Paper Name: Software Engineering

Topic: Software Quality

Paper Code: BCA CC409

Semester IV

By Ms. Amrita Prakash

Assistant Professor,

Department of Computer Science

Patna Women's College

E-mail: amrita.bca@patnawomenscollege.in

Software Quality

- Software quality depends on various characteristics, such as correctness, reliability and efficiency.
- The main objective of the software development team is to design the software with required functionality and minimum errors according to specified user requirements.
- Quality refers to the features and characteristics of a product or service, which define its ability to satisfy user requirements.
- **Software** quality product is defined in term of its fitness of purpose.
- That is, a quality product does precisely what the users want it to do.
- For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document.
- Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

Software Quality refers to the extent to which the software is able to perform correctly along with required features and functions. For this, a planned and systematic set of activities is performed.

IEEE defines software quality as '(1) the degree to which a system, component or process meets specified requirements.
(2) The degree to which a system, component or process meets customer or user needs or expectations.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

The principles followed for achieving good quality software:-

- Prevention of Errors: In order to avoid errors in the code, software tools are extensively used.

- Detection and Correction of Errors: Errors should be detected in the early stages of Software Development, so that much time is not consumed to correct them in the later stages. Thus, quality control should be carried out during all phases of SDLC.

- Elimination of Cause (s) of Errors: After detection of errors, it is necessary to eliminate their cause. Eliminating errors increases the quality of the software.

- Independent Audit: The objective of an audit is to assess the product and process activities. Thus, independent audit for product and processshould be conducted in accordance with the standards and procedures established in the quality process.

Software Quality Control

Software Quality Control is concerned with product quality and refers to the functions of software quality, which determine that the software is developed in accordance with standards and procedures.
In addition, Software Quality Control checks whether the product meets user requirements or not.

Objectives of Software Quality Control are :-

- Comparison on product quality procedures with the established standards.
- Identification of errors in order to rectify them.
- Focus on reviews and testing the product.

- Identification of outputs generated by the product.

Software Quality Assurance

Software Quality Assurance is concerned with process quality and refers to planned and systematic sets of activities that ensure that software lifecycle processes and products conform to requirements, standards and procedures.

Objectives of Software Quality Assurance are: -

- To determine the objectives of the software which are to be accomplished
- To establish the software plans, when the objectives are determined
- To monitor and adjust software plans to satisfy user requirements.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

- Auditing of projects
- Review of the quality system
- Development of standards, methods, and guidelines, etc.
- Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Evolution of Quality Management System

Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

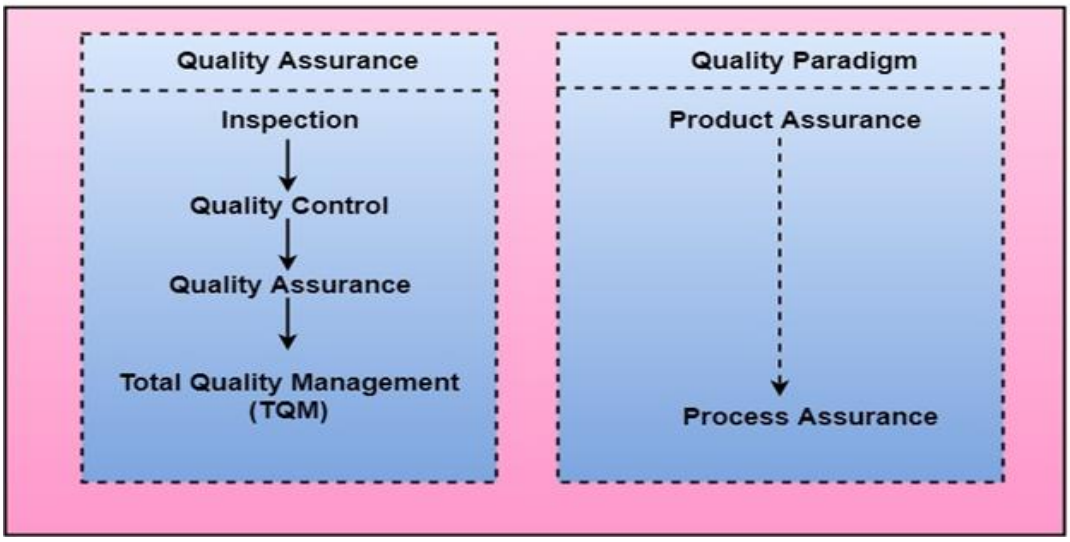
Quality control target not only on detecting the defective devices and removes them but also on determining the causes behind the defects. Thus, quality control aims at correcting the reasons for bugs and not just rejecting the products. The next breakthrough in quality methods was the development of quality assurance methods.

The primary premise of modern quality assurance is that if an organization's processes are proper and are followed rigorously, then the products are obligated to be of good quality. The new quality functions include guidance for recognizing, defining, analyzing, and improving the production process.

