Software Design Metric

Aashish Mahajan

**Abstract**— There is a great need to measure the software for the degree to which they are prone to error or property. There should be a paradigm that one could look up to, to measure the efficacy and efficiency of the system so designed or coded and to measure the extent to which they are prone to the bugs and error. Its important to have some measurement tool that one could use to know the degree to which the system so designed on the basis of user requirement, meet the requirement and is able to deliver what is expected from it, in all possible scenarios. How the system could possibly react in case of expected and unexpected input, how the system responds in such scenario. Such Metric helps us way the future performance and proneness to exception. The main objective for such metric is to obtain a quantifiable measurement, objective- that could help one to design the application schedule, plan the budget, estimate the most important factor- “cost”, quality, performance and debugging.

**Index Terms**— Software Metric, Software development life cycle

——————————  ——————————

# 1 Introduction

T

here has been a great surge in development of software system that are object oriented with basic feature dependent on properties namely as Inheritance, Polymorphism, Class, Coupling, Abstraction, Information hiding. [1] This surge has led researcher and designer to propose many metric to weigh the software system hence designed on various parameter as application schedule, plan the budget, estimate the most important factor- “cost”, quality, performance and debugging.[2] The metric that one could use in the past were limited to weigh the aspects such as requirement analysis, design efficiency implementation of documents: all are important stages of software system development. (SDLC- Software Development Life cycle).

[3]Software System failure could occur mainly because of 2 reasons: 1. Error in code logic. 2. Exception Failure.   
Error in code logic: it could be result of improper code implementation, a simple scenario could be ex: The interest calculated on the amount available in account of person.[3] A slight miscalculation could result in low efficiency of the system hence designed. Whereas on the other hand Exception Failure occurs when the software designed is prevented by unexpected circumstances to deliver the service that it promises to deliver. Fact is that Exception failures can account for up to two-thirds of system crashes .Also the Standish Chaos Report also estimated that the annual cost of cancelled projects was $55 billion .

# 2 What Are Software Metrics

S

Oftware metrics are: 1. Tools that if used help one to understand varying aspects of a code base and project progress in SDLC. 2. Apart from the functionality of testing the software for errors they could provide a wider range of information about the aspect of software systems such as: a. the cost associated to the project-development-maintenance –research and other typical cost associated with SDLC. b. Project scheduling –some related to product scheduling and some related to document scheduling. c. Quality and Code complexity of the software. It focuses on the inter-dependencies within the project and on other projects.

T

Erm **Measurement** [ 5] that would be referred here in the paper refers to an empirical, objective assignment of numbers, according to a rule derived from a model or theory, to attributes of objects or events with the intent of describing them.

These measurements are performed with a sole purpose of: 1. Facilitating private self-assessment and improvement amongst the software developing organization. 2. Evaluating project status (to facilitate management of the project or related projects)

3. Evaluating staff performance. 4. Informing others (e.g. potential customers) about the characteristics (such as development status or behavior). What are we trying to measure using the Software Metrics: A software metric is used to evaluate many aspect of software as well the development team.

W

Hat are we trying to measure using the Software Metrics:[6] A software metric is used to evaluate and measure many aspect of software system in consideration as well the development team. The parameter that are basically weighed in this process are: 1. **Skill:** using the process helps we analyze the skill set and efficiency at the task assigned for a particular person or a team in combined. 2. **Effectiveness**: it helps us identify the member which are good at task assigned for ex a person is not considered to be good tested if he/she finds the maximum number of bugs but considered to be good for the total number of bugs one is able to fix. 3. **Efficiency**: it evaluates work effort ratio. How much time a person takes to what amount of task. One should take minimum time to complete the task in hand. 4. **Productivity:**  it refers to how much the tester delivers per unit time. [7]The distinction that one can draw between efficiency and productivity is that efficiency refers to the way the person does the job whereas productivity refers to what she gets done. For example, a tester who works on a portion of the code that contains no defects can work through the tests efficiently but produce no bug reports. 5. Diligence: it refers to how carefully and hard the tester does the work assigned. 6. Courage: It could be summed too willing to attempt difficult and risky tasks; willing to honestly report findings that key stakeholders would prefer to see suppressed or unknown. 6. Credibility: it refers to the extent to which others trust the reports and commitments of this tester.

