A Survey on Software Design Based and Project Based Metrics

Shweta Sharma and S. Srinivasan

***Abstract*—Software metrics are prevalent in the software industry to measure the features of software regarding its size, design, complexity and performance so that quality software can be deployed to the customers. In this paper, we've presented a survey of existing software metrics so that appropriate metrics can be chosen depending upon the need of the project. The paper first provides the overview of product attributes which are measured by the metrics. The existing software metrics have been classified based on software design and project usage. We've gone through fifty research papers to cover all existing metrics in the aforesaid categories. The motive of this survey is to reinforce the uses and limitations of these metrics so that suitability of specific metrics can be defined for an application purpose.**

***Index Terms*—Object-oriented metric, software complexity, software development life cycle, software metric.**

I. Introduction

Software engineering is the discipline of engineering which deals with building the quality software reducing the software cost and time utilizing the best engineering and management practices. The software development field faces abundant challenges and setbacks to survive in the competitive world of globalization. The software industry is characterized by standards and technological dynamics with the emergence of new programming languages, new development approaches and innovative frameworks. Hence it becomes quite essential to evaluate the performance of software before releasing it in the market for widespread use. Software metric is a standard way to evaluate the quality related to a software system or software process based on some property [1]. The metrics are used by all engineering disciplines to measure the attributes like weight, pressure, size, density, temperature and wavelength. The measurement may be in quantitative as well as in qualitative terms. Software metrics are basically related with the measurements of software features or properties for example size, complexity of software and rework in terms of some numbers to improve all aspects of overall management of that specific process [2], [3].

For example, size is a product property or attribute which is generally measured in kilo lines of code (KLOC) [4]. Function Point (FP) metric is also used to estimate or measure the size of the project. Rework is an attribute of the process and the effort spent on rework is a measure of the rework attribute. The measurements can be direct or indirect.

Manuscript received November 29, 2021; revised January 14, 2022.

Shweta Sharma is with Mewar University, Chittorgarh, Rajasthan, India (e-mail: bhardwaj.shweta28@gmail.com).

S. Srinivasan is with Department of Computer Science and Applications, PDM University, India (e-mail: dss_dce@yahoo.com).

The direct measurements of software products include the lines of source code, number of defects detected, execution speed of software, throughput per hour, response time and turnaround time etc. The indirect measurements of software constitute efficiency, portability, comprehensiveness, simplicity, maintainability, understandability etc.

The metrics are the best way to measure internal and external properties of software.

A. Internal Properties/Attributes

Internal attributes are specific to the process, resource or product themselves. They are measured on their own regardless of the operating environment [5].

B. External Properties/Attributes

Every software product is operated in an environment and hence it is required to measure how the process, resource or software product relates to the external environment [5]. The software metrics answers all questions related to the software measurements in terms of size, complexity, number of resources required, bugs detected *and* so on [6].

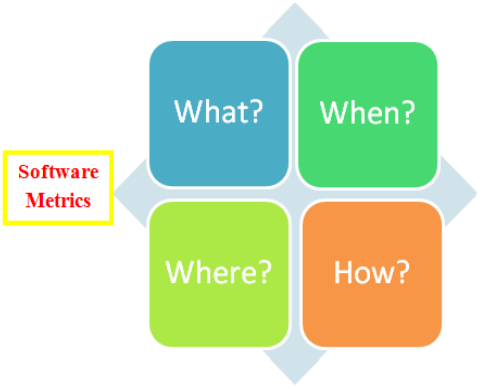


Fig. 1. Questions answered by software metrics.

To see what sort of metric, we require, we need to concentrate on the following questions (Fig. 1):

- How to measure the software Size?
- How much cost will be required to build the software?
- What must be the staff size of a project?
- What is the complexity of a module?
- How many test cases should we apply?
- When can we stop testing?
- When can the software be deployed?
- Which test technique is appropriate for the software?
- What will be the productivity?

The software metrics, if applied properly to all types of software products, helps in developing a clear blueprint of a system in context with whether all the features of an expected software product have been properly integrated in it or not [7].

DOI: 10.7763/IJCTE.2022.V14.1310

II. Related Literature

There are several properties in the physical world that are used to measure anything such as mass, length, and time. Weyuker (1988) has proposed nine attributes for measuring a software using software metric. Some of these properties constitute: "monotonicity, interaction, non-coarseness, non-uniqueness, and permutation". These attributes are equally applicable on traditional and object-oriented approaches. When the software is first being developed, these measurements provide designers a chance to modify it, reducing complexity and improving the product's long-term viability. In order to accurately portray the program that is being measured, software measures and metrics must be used [8]. The Fig. 2 below shows the metrics hierarchy according to our classification.

