## 1. Write major differences between Validation and Verification activities.

*The terms Verification and Validation are commonly used in software engineering to mean two different types of analysis. The usual definitions are:*

- *Validation: Are we building the right system?*
- *Verification: Are we building the system, right?*

In other words, validation is concerned with checking that the system will meet the customer's actual needs, while verification is concerned with whether the system is well-engineered, error-free, and so on. Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. The distinction between the two terms is largely to do with the role of specifications. Validation is the process of checking whether the specification captures the customer's needs, while verification is the process of checking that the software meets the specification.

Verification includes all the activities associated with the producing high-quality software: testing, inspection, design analysis, specification analysis, and so on. It is a relatively objective process, in that if the various products and documents are expressed precisely enough, no subjective judgements should be needed in order to verify software.

In contrast, validation is an extremely subjective process. It involves making subjective assessments of how well the (proposed) system addresses a real-world need. Validation includes activities such as requirements modelling, prototyping and user evaluation.

In a traditional phased software lifecycle, verification is often taken to mean checking that the products of each phase satisfy the requirements of the previous phase. Validation is relegated to just the beginning and ending of the project: requirements analysis and acceptance testing. This view is common in many software engineering textbooks, and is misguided. It assumes that the customer's requirements can be captured completely at the start of a project, and that those requirements will not change while the software is being developed. In practice, the requirements change throughout a project, partly in reaction to the project itself: the development of new software makes new things possible. Therefore, both validation and verification are needed throughout the lifecycle.

Finally, V&V is now regarded as a coherent discipline:" Software V&V is a system engineering discipline which evaluates the software in a systems context, relative to all system elements of hardware, users, and other software".

## 2. Write a short note on following quality standards

### A. SEI CMM

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

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. It also provides guidelines to further enhance the maturity of the process used to develop those software products.

It is based on profound feedback and development practices adopted by the most successful organizations worldwide. This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.

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.

The Capability Maturity Model was originally developed as a tool for objectively assessing the ability of government contractors' processes to implement a contracted software project. The model is based on the process maturity framework first described in IEEE Software and, later, in the 1989 book Managing the Software Process by Watts Humphrey. It was later published in a report in 1993 and as a book by the same authors in 1995.

Though the model comes from the field of software development, it is also used as a model to aid in business processes generally, and has also been used extensively worldwide in government offices, commerce, and industry.

### B. SEI CMMi

Capability Maturity Model Integration (CMMI) is a process level improvement training and appraisal program. Administered by the CMMI Institute, a subsidiary of ISACA, it was developed at Carnegie Mellon University (CMU).

Originally CMMI addresses three areas of interest:

> *Product and service development – CMMI for Development (CMMI-DEV), Service establishment, management, – CMMI for Services (CMMI-SVC), and Product and service acquisition – CMMI for Acquisition (CMMI-ACQ).*

In version 2.0 these three areas (that previously had a separate model each) were merged into a single model.

CMMI was developed by a group from industry, government, and the Software Engineering Institute (SEI) at CMU. CMMI models provide guidance for developing or improving processes that meet the business goals of an organization. A CMMI model may also be used as a framework for appraising the process maturity of the organization. By January 2013, the entire CMMI product suite was transferred from the SEI to the CMMI Institute, a newly created organization at Carnegie Mellon.

CMM Integration project was formed to sort out the problem of using multiple CMMs. CMMI Product Team's mission was to combine three Source Models into a single improvement framework to be used by the organizations pursuing enterprise-wide process improvement.

### C. ISO 9000

The ISO 9000 family of quality management systems (QMS) is a set of standards that helps organizations ensure they meet customer and other stakeholder needs within statutory and regulatory requirements related to a product or service. ISO 9000 deals with the fundamentals of QMS, including the seven quality management principles that underlie the family of standards. ISO 9001 deals with the requirements that organizations wishing to meet the standard must fulfill.

The mission of the International Standard Organization is to market the development of standardization and its related activities to facilitate the international exchange of products and services. It also helps to develop cooperation within the different activities like spheres of intellectual, scientific, technological, and economic activity.

