Module-V
Software Quality
&
Metrics

Syllabus: Software Quality, Software Quality Management System (SQMS),ISO 9000, SEI Capability Maturity, **Model Product metrics:** Metrics for Requirements Model, Metrics for Design Model, Metrics for source code ,Metrics for testing, Metrics for maintenance

2.Discuss about Software Quality and Explain Software Quality Management System

(Or)

9.What is Software Quality? Explain about metrics for software quality

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.

The modern view of a quality associated with quality methods:

1. 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.

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

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

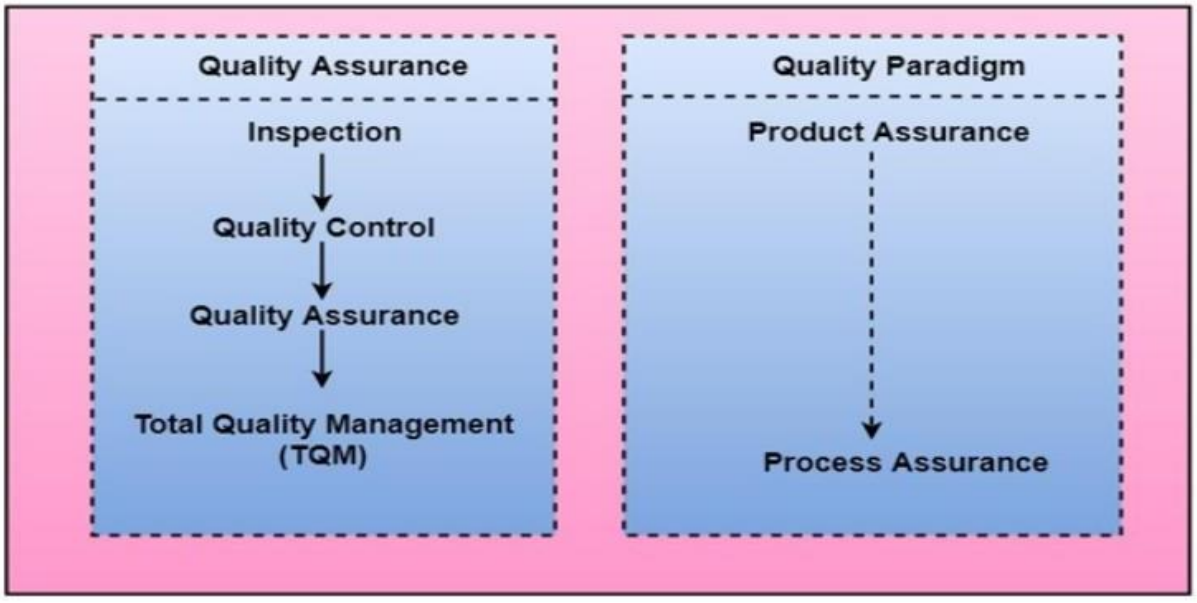
4. Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

5. 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.

6. Flexibility: Software flexibility is one of the properties that indicate if the software is easy to change. Flexible software can easily adapt to user requirement and/or environment changes during the software development period or after the software is deployed.

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 system and corresponding shift in the quality paradigm



SOFTWARE QUALITY ATTRIBUTES:

1. Functional suitability
2. Reliability
3. Operability
4. Performance efficiency
5. Security
6. Compatibility
7. Maintainability
8. Transferability

SOFTWARE QUALITY CHARACTERISTICS:

1. Effectiveness
2. Efficiency
3. Satisfaction
4. Safety
5. Usability

Software Quality Management System(SQMS):

→Software Quality Management ensures that the required level of quality is achieved by submitting improvements to the product development process. SQA aims to develop a culture within the team and it is seen as everyone's responsibility.

→Software Quality management should be independent of project management to ensure independence of cost and schedule adherences. It directly affects the process quality and indirectly affects the product quality.

Activities of Software Quality Management:

1. Quality Assurance(QA)- Aims at developing Organizational procedures and standards for quality at Organizational level.

2. Quality Planning(QP) - Select applicable procedures and standards for a particular project and modify as required to develop a quality plan.

3. Quality Control(QC) - Ensure that best practices and standards are followed by the software development team to produce quality products.

Quality Assurance(QA): Activities include -

→Identifying standards if any used in software development processes.

→To conduct conventional processes, such as quality reviews.

→Perform in-process test data recording procedures.

→Encouraging documentation process standards.

