International Journal of Modern Electronics and Communication Engineering (IJMECE) ISSN: 2321-2152
Volume No.-7, Issue No.-3, May, 2019

# Evaluation and Measurement of Software Product Metrics for Object Oriented Environment to Improve Software Product

Piyush Prakash
Department of Computer Science
Mewar University, Gangrar, Chittorgarh,
Rajasthan, India
E-mail: piyush19sept@gmail.com

Sarvottam Dixit
Department of Computer Science
Mewar University, Gangrar, Chittorgarh,
Rajasthan, India
E-mail: sdixit_dr@rediffmail.com

S. Srinivasan
Department of Computer Science & Applications
PDM University, Bahadurgarh
Haryana, India
E-mail: dss_dce@yahoo.com

**Abstract:** Software metrics have a clear connection with software engineering measurement. As companies grow, to mature and enhance enterprise qualities, in corporate sectors metrics are attaining significance as well as recognition/acceptance. For an effective software test manager, the requirement is the measurement of a test process to develop and assess a cost-effective test strategy. As the size and complexity of software is increasing, correct measurement is primarily required because manual inspection of software becomes a more difficult task. Prior the software quality i.e. how to measure and improve its quality, is of concern to most software engineers. The underlying purpose of the study was to evaluate and analyze the measurement of software quality by using software product metrics.

**Keywords:** Software metrics, Product metrics, Quality improvements, Software engineering.

## I. INTRODUCTION

Metrics are more accurate when they are derived from well-defined completion criteria for software products and their intermediate modules. Also known as quality metrics, product metrics are used to evaluate the software's properties. Weighted defect are derived from defect information. Weighted defects are calculated with the help of severity of errors found in the software. Each open defect, as its severity, is accompanying with a number.

Software metric is defined as an estimation of software properties or software specifications. Subsequently, quantitative ways have been verified influential within alternative sciences, theoreticians and computer science specialists done hard work to bring alike methodologies for development of software. The modern software developer may point out that ingenuous and simplified measurements may be harmful than good. Tom DeMarco specified, "You can't control what you can't measure." [13].

In general, software metrics are categorized such as process, product and project metrics. Process metrics are used for measuring the characteristics of software methods which are employed to develop the software it is also known as management metrics. It includes the effort metric, cost metric and reuse metric. The performance, complexity, size and quality of the product are measured by product metrics. Project metric describe the features and execution of the project.

The prediction whether or not the software module is defective before the testing process is applied is done by the software fault prediction mechanism. In a module predicted to be as poor compared to a module predicted to be defective, more testing efforts are made [1].If there is any bug undetected, then in many software systems ,severe damage can be caused e.g. financial system, banking, satellite system, medical system etc. Therefore, in software system development, testing is very important [2]. For inter releases software, in the event of a prediction of software failure, the software's previous version data used for the "classifier" training may always not have the identical granularity as of test data that could be a major problem. In cross project failure prediction, the same scenario may occur. Hence, it's essential to bring the metrics at same level. Herein, the class level software metrics are accumulated by calculating the IQR and AAD values for the class - level metrics. The most commonly used metrics for the fault prediction are McCabes metrics, LOCs (Line Of Codes), chidamber and kemerer(C&K) metrics , Halsteads metrics etc. and naive bayes, techniques for machine learning [4][3], artificial neural networks [6], logistical regression [5][2], and common machine learning techniques [5]. Three techniques of machine learning were used in this paper: support vector [8] logistic regression [2], and decision making [7]. For performance analyzes, four different performance assessment

measures [2], i.e. precision, accuracy, recall and F -measure [6] [4]. The data sets in the public PROMISE [9] data repository are used for testing. The metrics are more precise when drained from well - defined product and intermediate product completion criteria [10].

To promote business objectives, the measurement activities should be designed and they should provide effective and cost-effective decision-making information. Technology or software product per se can neither be effective nor practical without measurements. We analyze the software in order to understand its behavior with respect to both time and space to improve, if derived so.

To quantify the software products, software development resources and/or processes, software metrics are used. This involves items that can be measured directly e.g. lines of code and measurement calculated items such as the quality of the software. Metrics must have well defined goal and must be reviewed regularly and acted upon. Metrics will be maintained and not perceived as a burden when the raw data, used to construct the metrics, are recorded as a natural part of work/process. In the field of software development, software metrics are collected at various stages in the development cycle, and utilized to evaluate the quality of a software product. They are also considered as the most critical factors to identify potentially error-prone modules in software systems, so that extra development and maintenance effort can be measured in those modules.

## II. DETAILS OF PROPOSED SOFTWARE METRICS

**Metric 1:**

**DEFECT_FIXED_FOCUS%_METRIC (DFF%)**

**Definition:** It demonstrates the relation of fixed defects (i.e. detected and corrected) with the total defects in a project.

