UNIT – V

Software Reliability And Quality Management: Software Reliability, Statistical Testing,Software Quality, Software Quality Management System, ISO 9000, SEI Capability MaturityModel.

Computer Aided Software Engineering: Case and its Scope, Case Environment, Case Support in Software Life Cycle, Other Characteristics of Case Tools, Towards Second Generation CASE Tool, Architecture of a Case Environment

Software Reliability:

Introduction
Hardware vs Software Reliability
Reliability Metrics
Reliability Growth Modelling

Introduction:

Reliability of a software product essentially denotes its trustworthiness or dependability. Alternatively, reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time. It is obvious that a software product having a large number of defects is unreliable. It is also clear that the reliability of a system improves, if the number of defects in it is reduced. It has been experimentally observed by analyzing the behavior of a large number of programs that 90% of the execution time of a typical program is spent in executing only 10% of the instructions in the program. These most used 10% Instructions are often called the ***core of the program.*** The rest 90% of the program statements are called non-core and are executed only for 10% of the total execution time. It is clear that the quantity by which the overall reliability of a program improves due to the correction of a single error depends on how frequently the corresponding instruction is executed.

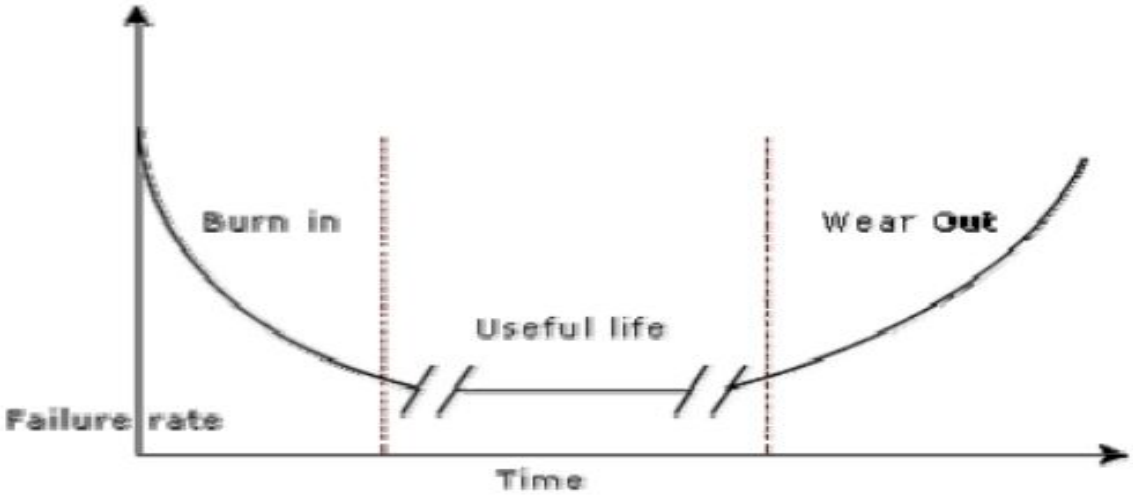
Reasons for software reliability being difficult to measure

- The reliability improvement due to fixing a single bug depends on where the bug is located in the code.
- The perceived reliability of a software product is highly observer dependent.
- The reliability of a product keeps changing as errors are detected and fixed.

