25.3 METRICS FOR SOFTWARE QUALITY

The goal of software engineering is to produce a high-quality system, application, or product within a time frame that satisfies a market need. To achieve this, you must apply effective methods coupled with modern tools within the context of a mature software process.

The quality of a system, application, or product is only as good as

i. The requirements that describe the problem,
ii. The design that models the solution,
iii. The code that leads to an executable program, and
iv. The tests that exercise the software to uncover errors.

You can use measurement to assess the quality based on above four parts. To accomplish this real-time assessment, you apply product metrics to evaluate the quality of software engineering work products in objective, rather than subjective ways.

A project manager must also evaluate quality as the project progresses. Metrics collected by individual software engineers are combined to provide project level results. Although many quality measures can be collected, ***the primary thrust at the project level is to measure errors and defects***. Metrics derived from these measures provide an indication of the effectiveness of individual and group software quality assurance and control activities.

Metrics such as

a) Work product errors per function point,
b) Errors uncovered per review hour, and
c) Errors uncovered per testing hour

provide insight into the efficacy of each of the activities implied by the metric. Error data can also be used to compute the defect removal efficiency (DRE) for each process framework activity.

25.3.1 Measuring Quality

There are many measures of software quality

1) **Correctness** is the degree to which the software performs its required function. It is measured in terms of defects per KLOC. Defects are those problems reported by a user of the program after the program has been released for general use. For quality assessment purposes, defects are counted over a standard period of time, typically one year.

2) **Maintainability** consumes more effort. Maintainability is all about to correct an error if encountered, adapt any change in environment, or enhancement of product if the customer desires a change in requirements. There is no way to measure maintainability directly. It can be measured by a simple time-oriented metric i.e. ***mean-time-to-change (MTTC)***. MTTC maintain the project as follows:-

   a) the time it takes to analyze the change request,
   b) design an appropriate modification,
   c) implement the change,
   d) test it, and
   e) Distribute the change to all users.

On average, programs that are maintainable will have a lower MTTC than programs that are not maintainable.

3) **Integrity** This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security. Attacks can be made on all three components of software: programs, data, and documentation.

   It can be measured in terms of threat and security.

   a) Threat is the probability that an attack of a specific type will occur within a given time.
   b) Security is the probability that the attack of a specific type will be repelled. The integrity of a system can then be defined as:

      $$\text{Integrity} = \Sigma\,[1 - (\text{threat} \times (1 - \text{security}))]$$

      For example, if threat probability is 0.25 and security probability is 0.95, the integrity of the system is 0.99 (very high). If, on the other hand, the threat probability is 0.50 and the security probability is only 0.25, the integrity of the system is 0.63 (unacceptably low).

4) **Usability**- If a program is not easy to use, it is often doomed to failure, even if the functions that it performs are valuable. Usability is an attempt to quantify ease of use.

25.3.2 Defect Removal Efficiency

A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE). When considered for a project as a whole, DRE is defined in the following manner:

$$\text{DRE} = \frac{E}{E + D}$$

Reference:- Roger S Pressman - Software engineering _ a practitioner's approach-McGraw-Hill Higher Education (2010)

Where E is the number of errors found before delivery of the software to the end user and D is the number of defects found after delivery.

The ideal value for DRE is 1. That is, no defects are found in the software. Realistically, D will be greater than 0, but the value of DRE can still approach 1 as E increases for a given value of D.

As E increases, it is likely that the final value of D will decrease (errors are filtered out before they become defects).

DRE encourages a software project team to institute techniques for finding as many errors as possible before delivery.

DRE also assess a team's ability to find errors before they are passed to the next framework activity. For example, requirements analysis produces a requirements model that can be reviewed to find and correct errors. Those errors that are not found during the review of the requirements model are passed on to design (where they may or may not be found). When used in this context, we redefine DRE as

$$\mathrm{DRE}_i = \frac{E_i}{E_i + E_{i+1}}$$

Where E_i is the number of errors found during software engineering action i and E_{i+1} is the number of errors found during software engineering action i + 1 that are traceable to errors that were not discovered in software engineering action i.

A quality objective for a software team (or an individual software engineer) is to achieve DRE_i that approaches 1. That is, errors should be filtered out before they are passed on to the next activity or action.

Reference:- Roger S Pressman - Software engineering _ a practitioner's approach-McGraw-Hill Higher Education (2010)