**Formula Used**:

$$DFF\% = \left[\frac{TD - TDNF}{TDNF}\right] * 100$$

**Variables Used:** TD denotes the "total number of defects for the whole project" whose status is Closed, New, Verified, Approved, Open, Resolved, Work in progress.
TDNF denotes the number of defects associated with the whole project which are not fixed or closed.

**Effect:** As testing proceeds, defect fixed focus should approach to unity.

**Metric 2:**

**ERROR_FIXED_EFFECTIVENESS% (EFE%)**

**Definition:** It demonstrates the relation of actual count of man hours (the fix took for implementing defect) with estimated count of man hours (it took for implementing the fix).

**Formula Used:**

$$EFE\% = \left[\frac{TFH}{TEFH}\right] * 100$$

**Variables Used:** TFH means the actual count of man hours (the fix took for implementing the defect).
TEFH means the estimated count of man hours (it took for implementing the fix for the defect).

**Effect**: Higher the percentage of ERROR_FIXED_EFFECTIVNESS, more the testing efficiency of the system.

**Metric 3:**

**DEFECT_HOW_FOUND_FIXED% (DHFF %)**

**Definition:** It is the metric that captures the relation of the total count of weighted defects which are fixed and the total count of this type of defect in the product.

**Formula Used**:

$$DHFF\% = \left[\frac{W_p}{W_p + W_v}\right] * 100$$

**Variables Used**: $W_P$ = weighted defect count of a specific problem type which are closed.
Wv = weighted defect count a specific problem type which are not closed.

**Effects**: higher the number shows: defects higher ratio was closed of the "appropriate" problem type, to drive out its defects; the higher is the effectiveness for this problem type.

## III. DATA COLLECTION

A list of variables for which data collection is required in this phase has been drawn on the basis of the definitions of the metrics. It can be extracted from the NASAs Metric Data Program website — Access to a repository that provides the data for such projects. Three projects as KC1, KC3 and KC4 are from The Metrics Data Repository [11], for proposed metrics' analysis and calculation. The projects data contained various files categorizing the data with respect to certain modules along with their attributes and the different defect attributes connected with those modules. Each project consist of the same set of files, hence sample data of one project (KC3) is mentioned. The listings of the various files used from the data of three projects are as follows:

The main tables and their corresponding fields which were used for the projects process along with the field details are as follows:

**Table1.** Field descriptions for product module metrics

| | |
|---|---|
| **MODULE** | The unique product numeric identifier |
| **ERROR_COUNT** | The defects count associated with a module. |
| **LOC TOTAL** | Number of lines of a module. |

**Table2.** Field descriptions for static defect data

| | |
|---|---|
| **DEFECT_ID** | Defect numeric identifier which is unique. |
| **SEVERITY** | Severity of defect. |
| **STATUS** | Current status of the defect. |
| **CLOSE_REASON** | The reason for error report closure. |
| **EST_FIX_HOURS** | The estimated number of man hrs. "Took to implement fix". |
| **FIX_HOURS** | The actual no. of man hrs. "The fix took to implement." |
| **HOW_FOUND** | The defect found stage. |
| **PROBLEM_TYPE** | The error report closure reason. |
| **SLOC_COUNT** | Count of SLOC added / changed. |

**Table3.** Field descriptions for defect product relations

| | |
|---|---|
| **MODULE_ID** | Module numeric identifier which is unique. |
| **DEFECT_ID** | Unique associated defect numeric identifier. |

Here the concept of weighted defect number has been inducted & used frequently. Defect information is used to derive weighted defect data. Real in-house testing initiated a very important factor of software quality [12] i.e. severity of defects which tells the seriousness and importance of defect. if any defect is opened then it was correlated with severity in the form of a number (i.e. an integer between 1to 5, 1 represents : most serious defects and 5 represents : least serious defects).To integrate this aspect, instead of merely defect numbers ,weighted defect numbers were used. Each severity is assigned a weight, i.e., severity one defects assigned weight 10, severity two defects assigned weight 7, severity three defects assigned weight 5, severity four defects assigned weight 3, and severity five defects assigned weight 1 . For each severity level, W (weighted defect number) is the total of number of defects, which is multiplied by correlated weights. Table 4 shows the data to explicate the calculation of W. e.g., suppose the following distributions of defects are associated with product:

**Table4.** Example Weighted Defect Numbers

| **Severity** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| Weighted Assigned | 10 | 7 | 5 | 3 | 1 |
| Number of Defects | 11 | 9 | 12 | 25 | 82 |

Here, the calculated value of weighted defect number W is:

W = (11*10) + (9*7) + (12*5) + (25*3) + (82*1) = 390

## IV. Result of Proposed Metrics

The evaluation of the derived/proposed set of metrics over real life projects conducted by the NASAs MDP team. Metrics value can be calculated by applying the value of each variable to the respective metrics. Table 5, 6 and 7 shows calculated/output values and compare them for all the three projects of NASAs Metrics Data Program Repository on each of the derived/proposed metrics.

