Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 8, Issue, 05, pp.31565-31569, May, 2016

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

REVIEW ARTICLE

REVIEW OF SOFTWARE QUALITY METRIC USED IN TODAY'S MARKET

***Poonam Bhagwandas Godhwani and Anita Natubhai Gianchandani**

Department of Computer Science and Technology, UTU-Bardoli (Gujarat), India

ARTICLE INFO

Article History:
Received 18th February, 2016
Received in revised form
24th March, 2016
Accepted 10th April, 2016
Published online 30th May, 2016

Key words:

LOC-Line of Code, PUM-Per User Month, MTTF-Mean Time to Failure, OFE-opportunities for error, LOP-life of the product

ABSTRACT

The major focus in this technological era is on software development and the factors used in measuring the software quality, different types of the metric used in the current market to measure the software quality of their proposed system. It also discusses the issues in which users are concerned to the software quality of the product or system.

Citation: Poonam Bhagwandas Godhwani and Anita Natubhai Gianchandani, 2016. "Review of software quality metric used in today's market", *International Journal of Current Research*, 8, (05), 31565-31569.

INTRODUCTION

Software Quality measures how well software is designed (quality of design) and how well software conforms to that design (quality of conformance) (Jones, 1994). Quality of design is concerned with the implementation and quality of conformance refers to validity of design and its requirement (Daskalantonakis, 1992). Software quality may be defined as conformance to explicitly state functional and performance requirements, explicitly documented developed standards and implicit characteristics that are expected of all professionally developed software.

Above definition emphasis three points:

- Software requirements are the foundations from which quality is measured. Lack of conformance to requirement is lack of quality/
- Specified standards define set of development criteria that guide the software engineer. If criteria are not followed, lack of quality will almost result.
- A set of implicit requirements often goes unmentioned like ease of use, maintainability, etc. If software conforms to explicit requirements but fails to meet implicit requirements, software quality is suspected.

****Corresponding author: Poonam Bhagwandas Godhwani,***
Department of Computer Science and Technology,
UTU-Bardoli (Gujarat), India.

Steve McConnell's Code Complete says software possesses internal and external quality characteristics. External quality characteristics are those characteristics that face the user while internal quality characteristics are those characteristics that do not face the user (Myers, 1979). Dr. Tom DeMarco says product's quality is a function of how much it changes the world for better (DeMarco, 1999).

Important quality factors

Quality factors which are important are as follows:

Understandability: Purpose should be clarified very well, i.e., design and user documentation must be written clearly and properly so that it is easily understandable. In other words, user context must be taken into consideration. For example, if software engineer is going to use the software, it is not mandatory that layman should understand it.

Completeness: All parts must be present and fully developed. So if the code calls subroutine from external library, software package must provide reference to that library with all the required parameters and input data.

Conciseness: Excessive or redundant information or processing should be minimized especially when memory capacity is limited. This can be improved by placing repeated lines of code into subroutine or function which achieve that functionality, in case of documents also.

Portability

Portability refers to the proper execution of software with easiness with multiple computer configurations. This is applicable to both the hardware and operating system.

Consistency

Consistency refers to uniformity in notation, symbology, appearance and terminology.

Maintainability

Software product should be well maintained, that is, it should be well-documented with less complexity and should provides spare capacity for memory, storage and processor utilization and other resources.

Testability

If product is to be easily tested, disposition to support acceptance criteria and evaluation of performance feature should be included during design phase. Complex design leads to poor testability.

Usability

Usability refers to software product should be convenient and practical to use. Common example to this is Human-Computer Interface (HCI)

Reliability

Reliability refers to intended functions should do expected functionality within specific period of time and environmental conditions such as robustness.

Efficiency

Efficiency refers to fulfillment of purpose without waste of resources such as memory, space and processor utilization, network bandwidth, etc.

Security

Security refers to ability to protect data from unauthorized data and malicious or inadvertent interference with its operations through mechanisms like authentication, access control and encryption. Security also refers to resilience in the face of malicious, intelligent and adaptive attackers.

Integrity

Integrity refers to measurement of system's ability to stand against attacks with security.
Thus, software quality metric includes matrix to measure above stated quality factors needed to develop software properly.

