



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

(Open Access, Double Blind Peer-reviewed, Refereed and Indexed Journal)

www.iasir.net

An Empirical Study of Software Metrics and Measurement in Software Development Life Cycle: Extensive Study

AnaghaVaidya and Prafull Bafana

Symbiosis Institute of Computer Studies and Research (SICSR), Gokhale Cross Road,
Model Colony, Pune, Maharashtra 411016, INDIA

Abstract: The matrix is one of the important tools used for the measurement of software quality. They are an indicator for the organization to obtain information for continuous improvement in its productivity, reduced errors and improve acceptance processes. As the growing importance of measurement, many matrices are proposed in software literature. Some of these matrices are provided overlapping and redundant information. It is very difficult to understand which matrices are more useful in which area. This paper highlights the significance of different matrices in software development phases from requirement engineering till maintenance. The paper studies different measurement attributes in SDLC phases and how the different matrices are used for this measurement. The study also reflects how existing matrices leads towards the continuous process and product quality improvement.

Keywords: SDLC, STLC, RTM, OOAD, SSAD

I. Introduction

Development of any software product involved the tremendous amount of cost and efforts. Therefore, a systematic measurement technique is needed for controlling the cost, efforts, and quality of the software product. Measurement is nothing but provides constitute values which suggest the trend of the objective. The measurement of the software quality product is difficult because it is an intangible product. The software quality measurement is subjective, it varies from novice to expert user. Therefore, a standard technique is needed for it, software metrics used for this purpose. It provides the quantitative measurement and provides the rationale for selecting the best alternatives among the process; it saves the development efforts and money and provides the quality product to end user [37].

Metrics precisely describe the status of software project processes and products. It always makes complex projects easy to understand, can identify potential problem areas of the product early and It describes the project and product status, clearly, make easier to manage workload and recourses and maintain the process, this reduces the ambiguity and constrained software projects. These measurements identify the problems, strong and weak points of the product as well as developing processes. [18]

Three types of metrics are defined in the theory: product metrics and process and project metrics [13]. The objective of project metrics is to set and meet the achievable cost, schedule, quality and software functions of the product. The aim of the process measurement is to ensure the process is performing as per the expectation, it has achieved the business objective and the process is deserved for the respective product. The product metrics are used for customer satisfaction of the product. These measurements are measured either objectively or it may measure in terms of subjectively. e.g. product measurement in terms of a number of lines is the objective measurement while measurement of programmer's experience is subjective measurement. These measurements are primitive and it needs to do the calculation of the different attributes.

The objective of this paper is to identify the different measurement parameters used in SDLC and how these parameters are measured. Hence, the objective of this paper is

1. Recognize the current usage of software metrics for software development projects.
2. Discover the importance of software metrics for software development project.
3. Study the advantages and disadvantages of the respective metrics.

The present paper is organized as follows section –II literature review, Section -III explain the methodology used for the study. In section-IV illustrates difference metrics used in of SDLC Section V conclusion and future scope.

Section II: Literature Review

The quality is an uncompromised factor in software, development. Defining the quality for the software is very difficult because it is an intangible product. In literature five different perspectives are available for the software quality, these are: User, manufacturer, product etc. To achieve the software quality, the different SDLC models are designed and with respective these different measurement techniques are also introduced in literature. Although it is a systematic process, but still the crises are arrives. These crises can avoid by applying proper measurement [20]. These measurements are applicable to project, processes and people in every SDLC phases.

E.g. Software deliverable are checked for quality at every phase of the software development to ensure the quality of the end product [47].

In SDLC, the first phase is requirement engineering. Quality process in requirement engineering confirms the degree of change of requirement [47] and requirement tractability links the high level requirement to lower level requirement. The end product of requirement is SRS, the quality SRS needs to document [57]. These quality attributes are understood ability, consistency, correctness and clarity. These attributes not only manages the requirements, but also has an impact on a complete project developing [46].

Once these requirements are clear then it must properly translate to the next phase i.e. on development phase. The software is developed either SSAD approach or by OOAD approach. In the SSAD approach more emphasis on the flow of information from one module to another module, hence different information flow metrics are introduced [40] [42] In OOAD approach follow the principle of Object Orientation like encapsulation, abstraction, etc. OOAD metric measures these attributes [38] [39] [2].

The code complexity and size are the two parameters for the measurement of software coding. Code complexity is measured by the number of loops, dead code, etc. The cyclamate complexity is used for calculating the complexity of the metrics. Three types of metrics Primitive measurement, abstract software metrics and structural metrics are proposed for it [56] The Primitive metrics [56] are Halstead's Metric [58], Albrecht A.J Function Points Metric[59], Abstract metrics are: Knot's Metric, McCabe's Metric [60], Structured Metrics: Haney's Stability Metric Myer's Metric [56].

Different metrics are introduced for the measurement of the testing process as well as for testing quality test cases. The software testing is measured with reference to cost, schedule, development of quality test cases, number of resources and for the testing, etc. [54]. Software metrics measure the size, data and control flow complexity. On this complexity is used in software testing, which calculate efforts of testing, less complexity provides good testing efficiency.

