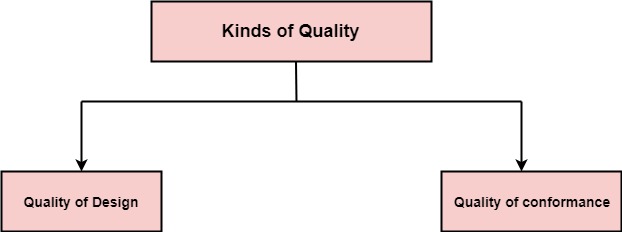
**Software Quality Assurance**

What is Quality?

Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

**There are two kinds of Quality:**



**Quality of Design:** Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

**Quality of conformance:** Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

**Software Quality:** Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.

**Quality Control:** Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.

**Quality Assurance:** Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

**Quality Assurance** focuses on how the engineering and management activity will be done.

As anyone is interested in the quality of the final product, it should be assured that we are building the right product.

It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

## Importance of Quality

We would expect the quality to be a concern of all producers of goods and services. However, the distinctive characteristics of software and in particular its intangibility and complexity, make special demands.

**Increasing criticality of software:** The final customer or user is naturally concerned about the general quality of software, especially its reliability. This is increasing in the case as organizations become more dependent on their computer systems and software is used more and more in safety-critical areas. For example, to control aircraft.

**The intangibility of software:** This makes it challenging to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce 'deliverables' that can be examined for quality.

**Accumulating errors during software development:** As computer system development is made up of several steps where the output from one level is input to the next, the errors in the earlier or deliverables will be added to those in the later stages leading to accumulated determinable effects. In general the later in a project that an error is found, the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly challenging to control.

## Software Quality Assurance

Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

## SQA Encompasses

* A quality management approach
* Effective Software engineering technology (methods and tools)
* Formal technical reviews that are tested throughout the software process
* A multitier testing strategy
* Control of software documentation and the changes made to it.
* A procedure to ensure compliances with software development standards
* Measuring and reporting mechanisms.

## SQA Activities

Software quality assurance is composed of a variety of functions associated with two different constituencies ? the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

**Following activities are performed by an independent SQA group:**

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g. ISO-9001), and other parts of the software project plan.
3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.
5. **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project method, process description, applicable standards, or technical work products.
6. **Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

Quality Assurance v/s Quality control

|  |  |
| --- | --- |
| **Quality Assurance** | **Quality Control** |
| **Quality Assurance (QA)** is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use. | **Quality Control (QC)** is described as the processes and methods used to compare product quality to requirements and applicable standards, and the actions are taken when a nonconformance is detected. |
| **QA** is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA. | **QC** is an activity that demonstrates whether or not the product produced met standards. |
| **QA** helps establish process | **QC** relates to a particular product or service |
| **QA** sets up a measurement program to evaluate processes | **QC** verified whether particular attributes exist, or do not exist, in a explicit product or service. |
| **QA** identifies weakness in processes and improves them | **QC** identifies defects for the primary goals of correcting errors. |
| Quality Assurance is a managerial tool. | Quality Control is a corrective tool. |
| Verification is an example of QA. | Validation is an example of QC. |

# Software Quality Metrics

**Software quality metrics** are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

Software quality metrics can be further divided into three categories −

* **Product quality metrics**
* **In-process quality metrics**
* **Maintenance quality metrics**

## 1.Product Quality Metrics

This metrics include the following −

* Mean Time to Failure
* Defect Density
* Customer Problems
* Customer Satisfaction

### **Mean Time to Failure**

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

### **Defect Density**

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

### **Customer Problems**

It measures the problems that customers encounter when using the product. It contains the customer’s perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**.

PUM = Total Problems that customers reported (true defect and non-defect oriented

problems) for a time period + Total number of license months of the software during

the period

Where,

Number of license-month of the software = Number of install license of the software ×

Number of months in the calculation period

PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year.

### **Customer Satisfaction**

Customer satisfaction is often measured by customer survey data through the five-point scale −

* Very satisfied
* Satisfied
* Neutral
* Dissatisfied
* Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, 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 customers
* Percent of dis-satisfied customers
* Percent of non-satisfied customers

Usually, this percent satisfaction is used.

## 2. In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes −

* Defect density during machine testing
* Defect arrival pattern during machine testing
* Phase-based defect removal pattern
* Defect removal effectiveness

### **Defect density during machine testing**

Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.

This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

### **Defect arrival pattern during machine testing**

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following −

* The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.
* The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
* The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

### **Phase-based defect removal pattern**

This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

### **Defect removal effectiveness**

It can be defined as follows –

DRE=Defect removed during a development phase / Defects latent in the product × 100%

This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used for the front-end and **phase effectiveness** for specific phases. The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

## 3.Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

* Fix backlog and backlog management index
* Fix response time and fix responsiveness
* Percent delinquent fixes
* Fix quality

### **Fix backlog and backlog management index**

Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.

Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

BMI=Number of problems closed during the month /Number of problems arrived during the month ×100

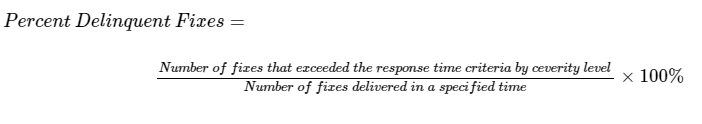
If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

### **Fix response time and fix responsiveness**

The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.

The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

### **Percent delinquent fixes**



It is calculated as follows −

### **Fix Quality**

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.

A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.

Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

# Formal Technical Review (FTR) in Software Engineering

**Formal Technical Review (FTR)** is a software quality control activity performed by software engineers.

**Objectives of formal technical review (FTR):** Some of these are:

* Useful to uncover error in logic, function and implementation for any representation of the software.
* The purpose of FTR is to verify that the software meets specified requirements.
* To ensure that software is represented according to predefined standards.
* It helps to review the uniformity in software that is development in a uniform manner.
* To makes the project more manageable.

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely. FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise. Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successful only if it is properly planned, controlled and attended.

**Example:**

suppose during the development of the software **without FTR** design cost 10 units, coding cost 15 units and testing cost 10 units then the total cost till now is 25 units without maintenance but there was a quality issue because of bad design so to fix it we have to re design the software and final cost will become 50 units. that is why FTR is so helpful while developing the software.

**The review meeting:** Each review meeting should be held considering the following constraints- *Involvement of people*:

1. Between 3, 4 and 5 people should be involve in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1. Accept the product without any modification.
2. Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3. Accept the product provisional (minor errors are encountered and should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

**Review reporting and record keeping :-**

1. During the FTR, the reviewer actively records all issues that have been raised.
2. At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3. Finally, a formal technical review summary report is prepared.

It answers three questions :-

1. What was reviewed ?
2. Who reviewed it ?
3. What were the findings and conclusions ?

**Review guidelines :-** Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

1. Review the product, not the manufacture (producer).
2. Take written notes (record purpose)
3. Limit the number of participants and insists upon advance preparation.
4. Develop a checklist for each product that is likely to be reviewed.
5. Allocate resources and time schedule for FTRs in order to maintain time schedule.
6. Conduct meaningful training for all reviewers in order to make reviews effective.
7. Reviews earlier reviews which serve as the base for the current review being conducted.
8. Set an agenda and maintain it.
9. Separate the problem areas, but do not attempt to solve every problem notes.
10. Limit debate and rebuttal.