Software Metric

Common software metric includes following things: (Jones, 1992)

- Bugs per line of code
- Code Coverage
- Cohesion
- Coupling
- Cyclomatic Complexity
- Function Point Analysis
- Number of Classes and Interfaces
- Number of line of Customer Requirements
- Order of Growth
- Source lines of Code
- Robert Cecil Martin's Software Package Metric

Software metric can be classified into following three categories:

- Product Metric – This metric describes the characteristics of the product such as size, complexity, design features, performance and quality test.
- Process Metric – This metric is used to improve software development and maintenance such as pattern testing, defect arrival, response time of the fix process and effectiveness of the defect removal during development.
- Project Metric – This metric describes the characteristics and execution of the software such as life cycle of the software, cost, schedule, productivity, etc.

Software quality metric is the subset of the metric that focuses on the quality aspect of the software, that is, product, process and project metric. Software quality can be again divided into end-product quality and in-process quality metrics.

Product Quality Metrics

Definition of software quality contains both intrinsic product quality and customer satisfaction. Following are the metrics available for these both quality:

Intrinsic product quality is measured by the number of bugs (functional defect) and how long software can run without encountering crash. Defect Density Rate and Mean Time to Failure (MTTF) are the metrics used for intrinsic product quality. MTTF metric is useful for safety-critical systems such as airline traffic control system, avionics, and weapons. Defect Density metric is used in many commercial systems.

Defect Density Metric

Defect rate is the number of defects over the opportunities for error (OFE) during specific time frame. Denominator is the size of the software which is usually thousand lines of code (KLOC) or in the number of function points (DeMarco, 1999; Jones, 1992) Time frame is expressed as life of the product (LOP) ranging from one year to many years after the software product's release to the general market. For counting KLOC, Line of Code (LOC) metric is used. In assembly programming, one physical line was same as one instruction but with the availability of high level languages there was difference between physical lines and instruction statements i.e. physical and logical lines. Certain variation which leads to this difference is as follows:

- Count only executable lines.
- Count executable lines plus data definitions.
- Count executable lines, data definitions, and comments.
- Count executable lines, data definitions, comments and job control language.
- Count lines as physical lines on an input screen.
- Count lines as terminated by delimiters.

In Boehm's book of Software Engineering Economics (1981), LOC counting method counts lines as physical lines and includes executable lines, data definitions and comments. In Software Engineering Metrics and Models by Conte et al (1986), LOC is defined as any line of program text that is not comment or blank line but the number of statements or fragments of statements in the line. This line includes all line containing program headers, declarations, and executable and non –executable statements. Efficient design provides the functionality with lower implementation effort and fewer LOCs. Therefore using LOC data to measure software productivity is like using the weight of an airplane to measure its speed and capacity (Jones, 1992). LOC data do not reflect non-coding work such as the creation of requirements, specifications, and user manuals. Defect density metric measures code quality per unit. But from customer's point of view, defect rate is not as relevant as the total number of defects that might affect their business. Therefore, a good defect rate should reduces the total number of defects from release-to-release i.e. if new release is larger than its predecessors, it means that defect rate for the new and changed code has to be significantly better than that of the previous release in order to reduce the total number of defects.

Function Point

Conte et al. (1986) defined function as collection of executable statements that performs certain task, together with declarations of the formal parameters and local variables manipulated by those statements (Jones, 1992; Jones,, 2000). The ultimate measure for software productivity is the number of functions a development team can produce given a certain amount of resources, regardless of the size of the software in the lines of code. The defect rate is indexed to the number of functions software provides. If defects per unit of functions are low, the software should have better quality even though the defects per KLOC value could be higher when the functions were implemented by fewer lines of code. Function Point metric is weighted total of five major component s that comprises an application:

- Number of external inputs (transaction types)
- Number of external outputs (report types)
- Number of logical internal files (file as user conceives and not the physical files)
- Number of external interface files (files accessed by the application and not maintained by it)
- Number of external inquiries (types of online inquiries supported)

Kemerer and Porter (1992) and Sprouls (1990) define low and high weighing factors depending on the complexity of the application in terms of above five components as follows:

- External Input – low complexity: 3, high complexity: 6
- External Output – low complexity: 4, high complexity: 7
- Logical Internal File – low complexity: 7, high complexity: 15
- External Interface File – low complexity: 5, high complexity: 10
- External Inquiry – low complexity: 3, high complexity: 6

In application contract work, the function point is used to measure the amount of work and quality is expressed as defects per function point whereas in system and real-time software, function point is used to measure the amount of work and quality as defects per function point but only in information system and not any other systems (Jones, 2000). This is due to inertia of the LOC related practices and the effort required for function point counting. Function point counting can be more time-consuming and expensive as compared to LOC based data as it involves studies involving multiple languages and those for productivity evaluation.

