IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

Use of Software Metrics on Software Development Projects Life Cycles in OOE to Improve Software Quality

Piyush Prakash[1*], Sarvottam Dixit[2], S. Srinivasan[3]

[1,2]Department of Computer Science, Mewar University, Gangrar, Chittorgarh, Rajasthan, India
[3] Department of Computer Science & Applications, PDM University, Bahadurgarh, Haryana, India

[*]*Corresponding Author: piyush19sept@gmail.com, Tel.: +91-9466429362*

DOI: https://doi.org/10.26438/ijcse/v7i5.18581864 | Available online at: www.ijcseonline.org

Accepted: 18/May/2019, Published: 31/May/2019

***Abstract*—** Software quality metrics are part of software metrics focusing primarily on process, product and project quality aspects. Software quality metrics primarily focus on software measurement and its development process. The main objective of software testing is to enhance the quality of software. Many experts in the field of software testing propose various number of metrics. With the help of these we can detect trends and can prevent problems in efficient cost control, quality improvements, time and risk reduction with their probable solutions. Thus, in the global competitive market, it facilitates ensuring and achieving optimal business goals.

***Keywords*—**Software Testing, Software Testing Metrics, Software Testing Product Metrics, Software Testing Process Metrics.

I. INTRODUCTION

Software testing plays a major role in the construction of critical and complex software systems and in the development of quality software engineering [4]. There's the question: Why are we testing? The two main reasons for this are: to assess acceptability or quality and to identify issues. While it is a costly business, the non-testing of software can result in costs that can be far higher than that of testing [1]. The primary benefit of the testing is that it can execute the developed software in the ideal environment and the test results along with test cases gives confidence that the software must meet user requirements and will perform as intended [12]. Object-oriented Software Engineering, classically refers to OOSE, is the object modeling methodology in software architectures [19].

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 [9]. Tom DeMarco specified, “You can’t control what you can't measure.” [3].

In general, software metrics can be categorized as process metrics, product metrics and the 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 [15]. If there is any bug undetected, severe damage can be caused in many software systems like financial systems, banking, satellite systems, medical systems etc. Therefore, testing is actually very important in software systems development [2]. In the event of a software failure prediction for inter releases software, the data of the previous version of software used for the classifier training may not always have the same granularity as the test data, which could be a major problem. In the cross project failure prediction, the same scenario may occur. Hence, there is a need to bring the metrics at the same level. In this paper, 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 [17], [16], logistical regression [2], [18], artificial neural networks [10], and common machine learning techniques [18]. Three machine learning techniques were employed in this paper: logistic regression [2], decision making [6] and support vector [5] .For performance analyzes, four different performance assessment measures [2], i.e. precision, accuracy, recall and F -measure [16] [10]. The data sets in the public PROMISE [11] data repository is used for testing. The metrics are more precise when drained from well - defined product and intermediate product completion criteria [14].

Software development life cycle (SDLC) is generally much more vast and big life cycle in real that what it is exactly defined in books. We have categorized software development life cycle in slightly a different way which comprises the following phases in its life cycle-

• Initiation
• Planning
• Resource Management
• Configuration and Change
• Reviews and Inspections
• Customer Engagement
• Monitoring and Control
• Closure