Quality Control(QC): Activities include -

→A follow-up review of software to ensure any required changes detailed in the previous testing addressed.
→Release testing of software with proper documentation of the testing process
→Apply software measurement and metrics for assessment.
→Examine software and associated documentation for non-conformance with standards.

Quality Planning(QP): Activities include -

Predictability

Software quality drives predictability. Predictability decreases as re-work grows, and lower quality product increases. Do it once and right, and there will be less variation in productivity, less re-work, and better performance overall. Products get delivered on time. Poor quality is hugely more difficult to manage.

Reputation

A significant, solid reputation is hard to establish and easy to lose, but when the company has it, it's a powerful business driver. A few mistakes and fame can be gone, creating significant obstacles to sales, and consequently, your bottom line.

Employee Morale

The happiest and productive employees have pride in the work. Enabling employees to build software will drive a higher level of productivity and morale. Inferior products, lots of re-work, unhappy customers and difficulty making deadlines have the opposite effect, leading to a less productive workforce and high turnover.

Customer Satisfaction

A quality product satisfies the customer. A satisfied customer comes back for provides positive referrals. Customer loyalty heavily drives by the quality of the software produced and the service offer. With social media channels such as Facebook and Twitter, positive references can spread quickly. Poor quality and dissatisfaction can also rapidly communicate, if not even quicker than the good ones.

Bottom Line

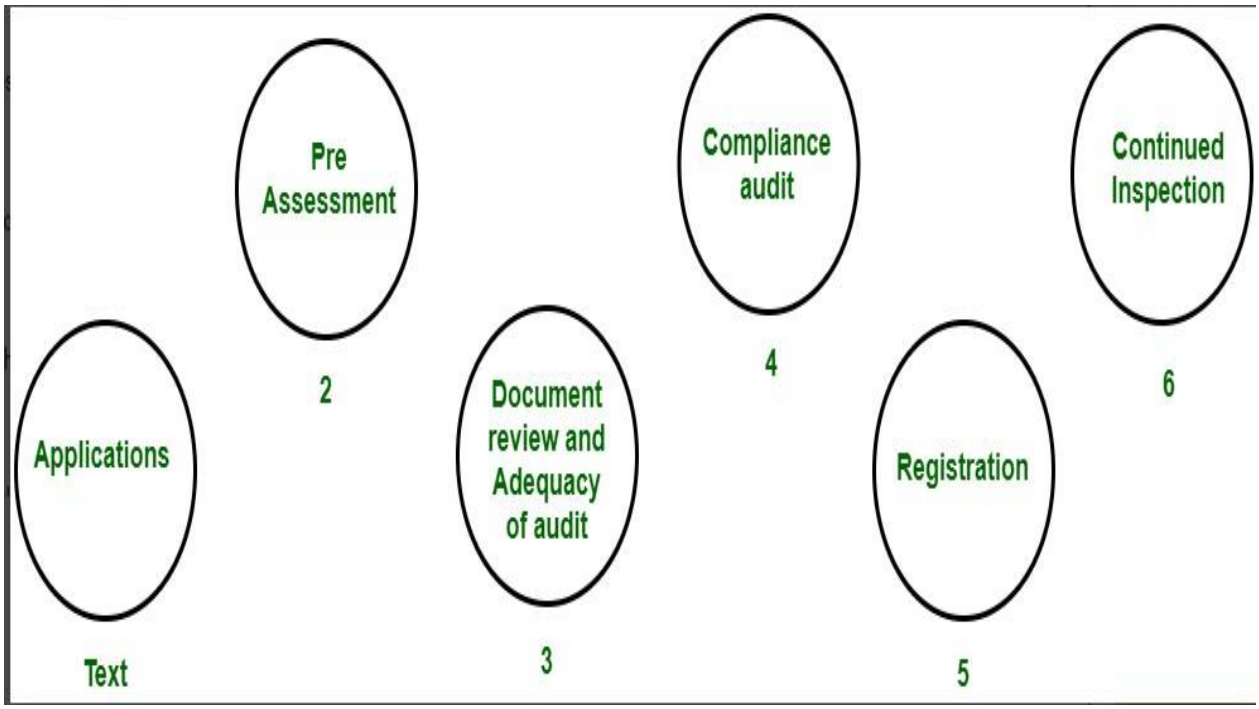
Predictable and productive performance, happy employees, a stellar reputation, and satisfied customers are the formula for a successful software business. It all drives the bottom line. Quality impact various areas of software development projects.