Customer Problems Metric

Major developers in the software industry measure the problems customer's encounters when using the product (Stevens and Myers, 1974). Defect density metric measures all the valid defects but from customer's perspective not only the valid defects but the problems that terminates the execution of the software like usability problem, unclear documentation or information, duplicates of valid effects or even user errors which is commonly known as non-defect oriented problems are the problems with the software. Problem metric is usually expressed as problems per user month (PUM) = Total problems that customer reported (true defects and non-defect oriented problems) for a time period. Time period may be total number of license months of the software during the period where number of license months = number of install licenses of the software i.e. number of months in calculation period. PUM is calculated for each month after software has been released to the market and also for monthly averages by year. This metric is related to problems to usage. This metric can be treated as intermediate between defects measurement and customer satisfaction. To reduce customer problems, one has to reduce the functional defects in the product and improve other factors like usability, documentation, problem rediscovery, etc. To improve customer satisfaction, one has to reduce defects and overall problems and manage factors of broader scope such as timing and availability of the product, company image, services, total customer solutions, and so forth.

Customer Satisfaction Metric

Following five point scales is used to determine the satisfaction level of the customer:

- Very satisfied
- Satisfied

- Neutral
- Dissatisfied
- Very dissatisfied

Customer satisfaction in software monitored by IBM includes CUPRIMDSO categories (capability, functionality, usability, performance, reliability, install ability, maintainability, and documentation/information, service and overall). Based on five-point scale analysis, several metrics with slight variations can be constructed and used depending on the purpose of analysis. For example,

- Percent of completely satisfied customers
- Percent of satisfied and completely satisfied customers
- Percent of dissatisfied and completely dissatisfied customers
- Percent of neutral, dissatisfied and completely dissatisfied customers

Usually percent satisfaction is used. Reduction of percent of non-satisfaction leads to reducing product defects. The weighted index approach is used. Some companies uses net satisfaction index (NSI) to facilitate comparisons across product. NSI has following weighting factors:

- Completely satisfied: 100%
- Satisfied: 75%
- Neutral: 50%
- Dissatisfied: 25%
- Completely dissatisfied: 0%

But this weighting factor may mask the satisfaction profile of one's customer set. For example, if half of the customers are completely satisfied and half are neutral, NSI value is 75% which is equivalent to the scenario when all customers are satisfied.

Cyclomatic complexity

Cyclomatic complexity is software metric developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of the program. It directly measures the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using control flow graph of the program. The nodes of the graph correspond to indivisible groups of command of program and directed edge connects two nodes (Stevens and Myers, 1974; Jones, Capers, 2009). Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within the program. Strategy is to use Basis Path Testing which tests for each linear independent path through the program where number of test cases will equal the cyclomatic complexity of the program (Jones, 2000). Complexity is defined as $M=E-N+2P$ where M=Cyclomatic complexity, E = Number of edges of the graph, N = Number of nodes of the graph, P = Number of connected components (Stevens and Myers, 1974).

Cohesion

One thinks that module with higher complexity tends to have lower cohesion than a module with lower complexity. Cohesion is a measure of how strongly related is the functionality expressed by the source code of software module. Cohesion is an ordinal type of measurement which can be expressed as high cohesion or low cohesion. Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand. Cohesion and Coupling were invented by Larry Constantine (Jones, Capers, 2009) based on the characteristics of good programming practices that reduced maintenance and modification costs. If the methods that serve the given class tend to be similar in many aspects, the class is said to have high cohesion. In highly cohesive system, code readability and likelihood of reuse is increased, while complexity is kept manageable. Cohesion is decreased if the functionalities embedded in the class accessed through its methods have little in common and methods carry out many varied activities often using coarsely-grained or unrelated sets of data. Cohesion can be any of the following:

- Coincidental Cohesion (worst)
- Logical Cohesion
- Temporal Cohesion
- Procedural Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Functional Cohesion (best)

Coupling

Coupling or Dependency is the degree to which each program module relies on each of the other modules. Module Coupling can be described as follows: For data and control flow coupling:

- di – number of input data parameters
- ci – number of input control parameters
- d0 – number of output data parameters
- c0 – number of output control parameters

For global coupling:

- gd – number of global variables used as data
- gc – number of global variables used as control

For environmental coupling:

- w – number of modules called (fan-out)
- r – number of modules calling the module under consideration (fan-in)

Coupling can be any of the following:

- Content Coupling (high)
- Common Coupling
- External Coupling
- Control Coupling
- Stamp Coupling (Data-structured Coupling)
- Data Coupling
- Message Coupling (low)
- No Coupling

