International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 3 | ISSN : 2456-3307

A Detailed Phasewise Study on Software Metrics : A Systematic Literature Review

Ausaf Ahmad[1], Tamanna Siddiqui[2], Najeeb Ahmad Khan[3]
[1]Department of Computer Science, Aligarh Muslim University, Aligarh, Uttar Pradesh, India
[2]Department of Computer Science, Aligarh Muslim University, Aligarh, Uttar Pradesh, India
[3] SRB International Private Limited, Noida, Uttar Pradesh, India

ABSTRACT

As we all aware that the measurement becomes an essential part of all engineering fashion and software development lifecycle is no exception. Software metric applied to create quantitative/qualitative decisions regarding development process and product as well as in risk assessment. The verities of software metrics have been developed that have utilized by different project personnel at different phase of software development to fulfill the objective in an effective way. In this paper, we discuss about software measurement and metrics along with their standard role in the software development. A systematic phase wise studies on the metrics used in different phase of software development lifecycle have been carried out.

Keywords : Software Metrics, Measurement, Software Development, Complexity, Lines of Codes, Systematic Review.

I. INTRODUCTION

Since last two decades, human dependency on software system has been increased exponentially as peoples are working around direct or indirect influence of electronic artifacts and software used in different services for different intention. Software quality and reliability modeling is extremely important as the software is being utilized in diverse areas of several applications. M.R. Lyu illustrates the impact of software failure encountered throughout the globe [1]. The consequences of failures may lead to economical, time and effort losses. Therefore, predicting the quality and reliability before the product release is a serious challenge and it has become an interesting research field of software engineering.

The field of software metrics has evolved continues mainly for two perspectives. First one for software development personnel to be able to effectively manage the development process under all defined and possible constraints. For example, estimation of resources and time required to develop the product successfully. Other for the researchers, who always concentrate on defining objective and establishing metrics to estimate the software attributes to get a better understanding about software engineering [2]. There is no standard metric to measure the all the attributes as this field of software engineering is changing continuously and new metrics are always being proposed. Researchers are continuously contributing to enhance their existing metrics to accommodate the new measures and it is really a challenging task [5].

One notable point is that a lot of research has been done on software measurement and a number of measuring techniques have been suggested and condescend, a lot of computing tools have been designed also, Which arises the possibility of losing factual information, mislead and get confused. Due to this cause, it becomes necessary to adhere certain specific and well-defined techniques and methods for investigating the existing literature. We studied many literatures and find out that software measurement to be among the youngest disciplines of software engineering because presently, in this field the introduction of new terminology, concepts and techniques remain being defined [6]. Since there is no method that measure all attributes.

Different metrics employed used in different states of the software development practice to measure the attributes that affect the software projects, product and process.

In concern to above identified issues, this effort carries out an organized literature review with a predetermined search tactic to summarize and systematize the contributions of the state-of-art towards software measurement. The organization of this document is as follows. Section 2 describes the background Information of software metrics. Section 3 describes the methodology followed to conduct the studies. Section 4 describes the objective of metrics. Section 5 reports various metrics sets used in different phase of development lifecycle. Section 6 draws conclusions about the studies carried out in this paper.

II. RELATED WORK

Unterkalmsteiner [7] conducted a systematic literature review study to examine a variety of software metrics applied for the purpose of software process improvement (SPI), evaluation and assessment strategies. To conduct the systematic review they select and analyze total 148 research papers that were published from the year 1991 to 2008. The studies ware focus mainly on all the factors which made affect the software development, product and process like effort estimation, defect, time, cost, product quality, process quality, productivity, client satisfaction along with additional success pointers. The summary of this literature review demonstrates that 39% primary studies pay attention on quality measure, 38% on estimate approximation as well as 35% on productivity. The classification of included papers is based on quality characteristics like functionality, reliability, reusability, portability, maintainability, and efficiency. The measurements are mainly categories into three categories, i.e. software product, project and organization where product and project categories provide great opportunities for measurement research.

Kitchenham [8] made a literature review to analyze and examine a variety of research studies published on software measurement as well as they investigated the possibility for an aggregation relationship between them. They select 25 research papers published from the year 2000 to 2005. The study reported that there is requirement of assessment and aggregation of finding reported in research. They also suggested that there is also need to use the industrial datasets to identify and address the issues encountered relevant to software measurement in the industry.

Gómez [6] conducted a SLR to summarize the state of art in software measurement. They categorized the identified metrics based on the variety of entities measured, i.e. project, product and process. Moreover, additionally they recognized whether the determined attributes are internal or external. Their report stated that approximately 79% selected research papers focused on product metrics, 12% on project as well as 9% on process metrics. When they mapped these metrics with software development life cycle founded that 48% primary studies focused on initial, 36% and 16% on intermediate and final phase of SDLC respectively. While design phase is most likely (42%) measured phase of the SDLC and the most likely measured attributes are the size and complexity attributes. All these results ware answer to the three research questions, i.e. what to measure?

How to measure and when to measure?. The conclusion of the study stated that there is an absolute need of empirically and theoretically, validation of metrics for mapped these identified metrics to software development phase and process.

III. METHODOLOGY USED

In this phase, we illustrate the design and the execution of the review according to which this review work has been done. We have performed this SLR to summarize the state of the art in the field of software metrics by following the standard guidelines and recommendations for conducting SLR in the field of software engineering [9][12][13]. According to these guidelines and recommendations, the literature review has accomplished in three phases: planning, conducting and reporting phase presented in the fig. 1 also motivated from [14][11].

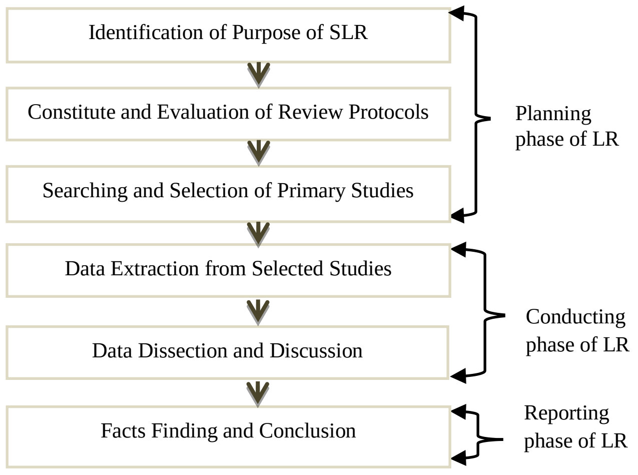

Identification of Purpose of SLR
Constitute and Evaluation of Review Protocols
Searching and Selection of Primary Studies
Data Extraction from Selected Studies
Data Dissection and Discussion
Facts Finding and Conclusion
Planning phase of LR
Conducting phase of LR
Reporting phase of LR

Figure1. Methodology Adapted to Conduct the Literature Review

In the first stage, we identify the purpose and need of a systematic literature review. The objective of performing the systematic review is mentioned in the introduction section. We identified and reviewed the current systematic reviews on the topic in section 2. The review protocol was evolved to perform the review and conduct the research in a systematic way which free from possibility of biasness. Its concern with research questions, searching approach which helps us to find the more related studies. We select many popular digital libraries like IEEExplore, ScienceDirect, ACM Digital library, Google scholar and many conference proceedings for searching the papers and articles related the topic. The data extracted from the selected primary studies concern to the answer of the research question. Data dissection and discussion relates how the data concern with the topic and at the last, facts finding and conclusion are disused.