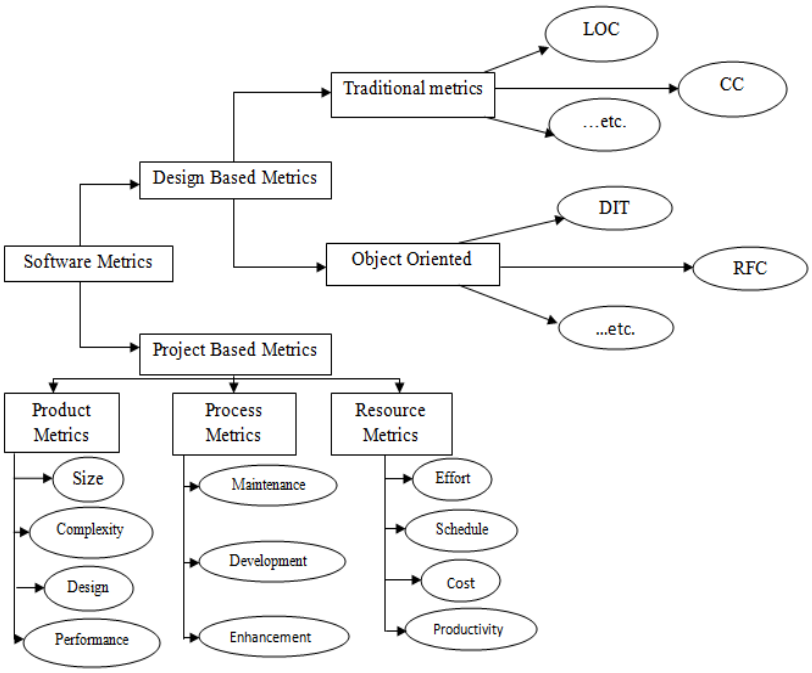


Fig. 2. Classification of Software Metrics.

The above Fig. 2 has divided the metrics in two basic categories: Design Based Metrics and Project Based Metrics.

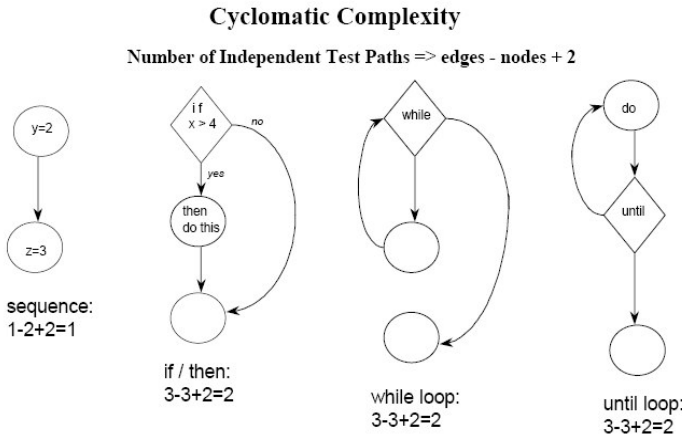
A. Design Based Metrics

The design-based metrics are based on the software product's design features. The two main types of design-based metrics are traditional and object-oriented metrics explained below.

Traditional Metrics: There are various metrics that are applied to conventional functional development [9]. The SATC (Software Assurance Technology Center) has acknowledged three such metrics which are appropriate to object-oriented development: "Complexity, Size, and Readability".

CC (Cyclomatic Complexity) (Fig. 3): McCabe's Cyclomatic Complexity method is used to assess the complexity of a technique's algorithm [10]-[12]. It is a calculation of the number of test cases required to thoroughly test the procedure.

"The number of edges minus the number of nodes plus 2 is the formula for calculating cyclomatic complexity" [13], [14]. Only one test case is required for a sequence with only one path and no choices or options. An IF loop, on the other hand, offers two options: Test the specific path for which it satisfies the condition, and test another path if the condition fails [15].



examples of calculations for the cyclomatic complexity for four basic programming structures.

Fig. 3. Cyclomatic Complexity for different types of structures (source: [16]).

An approach with "a low cyclomatic complexity" is often superior, as shown in the graph. This could indicate that there is less testing and more comprehensibility, or the decisions are postponed by message passing, but it does not suggest that the approach is simple. The inheritance makes it difficult to use cyclomatic complexity in estimating the complexity in a class but if cyclomatic complexity of individual methods is computed then combined complexity of all such methods with other measures can determine the complexity of the class [17]. Although this metric is specifically for assessing Complexity, it is also related to all the other qualities.