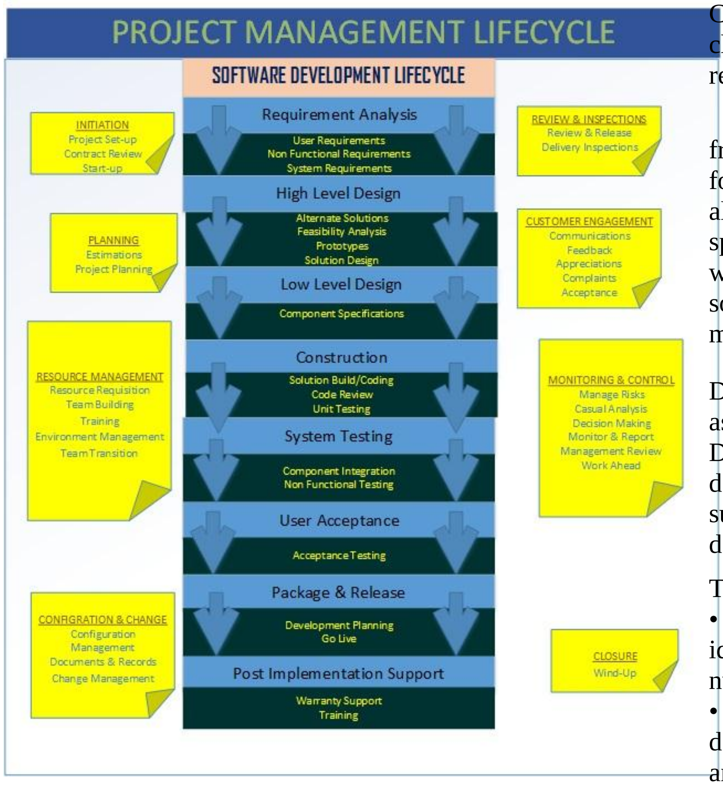


Figure 1. Detailed Life Cycle of Development Projects

This paper is comprised of three sections in which Section I contain the introduction of Software metrics, software testing and SDLC project management life cycle. Section II contain the analysis of software process metrics and the results of proposed software process metrics. Section III deals with conclusion report in which future scope of the proposed metrics introduced has been mentioned.

II. ANALYSIS AND RESULTS OF SOFTWARE METRICS

A. Metrics for Software Development Projects

The metrics are the tools to measure the size, complexity, defects, bugs fixing and various such more attributes in software. The metrics for development projects play a very important role as metrics can be used to detect the hidden defects which give a clear blueprint of the efficiency of a software project newly developed from scratch. The following are the software development project metrics-

(i) Defects Metric (DM)
(ii) Total Defect Suppression Effectiveness Metric (TDSEM)

The metrics can be applied on the projects such as development projects, conversion projects and maintenance projects. There is some implementation of few metrics on the basis of the data collected from an IT company.

(i) Defect Metric (DM)
Objectives of the Measure: -To measure and monitor the quality of the product. Measuring defects provides us a basis for model based defect prediction and management. Collecting post-delivery metrics will provide us a measure of client satisfaction. The defect data will be used to update review checklists, standards and training materials.

Implementation of Defect Metric: -any anomaly from the expectations of users is treated as defect. The norms for defect data collection as given below will be applied in all defect related metrics. A defect metric can be used specifically for development and conversion projects. Here we are showing the implementation of Defect Metric for software conversion metric as the data collected for this metric is relevant to conversion projects.

Data on the defects found in the programming products such as the SRS, code and test specifications should be collected. During the analysis, design and coding phases, data on the defects will be collected at the system level or at the subsystem level. From the coding phase onwards, data on the defects will be collected at the program level.

The following data has been collected for Defect Metric: -
• Error ID: - The Defect ID of defect metric is used for identification of a particular defect in the coding. The coding number can also be specified along with Defect ID.
• Date of Login: - It is the date which is required to determine the date at which that particular defect is noticed and recorded.
• Weighted Defect: - defect information is used to derive the data. It is found that the severity of defects (that is how important or serious the defect is) is important to software quality when learning from real- life in-house tests. The seriousness of a defect is related to a number. This is a

number between 1 to 5,where 1 is the most severe deficiency while 5 is the least severe one. In Table 1 we define all severity.

Table1 Description of Each Severity

SEVERITY	WHAT IT MEANS
1	The basic product functionality failing or product crashes.
2	Unexpected error condition or a functionality not working.
3	A minor functionality is failing or behaves differently than expected.
4	Cosmetic issue and no impact on the users.
5	Least Serious Defects

• Status of Error: - It is basically done to make sure that the errors or defects are closed and analyzed by more than one person specially the persons other than the testing team.