ISO 9000 Quality Standards: It is defined as the quality assurance system in which quality components can be organizational structure, responsibilities, procedures, processes, and resources for implementing quality management. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.

Different types of ISO standards: Here, you will see different types of ISO standards as follows.

ISO 9000: 2000 – ISO 9000: 2000: contains Quality management systems, fundamentals, and vocabulary.

ISO/IEC 9075-9: 2001 –This series of standards has information technology, Database languages, and SQL/MED (Management of External Data).

ISO/IEC 9075-10: 2000 -This series of standards has information technology, Database languages, and SQL/OLB (Object Language Bindings).

ISO/IEC 9075-13: 2002 –This series of standards has information technology, Database languages, SQL routines, and Java Programming language. (SQL/JRT).

3. **Define following and terms and relationship between them**
   **a. Error** **b. Fault** **c. Failure**

**Fault**: It is a condition that causes the software to fail to perform its required function.
**Error**: Refers to difference between Actual Output and Expected output.
**Failure**: It is the inability of a system or component to perform required function according to its specification.
**IEEE Definitions**

- **Failure:** External behavior is incorrect
- **Fault:** Discrepancy in code that causes a failure.
- **Error:** Human mistake that caused fault

**Note:**

- **Error** is terminology of Developer.
- **Bug** is terminology of Tester

**ERROR:** An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of the developer, we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a de-sign notation, or a programmer might type a variable name incorrectly – leads to an Error. It is the one that is generated because of the wrong login, loop or syntax. The error normally arises in software; it leads to a change in the functionality of the program.

**FAULT:** An incorrect step, process or data definition in a computer program that causes the program to perform in an unintended or unanticipated manner. A fault is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. It is the result of the error.

**FAILURE:** A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development, Failures are usually observed by testers.

4. **Write a short note on the components of software quality assurance.**

**Software quality assurance (SQA)** is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc. SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

**Components of SQA System**

An SQA system always combines a wide range of SQA components. These components can be classified into the following six classes −

**Pre-project components**

This assures that the project commitments have been clearly defined considering the resources required, the schedule and budget; and the development and quality plans have been correctly determined.

**Components of project life cycle activities assessment**

The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

The development life cycle stage components detect design and programming errors. Its components are divided into the following sub-classes: Reviews, Expert opinions, and Software testing.

The SQA components used during the operation–maintenance phase includes specialized maintenance components as well as development life cycle components, which are applied mainly for functionality to improve the maintenance tasks.

**Components of infrastructure error prevention and improvement**

The main objective of these components, which is applied throughout the entire organization, is to eliminate or at least reduce the rate of errors, based on the organization's accumulated SQA experience.

**Components of software quality management**

This class of components deal with several goals, such as the control of development and maintenance activities, and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.

**Components of standardization, certification, and SQA system assessment**

These components implement international professional and managerial standards within the organization. The main objectives of this class are utilization of international professional knowledge, improvement of coordination of the organizational quality systems with other organizations, and assessment of the achievements of quality systems according to a common scale. The various standards may be classified into two main groups: quality management standards and project process standards.

**Organizing for SQA – the human components**

The SQA organizational base includes managers, testing personnel, the SQA unit and the persons interested in software quality such as SQA trustees, SQA committee members, and SQA forum members. Their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements.

**Pre-project Software Quality Components**

These components help to improve the preliminary steps taken before starting a project. It includes −Contract Review and Development and Quality Plans

**Contract Review**

Normally, a software is developed for a contract negotiated with a customer or for an internal order to develop a firmware to be embedded within a hardware product. In all these cases, the development unit is committed to an agreed-upon functional specification, budget and schedule. Hence, contract review activities must include a detailed examination of the project proposal draft and the contract drafts.

Specifically, contract review activities include −

- Clarification of the customer's requirements
- Review of the project's schedule and resource requirement estimates
- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfil his obligations
- Evaluation of development risks

**Development and Quality Plans**

After signing the software development contract with an organization or an internal department of the same organization, a development plan of the project and its integrated quality assurance activities are prepared. These plans include additional details and needed revisions based on prior plans that provided the basis for the current proposal and contract.