Writ after delivering the product to the customer the product is in the maintenance phase. If the programmer understands the requirement more clearly than the software development is perfect hence the result led to the product is in less maintenance phase. This is achieved, if the product is properly measured in each SDLC phases. It prevents failure before entering the project into maintenance steps [55]. In SDLC some of the measurements are direct and some of them are indirect measurement e.g. number of line of code is direct measurement but code complexity , the performance of code , security are indirect measurement of it. Both the measurement are important for considering development good software quality factors.[52]

III Methodology

In this paper, we perform the systematic review of the literature with reference to the metrics and measurement. We perform the study with guidelines of [16] [17] and following research questions are addressed by this paper

Table 1: Research Question Address

	Questions	Aim
RQ1	What are the different measurement parameters are used in the software product development	Identify the different measurement parameters for different software development phases
RQ2	How to quantify this measurement and what are the standard accepted values of this measurement?	The aim of this is how the quantification is done for the measurement parameters
RQ3	How these parameters are bounded into metrics?	Identification of different metrics used in SCLD
RQ4	How the software quality factors are measured in software product development?	Since the software is intangible ,so it is difficult to measure the quality of the software. The aim of this research question is to identify the quality parameters of the software and how to measure

In this paper, we considered the software development in traditional as well as the Object-Oriented development methodology. For the collection of data, we follow GQM [19] method. For each SDCL phases, we formulate the goal, with reference to that goal formulate the different questions and defined different parameters for measurement. For the quality measurement, Mc-call quality factors [36] are considered. These parameters are:-

Table 2: Software Quality Parameters [36]

	Quality Parameters	Question	What to measure
1	Correctness	The software does what the user wants	Traceability, Completeness, Consistency
2	Reliability	Software work accurately all time	Consistency, Accuracy, Error tolerance
3	Efficiency	Run on the machine as per the specification	Execution Efficiency
4	Integrity	Security of software	Access Control, Access audit
5	Usability	The Software is easy o use , operate and understand	Operability, Training, communicativeness
6	Maintainability	Can the software stable, well?	Simplicity, conciseness, Self-Descriptiveness, Modularity
7	Testability	Can the software test easily?	Instrumentation, self-descriptiveness, Modularity

8	Flexibility	Can easily software changed?	Self-Descriptiveness, Generality, Expandability
9	Portability	Can the software operate it on another machine	Self-Descriptiveness, Software System independence, Machine Independency
10	Reusability	Software is written well, so it can be reused by the system easily	Self-Descriptiveness, Generality, Modularity , Software System independence, Machine Independency
11	Interoperability	Is the software interface with another machine	Modularity, communication, common ability, data commonality

IV Metrics for SDLC Phases

Requirement Engineering is very vital and decisive phase in the development of any software product. It defines the system scope and objective of the system. Hence each requirement must be clear and complete and clear. Therefore correctness and completeness are the measuring criteria. to measure the completeness in requirement engineering. [46]. All these requirements are documented in SRS and use case specification document. Thus, the SRS, each requirement are measurable objects.

Table 3: GQM for Software Requirement Phase

Goal	- Problem Specification - All collected Requirements are correct and complete, manageable and transferable from one phase into another phase
Questions	How to measure the stability of Requirements? How to measure the Requirement changes are manageable? How to measure the Requirements are specified correctly? How to assure all requirements are collected are complete? How to measure all requirements are properly conveyed to next phases?
Measurement Parameters	Requirement stability , requirement manageability , requirement correctness and completeness , Requirement change request frequency
Quality Measurement	Correctness, Flexibility , maintainability, testability

The Following metrics are identified in the requirement phases:

Table 4: Requirement Metrics

Sr No	Metrics	Purpose	Metrics suit
1	Requirement Traceability[3]	This is an association between the requirements of the system. It addresses the dependencies between requirements, i.e. Traceability from highest to lowest level requirement specification. It identifies inconsistency in the requirements.	
2	Requirement Completeness [4]	The requirement is completeness are measured . If all pages are numbered, all references are present. . Requirement specification level, i.e. the requirement is specified, in which hierarchical level. So it can effectively use in the next level. . Requirement complexity . Following are additional metrics are introduced for the measurement :- Requirement decomposition [4] Requirement Specification Development Metrics [4] Requirement Specification Metrics[4]	
3	Requirement Stability Metrics [3]	This metric is used to control, organize and track the changes to the original requirement . This track changes to the functional requirements and the cumulative changes requested over time against the number of requests that have been resolved .	
Requirement Quality measurement Parameter :-			
4	Lack of ambiguity in requirement [46]	Requirements that are uniquely identified by all reviewers. If the ration is closer to 1 then lower the ambiguity.	
5	Understandable [45][46]	Number of requirements that are understood by all reviewers. If the ratio is closer to 0 then the requirements are not understood clearly . If it is close to 1 then all requirements are understood	
6	Uniqueness[45][46]	Requirements that have been uniquely explained by all reviewers. The result of this metrics is less than 95%.	
7	Correctness[45] [46]	All requirements that have been correctly validated. If result >80%, then requirements are accepted.	
8	Complete[45] [46]	To measure the total number of functions correctly specified. If it is closer to 1, then it is more complete.	
9	Change requirements[45] [47]	Metrics counts the requirements changed during each phase of system development. The result describes the percentage of the total requirements to be changed. The ratio must be less than 1 means the change requirements are not accepted.	
10	Misinterpreted Requirements[45] [47]	Metrics calculate the requirements that have been misinterpreted during the elicitation phase If result > 5 %, then the elicitation process needs to redo for confirmation	

11	Modifiable [45] [47]	Metrics counts the number of requirements that are required to be changed. If the ratio is closer to 1 means the requirement modification is high and if it is close to 0 means requirement modification is low.
12	Verifiable[45] [47]	Metrics measure verifiability of requirement. This metrics calculate the cost of requirement and time required for verification of this requirement . If the result is 0 means very poor requirements and 1 mean very good requirement.
13	SRS Quality[47]	Check the correctness in the requirement. This metrics is finding total number of requirement in SRS , errors found in SRS, error deleted from SRS, error modified in SRS.

