Research Paper Information Technology

# A Comparative Survey of Software Quality Metrics

* Kirti Mathur ** Amber Jain

* International Institute of Professional Studies, D. A. University, Indore

** International Institute of Professional Studies, D. A. University, Indore

**ABSTRACT**

*Even though software problems are numerous (e.g. cancellation, cost overruns, schedule overruns, litigation, low levels of user satisfaction, high maintenance cost), still software metrics are not commonly used in most software companies. This paper will give a comprehensive survey of different software quality metrics (with empirical proofs) so that software organizations can integrate software metrics to estimate project attributes.*

**Keywords : metrics, software, quality, estimation**

## II. Introduction

A software metric is a quantitative measure of properties of a software or its specifications with the goal to obtain objective, reproducible and quantifiable measurements, which are useful in cost estimation, schedule and budget planning, optimal personnel task assignments, software performance optimization, software debugging and quality assurance testing of software projects [17].

The need for this survey arose from the fact that even though many software projects fail due to cost and schedule overruns, cancellations etc, still software metrics are not mainstay in most software companies. Quoting [1]: "... most companies still do not use systematic software measurement to assess software quality. One reason is that, in many companies, the software processes are poorly defined and controlled, and are not sufficiently mature to make use of measurements. Another reason is that there are no standards for metrics and hence there is limited tool support for data collection and analysis. Most companies will not be prepared to introduce measurement until these standards and tools are available."

In this paper, we'll analyze industry standard software metrics so that organizations will be able to draw useful conclusions about the quality of software.

## III. LITERATURE REVIEW

ISO standard for the evaluation of software quality using software metrics [2] classifies software quality in a structured set of characteristics as follows:


Figure-1: ISO/IEC 9126-1:2001 classification of Software Quality

**[2] further classifies software attributes into:**

- Internal attributes: which do not rely on software execution (static measure).
- External attributes are those metrics which are applicable to running software.

**Software metrics can be divided in two parts:**

| | |
|---|---|
| Product Metrics | · Measures the quantifiable attributes of a software<br>· Examples include size, complexity, code reuse |
| Process Metrics | · Measures of process of creating a software<br>· Examples include time spent, defects found and stability of requirements |
| Resource Metrics | · Measures of entities required by a process activity |

Table-1: Software metrics classification

**Fenton and Pfleeger suggested following classification of components of software measurement [1]:**

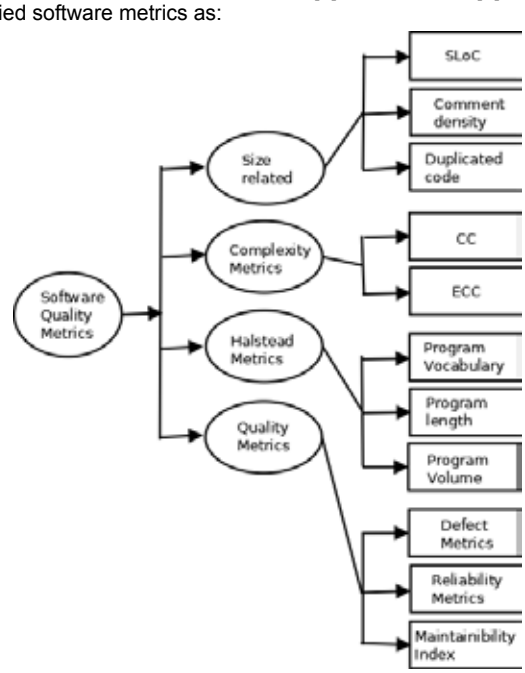
| Entities | Attributes | |
|---|---|---|
| | Internal | External |
| **Products** | | |
| Specifications | Size, reuse, modularity, redundancy, functionality, syntactic correctness... | Comprehensibility, maintainability, ... |
| Designs | Size, reuse, modularity, functionality, coupling, cohesiveness,... | Quality, complexity, maintainability, ... |
| Code | Size, reuse, modularity, functionality, coupling, algorithmic complexity, control flow structuredness... | Reliability, usability, maintainability, ... |
| Test data | Size, coverage level,... | Quality,... |
| ... | ... | ... |