• Root Cause of Errors: - The root cause of the defects is also recorded so as to make it correct and also for the future reference. The root cause can be any of the following:-

- Unwritten assumptions
- Fault state transactions
- Missing logic
- Conflicting requirements
- Unable to handle missing data
- Incorrect sequences
- Incorrect defined data types
- Faulty software requirements
- Faulty interface design
- Faulty detailed design

The data collected for above metric is shown in the following table: -

Table 2 Data Collected for defects

Defect Logged for the month Sept-Oct 2018				
Error ID	Date of Login	Weighted Defects (Severity)	Status of Error	Root Cause of Errors
D001	4-Sep-18	4	Closed	Unwritten assumptions
D002	4-Sep-18	2	Closed	Fault state transactions
D003	5-Sep-18	2	Closed	Missing logic
D004	6-Sep-18	3	Closed	Conflicting requirements
D005	7-Sep-18	2	Closed	Unable to handle missing data
D006	7-Sep-18	3	Closed	Incorrect sequences
D007	11-Sep-18	3	Closed	Incorrect defined data type
D008	11-Sep-18	3	Closed	Faulty software requirements
D009	12-Sep-18	3	Closed	Faulty interface design
D010	13-Sep-18	3	Closed	Faulty detailed design
D011	14-Sep-18	2	Closed	Missing logic
D012	19-Sep-18	3	Closed	Conflicting requirements
D013	20-Sep-18	2	Closed	Unable to handle missing data
D014	21-Sep-18	3	Closed	Incorrect sequences
D015	21-Sep-18	4	Closed	Unwritten assumptions
D016	21-Sep-18	3	Closed	Incorrect sequences
D017	21-Sep-18	4	Closed	Unwritten assumptions
D018	24-Sep-18	2	Closed	Fault state transactions
D019	24-Sep-18	2	Closed	Missing logic
D020	25-Sep-18	3	Closed	Conflicting requirements
D021	25-Sep-18	2	Closed	Unable to handle missing data
D022	26-Sep-18	3	Closed	Incorrect sequences
D023	28-Sep-18	3	Closed	Incorrect defined data type
D024	28-Sep-18	3	Closed	Faulty detailed design
D025	1-Oct-18	2	Closed	Missing logic
D026	3-Oct-18	3	Closed	Conflicting requirements
D027	4-Oct-18	2	Closed	Unable to handle missing data
D028	4-Oct-18	3	Closed	Incorrect sequences

The next table shows the defects list along with the Error ID

Table 3 Defect Causes

Root Cause	Count of Error

Unwritten Assumptions	3
Fault State Transactions	3
Missing Logic	4
Conflicting Requirements	4
Unable to Handle Missing Data	4
Incorrect Sequences	5
Incorrect Defined Data Type	2
Faulty Software Requirements	1
Faulty Interface Design	1
Faulty Detailed Design	2