Analysis:

The project organization and management is considered under analysis phase. Most of the project fails due to unrealistic deadlines, underestimation of efforts, predictable and unpredictable risk, and miscommunication between the project staff. To overcome this difficult different project planning, technology are defined like- PERT/CPM, Gantt Charts are used. The size of the project, the number of potential users, stability of requirement, and maturity of application technology, criticality of objective or problem statement, application durability needs to be considered for measurement of project success. This following GQM matrix is designed

Table 5: GQM metrics for Analysis

Goal	Identification of best suitable model for project development. Defining a scope of the project and project team analyzes. Planning of project Scheduling and resource utilization
Questions	How to identify the best software development life cycle model for the project? How to identify how many project resources are needed for the project development? How to identify project is running out of schedule and out of cost?
Measurement Parameters	Resource measurement , schedule measurement , project cost estimation

The different project metrics are introduced for the accessing status of the ongoing project, schedule, cost, project profitability and project risk. The project scheduling is performed on size of the project. In LOC [34] and FP [32] are two traditional metrics are used for size measurements. For LOC measurement, two types of metrics are introduced: Direct estimation and Derived estimation [33]. The direct estimation measurement is expert judgment while Derived estimation is by statistical methods for calculating . In OOAD different metrics are introduced for size measurement, e.g. Use case Points method [35], Predictive object point [31], Class Point method [30], Pattern point measurement [28] etc. Some static and dynamic pages are measure in Web application [29] size measurement. The project measurement set of matrices:-

Table 6: Analyses Metrics

Sr No	Metrics	Purpose
Earn Value Approach: - It is integrated metrics which calculate project scope, cost of the project and schedule measurement. It also measures, the project performance and project progress.		
1.1	The actual cost of work performed (ASWP)	It measures project performance and project progress
1.2	Software Project Index	Measure schedule performance of the project . SPI close to 1.0 indicates the efficient execution of the project schedule otherwise Negative means behind schedule .
1.3	Schedule variance (SV)	Indication of variance from the planned schedule
1.4	Cost Variances(CV)	It is an indication of cost savings (against planned costs) or shortfall at a particular stage of a project Positive means under budget Negative means over the budget
1.5	A cost performance index (CPI)	Close to 1.0 means project is within budget
1.6	Percent scheduled for completion (PSFC)	Indicator of the percentage of work that should have been completed within time 't'
1.7	Percent complete (PC)	Provides a quantitative indication of the percent of the work that has been completed at a given point in time t
1.8	Productivity : Resource Utilization	This metric measure project profitability
SIZE measurement		
2.1	Lines of Code [34].	Measurement of productivity and evaluation of programs.
2.2	Function Points [32]	Total number of functions delivered by software
2.3	Use case Points method [35]	Number of use cases and actors who are accessing these use cases. Use cases are also used for size measurement [4] -it is calculated by the number of actors weighted by their complexity
2.4	Predictive object point [31]	The size of the software is measured with reference to inheritance, number of methods per class

2.5	Class Point method [30]	The total number of problem domain classes, human interaction classes, data management classes'. Number of services provided by the class and a number of attributes provide these services.
2.6	Pattern point measurement [28].	Identify the number of design pattern [56] used. These patterns are identified the degree of structural complexity and difficulty.
2.7	Web Application	Number of static pages, Number of dynamic Pages, Number of Internal link, Number of persistent object, Number of external System interface, Number of static content object, Number of dynamic content, Number of executable functions
2.8	Weighted Micro Function Points (WMFP) [51]	Measured Parameters are :- Flow complexity (FC) – Measures the complexity of a programs' flow control path. Object vocabulary (OV) – Measures the quantity of unique information contained in the programs' source code, Object conjuration (OC) – Measures the quantity of usage done by information contained in the programs' source code. Arithmetic intricacy (AI) – Measures the complexity of arithmetic calculations across the program Data transfer (DT) – Measures the manipulation of data structures in the program Code structure (CS) – Measures the amount of effort spent on the program structure such as separating code into classes and functions Inline data (ID) – Measures the amount of effort spent on the embedding hard coded data Comments (CM) – Measures the amount of effort spent on writing program comments
Project Estimation model		
1	PERT/CPM	Identification of different factor which are affecting the project planning, Forecasting the impact of schedule change, improve the communication between team members
2	COCOMO	Effort estimation which calculates the number of people required for developed project Effort Applied (E) = ab(KLOC) ^{bb} [man-months] Development Time (D) = cb(Effort Applied) ^{db} [months] People required (P) = Effort Applied / Development Time [count]
3	Putnam model [50]	Software Effort estimation $\frac{B^{1/3} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} \cdot \text{Time}$ Where Size :- product size, B- scaling factor, productivity – process effectiveness , Effort :- efforts applied per person per year on the project, time :-project schedule time
4	Project effort and duration calculation [50]	$K = D^{0.4} * (\frac{S_e}{C_{te}})^E$ Where: - Se project size, quite effective technology, D – staffing complexity- entropy
5	Project duration calculation [50]	$t_d = D^{-0.2} * (\frac{S_e}{C_{te}})^{0.4}$ Where :-Se project size, Cte effective technology D – staffing complexity
6	Planning poker[49],	It is cost estimation method used in Agile development Group members make the estimation by playing cards of individual estimation of the project. The people have higher or lower value had a discussion. This process is continuing till the consensus is reached.
7	Wideband Delphi Process	Used for project estimation

Software Design

The design specification is the outcome of this phase. Design specification includes Software architecture, database design as well as test case design. Traditional architecture design is more focused on programming and data is global, reusability concept is not considered more extensively, OOAD overcome this drawback. Therefore, complexity, module organization, coupling, and cohesion measurement are varying for both these approaches. Hence, the different set of metric sets is used for these measurements.

Table 7: GQM: Design

Goal	- Best architecture of the product - Identification of inputs and outputs of the system - Database designing and definition of test strategy
Questions	How to define the architecture of the product? How to identify the averaging inputs and outputs of the product? What are the different strategies should be used for testing? How many test cases are needed for testing the product (average test cases)? How to identify the quality test cases are designed for testing the software How to measure the database for the system?
Measurement Parameters	Architecture measurement is nothing but measurement of Coupling , Cohesion , database measurement – nothing but the data inflow and data outflow , test case measurement
Quality parameters	Correctness, Reliability, efficiency, Integrity, usability , reusability

Metrics for Structural Design: - In the structural analysis more emphasis on inflow and outflow information and how these flows carry the different data information from it. Coupling and Cohesion measurement is on inflow and outflow information and the data which they carry across these different modules. The complexity of the module is also depending upon the data, information the module is heavily loaded if they operate it on data and it's less loaded if it is more focused on logic

