e-ISSN: 2349-9745
p-ISSN: 2393-8161

Scientific Journal Impact Factor (SJIF): 1.711

International Journal of Modern Trends in Engineering and Research

www.ijmter.com

Software Defect Prediction Using Local and Global Analysis

Ms. N. Narmadha[1], Dr. M. Latha[2], Mr. R. Subramanian[3]

[1]Research Scholar

[2]M.Sc., M.Phil., Ph.D, Associate Professor, Department of Computer Science.

[1,2]Sri Sarada College for Women, Salem, Tamilnadu, India

[3]Erode Arts and Science College, Erode

Abstract -The software defect factors are used to measure the quality of the software. The software effort estimation is used to measure the effort required for the software development process. The defect factor makes an impact on the software development effort. Software development and cost factors are also decided with reference to the defect and effort factors. The software defects are predicted with reference to the module information. Module link information are used in the effort estimation process.

Data mining techniques are used in the software analysis process. Clustering techniques are used in the property grouping process. Rule mining methods are used to learn rules from clustered data values. The "WHERE" clustering scheme and "WHICH" rule mining scheme are used in the defect prediction and effort estimation process. The system uses the module information for the defect prediction and effort estimation process.

The proposed system is designed to improve the defect prediction and effort estimation process. The Single Objective Genetic Algorithm (SOGA) is used in the clustering process. The rule learning operations are carried out sing the Apriori algorithm. The system improves the cluster accuracy levels. The defect prediction and effort estimation accuracy is also improved by the system. The system is developed using the Java language and Oracle relation database environment.

I. INTRODUCTION

Software engineering is the technological and managerial discipline with systematic production and maintenance of software products that are developed and modified on time and within cost estimates. Successful software maintenance, like all software activities, requires a combination of managerial skills and technical expertise. The primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers. Software engineering is concerned with development and maintenance of technological products, problem-solving techniques common to all engineering disciplines

Engineering problem-solving techniques provides the basis for project planning, project management, systematic analysis, methodical design, careful fabrication, extensive validation and ongoing maintenance activities. Appropriate notations, tools and techniques are applied in each of these areas. A fundamental principle of software engineering is to design software products that minimize the intellectual distance between problem and solution; the variety of approaches to software development is limited on lay by the creativity and ingenuity of the programmer.

The software industry's inability to provide accurate estimates of development cost, effort and/or time is well known. This inability is described in reports from project management consultancy companies, case studies on project failures, articles in the computer press and estimation surveys. The common belief seems to be that the cost overruns are very large and few software professionals and researchers react with disbelief when being presented with the inaccuracy values reported in Standish

Group's Chaos Report, i.e., an average cost overrun of 89%. There may be several reasons for this attitude.

It is difficult to get a balanced view on the software industry's estimation performance without unbiased information from a representative set of projects and organizations. The surveys presented in scientific journals and conferences may be a source for such unbiased information. The system summarizes estimation results from surveys on software estimation. There has not been conducted any structured review of the estimation surveys with the aim of summarizing the knowledge of software effort estimation performance. Besides summarizing results from estimation surveys, the main purposes of the system is to challenge some common estimation beliefs that may be inaccurate, to discuss methodical aspects related to completion of industrial surveys and to support future surveys on software effort estimation.

II. RELATED WORK

The main goal of this section is not to provide a generic review of defect prediction and effort estimation work, but to highlight work that documents contradictory results which make it difficult to generalize solutions for defect prediction or effort estimation.

2.1. Effort Estimation Process

Conclusion instability in effort estimation may be a fundamental property of the datasets we are exploring. For example, tests the stability of Boehm's COCOMO software development effort estimation model. In that analysis, 20 times, we learned effort = $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots$ using a random 2/3rds sample from 93 NASA projects. Only the coefficient on lines of code (loc) was stable. The observed ranges on the other βi coefficients were very large. In fact, the signs of five coefficients even changed from positive to negative. This coefficient instability is particularly troubling because we know of NASA project managers who have made acquisition decisions worth tens of millions of dollars based on the COCOMO coefficients.