Line of Code Metric: - Estimation based on "Lines of Code (LOC)" is a legacy method still utilized in many organizations for sizing commercial applications. Typically, for technology conversion projects, the LOC technique can be used for estimation. In this method, all the functions are estimated for their size (in LOC) and appropriate productivity factors applied for arriving at realistic estimates [18]. Based on past experience and relative productivity ratios for various programming languages and tools, productivity factors are defined for the same. Adjustments are also made for the known weaknesses in the method, to arrive at the final estimates. There are also other factors that can influence the definition of LOC like programming language, count type (logical or physical), compiler directives, comments, blank lines, statement types (executables vs. declarations), consideration of software reuse, and so on. This method has several disadvantages [18], [19]. For instance, it usually tends to reward profligate design and penalize concise design, it does not have any industry standard (IEEE or otherwise) and it is very difficult to normalize across the platform, language, or organization.

Object Oriented Metrics: R. Hudli, C. Hoskins, and A. Hudli (1993) have suggested Object Oriented Metrics which are as below: -

Depth of Inheritance Tree (DIT): DIT is a design-based metric used at class level. It is the measurement of inheritance at a class level from root to a specific class. The maximum fault sensitive classes reside in the middle of the inheritance tree [20], [21]. The reusability is increased in deep trees due to inheritance. The number of faults is increased in case of high DIT. The recommendation for DIT number is five or less [22].

Number of Children: NOC is the count of subclasses which are directly derived from a class. It is the number of immediate sub classes which are derived from a class using inheritance feature. In general, it is used to measure the width of a class hierarchy. High value of NOC promotes the reusability and is also an indication of less faults [21], [22].

Coupling between Objects Classes: This metric is used to calculate the strength of interdependence among classes. CBO is a count of the total number of classes which are coupled to a specific class. Coupling between classes is detected if one class's methods make use of methods defined by some other class. Coupling should be minimized whenever possible as it always increases errors and faults [20]. If a method is involved in overloading or overriding then it is considered to be polymorphic and all classes called by such method's class are counted in the coupled class [23].

Response for a Class (RFC): This metric is also used at class level. It is defined as the count of methods executed within a class in response to the messages sent by other classes to the object of this class. RFC is the actual number of available methods in a specific class [24], [25]. The methods availability of a class is calculated by adding the number of methods local to this class and other methods called by these local methods. If the calculated value is high then it is the indication of requirement of more rigorous testing due to coupling related problems [21].

Lack of Cohesion (LCOM): LCOM is also a design-based metric used at class level. It is used to measure the degree of cohesion inherent in the elements of a class. It is calculated by counting the local methods which are disjoint in a class. Disjoint sets of methods imply that they have nothing common among them [26].

Weighted Methods per Class: WMC is a count of the number of methods executed in a class. In other words, it is the addition of complexity of all methods defined within a class proximity. The effort and time needed to create and maintain a class can be predicted by calculating the number of methods and complexity of the concerned methods. Hence WMC with lower value is desirable as high value is an indication of greater complexity [27].

B. Project Based Metrics

Project based metrics are those metrics which are concerned with the quality of the deliverable product as well as the process and resources involved in constructing that product. It deals with the basic quality of the product like product size, design features, complexity and its performance [28]. Project base metrics also measure the processes used in the development of the product for example the product which is built from scratch will have a different process life cycle from the one which has been converted into a new version with some enhancements. As a set of software-related actions, processes observe and control the status and development of a system's design and estimate future impacts. Most processes are associated with a timeframe. When a task must be completed by a certain date, the timing can be explicit or implicit. According to the existing literature, the following examples of process related metrics are suggested to be collected [24]: -

- Total hours spent on development: - Every process and sub process have its own amount of time allocated to it. It requires a lot of effort and time to alter models from earlier processes, all sorts of modules, such as "use case specification and object specification, as well as use case design and block testing" for each individual object must be collected. The number of various types of defects detected during reviews.
- Costs associated with quality assurance.
- Costs associated with introducing new development processes and tools.
- Project schedule involved in the development of the product.

Metrics for software products are utilized to measure the quality of the software. Incomplete software products are analyzed using these metrics just to determine the level of complexity and anticipate the attributes of the final product. Anything that comes out of a process is called a product. Not all products are those that management has promised a client. Software lifecycle artifacts and documents can be evaluated.

Many types of product-related metrics have been suggested. None of these have been shown to be beneficial as a broad interpreter of overall quality.

Product metrics: When it comes to software development life cycle metrics, it's all about the work product. As a result of these measurements, the project's qualities and execution may be seen [29]. The number of defects per unit size is a metric which actually does the measurement of software product's quality. The turnaround time taken for servicing a maintenance request can be looked at as a measure of service quality. There are various dimensions of a product which can be measured using product metrics such as-

- Size of Product
- Inherent Complexity of product
- Design Features
- Performance

Size estimation is a sophisticated process that requires the results to be updated with real counts throughout the life cycle. Source lines of code, function points, and feature points are all size measures [30], [31]. Complexity is a function of scale, which has a significant impact on design flaws and hidden defects, resulting in quality issues, cost overruns, and schedule slippages.