Table 8 : SSAD Metrics: Software Development

Structural Design		
Sr No	Metrics	Purpose
1	Network Metrics [40]	Measure the complexity in architecture design. It captures the structural chart and measure the complexity between the chart. This is calculated by :- Graph impurity = $n - e - 1$ Where n - number of nodes in the structure chart and e- number of edges in the structure chart If it is 1 then the graph impurity is increased, i.e. complexity of the module is increased and coupling of the metrics is also increased
2	Information flow metrics [42]	This metrics measure the data is moving from and through the module. It is calculated as :- $Dc = size * (inflow * outflow)$ Inflow: total information flow in the module Outflow: total information flowing out of the module If Dc is high the module interaction is high and the module is difficult to test
3	Information flow metrics [62]	The module complexity measurement . It is calculated by following formula:- $Dc = fan\ in * fan\ out + inflow * outflow$ The module complexity not only depends upon the information flowing in the module but also on the number of modules to or from which data is flowing
4	Architecture Design Metrics – structural complexity [63]	Architecture complexity measures the total number of information moving from the system and toward the system. $S(j) = F(out) (j)$ where $f(out) = fan-out$ of module 'j'
5	Data Complexity[63]	$D(j) = V(j) / [f\ out (j) + 1]$ where V(j) number of input and output variable passed to and from the module 'j'
6	System Complexity[63]	$C(j) = S(j) + D(j)$ where S(j) – System Complexity and D(j) is data complexity
7	Measuring call- return architecture [62]	Size = $n * a$ where n= number of nodes and a= number of arcs
8	Data [63]	These are calculated from : Number of components in program Number of components whose correct function is determined by the source of input data Number of component whose correct input depends upon the previous processing Number of database items Components having single entry and exit points Program structure, module independently, Module not depend upon prior processing, Database size and module entrance and exit points
9	Coupling metrics- Data and control Flow coupling[63]	Measurement of input data parameters, total number of input control parameters, output flow parameters and output control parameters
10	Coupling metrics- Global coupling	Number of global variables used as data Number of global variable used as control
11	Environmental coupling	Number of modules called and number of the module calling the module under consideration

OOAD Metrics:

Extensive literature is found in OO-metrics. The coupling and cohesion are measured on inheritance, class responsibility, class method complexity, etc. the metrics are listed below:

Table 9: OOAD metrics for Software Development

Sr No	Metrics Name	Measurement
Chadamber and Kemerer Metrics Suit[38]		
1	Weighted methods per class	Measure the complexity of the class Indicator of the amount of effort required for maintenance and testing of the class Larger the number of method the complexity of the class is high but low maintainability A larger number of classes, then it is more on an application specific and limit the possibility of reused
2	Depth of Inheritance Tree	Determine the maximum length of the root node till leaf node of a given tree A number of levels are increased it increases the design complexity, but reusability is high, but class behavior prediction is difficult High DPT lower maintainability
3	Number of Children	Total number of immediate children of the class A number of children are increasing testing efforts, increase complexity and increase reusability Testing efforts also increase If greater number of children improper abstractions of parent class Higher the number of children lowers the maintainability

4	Coupling between Objects	It measures the interdependence of two classes/ objects High CBO indicates complex design, decrease modularity and increase the testing efforts of the class High CBO indicates less reusability Higher the coupling lowers the maintainability
5	Response for a class	It counts the total number of methods which are invoked for a single method call High RFC more effort required for testing, greater design complexity and fault proneness High RFC lowers maintainability
6	Lack of Cohesion in methods	Calculate the number of methods accessing the one or more attribute of the class If LOCM is high, then the method may coupled to another which increase the complexity of class design
Mood Metrics Suite		
1	Method inheritance factor [mood metrics]	Calculate the use of inheritance for operation and attributes
2	Coupling factors	Connection between elements between the OO design Coupling factors affect the class complexity, understandability and maintainability
OOAD metrics[2] by Lorenz and Kidd		
1	Class size measure	Total number of operations that are encapsulated within the class Number of attributes that are encapsulated by the class
2	Number of scenario script	Find out the size of application Define number of test cases need to develop for it.
3	Number of Key classes	Key classes indicate the amount of effort required for developing software and also an indication of the potential reuse of classes during system development
4	Number of supported classes	These classes are required for implementation of the system. They may not be directly related to the problem domain It is calculated as amount of efforts required to develop software and also indicate the amount of the potential reuse to be applied during system development
5	Average number of support classes per key class	These types of classes are GUI-implementation classes, other class which is needed for implementation of key classes They are used for calculating effort estimation of the system
6	Number of subsystems	This calculation is easy for scheduling the project work
7	Average operation size	Number of lines code or number of messages sent from the operation.
8	Operation Complexity	It should low because operation should limited to basic responsibility
9	Average number of parameters per operation	Total number of parameters are used per operation
10	Number of Public Methods(PM)	Estimating the amount of work in developing a class or subsystem
11	Number of Methods	Provide too much functionality. The class which has much functionality is having too much work
12	Number of Public Variable per class	Used for identification of key classes. The class which has more number of public variables than other imply that class has more relationship with other objects These classes are the central point of coordination of objects within the system
13	Number of Variables per class	Indicates the efforts required by that class in providing information to other classes
14	Number of Methods Inherited by a subclass	Calculate inheritance of the system
15	Number of methods Overridden by subclass	A larger number of overridden methods indicate the design problem. Subclass is a really a specialization of its super class
16	Number of methods Added by subclasses	New method name in the subclass
17	Average Method Size	Calculate the number of non-commented and non-blank line code in the class, divided by the number of its methods It is size metrics and find out abnormally large method in the class
18	Number of times a class is reused	Number of times the class is reused no other classes. It is similar to coupling It measures interclass communication and high value of it is undesirable
19	Number of Friends of a class	It defines the class coupling Friend classes violated the principle of encapsulation. High number of friends indicates the potential design flaw
Abereu Metrics: Measurement features of inheritance, encapsulation and coupling [39]		
1	Polymorphism Factor(PF)	It is the ratio of overriding methods in a class to the total possible overridden method It reduces complexity, so increase understandability and maintainability
2	Coupling Factor(CF)	Count the number of inter-class communications. Coupling increasing complexity and reducing encapsulation and potential reuse and limiting understandability and maintainability

