An Analysis of Quality in Software Development

Umakant Singh[1] Kamal Kant[2]
[1]Department of Computer Application [2]Department of Computer Science & Engineering
[1]Naraina College of Management, Kanpur, U.P, India [2]Naraina Vidyapeeth Engineering &Management Institute, Kanpur, U.P, India

Abstract— This paper describes software quality, software quality criteria definitions and CISQ's (Consortium for IT Software Quality) quality model and provides the relevant matter about the requirements of quality in software field. What is the role of basic quality attributes to develop the software business and IT industries, is described. There is direct impact of measurement techniques on software development and how can we improve the quality of business using software quality is also mentioned. Quality in fact aids easy and high productivity, which has brought software metrics to the forefront. Today there are many software quality modes to improve the quality of software but which model is suitable for our project is very important.
Keywords: Quality; Software Quality; Software Quality Criteria

I. INTRODUCTION

"If you cannot measure, you cannot control" – Tom DeMarco. According to this statement software quality is related to the measurement. So correct measurements and valid methods are very important for improving the quality of software.

Software quality management manages the quality of software and shows software development process. It ensures that the quality of required level is achieved in a software product. It encourages to a company-wide "Quality Culture" where everyone's responsibility is quality. It enables process fault avoidance and fault prevention throughout in development. It also reduces the curve of learning and help with continuity in case if team members change position in organization. If one does not have any information about quality of software he/she cannot predict the future of software. In 2004, the Standish Chaos report found only 29% project met their quality criteria for success of project and this report also told that cancelled cost of that projects were near about $55 billion. If quality of software is not good then probability of failure of projects may increase.

II. SOFTWARE QUALITY

There are numerous definitions of software quality. First definition is given within the beginning 20th century from Shewhart. "There are two common aspects of quality: one among them has got to do with the consideration of the standard of a thing as an objective reality independent of the existence of man. the opposite has got to do with what we expect , feel or sense as a results of the target reality. In other words, there's a subjective side of quality"[1].

Dr. Tom DeMarco has proposed that "a product's quality is a function of what proportion it changes the planet for the better" [2]. Another software quality definition is given by Feigenbaum "Quality may be a customer determination, not an engineer's determination, not a marketing determination, nor a general management determination. it's supported upon the customer's actual experience with the merchandise or service, measured against his or her requirements - stated or unstated, conscious or merely sensed, technically operational or entirely - subjective and always representing a moving target during a competitive market" [3].

III. QUALITY CRITERIA DEFINITIONS

Some important quality criteria definitions have been shown in table I.

Acquisition Concern : GENERAL		
Acronym	Criterion	Definition
MD	Modularity	Program subunits that are developed in dependently or characteristics of software which provide a structure of Highly cohesive components with optimum coupling of software.
SD	Self-Descriptivenes s	Those properties of software which provide explanation of the implementation of functions.
SI	Simplicity	Simplicity is the extent that it uses data and control structures for organizing the program, and uses easily understood constructors [4].
Acquisition Concern : ADAPTATION		
AP	Application Independence	Those attributes of software which determine its nondependency on computer architecture, system microcode,
		algorithms and database.
AT	Augmentabilit y	Those properties of software which provide capability for functions and data for expansion.
CL	Commonality	Those attributes of software which provide for the use of interface standards for data representations, protocols, and routines.
DO	Document Accessibility	Those properties of software which provide for easy access.
FO	Functional Overlap	Those qualities of software which provide common functions to both systems.
FS	Functional Scope	Those attributes of software which provide commonality of functions amongst applications.
FE	Generality	Those properties of software

		which provide expend to the functions performed with respect to the application.
ID	Independence	Those qualities of software which determine its nondependency on software environment (libraries, input, output routines, utilities, computing system and operating system).
ST	System Clarity	Those attributes of software which provide for clear description of program structure in a understandable manner non- complex.
SY	System Compatibility	Those qualities of software which provide the software, hardware, and communication compatibility of two systems.
Acquisition Concern : DESIGN		
VR	Virtuality	Those properties of software which present a system that does not require user knowledge of the logical, physical, or topological characteristics.
CP	Completeness	It provides full implementation of the functions required [5] [6] [7].
CS	Consistency	Those qualities of software which provide for uniform design and notation implementation techniques [6].
TC	Traceability	Those attributes of software which provide a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment.
VS	Visibility	Those attributes of software which provide status monitoring of the development process and operations.
Acquisition Concern : PERFORMANCE		
AC	Accuracy	Those qualities of software which provide the required precision in calculations and outputs [5] [6] [8].
AM	Anomaly Management	Those qualities of software which provide for continuity of operations under and recovery from non- nominal conditions.
AU	Autonomy	Those properties of software which determine its non-dependency on interfaces and functions.
D	Distributedness	Determine the degree to which software functions are logically
		geographically separated within the system.
EC	Effectiveness-Comm.	Those qualities of the software which provide for minimum utilization of communication resources in performing functions.
EP	Effectiveness-Processing	Those properties of software which provide for minimum utilization of processing resources in performing functions.
ES	Effectiveness-Storage	Those qualities of the software which provide for minimum utilization of storage resources.
OP	Operability	Attributes of software that relate to the users' effort for operation and operation control.
RE	Reconfigurability	Those qualities of software which provide for continuity of system operation when one or more storage units, communications processors, or links fail(s).
SS	System Accessibility	Those properties of software which provide for control and audit of access to the data and software.
TN	Training	Those qualities of software which provide transition from current operation and provide initial familiarization.