Complexity must also be evaluated, monitored, and controlled on a regular basis. Changing requirements is another issue that contributes to size estimate mistakes, and it, too, must be baselined and closely monitored.

SLOC (Source Line of Code), FP (Function Point) and CC (Cyclomatic Complexity) are the basic metrics utilized to evaluate software size and software product complexity which have already been discussed in previous section [32].

Design features of a software product can be design reusability, architecture independence, information hiding, simplicity, bonding of module's elements, interdependence on other modules expandability and maintainability. The Object Oriented Metrics can also be used to evaluate the design features such as coupling, cohesion and information hiding features[33].

The performance of software products can be measured through many testing and reliability metrics such as MTBF (Mean Time Between Failure), MTTR (Mean Time To Repair), Average Turnaround time, Average Response Time, Throughput and so on [34], [35].

Process metrics: These are the metrics used for evaluating the effectiveness and quality of software processes, including maturity of the process, work required in the process and the efficiency of fault correction during development, among other things [36]. The process metrics are concerned with the development cycle of the software from beginning to the end. The metrics can be used in any phase of the life cycle of a software product from requirement analysis to implementation and maintenance. The software product life cycle can take any of the following forms-

- Software built from scratch (development project)
- Existing software undergoing for some changes (Conversion/Enhancement project)
- Software with extreme modifications (Maintenance Project)

The process metrics used in life cycles of the projects of these above-mentioned categories of projects can vary. However, there can also be some common set of metrics which can be used for all types of software projects. The next section will discuss such categories of software project metrics existing in current literature.

Development Project Metrics: These are the metrics which can be used in development phases of software. The Software Development Life Cycle consists of phases such as requirements engineering, software design, coding, testing and implementation and operation and maintenance.

Requirement Coverage metrics: These metrics encompass the requirement accumulated for the entire project [37], [38]. This can be done by drawing a requirements traceability matrix. The metrics checks the following parameters: -

- The number of requirements which have test cases.
- The number of requirements which have been tested so far.
- The number of requirements which have cleared the design specification criteria.

The requirements are checked for completeness and correctness. Every requirement must have a single interpretation and can be checked for backward and forward traceability with its origin.

Design Metrics: These metrics are used at the design phase of the development life cycle. The software design can be checked for its size, complexity, performance and other significant features such as encapsulation and reliability [39]. The metrics discussed in previous sections can be used to measure software design phase.

Code Coverage Metrics: Code coverage is a software testing measure that evaluates the number of lines of code that are effectively validated throughout a test procedure, which aids in determining how thoroughly a software is tested [40]. This metric is used to test the code from angles:

- Statements Coverage
- Loops Coverage
- Conditions Coverage
- Branch Coverage

There are many benefits of using this metric: -

- Saving maintenance efforts: - The software maintenance phase is a very expensive and time-consuming phase. If the code is thoroughly tested and validated, then the maintenance process takes lesser time and effort.
- Exposure of faulty code: - The unused, dead and bad code can be easily revealed with continuous improvements and analysis which ultimately results in better quality product. Fast development process: The software development process can be completed early with increased efficiency if source code is properly tested and verified.

Test and implementation Metrics: Software metrics are the "Standards of Measurements" with predefined guidelines, methods and resources to test the software. The metrics used in the testing phase can be for quantitative and qualitative measurement of software. The quantitative measurement focuses on "extent, amount, dimension, capacity, or size of some attribute of a product or process". And qualitative measurement reinforces the quality attributes such as efficiency, reusability, maintainability and robustness [41]. The test metrics can be of many types: -

- Percentage of tests executed

- Percentage of tests not executed
- Number of defects found after testing
- Test cases percentage qualified by software
- Test cases percentage not qualified by software
- Blocked test cases percentage
- Defects Density
- Defects Leakage

As the intention of every test strategy is to break the software so that all possible bugs can be detected hence the rigorous test case approach is used.

Operation and Maintenance Metrics: This is generally the last phase of the development life cycle when the newly built software comes into an operative environment after qualifying alpha and beta testing. The maintenance phase becomes very time consuming and expensive if the bugs are not traced and rectified earlier in the life cycle [42], [43]. Being the last phase of the cycle, much cannot be done to amend the product quality at this stage. The existing literature suggests that the following fixes can be incorporated to eliminate the errors: -

Fix backlog and backlog management index: - The defect arrivals rate and the rate at which fixes for reported problems become available are related to fix backlog [44]. It's a simple tally of reported issues that haven't been resolved at the end of each month or week. The BMI is calculated (Fig. 4) as [45]:

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

Fig. 4. BMI calculation formula.

If BMI value is greater than 100; it signifies that the backlog of defects has decreased and value less than 100 is an indication of increased defects.