IV. OBJECTIVE OF SOFTWARE METRICS

Software metric and measurement used as synonyms in software development. Basically, software metric is a way of measuring anything which directly associated with software programs or its actual development. From initial stage, measurement becomes necessary for analyzing the feasibility and find out the status of products, processes followed and resources used in the development. The organization of every single measurement activity ought to be in the manner of instating its plainly defined objectives. Modeler and Norman [15][16] enlisted the objectives of software metrics as

A. Understanding

Metric assist to perceive what is happening throughout the development and the maintenance phase that can be used by the project personnel in managing the development in better ways to deliver high quality software products with reliability, safety and security. Measurement should really be carried out in sequence to obtain a model of process and analyze important relationships between process parameters that contributes towards developing an excellent understanding together with enhanced software projects.

B. Early Problem Identification

Software metric or measure enables to identify problems in early phase or before it occurs and assists to correct the same as a software management strategy. In cases where issues are identified in the early phase of development, leads to resource saving because resolving the problems occurred in later phases of development are more economic, difficult and wastage of resources.

C. Planning/Estimation

Preferable project planning and estimation is another aspect of software metric. A well-planned strategy helps in reaching the specified goal in an effective way. The amount of rework or maintenance, mainly perfective maintenance and adaptive maintenance is a significant reason of failure of estimated budget. As the cost of fixing the defects encountered increases with increase in defects.

D. Quality

Software quality refers to delivery of defect free product, meets stakeholders' expectations, developed in estimated budget and time and is maintainable. There are various ways to think about the quality of the software product. It's a very complicated area and depends on the number of defects as software quality inversely proportional to the number of defects and problems encounter during entire development and maintenance phase.

E. Schedule

The volume of workload, the number of project personnel as well as the way processes deployed are pre-requisite factors that need more focus during schedule preparation to monitor the projects on a regular basis. To enhance the productivity, schedule assists to manage what is happening during development and maintenance [17] assures that if the metrics are not implemented at the beginning of development, may increase the possibility of errors in afterwards stages.

V. STUDY ON SOFTWARE MERTICS

In this section, we discuss here a variety of software metrics used in the various phases of software development in a systemic way.

A. Requirements Phase Metrics

Requirements are brief summary of features and functionalities of the targeted software system, which describes the stakeholder needs. From the client's point of view, the requirement could be clear or sometime hidden, expected or unexpected. The process of gathering software requirements from clients, analyze them and prepare document is referred as requirement engineering [18]. The requirement engineering process includes a set of activities during requirement gathering such as Problem synthesis, Requirements Elicitation, Requirements Analysis and Negotiation [19], Requirements Specification, System modeling, Requirements Verification and Validation, Requirements Documentation and Requirements Management [20]. As the name states, these metrics are employed throughout inspection, elaboration and development stage while the business/enterprise model is to be established.

Table 1. List Of Metrics Used In The Requirement Phase Of Development

Metric	Description
Specification $m_1 = \frac{N_{ui}}{N_r}$	It is a complete description of requirements of a software system which to be developed which demonstrates the functional and nonfunctional requirements both, it may also incorporate a set of use cases that illustrate the user interaction which the system must contain.
Completeness $m_2 = \frac{N_u}{N_i * N_s}$	It states that the requirements must be fully identified in one place without missing any information. It means completeness of requirement measure the aggregate number of functions presently specified

Correctness $m_3 = \frac{N_c}{N_c * N_{nv}}$	It measures the requirements, which are exactly specified as the users expect to get into a software system.
Concise $m_4 = \frac{1}{\text{Size} + 1}$	A requirement report is considered to be concise if requirements describe in short length without affecting other features. This metric can be used to find out the best SRS document between two describing the same.
Understandability $m_5 = \frac{N_{ur}}{N_r}$	It defines the amount of requirements which are clearly understood by all team members especially by the reviewers.
Verifiability $m_6 = \frac{N_r}{N_r + Z}$ $\&Z = \sum_i C(r_i) + \sum_i t(r_i)$	Verification is the cost effective process of confirming that the built software product completely addresses the requirements gathered which is checked by individual or machine.
Internal Consistency $m_7 = \frac{N_u - N_n}{Nu}$	It determines the rate of unique features, which are deterministic provided that SRS might be referred quite as a feature which correlates input states into outputs.
Precise $m_8 = \frac{N_p}{N_p + N_f}$	Preciseness of requirements discus that requirements document should be briefly summarizes the requirements and should not consist of an obscure and unreadable detail.
Not Redundant $m_9 = \frac{N_{fs}}{N_u}$	It measures the rate of unique function, which is not repeated in requirements. Redundant information usually stored at different places, which arises problems while update takes place.

Index:N_{ui}: Number of requirements having identical interpretations, **N_r**: Total number of requirements, **N_u**: Number of unique functional requirements, **N_i**: Number of inputs entailed by the specification document, **N_s**: Number of states in the specification, **N_c**: Total no of correct requirement, **N_{nv}**: Not valid requirement, **Size**: Number of pages in SRS, **N_{ur}**: Number of requirements understood by all team members, **C:** Cost obligatory to verify existence of requirements, **t** : Time obligatory to verify existence of requirements, **N_p** : Number of true positive, **N_f** :Number of False positive, **N_{fs}** : Total number of functions specified.

Obviously, the optimal value of these metrics will be one and for unambiguous requirement, we get its value more closure to one. If its lower the value, a requirement will be more ambiguous, which arises the problems in the latter phases of the development. Apart from these, many other requirement metrics such as design independence, traceability, reusability of SRS and requirements volatility that used in requirements phase to gather quality requirement for producing a better and quality product. Detecting errors in the early phase of development are cost effective process as removal of there one error in later phases of developments arises many errors in different modules of the software product. Consequently, for uniquely identify the requirement, it is necessary to constantly discuss and review the requirement by all team staff and reviewers till they all comprehend and identify the requirements uniquely.

B. Design Phase Metrics

New software products surely be ten times more better and desirable than the existing one for end users to switch along with the failure rate associate with new product introduction is about 35 to 40 percent. Therefore, which way a product is designed is the main key to its success that explains twice impressive as branding of the product [21]. Thus, software design is considered as the procedure by which a designer tends to create the specification of software artifacts, employing a set of primitive components and acceptable constraints to achieve the goal [22]. It may refer to either "all the activities associated on conceptualizing, framing, implementing and finally modifying complex systems" or "all the activities following requirements specification and before programming" [23] and to accomplish these task, some design metric or a set of design metrics takes into consideration that are enlisted below.

