br>DEL<br>VG | Number of lines added<br>Number of lines deleted<br>Increase of cyclomatic complexity |
| Design Metrics | Base Metrics | PAGE<br>MOD | Number of pages<br>Number of workflow modules |
| | Change Metrics | PAGE<br>MOD | Increase of page from previous release<br>Increase of module from previous release |

b. Single Linkage clustering

The single linkage method is the best hierarchical methods and operates by joining the two most similar objects at each step, which are not yet in the same cluster. The similarity of two clusters is the similarity of their most similar members. As shown in fig.2, the link between two clusters is made by a single element pair, namely those two elements that are closest to each other. In single linkage clustering, the modules are analyzed and then their dependency information's are extracted. Based on the dependency information the modules distances are calculated

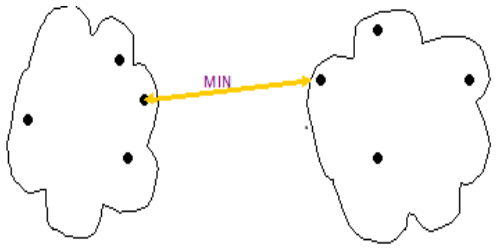


Fig.2. Single Linkage Clustering

As shown in fig.3, the two clusters separated by the shortest distance are combined into cluster first and at each step, we merge the closest pair of clusters until certain termination condition satisfied. For calculating, jaccard similarity measure is used. Jaccard similarity measure follows single linkage method i.e., it identifies the minimum distance between any two points in the two clusters. Jaccard similarity measure is defined as number of attributes common in two clusters divided by total number of attributes in both clusters.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

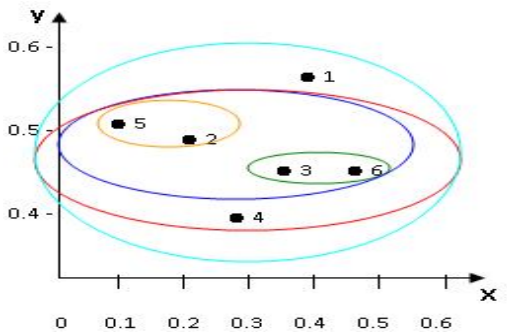


Fig.3. Jaccard Similarity

The modules are then clustered based on the jaccard distance metrics. As shown in equation 1, Jaccard similarity is a measurement that is used for identifying the degree of similarity and diversity of two cluster C1and cluster C2. The Jaccard coefficient used to measure the similarity between modules is defined as the size of the intersection divided by the size of the union.

$$\text{Jaccard}(C1,C2) = \frac{\text{Cadinality of intersection}}{\text{Cardinality of union}} \qquad \text{equ(1)}$$

c. Test Effort Estimation

In this module 7 possible test strategies are compared to choose the best strategy, to allocate test effort to each module.

[1] Equal test effort to all modules.

This strategy is considered as a baseline strategy.

[2] Test effort based on new module size.

$$t_i = t_{total} . S_i / S_{total}$$

Where $t_{total}$ is total test effort of all modules, Si is size of ith module and S total is total size of all modules.

To assign more test effort to larger modules, industries used this basic strategy. Arisholm et al. [1] pointed out that effort needed to test or review the module is roughly proportional to size. Indeed, many companies use baseline values for test case density, for e.g. in operational testing one must run at least 10 test cases per thousand lines of code

[3] Test effort based on new and modified modules.

$$t_i = t_{total} . (S_i^{new} + .1 \text{ X } S_i^{reused}) / (S_{total}^{new} + .1 \text{ X } S_{total}^{reused})$$

Where $S_i^{new}$ is the lines of new or modified code of the ith module, $S_i^{reused}$ is the lines of reused code of ith module, $S_{total}^{new}$ is the total lines of new or modified code, $S_{total}^{reused}$ is the total lines of reused code. This strategy distinguishes new/modified code and reused code. Since new/modified code is more faulty than new code. This strategy counts only 10% of reused lines and 10% may not be the optimal value.

[4] Test effort based on predicted faults.

$$t_i = t_{total} . \hat{F}_i / \hat{F}_{total}$$

$\hat{F}_i$ is the number of predicted faults in ith module and $\hat{F}_{total}$ is the total number of predicted faults in all modules. The more test effort is allocated where more faults are predicted to find more faults by using this straightforward strategy.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

[5] Test effort based on predicted fault density.

$$t_i = t_{total} . (\hat{F}_i / S_i) / (\hat{F}_{total} / S_{total})$$

This strategy reduces the effect of size by allocating more effort to higher fault density modules.

[6] Test effort based on predicted faults and module size.

$$t_i = t_{total} . \hat{F}_i / \hat{F}_{total} . S_i / S_{total}$$

This strategy is used to allocate more test effort on larger modules if they are likely to contain faults. This is a combination of strategy 2 and 4.