| Processes | | |
|---|---|---|
| Constructing specification | Time, effort, number of requirement changes,... | Quality, cost, stability,... |
| Detailed design | Time, effort, number of specification faults found,... | Cost, cost effectiveness,... |
| Testing | Time, effort, number of coding faults found,... | Cost, cost effectiveness, stability,... |
| ... | ... | ... |

| Resources | | |
|---|---|---|
| Personnel | Age, price,... | Productivity, experience, intelligence,... |
| Teams | Size, communication level, structuredness,... | Productivity, quality,... |
| Software | Price, size,... | Usability, reliability,... |
| Hardware | Price, speed, memory size,... | Reliability,... |
| Offices | Size, temperature, light,... | Comfort, quality,... |
| ... | ... | ... |

Table-2: Components of Software Metrics

There are two kinds of metrics: those that can be evaluated in absolute terms and relative terms [4]. Sharma et al [3] classified software metrics as:



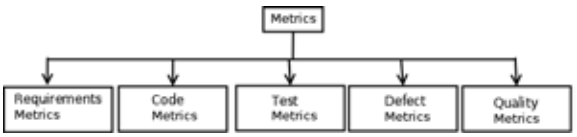




Figure-3: Software metrics classification (Adam Kolawa et al)

Some metrics suites were proposed by M. Subramanyam et al. [6] and they concluded that for the developers, designs metrics are very important to enhance the quality of software.

H. Lieu. et al. [7] have inferenced from their study that quality of software also plays an important role in terms of safety aspects and financial aspects. They bridged the gap between quality measurement and design of these metrics during development/re-development process of the software.

Standalone metrics provide a convenient, informative, at-a-glance snapshot. For example, LOC - a trivial measure that has little value in terms of assessing developer productivity - is an effective tool for understanding other metrics. Almost any code or testing metric that suffers a sharp spike or sudden drop requires a look at total LOC to be understandable.

Another example is Requirements metrics, which measure the number of requirements that have been implemented and tested, is maximized when measured over time.

Yet another example include test metrics (such as code coverage) that have now been widely discredited as a standalone measure. It is now well established that 100% code coverage is rarely a valid goal, so the code coverage as absolute values is useful only to meet a certain base coverage target.

Another valuable ratio compares changes in LOC to changes in the number of tests. In theory, the number of tests should change in direct proportion to the LOC.

**IV. Comparison of software metrics in use at at&t, nasa and open source projects**

NASA has built a repository of various metrics for both procedural and object oriented programming languages [8]. Another repository of metrics created by the NASA also includes metrics about errors and requirements [9]. Researchers of the NASA disseminated many of their results based on their studies on software metrics.

In another study, open source projects were compared against each other and one originally closed source system against its open source successor [11]. The Maintainability Index [10] metric was measured in time (for each successive version) to detect whether the system's source code quality is improving or deteriorating.

Another approach, usually referred to as Goal-Question-Metric (GQM) paradigm, measures what is needed (rather than what is convenient to measure). This approach provides a 3-steps framework:

- List major goal of development project.
- Derive from each goal the questions that must be answered to determine if the goals are being met.
- Decide what must be measured in order to be able to answer questions adequately.

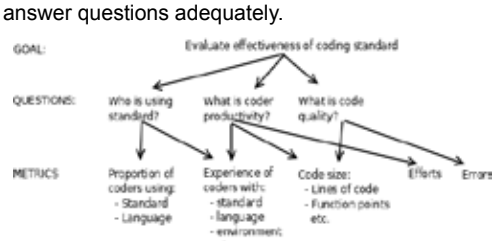



Figure-4: GQM framework

AT&T used GQM [20] to determine which metrics were appropriate for their inspection process:

| Goal | Questions | Metrics |
|---|---|---|
| Plan | How much does inspection process cost?<br>How much calender time does the inspection process takes? | Average effort per KLOC<br>Percentage of reinspections<br>Average effort per KLOC<br>Total KLOC inspected |
| Monitor and control | What is the quality of inspected software? | Average faults detected per KLOC |
| | What is the productivity of the inspection process? | Average effort per fault detected<br>Average inspection rate<br>Average preparation rate<br>Average lines of code inspected |
| | | Average Inspection rate<br>Average preparation rate |
| | To what degree did the staff conform to the procedures? | Average inspection rate<br>Average preparation rate<br>Average lines of code inspected<br>Percentage of reinspections |
| | What is the status of the inspection process? | Total KLOC inspected |
| Improve | How effective is the inspection process? | Defect removal efficincy<br>Average faults detected per KLOC<br>Average inspection rate<br>Average preparation rate<br>Average lines of code inspected |

Table-3: AT&T's GQM analysis

## IV. SURVEY OF SOFTWARE QUALITY METRICS

### 3.1 Source Lines of Code (SloC):

- Counting lines is used for estimating the amount of maintenance required
- Software Engineering Institute has published a set of recommendations [13] to standardize the counting.

### 3.2 Cyclomatic Complexity (CC):

Cyclomatic complexity, also known as CC, (proposed by McCabe in [12]) is used to evaluate the complexity of an algorithm in a method. This metric measures the number of independent paths through a software module.

A method with a low CC is generally better, although it may mean that decisions are deferred through message passing, not that the method is not complex.

Mc .Cabe describe as: V (G) =e-n+2
where, V(G)=CC of flow graph
G of method in which we interested
e=number of edges in G
n= number of nodes in G

[12] proposes an upper limit of 10 for CC because higher values would indicate less manageable and testable modules.

Although CC is widely used, many criticize its usage on it exists. Many experts claim that CC is based on poor theoretical foundations and an inadequate model of software development.

### 3.3 Extended Cyclomatic complexity (ECC):

CC measures the program complexity but fails to differentiate in the complexity of cases involving single condition in conditional statement. Myers suggested extended cyclomatic complexity that may be defined as: ECC=eV(G)=Pe+1

Where, Pe=number of predicate nodes in flow graph

G weighted by number of compound statements

### 3.4 Comment percentage:

Comment percentage or Comment Density includes both online (with code) and stand-alone comments and is used to evaluate the attributes of Understandability, Re-usability, and Maintainability.

Comment % = (Total number of comments) / (Total lines of code – blank lines) Empirical evidence has confirmed that a comment percentage of about 30% is most effective.

### 3.5 Duplicated code:

When code is duplicated it becomes more error prone and harder to make changes. To measure duplicated code:

- Line based text matching.
- Matching layout, expression and control flow metrics.

### 3.6 Halstead Metrics:

The main aim of these metrics is to find out the overall software production effort [15].

| Name | Notation | Description/Formula |
|---|---|---|
| Length | N | · N=N1+N2<br>· N1: Number of operators<br>· N2: Number of operands |
| Vocabulary | n | · n = n1 + n2<br>· n1: number of unique operators<br>· n2: number of unique operands |
| Volume | V | · Defined as a count of the number of mental comparisons required to generate a program.<br>· V = N * log2n |

| | | |
|---|---|---|
| Programming Effort | E | · Defined as measurement of the mental activity required to reduce a preconceived algorithm to a program P<br>· E = V/L<br>where L = Program Level<br>· E = (n1 * N2 * N log2n)/(2 * n2) |
| Programming Time | T | · T = E/S<br>where S is the Stroud number 4, defined as the number of elementary discriminations performed by the human brain per second<br>S value for software scientists is set to 18<br>· T = (n1 * N2 * N log2n) / (2 * n2 * S) |

**Table-4: Halstead Metrics**

**3.7 COCOMO model:**

Barry Boehm's COCOMO model asserts that effort required to develop a software system (measured by E in person months) is related to size (measured by S in thousands of delivered source statements) by the equation:

E = a Sb (where a and b are parameters determined by type of software system).