# 3 Some Common Software Metrics

S

Ome of the most common software measurements or software metrics are [8] 1. **Bugs per line of code**. 2. **Design structure quality index**. 3. **Control Flow Complexity**, 4. **Number of classes and interface.** 5. **Program execution time**. 6. **Maintainability index**. They all are quantitative measure of a degree to which the software developed or maintained possess the properties.

1. **Bugs Per Line of Code[9]:** It measure the number of total bugs found in total lines of code. One of the famous stock phrase –It cost 100 times as much to fix a defect after the software system has been developed in comparison to the cost incurred to fix a bug found during the early phase of development. According a detailed study, it has been found that the cost per defect comes out to be a major concern for software upgrade. There are a lot of hidden issues with the problem of cost per defect such as **a**. Cost per defect penalize the quality software. **b**. The cost related to defect per line ignores the fixed cost. Here the Fixed cost refers to the cost of testing, code inspection, static analysis and maintenance. **c**. As more of the bugs are found during the early stages of software development, the cost of fixing a defect at the end of process is virtual. **d**. Once the defects are fixed, the cost per defect cannot be weighed against the improved software quality. The savings done in form of reduction in overall development cost, improved quality are not computed to the cost of defect calculation. The Term in light Defect Density could be summed as average occurrence of programming faults per Lines of Code (LOC). Though it gives a high level view of the code quality but doesn’t do much more. Fault density on its own does not give rise to a pragmatic metric. It would cover minor issues as well as major security flaws in the code; all are treated the same way. Security of code cannot be judged accurately using defect density alone

**2. Design Structure Quality Index:** [10]Design Structure Quality Index also known as DSQI yet another type of ware metrics. It is an architectural design metric which is used to evaluate a software design structure and the efficiency of its modules (classes, interfaces, database, and interaction with the rest of system). It has been developed by the United States Air Force Systems Command. Generally the output one get after the software system is weighed, lies between number 0 and 1. According to the process metrics designer, the closer the value to 1, higher the effective quality of software. It is best used on a comparison basis, i.e. a new software process is compared with previous successful projects.

***The basic calculation formula used in DSQ is as follow:***

***S1*** = The total number of modules defined in the Program Architecture.

**S*2***= The number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere.

**S*3***=  The number of modules whose correct function depends on prior processing.

**S*4*** =  The number of database items (includes data objects and all attributes that define objects).

**S*5*** =  The total number of unique database items.

**S*6*** =  The number of database segments (different records or individual objects).

**S*7***=  The number of modules with a single entry and exit (exception processing is not considered to be a multiple exit). Once values ***s1*** through ***s7*** are determined for a software program under review, calculating the following intermediate values becomes really easy. **Program structure**: D1, where D1 is defined as follows: If the architectural design was developed using a distinct method e.g., data flow-oriented design or object-oriented design, then D1 has a value =1, otherwise D1 has a value= 0.   
Module independence: D2 = 1 - **(s2/s1**), Modules not dependent on prior processing: D3 = 1 - (**s3/s1**),   
Database Size: D4=1-(**s5/s4**), Database compartmentalization: D5 = 1 - (**s6/s4**)   
Module entrance/exit characteristic: D6 = 1 - (**s7/s1**). All the above values help to calculate the DSQI : DSQI = SUM(***wi***Di)

Where i = 1 to 6, ***wi*** is the relative weight of the importance of each of the intermediate values, and S ***wi*** = 1 (if all Di are weighted equally, then ***wi*** = 0.167). **a**. Value of DSQI for past software designs can be determined from documents and compared to the value of DSQI software design that is currently under development. **b**. If the DSQI is significantly lower than average, further design work and review are indicated.