ISO 9000:

The International organization for Standardization is a world wide federation of national standard bodies.

The **International standards organization (ISO)** is a standard which serves as a for contract between independent parties. It specifies guidelines for development of **quality system**.
Quality system of an organization means the various activities related to its products or services.
Standard of ISO addresses to both aspects i.e. operational and organizational aspects which includes responsibilities, reporting etc.
An ISO 9000 standard contains set of guidelines of production process without considering product itself.

ISO 9000 Certification:



There are several reasons why software industry must get an ISO certification.
Some of reasons are as follows :
1. This certification has become a standards for international bidding.
2. It helps in designing high-quality repeatable software products.
3. It emphasis need for proper documentation.
4. It facilitates development of optimal processes and totally quality measurements.

Features of ISO 9001 Requirements:

Document control –
All documents concerned with the development of a software product should be properly managed and controlled.
Planning –
Proper plans should be prepared and monitored.
Review –
For effectiveness and correctness all important documents across all phases should be independently checked and reviewed .
Testing –
The product should be tested against specification.
Organizational Aspects –
Various organizational aspects should be addressed e.g., management reporting of the quality team.

Advantages of ISO 9000 Certification:

Some of the advantages of the ISO 9000 certification process are :
→Business ISO-9000 certification forces a corporation to specialize in "how they are doing business". Each procedure and work instruction must be documented and thus becomes a springboard for continuous improvement.
→Employees morale is increased as they're asked to require control of their processes and document their work processes
→Better products and services result from continuous improvement process.
→Increased employee participation, involvement, awareness and systematic employee training are reduced problems.

Software Engineering Institute Capability Maturity Model (SEICMM):

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
1. It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
2. It also provides guidelines to further enhance the maturity of the process used to develop those software products.
3. It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

4. This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
5. Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Key Process Areas (KPA's):

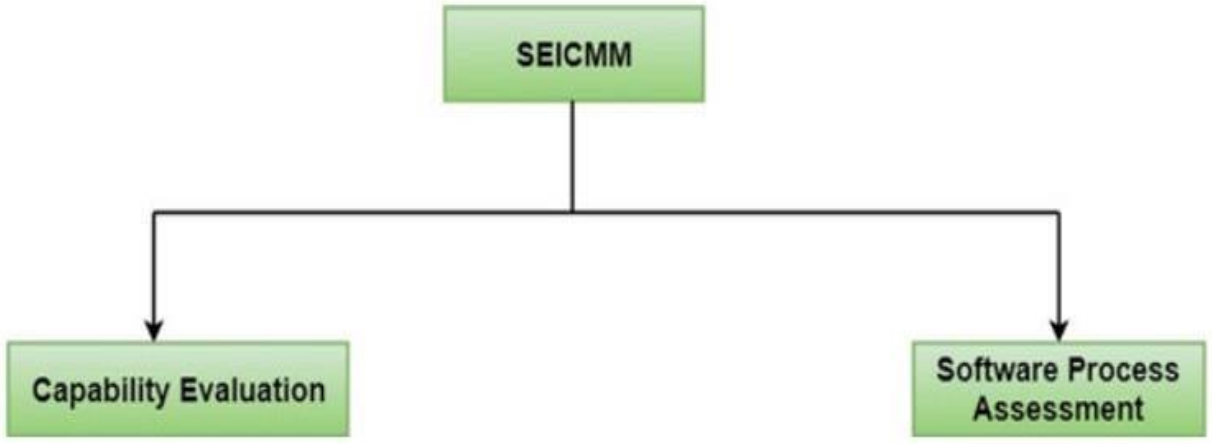
Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

Shortcomings of SEI/CMM:

→It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.
→It only helps if it is put into place early in the software development process.
→It has no formal theoretical basis and in fact is based on the experience of very knowledgeable people.
→It does not have good empirical support and this same empirical support could also be constructed to support other models.