McCabe's Cyclomatic Complexity

Thomas J. McCabe, Sr. Considered the source code of a program as a graph, and try to find out the total number of distinct linearly independent routes from one end to another. This cyclomatic complexity employed to signify the complexity of a program and computed through a control flow graph of the program that consists of sets of nodes and edges where node corresponds to indivisible groups or segments of command of a program and a directed edge connects two different nodes as second command tends to be executed subsequent to the first command. When a program becomes larger in complexity and length, the number of paths can't be counted in a short span of time [24]. Consequently, McCabe recommends reckoning the number of underlying routes or all paths consisting of essential path termed as "Cyclomatic Number" denoted by v(G) and defined as Eq. (1)

$$v(G) = E - N + 2P \qquad (1)$$

Where **v(G)**: Cyclomatic Complexity, **E**: Number of edges, **N**: Number of nodes,**P** : Number of connected components or parts.

A programs created using only binary decision nodes need to calculate the number of predicates and increase it by one to get the complexity of the program and formulated as Eq. (2)

$$v(G) = P + 1 \qquad (2)$$

Where **v(G)**: Cyclomatic Complexity, **P**: Number of predicates or binary nodes

It furthermore applied to individual functions, modules and methods within a program and easy to apply.

Information Flow Metric

Kafura and Henry proposed a metric called Information flow that utilized to measure the complexity of the software component [26]. The techniques confirms to figuring out the number of calls to a module, called as fan-in while figuring out the number of calls from a module called fan-out. So the complexity in information flow could be measured by the Eq. (3)

$$C = [\text{procedure length}] * [\text{fan} - \text{in} * \text{fan} - \text{out}]^2 (3)$$

Where **C:** represent the complexity of the module, **procedure length:** full length of the module.

Functions Points (FP) Analysis

Allan J. Albrecht was proposed function point analysis in the mid-1970s that cover-up the difficulties related to lines of code (LOC) to measure the software size that also associated to predict the effort. The objective of function point estimation is briefly mentioned as

- Measures the software size by quantifying the features required by and delivered to stockholders based primarily on logical design.
- Measures the software development and its maintenance task independently of technology employed for implementation.
- Measures the software developments and maintenance uninterruptedly across all projects and business groups.

It concentrates on measuring the features of the product accordance with following five components:

- User Inputs
- User Outputs
- User Inquiries
- Number of Files
- Number of External Interfaces

Function point analysis is widely accepted as a standard metric to measure the software size since 40 years. After the classification of components as one of the above, a ranking or a rating is assigned to the projects as low, average or high. These rankings depend on the number of files referenced and data element types. After all the function point is obtained by the relationship expressed in Eq. (4)

$$FP = UFP + CAF \qquad (4)$$

Where **UFP :** Unadjusted function point and calculated as$UFP = \sum_{i=1}^{5} W_{ij} Z_{ij}$, where the value of j can be either 1 or 2 or 3, depending on the ranking of project and Z_{ij} is count for components i at rank j that is fixed weight assign by the Albreht procedure and calculated by using the weights as given in table 2:

Table 2. Software components and their corresponding weight factors

Software Components	Weight Factors		
	Low	Average	High
No. of user inputs	3	4	6
No. of user output	4	5	7
No. of user inquiries	3	4	6
No.of files	7	10	15
No. of external interfaces	5	7	10

CAF: Complexity adjustment factor and computed as $CAF = 0.65 + 0.01 * \sum F_i$, the value of CAF depends on 14 general system characteristics that rate the regular functionality of the projects.. Every single characteristic incorporates the interlinked details that guide to recognize the amount of influence of the functionality. The degree of influence extent varies on the scale of 0-5 from no influence to strong influence.

In summary, the function point concept facilitates an objective, comparative benchmark that aids in assessment, planning, management and control over software processing and production.

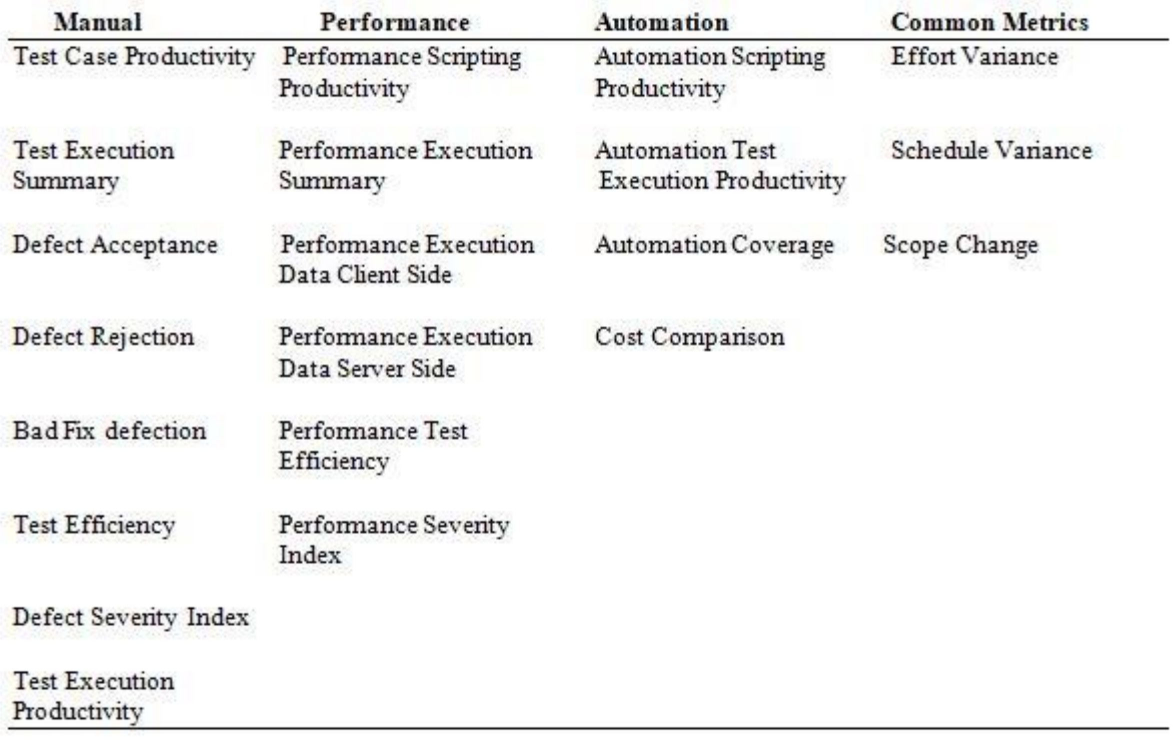
C. Testing Phase Metrics

Various types of testing carried out to produce a quality and defect free software products and services. Depending on types of t testing performed, software testing metrics broadly categories in three major categories.

- Manual Testing Metrics
- Performance Testing Metrics
- Automation Testing Metrics

Each category contains a set of different testing metrics that are as presented in table 3.

