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 upto, to measure the efficacy and efficiency of the system so desgined or coded and to measure the extent to which they are prone to the bugs and error. Its imporatant to have some measurement tool that one could use to know the degree to which the system so desgied on the basis of user requirement, meet the requirement and is able to deliver what is expected from it, in all possible scearios. 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 proness to exception. The main objective for such metric is to obtain a quaintfiable 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**— Sofware 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, Absraction, Information hiding. 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. 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).

Software System failure could occure mainly because of 2 reasons: 1. Error in code logic. 2. Exception Failure.   
Error in code logic: it could be result of imporper code implemtation, a simple scenario could be ex: The interest calculated on the amount available in account of person. A slight miscalculation could result in low efficiency of the system hence designed. Where as on the other hand Exception Failure occours 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 [1] [Cristian95]. Also the Standish Chaos Report also estimated that the annual cost of cancelled projects was $55 billion [2].

# 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-dependecies within the project and on other projects.

# 3 Some Common Software Metrics

S

Ome of the most common software meausrments or software metrics are 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 posses the properties.

1. **Bugs Per Line of Code:** It measure the number of total bugs found in total lines of code. One of the famous stck phrase –It cost 100 times as much to fix a defect after the softwar 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 its has been found that the cost per defect comes out to be a major concer 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 realted to defect per line ignores the fixed cost. Here the Fixed cost refers to the cost of testing, code inspection, static analysis and maintaince. **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 can not be judged accurately using defect density alone

**2. Design Structure Quality Index:** Design Structire Quality Index also known as DSQI  yet another stype 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.

3. **Control Flow Complexity (McCabe Metric):** 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 basicly represent processing tasks (one or more code statements). Edges basicly 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 exceded, 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

While loop

**4. Number of classes and interface:** 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 abastraction. 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 simplfy the design. An approach to abstraction should be kept in mind while designing the system.

**5. Program Execution Time:** It is a software metric that depends upon the time taken by the software process to execute a user query. It 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 varraible STOP that keeps a track of clock when the execution completes, so the Program Exceution 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, Its important to keep them in notice as in the end, end user could encounter a problem.is the time taken to handle the exceptions. 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 tought situation when the performance comparison for same operation is done in two differnet environment or coding language. A desiarable performance model should incorporate the effect of all not limited to code feature and not including the inaccesbible architecture feature. The performance models should calculates PC value which would reflect the (possible) behavior of a programmer,that 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 accesability, thread sysnchornization, 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 varraible 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 | [] |
| **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] |

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 lnteraction within the system is simplified to an extent that it help reduce dependencies within the system.

V= N \times \log_2 n   \, **6. Maintainability Index:** 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 varriables as 1. Lines of code, 2. Cyclomatic Complexity, 3. Halstead volume. 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 varraible 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**: 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: D= { n_1 \over 2  } \times { N_2 \over n_2 } \,, 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 instructured, 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

## 2.2 Final Stage

When you submit your final version, after your paper has been accepted, print it in two-column format, including figures and tables. Send it as a pdf file.

Also send a file with complete contact information for all authors. Include full mailing addresses, telephone numbers, fax numbers, and e-mail addresses. This information will be used to send each author a complimentary copy of the journal in which the paper appears. In addition, designate one author as the "corresponding author." This is the author to whom proofs of the paper will be sent. Proofs are sent to the corresponding author only.

## 2.3 Figures

All tables and figures will be processed as images. The figures that appear in the final version you submit of your paper (pdf) will be used as the guide as to how you want your figures to look. If possible, also supply individual files for each figure and table, eps, ps, tif, or pdf are the preferred file formats. Images that will fit into a single column should be sized no greater than 3.4” wide, and double column images should be no greater than 7.0” wide at 600dpi.

## 2.4 Electronic Image Files

Please name these files according to their figure or table number such as, Fig1.eps, Table4.ps, Fig8.pdf. If your paper is to be printed in color, then all color images should be in CMYK format and not RGB.

No matter how you convert your images, it is a good idea to print the files to make sure nothing was lost in the conversion.

If you have a question about formatting your paper, or a suggestion on improving these instructions, please contact your publication associate.

# 3 Helpful Hints

## 3.1 Figures and Tables

Because IEEE CS will do the final formatting of your paper, you do not need to position figures and tables at the tops and bottoms of columns. Large figures and tables may span both columns. Place figure captions below the figures; place table titles above the tables. If your figure has two parts, for example, include the labels "(a)" and "(b)" as part of the artwork. Please verify that figures and tables that you mention in the text actually exist. **Please do not include captions as part of the figures. Do not put captions in “text boxes” linked to the figures. Do not put borders around the outside of your figures.** Use the abbreviation "Fig." even at the beginning of a sentence. Do not abbreviate “Table.” Tables are numbered with Arabic numbers.

Color printing of figures is available for some CS transactions. Please check the Journals Author Center at http://www2.computer.org/portal/web/peerreviewjournals/author#color for a list of journals that print color figures.