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

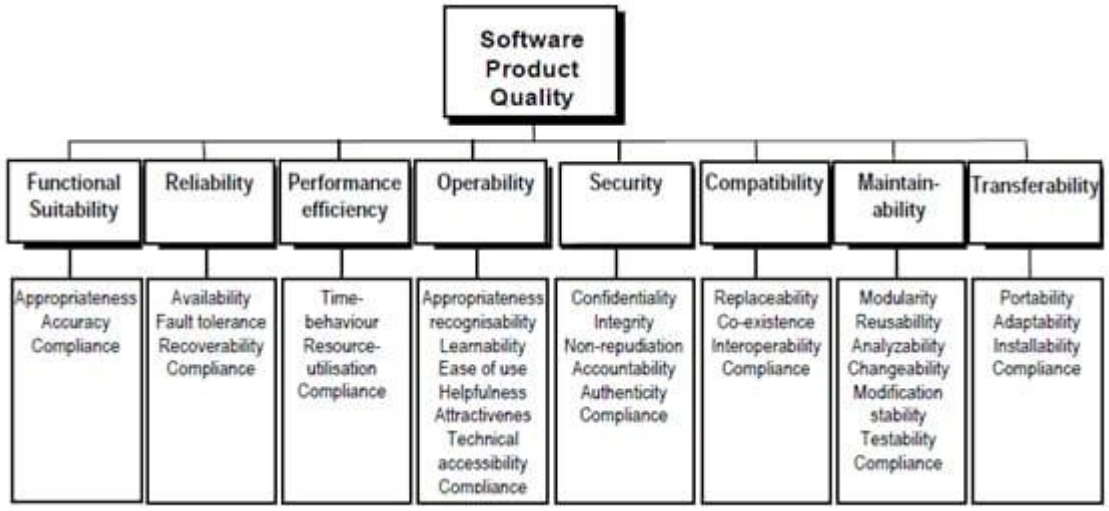
BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.

Evolution of quality system and corresponding shift in the quality paradigm



In broader terms, the software quality definition of "fitness for purpose" refers to the satisfaction of requirements. But what are requirements? Requirements, also called user stories in today's Agile terms, can be categorized as functional and non-functional. Functional requirements refer to specific functions that the software should be able to perform. For example, the ability to print on an HP Inkjet 2330 printer is a functional requirement. However, just because the software has a certain function or a user can complete a task using the software, does not mean that the software is of good quality. There are probably many instances where you've used software and it did what it was supposed to do, such as find you a flight or make a hotel reservation, but you thought it was poor quality. This is because of "how" the function was implemented. The dissatisfaction with "how" represents the non-functional requirements not being met.

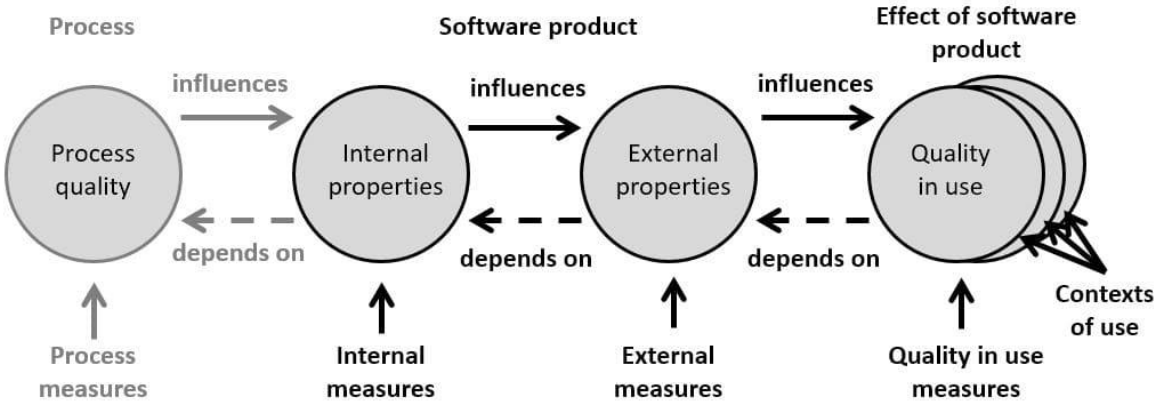
For this purpose the International Organization for Standardization (ISO) developed ISO 25010 as a model for specifying non-functional requirements. The model shown below illustrates the categorization of non-functional requirements.



At first glance, you may think that the left most characteristic, Functional Suitability, is equivalent to a functional requirement, but it's not. Sub-characteristics functional completeness, functional correctness and functional appropriateness apply to functions that have been implemented and are characteristics of those functions. For instance, functional completeness is defined as the degree to which the set of functions covers all the specified tasks and user objectives. So, "Print from HP Inkjet 2330 Printer" could have been implemented from a functional requirement point of view. But how was it

implemented? Was it complete for all options? Did it have double-sided printing? If not, then it might not be good quality from a functional completeness point of view.

Software Quality Lifecycle



Here is yet another model: quality lifecycle. This model may be closer to how you perceive quality. That is, quality from an end user viewpoint when they are actually using the software in real life and not in a lab. That's what "contexts of use" means. For example, software quality testing can happen on a test server and have perfect test results, but users in their environment may have different results. They may not be able to find a button or control as easily as a tester would, or maybe they want to print directly from a place in the application that you never thought of.

Also important to note in Figure 2 is the use of the arrows and dotted lines. You'll notice that there is a relationship between internal quality, external quality and quality in use. Namely, internal quality has an influence on, but not a direct correlation with, external quality and that external quality depends on internal quality. Let's think about this. This means that you can have great code quality (internal quality) and still have poor external quality (software behaviour). This makes sense in reverse too. The software might work okay, but the internal quality could be terrible.