Fix response time and fix responsiveness: This is the time taken to fix the problems encountered. The average time spent in handling the change request from open to close is called fixed time. If the response time is short, it gives satisfaction to the customer [35], [44].

Percent delinquent fixes: If the time taken from opening to closing a fix exceeds the response time greatly then it is called as delinquent [35], [44]. The delinquent can be calculated using the following formula (Fig. 5):

$$\text{Percent Delinquent Fixes} = \frac{\text{Number of Fixes that exceeded the response time criteria by severity level}}{\text{Number of Fixes delivered in a specified time}} * 100$$

Fig. 5. Percent delinquent calculation formula.

Fix quality: Fixes are used to rectify the problems encountered during the maintenance phase. But when a fix becomes another problem by rectifying the original defect but introducing new defects in the software then that fix is called a defective fix and needs to be traced. This metric takes the count of defective fixes in a regular time interval, say monthly or weekly. A defective fix might be documented in one of two ways: in the month it was identified or in the month it was delivered. The former is a measure from the customer's viewpoint, while the latter is a measure of the process. The latent duration of the flawed fix is the difference between the two dates [36], [44]. Sometimes the software products require too many amendments in the maintenance phase of the development life cycle. The reason for alterations may be changing requirements of customers or clients, emergence of a new technology, change in the project team mates or project manager and so on. There are numerous bugs and faults detected at this phase that either they cannot be handled or make the maintenance process too costly and out of budget. Such problems give rise to another development cycle called a software transition or maintenance project. The metrics used in such cases are described in the next subsection.

Resource Based Metrics: **"**Entities required by a process activity are known as resources". For our purposes, the resources include any inputs used in software development. These are candidates for measurement: individuals, materials, tools, and techniques using resource-based metrics. Resource metrics are also very crucial to select and examine as the entire budget of software is dedicated in organizing and utilizing the system resources such as manpower, tools and machinery, money and the last but not the least time. The metrics used for resource estimation and evaluation are explained below.

Effort Measurement: Efforts are measured to evaluate the efficiency of the project team by comparing the actual efforts with the expected efforts [46]. The project team size is always pre-determined at the beginning of the project. For example, how many people will work for how many days or months in a specific phase of the life cycle? The unit of measurement is generally Person Days or Person Months. The effort slippage percentage can be calculated phase wise to know the deviations from the expected efforts. Finally, a causal analysis can also be done to know the reasons for deviations.

Schedule Measurement: Schedule is a timeline for a project consisting of milestones and deadlines for activities and tasks to be performed during the lifetime of the project [47]. Time is the most critical and significant factor for any software development industry. Software teams have to undergo a lot of pressure to complete each milestone in a predetermined time frame. Time and Budget are two very important criteria to evaluate success of a project. Hence following a proper schedule is essential in project management. But if there is any kind of slippage from the schedule then that slippage must be detected and reasons for slippages must be investigated.

Schedule Slippage metric can also be used to evaluate the performance of software testers and testing teams [48]. This is an important service metric used to measure the schedule adherence and the quality of the process. It is computed by comparing the actual time spent and the expected time that must be spent on an activity. After calculating schedule slippage, schedule variance can also be computed. Schedule variance is used to measure the deviation in overall completion of the software project [49]. It is calculated as the percentage of difference between the expected completion date and actual completion date of the project.

Cost Estimation: Cost is one of the most important factors to measure in a project environment. Cost of every project is estimated at initial phases of development [50], [51]. Sometimes, past experience is used as a guide for cost estimation, but one project may be different from the other, so only past experience is not adequate to measure. Cost estimation models such as COCOMO (Constructive Cost

Model) can be used for cost prediction which has pre-defined set of formulas for cost estimation [52], [53].

Productivity measurement: Productivity is also a project measurement which is the measure of line of code produced per person/month (year) [54], [55]. Hence if project size and effort are known in advance then productivity can be calculated as-

Productivity (P) = Project Size / Efforts in Person Month Unit.

III. Analysis and Interpretation of Survey

After doing an in-depth study of existing design based and project-based metrics, the limitations are analyzed. And it is observed that because the software does not need to be executed, static measurements are easier to collect. These metrics are generally available since they may be gathered at an early stage of program development. The static metric LOC was introduced to measure the program productivity but later on it was declined as there is no standard definition of LOC for example, the comments in a program should be counted as line of code or should be ignored. Later, the Cyclomatic Complexity metric given by McCabe came into picture which used flow graphs and mathematical equations to compute software complexity.

When it comes to measuring quantity attributes such as size and complexity, static metrics work well, but when it comes to quality attributes such as testability and reliability, static metrics fall short because they can only be evaluated through a static inspection of the software artifacts themselves.

Based on the data accumulated during the execution of the actual system, dynamic metrics are calculated, reinforcing quality attributes such as defect probability and performance. We can observe that dynamic metrics are comparatively more accurate than static metrics when considering the constraints of static metrics.