[7] Test effort based on expected faults in module with log (module size).

$$t_i = t_{total} . \hat{F}_i / \hat{F}_{total} . \log(S_i) / \sum_{i=1}^{n} \log (S_i)$$

Strategy 6 allocates more test effort to larger modules; if the total test effort is limited the faults in smaller modules might not be found. Strategy 7 tries to reduce the effect of very large modules, while still giving additional test effort to larger modules. Simulation model showed that the best strategy was to let the test effort be proportional to "the number of expected faults in a module × log (module size)". Based on the prediction results and total test effort, the allocated test effort for each module is computed based on the given test effort allocation strategy.

d. Fault Discovery Model

Fault discovery model estimates the discoverable faults with respect to the resource allocation strategy, given test resources and set of modules to be tested. This model computes discoverable faults in every module based on the given test effort and module size. In this fault discovery model, the fault detection rate is inversely proportional to the size of the module. All the modules have same parameter i.e, given a certain amount of test effort and same module size, the ease of finding a fault becomes same. Then the number of initial faults before testing and the Probability of detecting each fault per unit time are identified and the expected number of discoverable faults in each module is identified.

## IV RESULTS AND DISCUSSION

The results of this proposed methodology rely on the fault discovery model, which we extended from the exponential software reliability growth model. This proposed methodology used datasets which is collected from different releases of one software project. Metrics are analyzed in both source code and design document of the suitable dataset. The modules are clustered based on jaccard similarity measures and follows single linkage clustering method to analyze the modules dependency information. After that a set of possible test strategies are applied to detect the fault prone modules using simulation model. Given the prediction results and total test effort, the allocated test effort for each module is computed based on the given test effort allocation strategy. These seven strategies are by no means complete. To find better strategies in the future, we need to conduct simulation model with other strategies. Using the fault discovery model the expected number of faults is predicted. We expect further researches to improve this model to be more accurate and more realistic.

## V CONCLUSION

In this proposed methodology, we presented a method called single linkage clustering that helps testers to locate the related faults and reduces the test effort by analyzing the modules dependency information. Using the simulation model, there are seven test effort allocation strategies are compared to evaluate the cost effectiveness of fault prediction. Strategy 4 and 7 were the two best strategies that could possibly reduce the test effort. By using the 7th strategy with our

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Special Issue 2, April 2014**

best fault prediction model, the test effort could be reduced by 25% to detect as many faults as actual testing. The reduction of test effort is achieved only if the appropriate test strategy is employed with high fault prediction accuracy. In the future, it will be important to study whether it is possible to provide another tool for practitioners to reduce the testing costs or increasing the quality produced by testing.

## REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," J. Systems and Software, vol. 83, no. 1, pp. 2–17, 2010.
[2] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," IEEE Trans. Software Engineering, vol.26, no.7, pp. 653-661, 2000.
[3] Information-technology Promotion Agency, Japan (IPA) Software Engineering Center (SEC) ed., "White papers on software development projects in Japan, 2010-2011 Edition", ISBN978-4-9905363-3-6, 2010.
[4] Y. Kamei, A. Monden, and K. Matsumoto, "Empirical evaluation of SVM-based software reliability model," Proc. 5th ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE2006), vol. 2, pp. 39-41,2006.
[5] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A.E. Hassan, "Revisiting common bug prediction findings using effort aware models," Proc. 26th IEEE.
[6] T. M. Khoshgoftaar, A. Pandya, and D. Lanning, "Application of neural networks for predicting program fault," Annals of Software Engineering, vol. 1, pp. 141-154, 1995.
[7] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners", Proc. 3rd Working Conference on Mining Software Repositories (MSR2006), pp. 119-125, 2006.
[8] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," IEEE Trans. on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.
[9] P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson, "Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc.," Proc. 28th Int'l Conf. on Software Engineering, pp.413-422, 2006.
[10] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," Proc. Int'l Conference on Predictor Models in Software Engineering(PROMISE'09), pp. 1–10, 2009.
[11] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," Proc. 28th Int'l Conf. on Software Engineering (ICSE2006), pp.452-?, 2006.
[12] N. Ohlsson, and H. Alberg, "Predicting fault-prone software modules in telephone switches," IEEE Trans. Software Engineering, vol. 22, no. 12, pp. 886-894, 1996.
[13] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," IEEE Trans. on Software Engineering, vol. 31, no. 4, pp. 340-355, 2005.
[14] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," Proc. 2006 ACM/IEEE Int'l Symposium on Empirical Software Engineering (ISESE'06), pp. 18–27, 2006.
[15] B.Turhan, T. Menzies, A.Bener, and J. Distefano, "On the relative value of cross-company and within-company data for defect prediction", Empirical Software Engineering, vol. 14, no. 5, pp. 540-578, 2009.

## BIOGRPHY

**K.C.Sujitha** received the B.Tech. degree in Information Technology from Jaya Engineering College affiliated to Anna University, Chennai, in 2009. Now she is currently pursuing M.E. degree in Software Engineering, Easwari Engineering College affiliated to Anna University, Chennai.

**S. Leninisha** received the B.E. degree in Computer Science and Engineering from Madurai Kamaraj University, Tamil Nadu, Chennai, in 2000, the M.E. degree in Computer Science and Engineering from Anna University, Tamil Nadu, Chennai, in 2006 and the currently pursuing Ph.D.degree in Anna University, Chennai in Faculty of Information and Communication Engineering, registered in July 2011. She has Around 10 years of teaching experience. Now she is working as a Assistant Professor (Selection Grade) in Easwari Engineering College affiliated to Anna University, Chennai, Tamil Nadu. Also, she is a Member of IEEE and IET professional body. Her main research interests are feature extraction in remotely sensed images, pattern recognition, and invariants for object recognition. She published two papers in international conference in her research area.