UNIT -7_8

Quality Assurance and Management

and Software Maintenance and

Configuration

1) Explain quality control and also explain

cost of quality. 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.

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 inter process 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 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

2. Define quality for software. List and explain SQA activities. (importance of

SQA activities) Software Quality

- Software quality can be defined as "the conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software "

SQA Activities

- Software engineers address quality by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing.
- The charter of the SQA group is to assist the software team in achieving a high-quality end product.
- The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting.
- These activities are performed (or facilitated) by an independent SQA group that:

Prepares an SQA plan for a project.

The plan is developed as part of project planning and is reviewed by all stakeholders. Quality assurance actions performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be conducted, standards that are applicable to the project, procedures for error reporting and tracking, work products that are produced by the SQA group, and feedback that will be provided to the software team.

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.

Reviews software engineering activities to verify compliance with the defined software process.

The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process.

The SQA group reviews elected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

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 plan, process description, applicable standards, or software engineering work products.

Records any noncompliance and reports to senior management.

Noncompliance items are tracked until they are resolved.

3) Explain approaches to SQA.

- In software quality there is one common job, ie. It can be achieved through competent analysis, design, coding and testing, as well as through the application of formal technical reviews, a multitier testing strategy, better control of software work products and changes made to them, and the application of accepted software engineering standards.
- In addition, quality can be defined in terms of broad array of quality factors and measured using a variety of indices and metrics.
- A small, but vocal, segment of the software engineering community has argued that a more formal approach to software quality assurance is required.
- It can be argued that a computer program is mathematical object.
- A rigorous syntax and semantics can be defined for every programming language, and work is underway to develop a similarly rigorous approach to the specification of software requirements.
- If the requirements model (specification) and the programming language can be represented in a rigorous manner, it should be possible to apply mathematic proof of correctness to demonstrate that a program conforms exactly to its specifications.

4) List set of guidelines for formal technical reviews. 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.

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.

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.

Limit the number of participants and insist upon advance preparation.

All review team members must prepare in advance. Written comments should be solicited by the review leader.

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.

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.

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.

10. Review your early reviews.

Debriefing can be beneficial in uncovering problems with the review process itself.

5) Explain different CMM level. Or five levels of SEI – CMM.

- The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity.
- To determine an organization’s current state of process maturity, the SEI uses an assessment that results in a five point grading scheme. The grading scheme determines compliance with a capability maturity model (CMM) that defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that

are defined in the following manner:

Level 1: Initial.

- The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable.

- Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined.

- The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed.

- A detailed measure of the software process and product quality is collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing.

- Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.
- This level includes all characteristics defined for level 4.

- The SEI has associated key process areas (KPAs) with each of the maturity levels.
- The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level.

- **Goals**—the overall objectives that the KPA must achieve.
- **Commitments**—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- **Abilities**—those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
- **Activities**—the specific tasks required to achieve the KPA function.
- **Methods for monitoring implementation**—the manner in which the activities are monitored as they are put into place.
- **Methods for verifying implementation**—the manner in which proper practice for the KPA can be verified.

6) What is software reliability? What is the role of software maintenance in software product?

- Software reliability can be defined it as probability of failure free operations of a computer program in a specified environment for a specified time.

Measures of Reliability and Availability

- 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 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; wear does not enter into the picture.
- If we consider a computer-based system, a simple measure of reliability is meantime 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)] x 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 (non programmable) 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.

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

The maintenance of existing software can account for over 60 percent of all effort expended by a development organization, and the percentage continues to rise as more software is produced.

Change is inevitable when computer-based systems are built; therefore, we must develop mechanisms for evaluating, controlling, and making modifications.

6) What do you mean by software configuration? Or what is meant by software configuration management?