Table 1: Software Quality Criterias with Their Definitions

IV. CISQ'S QUALITY MODEL

Software quality is a perceptual and conditional subjective attributes and may be differ by different people. The characteristics of software quality have been defined by Consortium for IT Software Quality (CISQ). CISQ has defined five major characteristics of software needed to provide business value, under the guidance of CISQ's first director; Bill Curtis, co-author of the Capability Maturity Model framework (CMM) and Capers Jones, CISQ's distinguish advisor. There are following "Whats" that need to be achieved in the house of quality.

A. Efficiency:

To measure the execution behavior (storage speed and execution speed) of program is called efficiency. The software architecture attributes and source code are the elements that ensure high performance once the application is in run-time mode. Efficiency is very important in high execution environments such as transaction of money from ATM, RADAR system in defense, artificial satellite systems, mars and moon projects etc., where execution speed plays an important role. A clear picture of the latent business risks and the harm is provided by analysis of source code efficiency and scalability. It is extent to which resources are utilized. If any system has high efficiency it means high software efficiency and usability. The efficiency

depends on the machine and operating system environment on which program or software is running and implementation language.

B. Reliability:

Here are two groups definitions generally related to software and hardware. According to the first group definition reliability is the measures of the number of error encountered in a program. In a broader definition relates that it is the extent to which a program can be expected to perform its intended functions satisfactorily (according to Thayar 1976). In another critic adds a qualifier: the functions of software must be performed correctly in spite of computer component failures (McCall, Richards, and Walters 1977). In this view correctness and performance of software are closely related to the reliability.

On the other hand hardware reliability has a certain physical and tangible element. Reliability measures potential application failure and the level of risk. The goal for monitoring and checking reliability measure the execution behavior (storage speed and execution speed) of program is called efficiency. It affects users directly and improve the image of IT and business performance.

C. Maintainability:

To measure the effort and time required to fix a bug in the program is called maintainability (McCall, Richards, and Walters 1977). According to Boehm "It is the ease or difficulty with which software can be updated to satisfy new requirements". Measurements of how easily software can be changed because of bugs encountered during operation, user requirements that were not fulfilled, changing requirements according to client etc., and upgrading or "obsoleting" a system (McCall, Richards, and Walters 1977). Maintainability is a software quality factor that addresses how well software can be maintained after it has been developed. Here is the question of how well the software adapts to changing requirements or to improvements made to the product. In such terms, maintenance can be closely aligned with flexibility and expandability. It may include notion of transferability (from one development team to another), portability and adaptability. Where change is driven by tight time-to-market schedules and where it is important for IT to remain responsive to business-driven changes measuring and monitoring, maintainability is a must for such mission-critical applications. Size of program plays a very important role in maintainability because according to size of program compilation speed may increase or decrease and processing of program will be affected.

D. Security:

Security is very important quality attribute which is related to security of business and it can damage the business. Security is only concerned with unauthorized access to information. It contributes directly to integrity. When a software program is not secure, the confidentiality of the data is compromised, whether or not the software program is subjected to adverse conditions. It measures the likelihood of potential security breaches and unwanted action due to poor coding practices and architectures. It quantifies the risk of encountering critical vulnerabilities that are harmful for our business.

E. Size:

We measure the source code size in in terms of Source Lines of Code (SLOC) by counting the lines of a program. The sizing of source code is a software characteristic that obviously impacts maintainability. After Combining with the above quality characteristics, software size can be used to assess the amount of effort produced and to be done by teams, as well as their productivity through correlation with time- sheet data, and other SDLC-related software metrics. It is also used to measure effort both after as an actual value and before as an estimate. Before as an estimate and after as an actual value both are measured by SLOC. It comes from the days of assembly coding and FORTRAN.

SLOC provide a much clearer image to software developers on the size of the project. When source code is written, integration and unit testing can be performed so measures of quality and programming productivity can be assessed.

SLOC themselves are not as meaningful as other software metrics. Just because if one project has more lines of code than other does not mean that first project is more complex or give better quality rating. SLOC is a direct metrics which can be measured directly by counting the numbers of lines of a program. This metrics play important role to derive the mathematics formulas of indirect metrics. These indirect metrics define the product quality. When product quality is mainly depend on SLOC, it gives a much better reflection on the overall project's quality; efficiency of the code; ratio of good code to buggy code etc.

In another point of view SLOC is the measuring the actual number of lines of code written in specific amount of time. If we are looking at a project level, the number of lines of code would typically come from the overall lines of code written throughout the project within a specific amount of period of time. If there is a team of many developers then total number of lines of code would obviously be measured on the lines written by the developer or team respectively. If developers are using auto-generated code software it can be lead to incorrect productivity measures. These lines of code are un useful and should be removed from the calculation to give a better indication of actual number of lines written by the developer or team.