**Metric1:**

**DEFECT_FIXED_FOCUS%_METRIC (DFF%)**

**Table5.** Result of Defect Fixed Metrics

| **Proposed Metrics** | **KC 1** | **KC 3** | **KC 4** |
|---|---|---|---|
| DEFECT_FIXED_FOCUS % _METRIC | 79.72% | 67.4% | 89.32% |

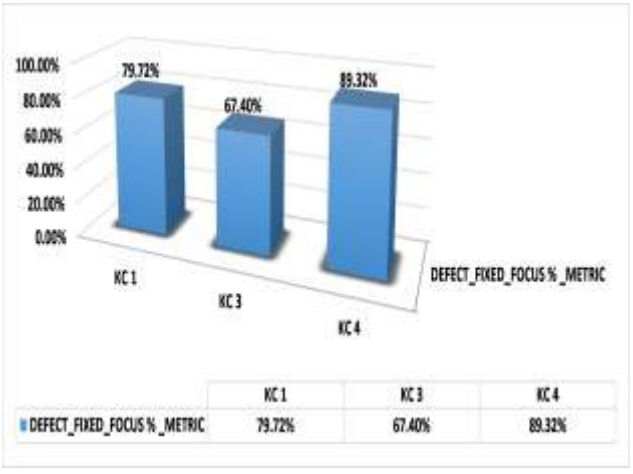


**Figure1.** Defect Fixed Focus Metric

**Metric 2:**

**ERROR_FIXED_EFFECTIVENESS% (EFE%)**

**Table6.** Result of Error Fixed Effectiveness Metric

| Proposed Metrics | KC 1 | KC 3 | KC 4 |
|---|---|---|---|
| ERROR_FIXED_ EFFECTIVENESS% | 83.57% | 96.24% | 99.94% |


**Figure2.** Error Fixed Effectiveness Metric

**Metric 3:**

**DEFECT_HOW_FOUND_FIXED% (DHFF%)**

**Table7.** Result of Defect How Found Fixed Metric

| DEFECT_HOW_FOUND_FIXED% (DHFF%) | | | |
|---|---|---|---|
| **HOW FOUND** | **KC1 (%)** | **KC3 (%)** | **KC4 (%)** |
| Acceptance Test | 78.51 | -------- | 100 |
| Analysis | 86.39 | 97.43 | 97.14 |
| Customer Use | 67.79 | 62.82 | 73.91 |
| Demo | 100 | -------- | ------ |
| Engineering Test | 85.69 | 36.36 | 95.45 |
| Inspection | 88.01 | 0 | ------ |
| Mission(Critical, Success, Essential) | 86.95 | -------- | ------ |
| Planned Test | 82 | 51.21 | 99.05 |
| Regression Test | 80.58 | 100 | 93.02 |
| Release_I&T | 85.71 | 100 | 88.88 |

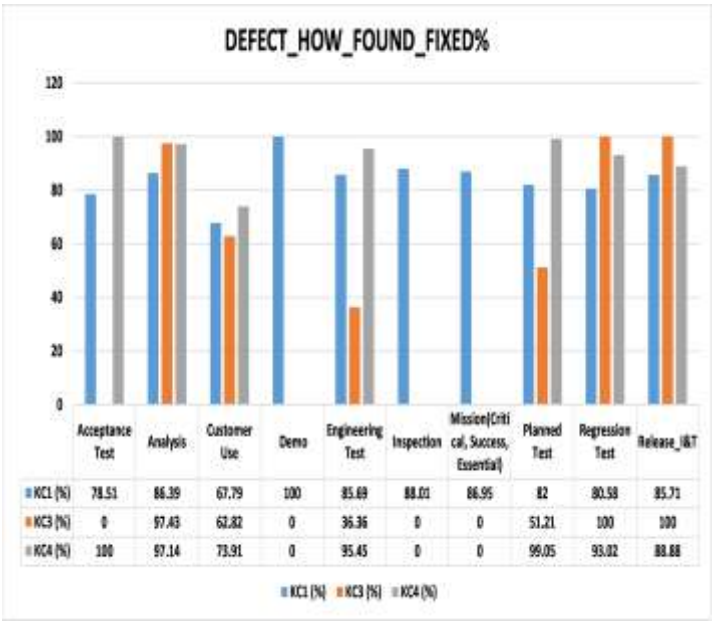


**Figure3.** Defect How Found Fixed Metric

## V. METRICS RESULT OBSERVATIONS

Here are certain observations and discussions with respect to individual metrics:

**Metric 1:** In an ideal case, as testing proceeds, defect focus should approach to unity (that is, all defects found were software defects and resulted in corrections to the software). From above calculated three project's defect focus, maximum defect focus is for project KC4.