Other papers also report contradictory findings about effort estimation. Jorgensen reviewed 15 studies comparing model-based to expert-based methods. Five of those studies favored expert-based methods, five found no difference and five favored model-based methods. Kitchenham et al. reviewed studies that checked if data imported from other organizations were as useful as local data. From a total of seven studies, three found that models from other organizations were not significantly worse than those based on local data, while four found that they were significantly different. MacDonell and Shepperd also performed a review on the value of local versus global effort estimation models through a replication. From a total of 10 studies, two were found to be inconclusive, three supported global models and five supported local models. Similarly, Mair and Shepperd compared regression to analogy methods for effort estimation and found conflicting evidence. From a total of 20 empirical studies, seven favored regression, four were indifferent and nine favored analogy.

2.2. Defect Prediction Process

In the area of defect prediction, there are also many contradictory findings. For example, Zimmermann et al. learned defect predictors from 622 pairs of projects hproject1; project2i. In only 4 percent of pairs did the defect predictors learned in project1 work in project [2]. Similar findings concern the OO metrics as well. For the manager of a software project is particularly troubling. Each study makes a clear, but usually different, conclusion. Hence, it is difficult for a manager to make a clear decision about, for example, the merits of a proposed coding standard, where maximum depth of inheritance is required to be less than some expert specified threshold.

As to the root cause of the instabilities offer the following conjecture. The models learned from different regions within effort data can have very different properties. If defect data were as varied as effort data, then we would naturally expect that different samples of different projects would yield different models due to dataset shift [4]. If this conjecture is correct, then we would expect that clusters within the data should produce different models. This paper is a test of that conjecture. In short, we show that different regions of the data generate different models. Further, the models built from specialized regions within the dataset perform better than those learned across all data.

III. LOCAL AND GLOBAL ANALYSIS ON SOFTWARE COMPONENTS

Process and product data are used in software engineering (SE) to support a variety of tasks, such as defect prediction, effort estimation, refactoring of source code, determination of the social networks of programmers, learning the expected interface between modules and so on [3]. Two questions are at the center of much of the research in the field: 1) What data are best suited to support specific tasks? and 2) what is the best way to reason about SE data? This paper explores the latter question, in the context of: software effort reduction: finding rules for reducing a project's development time;

- software defect reduction: finding rules for reducing a project's defect count.

Our focus in this paper is not on what data are used for building models for defect prediction or effort estimation, but rather on the source of the data and implicitly, the applicability of the lessons derived by the models. Software data come from some sources. These data show the defects or effort associated with examples from that source. How should we reason about these data? On this point, the literature is contradictory. Some existing work argues that data from multiple sources can generate rules that apply in any context. We call these global lessons. On the other hand, other papers indicate that the best lessons are learned from within one source, which implies that these lessons are only useful in their context [5], [6], [7]. We call these local lessons.

When project managers want to make changes in their projects to minimize the development effort or the rate of defects, they are faced with two options: 1) make changes based on global lessons available from existing data, or 2) mine data about the current project and infer local lessons. The dilemma of the manager is obvious. In the first case, expensive changes may be undertaken without reaching the desired goal if the global lessons prove to be wrong for this context. In the second case, an upfront investment is needed to collect and analyze data and to generate the local lessons, which may be unnecessary if the global lessons apply to this context. This paper addresses this dilemma and reports on experiments where:

- Data from different sources are combined.
- Within that combination, automatic tools find clusters of related examples. Note that clusters may contain examples from multiple sources.
- Data mining is then used to learn lessons from the examples in each cluster. The generated lessons are compared.

Before this experiment, our previous work indicated that local lessons seem to be best for defect prediction and effort estimation [8]. The surprising result of the experiment presented in this paper is that the best lessons for a project from one source come from neighboring clusters with data from nearby sources, but not inside that source. We will call such lessons neighbor lessons.

