

CHAPTER – 4

SOFTWARE QUALITIES AND SOFTWARE QUALITY ASSURANCE

SOFTWARE QUALITIES:

The term software quality describes to what extent is the software good rather best to be implemented for the purpose it has been proposed and developed. It is very important to maintain the software quality because every individuals or every companies are always willing to use the best system.

Thus, quality refers to the measurable characteristics things that we are able to compare to known standards such as length, color, electrical properties and malleability. Based upon the measurable characteristics of an item, software quality can be classified into two kinds:

1. Quality of Design:

It refers to the characteristics that designer specify for an item. The quality of software on the design basis is influenced by the grade of the materials, tolerances and performance specifications. Use of higher grade materials contribute to higher tolerances and thus specify greater levels of performances, finally increasing the design quality of product.

2. Quality of Conformance:

It is the degree to which the design specifications are followed during manufacturing. The greater the degree of conformances the higher the level of quality of conformances will be.

SOFTWARE QUALITY FACTORS:

Developing methods that can produce high-quality software is another fundamental goal of software engineering. Factors that affects the quality of software are explained below:

1. Correctness:

Correctness is the extent to which a program satisfies its specifications.

2. Reliability:

Reliability is the property that defines how well the software meets its requirements.

3. Efficiency:

Efficiency is a factor relating to all issues in the execution of software; it includes considerations such as response time, memory requirement, and throughput.

4. Usability:

Usability, or the effort required locating and fixing errors in operating programs.

5. Portability:

Portability is the effort required to transfer the software from one configuration to another.

6. Reusability:

Reusability is the extent to which parts of the software can be reused in other related applications.

7. Maintainability:

Maintainability is the effort required to maintain the system in order to check the quality.

8. Testability:

Testability is the effort required to test to ensure that the system or a module performs its intended function.

9. Flexibility:

Flexibility is the effort required to modify an operational program.

SOFTWARE QUALITY ASSURANCE (SQA):

Software Quality Assurance is the process of providing the management with the data necessary to be informed about product quality, thus getting confidence that the product quality is meeting its goal. It provides the management with data that identify problems as a result the management will be conscious and responsive towards the problems and apply the necessary resources to solve the quality issues.

Various persons like software engineers, project managers, customers, sales persons and individuals who serve within SQA group are responsible for the software quality assurance. The SQA group serves as the customer's representative i.e. people who perform SQA must look at the software from customers point of view.

SOFTWARE QUALITY ASSURANCE ACTIVITIES:

Some of the SQA activities performed by the SQA group are as follows:

1. Prepares the SQA plan for the project.
2. Participates in the development of the project's software process description.
3. Review software engineering activities.
4. They audits designated software work products.
5. They ensure that deviation's in documented and handled according to a documented procedure.
6. Records any non-compliance reports to senior management. Non-compliance items are tracked until they are resolved.





SOFTWARE QUALITY STANDARDS:

Many organizations around the globe develop and implement different standards to improve the quality needs of their software.

1. INTERNATIONAL STANDARD ORGANIZATION:

❖ ISO/IEC 9126:

This standard deals with the following aspects to determine the quality of a software application:

-  Quality model
-  External metrics
-  Internal metrics
-  Quality in use metrics

This standard presents some set of quality attributes for any software such as:

-  Functionality
-  Reliability
-  Usability
-  Efficiency
-  Maintainability
-  Portability

❖ ISO/IEC 9241-11:






Part 11 of this standard deals with the extent to which a product can be used by specified users to achieve specified goals with Effectiveness, Efficiency and Satisfaction in a specified context of use.

This standard proposed a framework that describes the usability components and the relationship between them. In this standard, the usability is considered in terms of user performance and satisfaction. According to ISO 9241-11, usability depends on context of use and the level of usability will change as the context changes.





❖ ISO/IEC 25000:2005:

ISO/IEC 25000:2005 is commonly known as the standard that provides the guidelines for Software Quality Requirements and Evaluation (SQuaRE). This standard helps in organizing and enhancing the process related to software quality requirements and their evaluations. In reality, ISO-25000 replaces the two old ISO standards, i.e. ISO-9126 and ISO-14598.

SQuaRE is divided into sub-parts such as:

-  ISO 2500n - Quality Management Division
-  ISO 2501n - Quality Model Division
-  ISO 2502n - Quality Measurement Division
-  ISO 2503n - Quality Requirements Division
-  ISO 2504n - Quality Evaluation Division

The main contents of SQuaRE are:

-  Terms and definitions
-  Reference Models
-  General guide
-  Individual division guides

- ✚ Standard related to Requirement Engineering (i.e. specification, planning, measurement and evaluation process)
- ❖ **ISO/IEC 12119:**

This standard deals with software packages delivered to the client. It does not focus or deal with the clients' production process. The main contents are related to the following items:

- ✚ Set of requirements for software packages.
- ✚ Instructions for testing a delivered software package against the specified requirements.