Most of the time, it takes several months between the tender submission and the signing of the contract. During these periods, resources such as staff availability, professional capabilities may get changed. The plans are then revised to reflect the changes that occurred in the interim.

The main issues treated in the project development plan are −

- Schedules
- Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans

The main issues treated in the project's quality plan are −

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities

**5. Write a short note on different views of the software quality.**

In an influential paper examining views of quality, David Garvin studied how quality is perceived in various domains, including philosophy, economics, marketing, and operations management.2 He concluded that "quality is a complex and multifaceted concept" that can be described from five different perspectives. The transcendental view sees quality as something that can be recognized but not defined. The user view sees quality as fitness for purpose. The manufacturing view sees quality as conformance to specification. The product view sees quality as tied to inherent characteristics of the product. The value-based view sees quality as dependent on the amount a customer is willing to pay for it.

**Transcendental view**

This view of software quality is much like Plato's description of the ideal or Aristotle's concept of form. Just as every table is different but each is an approximation of an ideal table, we can think of software quality as something toward which we strive as an ideal, but may never implement completely. When software gurus exhort us to produce products that delight users, this delight represents the strived-for "recognition" in the transcendental definition of quality.

**User view**

Whereas the transcendental view is ethereal, the user view is more concrete, grounded in product characteristics that meet the user's needs. This view of quality evaluates the product in a task context and can thus be a highly personalized view. In reliability and performance modeling, the user view is inherent, since both methods assess product behavior with respect to operational profiles (that is, to expected functionality and usage patterns). Product usability is also related to the user view: in usability laboratories, researchers observe how users interact with software products.

**Manufacturing view**

The Manufacturing view focuses on product quality during production and after delivery. This view examines whether or not the product was constructed "right the first time," in an effort to avoid the costs associated with rework during development and after delivery. This process focus can lead to quality assessment that is virtually independent of the product itself. That is, the manufacturing approach adopted by ISO 90013 and the Capability Maturity Model4 advocates conformance to process rather than to specification.

**Product view**

Whereas the user and manufacturing views examine the product from without, a product view of quality looks inside, considering the product's inherent characteristics. This approach is frequently adopted by software-metrics advocates, who assume that measuring and controlling internal product properties (internal quality indicators) will result in improved external product behavior (quality in use). Assessing quality by measuring internal properties is attractive because it offers an objective and context independent view of quality. However, more research is needed to confirm that internal quality assures external quality and to determine which aspects of internal quality affect the product's use. Some researchers have developed models to link the product view to the user view.

**Value-based view**

Equating quality to what the customer is willing to pay for encourages everyone to consider the trade-offs between cost and quality. A value-based perception can involve techniques to manage conflicts when requirements change. Among them are "design to cost," in which design possibilities are constrained by available resources and "requirements scrubbing," in which requirements are assessed and revised in light of costs and benefits.

6. **Explain in details about quality Management components.**

When broken down, quality control management can be segmented into four key components to be effective: quality planning, quality control, quality assurance, and quality improvement. Here is a closer look at these four crucial steps, as well as some insight into how our QMS software can make the job easier.

**Quality Control Planning**

The first step of quality management is planning. You need to take the time to identify your goals and what you want your baseline to be. You should determine what your quality standards are, the requirements necessary to meet these standards, and what procedures will be used to check that these criteria are being met. In this planning stage, you will want to consider:

- What your stakeholder's expectations and priorities are, if applicable
- What your company's definition of success is
- What legal standards or requirements are in place that must be abided by
- Who will handle each role in the quality management process (supervision, testing, etc.)
- How often processes will be evaluated for improvement

**Quality Control**

Once you have a plan in place, quality control comes into play. This is the process of physically inspecting and testing what you laid out in the planning stage to make sure it is obtainable. You need to confirm that all the standards you have put into place are met, and you need to identify any mishaps or errors that need to be corrected. The sooner you can catch these errors, the better. As such, you should be paying attention to all aspects of the product, including both the materials used and the process of putting them together.

Once the inspection data has been collected, it should be displayed in a way that makes it easy to analyze. You can create histograms, run charts, or cause and effect display, and then easily share them through your document management software to make sure everyone has access to them.