Methods of SEICMM

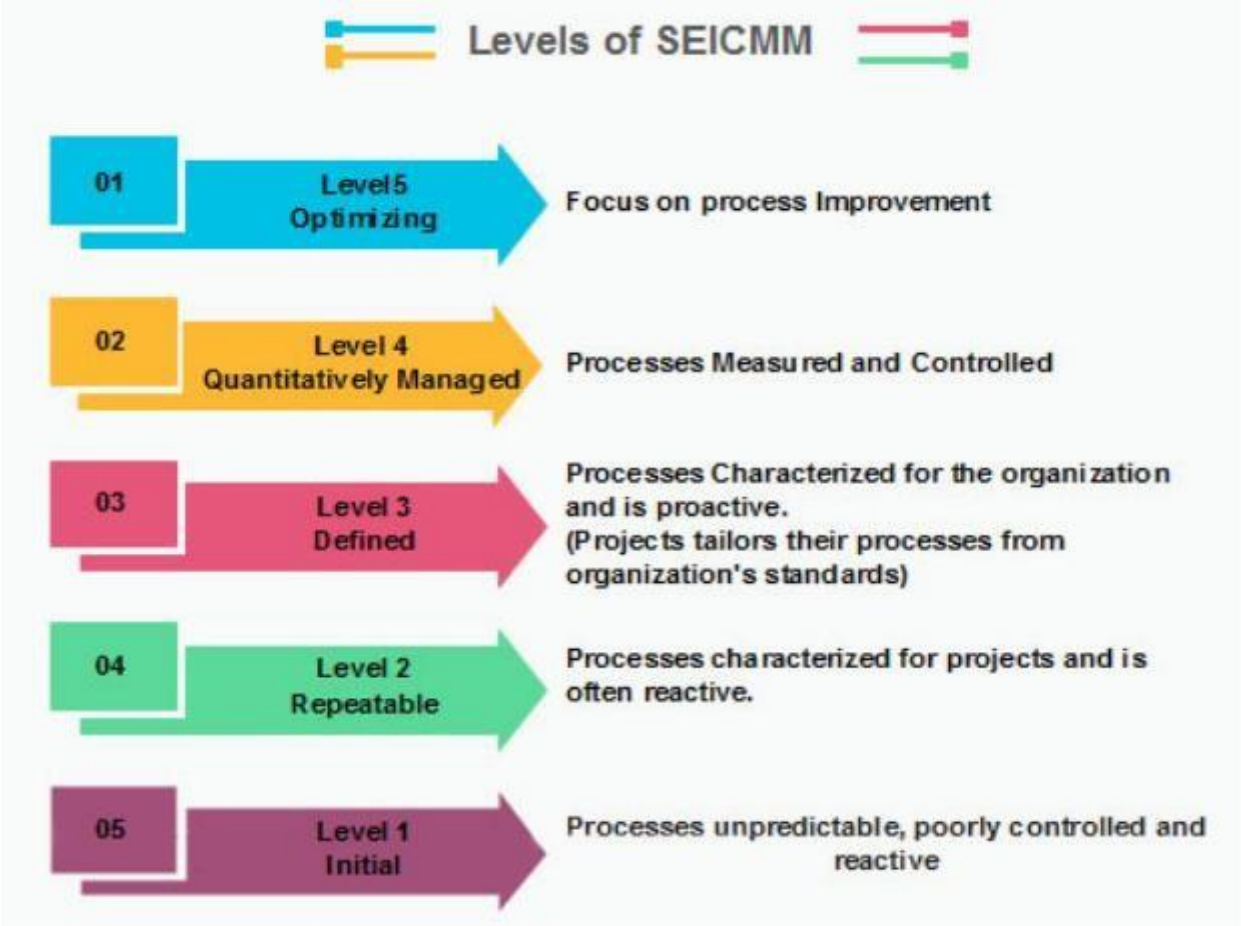
There are two methods of SEICMM:



A) Capability Evaluation: Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work. Therefore, the results of the software process capability assessment can be used to select a contractor.

B) Software Process Assessment: Software process assessment is used by an organization to improve its process capability. Thus, this type of evaluation is for purely internal use.

SEI CMM categorized software development industries into the following five maturity levels. The various levels of SEI CMM have been designed so that it is easy for an organization to build its quality system starting from scratch slowly.



Level-1: Initial –

→No KPA's defined.

→Processes followed are Adhoc and immature and are not well defined.

→Unstable environment for software development.

→No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

Focuses on establishing basic project management policies.
Experience with earlier projects is used for managing new similar natured projects.
Project Planning- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
Configuration Management- The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
Requirements Management- It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
Subcontract Management- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
Software Quality Assurance- It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –

At this level, documentation of the standard guidelines and procedures takes place. It is a well-defined integrated set of project-specific software engineering and management processes.
Peer Reviews- In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
Intergroup Coordination- It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
Organization Process Definition- Its key focus is on the development and maintenance of the standard development processes.
Organization Process Focus- It includes activities and practices that should be followed to improve the process capabilities of an organization.
Training Programs- It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

At this stage, quantitative quality goals are set for the organization for software products as well as software processes.

