UNIT-6

Quality Management: Quality concepts, Software quality assurance, Software Reviews, Formal technical reviews, Statistical Software quality Assurance, Software reliability.

➢ Quality Concepts

Variation control is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating diskettes. Surely, this cannot be a problem duplicating diskettes is a trivial manufacturing operation, and we can guarantee that exact duplicates of the software are always created.

Quality

Quality of design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design. As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications.
Quality of conformance is the degree to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance.

Quality Control

Variation control may be equated to quality control. But how do we achieve quality control? Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications. This approach views quality control as part of the manufacturing process.
Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction. A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

Quality Assurance

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals. Of course, if the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

Cost of Quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities. Cost of quality studies are conducted to provide a baseline for the

current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison.

Quality costs may be divided into costs associated with prevention, appraisal, and failure. Prevention costs include
• quality planning
• formal technical reviews
• test equipment
• training

Appraisal costs include activities to gain insight into product condition the "first time through" each process. Examples of appraisal costs include
• in-process and interprocess inspection
• equipment calibration and maintenance
• testing

Failure costs are those that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs. Internal failure costs are incurred when we detect a defect in our product prior to shipment. Internal failure costs include
• rework
• repair
• failure mode analysis

External failure costs are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are
• complaint resolution
• product return and replacement
• help line support
• warranty work

➢ Software Quality Assurance(SQA):

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.

➢ Software Review

Software review is an important part of **"Software Development Life Cycle (SDLC)"** that assists software engineers in validating the quality, functionality, and other vital features and components of the software. As mentioned above, it is a complete process that involves testing the software product and ensuring that it meets the requirements stated by the client.

It is systematic examination of a document by one or more individuals, who work together to find & resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC). Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, **"test plans"**, & **"test cases"**.

Why is Software Review Important?

The reasons that make software review an important element of software development process are numerous. It is one such methodology that offers an opportunity to the development team & the client, to get clarity on the project as well as its requirements. With the assistance of software review, the team can verify whether the software is developed as per the requested requirements or not, and make the necessary changes before its release in the market. Other important reasons for Software Review are:

- It improves the productivity of the development team.
- Makes the process of testing time & cost effective, as more time is spent on testing the software during the initial development of the product.
- Fewer defects are found in the final software, which helps reduce the cost of the whole process.
- The reviews provided at this stage are found to be cost effective, as they are identified at the earlier stage, as the cost of rectifying a defect in the later stages would be much more than doing it in the initial stages.
- In this process of reviewing software, often we train technical authors for defect detection process as well as for **"defect prevention process"**.

- It is only at this stage the inadequacies are eliminated.
- Elimination of defects or errors can benefit the software to a great extent. Frequent check of samples of work and identification of small time errors can lead to low error rate.
- As a matter of fact, this process results in dramatic reduction of time taken in producing a technically sound document.

Types of Software Reviews:

There are mainly three types of software reviews, all of which are conducted by different members of the team who evaluate various aspects of the software. Hence, the types of software review are:

1. **Software Peer Review:**

   Peer review is the process of evaluating the technical content and quality of the product and it is usually conducted by the author of the work product, along with some other developers. According to **"Capacity Maturity Model"**, the main purpose of peer review is to provide *"a disciplined engineering practise for detecting or correcting defects in the software artifacts, preventing their leakage into the field operations"*. In short, peer review is performed in order to determine or resolve the defects in the software, whose quality is also checked by other members of the team.

   Types of Peer Review:

   - **"Code Review":** To fix mistakes and to remove vulnerabilities from the software product, systematic examination of the computer source code is conducted, which further improves the quality & security of the product.
   - **"Pair Programming":** This is a type of code review, where two programmers work on a single workstation and develop a code together.
   - **Informal:** As suggested by its name, this is an informal type of review, which is extremely popular and is widely used by people all over the world. Informal

review does not require any documentation, **"entry criteria"**, or a large group of people. It is a time saving process that is not documented.

- **"Walkthrough"**: Here, a designer or developer lead a team of software developers to go through a software product, where they ask question and make necessary comments about various defects & errors. This process differs from **"software inspection"** and technical review in various aspects.

- **Technical Review:** During the process of technical review a team of qualified personnel review the software and examine its suitability to define its intended use as well as to identify various discrepancies.

- **Inspection:** This is a formal type of peer review, wherein experienced & qualified individuals examine the software product for bugs and defects using a defined process. Inspection helps the author improve the quality of the software.

2. **Software Management Review:**

   These reviews take place in the later stages by the management representatives. The objective of this type of review is to evaluate the work status. Also, on the basis of such reviews decisions regarding downstream activities are taken.

3. **Software Audit Reviews:**

   "Software Audit" review or software review is a type of external review, wherein one or more auditors, who are not a part of the development team conduct an independent examination of the software product and its processes to assess their compliance with stated specifications, standards, and other important criterion's. This is done by managerial level people.