In contrast to static metrics, the calculating method for dynamic metrics is far more challenging. Further research is needed in areas such as dynamic coupling and cohesion measures. So, we may infer that a hybrid method of static and dynamic measures can prove to be advantageous for avoiding computational efforts and for qualitative measurements. Similarly project based metrics too have their own drawbacks for example improper estimation of technical expertise of project team members may lead to inaccurate estimation of software schedule, cost and productivity. The pros and cons of every metric discussed in this study have been pointed out in the Table I below.

TABLE I: Pros and Cons of Metrics

Metric Name	Type	Use	Limitations
LOC	Traditional Product Metric	To measure size of Program Code	Highly dependent on programming style. Not a standardized measuring technique.
CC	Traditional Product Metric	To measure complexity of program	Only measures control complexity not data complexity, does not support object-oriented features
NOC	Object Oriented Metric	To count number of classes directly derived from the base class in a class hierarchy	Very difficult to modify since it affects all its children because of having a heavy dependence on the base class. More testing is required.
CBO	Object Oriented Metric	To count number of coupled classes to a class	Only variable references and method calls are considered. Other references like utilizing used defined types, API calls, use of constants, handling of events and object instantiations are neglected.
WMC	Object Oriented Metric	To calculate the total complexity of a class by adding all methods' complexities of that class.	Only considers complexities arising because of inheritance ignoring encapsulation and polymorphism features
LCOM	Object Oriented Metric	To calculate the amount of cohesiveness in the methods of a class	Not appropriate for the classes which access their data internally using their attributes
DIT	Object Oriented Metric	To measure the inheritance factor among super and sub classes	Ambiguous results in case of multiple inheritance
RFC	Object Oriented Metric	The count of available methods in a class	Considers the first level of calls outside of the class instead of calling the entire call tree
Design Metric	Product Metrics	To measure product design such as reusability, architecture independence, information hiding and so on	Higher Level of Language Paradox
Performance Metric	Product Metrics	To measure software performance such as reliability, response time and throughput	Highly dependent on accuracy of documents for example SRS, SDD for correct results
Maintenance Metric	Process Metrics	To measure impact of changes introduced during maintenance phase of software.	Too expensive to use as cost sometimes exceeds the development cost, extensive amount of time is required to understand the code and implement the metric
Development	Process Metrics	To measure the development phases of SDLC such as requirements, design, coding and testing	Heavily dependent on technical staff and working environment which are difficult to predict and measure
Effort	Project Resource Metrics	To measure efficiency of Project Team	Technical skills of individual cannot be measured which are directly related to productivity
Schedule	Project Resource Metrics	To measure the schedule adherence when development process is running	Improper estimation of technical complexities and unplanned dependencies on tools and people leads to wrong results
Cost	Project Resource Metric	To measure the cost of software development	Attributes values in estimation model are dependent on historical and empirical data making cost estimation difficult
Productivity	Project Resource Metric	To measure efforts required according to size of the project	Efforts are directly proportional to technical expertise of employees which is difficult to measure

IV. Conclusion

This paper has presented a survey on the existing set of metrics classified according to their design and project-based needs. We observed that a lot of metrics have been proposed to measure software quality attributes such as size, design, complexity and efficiency but every metric suffers from some limitations. From this in-depth study of existing traditional, design based and object-oriented metrics, we can conclude that as object-oriented metrics are totally distinct from other existing software metrics so it is unfortunate that no side-by-side comparisons can be drawn between object-oriented projects and other traditional procedural projects by means of object-oriented metrics available currently.

These metrics are lacking many features for example existing object-oriented metrics are not as efficiently applicable on software maintenance and conversion projects as they are applied for software development projects. These metrics do not consider studies outside of the OO paradigm and not supported by software estimating tools.

The Research study has focused on the basic fact that despite presence of many metrics related to software metrics, there should be proper categorization of specific metrics that should be used on specific types of projects. Since it is not necessary that all metrics are suitable for all types of projects, the metrics should be properly studied and must be applied for proper software project.

Conflict of Interest

The submitted work was carried out with no conflict of interest. Hence, the authors declare no conflict of interest.

Author Contributions

Ms. Shweta Sharma collected and analyzed the data from different research articles and wrote the paper. Dr. S. Srinivasan provided guidance in the preparation of the paper and has checked for plagiarism and grammar.

References

[1] E. E. Mills, "Software metrics," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst., 1988.

[2] H. K. Stephen, *Metrics and Models in Software*, Boston: Addison Wesley, 2002.

[3] G. Keshavarz, N. Modiri, and M. Pedram, "Metric for early measurement of software complexity," *Interfaces*, vol. 5, no. 10, p. 15, 2011.