**Quality Assurance**

While quality control involves inspecting the actual products or services in the field, quality assurance is reviewing the delivery process of services or the quality management manufacturing of goods. By inspecting your goods or services at the source, you can catch mistakes before they reach the customer. You can also fine tune your processes to prevent errors in the future. When reviewing your product or service during this stage of quality control management, you will want to follow these steps:

- Confirm that everything is operating as it was agreed upon during the quality planning stage
- Measure how effective your pre-determined processes are and confirm that all compliance needs are being met
- Take note of any lessons learned
- Identify areas where there is an opportunity for a smoother process
- To be effective, quality assurance must be completed regularly through independent audits. For the best results, have the audit completed by a third-party that is not financially or emotionally invested in the outcome.

**Quality Improvement**

Finally, after completing the quality control process, you need to thoroughly review your findings and come up with a way to improve your methods going forward. Quality control management is fruitless if you are not willing to make changes when they are necessary. The desire for continual improvement is the goal for every successful company. So, gather all your data, re-evaluate both the processes and the product—always keeping compliance in mind—and then begin the quality control management process again. With each cycle, you will end up with a better product, happier customers, and more profit in your pocket.

**7. Describe product and process quality attributes**

The following factors are used to measure Software Product and Process Quality. Each attribute can be used to measure product performance. These attributes can be used for Quality assurance as well as Quality control.

**Reliability**
Measure if the product is reliable enough to sustain in any condition. Should give consistently correct results. Product reliability is measured in terms of working of the project under different working environments and different conditions.

**Maintainability**
Different versions of the product should be easy to maintain. For development it should be easy to add code to the existing system, should be easy to upgrade for new features and new technologies from time to time. Maintenance should be cost-effective and easy. The system is easy to maintain and correcting defects or making a change in the software.

**Usability**
This can be measured in terms of ease of use. The application should be user-friendly. Should be easy to learn. Navigation should be simple. The system must be:

- Easy to use for input preparation, operation, and interpretation of the output.
- Provide consistent user interface standards or conventions with our other frequently used systems.
- Easy for new or infrequent users to learn to use the system.

**Portability**
This can be measured in terms of Costing issues related to porting, technical issues related to porting, Behavioral issues related to porting.

**Correctness**
The application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means the application should adhere to functional requirements.

**Efficiency**
Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example, the system should utilize processor capacity, disk space, and memory efficiently. If the system is using all the available resources, then the user will get degraded performance failing the system for efficiency. If the system is not efficient then it cannot be used in real-time applications.

**Integrity or Security**
Integrity comes with security. System integrity or security should be sufficient to prevent unauthorized access to system functions, preventing information loss, ensure that the software is protected from virus infection, and protecting the privacy of data entered into the system.

**Testability**
The system should be easy to test and find defects. If required should be easy to divide into different modules for testing.

**Flexibility**
Should be flexible enough to modify. Adaptable to other products with which it needs interaction. Should be easy to interface with other standard 3rd party components.

**Reusability**
Software reuse is a good cost-efficient and time-saving development way. Different code library classes should be generic enough to use easily in different application modules. Dividing the application into different modules so that modules can be reused across the application.

**Interoperability**
Interoperability of one system to another should be easy for the product to exchange data or services with other systems. Different system modules should work on different operating system platforms, different databases, and protocol conditions.

*Applying the above quality attributes standards, we can determine whether the system meets the requirements of quality or not.*

**Availability**
This attribute is indicative as to whether an application will execute the tasks it is assigned to perform. Availability also includes certain concepts that relate to software security, performance, integrity, reliability, dependability, and confidentiality. In addition, top-notch availability indicates that a software-driven system will repair any operating faults so that service outage periods would not exceed a specific time value.

**Performance**
This attribute pertains to the ability of a software-driven system to conform to timing requirements. From a testing point of view, it implies that Software Testing engineers must check whether the system

responds to various events within defined time limits. These events may occur in the form of clock events, process interruptions, messages, and requests from different users, and others.

**Security**

This attribute measures the ability of a system to arrest and block malicious or unauthorized actions that could potentially destroy the system. The attribute assumes importance because security denotes the ability of the system to protect data and defend information from unauthorized access. Security also includes authorization and authentication techniques, protection against network attacks, data encryption, and such other risks. It is imperative for Software Testing professionals to regularly conduct updated security checks on systems.