We conclude that when project managers attempt to draw lessons from some historical data source, they should: 1) ignore any existing local divisions into multiple sources, 2) cluster across all available data, then 3) restrict the learning of lessons to the clusters that are nearest to the test data. While the paper focuses on defect prediction and effort estimation data, we believe that our conclusions can translate to other types of data, related to different SE tasks. This paper extends a prior publication [1] in the following way:

- That paper only explored four datasets. Here, we explore over twice that number of datasets.

That prior publication only explored local versus global lessons and found evidence that supported local lessons over the global ones. In this paper, explore clusters from multiple sources.

IV. PROBLEM STATEMENT

The software analysis system generates lessons learned for defect prediction and effort estimation. Lessons are global to multiple projects or just local to particular projects. The system comparatively evaluates local versus global lessons learned for effort estimation and defect prediction. The system applies automated clustering tools to effort and defect datasets from the PROMISE repository. Rule learners generated lessons learned from all the data, from local projects, or just from each cluster. The lessons learned after combining small parts of different data sources were superior to either generalizations formed over all the data or local lessons formed from particular projects. The following problems are identified from the software effort estimation and defect prediction methods.

- The clustering system produces low accuracy in partitioning process.
- The rule learning system handles minimum of set of rules only.
- Defect prediction and effort estimation accuracy is low.
- Software module relationships are not fully focused in the defect and effort analysis.

V. GENETIC ALGORITHM BASED SOFTWARE ANALYSIS

The software defect prediction process is designed to identify the defects that are raised in the software system. The effort estimation mechanism is applied to estimate the effort needed for the system development process. The main goal of the system is not to provide a generic review of defect prediction and effort estimation work, but to highlight work that documents contradictory results which make it difficult to generalize solutions for defect prediction or effort estimation. In the area of defect prediction, there are also many contradictory findings. Defect predictors from 622 pairs of projects hproject1; project. In only 4 percent of pairs did the defect predictors learned in project work in project. Similar findings concern the OO metrics as well. Lists 28 studies that offer contradictory conclusions regarding the effectiveness of OO metrics for predicting defects.

Conclusion instability in effort estimation may be a fundamental property of the datasets. Only the coefficient on lines of code was stable. The observed ranges on the other βi coefficients were very large. In fact, the signs of five coefficients even changed from positive to negative. This coefficient instability is particularly troubling. NASA project managers who have made acquisition decisions worth tens of millions of dollars based on the COCOMO coefficients. The system is designed to perform defect prediction and effort estimation process for the software systems [9]. The clustering and rule learning methods are used to identify the defects and efforts. The Single Objective Genetic Algorithm (SOGA) is used for the clustering process. The rules are learned using the Apriori algorithm. The system improves the accuracy in clustering process. Defect prediction and effort estimation operations are also improved with internal relationship of the software components.

5.1. Software Module Clustering

Existing approaches to the twin objectives of high cohesion and low coupling have combined these two objectives into a single-objective function, with all of the drawbacks. Pareto optimality is an alternative approach to handle multiple objectives that retains the character of the problem as a multi-objective problem. Using Pareto optimality, it is not possible to measure “how much” better one solution is than another, merely to determine whether one solution is better than another. In this way, Pareto optimality combines a set of measurements into a single ordinal scale metric.

A Pareto optimal search yields a set of solutions that are mutually no dominating and which form an approximation to the Pareto front. The Pareto front is the set of elements that are not dominated by any possible element of the solution space. The Pareto front thus denotes the best results achievable; it is the equivalent of the set of globally optimal points in a single-objective search. As with the single-objective formulation, it is not possible to guarantee to locate this globally optimal solution set, merely to attempt to approximate it as closely as possible. Each set of objectives leads to a different multi-objective formulation of the problem. Two sets of objectives will be considered in the system. They are the Maximizing Cluster Approach and the Equal-size Cluster Approach.