Below table discusses different metrics used for different purpose:

Measure	Metrics
1.Customer satisfaction index	Number of system enhancement requests per year Number of maintenance fix requests per year User friendliness: call volume to customer service hotline User friendliness: training time per new user Number of product recalls or fix releases (software vendors) Number of production re-runs (in-house information systems groups)
2.Delivered defect quantities	Normalized per function point (or per LOC) At product delivery (first 3 months or first year of operation) Ongoing (per year of operation) By level of severity By category or cause, e.g.: requirements defect, design defect, code defect, documentation/on-line help defect, defect introduced by fixes, etc.
3.Responsiveness (turnaround time) to users	Turnaround time for defect fixes, by level of severity Time for minor vs. major enhancements; actual vs. planned elapsed time (by customers) in the first year after product delivery
7.Complexity of delivered product	McCabe's cyclomatic complexity counts across the system Halstead's measure Card's design complexity measures Predicted defects and maintenance costs, based on complexity measures
8. Test coverage	Breadth of functional coverage Percentage of paths, branches or conditions that were actually tested Percentage by criticality level: perceived level of risk of paths The ratio of the number of detected faults to the number of predicted faults.
9. Cost of defects	Business losses per defect that occurs during operation Business interruption costs; costs of work-around Lost sales and lost goodwill Litigation costs resulting from defects Annual maintenance cost (per function point) Annual operating cost (per function point) Measurable damage to your boss's career
10. Costs of quality activities	Costs of reviews, inspections and preventive measures Costs of test planning and preparation Costs of test execution, defect tracking, version and change control Costs of diagnostics, debugging and fixing Costs of tools and tool support Costs of tools and tool support Costs of test case library maintenance Costs of testing & QA education associated with the product Costs of monitoring and oversight by the QA organization (if separate from the development and test organizations)
11. Re-work	Re-work effort (hours, as a percentage of the original coding hours) Re-worked LOC (source lines of code, as a percentage of the total delivered LOC) Re-worked software components (as a percentage of the total delivered components)
12. Reliability	Availability (percentage of time a system is available, versus the time the system is needed to be available) Mean time between failure (MTBF) Mean time to repair (MTTR) Reliability ratio (MTBF / MTTR) Number of product recalls or fix releases Number of production re-runs as a ratio of production runs

CASE STUDY

Scalability Overview of Windows Server 2008 Terminal Services Gateway

Terminal Services Gateway (TS Gateway) is server role in Windows Server 2008. It enables authorized users to connect to Remote Desktop Protocol (RDP) accessible resources on internal corporate networks from any Internet-connected device that can run the Remote Desktop Connection (RDC) client. Metric that they haves used is the response time to calculate the performance of TS Gateway. Response time is the time taken for data packet to travel from the Terminal Services client through TS Gateway to the Terminal Server and back to Terminal Services client. Based on these data, they have developed Knowledge worker scenario. These metrics were used to average typical knowledge workers usage in Terminal Services which includes MS Office application usage. Scenarios developed using these data are as follows:

- Knowledge worker data rate
- Number of Processors Variation Test
- Amount of Physical Memory (RAM) Variation Test
- Frequency Variation Test
- Packet Size Variation Test
- Central vs. Local Network Policy Server (NPS)
- TS Gateway Server Farm Test

Conclusion

Several components are crucial while measuring the quality of software using various metric. The response time of the product can be considered as one of the software quality metric by studying the case study. Here major focus is on to determine that in which circumstances which metrics should be used.

REFERENCES

Daskalantonakis, M. K. 1992. "A Practical View of Software Measurement and Implementation Experiences Within Motorola," IEEE Transactions on Software Engineering, Vol. SE-18, pp. 998-1010

DeMarco, T. 1999. *Management Can Make Quality (Im)possible*, Cutter IT Summit, Boston.

Jones, C. 1992. "Critical Problems in Software Measurement," Burlington, Mass.: Software Productivity Research.

Jones, C. 1994. Assessment and Control of Software Risks, *Englewood Cliffs, N. J.*: Yourdon Press.

Jones, C. 2000. Software Assessments, Benchmarks, and Best Practices, Boston: Addison-Wesley.

Jones, Capers, 2009. A Short History of Lines-of-Code Metrics; Capers Jones & Associates LLC; Narragansett, RI; September, 20 page

Myers, G. J. 1979. The Art of Software Testing, New York: John Wiley & Sons.

W. Stevens, G. Myers, L. 1974. Constantine, "Structured Design", *IBM Systems Journal*, 13 (2), 115-139.

****