- Software configuration management (SCM) is an umbrella activity that is applied throughout the software process.
- The output of the software process is information that may be divided into three broad **Categories:**

1) Computer programs
2) Documents that describe the computer programs
3) Data

- The items that contain all information produced as part of the software process are collectively called a software configuration.
- When you build computer software, change happens. And because it happens, you need to control it effectively.
- It is a set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products.
- Everyone involved in the software engineering process is involved with SCM to some extent, but specialized support positions are sometimes created to manage the SCM process.
- If you don't control change, it controls you. And that's never good.
- There are four fundamental sources of change:
- New business or market conditions dictate changes in product requirements or business rules.
- New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer based system.
- Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
- Budgetary or scheduling constraints cause a redefinition of the system or product.
- Software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software.
- SCM can be viewed as a software quality assurance activity that is applied throughout the software process.

8) What do you mean by quality assurance? Explain various factors that affect software quality.

- The factors that affect software quality can be categorized in two broad groups: Factors that can be directly measured
  Factors that can be measured only indirectly
- In each case measurement must occur.
- McCall, Richards, and Walters propose a useful categorization of fac-tors that affect software quality.

1) Correctness: -

- The extent to which a program satisfies its specification and fulfils the customer's mission objectives.

2) Reliability:-

- The extent to which a program can be expected to perform its intended function with required precision.

3) Efficiency.

- The amount of computing resources and code required by a program to perform its function.

4) Integrity: -

- Extent to which access to software or data by unauthorized persons can be controlled.

5) Usability: -

- Effort required learning, operating, preparing input, and interpreting output of a program.

6) Maintainability: -

- Effort required locating and fixing an error in a program.

7) Flexibility: -

- Effort required modifying an operational program.

8) Testability: -

- Effort required testing a program to ensure that it performs its intended function.

9) Portability

- Effort required transferring the program from one hardware and/or software sys-tem environment to another.

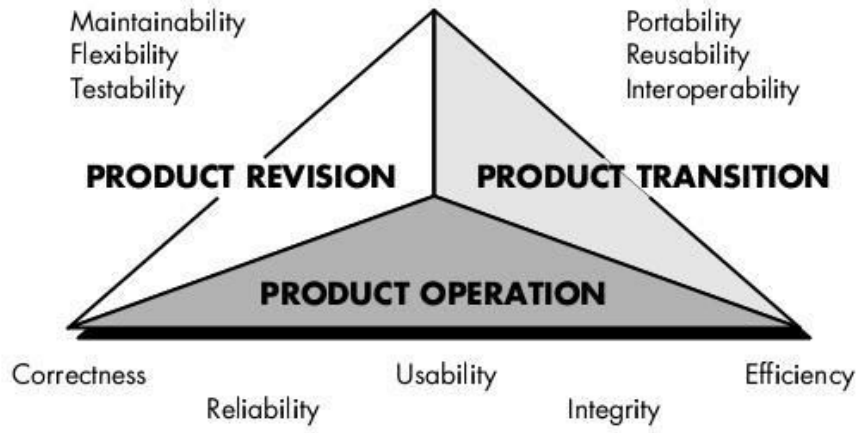
10) Reusability.

- Extent to which a program can be reused in other.
- Applications—related to the packaging and scope of the functions that the program performs.

11) Interoperability:-

- Effort required to couple one system to another.

McCall's Quality Factor



---

9) Explain quality standards – ISO 9000 and 9001.

- A quality assurance system may be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
- In order to bring quality in product and service many organizations are adapting quality assurance system.
- Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.
- These systems cover a wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process.
- ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.
- The ISO 9000 standards have been adopted by many countries. A country typically permits

only ISO registered companies to supply goods and services to government agencies and public utilities.

- Telecommunication equipment and medical devices are examples of product categories that must be supplied by ISO registered companies.
- In turn, manufacturers of these products often require their suppliers to become registered.
- The ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes.
- ISO 9000 describes the elements of a quality assurance system in general terms.
- These elements include the organizational structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance and quality improvement.
- However, ISO 9000 does not describe how an organization should implement these quality system elements.
- The ISO 9001 Standard
- ISO 9001 is the quality assurance standard that applies to software engineering.
- Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guideline have been developed to help interpret the standard for use in the software process.

10) What is software measurement? How to calculate cost of software? Explain software metrics used for s/w cost estimation.