5.2. Module Clustering with GA

Genetic algorithms use concepts of population and of recombination inspired by Darwinian Evolution A generic genetic algorithm is presented. An iterative process is executed, initialized by a randomly chosen population. The iterations are called generations and the members of the population are called chromosomes because of their analogs in natural evolution. The process terminates when a population satisfies some predetermined condition.

```
Set generation number, m: = 0
Choose the initial population of candidate
Solution ,P(0)
Evaluate the fitness for each individual of
P(0),F(P_i(0))
loop
      Recombine: P(M): = (P(m))
      Mutate : P(m) : = M(P(m))
     Evaluate : F(P(m)
      Select   : P(m+1) : = S(P(m))
      m : = m +1
      exit when goal or stopping
condition is satisfied
end loop;
```

A Generic Genetic Algorithm.

VI. LOCAL AND GLOBAL ANALYSIS MODEL FOR SOFTWARE DEFECT PREDICTION

The software defect prediction and effort estimation process is designed to measure the defect and effort values from the software properties. The clustering and rule learning mechanism is used for the system. The Single Objective Genetic Algorithm (SOGA) is used for the clustering process. The Apriori algorithm is used for the learning process. The system divided into four major modules. They are product data analysis, criteria property extraction, single objective clustering and defect prediction and effort estimation process. The modules and their link information are extracted from the product data analysis module. Criteria property extraction module is designed to identify the module property values. Single objective clustering is performed with only one objective functions. The defect prediction and effort estimation operations are carried out under the defect and effort estimation module.

6.1. Product Data Analysis

The product information is provided in module dependency graphs. Module link information is provided under Module Dependency Graph (MDG) files. The MDG files are used as the input for the system. Modules, links and weight information are extracted from the product information.

6.2. Criteria Property Extraction

The modules and their links are used for the criteria property extraction process. Communication bandwidth is estimated with module link information. Each module is analyzed with it's in and out link information. The module length and feature location are also identified.

6.3. Single Objective Clustering

The genetic algorithm is used for the clustering process. Single objective clustering model is used in the system. The objective function uses the module property values. Only one criteria is considered in the clustering process.

6.4. Defect Prediction and Effort Estimation

The defect prediction and effort estimation module is divided into two sub modules. They are defect prediction sub module and effort estimation sub module. The software defects are predicted with reference to the internal relationship values. The effort estimation is carried out with the support of the software modules and their length values. The system produces the predicted values and actual values for each modules. The rule learning mechanism is used in the defect prediction and effort estimation process.

VII. PERFORMANCE ANALYSIS

The software defect prediction and effort estimation system is implemented and tested with a set of benchmark datasets. The Genetic Algorithm and Variational clustering (VC) schemes are used in the testing process. The VC scheme is composed with "WHICH" and "WHERE" techniques. Three performance metrics are used to evaluate the quality of the system. They are entropy, separation index and prediction accuracy. The entropy and separation index measures are used to evaluate the quality of the clustering process. Defect prediction and effort estimation process results are analyzed using the prediction accuracy measure.

7.1. Entropy

Entropy reflects the homogeneity of a set of objects and thus can be used to indicate the homogeneity of a cluster. This is referred to cluster entropy. Lower cluster entropy indicates more homogeneous clusters. On the other hand, we can also measure the entropy of a prelabeled class of objects, which indicates the homogeneity of a class with respect to the generated clusters. The less fragmented a class across clusters, the higher its entropy and vice versa. This is referred to as class entropy.

The entropy measurement is used to estimate the quality of the clusters. The cluster entropy and overall entropy are calculated for different data intervals and MPC based models. The entropy values for VC Scheme and GA Scheme models are produced in figure 7.1. The comparison result shows that the GA Scheme outperforms than the VC Scheme. The GA scheme improves the cluster accuracy 5% than the VC scheme.