Table 3. Taxonomy of testing phase metric



Manual	Performance	Automation	Common Metrics
Test Case Productivity	Performance Scripting Productivity	Automation Scripting Productivity	Effort Variance
Test Execution Summary	Performance Execution Summary	Automation Test Execution Productivity	Schedule Variance
Defect Acceptance	Performance Execution Data Client Side	Automation Coverage	Scope Change
Defect Rejection	Performance Execution Data Server Side	Cost Comparison	
Bad Fix defection	Performance Test Efficiency		
Test Efficiency	Performance Severity Index		
Defect Severity Index			
Test Execution Productivity			

a) Manual Testing:

Test Case Productivity (TCP)

TCP metric provides the test case crafting productivity consistent with the conclusive remark. TCP is also known as Test Case Design Productivity Metrics and calculated by the ratio of the total Number of test prepared to effort spent on test case preparation.

$$\mathrm{TCP} = \left[\frac{\text{Total No. of test steps}}{\text{Effort(hrs)}} * 100\right] \frac{\text{Steps}}{\text{hrs}}$$

or

$$\mathrm{TCP} = \left[\frac{\text{Total No.of test case design}}{\text{Effort(hrs) spend on TC preparation}} * 100\right] (4)$$

$$\text{Test Case/person day}$$

Example: Let efforts required for writing 191 steps is 9 hours. TCP=191/9=23.88

Test case productivity = 24 steps/hour

By comparing the test case productivity values with the preceding release(s) and extract the more appropriate results and recommendations from it. The give below fig. 2 shows a test case productivity pattern.

Table 4

Test Case Name	Raw Steps
ABC_1	32
ABC_2	35
ABC_3	41
ABC_4	37
ABC_5	46
Total Raw Steps	**191**

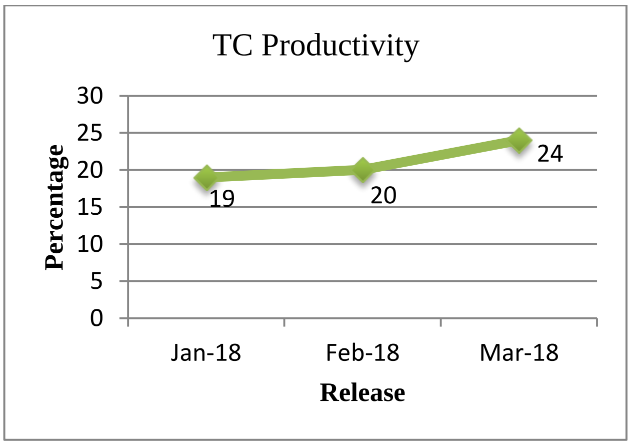


Figure 2. Test Case Productivity Pattern

Test Execution Summary (TES

This metric classifies the test cases on the behalf of project status together with a key fact explanation, if exist, for different test cases. It provides the statical viewpoint of the software release. Software tester may gather the data to determine the number if the test case implemented with under mention status:

- Success
- Fail with key points of failure
- Unable to test with cause such as time crunch, setup issue, out of scope etc.

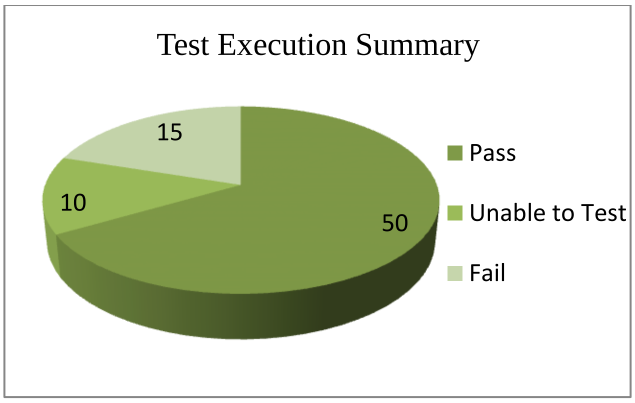


Figure 3. Test Execution Summary Pattern

It is possible to display the identical tendency for the assortment of causes of miscellaneous unable to test and fail cases also.

Defect Acceptance (DA)

DA metric use to find out the number of reasonable defects identified by the testing staff during execution. The values obtained by Defect Acceptance can be compared with precedent launching to obtain an improved picture about defects. Defect Acceptance can be referred as the ratio of the number of valid defects to total number of defects found during testing of software.

$$\mathrm{DA} = \left[\frac{\text{Number of Valid Defects}}{\text{Total Number of Defects}} * 100\right]\% \qquad (5)$$

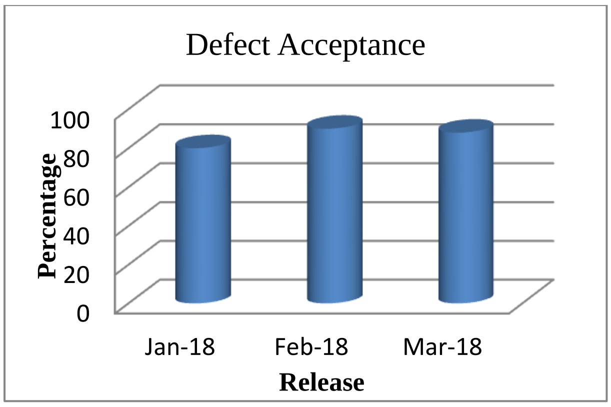


Figure 4. Defect Acceptance Pattern

Defect Rejection (DR)

Defect rejection metric used to obtain the number of defects refused during the execution of the program. This metrics offers the percentage of invalid defects that the analyzing team has deployed as well as able to control it whatever is essential. Defect rejection metric referred as the ratio of the number of defects rejected to the total number of defects encountered during testing.

$$\mathrm{DR} = \left[\frac{\text{Number of Defect rejected}}{\text{Total Number of Defects encountered}} * 100\right]\% \quad (6)$$

The values obtained from defect rejection metric could be correlated with prior issue for getting excellent results.

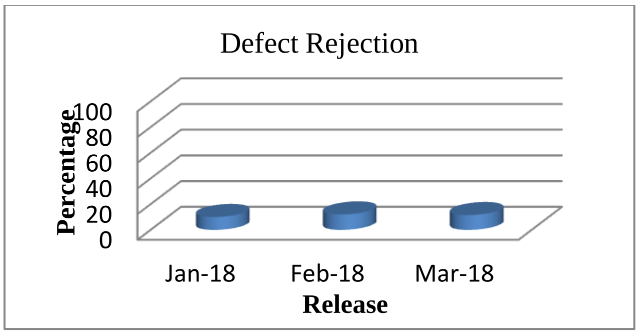


Figure 5. Defect Rejection Pattern

Bad Fix Defect (BD)

The defects that contribute to rising new defects are considered as bad fix defects. Such bugs in one module can give rise to other new defects in other modules.The effective software testing concentrates on removing such defects to ensure quality. Such metric establish the proficiency of the defect settlement practice and defined as

$$\mathrm{BD} = \left[\frac{\text{Number of bad fix defects}}{\text{Total number of Valid defects}} * 100\right]\% \quad (7)$$

This metric provides the percentage of bad defect settlement in the module which ought to be managed.