While loop

3. **Control Flow Complexity (McCabe Metric):[11]** This software metric is also known as McCabe’s metrics after the name of McCabe Software. Inc. and is based on a control flow representation of the program. It uses a program graph that is a diagram used to depict control flow within the software, it shows which module works on the processed inputs, which module works independently and which process the information forward in short the inter dependency of flow of control. In the diagram, nodes basically represent processing tasks (one or more code statements). Edges basically represent control flow between nodes. The basic principle on which McCabe’s Complexity Metric works is that it counts the number of distinct paths through a block of code. It takes its name from counting the number of cycles in the program flow control graph. It has been concluded that a lower the value of count better is the software system; McCabe suggested using 10 as a threshold value, which once exceeded, a module should be broken into simpler –sub smaller units. This Software metric measures the complexity in terms of total number of edges in control flow graph for as software system. The following Fig. 1 shows flow graph notation

**4. Number of classes and interface:** [12]It is a software metric that depends upon the number of classes and interface in a Design Structure analysis of the software project in hand. Assumption made in this is that all the effort in developing a class is determined by the number of method a class has, so the overall complexity of the system could be measured as a function of complexity of its method. So it brings the concept of weighted method per class aka WMC which could be calculated as: let c be a class with M1, M2…. Mn methods, and Ci be the complexity of method Mi then WMC could be said to equal =>. Most classes tend to have smaller number of method which makes them simple and specific for a function or abstraction process. This software Metric brings into light the concept of number of child classes. It shows how complex a software system could get with multilevel inheritance, it shows the extent of influence of a class on other elements of design. A higher value suggest that a large number of subclasses are involved in the code reusability. A lower value is certainly required from this, as the idea is to reduce the coupling and interdependency and increase abstraction. Then the concept of coupling is brought into light, it refers to a scenario when a method of class C1 is used by C2. Hence C2 is said to be coupled with the class C1, any change in the definition of C1 could cause a change in definition of class C2. It certainly leads to interdependency and software metrics are used with an aim to reduce interdependency and simplify the design. An approach to abstraction should be kept in mind while designing the system.

**5. Program Execution Time:[14]** It is a software metric that depends upon the time taken by the software process to execute a user query. It could be a simple data add transaction or it could be as complicated as inter related data retrieval query by a bank. Execution Time matric runs a clock what we call “Stopwatch” that keeps a variable START that’s takes a note of time interval when the execution begins and a variable STOP that keeps a track of clock when the execution completes, so the Program Execution time could be calculated as Elapsed Time = STOP-START. The clock used in here is the processor clock. This Execution time includes everything such as Background OS task, virtual to physical page mapping, random cache mapping and replacement, variable system load. The issue that come in picture are debugging of the program it could be pre compiled or at run time. Run time could cause an issue for logic error or array bounds check. It is really hard to discover some bugs until the system is completely tested in a live environment, It’s important to keep them in notice as in the end, end user could encounter a problem.is the time taken to handle the exceptions.[15] Performance for a system could be actually weighed relative to the workload description, so it is important to select a particular workload benchmark, beyond which the measured performance is of no applicable comparison. It could become a tough situation when the performance comparison for same operation is done in two different environment or coding language. A desirable performance model should incorporate the effect of all not limited to code feature and not including the inaccessible architecture feature. The performance models should calculates PC value which would reflect the (possible) behavior of a programmer which is PC reflects the performance transparency and complexity of performance control the programmer will experience. We could say that Goodness of fit measures for modeling measurements completely depends on the Sum of Squared Errors (SSE). To consider a scenario, let us consider a system Si with the performance Pij of a set of n codes Cj is measured. Also Different measurements j might also be obtained with the same code executed with different problem parameters such as problem sizes and resource parameters such as accessibility, thread synchronization, concurrency levels etc. A performance model **Mkl** is used to predict performance **Mikl** for the same set of experiments. The basic set of operations and variable with dimensions are defined as below

|  |  |
| --- | --- |
| **Operation** | **Dimension\*** |
| Initial data: Pj, Mj | [ops/sec] |
| Transform Pj, Mj to ideal flat metric | [ops/cycle] |
| Log-transformation:  P'j=log(Pi), M'j=log(Mi) | [log(ops/cycle)] |
| **Basic Calculations for the Exceutive time software metric** | |
| Formula | [log(ops/cycle)] |
| Formula | [(log(ops/cycle))2] |
| Formula | [(log(ops/cycle))2] |
| Formula | [(log(ops/cycle))2] |
| **Back-Transformations Exceutive time software metric** | |
| Formula | [ops/cycle] |
| Formula | [ops/cycle] |
| Formula | [] |
| Back-transformation of P¯;PCa to original scale and metrics | [ops/sec] |

D= { n_1 \over 2  } \times { N_2 \over n_2 } \,

This could be simplified to give the PC and measurement of the execution time matric for the system in consideration. Lesser the value of time of execution time, better is the performance and control flow within the system, it simplifies that the interaction within the system is simplified to an extent that it help reduce dependencies within the system.