For example if we are using two different languages FORTRAN and C to develop the same function code and C takes 10 lines of code and FORTRAN takes 30 lines of code, it is short coming of SLOC. There may be different measurements for the same function in different programming languages so here function points analysis metrics is better than SLOC.

There are two ways to measure lines of code:

1) Physical SLOC = Lines of Source Code + Comments + Blank lines*
2) Logical SLOC = Number of lines of functional code ("statements")

* If blank lines are less the 25% of the section.

Names of SLOC measures:

- K-LOC – 1,000 Lines of Code

- K-SLOC - 1,000 Source Lines of Code
- K-DLOC - 1,000 Delivered Lines of Code
- M-LOC - 1,000,000 Lines of Code
- G-LOC - 1,000,000,000 Lines of Code
- T-LOC - 1,000,000,000,000 Lines of Code

[K – Kilo, M – Mega, G – Giga, T – Tera]

1) Physical SLOC (PLOC)

The most common definition of physical SLOC is a count of lines including comment lines in the text of the program's source code. Blank lines are also counted unless the line of code in a section consists of more than 25% blank lines. In this case blank lines more than 25% are not counted toward lines of code.

2) Logical SLOC (LLOC)

Logical SLOC measures the number of executable "statements", but their specific definitions are tied to particular computer languages (one simple logical SLOC measure for C-like programming languages is the number of statement-terminating semicolons). It is much easier to create tools that measure physical SLOC, and physical SLOC definitions are easy to explain.

By comparing logical SLOC and physical SLOC we find that measurements of physical SLOC are sensitive to logically irrelevant formatting and style conventions, on the other hand logical SLOC is less sensitive to formatting and style conventions. However, logical SLOC can often be significantly different from physical SLOC and SLOC measures are often stated without giving their definition. Let us take some snapshot C code as an example of the ambiguity encountered when determining SLOC:

```
for (j = 0; j < 100; j++) printf("hi"); /* How many lines of code is this? */
```

In this example we have:

- 2 Logical Lines of Code (LLOC) (for statement and printf statement)
- 1 Physical Line of Code (LOC)
- 1 comment line

According to different programmer and coding standards, the above "line of code" could be written on many separate lines in the following way:

```
/* Now find how many lines of code in this? */
for (j = 0; j < 100; j++)
{
   printf("hi");
}
```

In the above example we have:

- 2 Logical Line of Code (LLOC): what about all the work writing non-statement lines?
- 5 Physical Lines of Code (LOC): is placing braces work to be estimated?
- 1 comment line: tools must account for all code and comments regardless of comment placement.

Even the "physical" and "logical" SLOC values can have a large number of different definitions. Robert E. Park (while at the Software Engineering Institute) et al. developed a framework for defining the SLOC values, to enable people to carefully explain and define the SLOC measure criteria used in a project. Considering an example, most software systems reuse code, and determining which (if any) reused code to include is important when reporting a measure.

V. Discussion, Conclusion and Future Work

In this paper we described the quality, quality criteria of software by which we can improve the performance of software. We have presented CISQ's software quality model that is used to solve software business problems. We have described here what is the role of program size (SLOC), is to develop the software and what impact of SLOC to improve the business is and what the disadvantages of SLOC are. In present time there are so many software quality models that are being used in software industry to improving the software quality but we have to choose best software quality model according to our requirement. First of all we have to select quality attributes that are used in our project or software after that we have to select software quality models and lastly according to our priority we have to select one of the software quality models and apply it into our project. There may be different combinations of quality attributes to make many different quality models. These quality models have different properties. We have many software quality models at the present time so we have to choose very carefully according to our need. It is not necessary that we should use only one quality model in all projects, we should use such quality model that provides better and efficient result. Software engineering provides us many rules and regulations to select the appropriate software quality models.

In this area of research work there are a lot of queries, and their solutions which may be found in our future work. These queries may be as given below:

1) What are the parameters to relate these quality models?
2) How a business performance is related to the software quality models in real world?
3) How we identify which software quality model is more important than other software quality model and why?

References

[1] W. A. Shewhart, "Economic control of quality of manufactured product", Van Nostrand, 1931.
[2] DeMarco, T., "Management Can Make Quality (Im)possible", Cutter IT Summit, Boston, April 1999.
[3] McCall, Jim A., Paul K. Richards, and Gene F. Walters,"Factors in Software Quality", RADC-TR-77-369, Volumes I, II, and III, RADC, Griffiss Air Force Base, NY, November 1977.
[4] Gilb, T., "Software Metrics", Winthrop, Inc., Cambridge, MA, 1977.
[5] Boehm, B. W. et al., "Characteristics of Software Quality", North- Holland Publishing Company, New York, NY, 1978.
[6] Bowen, Thomas P., Gary B. Wigle, and Jay T. Tsai, "Specification of Software Quality Attributes", RADC-TR-85-37, RADC, Griffiss Air Force Base, NY, Volumes I, II, and III, February 1985.
[7] Walters, G. F., "Applications of Metrics to a Software Quality Management (QM) Program", Software Quality Management, Petro celli, New York, NY, 1979