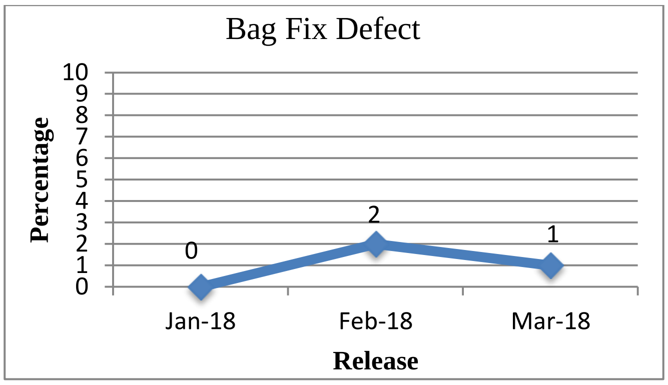


Figure 6. Bad Fix Defect Pattern

Test Execution Productivity (TEP)

Test Execution Productivity metric determines the average test execution productivity whose deeper examination can provide decisive results. This metric utilized for performance evolution and time estimation.

$$\mathrm{TEP} = \left[\frac{\text{Total No. of Test Case executed(TC)}}{\text{Execution Effort(hrs)}} * 8\right]\frac{\text{Exe}}{\text{day}} (8)$$

Where T can be calculated as

$$\mathrm{Tc} = \text{Base Test Case} + \{(\mathrm{T}(0.33) * 0.33) + (\mathrm{T}(0.66) * 0.66) + (\mathrm{T}(1) * 1)\}$$

Where, Base Test Case is the number of Tc at least once. T(1), T(0.66)and T(0.33) are the number of Tc retested with 71% to 100%, 41% to 70%, 1% to 40% of total Tc Steps respectively.

Example:

Table 5

Test Case Name	Run Base Effort (Hr)	Re-Run1 Status	Re-Run 1 Effort (Hr)	Re-Run 2 Status	Re-Run 2 Effort (Hr)	Re-Run 3 Status	Re-Run 3 Effort (Hr)
ABC_1	2.0	T(.66)	1.1	T(.66)	0.46	T(1)	2.0
ABC_2	1.4	T(.33)	0.3	T(1)	2.0		
ABC_3	2.4	T(1)	1.3				
ABC_4	2.0	T(1)	2.0				
ABC_5	2.16						

From the above example:

Table 6

Base Test Case	5
T(1)	4
T(.66)	2
T(.33)	1
Total Efforts (In Hour)	19.12

Te = 5 + ((1*0.33) + (2*0.66) + (4*1))) = 10.65

Therefore, Test Execution Productivity = (10.65/19.12)*8 = 4.46 Execution/day

By comparing the productivity with preceding release data can get a valuable conclusion and recommendations.

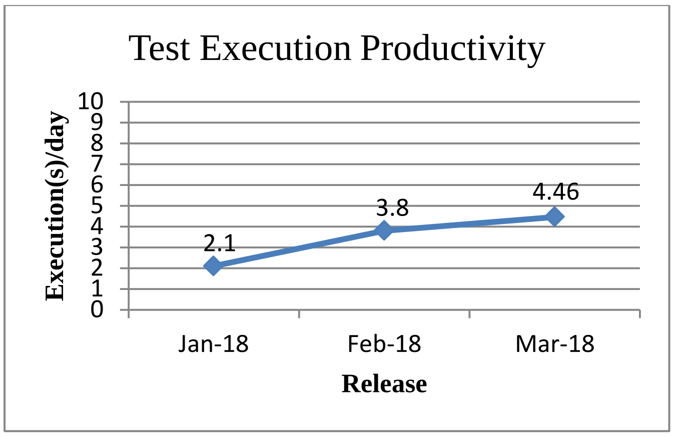


Figure 7. Test Execution Productivity Pattern

Test Efficiency (TE)

Test efficiency measures the efficiency of testing personnel working on identifying the faults. It additionally demonstrates the faults skipped through testing phase that moved to forward phase.

$$\mathrm{TE} = \left[\frac{\mathrm{di}}{\mathrm{di} + \mathrm{dp}} * 100\right]\% \qquad (9)$$

Where

di Indicates the Number of valid defects acknowledged while testing performed.

dp Indicates the Number of valid defects acknowledged by end users after post-testing.

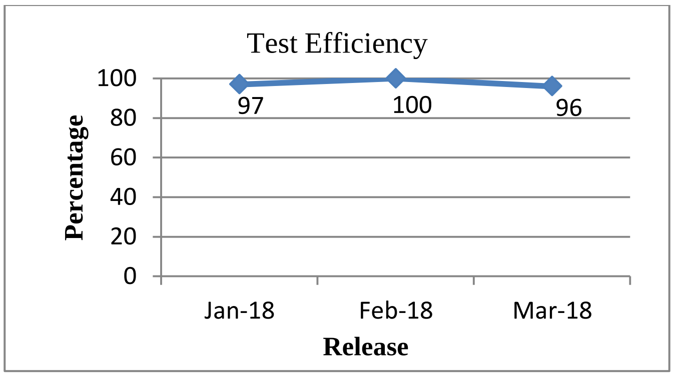


Figure 8. Test Efficiency Pattern

Defect Severity Index (DSI)

DSI measures the quality of products during the test as well as the release, depending on who take decision for project release. Defect severity suggests the amount of negative effect on the quality of software.

$$\mathrm{DSI} = \left[\frac{\sum(\text{Severity Index} * \text{Number of Valid defects})}{\text{Total Number of valid defects}} * 100\right]\% \qquad (10)$$

b) Performance Testing Metrics

Performance Scripting Productivity (PSP)

PSP metric measures the scripting productivity with regards to performance test script. The PSP metric evolution process used in scripting productivity incorporates logic encapsulate in the script that is rarely applied and have tendency for a long time.

$$\mathrm{PSP} = \left[\frac{\sum \text{Operations Peerformed}}{\text{Effort(hrs)}} * 100\right] \text{Operations}/\text{hrs} \qquad (11)$$

Where operation performed includes

- Number of hits on which data change
- Number of input parameters
- Number of correlation parameters

Example:

Table 7

Operation Performed	Total
No. of Clicks	9
No. of Input Parameter	5
No. of Correlation Parameter	5
Total Operation Performed	**21**

Let the effort took in scripting = 9 hours.

ASP calculated =21/9= 2.1 Operations/hour

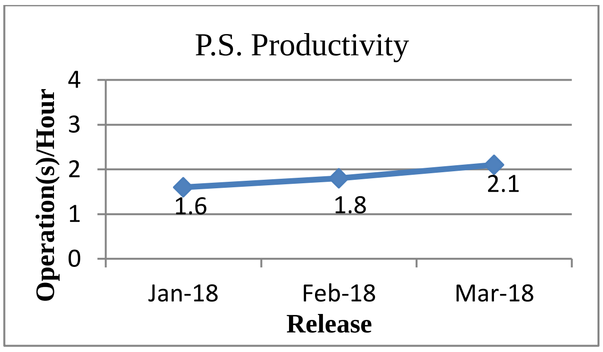


Figure 9. Performance Scripting Productivity Pattern

Performance Execution Summary (PES)

PES metric utilized for classification of performance testing on the basis of number of tests organized and carry out along with test status i.e. pass or fail. Some performance testing are enlisted as