**6. Maintainability Index:** [16] Maintainability Index is yet another type of software metric which measures how easy is maintaining (easy to support and change and update) the source code for a software process. Maintainability Index MI is a single number value for estimating the relative maintainability of software system in consideration. The maintainability index is calculated as a factored formula consisting of variables as 1. Lines of code, 2. Cyclomatic Complexity, 3. Halstead volume. [17]1. **Line of Code**: could be explained to be approximate number of lines in the code. The count depends on the IL code and is therefore it is not an exact number of lines in the source code file. A very high count might indicate that a type or method is trying to do too much work and should be split up. **2. Cyclomatic Complexity:** it is the Measure of the structural complexity of the code. It is a variable achieved by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable. It might also indicate that the type or method might be hard to maintain. 3. **Halstead Volume**[18]: Halstead made the observation that metrics of the software should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. These metrics are therefore computed statically from the code. *n*1 = the number of distinct operators, *n*2 = the number of distinct operands, *N*1 = the total number of operators, *N*2 = the total number of operands. From these numbers, five measures can be calculated: Program length: N = N1 + N2  \,, Program vocabulary: n = n1 + n2 \,, Volume: ,Difficulty: , Effort: E= D * V \,. It is used in several automated software metric tools, namely C#, Microsoft visual studio 2010. The metric computes the weigh on a scale of 0 to 100, which is classified in range: 20-100- is green region and score makes the software maintainability good, with a score 10- 19 is yellow region, the software has a moderate maintainability and for any score less than 10 makes the software code to be bad and classified as red region, its prone to unstructured, unformatted and undocumented form, maintaining such code is really difficult. There are two variants of Maintainability Index namely one that contains code comments (MI) and other one that does not contain comments (MIwoc). The tools are efficient enough to calculate Maintainability Index for functions (and class/struct definitions), for files and for the whole software. Actually there are three measures:

1. **MIwoc**: Maintainability Index without comments. 2. **MIcw**: Maintainability Index comment weight.

3. **MI**: Maintainability Index = MIwoc + MIcw

**The general formulae for calculating MI is the following:**

**MIwoc** = 171 - 5.2 \* ln(aveV) -0.23 \* aveG -16.2 \* ln(aveLOC)

**MIcw** = 50 \* sin(sqrt(2.4 \* perCM))

**MI** = MIwoc + MIcw

Where **aveV** = average Halstead Volume per module,

**aveG** = average extended cyclomatic complexity per module   
**aveLOC** = average count of lines per module,

**perCM** = average percent of lines of comments per Module.

There are couple of **Issues** that arise in the case of the Maintainability index software metric[19]. 1. There is no clear explanation for the derived formula for the MI software metric. The only explanation that is given is for the formula dependency on the lines of code, but in that scenario, using line of code matric is much better option, it has a vague explanation for the dependency on the halstead, Cyclomatic complexity and line of code. 2. The formula calculates the average per file example is Cyclomatic complexity. Calculating the value of MI over an average file would mask the effect in larger picture.

# 4 Some Common Issue Related Software Metrics

S

Ome of the most common issues related to software measurements or software metrics occur because the task in hand i.e. to evaluate a software for its efficiency, error proneness, maintainability, upgrade etc. is really cumbersome. It is difficult to manage the software and human aspect of the software measurement. Some common issues[21] that hinder the way are: 1. **Lack of management Commitment**: It has been observed that though the management implements the software metric, they lack in deploying the service into practice. Engineer do not collect and document the data as management lacks the sense of importance of the data. 2. **Collecting data that is not used:**  It has been observed that Data collected during the measurement process is not up to the mark or satisfactory. Data collected should be such that it can be used to enhance the process, project, or product. It is important because collecting incorrect data or inefficient data could results in wrong decision making, which in turn leads to deviation from the software development plan. 3. **Imprecise metrics definitions:** Vague or ambiguous metrics definition or knowledge to management could lead to big misinterpretation. It is really important for the management to be educated about the usage and implications of the metrics that they could implement on the system in hand. For example, some software engineers may interpret a software feature or a code block as unnecessary while some software engineers may not. 4. **Lack of communication and training:** It has been evidently seen that inadequate training and lack of communication within management results in poor understanding of software metrics and measurement of unreliable data. In addition, communicating metrics data in an ineffective manner results in misinterpretation of the data.