**Metric 2:** Trends shows that higher the percentage of DEFECT_FIXED_EFFECTIVENESS, more the testing efficiency of the system. In the above three projects, the maximum DEFECT_FIXED_EFFECTIVENESS is with project KC4.

**Metric 3:** The expected result for this metric states that higher the percentage DEFECT_HOW_FOUND_FIXED no., representing defects higher ratio i.e. closed of the "appropriate problem type"; the higher the "effectiveness "for such problem type to conclude the defects. From above calculated %DEFECT_HOW_FOUND_FIXED for the three projects, project KC1 has the maximum value for %_DEFECT_HOW_FOUND_FIXED.

## VI. CONCLUSION

The proliferation of avoiding failure of software/project gives a pressing need for the introduction of software testing metrics. Hence, the existing metrics in the field of software testing focusing on product level and process level has been surveyed. In particular, the metrics were analyzed in respect of the quality improvement, cost control coverage effectively, time and risk reduction, test progress curve, defect arrivals, defect backlog, Complexity measure, test quantification. The survey doesn't deliberate the explicit concerns of the distinct metric as single

metric alone isn't adequate to gain in the "quintessence" of efficiency of testing. This survey uncovers gaps in existing research and offers a historical view. The existing literature is based on existing metrics' uniform representation and supported by structured information storing. Consequently, the result of survey signifies that for software product management, testing metric suite, which is chosen carefully can be very beneficial which can also offer helpful information to test managers for future improvements and decision making. Apart from this, at product level, a software testing metrics set have proposed and derived spontaneously. Empirical data was collected from the website of NASAs Metric Data Program from which the result for individual metric has been calculated, represented graphically and compared against the expected output or as the general trend should go. The results are quite encouraging and measure performance of individual factors, affecting the software product.

## REFERENCES

[1] Rathore, S. S. and Kumar, S. (2017). Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems, 119:232–256.

[2] Arar, O. F. and Ayan, K. (2016). Deriving thresholds of ¨ software metrics to predict faults on open source software: Replicated case studies. Expert Systems with Applications, 61:106–121.

[3] Yang, X., Lo, D., Xia, X., and Sun, J. (2017). Tlel: A twolayer ensemble learning approach for just-in-time defect prediction. Information and Software Technology, 87:206–220.

[4] Turhan, B., Mısırlı, A. T., and Bener, A. (2013). Empirical evaluation of the effects of mixed project data on learning defect predictors. Information and Software Technology, 55(6):1101–1118.

[5] Zhao, Y., Yang, Y., Lu, H., Liu, J., Leung, H., Wu, Y., Zhou, Y., and Xu, B. (2017). Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. Automated Software Engineering, 24(2):393–453.

[6] Kumar, L., Misra, S., and Rath, S. K. (2017). An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Computer Standards & Interfaces, 53:1–32.

[7] Ghotra, B., McIntosh, S., and Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the 37th International Conference on Software Engineering-Volume 1, pages 789–800. IEEE Press.

[8] Erturk, E. and Sezer, E. A. (2015). A comparison of some soft computing methods for software fault prediction. Expert Systems with Applications, 42(4):1872–1879.

[9] Menzies, T., Krishna, R., and Pryor, D. (2015). The promise repository of empirical software engineering data (2015).

[10] Prakash, P. (2018). 'Using weighted defects metrics to improve software quality: An analysis and review'. International conference on recent trends nd advances in computer science and engineering, LIET, Alwar, Rajasthan, India, 14-15 April, 2018, pages 50-53.

[11] http://mdp.ivv.nasa.gov/repository.html

[12] Yanping Chen, Robert L. Probert, Kyle Robenson "Effective Test Metrics for Test Strategy Evolution" Proceedings of the 2004 Conference of the centre for Advanced Studies on Collaborative Research CASCON'04,2004.

[13] DeMarco, T. (2013), 'Controlling Software Projects: Management, Measurement and Estimation.' ISBN 0-13-171711-1.

## AUTHOR'S BIOGRAPHIES

**Piyush Prakash** is a PhD student in the Department of Computer Science, Mewar University. He earned his MCA & M.Tech degrees from West Bengal University of Technology and Maharshi Dayanand University, Rohtak, India. His research interests include software testing and software engineering.

**Sarvottam Dixit** earned his PhD from Agra University, India in 1990. Currently, he is serving as the Professor and Advisor to Chancellor at Mewar University, India. His general research interests includes artificial intelligence, software engineering, multiagent etc.

**S. Srinivasan** completed his PhD from the Madurai Kamraj University, Madurai in 1978. Presently, he is a Professor at the PDM University, Bahadurgarh, Haryana – 124507. His research interests include artificial intelligence, software engineering, software testing etc.