- Soak/Endurance test
- Peak Test
- Failover test
- Stress/Breakpoint Test

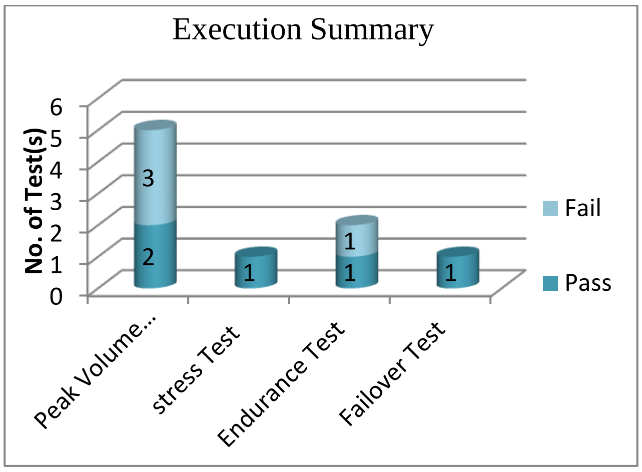


Figure 10. Performance Execution Pattern

Performance Execution Data - Client Side

Performance execution data-client side metric provides a extensive knowledge about the client side data for execution. Fewer data points of this measurement are enlisted as

- Response time
- Clicks/second
- Running Users
- Throughput
- Total Transaction/second
- Error/second

Performance Execution Data - Server Side:

Performance execution data-server side metric provides the extensive knowledge about the server side data for execution. Fewer data points of this measurement are enlisted as

- CPU utilization
- Memory Utilization
- Database connectivity per second

Performance Test Efficiency (PTE)

PTE metric used to measure the quality of performance testing personnel with respect to requirements meet that can be used as an input for further extensive action of improvising, if essentially required. Performance test efficiency metric use the following formula for measurement.

$$PTE = \left[\frac{\text{RM PT} - \text{RnMPT}}{\text{Requirements meet during PT}} * 100\right]\% \quad (12)$$

Where**RM PT:** Requirements meet during PT

RnMP:Requirements not meet after wrep up of PT

To analyze this, it is necessary to gather data point during and wrap up of performance testing. Some requirements of performance testing are enlisted as

- Transaction per second
- Average response time
- Server Stability
- Applications should be capable to handle predefined maximum load.

Let during the Performance testing, above discussed requirements found. Therefore, throughout the performance Testing requirement met = 4. In production, the average response time is higher than estimated so requirements not cover after completion of Testing = 1. Therefore, PT efficiency = {4/(4+1)}*100 = 80%

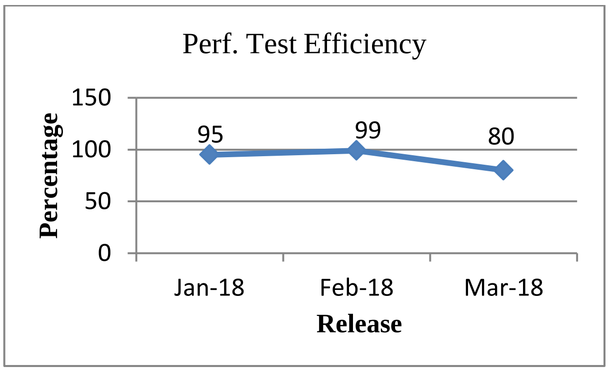


Figure 11 Performance Test Efficiency Pattern

Performance Severity Index (PSI)

This metric used to measure the quality of products accordance with performance benchmarks and the information obtained from PSI metric can be used for decision making to release the product to the next phase. It reports the quality of products against performance and measured by the formula

$$PSI = \left[\frac{\sum(\text{Severity Index} * \text{No. of requirements not meet for this severity})}{\text{Total No. of requirements not meet.}} * 100\right] \quad (13)$$

If requirement not fulfilled, one can possibly allot severity for the requirements with the intension that decision can be taken regarding product release on the behalf of performance.

Example:Assume, Average response time is essential requirement that has not covered, then the tester will be able to open defect with severity quite as critical. Therefore, performance severity index = (4*1/1) = 4

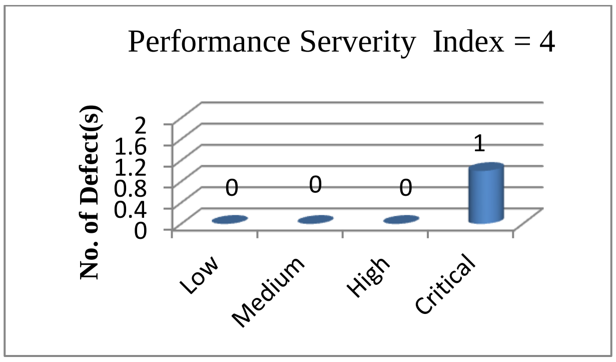


Figure 12. Performance Severity Index Pattern

c) Automation Testing Metrics

Automation Scripting Productivity (ASP)

In such testing, testing staff generates the test script employing the automation tools to examine the correctness of software. ASP metric employed for assessment of progress of automation testing [Keele (2007)].

$$ASP = \left[\frac{\sum \text{operration performed}}{\text{Efforts(hrs)}}\right] \text{Operations} / \text{hours} \quad (14)$$

Where, Operation performed includes

- Number of hits on which data are refreshed
- Number of input parameters
- Number of Checkpoint added

All above operations does incorporate logic embedded into the script that often occasionally used.

Example:

Table 8

Operation Performed	Total
No. of Clicks	11
No. of Input Parameter	5
No. of Checkpoint Added	10
Total Operation Performed	**26**

Assume effort took in scripting = 9 hours. Therefore, ASP=26/9=2.9

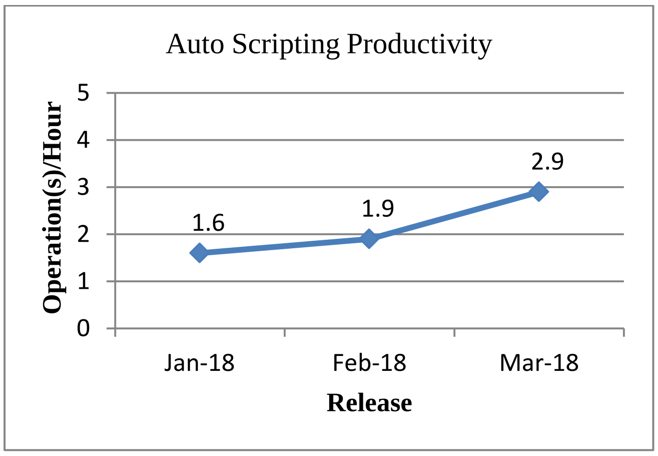


Figure 13. Automation Scripting Productivity, Pattern

Automation Test Execution Productivity (ATEP)

This metric measures the productivity of automated test case execution.

$$\text{ATEP} = \left[\frac{\text{Total number of Automated test case executed}}{\text{Execution efforts(hrs)}} * 8\right] \text{executions/day} \qquad (15)$$

Where Test case executions can be estimated as

$$\text{Tc} = \text{Base Test Case} + \{(\text{T}(0.33) * 0.33) + (\text{T}(0.66) * 0.66) + (\text{T}(1) * 1)\}$$

Where, Base Test Case is the number of Tc at least once. T(1), T(0.66)and T(0.33) are the number of Tc retested with 71% to 100%, 41% to 70%, 1% to 40% of total Test case Steps respectively. Evaluation process involved in the ATEP is identical to manual test execution productivity.