➢ Formal Technical Reviews

A formal technical review is a software quality assurance activity performed by software engineers (and others). The objectives of the FTR are

(1) to uncover errors in function, logic, or implementation for any representation of the software;
(2) to verify that the software under review meets its requirements;
(3) to ensure that the software has been represented according to predefined standards;
(4) to achieve software that is developed in a uniform manner; and
(5) to make projects more manageable.
The FTR is actually a class of reviews that includes walkthroughs, inspections, round-robin reviews and other small group technical assessments of software. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended. In the sections that follow, guidelines similar to those for a walkthrough are presented as a representative formal technical review.

The Review Meeting

Regardless of the FTR format that is chosen, every review meeting should abide by the following constraints:
• Between three and five people (typically) should be involved in the review.
• Advance preparation should occur but should require no more than two hours of work for each person.
• The duration of the review meeting should be less than two hours.
At the end of the review, all attendees of the FTR must decide whether to **(1)** accept the product without further modification, **(2)** reject the product due to severe errors (once corrected, another review must be performed), or **(3)** accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required). The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings.

Review Reporting and Record Keeping

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting and a review issues list is produced. In addition, a formal technical review summary report is completed.

A review summary report answers three questions:
1. What was reviewed?
2. Who reviewed it?
3. What were the findings and conclusions?

The review summary report is a single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.
The review issues list serves two purposes:
(1) To identify problem areas within the product and
(2) To serve as an action item checklist that guides the producer as corrections are made. An issues list is normally attached to the summary report.

It is important to establish a follow-up procedure to ensure that items on the issues list have been properly corrected. Unless this is done, it is possible that issues raised can "fall between the cracks." One approach is to assign the responsibility for follow up to the review leader.

Review Guidelines

Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed. A review that is uncontrolled can often be worse that no review at all. The following represents a minimum set of guidelines for formal technical reviews:

1. **Review the product, not the producer.**
   An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Conducted improperly, the FTR can take on the aura of an inquisition. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent should not be to embarrass or belittle. The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.

2. Set an agenda and maintain it. One of the key maladies of meetings of all types is drift. An FTR must be kept on track and on schedule. The review leader is chartered with the responsibility for maintaining the meeting schedule and should not be afraid to nudge people when drift sets in.

3. Limit debate and rebuttal. When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

4. Enunciate problem areas, but don't attempt to solve every problem noted. A review is not a problem-solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

5. Take written notes. It is sometimes a good idea for the recorder to make notes on a wall board, so that wording and priorities can be assessed by other reviewers as information is recorded.

6. Limit the number of participants and insist upon advance preparation. Two heads are better than one, but 14 are not necessarily better than 4. Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance. Written comments should be solicited by the review leader (providing an indication that the reviewer has reviewed the material).

7. Develop a checklist for each product that is likely to be reviewed. A checklist helps the review leader to structure the FTR meeting and helps each reviewer to focus on important issues. Checklists should be developed for analysis, design, code, and even test documents.

8. Allocate resources and schedule time for FTRs. For reviews to be effective, they should

be scheduled as a task during the software engineering process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

9. Conduct meaningful training for all reviewers. To be effective all review participants should receive some formal training. The training should stress both process-related issues and the human psychological side of reviews. Freedman and Weinberg estimate a one-month learning curve for every 20 people who are to participate effectively in reviews.

10. Review your early reviews. Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed should be the review guidelines themselves.

➢ Statistical Software Quality Assurance (SSQA):

Software quality can be achieved through competent analysis, design, coding, and testing, as well as through the application of formal technical reviews, a testing strategy, better control of software work products and the changes made to them, and the application of accepted software engineering standards. In addition, quality can be defined in terms of a broad array of quality factors and measured (indirectly) using a variety of indices and metrics.

Statistical software quality assurance in software engineering involves tracing each defect to its underlying cause, isolating the vital few causes, and moving to correct them.

Steps required to perform statistical SQA :

1. Information about software defects is collected and categorized.

2. An attempt is made to trace each defect to its underlying cause (e.g., non-conformance to specifications, design error, violation of standards, poor communication with the customer).

3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").

4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.

This relatively simple concept represents an important step towards the creation o: an adaptive software engineering process in which changes are made to improve those elements of the process that introduce error.

A Generic Example:

To illustrate the use of statistical methods for software engineering work, assume that a software engineering organization collects information on errors and defects for a period of one year. Some of the errors are uncovered as software is being developed. Others (defects) are encountered after the software has been released to its end users. Although hundreds of different problems are uncovered, all can be tracked to one (or more) of the following causes:

♣ Incomplete or erroneous specifications (IES)

- ♣ Misinterpretation of customer communication (MCC)
- ♣ Intentional deviation from specifications (IDS)
- ♣ Violation of programming standards (VPS)
- ♣ Error in data representation (EDR)
- ♣ Inconsistent component interface (ICI)
- ♣ Error in design logic (EDL)
- ♣ Incomplete or erroneous testing (IET)
- ♣ Inaccurate or incomplete documentation (IID)
- ♣ Error in programming language translation of design (PLT)
- ♣ Ambiguous or inconsistent human/computer interface (HCI)
- ♣ Miscellaneous (MIS)

To apply statistical SQA, the above table is built. The table indicates that IES, MCC, and EDR are the vital few causes that account for 53 percent of all errors. It should be noted, however, that IES, EDR, PLT, and EDL would be selected as the vital few causes if only serious errors are considered. Once the vital few causes are determined, the software engineering organization can begin corrective action.