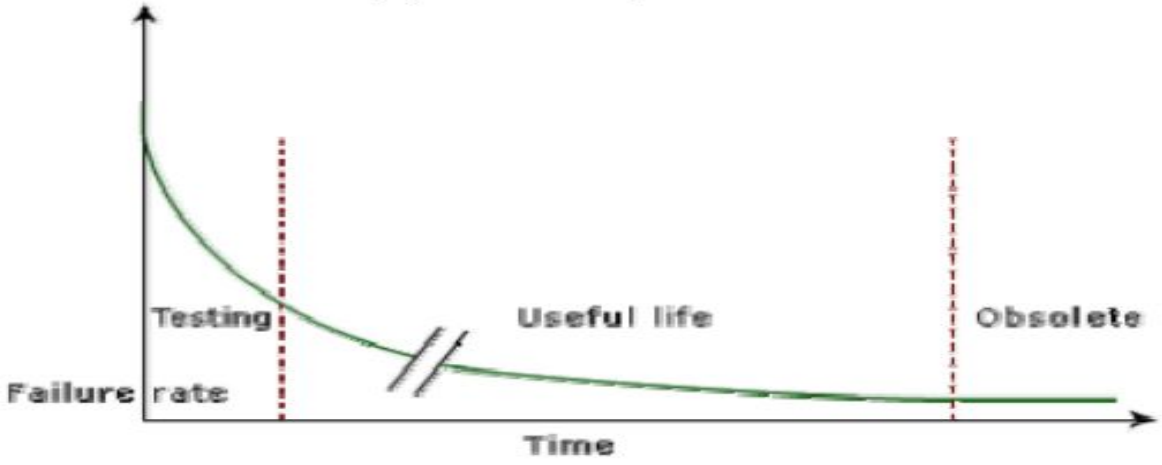
Hardware vs Software Reliability:

Reliability behavior for hardware and software are very different. For example, hardware failures are inherently different from software failures. Most hardware failures are due to component wear and tear. On the other hand, a software product would continue to fail until the error is tracked down and either the design or the code is changed. For this reason, when hardware is repaired its reliability is maintained at the level that existed before the failure occurred; whereas when a software failure is repaired, the reliability may either increase or decrease (reliability may

decrease if a bug introduces new errors). To put this fact in a different perspective, hardware reliability study is concerned with stability (for example, inter-failure times remain constant). On the other hand, software reliability study aims at reliability growth (i.e. inter-failure times increase).



(a) Hardware product



(b) Software product

Reliability Metrics:

Rate of occurrence of failure (ROCOF). ROCOF measures the frequency of occurrence of unexpected behavior (i.e. failures). ROCOF measure of a software product can be obtained by observing the behavior of a software product in operation over a specified time interval and then recording the total number of failures occurring during the interval.

Mean Time To Failure (MTTF). MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures.

Mean Time To Repair (MTTR). Once failure occurs, some time is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.
Mean Time Between Failure (MTBR). MTTF and MTTR can be combined to get the MTBR metric: MTBF = MTTF + MTTR. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours. In this case, time measurements are real time and not the execution time as in MTTF.
Probability of Failure on Demand (POFOD). Unlike the other metrics discussed, this metric does not explicitly involve time measurements. POFOD measures the likelihood of the system failing when a service request is made. For example, a POFOD of 0.001 would mean that 1 out of every 1000 service requests would result in a failure.
Availability. Availability of a system is a measure of how likely shall the system be available for use over a given period of time. This metric not only considers the number of failures occurring during a time interval, but also takes into account the repair time (down time) of a system when a failure occurs.

Reliability Growth Modelling

Jelinski and Moranda Model:

The simplest reliability growth model is a step function model where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. Such a model is shown in fig. 13.2. However, this simple model of reliability which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since it is already known that correction of different types of errors contribute differently to reliability growth.