Automation Coverage (AC)

Test coverage indicates the volume of test carried out by testing tools or the volume of manual process converted into automation. Actually, this metric employed to identify how much automation ensured by deploying automation tools [Gulechha (2017)].

$$\text{AC} = \left[\frac{\text{No. of test case automated}}{\text{Total no of manual test cases}} * 100\right]\% \qquad (16)$$

Example: Consider, we have 100 Manual test cases and the other has automated 70 test cases then in this case Automation Coverage = 70%.

Cost Comparison

The test automation continues to be projected as a strategy to reduce the testing expense [28]. Automation tools offer the services to run the repetitive test case to boost the testing activity, however this requires appropriate guidance and training to run test by a specific testing tool. The cost comparison can be done by separating the manual and automated testing cost that is given below

$$\text{Cost (Manual)} = \text{Execution Efforts (In hours)} * \text{Billing Rate}$$

$$\text{Cost (Auto)} = \text{Tool buying cost (One Time)} + \text{Maintenance Cost} + \text{Script Development Cost} + \{\text{Execution Efforts (In hours)} * \text{Billing Rate}\} \qquad (17)$$

Cost comparison metric is used for the analysis of ROI (return on investment). In a case where script reused, development cost is actually the script update cost. This metric offer an excellent result with regards to cost that contribution an imperative role in software industries.

d) Common Testing Metrics

Almost, every software developing organization specially gives extra effort on testing phase of software life cycle to ensure the quality of product to achieve users' satisfaction. A number of testing metrics have been developed in the last three decades; some of them are disuses in the above section under manual testing, performance testing and automation testing. Otherthan these some other common testing metrics also have been proposed that used for all types of testing of software are discussed as

Effort Variance (EV) metric

Effort Variance measures the variance in the estimated effort required in the testing phase of the products. This metric tries to fill up the gap between actual and estimate effort required. Determined by the formula

$$\text{EV} = \left[\frac{\text{Actual effort} - \text{Estimated effort}}{\text{Estimated effort}} * 100\right]\% \quad (18)$$

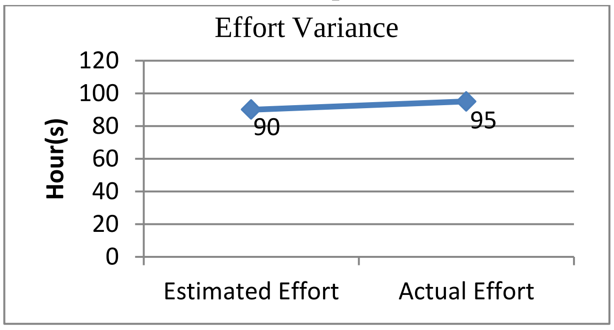


Figure 14. Effort Variance Pattern

Schedule Variance (SV) Metric

The schedule is a one of most important constraint of the software life cycle. Schedule variance determines whether project is running accordance with schedule or not.

$$\text{SV} = \left[\frac{\text{Number of acutal days} - \text{No. of Estimated days}}{\text{No. of estimated days}} * 100\right]\% \quad (18)$$

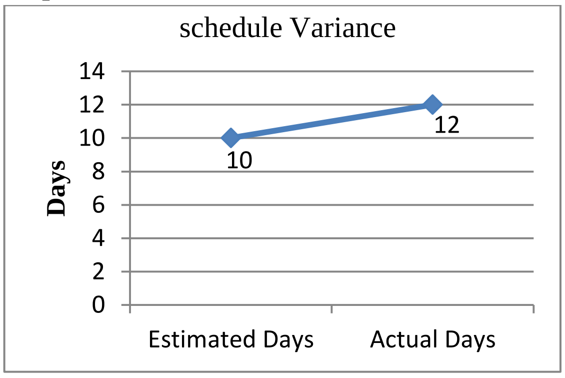


Figure 15. Schedule Variance Pattern

Scope Change (SC) Metric

Scope change metric measures the stability scope of testing and it may increase or decrease depending on circumstances.

$$\text{Scope Change} = \left[\frac{\text{Total Scope} - \text{previous scope}}{\text{Previous Scope}} * 100\right]\% \quad (19)$$

Where,

Total Scope = Previous Scope + New Scope, if Scope increases.

Total Scope = Previous Scope - New Scope, if Scope decreases.

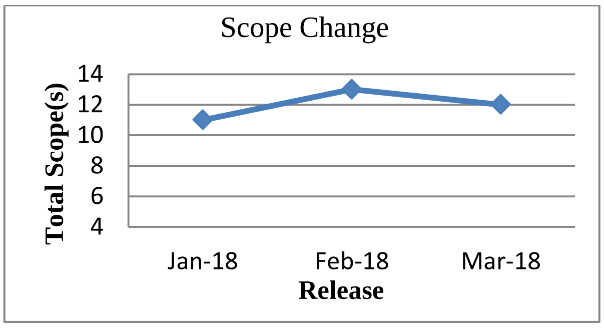


Figure 16. Scope Change Pattern

VI. CONCLUSION

Employing software metrics are an excellent exercise that could deliver a wealth of benefit to a project. However, it needs time, work as well as capital by taking advantage of metrics in a frequent manner, project personnel realize positive change and improvement in the development process and in the software product. Software metrics perform a decisive contribution in managing the complexities to reduce the manpower to develop, maintain the software and improve the efficiency of testing along with quality. In this paper, we have tried to describe the various existing software metrics with suitable example used in different stage of the application development life-cycle. Our work contributes to the field of software engineering and maintenance by demonstrating that software metrics assist the software developers and managers to effectively manage the software development activities economically and timely. Hopefully this work will influence the personnel from both academia and industries to understand, devise and design a new software metric that could implement in distinct phase of the application development life-cycle to achieve the required goal effectively.

VII. REFERENCES

[1]. Lyu, M.R. (Revised 2005). Handbook of Software Reliability Engineering. IEEE Computer Society Press, California.

[2]. Fenton, N.; Bieman, J. (2014).Software Metrics a Rigorous and Practical Approach.3rd edition, CRC press.

[3]. Lima, P., Guerra, E., Meirelles, P., Kanashiro, L., Silva, H., &Silveira, F. F. (2018). A Metrics Suite for code annotation assessment. Journal of Systems and Software, vol. 137,pp. 163-183.

[4]. Padhy, Neelamadhab, Satapathy, S., and Singh, R.P., (2018).State-of-the-art object-oriented metrics and its reusability: a decade review.In Smart Computing and Informatics, pp. 431-441. Springer

[5]. Scotto, M.; Sillitti, A., Succi, G., &Vernazza, T. (2004, March). A relational approach to software metrics. In Proceedings of the 2004 ACM symposium on Applied computing. pp. 1536-1540.