3	Method Hiding Factor(MHF)	It measures encapsulation. It is hidden methods to total methods
4	Attribute Hiding Factor (AHF)	Used for hiding total attributes of the methods, it measures encapsulation
5	Method Inheritance factor (MIF)	Count the number of inherited methods as a ratio of total methods It measures inheritance and level of reuse of the system
6	Attribute Inheritance Factor(AIF)	It counts the number of inherited attributes as a ratio f total attributes. It expresses the level of reuse of the system

Coding: In this phase the program is written in machine level language. The program complexity and size of the code is important measures of this phase.

Table 10: GQM for Software Coding

Goal	Measure the Product code
Questions	What is the complexity of the code How many programmers need to work on the code All programming constructs are used properly (Variables and loop etc) Code maintain proper security
Measurement Parameters	Complexity measurement, line of source code written per month, pages of documentation per programmer month, Test cases written and executed per programmer month. , size of the program
Quality Measurement	Correctness, Reliability, efficiency, Integrity, interoperability, reusability, Portability

Table 11: Metrics for Code Design

Sr No	Metrics	Formula
1	Halstead complexity measurement matrices	Complexity is measured by counting the operand and operators. Three measurements are defined: 1) Length of program 2) Vocabulary of program 3) Volume of the program.) The length (N) of a program P is: $N = N1 + N2$) Program volume: - The program volume is measured on three parameters: program length, difficulty of program writing and efforts. The efforts measure is nothing but the time required to translate coding time.) The vocabulary (n) of a program P is: $n = n1 + n2$ Program Volume (V) The programming vocabulary and $= n1 + n2$ leads to another size measures which may be defined as : $V = N \log_2 n$ Potential Volume (V^*) It may be defined as $V^* = (n1^* + n2^*) \log_2 (n1^* + n2^*)$ Where $n1^*$ is the minimum number of operators and $n2^*$ is) The minimum number of operands.
2	McCABE'S Cyclomatic Complexity	It also finds out all probable execution of all programming paths and identification of dead code in the program. $E - N + P$ where E =number of edges in the flow graph N =number of nodes P =number of nodes that have exit points
	Starters' Program Complexity Measure [26]	Flow of control from one statement node say s_i to another s_j . Control flow from a statement node d_j to a statement node s_i which is declared in d_j . Flow from a declaration node d_j to statement node s_i through a read access of a variable or a constant in s_i which is declared in d_j . This measure is defined as $F(P) = E - n_s + n_t$ Where n_s = number of entry nodes. n_t = number of exit nodes
3	Code Grammar Metric:- These metrics measure the quality and complexity of the program	
3.1	Line Count[4]	It is an indicator of size. It is measured defects as a ratio of the number of defects/1000 lines of code.
3.2	Nesting Levels	Indicates the level of complexity of algorithms in a unit of code. It indicates unnecessarily complex conditions, which may make future modification increasingly difficult.
3.3	Counts of Decision Types	It is used for listing of single and multiple outcomes of decision statements. When used together with Cyclomatic Complexity, the value can determine the complexity and it identifies a suitable candidate for Refactoring.
3.4	Maximum Number of Predicates	This measure indicates overly complex decision statements, which are candidates for Refactoring.
3.5	Comment Lines	This measure is used for understandability.

		The measure can also be used in the ratios of comment lines to lines of code or comment lines to Cyclomatic Complexity to give an indication documentation levels within the code.
3.6	Token Count[26]	Tokens are nothing but distinct operators and operand $N = N1 + N2$ $n1$ = count of unique operators. $n2$ = count of unique operands. $N1$ = count of total occurrences of operators. $N2$ = count of total occurrence of operands
3.7	Jensen's Program Length Estimator [N1][26]	$N1 = \text{Log}_2 (N1!) + \text{Log}_2 (n2!)$ $n1$ = count of unique operators. $N1$ = count of total occurrences of operators.
3.7	Zipf's Program Length Estimator [Nz][26]	$Nz = n [0.5772 + \ln (n)]$ Where n is program vocabulary given as $n = n1 + n2$
3.8	Bimlesh's Program Length Estimator [Nb][26]	$Nb = n1 \text{Log}_2 (n2) + n2 \text{Log}_2 (n1)$ Where $n1$: Number of unique operators which include Basic operators, keywords/reserve- words and Functions/procedures. $n2$: Number of unique operands

Matrices for Testing: Testing is one of the important steps in SDLC, it assure the product quality before delivering the product to the end customer. With reference to three types of metrics product measurement metrics, measures the defect in the product and a number of test cases executed, how many of them are passed and failed. People efforts are measured in terms of a number of man-hours used for the testing product, test planning, test case development .The project metrics measure project scheduling, costing. The quality parameters measure tests quality plan and test case quality test completeness. [13]

Table 12: GQM Software Coding

Goal	Make sure that the product is solved the needs which is addressed at requirement phase. Identification of product defect and product is achieved all quality standard which are defined at SRS.
Questions	1. Test case development is correct and complete which test a complete system 2. Are Testing Processes processed in the proper direction? 3. Are quality test cases are developed? 4. All test cases developed
Measurement Parameters	Testing process efficiency, test cases, effectiveness, Quality of test cases. Number of test cases needed for testing the system
Quality Parameters	Testability

For the testing process measurement parameters are cost estimation, test data, test case development efforts, and schedule of testing need to measure. Apart from the process, the Quality measurement parameters are the effectiveness of test cases, qualitative test plan, and test completeness .The scope of testing [1], Test progression [2], Defect backlog [3][2],Staff Productivity [4] are also measurement parameters are used to it. The detail set of the matrices [14] are as follows:-