- Measurements in the physical world can be categorized in two ways: direct measures and indirect measures.
- Software metrics can be categorized similarly. Direct measures of the software engineering process include cost and effort applied.
- Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.
- Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability.
- The cost and effort required building software, the number of lines of code produced, and other direct measures are relatively easy to collect, as long as specific conventions for measurement are established in advance.

Size-Oriented Metrics

- Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced.
- If a software organization maintains simple records, a table of size-oriented measures can be created.
- The table lists each software development project that has been completed over the past few years and corresponding measures for that project.

11) In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value.
12) From the rudimentary data contained in the table, a set of simple size-oriented metrics can be developed
13) For each project:
14) Errors per KLOC (thousand lines of code).
15) Defects per KLOC.
16) $ per LOC.

17) Page of documentation per KLOC.
18) In addition, other interesting metrics can be computed:
19) Errors per person-month.
20) LOC per person-month.
21) $ Per page of documentation.

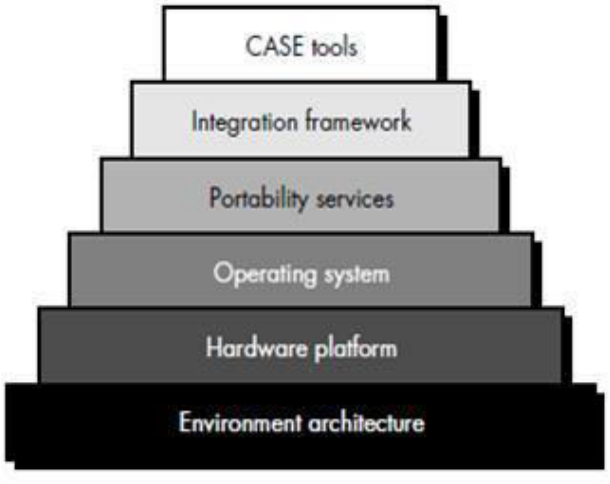
UNIT – 10
Advanced Topic of Software Engineering

1) What are case tools? Explain its importance in SE. OR explain building blocks for case. CASE Tools

- A good workshop for any craftsperson—a mechanic, a carpenter, or a software engineer—has three primary characteristics:
  1) A collection of useful tools that will help in every step of building a product
  2) An organized layout that enables tools to be found quickly and used efficiently
  3) A skilled person who understands how to use the tools in an effective manner.
- Software engineers now recognize that they need more and varied tools along with an organized and efficient workshop in which to place the tools.
- The workshop for software engineering has been called an integrated project support environment and the tools that fill the workshop are collectively called computer aided software engineering.
- CASE provides the software engineer with the ability to automate manual activities and to improve engineering insight.
- Like computer aided engineering and design tools that are used by engineers in other disciplines, CASE tools help to ensure that quality is designed in before the product is built.

Building Blocks

- Computer aided software engineering can be as simple as a single tool that supports a specific software engineering activity or as complex as a complete "environment" that encompasses tools, a database, people, hardware, a network, operating systems, standards, and other components.
- Each building block forms a foundation for the next, with tools sitting at the top of the heap.
- The environment architecture is composed of the hardware platform and system support. It lays the ground work for CASE. But the CASE environment itself demands other building blocks.
- A set of portability services provides a bridge between CASE tools and their integration framework and the environment architecture.
- The integration framework is a collection of specialized programs that enables individual CASE tools to communicate with one another, to create a project database, and to exhibit the same look and feel to the end-user.
- Portability services allow CASE tools and their integration framework to migrate across



different hardware platforms and operating systems without significant adaptive maintenance.

Building Blocks of CASE

- A tool is used to assist in a particular software engineering activity but does not directly communicate with other tools, is not tied into a project database, is not part of an integrated CASE environment.
- When individual tools provide facilities for data exchange, the integration level is improved slightly. Such tools produce output in a standard format that should be compatible with other tools that can read the format.

  In some cases, the builders of complementary CASE tools work together to form a bridge between the tools.
- Single-source integration occurs when a single CASE tools vendor integrates a number of different tools and sells them as a package.
- Although this approach is quite effective, the closed architecture of most single-source environments precludes easy addition of tools from other vendors.

2. Explain taxonomy of case tools

A number of risks are inherent whenever we attempt to categorize CASE tools.

α. CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software engineering process, by the environment architecture that supports them, or even by their origin or cost. The taxonomy presented here uses functions a primary criterion.

i. Business process engineering tools:

By modeling the strategic information requirements of an organization, business process engineering tools provide a "meta-model" from which specific information systems are derived.

ii. Process modeling and management tools:

If an organization works to improve a business (or software) process, it must first understand it. Process modeling tools are used to represent the key elements of a process so that it can be better understood.

iii. Project planning tools:

Tools in this category focus on two primary areas: software project effort and cost estimation and project scheduling. Estimation tools compute estimated effort, project duration, and recommended number of people for a project. Project scheduling tools enable the manager to define all project tasks.

iv. Risk analysis tools:

Identifying potential risks and developing a plan to mitigate, monitor, and manage them is of paramount importance in large projects. Risk analysis tools enable a project manager to build a risk table.

v. Project management tools:

The project schedule and project plan must be tracked and monitored on a continuing basis. In addition, a manager should use tools to collect metrics that will ultimately provide an indication of software product quality.

6. **Requirements tracing tools:**

   When large systems are developed, things "fall into the cracks." That is, the delivered system does not fully meet customer specified requirements. The objective of requirements tracing tools is to provide a systematic approach to the isolation of requirements, beginning with the customer request for proposal or specification.

7. **Metrics and management tools:**

   Software metrics improve a manager's ability to control and coordinate the software engineering process and a practitioner's ability to improve the quality of the software that is produced.

8. **Documentation tools:**

   Document production and desktop publishing tools support nearly every aspect of software engineering and represent a substantial "lever-age" opportunity for all software developers like, word processing unit.

9. **System software tools:**

   CASE is a workstation technology. Therefore, the CASE environment must accommodate high-quality network system software, object management services, distributed component support, electronic mail, bulletin boards and other communication capabilities.

10. **Quality assurance tools:**

    The majority of CASE tools that claim to focus on quality assurance are actually metrics tools that audit source code to determine compliance with language standards.

11. **Database management tools:**

    Database management software serves as a foundation for the establishment of a CASE database (repository) that we have called the project database.

12. **Software configuration management tools:**

    Software configuration management lies at the kernel of every CASE environment. Tools can assist in all five major tasks identification, version control, change control, auditing, and status accounting.

13. **Analysis and design tools:**

    Analysis and design tools enable a software engineer to create models of the system to be built. The models contain a representation of data, function, and behaviour and characterizations of data, architectural, component-level, and interface design.

14. **PRO/SIM tools:**

    PRO/SIM (prototyping and simulation) tools provide the software engineer with the ability to predict the behavior of a real-time system prior to the time that it is built.

15. **Interface design and development tools:**

Interface design and development tools are actually a tool kit of software components (classes) such as menus, buttons, window structures, icons, scrolling mechanisms, device drivers, and so forth.

16. **Prototyping tools:**

    A variety of different prototyping tools can be used. Screen painters enable a software engineer to define screen layout rapidly for interactive applications. More sophisticated CASE prototyping tools enable the creation of a data design.

17. **Programming tools:**

    The programming tools category encompasses the compilers, editors, and debuggers that are available to support most conventional programming languages.

18. **Web development tools:**

    The activities associated with Web engineering are sup-ported by a variety of tools for WebApp development. These include tools that assist in the generation of text, graphics, forms, scripts, applets, and other elements of a Web page.

19. **Integration and testing tools:**

    In their directory of software testing tools, Software Quality Engineering defines the following testing tools categories:

    - Data acquisition
    - Static measurement
    - Dynamic measurement
    - Simulation
    - Test management
    - Cross-functional tools

20. **Static analysis tools:**

    Static testing tools assist the software engineer in deriving test cases. Three different types of static testing tools are used in the industry: code-based testing tools specialized testing languages, and requirements-based testing tools.

21. **Dynamic analysis tools:**

    Dynamic testing tools interact with executing program, checking path coverage, testing assertions about the value of specific variables, and otherwise incrementing the execution flow of the program.