[6]. Gomez, O., Oktaba, H., Piattini, M., &García, F. (2006, September). A systematic review measurement in software engineering: State-of-the-art in measures. In International Conference on Software and Data Technologies (pp. 165-176).Springer, Berlin, Heidelberg.

[7]. Unterkalmsteiner, M.; Gorschek, T.; Islam, A. K. M. M.; Cheng, C.K.; Permadi, R. B.; Feldt, R. (2012). A Systematic review measurement in Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review, IEEE Transactions on Software, 38(2), 10.1109/TSE.2011.26.

[8]. Kitchenham, B. (2010). What's up with software metrics?-A preliminary mapping study. Journal of systems and software, 83(1), 37-51.

[9]. Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., & Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software, 80(4), 571-583.

[10]. Siddiqui, T., & Ahmad, A., (2018) . "Data Mining Tools and Techniques for Mining Software Repositories: A Systematic Review". In: Aggarwal V., Bhatnagar V., Mishra D. (eds) Big Data Analytics. Advances in Intelligent Systems and Computing, vol 654. Pp. 717-726 Springer, Singapore.

[11]. Siddiqui, T., Ahmad , A., (2018) Complexity Clarification through Code Metrics. Proceedings of the 12th INDIACom, pp. 3746-3749.

[12]. Keele, S. (2007).Guidelines for performing systematic literature reviews in software engineering. In Technical report, Ver. 2.3 EBSE Technical Report. EBSE.sn.

[13]. Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering-a systematic literature review. Information and software technology, 51(1), 7-15.

[14]. Unterkalmsteiner, M., Gorschek, T., Islam, A. M., Cheng, C. K., Permadi, R. B., &Feldt, R. (2012 b). Evaluation and measurement of software process improvement—a systematic literature review. IEEE Transactions on Software Engineering, 38(2), 398-424.

[15]. Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements engineering: A roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 35-46).

[16]. Davis, A. M. (1993). Software requirements: objects, functions, and states. Prentice-Hall, Inc.

[17]. Zave, P., & Jackson, M. (1997).Four dark corners of requirements engineering.ACM

transactions on Software Engineering and Methodology (TOSEM), 6(1), 1-30.

[18]. Bokhari, M. U., & Siddiqui, S. T. (2011, March). Metrics for Requirements Engineering and Automated Requirements Tools. In Proceedings of the 5th National Conference.

[19]. Soren Petersen, Design Metrics -- Is It Possible to Optimize Design for Innovation?, available at http://www.huffingtonpost.in/entry/design-metrics-is-it-poss_b_1411593.

[20]. Wikipedia of Requirements analysis http://en.wikipedia.org/wiki/Requirements_analysis.Retrived 2018.

[21]. Ralph, P., & Wand, Y. (2009).A proposal for a formal definition of the design concept. Design requirements engineering: A ten-year perspective, 14, 103-136.

[22]. Freeman, P., & Hart, D. (2004).A science of design for software-intensive systems. Communications of the ACM, 47(8), 19-21.

[23]. Booch, G.; (2004). Object-Oriented Analysis and Design with Applications (3rd Ed.). MA, USA: Addison Wesley. ISBN 0-201-89551-X. Retrieved 30 January 2015.

[24]. Kafura, D., & Reddy, G. R. (1987).The use of software complexity metrics in software maintenance. IEEE Transactions on Software Engineering, (3), 335-343.

[25]. Omarsson, O.T. (2017). Assessment Methods When Choosing a Software Architecture Alternative Software Maintainability Prediction. University of Gothenburg Gothenburg, Sweden. https://quandarypeak.com/2015/02/measuring-software-maintainability/.

[26]. Gulechha, L.; Software Testing Metrics, available at: https://www.stickyminds.com/article/white-paper-software-testing-metrics.

[27]. Ahamed S.S. (2009): Studying The Feasibility and Importance of Software Testing: An Analysis. International Journal of Engineering Science and Technology, 1(3), pp.119-128.

[28]. Catal, C., &Diri, B. (2009). A systematic review of software fault prediction studies. Expert systems with applications, 36(4), 7346

[29]. Cyclomatic complexity available at https://en.wikipedia.org/wiki/Cyclomatic_complexity.

[30]. Ebert, C., Dumke, R., Bundschuh, M., &Schmietendorf, A. (2005). Best Practices in Software Measurement: How to use metrics to improve project and process performance. Springer Science & Business Media.

[31]. http://www.informit.com/articles/article.aspx?p=30306&seqNum=3.

[32]. https://en.wikipedia.org/wiki/Halstead_complexity_measures retrived october,2016.

[33]. Jayanthi,B.; Kumari k. (2014): Brief study on Software quality metrics and software complexity metrics in Web application. International Journal of Engineering Sciences & Research Technology, 3(12) pp. 441-444.

[34]. Kafura, D., & Reddy, G. R. (1987).The use of software complexity metrics in software maintenance. IEEE Transactions on Software Engineering, (3), 335-343.

[35]. Khan,A. A.; Mahmood, A.; Amralla, S.M.; Mirza, T.H. (2016): Comparison of Software Complexity Metrics. International Journal of Computing and Network Technology, 4(1), pp. 19-26.

[36]. Kushwaha, D. S., &Misra, A. K. (2006). Improved cognitive information complexity measure: a metric that establishes program comprehension effort. ACM SIGSOFT Software Engineering Notes, 31(5), pp. 1-7.

[37]. Misra, S. (2006).A complexity measure based on cognitive weights.International Journal of Theoretical and Applied Computer Sciences, 1(1). Pp. 1-10.

[38]. Moore, J. W. (1998). Software engineering standards. John Wiley & Sons, Inc..

[39]. Nirpal, P. B., & Kale, K. V. (2011).A brief overview of software testing metrics.

International Journal on Computer Science and Engineering (IJCSE), 3(1), 204-211.

[40]. Shao, J., & Wang, Y. (2003). A new measure of software complexity based on cognitive weights. Canadian Journal of Electrical and Computer Engineering, 28(2), 69-74.

[41]. Siddiqui, T., Wani, M. A., & Khan, N. A. (2011).Efficiency Metrics. BVICAM's International Journal of Information Technology, 3(2), pp. 61-65.

[42]. Artemev, Vasilii, Vladimir Ivanov, Manuel Mazzara, Alan Rogers, Alberto Sillitti, Giancarlo Succi, and Eugene Zouev(2017). "An architecture for non-invasive software measurement." In International Andrei Ershov Memorial Conference on Perspectives of System Informatics, pp. 1-11. Springer, Cham,.

[43]. O'Regan, Gerard.(2017). "Software Metrics and Problem-Solving." In Concise Guide to Software Engineering, pp. 139-170. Springer,

[44]. 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, pp.106-121.

[45]. Toure, F., Badri, M., &Lamontagne, L. (2018).Predicting different levels of the unit testing effort of classes using source code metrics: a multiple case study on open-source software. Innovations in Systems and Software Engineering, Vol. 14(1), pp. 15-46.

[46]. Gasparic, M., & Janes, A. (2016). What recommendation systems for software engineering recommend: A systematic literature review. Journal of Systems and Software, 113, 101-113.