Table 13: Software Metrics for Testing

Sr No	Test Matrix	Purpose	Formula
1	Test Case Productivity	Test case writing productivity	Total test row steps/Efforts(hours) * Steps /hour
2	Defect Acceptance	Number of valid defect identified during the testing process	Defect Acceptance = [Number of Valid defects / Total Number of Defect * 100] %
3	Defect Rejection	Number of defects rejected during execution Testing team identified invalid defect open which facilitated the product development	Defect Rejected = [Number of defects rejected / Total number of defects] *100] %
5	Bad Fix defect	Determine effectiveness of defect resolution process. Identify the defect which gives the rise to new defect after solving it	Bad Fix Defect = [Number of Bad Fxi defect / Total number of Valid defects * 100] %100
7	Test Execution Productivity	Measure the test execution productivity	T= Number of Test executed / Execution Efforts per hours * 8] Execution /Day
8	Test Efficiency	Determine the efficiency of the test team in identifying defects. It also determines the defect missed out during the testing phase	Test Efficiency =[DT / DT+DU *100]% Where DT = number of valid defect identified during testing DU = Number of valid defects identified by user after release of application (i.e. Post testing, defect)

11	Defect Severity Index(DSI)	Quality of the product under test	Defect Severity Index =[(severity Index * N of valid Defect(s) for this severity / Total Number of valid Defects]
12	Cost Measurement	Costing of the testing process	Cost =Execution Efforts(hour) * Billing Rate
13	Effort Variance	variance in the Estimated effort	Effort variance = [Actual Effort - Estimated Effort / Estimated Efforts * 100] %
14	Schedule Variance	variance in the Estimated schedule , i.e. measure in number of days	Schedule variance = [Actual-Number of days - Estimated number of days/ Estimated number of days * 100] %
15	Scope Change	Measure the stability of testing process	[Total Scope - Previous Scope / Previous Scope *100]% where Total Scope = Previous Scope + New Scope , if new scope increase Total Scope = Previous scope - New Scope if Scope decrease

Table 13: OOAD testing matrices

Binder metrics set[43]		
1	Lack of Cohesion in methods(LCOM)	Higher the value of LCOM more state must be tested to ensure that methods do not generate side effects
2	Percent Public and Protected (PAP)	Calculate the percentage of attributed, are public and protected in the class. The high value of PAP increases the side effect among the classes. It leads to high coupling. Test cases need to designed to ensure that side effects are uncovered
3	Public Access Data members(PAD)	Encapsulation principle is violated. Hence, high PAD leads to the potential for side effects among the classes Test cases need to designed to ensure that side effects are uncovered
4	Number of root classes	Count number of distinct class hierarchies that are descriptive in designer model As NOR increases, testing efforts are also increasing
5	Fan-in(FIN)	Inheritance hierarchy -indication of multiple inheritance FIN >1 indicate class inherits attribute and operation from more than one class It should avoid
6	Number of children (NOC) and depth of the inheritance tree (DIT)	Super class methods need to be retested for each subclass

Implementation and Maintenance:

When the product is released it is in the maintenance phase. It addresses the customer defect. This increase the customer satisfaction.

Table 14: GQM Software Maintenance

Goal	Keep the software operational. User training for the operation of the software Any problem occurs during the software operational phase, it should solve immediately. Any modification can do easily
Questions	. What is the time required for repairing the software after failure? . Is it easy to modify the software? . Is software easily accommodating any change?
Measurement Parameters	Software documentation and organization
Quality Parameters	Maintainability, Efficiency, flexibility,

Table 15: Software maintenances Metrics

Sr No	Metrics	Purpose	
1	Software Maturity Index	It indicates the stability of software products	$SMI = M_T - (F_a + F_c + F_d) / M_T$
			M_T - number of modules in the current release
			F_c - number of modules in the current release that have been changed
			F_a - number of modules in the current releases that have been added
			F_d - number of modules from the preceding release that were deleted in current release

2	Software Reliability Measurement	Mean time between failure	MTBF= MTTF +MTTR
			MTTF - mean time to failure
			MTTR- mean time to repair
		Failure-in-time	a statistical measure of how many failures a component over a one billion hours of operation
		Software availability	Availability= MTTF/ MTTF +MTTR * 100 %
3	Fix backlog and backlog management index	It is a simple count of the reported problems that remain at the end of each month or each week.	$BMI = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$
4	Fix Response Time and Fix Responsiveness	IT calculate total time required to solve the customer problems	Mean time of all problems from open to closed
5	Percent Delinquent Fixes	If the time is increased for each defect fix then the required response time, then it is called as delinquent. This metric calculates the percentage of this delinquency.	$\text{Percent delinquent fixes} = \frac{\text{Number of fixes that exceeded the response time criteria by severity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$
6	Fix Quality	Number of defects fixed in a specified period.	
7	Problems per user month (PUM):	PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year	PUM =Total problems that customers reported (true defects and non-defect-oriented problems) for a time period ÷Total number of licenses-months of the software during the period Where, Number of licenses-months = Number of installing licenses of the software × Number of months in the calculation period

V Observations

In the literature number of matrices is proposed for it, but we find the metrics are more focused on requirements correctness and completeness. There are 24 quality parameters are mentioned for good SRS [46]. It has been observed defined metrics only measure the 50% of these parameters. The remaining quality measurement parameters are : Achievability of the requirement , conciseness, design independence, conciseness , Internal and external consistency , reusability of the requirement, risk analysis is not measured . There are no metrics for measuring the identified requirements are correctly transferred into functional and behavior aspects. The metrics need to design for requirement risk analysis, requirement organization, requirement achievable, and requirement storage and requirement reusability measurement. There is probability that too many nonfunctional requirements may create uncertainty in development. Hence there is a need for strong measurement of non-functional requirement. The maintainability is one of the important measurements for the requirement engineering. The stability metrics determine the requirement constancy, but the requirements maintain need to measure. The analysis phase is more focus on project management. Plenty of metrics are designed for project estimation, but we are not finding the metrics which suggest the correction of adopted software development process. In coding, most of the matrices measure the code complexity, but some of the quality parameters are left out like-code reusability, interoperability, code efficiency. Testing metrics are defined for the total number of test cases needed to execute the product, but there is no metric which defined how many levels of testing is needed for the product. There is no metrics suggested for measuring non-functional testing and risk. To test the requirement, there is a dependency between two test cases. There is a need to measure these dependencies and association between test cases. In future studies, we will find out these measurements.