## 3.2 References

Number citations consecutively in square brackets [1]. The sentence punctuation follows the brackets [2]. Multiple references [2], [3] are each numbered with separate brackets [1], [2], [3]. When citing a section in a book, please give the relevant page numbers [2]. In sentences, refer simply to the reference number, as in [3]. Do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] shows ... ." Type the reference list at the end of the paper using the “References” style. Do not make the reference list an end note.

Number footnotes separately in superscripts (Insert | Footnote).[[1]](#footnote-1) Place the actual footnote at the bottom of the column in which it is cited; do not put footnotes in the reference list (endnotes).

Note that the IEEE CS referencing style is distinctive. Please follow the reference styles for transactions found in the IEEE CS style guide, which is online at http://www2.computer.org/portal/web/publications/styleguide.

## 3.3 Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have already been defined in the abstract. Abbreviations such as IEEE, SI, ac, and dc do not have to be defined. Abbreviations that incorporate periods should not have spaces: write “C.N.R.S.,” not “C. N. R. S.” Do not use abbreviations in the title unless they are unavoidable (for example, “IEEE” in the title of this article).

## 3.4 Equations

Number equations consecutively with equation numbers in parentheses flush with the right margin, as in (1). First, use the equation editor to create the equation. Then, select the “Equation” markup style. Press the tab key and write the equation number in parentheses. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Use parentheses to avoid ambiguities in denominators. Punctuate equations when they are part of a sentence.

Be sure that the symbols in your equation have been defined before the equation appears or immediately following. Italicize symbols (*T* might refer to temperature, but T is the unit tesla). Refer to “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) is ... .”

## 3.5 Other Recommendations

Use one space after periods and colons. Hyphenate complex modifiers: “zero-field-cooled magnetization.” Avoid dangling participles, such as, “Using (1), the potential was calculated.” (It is not clear who or what used (1).) Write instead, “The potential was calculated by using (1),” or “Using (1), we calculated the potential.”

Use a zero before decimal points: “0.25,” not “.25.” Use “cm3,” not “cc.” Indicate sample dimensions as “0.1 cm × 0.2 cm,” not “0.1 × 0.2 cm2.” The abbreviation for “seconds” is “s,” not “sec.” Do not mix complete spellings and abbreviations of units: use “Wb/m2” or “webers per square meter,” not “webers/m2.”

A parenthetical statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.) In American English, periods and commas are within quotation marks, like “this period.” Other punctuation is “outside”! Avoid contractions; for example, write “do not” instead of “don't.” The serial comma is preferred: “A, B, and C” instead of “A, B and C.”

If you wish, you may write in the first person singular or plural and use the active voice (“I observed that ...” or “We observed that ...” instead of “It was observed that ...”). Remember to check spelling. If your native language is not English, please get a native English-speaking colleague to proofread your paper.

# Appendix

Appendices, if present, will appear online as supplemental material. In the event multiple appendices are required, they will be labeled “Appendix A,” “Appendix B, “ etc.

Acknowledgments

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write ”S.B.A. thanks ... .” Put sponsor acknowledgments here. Write the acknowledgments in the third person.

# References

1. D.S. Coming and O.G. Staadt, “Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection,” *IEEE Trans. Visualization and Computer Graphics*, vol. 14,  no. 1,  pp. 1-12,  Jan/Feb  2008, doi:10.1109/TVCG.2007.70405.
2. J. Clerk Maxwell, *A Treatise on Electricity and Magnetism,* 3rd ed., vol. 2, pp. 68-73. Oxford: Clarendon, 1892.
3. I.S. Jacobs and C.P. Bean, “Fine Particles, Thin Films and Exchange Anisotropy,” *Magnetism,* vol. III, G.T. Rado and H. Suhl, eds., pp. 271-350, Academic Press, 1963.
4. R. Nicole, “Title of Paper,” *J. Name Stand. Abbrev.,* submitted for publication.
5. C.J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Col., personal communication, 1992.
6. H. Goto, Y. Hasegawa, and M. Tanaka, “Efficient Scheduling Focusing on the Duality of MPL Representation,” *Proc. IEEE Symp. Computational Intelligence in Scheduling* *(SCIS ’07)*, pp. 57-64, Apr. 2007, doi:10.1109/SCIS.2007.367670.
7. M. Young, *The Technical Writer's Handbook*: Univ. Science, 1989.
8. J.M.P. Martínez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, “Integrating Data Warehouses with Web Data: A Survey,” *IEEE Trans. Knowledge and Data Eng.*, doi:10.1109/TKDE.2007.190746.

**First A. Author** and the other authors may include biographies at the end of regular papers. Biographies are not included in short papers. Biographies should include education, employment, and memberships in professional societies.

1. It is recommended that footnotes be avoided (except for the unnumbered footnote with the receipt date on the first page). Instead, try to integrate the footnote information into the text. [↑](#footnote-ref-1)