2. SOFTWARE ENGINEERING INSTITUTE:

Software Engineering Institute at Carnegie-Mellon University; initiated by the U.S. Defense Department to help improve software development processes.

CMM = 'Capability Maturity Model', developed by the SEI. It's a model of 5 levels of organizational 'maturity' that determine effectiveness in delivering quality software. It is geared to large organizations such as large U.S. Defense Department contractors. However, many of the QA processes involved are appropriate to any organization, and if reasonably applied can be helpful. Organizations can receive CMM ratings by undergoing assessments by qualified auditors.

Level 1: characterized by chaos, periodic panics, and heroic efforts required by individuals to successfully complete projects. Few if any processes in place; successes may not be repeatable.

Level 2: software project tracking, requirements management, realistic planning, and configuration management processes are in place; successful practices can be repeated.

Level 3: standard software development and maintenance processes are integrated throughout an organization; a Software Engineering Process Group is in place to oversee software processes, and training programs are used to ensure understanding and compliance.

Level 4: metrics are used to track productivity, processes, and products. Project performance is predictable, and quality is consistently high.

Level 5: the focus is on continuous process improvement. The impact of new processes and technologies can be predicted and effectively implemented when required.

SOFTWARE REVIEW:

Software review is the meeting conducted by technical people for technical people. It can thus be defined as technical assessment of a work product created during the software engineering process. It is a basis for SQA mechanism.

In other words, software review is the way of using the diversity of a group of people to:

- ✚ Point out needed improvement in the software.
- ✚ Confirm those portions of the software in which improvement is either not desired or not needed.
- ✚ Conforms for rework.

TYPES OF SOFTWARE REVIEW:

a. Informal Review:

Conducted on an as-needed i.e. informal agenda. The date and the time of the agenda for the Informal Review will not be addressed in the Project Plan. The developer chooses a review panel and provides and/or presents the material to be reviewed. The material may be as informal as a computer listing or hand-written documentation.

b. Formal Review:

Conducted at the end of each life cycle phase i.e. Formal Agenda. The agenda for the formal review must be addressed in the Project Plan. The acquirer of the software appoints the formal review panel or board, who may make or affect a go/no-go decision to proceed to the next step of the life cycle. The material must be well prepared *Example:* Formal reviews include the Software Requirements Review, the Software Preliminary Design Review, the Software Critical Design Review, and the Software Test Readiness Review.

For conducting Formal Technical Review (FTR) we can follow the following guidelines:

1. Review the product, not the producer.
2. Set an agenda and maintain it.
3. Limit debate.
4. Enunciate problem areas, but does not attempt to solve every problem noted.
5. Take written notes.
6. Limit the number of participants and insist upon advance preparation.
7. Schedule review as project tasks.
8. Review your early reviews.
9. Record and report all review results.

COST IMPACT OF SOFTWARE DEFECTS:

Since the defect/ fault/ error imply a quality problem that is discovered by software engineers (or others) before the software is released to the end user (or to another framework activity to the software process).

The primary objective of technical review is to find errors during the process so that they do not become defected after release of the software. The clear benefits of technical review is the early discovery of errors so that they do not propagate to the next step in software process.

A number of industry studies indicate that design activities introduce between 50% and 65% of all errors during software process. However, review technique have been shown up to 75% effective in covering design flows.

DEFECT AMPLIFICATION AND REMOVAL:

A defect amplification model can be used to illustrate the generation and detection of errors during the design and code generation actions of a software.

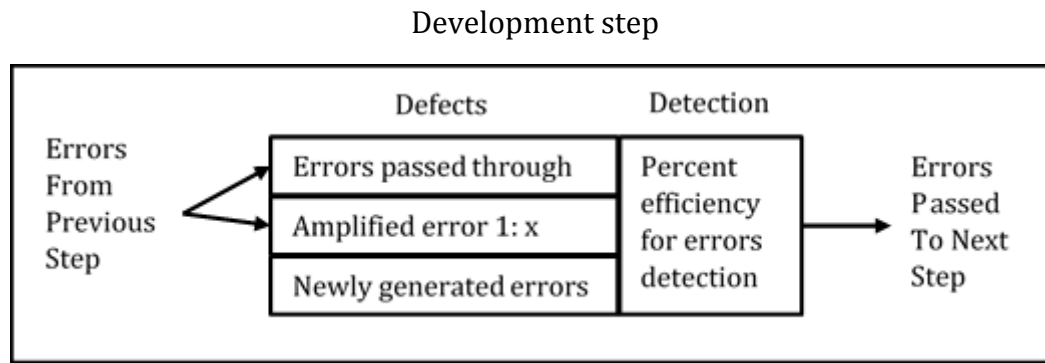


Fig: Defect Amplification Model

The box represents a software engineering action. During the action, errors may be introduced. Review may fail to uncover newly generated errors and errors from previous step resulting in some numbers of errors that are passed through. In some case, errors from previous steps are amplified (by amplification factor x) by current work.