The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

Software Quality Management- It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.

Quantitative Management- It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.

Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.

Process Change Management- Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.

Technology Change Management- It consists of the identification and use of new technologies to improve product quality and decrease product development time.

Defect Prevention- It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

Key Process Areas (KPA) of a software organization:

Except for SEI CMM level 1, each maturity level is featured by several Key Process Areas (KPAs) that contains the areas an organization should focus on improving its software process to the next level.

SEI CMM provides a series of key areas on which to focus to take an organization from one level of maturity to the next. Thus, it provides a method for gradual quality improvement over various stages. Each step has been carefully designed such that one step enhances the capability already built up

CMM Level	Focus	Key Process Areas
1. Initial	Competent People	NO KPA'S
2. Repeatable	Project Management	Software Project Planning software Configuration Management
3. Defined	Definition of Processes	Process definition Training Program Peer reviews
4. Managed	Product and Process quality	Quantitiative Process Metrics Software Quality Management
5. Optimizing	Continuous Process improvement	Defect Prevention Process change management Technology change management

The focus of each SEI CMM level and the Corresponding Key process areas.

5.2-Model Product metrics

1.Define Software Product and Explain about Product Metrics in detailed ?
(Or)
8.Discuss about metrics for Process and Products

Product metrics are software product measures at any stage of their development, from requirements to established systems. Product metrics are related to software features only.

Product metrics fall into two classes:

1. Dynamic metrics
2. Static metrics

→Dynamic metrics that are collected by measurements made from a program in execution.

→Static metrics that are collected by measurements made from system representations such as design, programs, or documentation.

Dynamic metrics help in assessing the efficiency and reliability of a program while static metrics help in understanding, understanding and maintaining the complexity of a software system.

Dynamic metrics are usually quite closely related to software quality attributes. It is relatively easy to measure the execution time required for particular tasks and to estimate the time required to start the system. These are directly related to the efficiency of the system failures and the type of failure can be logged and directly related to the reliability of the software.

On the other hand, static matrices have an indirect relationship with quality attributes. A large number of these matrices have been proposed to try to derive and validate the relationship between the complexity, understandability, and maintainability. several static metrics which have been used for assessing quality attributes, given in table of these, program or component length and control complexity seem to be the most reliable predictors of understandability, system complexity, and maintainability.

S. No.	Software Metric	Description
(1)	Fan-in/Fan-out	Fan-in is a measure of the number of functions that call some other function (say X). Fan-out is the number of functions which are called by function X.

(2)	Length of code	This is measure of the size of a program. Generally, the large the size of the code of a program component, the more complex and error-prone that component is likely to be.

(3)	Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability.
(4)	Length of identifiers	This 0is a measure of the average length of distinct identifier in a program. The longer the identifiers, the more understandable the program.
(5)	Depth of conditional nesting	This is a measure of the depth of nesting of if statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
(6)	Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document may be to understand.

Model Product metrics: Defined as per the SDLC

→Metrics for Requirements Model
→Metrics for Design Model
→Metrics for Source code
→Metrics for Testing
→Metrics for Maintenance

Software metrics can be classified into three categories −

Product metrics − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.

Process metrics − These characteristics can be used to improve the development and maintenance activities of the software.

Project metrics − This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

* Analysis, design, and code model
* Potency of test cases
* Overall quality of the software under development.

Various metrics formulated for products in the development process are listed below.

Metrics for analysis model: These address various aspects of the analysis model such as system functionality, system size, and so on.

Metrics for design model: These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.

Metrics for source code: These assess source code complexity, maintainability, and other characteristics.

Metrics for testing: These help to design efficient and effective test cases and also evaluate the effectiveness of testing.

Metrics for maintenance: These assess the stability of the software product.

3.Discuss about the Metrics for Requirements Model ?