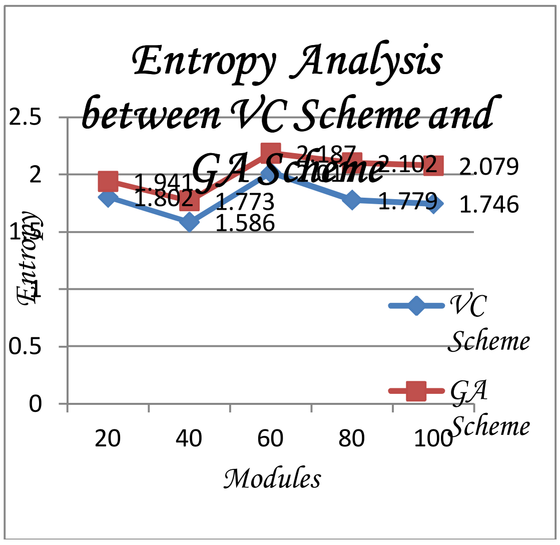

Figure No: 7.1. Entropy Analysis of VC Scheme and GA Scheme

7.2. Separation Index

Separation Index (SI) is another cluster validity measure that utilizes cluster centroids to measure the distance between clusters, as well as between points in a cluster to their respective cluster centroid. It is defined as the ratio of average within-cluster variance to the square of the minimum pairwise distance between clusters:

$$SI = \frac{\sum_{i=1}^{NC}\sum_{xj\in ci} dist(x_j, m_i)^2}{N_D \min_{1\le r,s\le N_c} dist(m_r, m_s)\}^2}$$

where m_i is the centroid of cluster c_i and distmin is

$$= \frac{\sum_{i=1}^{N_c}\sum_{xj\in ci} dist(x_j, m_i)^2}{N_D \cdot dist_{\min}^2}$$

the minimum pairwise distance between cluster centroids. Clustering solutions with more compact clusters and larger separation have lower Separation Index, thus higher values indicate better solutions. This index is more computationally efficient than other validity indices, such as Dunn's index, which is also used to validate clusters that are compact and well separated. In addition, it is less sensitive to noisy data.

The separation index measure is used to analyze the cluster intervals and transaction interval values. The separation index analysis is shown in figure 7.2. The analysis result shows that the GA Scheme improves the cluster quality 20% than the VC Scheme technique.

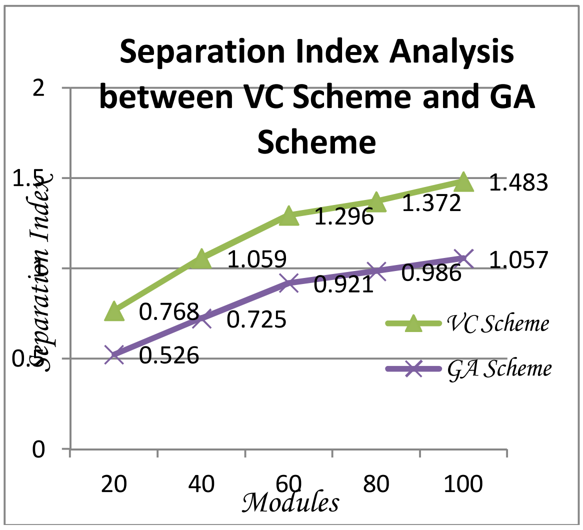

Figure 7.2: Separation Index Analysis between VC Scheme and GA Scheme

7.3. Prediction Accuracy

The prediction accuracy measure is used to evaluate the quality of the rule learning process with defect prediction and effort estimation process. The defect prediction and effort estimation results are compared with the predicted values and evaluated values. The prediction accuracy analysis is shown in figure 7.3. The GA scheme improves the prediction accuracy 20% than the VC scheme. The system produces better cluster results with effective rule learning results.