The following chart will show the defects and percentage of possible cause

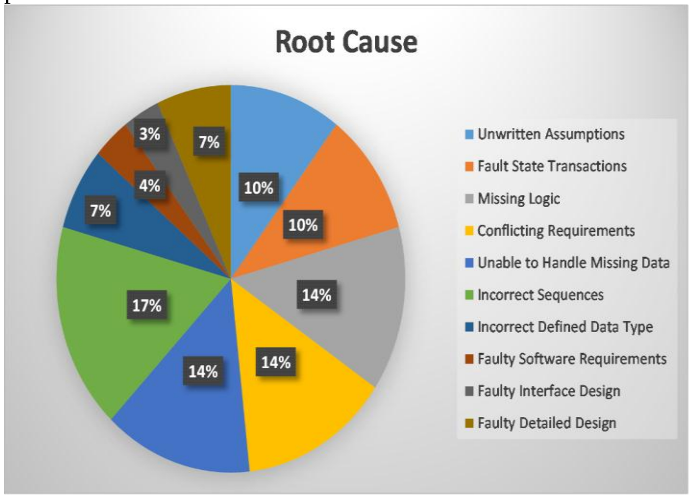


Figure 2. Percentage of Defect causes in overall Life Cycle

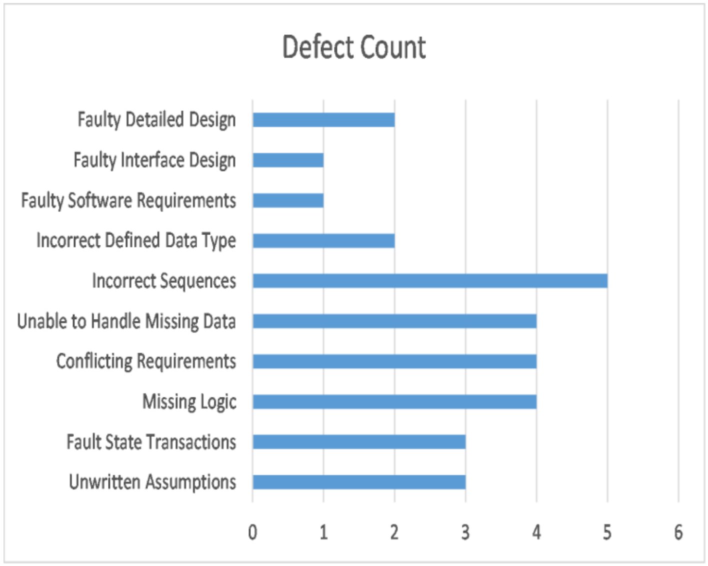


Figure 3. Defect Count Measurement

(ii) Total Defect Suppression Effectiveness Metric (TDSEM)

This is a mandatory metric to be collected by all projects which can be conversion and development. These are requires to be collected for deliverables upon acceptance.

Objectives of the Measure: -The objective of this measure is to Increase Defect Suppression.Specifically, it answers the question - "What is the efficacy of the defect detection process currentl y known before release?" Analysis of escaped defects feeds back into the checklists and standards of appropriate life cycle stages and technologies.

What to collect: - Data on the number of defects accepted for a deliverable detected prior and during acceptance for a deliverable. Defects for various severity levels need to be observed at the minimum for acceptance defects so that can gauge the productiveness of pre-delivery excellence guaranteed processes.

For code deliverable, if the client has additionally performed code review, include the code review defects in acceptance defects; otherwise take only test defects into consideration. For analysis/design documents, consider the review defects found in the pre-delivery and acceptance phases.

In case the development is done offshore, all errors reported by onsite team and client, which have been accepted may be measured separately, but both need to be included under acceptance errors. This is because this metric is intended to address all escaped defects after the delivery from the development site.

In some projects, the contract and plan define that some kinds of testing will be carried out onsite only, by either the onsite team or the customer. For example, this could be due to integration with components not developed as part of the offsite project or machine access limitations from offsite. Errors detected in these activities should be reported back to offsite in the respective phase, and classified as pre-delivery defects.

When to collect: -This metric should be collected for every anticipated outcome as specified in the plan. In the acceptance phase, data will be collected during acceptance test of the deliverable.

Factors affecting the Metric: -Some of the factors that affect the number of defects are:

- Complexity
- Skill Levels
- Applicability and appropriateness of Standards
- Efficiency of V & V process
- Automation
- Degree of adherence to process
- Extent of conformity among the acceptance test plan Phase and baseline requisites.

Metrics for Total Defect Suppression Effectiveness Metrics (TDSEM) is

$$\text{TDSEM}=\frac{\sum_{i=1}^{n} dp}{\sum_{j=1}^{n} di}*100$$

Where dp means total number of defects detected in detection phase and di means total number of defects injected in each phase.