VI Conclusion

To deliver the quality product to the end customer, it needs measurement techniques. The quality and functionality of the software product needs to measure. This paper reviews the different measurement metrics in SDLC. The paper review all SSAD and OOAD development metrics. We observed all direct measurements in SDLC, in future study we will try to evaluate all indirect measurements also.

References

- 1 Mohammad A Rob, " Issues of Structured Vs Object Oriented methodology of Systems Analysis and Design " Issues in Information Systems , Volume V No 1, pp 275-280 ,2004
- 2 M. Lorenz and J. Kidd, Object Oriented Software Metrics. Prentice Hall Object Oriented Series, 1994
- 3 Rita J Costello and Dar-Biau Liu "Metrics for requirement engineering " journal of systems and Software, Volume 29 Issue 1,pp39-63 ,April 1995

- 4 Zohra Bellahsene , Dillip Patel and Colette Rolland Object -oriented Information Systems pp 409-421,2002
- 5 Y Chernak,"Validating and improving test cases ", IEEE Software , Vol 16. No 1 pp 81-86, 2001
- 6 B Berger , Evaluating Test plans with Rubrics . International Conference on Software Testing Analysis and Review , 2004
- 7 P. Piwaowarski , M Ohba, J Caruso , Coverage, " Measurement Experience during Functionla Test. International Business Machines Corporaion .", IEEE Software Proceodings, 1993
- 8 M Marre, A. Bartolino , " Using Spanning Sets of coverage testing " , In IEEE Transaction on Software Engineering , Volume 29, November 2003
- 9 M. L. Hutchoon Software Testing Fundamentals; Methods and Metrics . John Willey and Sons
- 10 Burnstein I, Suwanassart T, and Carlson C "Developing a Testing Maturity Model Part-I, Crosstalk , Journal of defence Software Engineering , 9 , no 8, pp21-24 ,1996
- 11 S. R Rakkitin Software Verification and Validation for software Practitioners and Managers . Artech House Inc . Bosten London 2001
- 12 Stephan H Kan ,Metrics and Model in Software Quality Engineering Pearson Education , 2003
- 13 Naresh Chauhan "Testing Metrics " , " Software Testinf Principles and Practices", Oxford University Press 2012
- 14 Yanping Chen , Robert L. Probert , Kyle Robenson " Effective Test Metrics for Test Strategy Evolution " Proceeding of the 2004 conference of the Advance studies on collaborative Research CASXON 2004
- 15 Vanitha N, ThirumalaiSelvi R " A Report on the Analysis of Metrics and Measures on Software Quality Factors – A Literature Study" (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (5) , 2014, 6591-6595 , ISSN:0975-9646
- 16 B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering "(version 2.3), Keele University, UK, Tech. Rep. EBSE-2007-01, 2007.EBSE Technical Report
- 17 Stephen MacDonell, Martin Shepherd, Barbara Kitchenham and Emilia Mendes How Reliable Are Systematic Reviews in Empirical Software Engineering? May 7, 2009
- 18 Derick Bailey "The Introduction: Productivity Metrics In Software Development "2009
- 19 Linda Westfall Team "12 Steps to Useful Software Metrics " Report
- 20 Everald E. Mills "Software Metrics SEI Curriculum Module SEI-CM-12-1.1"
- 21 Claudiu VINȚE Software Architecture Coupling Metric for Assessing Operational Responsiveness of Trading Systems Informatica Economicăvol. 16, no. 4 ,2012
- 22 Mrinal Kanti Debbarma, Swapan Debbarma, Nikhil Debbarma, Kunal Chakma, and Anupam Jamatia A Review and Analysis of Software Complexity Metrics in Structural Testing International Journal of Computer and Communication Engineering, Vol. 2, No. 2, March 2013
- 23 T. J. M. Cabe, "A complexity measure," IEEE Transactions on Software Engineering, vol. 2, 1976
- 24 I. Herraiz, J. M. G. Barahona, and G. Robles, "Towards a theoretical model for software growth," in 29th International Conference on Software Engineering Workshops (ICSEW'07), 2007
- 25 W. Harrison, K. Magel, R. Kluczny, and A. Dekok, Applying Software Complexity Metrics to Program Maintenance Compute, vol. 15, pp. 65-79,1982
- 26 Gurdev Singh , Dilbag Singh, Vikram Singh "A Study of Software Metrics" IJCEM International Journal of Computational Engineering & Management, Vol. 11, , ISSN (Online): 2230-7893, pp22-27 ,January 2011
- 27 Tharwon Arnuphaptrairong The Development and Achievements of software Size Measurement" Proceedings of International MultiConference of Engineerings and Computer Scientists 2012 vol1 IMECS , pp14-16, 2012 , Hong Kong , March 2012
- 28 O. Adekile, D.B. Simmons, W.M. Lively, "Object oriented software development effort prediction using design patterns from object interaction analysis." in Proceeding of 2010 Fifth International Conference on Systems, Menuires, pp.47-53 , April, 2010,
- 29 D. Reifer, Web-Development: "Estimating quick-time-to-market software." IEEE software, vol. 17, no. 8, pp.57-64, November/December 2000
- 30 G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class points: an approach for the size estimation of object-oriented systems." IEEE Transaction on Software Engineering, vol. 31, no.1, pp.52-74,
- 31 A. Minkiewicz, "Measuring object-oriented software with the predictive object points," in Proceeding of 8th European Software Control and Metrics Conference, Atlanta, 1997.
- 32 A. J. Albrecht, "Measuring application development productivity," in Proceeding of the IBM Applications Development Symposium, California, , pp. 83-92 ,October 14-17, 1979
- 33 M. Nasir, "A survey of softwareestimation techniques and project planning practices," In Proceeding of the Seventh ACIS Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'06), pp. 305-310. , 2006.
- 34 B.W. Boehm, "Software engineering economics," IEEE Transaction of Software Engineering, vol.10, no.1, pp. 4-21, 1984.
- 35 B. Anda, H. Dreiem, Dag I. K. Sjoberg and M. Jorgensen, "Estimating software development effort based on Use Cases -- experiences from Industry, In G. Martin, and C. Kobryn (ed) UML 2001: the unified modeling language: modeling languages, concepts, and tools: 4th international conference, Toronto, Canada, October 1-5, 2001
- 36 McCabe, T. J.; (1976) "A Complexity Measure". IEEE Transactions of Software Engineering, Volume SE-2, page(s): 308-320., 1976
- 37 W.K.S.D. Fernando ,D.G.S.M. Wijayarathne ,J.S.D. Fernando ,M.P.L. Mendis,I. Guruge "THE IMPORTANCE OF SOFTWARE METRICS: PERSPECTIVE OF A SOFTWARE DEVELOPMENT PROJECTS " Research Symposium, Engineering Advancements , 2014
- 38 S. R Chidamber and C. F Kemerer A metric suit for object oriented design. IEEE Transaction on Software Engineering , pages 467-493, 1994
- 39 F. Brito e Abreu , M. Goulao and R Esteves , " Towards the design quality evaluation of OO Software System" , in Proceedings of the 5th International Conference on Software Quality, Austin, Texas, 23 to 26 October 1995
- 40 B. H. Yin and J. W. Winchester "The establishment and use of measures to evaluate the quality of software designs" ACM SIGSOFT Software Engineering Notes Volume 3 Issue 5, PP 45-52 , ACM SIGSOFT Software Engineering Notes
- 41 Henry and D. Kafura , " The evaluation of software systems' structures using quantitative software metrics.", Software Practice and Experience, 14(6):561–573, June 1984
- 42 W. M. Zage and D. M. Zage. Evaluating design metrics on large-scale software. IEEE Software, pages 75–81, July 1993.
- 43 Binder R, "Testing Object -Oriented System: A status Report ", American Programmer vol 7,No4, pp 23-28April 1994,