22. **Test management tools:**

    Test management tools are used to control and coordinate software testing for each of the major testing steps. Tools in this category man-age and coordinate regression testing perform comparisons between actual and expected output.

23. **Client/server testing tools:**

    The c/s environment demands specialized testing tools that exercise the graphical user interface and the network communications requirements for client and server.

24. Reengineering tools:

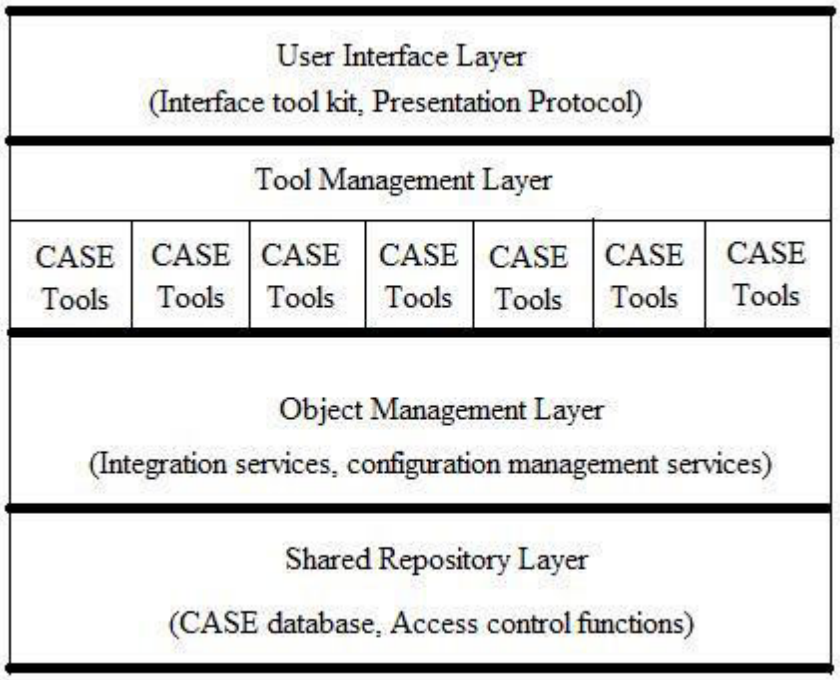
Tools for legacy software address a set of maintenance activities that currently absorb a significant percentage of all software-related effort.

3) Describe integrated case environment.

- Although benefits can be derived from individual CASE tools that address separate software engineering activities, the real power of CASE can be achieved only through integration.
- To define integration in the context of the software engineering process, it is necessary to establish a set of requirements for I-CASE:
- An integrated CASE environment should -
- Provide a mechanism for sharing software engineering information among all tools contained in the environment.
- Enable a change to one item of information to be tracked to other related information items.
- Provide version control and overall configuration management for all software engineering information.
- Allow direct, non sequential access to any tool contained in the environment.
- Establish automated support for the software process model that has been chosen, integrating CASE tools and software configuration items (SCIs) into a standard work breakdown structure.
- Enable the users of each tool to experience a consistent look and feel at the human/computer interface.
- Support communication among software engineers.
- Collect both management and technical metrics that can be used to improve the process and the product.
- CASE tools create a pool of software engineering information. The integrated CASE environment allows a transfer of information into and out of this pool.
- **For such transfer there is a need for some architectural components are :**

1. Database for storing of information.
2. Object management system. Before using the objects information can be transferred in the information pool.

Figure: Integrated CASE Environment



3. Control mechanism
4. User interface

- The fig shows the simple model for integrated CASE environment. This environment consists of various levels.

- The user interface layer consists of interface tool kit and presentation protocols.

- The interface tool kit consists of collection of software required for interface management and display objects.
- The presentation protocol decides a common look and feel of the presentation interface. Then it comes a tools layer. IT consists of set of tools management services (TMS).
- The next layer is object management layer (OML). It performs the configuration management.
- The services of this layer allow the identification of all the configuration objects. So that the case tool can be plugged into the integrated CASE environment.
- The bottom most layer is shared repository layer. It consists of CASE database and access control functions. These access control functions help the object management layer to access the CASE database.