**3.8 Object-Oriented metrics:**

Traditional metrics such as cyclomatic complexity cannot measure OO concepts such as classes, inheritance and message passing.

New metrics have been developed to measure OO systems. One commonly used set of OO metrics is Chidamber and Kemerer's suite of class level metrics [14]:

**3.8.1 Weighted Methods Per Class (WMC):**

WMC is the sum of the static complexity of the methods.

**3.8.2 Depth of Inheritance Tree (DIT):**

When a class is deeply nested it inherits more from it's ancestors. This can increase the complexity of the class.

**3.8.3 Number of Children (NOC):**

Classes that have many children are hard to change because of the tight couplings with its children.

**3.8.4 Coupling Between Objects (CBO):**

A high number of couplings with other classes is disadvantageous because when the interface of a class it is coupled to changes it needs to be modified as well.

**3.8.5 Response For a Class (RFC):**

RFC is a measure of the interaction of a class with other classes.

**3.8.6 Lack of Cohesion in Methods (LCOM):**

This metric calculates the usage of a class's attributes in its methods. A class lacks cohesiveness when methods do not make use of its attributes.

## REFERENCES

[1] Sommerville. (2008). Software Engineering, 8/E. Pearson Education India. | [2] ISO/IEC 9126-1:2001. Software engineering -- Product quality. ISO/IEC. | [3] Sharma, A., & Dubey, S. K. Comparison of Software Quality Metrics for Object-Oriented System. International Journal of Computer Science, 12. | [4] Integration Watch: Using metrics effectively - SD Times: Software Development News. (n.d.). Retrieved April 8, 2013, from http://www.sdtimes.com/link/34157 | [5] Huizinga, D., & Kolawa, A. (2007). Automated Defect Prevention: Best Practices in Software Management (1st ed.). Wiley-IEEE Computer Society Pr. | [6] M.Subramanyam and R.Krishnan: "Emphirical Analysis of CK metrics for OOD complexity: Implication for software defect", IEEE transaction on software engineering, 2003. | [7] H.Lilu, K.Zhou and S.Yang: "Quality metrics of OOD for Software development and Re-development", First Asia-Pacific Conference on Quality Software, August 2002. | [8] NASA Software Assurance Technology Center, Software Metrics Research | and Development. http://satc.gsfc.nasa.gov/metrics/. Last visited august 2006 | [9] NASA Independent Verification and Validation Facility, Metrics Data | [10] D. Coleman, D. Ash, B. Lowther, P. Oman, Using Metrics to Evaluate Software System Maintainability, Computer, vol. 27, no. 8, pp. 44-49, Aug., 1994 | [11] Samoladas, I., Stamelos, I., Angelis, L., and Oikonomou, Open source software development should strive for even greater code maintainability, Communications of the ACM 47, 10 (Oct. 2004), 83-87. | [12] T.J. McCabe, A Complexity Measure, Proceedings of the 2nd international conference on Software engineering, 1976 | [13] R.E. Park, Software Size Measurement: A Framework for Counting Source Statements, Software Engineering Institute (CMU/SEI-92-TR-020), 1992 | [14] C. Archer. Measuring Object-Oriented Software Product, Software Engineering Institute (SEI-CM-28), 1995 | [15] Kan, S. H., & Kan. (2003). Metrics And Models In Software Quality Engineering, 2/E. Pearson Education India. | [16] Pressman, R. S. (2010). Software engineering: a practitioner's approach. McGraw-Hill Higher Education. | [17] Software metric. (2013, April 3). In Wikipedia, the free encyclopedia. Retrieved from http://en.wikipedia.org/w/index.php?title=Software_metric&oldid=543825909 | [18] Fenton, N. E., & Pfleeger, S. L. (1998). Software Metrics: A Rigorous and Practical Approach (2nd ed.). Boston, MA, USA: PWS Publishing Co. | [19] Boehm, B. W. (1981). Software Engineering Economics. Pearson Education. | [20] Barnard, J., & Price, A. (1994). Managing code inspection information. Software, IEEE, 11(2), 59-69.