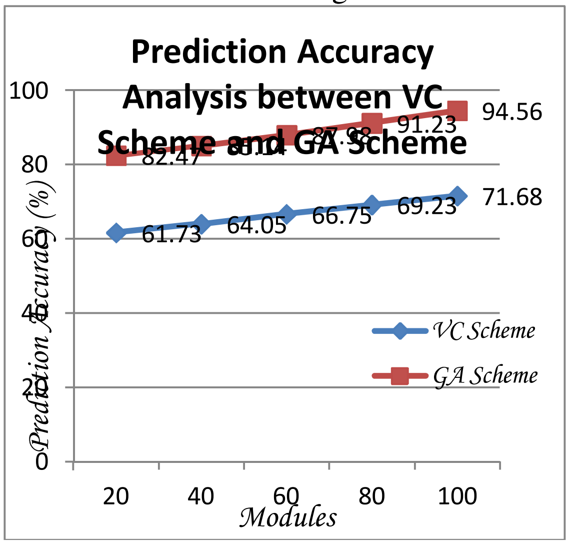

Figure 7.3: Prediction Accuracy Analysis between VC Scheme and GA Scheme

VII. CONCLUSION AND FUTURE WORK

The software defect prediction and effort estimation operations are performed in the system with the support of clustering and rule learning methods. The system has discussed ways to collect training data for learning lessons from SE projects and to generate rules for reducing the number of defects or the development effort. At issue here is how the chief information officers should set policies for their projects. Should they demand that all their projects rigorously apply all the advice from the standard SE textbooks.

The software defect prediction and effort estimation system is implemented to analyze the software products to measure the defect and effort factors. The Genetic Algorithm based clustering scheme is used for the data clustering process. The Apriori algorithm is used to carry out the rule learning process. The system improves the cluster accuracy and prediction accuracy levels. The system can be enhanced with the following features.

- The system can be enhanced to predict the software risk levels using design information.
- The system can be enhanced with Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) techniques in the prediction process.
- The system can be improved to measure the relationship level between the local and global rules.
- The system can be enhanced to perform the prediction process using the software and source code analysis mechanism.

REFERENCES

[1] T. Menzies, A. Butcher and D. Cok, “Local vs Global Models for Effort Estimation and Defect Prediction,” Proc. 26th IEEE/ACM Int’l Conf. Automated Software Eng., 2011.
[2] Martin Shepperd, David Bowes and Tracy Hall, “Researcher Bias: The Use of Machine Learning in Software Defect Prediction”, IEEE Transactions On Software Engineering, Vol. 40, No. 6, June 2014.
[3] Saehwa Kim, “Synthesizing Multithreaded Code from Real-Time Object-Oriented Models via Schedulability-Aware Thread Derivation”, IEEE Transactions On Software Engineering, Vol. 40, No. 4, April 2014.
[4] B. Turhan, “On the Dataset Shift Problem in Software Engineering Prediction Models,” Empirical Software Eng., vol. 17, pp. 62-74, 2012.
[5] D. Posnett, V. Filkov and P. Devanbu, “Ecological Inference in Empirical Software Engineering,” Proc. IEEE/ACM Int’l Conf. Automated Software Eng., 2011.
[6] N. Bettenburg, M. Nagappan and A.E. Hassan, “Think Locally, Act Globally: Improving Defect and Effort Prediction Models,” Proc. Ninth IEEE Working Conf. Mining Software Repositories, pp. 60-69, 2012.
[7] Y. Yang, L. Xie, Z. He, Q and R. Valerdi, “Local Bias and Its Impacts on the Performance of Parametric Estimation Models,” Proc. Seventh Int’l Conf. Predictive Models in Software Eng., 2011.
[8] Tim Menzies, Andrew Butcher, David Cok, Andrian Marcus, Lucas Layman, Forrest Shull, Burak Turhan and Thomas Zimmermann, “Local versus Global Lessons for Defect Prediction and Effort Estimation”, IEEE Transactions On Software Engineering, Vol. 39, No. 6, June 2013.
[9] Christian Kastner, Alexander Dreiling and Klaus Ostermann, “Variability Mining: Consistent Semi-automatic Detection of Product-Line Features”, IEEE Transactions On Software Engineering, Vol. 40, No. 1, January 2014.