The box sub-divisions represents each of these characteristics and the percent of efficiency for detecting errors a function of thoroughness of review.

FORMAL TECHNICAL REVIEW (FTR):

A FTR is a software quality control activity performed by software engineers and others. The objectives of 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 software has been represented according to predefined standards.
4. To achieve software that is developed in a uniform manner.
5. To make projects more manageable.

In addition, the FTR server as a training ground, enabling junior engineers to observe different approaches to software analysis, design and implementation. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled and attended.

THE REVIEW MEETING:

Every review meeting should abide by the following constraints:

- ✚ Between 3 to 5 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 review meeting should be less than two hours.

Given these constraints, it should be clear that an FTR focuses on a specific (and small) part of overall software, so that the FTR has a higher likelihood of uncovering errors. The focus of FTR

is on a work product (example: a portion of requirement modes, a detailed component design source code, etc.). The individual who has developed the work product (producer) informs the project leader that the product is complete and review is required.

The project leaders contacts a review leader, who evaluates the product for readiness, generates copies of product material and distribute them to reviewers for advance preparation. Each review is expected to spend 1 to 2 hours, reviewing the product, making notes, and otherwise become familiar to work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for next day.

The review meeting is attended by the review leader and all the reviewers and the producer. One of the reviewer takes on the role of recorder i.e. who records (in writing) all important issues raised during review.

The FTR begins with the introduction of the agenda and a brief introduction by the producer. When valid problems or errors are discovered the recorder note each. At the end of review, all attendees of the FTR must decide whether to:

- ✚ Accept the product without further modifications.
- ✚ Reject the product due to server errors (once corrected another review must be performed).
- ✚ Accept the product provisionally (minor errors have been encountered and must be correct, but no additional review will be required)

After complete, all reviewer needs to sign off.

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 the review issues list is produced. In addition a FTR 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 the conclusions?

The review summary report is 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 issue list provides/serves two purposes:

- i. To identify problem areas within the product and
- ii. To serve as an action item check list that guides the producer as corrections are made. An issue list is normally attached to the summary report.

FORMAL APPROACHES TO SQA:

Software quality can be achieved through competent software engineering practice as well as through the application of technical review, testing strategies, better control of software work products and the changes made to them, standard accepted, etc. But software engineering community has argued that a more formal approaches to SQA is required.

It can be argued that a computer program is a mathematical object. A syntax and semantics can be defined for every programming language and approach to specification of software requirements is available. So, different approaches are necessary for Software Quality Assurance.

PROOF OF CORRECTNESS:

- ✚ Performed using verification and validation in static way.
- ✚ Using testing strategies in dynamic aspects.
- ✚ Comparing output deviations.
- ✚ Follow of standards.

STATISTICAL SOFTWARE QUALITY ASSURANCE:

Statistical SQA reflects the growing trend throughout industry to become more quantitative about quality. Statistical quality assurance for software implies the following steps:

- ✚ Information about software errors and defects is collected and categorized.
- ✚ An attempt is made to trace each error, and defect to its underlying cause.
- ✚ Using the pareto principle (80% of the defects can be trace to 20% of possible causes) isolate the 20% (vital few).
- ✚ Once the vital few causes have been identified move to correct the problems that have caused the errors and defects.

Example:

Error	Total Number %		Serious Number %		Moderate Number %		Minor Number %	
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	35	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

Total	942	100	128	100	379	100	435	100
--------------	------------	------------	------------	------------	------------	------------	------------	------------

- ✚ IES: Incomplete or Erroneous Specifications
- ✚ MCC: Misinterpretation of Customer Communication
- ✚ IDS: Intentional Deviation from Specs
- ✚ VPS: Violation of Programming Standard
- ✚ EDR: Error in Data Representation
- ✚ EDL: Error in Design Logic

CLEAN ROOM ENGINEERING:

- ✚ It is a software development philosophy that is based on avoiding software defects by using formal methods of development and a rigorous inspection process.
- ✚ The name “clean room” was derived by analogy with semiconductor fabrication units. In these units, defects are avoided by manufacturing in an ultra-clean atmosphere.
- ✚ The objective of this approach is zero-defect software development.
- ✚ The clean room approach to software development is based on five keys strategies:
 - a. Formal Specification
 - b. Incremental Development
 - c. Structured Programming
 - d. Static Verification
 - e. Statistical Testing of System.