1.Metrics for the Requirements/Analysis Model

There are only a few metrics that have been proposed for the analysis model. However, it is possible to use metrics for project estimation in the context of the analysis model. These metrics are used to examine the analysis model with the objective of predicting the size of the resultant system. Size acts as an indicator of increased coding, integration, and testing effort; sometimes it also acts as an indicator of complexity involved in the software design. Function point and lines of code are the commonly used methods for size estimation.

a) Function Point (FP) Metric

The function point metric, which was proposed by A.J Albrecht, is used to measure the functionality delivered by the system, estimate the effort, predict the number of errors, and estimate the number of components in the system. Function point is derived by using a relationship between the complexity of software and the information domain value. Information domain values used in function point include the number of external inputs, external outputs, external inquire, internal logical files, and the number of external interface files.

b) Lines of Code (LOC)

Lines of code (LOC) is one of the most widely used methods for size estimation. LOC can be defined as the number of delivered lines of code, excluding comments and blank lines. It is highly dependent on the programming language used as code writing varies from one programming language to another. Fur example, lines of code written (for a large program) in assembly language are more than lines of code written in C++.

From LOC, simple size-oriented metrics can be derived such as errors per KLOC (thousand lines of code), defects per KLOC, cost per KLOC, and so on. LOC has also been used to predict program complexity, development effort, programmer performance, and so on. For example, Hasltead proposed a number of metrics, which are used to calculate program length, program volume, program difficulty, and development effort.

4.Discuss about the Metrics for Design Model

2. Metrics for Software Design:

The success of a software project depends largely on the quality and effectiveness of the software design. Hence, it is important to develop software metrics from which meaningful indicators can be derived.

With the help of these indicators, necessary steps are taken to design the software according to the user requirements. Various design metrics such as architectural design metrics, component-level design metrics, user-interface design metrics, and

metrics for object-oriented design are used to indicate the complexity, quality, and so on of the software design.

a) Architectural Design Metrics

These metrics focus on the features of the program architecture with stress on architectural structure and effectiveness of components (or modules) within the architecture. In architectural design metrics, three software design complexity measures are defined, namely, structural complexity, data complexity, and system complexity.

In hierarchical architectures (call and return architecture), say module 'j', structural complexity is calculated by the following equation.

$S(j) = f^2_{out}(j)$

Where

$f_{out}(j)$ = fan-out of module 'j' [Here, fan-out means number of modules that are subordinating module j].

b) Complexity Metrics

Different types of software metrics can be calculated to ascertain the complexity of program control flow. One of the most widely used complexity metrics for ascertaining the complexity of the program is cyclomatic complexity.

5.Discuss about the Metrics for source code

3. Metrics for Coding:

Halstead proposed the first analytic laws for computer science by using a set of primitive measures, which can be derived once the design phase is complete and code is generated. These measures are listed below.

n_1 = number of distinct operators in a program

n_2 = number of distinct operands in a program

N_1 = total number of operators

N_2= total number of operands.

By using these measures, Halstead developed an expression for overall program length, program volume, program difficulty, development effort, and so on.

Program length (N) can be calculated by using the following equation.

$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$.

Program volume (V) can be calculated by using the following equation.

$V = N \log_2 (n_1+n_2)$.

Note that program volume depends on the programming language used and represents the volume of information (in bits) required to specify a program. Volume ratio (L)can be calculated by using the following equation.

L = Volume of the most compact form of a program
Volume of the actual program

6.Explain in detail about metrics for Testing ?

4.Metrics for Software Testing:

Majority of the metrics used for testing focus on testing process rather than the technical characteristics of test. Generally, testers use metrics for analysis, design, and coding to guide them in design and execution of test cases.

Function point can be effectively used to estimate testing effort. Various characteristics like errors discovered, number of test cases needed, testing effort, and so on can be determined by estimating the number of function points in the current project and comparing them with any previous project.

Metrics used for architectural design can be used to indicate how integration testing can be carried out. In addition, cyclomatic complexity can be used effectively as a metric in the basis-path testing to determine the number of test cases needed.

Halstead measures can be used to derive metrics for testing effort. By using program volume (V) and program level (PL),Halstead effort (e)can be calculated by the following equations.

$e = V/ PL$

Where

$PL = 1/ [(n_1/2) * (N_2/n_2)]$

7.Discuss about metrics for maintenance

5.Metrics for maintenance:

When development of a software product is complete and it is released to the market, it enters the maintenance phase of its life cycle. During this phase the defect arrivals by time interval and customer problem calls (which may or may not be defects) by time interval are the de facto metrics. However, the number of defect or problem arrivals is largely determined by the development process before the maintenance phase. Not much can be done to alter the quality of the product during this phase. Therefore, these two de facto metrics, although important, do not reflect the quality of software maintenance. What can be done during the maintenance phase is to fix the defects as soon as possible and with excellent fix quality. Such actions, although still not able to improve the defect rate of the product, can improve customer satisfaction to a large extent. The following metrics are therefore very important:

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