[4] N. E. Fenton and M. Neil, "Software metrics: Successes, failures and new directions," *J. Syst. Softw.*, vol. 47, no. 2–3, pp. 149–157, 1999.

[5] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Trans. Softw. Eng.*, vol. 20, no. 3, pp. 199–206, 1994.

[6] M. Scotto, A. Sillitti, G. Succi, and T. Vernazza, "A relational approach to software metrics," in *Proc. the 2004 ACM Symposium on Applied Computing*, 2004, pp. 1536–1540.

[7] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, 2005.

[8] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Trans. Softw. Eng.*, vol. 14, no. 9, pp. 1357–1365, 1988.

[9] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "Effectiveness of traditional software metrics for object-oriented systems," in *Proc. the Twenty-Fifth Hawaii International Conference on System Sciences*, 1992, vol. 4, pp. 359–368.

[10] C. Ebert, J. Cain, G. Antoniol, S. Counsell, and P. Laplante, "Cyclomatic complexity," *IEEE Softw.*, vol. 33, no. 6, pp. 27–29, 2016.

[11] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE Trans. Softw. Eng.*, vol. 17, no. 12, p. 1284, 1991.

[12] M. M. S. Sarwar, S. Shahzad, and I. Ahmad, "Cyclomatic complexity: The nesting problem," in *Proc. Eighth International Conference on Digital Information Management (ICDIM 2013)*, 2013, pp. 274–279.

[13] A. Madi, O. K. Zein, and S. Kadry, "On the improvement of cyclomatic complexity metric," *Int. J. Softw. Eng. Its Appl.*, vol. 7, no. 2, pp. 67–82, 2013.

[14] T. Mccabe, "Cyclomatic complexity and the year 2000," *IEEE Softw.*, vol. 13, no. 3, pp. 115–117, 1996.

[15] C. Ikerionwu, "Cyclomatic complexity as a software metric.," *Int. J. Acad. Res.*, vol. 2, no. 3, 2010.

[16] L. Rosenberg, *Applying and Interpreting Object Oriented Metrics*, 1998.

[17] G. Seront, M. Lopez, V. Paulus, and N. Habra, "On the relationship between cyclomatic complexity and the degree of object orientation," in *Proc. of QAOOSE Workshop, ECOOP, Glasgow*, 2005, pp. 109–117.

[18] K. Bhatt, V. Tarey, P. Patel, K. B. Mits, and D. Ujjain, "Analysis of source lines of code (SLOC) metric," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 5, pp. 150–154, 2012.

[19] S. Yu and S. Zhou, "A survey on metric of software complexity," in *Proc. 2010 2nd IEEE International Conference on Information Management and Engineering*, 2010, pp. 352–356.

[20] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical study of object-oriented metrics.," *J. Object Technol.*, vol. 5, no. 8, pp. 149–173, 2006.

[21] N. I. Churcher, M. J. Shepperd, S. Chidamber, and C. F. Kemerer, "Comments on A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 21, no. 3, pp. 263–265, 1995.

[22] M. Bruntink and A. V. Deursen, "Predicting class testability using object-oriented metrics," in *Proc. Fourth IEEE International Workshop on Source Code Analysis and Manipulation*, 2004, pp. 136–145.

[23] G. Alkadi and D. L. Carver, "Application of metrics to object-oriented designs," in *Proc. 1998 IEEE Aerospace Conference (Cat. No. 98TH8339)*, 1998, vol. 4, pp. 159–163.

[24] F. B. Abreu and R. Carapuça, "Candidate metrics for object-oriented software within a taxonomy framework," *J. Syst. Softw.*, vol. 26, no. 1, pp. 87–96, 1994.

[25] I. Brooks, *Object-Oriented Metrics Collection and Evaluation with a Software Process*, 1993.

[26] B. Újházi, R. Ferenc, D. Poshyvanyk, and T. Gyimóthy, "New conceptual coupling and cohesion metrics for object-oriented systems," in *Proc. 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, 2010, pp. 33–42.

[27] C. G. Desai, "Object oriented design metrics, frameworks and quality models," *Inf. Sci. Technol.*, vol. 3, no. 2, p. 66, 2014.

[28] P. Pocatilu, "IT Project management metrics," *Inform. Econ. J.*, no. 4, p. 44, 2007.

[29] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin-Cummings Publishing Co., Inc., 1986.

[30] L. A. Laranjeira, "Software size estimation of object-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 16, no. 5, pp. 510–522, 1990.

[31] A. B. Nassif, L. F. Capretz, and D. Ho, "Software estimation in the early stages of the software life cycle," in *Proc. International Conference on Emerging Trends in Computer Science, Communication and Information Technology*, 2010, pp. 5–13.

[32] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig, "Software complexity and maintenance costs," *Commun. ACM*, vol. 36, no. 11, pp. 81–95, 1993.