**Functionality**

This attribute determines the conformity of a software-driven system with actual requirements and specifications. Most Software Testing professionals view this attribute as crucial and a foremost requirement of a modern application, and would therefore advocate the performance of tests that assess the desired functionality of a system in the initial stages of Software Testing initiatives.

**8. Explain IEEE standards for SQL Plan.**

- ❑ An important part of achieving quality is to plan for quality.
- ❑ A Software Quality Assurance Plan is **NOT** merely another name for a test plan, though test plans are included in an SQA plan.
- ❑ The IEEE Standards Association has developed a standard (Std 730- 1989) for software quality assurance plans.

The following is part of the sections specified in IEEE Std 730-1989

1. **Purpose:** This section shall list the software covered and the portions of software life cycle covered.
2. **Reference Documents:** This section shall list all the documents referenced in the plan.
3. **Management**
    1. Organization: This selection describes the structure of organization and the responsibilities, and usually includes an organizational chart.
    2. Tasks: This section lists all of the tasks to be performed, relationship between tasks and checkpoints, and the sequence of the tasks.
    3. Responsibilities: This section shall list the responsibilities of each organizational unit.
4. **Documentation**
    1. Purpose: required document and state how documents will be evaluated.
    2. Minimum documents: This section shall describe the minimum required documentation, usually including the following:

SRs—software Requirements Specification SDD—Software Design Description SVVP—Software Verification and Validation Plan SVVR—Software Verification and Validation Report
User documentation & SCMP—SW Configuration Management Plan

5. **Standards, Practices, Conventions, and Metrics**

- ❑ This section shall identify the S, P, C, and M to be applied and how compliance is to be monitored and assured.
- ❑ The minimal contents include documentation standards, logic structure standards, coding standards, testing standards, selected SQA product, and process metrics.

6. **Reviews and Audits:** This section shall define what reviews/audits will be done, how they will be accomplished, and what further actions are required.

7. Tests: This section shall include all tests that are not included in SVVP. [software verification and Validation Plan]
8. Problem Reporting: This section shall define practices and procedures for reporting, tracking, and resolving problems, including organizational responsibilities.
9. Tools, Techniques, and Methodologies: This section shall identify the special software tools, techniques, and methodologies and describe their use.
10. Code Control: This section shall define the methods and facilities to maintain controlled versions of the software.
11. Media Control: This section shall define the methods and facilities to identify, store, and protect the physical media.
12. Supplier Control (for outsourcing): This section shall state provisions for assuring that software provided by suppliers meets standards.
13. Records: This section shall identify documentation to be retained and methods to collection, maintain, and safeguard the documentation.
14. Training: This section shall identify necessary training activities.
15. Risk Management: This section shall specify methods and procedures for risk management.

## 9. Explain the different software testing principles.

Software testing is a process of executing a program with the aim of finding the error. To make our software perform well it should be error free. If testing is done successfully, it will remove all the errors from the software.

There are seven major principles in software testing:

- Testing shows presence of defects
- Exhaustive testing is not possible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is context dependent
- Absence of errors fallacy

Testing shows presence of defects: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn’t talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defects free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not removes all defects.

Exhaustive testing is not possible: It is the process of testing the functionality of a software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test cases. It can test only some test cases and assume that software is correct and it will produce the correct output in every test cases. If the software will test every test case, then it will take more cost, effort, etc. and which is impractical.

Early Testing: To find the defect in the software, early test activity shall be started. The defect detected in early phases of SDLC will very less expensive. For better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase.

Defect clustering: In a project, a small number of the module can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

Pesticide paradox: Repeating the same test cases again and again will not find new bugs. So, it is necessary to review the test cases and add or update test cases to find new bugs.

Testing is context dependent: Testing approach depends on context of software developed. Different types of software need to perform different types of testing. For example, the testing of the e-commerce site is different from the testing of the Android application.

Absence of errors fallacy: If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it also mandatory to fulfill all the customer requirements.