Implementation of Defect Phase Suppression Metric: This is a mandatory metric to be collected by all conversion and development projects and requires to be collected for deliverables upon acceptance. The objective of this measure is to Increase Defect Suppression. - This metric should be composed for every anticipated outcome as specified in the plan. In the acceptance phase, data will be collected during acceptance test of the deliverable. The data collected for the implementation of this metric contains the following: -

- Injection Phase: - As the objective of this metric is to increase to defects so as to determine the effectiveness of the defect detection process, hence the defects are introduced in the injection phase. Injection phases will remain the usual SDLC phases such as: -
  - Requirement Analysis
  - High Level Design
  - Low Level Design
  - Construction and Unit Testing (UT)
  - System Testing (ST)
  - User Acceptance Testing (AT)
- Detection Phase: - The following items will be used for calculation defect detection percentage.
  - Total Defect Suppression: - This will be the count of total defects contained in each phase.
  - Phase Suppression Effectiveness: - This will be the percentage of the defects detected in a particular phase in respect to the total defects injected in each phase.
- Defect Density Mean: - The next step is to calculate the defect density mean for each phase. The Lower and Upper limits for defects are already specified. The mean is then compared with the specified limit and then the effectiveness of defect detection process is ascertained.

The following table contains the data collected related to this metric:-

Table 4 Data Collection for Defect Density

Injection Phase	Detection Phase →						Total Errors Injected in Each Phase	TDSEM
	Requirement Analysis	High Level Design	Low Level Design	Construction & Unit Testing	System Testing	User Acceptance Testing		
Requirement Analysis	134	34	23	12	4	0	207	64.73%
High Level Design	NA	86	12	30	2	0	130	66.15%
Low Level Design	NA	NA	190	20	14	10	234	81.19%
Construction & Unit Testing	NA	NA	NA	65	14	6	85	76.47%
System Testing	NA	NA	NA	NA	30	7	37	81.08%
User Acceptance Testing	NA	NA	NA	NA	NA	6	6	100%
Total Errors Detected in all Phase	134	120	225	127	64	29	699	

The next table will focus on the Defect Density Mean and the lower and higher range

Table 5 A glance at Defect Density Mean

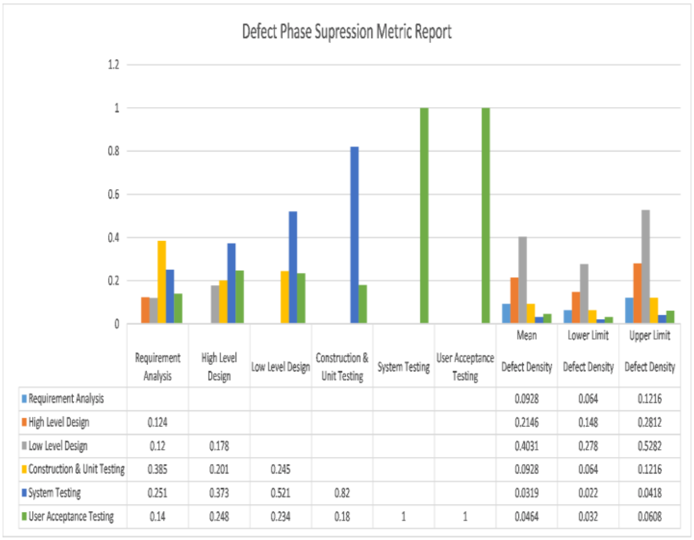
Platform	SDLC Phase	Defect Density (Defects/FP) Predicted Mean	Defect Density (Defects/FP) 90% Confidence Interval	
			Lower Limit	Upper Limit
C# (1100-2200 FP)	Requirement Analysis	0.0928	0.064	0.1216
	High Level Design	0.2146	0.148	0.2812
	Low Level Design	0.4031	0.278	0.5282
	Construction & Unit Testing	0.0928	0.064	0.1216
	System Testing	0.0319	0.022	0.0418
	User Acceptance Testing	0.0464	0.032	0.0608

The next table also shows the defect density on a broader way: -

Table 6 Defect density mean of each phase with lower and upper limits

C# (1100-2200 FP)	Requirement Analysis	High Level Design	Low Level Design	Construction & Unit Testing	System Testing	User Acceptance Testing	Defect Density		
							Mean	Lower Limit	Upper Limit
Requirement Analysis							0.0928	0.064	0.1216
High Level Design	0.124						0.2146	0.148	0.2812
Low Level Design	0.12	0.178					0.4031	0.278	0.5282
Construction & Unit Testing	0.385	0.201	0.245				0.0928	0.064	0.1216
System Testing	0.251	0.373	0.521	0.82			0.0319	0.022	0.0418
User Acceptance Testing	0.12	0.248	0.234	0.18	1	1	0.0464	0.032	0.0608
Total	1	1	1	1	1	1	0.8816	0.608	1.1552

The following chart shows the results of the above metric implementation based on the data collected.