[33] V. Gupta, "Object-oriented static and dynamic software metrics for design and complexity," PhD dissertation, University of Kurukshetra, India, 2010.

[34] G. Kaur and K. Bahl, "Software reliability, metrics, reliability improvement using agile process," *Int. J. Innov. Sci. Eng. Technol.*, vol. 1, no. 3, pp. 143–147, 2014.

[35] M.-C. Lee and T. Chang, "Software measurement and software metrics in software quality," *Int. J. Softw. Eng. Its Appl.*, vol. 7, no. 4, pp. 15–34, 2013.

[36] A. Tarhan and O. Demirors, "Assessment of software process and metrics to support quantitative understanding," in *Software Process and Product Measurement*, Springer, 2007, pp. 102–113.

[37] S. W. Ali, Q. A. Ahmed, and I. Shafi, "Process to enhance the quality of software requirement specification document," in *Proc. 2018 International Conference on Engineering and Emerging Technologies (ICEET)*, 2018, pp. 1–7.

[38] J. Cleland-Huang, C. K. Chang, H. Kim, and A. Balakrishnan, "Requirements-based dynamic metrics in object-oriented systems," in

Proc. Fifth IEEE International Symposium on Requirements Engineering, 2001, pp. 212–219.
[39] L. Briand, S. Morasca, and V. R. Basili, "Defining and validating high-level design metrics," 1994.
[40] S. K. Singh and A. Singh, *Software Testing*, Vandana Publications, 2012.
[41] A. Rainer and T. Hall, "A quantitative and qualitative analysis of factors affecting software processes," *J. Syst. Softw.*, vol. 66, no. 1, pp. 7–21, 2003.
[42] N. F. Schneidewind, "The state of software maintenance," *IEEE Trans. Softw. Eng.*, no. 3, pp. 303–310, 1987.
[43] R. Hall and S. Lineham, "Using metrics to improve software maintenance," *BT Technol. J.*, vol. 15, no. 3, pp. 123–129, 1997.
[44] E. A. Rajavat, "An impact-based analysis of software reengineering risk in quality perspective of legacy system," *Int. J. Comput. Appl.*, vol. 975, p. 8887, 2011.
[45] A. Desantis, A. Pleskus, and G. D. Howell, *Maintenance*, 2015.
[46] R. Solingen and P. Stalenhoef, *Effort Measurement of Support to Software Products*, 1997.
[47] T. K. Abdel-Hamid and S. E. Madnick, *Impact of Schedule Estimation on Software Project Behavior*, 1985.
[48] M. Aikhatib and S. Altarazi, "A customized root cause analysis approach for cost overruns and schedule slippage in paper-machine-building projects," *Manag. Prod. Eng. Rev.*, vol. 10, pp. 83–92, 2019.
[49] R. I. Carr, "Cost, schedule, and time variances and integration," *J. Constr. Eng. Manag.*, vol. 119, no. 2, pp. 245–265, 1993.
[50] F. J. Heemstra, "Software cost estimation," *Inf. Softw. Technol.*, vol. 34, no. 10, pp. 627–639, 1992.
[51] H. Leung and Z. Fan, "Software cost estimation," in *Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies*, World Scientific, 2002, pp. 307–324.
[52] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, no. 5, pp. 416–429, 1987.
[53] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2006.
[54] B. Kitchenham and E. Mendes, "Software productivity measurement using multiple size measures," *IEEE Trans. Softw. Eng.*, vol. 30, no. 12, pp. 1023–1035, 2004.
[55] J. S. Collofello, S. N. Woodfield, and N. E. Gibbs, "Software productivity measurement," in *Proc. the May 16-19, 1983, National Computer Conference*, 1983, pp. 757–762.

Shweta Sharma is working as an assistant professor in Computer Science Department in Government College for Girls, Sector-14, Gurugram. She has 13 years of teaching experience. She is pursuing the Ph.D (computer science) from Mewar University, Rajasthan. She did BCA from University College Kurukshetra (KUK) and MCA from Department of Computer Science and App Applications from Maharshi Dayanand University, Rohtak.

She has also done the MPhil (computer science) from Ch. Devilal University, Sirsa. She has also done the B.Ed in computer science and English from Maharshi Dayanand University. Her areas of research are software engineering, data base management system, computer networks and cybersecurity.

S. Srinivasan is currently working as a head of the Department of Computer Science and Application in PDM University, Bahadurgarh. He has a vast experience of nearly 40 years including 22 years in teaching and 18 years in Industry. He presented research papers in various National and International conferences. He was a research advisor of Bharathidasan University, Trichy, Tamilnadu. He has produced twelve Ph.Ds and guiding six research students.

His current research field is the area of artificial intelligent especially in multi-agent system technology and its applications.