5. **Software Metric to be used**: It has been seen that a lot of time, the management could face a problem of choosing an effective and efficient metric for the system. In this case it becomes very important to weigh the system in hand in different number.

# 4 Conclusion

T

O conclude I could certainly find and point the efficiency and quality of all the software metrics I have discussed about in the paper. The most efficient software metric according to me is **Number of Classes and Interface**. It is a very efficient matrix. Maximum reason of upgrade issue and bugs arise when there is a lot of inter dependency in the classes, where change in on class affects the functionality of the sub class or child classes. It hits and works on the fact that classes should be abstract if there functionality is supposed to be used in many child classes. It supports the understanding of having less or function specific classes,[22] so that change in Abstract class doesn’t change the specific function of the child class. It helps to reduce the interdependency in the classes. The interface implementation could also be minimized by making them abstract. It could help to simplify the code structure to a great extent. After the Number of Classes and Interface, the second best software metric according to me is Design structure quality index aka DSQI. Program execution time and Bugs per line of code are the least efficient usage wise and scope wise. They are limited to the average value estimation when they might be forgetting the issue of bugs in big picture. Program execution time take the measurement in terms of the time taken for execution for a process or software in whole, but avoid the issue of time delay to access the resources. Software in itself could be working efficiently but inefficient access could kill the system performance in whole.

**References**

http://www.testingeducation.org/a/metrics2004.pdfReferences

1. http://sunnyday.mit.edu/16.355/metrics.pdf
2. <http://www.jot.fm/issues/issue_2007_01/article4.pdf>
3. http://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/boughtonalexandra.pdf
4. <http://www.jot.fm/issues/issue_2007_01/article4.pdf>
5. <http://swreflections.blogspot.com/2012/05/software-development-metrics-that.html>
6. <http://ecomputernotes.com/software-engineering/issues-in-software-metrics>
7. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&uact=8&ved=0CFMQFjAG&url=https%3A%2F%2Fwww.cs.purdue.edu%2Fhomes%2Fapm%2Fcourses%2FBITSC461-fall03%2Fuml-slides%2Fdesign-metrics.ppt&ei=S2QQVfabNc21oQTO34GwCw&usg=AFQjCNHYoybXk9w3ayI4EGG4OH83gqjYEQ&sig2=v9Yu128mw5C4NzgHoORrww&bvm=bv.89184060,d.cGU>
8. <http://en.wikipedia.org/wiki/Software_metric>
9. <http://en.wikipedia.org/wiki/Software_bug>
10. <http://logicalprogram.blogspot.com/p/dsqi.html>
11. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.1040&rep=rep1&type=pdf>
12. <http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CB8QFjAA&url=http%3A%2F%2Fwww.mccabe.com%2Fppt%2FSoftwareQualityMetricsToIdentifyRisk.ppt&ei=TgkWVcf-FpPjoASb1IGYBQ&usg=AFQjCNEDKqkz__6SUgXuYGKq7jwRp-BHfQ&sig2=nftUxaLaHN8NM6W2LW42Vw&bvm=bv.89381419,d.cGU>
13. <http://en.wikipedia.org/wiki/Software_package_metrics>
14. <http://sunnyday.mit.edu/16.355/metrics.pdf>
15. <http://www.ctwatch.org/quarterly/articles/2006/11/performance-complexity-an-execution-time-metric-to-characterize-the-transparency-and-complexity-of-performance/>
16. http://www.virtualmachinery.com/sidebar4.htm
17. http://www.projectcodemeter.com/cost\_estimation/help/GL\_maintainability.htm
18. https://msdn.microsoft.com/en-us/library/bb385914.aspx
19. <http://en.wikipedia.org/wiki/Halstead_complexity_measures>
20. <http://ecomputernotes.com/software-engineering/issues-in-software-metrics>
21. <http://www.testingeducation.org/a/metrics2004.pdf>
22. http://www.testingeducation.org/a/metrics2004.pdf