For example, to correct MCC, you might implement requirements gathering techniques to improve the quality of customer communication and specifications. To improve EDR, you might acquire tools for data modelling and perform more severe data design reviews.

	Total		Serious		Moderate		Minor	
Error	**No.**	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

Table: Data Collection for Statistical Software Quality Assurance (SSQA)

It is important to note that corrective action focuses primarily on the vital few. As the vital few causes are corrected, new candidates pop to the top of the stack. Statistical quality assurance techniques for software have been shown to provide substantial quality improvement.

The application of the statistical SQA and the Pareto principle can be summarized in a single sentence: Spend your time focusing on things that really matter, but first be sure that you understand what really matters.

Six Sigma for software Engineering:

Six Sigma is the most widely used strategy for statistical quality assurance in industry today. Originally popularized by Motorola in the 1980s, the Six Sigma strategy "is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes". The term Six Sigma is derived from six standard deviations. The Six Sigma methodology defines three core steps and two additional important steps:

- ***Define*** customer requirements and deliverables and project goals via well-defined methods of customer communication.
- ***Measure*** the existing process and its output to determine current quality performance.
- ***Analyze*** defect metrics and determine the vital few causes.
- ***Improve*** the process by eliminating the root causes of defects.
- ***Control*** the process to ensure that future work does not reintroduce the causes of defects.
- ***Design*** the process to avoid the root causes of defects and to meet customer requirements.
- ***Verify*** that the process model will, in fact, avoid defects and meet customer requirements.

➢ Software reliability:

Software reliability is defined in statistical term as **"the probability of failure-free operation of a computer program in a specified environment for a specified time".** Reliability is a customer-oriented view of software quality.
It relates to operation rather than design of the program, and hence it is dynamic rather than static.

Whenever software reliability is discussed, a pivotal question arises: What is meant by the term failure? In the context of any discussion of software quality and reliability, failure is non-conformance to software requirements. Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic. One failure can be corrected within seconds while another requires weeks or even months to correct. Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

❖ Measures of Reliability and Availability:

1. Mean time to failure (MTTF):

● MTTF is the time between two successive failures, averaged over a large number of failures.

- To measure MTTF, we can record the failure data for n failures.
- It is important to note that only run time is considered in the time measurements.

2. Mean time to repair (MTTR):

- Once failure occurs, sometime is required to fix the error.
- MTTR measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean time between failure (MTBF):

- The MTTF and MTTR metrics can be combined to get the MTBF metric: MTBF=MTTF+MTTR.
- Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours.

Early work in software reliability attempted to extrapolate the mathematics of hard-ware reliability theory to the prediction of software reliability. Most hardware- related reliability models are predicated on failure due to wear rather than failure due to design defects. In hardware, failures due to physical wear the effects of temperature, corrosion, shock) are more likely than a design-related failure. Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems.

There has been debate over the relationship between key concepts in hardware reliability and their applicability to software (e.g., [LIT89], [ROO90]). Although an irrefutable link has yet be established, it is worthwhile to consider a few simple concepts that apply to both system elements.
If we consider a computer-based system, a simple measure of reliability is **mean-time-between-failure (MTBF)**, where
MTBF = MTTF + MTTR
The acronyms MTTF and MTTR are **mean-time-to-failure** and **mean-time-to-repair**, respectively.

In addition to a reliability measure, we must develop a measure of availability.
Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

Availability = [MTTF/(MTTF + MTTR)] 100%

The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

❖ Software Safety:

Before software was used in safety critical systems, they were often controlled by conventional (nonprogrammable) mechanical and electronic devices. System safety techniques are designed to cope with random failures in these [nonprogrammable] systems. Human design errors are not considered since it is assumed that all faults caused by human errors can be avoided completely or removed prior to delivery and operation.

Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk. For example, some of the hazards associated with a computer-based cruise control for an automobile might be

• causes uncontrolled acceleration that cannot be stopped

• does not respond to depression of brake pedal (by turning off)

• does not engage when switch is activated

• slowly loses or gains speed

Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence. To be effective, software must be analysed in the context of the entire system. For example, a subtle user input error (people are system components) may be magnified by a software fault to produce control data that improperly positions a mechanical device. If a set of external environmental conditions are met (and only if they are met), the improper position of the mechanical device will cause a disastrous failure.