	Requirement Analysis	High Level Design	Low Level Design	Construction & Unit Testing	System Testing	User Acceptance Testing	Mean Defect Density	Lower Limit Defect Density	Upper Limit Defect Density
🟦 Requirement Analysis							0.0928	0.064	0.1216
🟧 High Level Design	0.124						0.2146	0.148	0.2812
⬜ Low Level Design	0.12	0.178					0.4031	0.278	0.5282
🟨 Construction & Unit Testing	0.385	0.201	0.245				0.0928	0.064	0.1216
🟦 System Testing	0.251	0.373	0.521	0.82			0.0319	0.022	0.0418
🟩 User Acceptance Testing	0.14	0.248	0.234	0.18	1	1	0.0464	0.032	0.0608

Figure 4. Results of Implementation of Defect Suppression Metric

III. Conclusions

This discussion imitates the importance along with the advantage of usage of defect density throughout the method of package development. However defect density is treated insignificant as well as unessential by many software engineers, it's one amongst the finest approach to decide the area unit as that are extremely plagued by errors and bugs.

A recognised business common place, Defect Density may be a metric that describes that, "The additional defects within the package, the inferior the standard is". Consequently, it compute the defects that resides within the package and divided by the whole size of package or a module being measured. By the help of this metric, package engineers, testers, developers and additional can estimate the differentiate defects as well as testing effectiveness in software modules.

Furthermore, they will conjointly measure the testing and the need of rework because of the discovered bugs and defects. Hereafter, by accomplishing defect density, one cannot solely compute or estimate the defects per developed package, however, also can guarantee its performance, quality, effectiveness and more.

References

[1] Aggarwal, K.K., Singh, Y., Kaur, A., and Maihotra, R. (**2005**), '*Software Reuse Metrics for Object-Oriented Systems*' , Proceedings of the 2005 Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA '05).

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

[4] Dhavachelvan,P., V.S.K. Uma, Venkatachalapathy G. V. (**2006**) '*A new approach in development of distributed framework for automated software testing using agents'* , **Volume 19, Issue 4**.

[5] 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**.

[6] 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.

[7] Honglei, T., Wei, S., and Yanan, Z. (**2009**). The research on software metrics and software complexity metrics. In Computer Science-Technology and Applications, 2009. IFCSTA'09. International Forum on, volume 1, pages **131–136**. IEEE.

[8] Kamei, Y. and Shihab, E. (**2016**). Defect prediction: Accomplishments and future challenges. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, volume 5, pages **33–45**. IEEE

[9] Karner, C. and Bond, W.P. (**2004**), '*Software Engineer Metrics: What do they measure and how do we know?*' Proceeding of the 10th International Software Metrics Symposium, Metrics.

[10] 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**.

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

[12] Ogasawara, H., Yamada, A. and Kojo, M. (**1996**) '*Experiences of software Quality Management Using Metrics through Life cycle'*, Proceedings of ICSE.

[13] Paul, C. (**2002**) '*Software Testing - A Craftsman's Approach Second Edition*' CRC Press.

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

[15] 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**.

[16] 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.**

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

[18] 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**.

[19] Aanchal, Kumar S. (**2013**). '*Metrics for Software Components in Object Oriented Environments: A Survey'* International Journal of Scientific Research in Computer Science and Engineering, Volume-**1**, Issue-**2**, March-April-**2013**: pp. **25-29.**

Authors Profile

Mr. 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.

Dr. 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.

Dr. S. Srinivasan e 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.