- 45 Shahid Iqbal and M. Naeem Ahmed Khan, "Yet another Set of Requirement Metrics for Software Projects" , International Journal of Software Engineering and Its Applications , Vol. 6 , No. 1, pp19-28 ,January 2012
- 46 Alan Davice , Scott Overmyer et al, " Identifying and measuring Quality in Software Requirements Specification ", IEEE-08186-3740-4/93
- 47 Mohd.Haleem, Prof (Dr) Mohd.Rizwan Beg,Sheikh Fahad Ahmad "Overview of Impact of Requirement Metrics in Software Development Environment" international Journal of Advanced Research in Computer Engineering & Technology (IJARCET)Volume (2) No 5 pp1811-1815 No 5, May 2013 ISSN 2278-1323
- 48 Fischman, Lee; McRitchie,Karen; and Galorath, Daniel D. "Inside SEER-SEM", CROSSTALK , The Journal of Defense Software Engineering. Volume 18, Number 4, April 2005
- 49 Mike Cohn, "Agile Estimating and Planning". Mountain Goat Software,Sunday, November 9, 2008
- 50 Putnam, Lawrence H.. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-4, NO. 4, pp 345-361. 1978
- 51 Capers Jones "Software Engineering Best Practices", McGrawHill, October 2009,pp 318–320 [1]
- 52 Cem Kaner, Walter P. Bond "Software Engineering Metrics: What Do They Measure and How Do We Know?" 10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, METRICS 2004
- 53 Dimitris Stavrinoudis, Michalis Xenos And Dimitris Christodoulakis , "Relation Between Software Metrics And Maintainability" Technical Report No. Tr99/08/03
- 54 Mrinal Kanti Debbarma et al , " A Review and Analysis of Software Complexity Metrics in Structural Testing ", International Journal of Computer and Communication Engineering, Vol. 2, No. 2, March 2013
- 55 Geoffrey K Gill and Chris F Kemerer , "Cyclomatic Complexity Density and Software Maintenance Productivity " , IEEE Transaction of Software Engineering , vol 17 , No 12 December 1991
- 56 Zuhab Gafoor Dand , Prof. Hemlata Vasishtha , " Analysis and Evaluation of Quality Metrics in Software Engineering" , International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 4, April 2015
- 57 Fariba Dinari , "Halstead Complexity Metrics in Software Engineering" , Journal of Renewable Natural Resources Bhutan, ISSN: 1608-4330, Vol 3;7, pp 418-424, 2015
- 58 Alain Abran, Pierre N. Robillard "Function Points: A Study of Their Measurement Processes and Scale Transformations ", J. SYSTEMS SOFTWARE , 25, pp 171-184, 1994;
- 59 McCabe T.J., "A Complexity Measure ", IEEE Transaction on Software Engineering, SE-2, PP. 308-320, 1976 ,
- 60 Pankaj Jalote " Design " A Concise Introduction to Software Engineering , Springer,ch.6, 2008, 171-175
- 61 Roger Pressman " Product Metrics" Software Engineering A Practioner's Approach , seventh Edition McGrawHill,ch.23, 624-628
- 62 Joseph K. Kearney, Robert L. Sedlmeyer, William B. Thompson, Michael A. Gray, And Michael A. Adler , "Software Complexity Measurement" , Communications of the ACM , Volume 29 Number 11 , November 1986
- 63 Navneet Kaur, Ashima Singh "Component Complexity Metrics : A Survey" , International Journal of Advanced Research in Computer Science and Software Engineering ,ISSN: 2277 128X , Volume 3, Issue 6, June 2013