*More broadly speaking, there are other principles outlined as:*

*Principle 1. Testing is the process of exercising a software component using a selected set of test cases, with the intent of (i) revealing defects, and (ii) evaluating quality.*
*Principle 2. When the test objective is to detect defects, then a good test case is one that has a high probability of revealing a yet undetected defect(s).*
*Principle 3. Test results should be inspected meticulously.*
*Principle 4. A test case must contain the expected output or result.*
*Principle 5. Test cases should be developed for both valid and invalid input conditions.*
*Principle 6. The probability of the existence of additional defects in a software component is proportional to the number of defects already detected in that component.*
*Principle 7. Testing should be carried out by a group that is independent of the development group.*
*Principle 8. Tests must be repeatable and reusable*
*Principle 9. Testing should be planned.*
*Principle 10. Testing activities should be integrated into the software life cycle.*
*Principle 11. Testing is a creative and challenging task*

## 10. Describe following techniques

### a. Review

A review is a systematic examination of a document by one or more people with the main aim of finding and removing errors early in the software development life cycle. Reviews are used to verify documents such as requirements, system designs, code, test plans and test cases. Reviews are usually performed manually while static analysis of the tools is performed using tools.

Importance of Review Process:

- Productivity of Dev team is improved and timescales reduced because the correction of defects in early stages and work-products will help to ensure that those work-products are clear and unambiguous.
- Testing costs and time is reduced as there is enough time spent during the initial phase.
- Reduction in costs because fewer defects in the final software.

Types of Defects during Review Process:

- Deviations from standards either internally defined or defined by regulatory or a trade organization.
- Requirements defects.
- Design defects.
- Incorrect interface specifications.

### b. Walkthrough

Walkthrough in software testing is used to review documents with peers, managers, and fellow team members who are guided by the author of the document to gather feedback and reach a consensus. A walkthrough can be pre-planned or organized based on the needs. Generally, people working on the same work product are involved in the walkthrough process.

This review becomes more beneficial for those people who are away from software sphere and through this meeting they get a good insight about the product that is to be developed. The content is being explained by the author step by step to reach a common objective.

The audience is selected from different backgrounds in order to have a diverse point of view and thus, provide different dimensions to a common objective. This is not a formal process but is specifically used for high level documents like requirement specifications or functional specifications, etc.

- ✓ *The specific goals of a walkthrough are:*
- ✓ Gather information regarding the topic in the document by involving stakeholders, both within and outside the software discipline.
- ✓ Describe and justify the contents of the document.
  Reach a common consensus on the document.
  Check and discuss the different solutions to a problem and different suggested alternative.
- ✓ *The outcome of a walkthrough is following:*
  List of items not understood.
  List of items thought to be incorrect.

### c. Inspection

**Inspection** in software engineering, refers to peer review of any work product by trained individuals who look for defects using a well-defined process. An inspection might also be referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

An inspection is one of the most common sorts of review practices found in software projects. The goal of the inspection is to identify defects. Commonly inspected work products include software requirements specifications and test plans. In an inspection, a work product is selected for review and a team is gathered for an inspection meeting to review the work product. A moderator is chosen to moderate the meeting. Each inspector prepares for the meeting by reading the work product and noting each defect. In an inspection, a defect is any part of the work product that will keep an inspector from approving it. For example, if the team is inspecting a software requirements specification, each defect will be text in the document which an inspector disagrees with.

The process should have entry criteria that determine if the inspection process is ready to begin. This prevents unfinished work products from entering the inspection process. The entry criteria might be a checklist including items such as "The document has been spell-checked".

The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework and Follow-up. The Preparation, Inspection meeting and Rework stages might be iterated.

- **Planning:** The inspection is planned by the moderator.
- **Overview meeting:** The author describes the background of the work product.
- **Preparation:** Each inspector examines the work product to identify possible defects.
- **Inspection meeting:** During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.
- **Rework:** The author makes changes to the work product according to the action plans from the inspection meeting.
- **Follow-up:** The changes by the author are checked to make sure everything is correct.

The process is ended by the moderator when it satisfies some predefined exit criteria. The term inspection refers to one of the most important elements of the entire process that surrounds the execution and successful completion of a software engineering project.