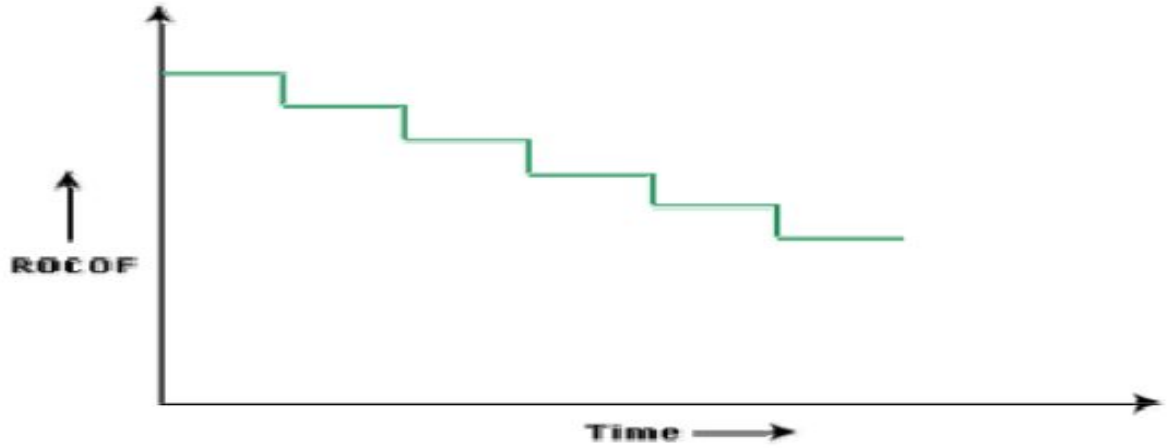


Fig. 13.2: Step function model of reliability growth

Littlewood and Verall's Model

This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases (Fig. 13.3). It treat's an error's contribution to reliability improvement to be an independent random variable having Gamma

distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as test continues.

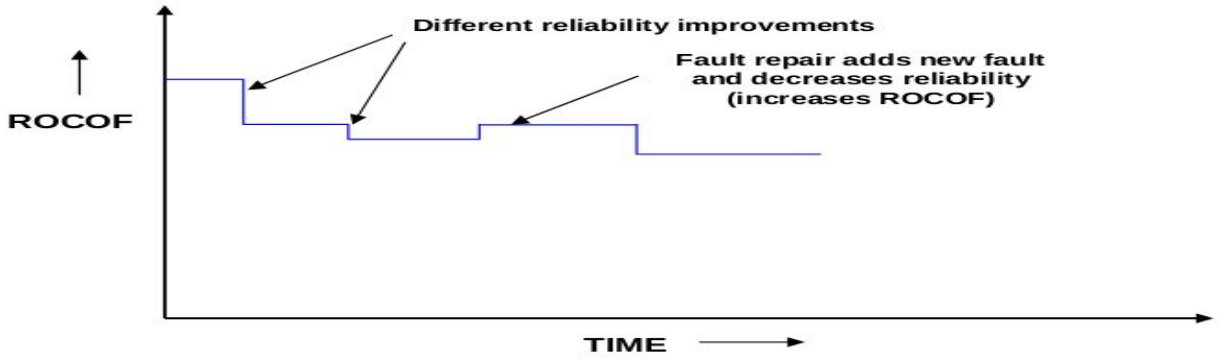


Fig. 13.3: Random-step function model of reliability growth

Statistical Testing:

Statistical testing is a testing process whose objective is to determine the reliability of software products rather than discovering errors. Test cases are designed for statistical testing with an entirely different objective than those of conventional testing.

Operation profile

Different categories of users may use software for different purposes. For example, a Librarian might use the library automation software to create member records, add books to the library, etc. whereas a library member might use to software to query about the availability of the book, or to issue and return books.

Steps in statistical testing

Statistical testing allows one to concentrate on testing those parts of the system that are most likely to be used. The first step of statistical testing is to determine the operation profile of the software. The next step is to generate a set of test data corresponding to the determined operation profile. The third step is to apply the test cases to the software and record the time between each failure. After a statistically significant number of failures have been observed, the reliability can be computed.

Advantages and disadvantages of statistical testing

Statistical testing allows one to concentrate on testing parts of the system that are most likely to be used. Therefore, it results in a system that the users to be more reliable. Reliability estimation using statistical testing is more accurate compared to those of other methods such as ROCOF, POFOD etc.

Software Quality:

The modern view of a quality associates with a software product several quality factors such as the following:

• **Portability:** A software product is said to be portable, if it can be easily made to work in different operating system environments, in different machines, with other software products, etc.
• **Usability:** A software product has good usability, if different categories of users (i.e. both expert and novice users) can easily invoke the functions of the product.
• **Reusability:** A software product has good reusability, if different modules of the product can easily be reused to develop new products.
• **Correctness:** A software product is correct, if different requirements as specified in the SRS document have been correctly implemented.
Maintainability: A software product is maintainable, if errors can be easily corrected as and when they show up, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System:

A quality management system (often referred to as quality system) is the principal methodology used by organizations to ensure that the products they develop have the desired quality. A quality system consists of the following:
• **Managerial Structure and Individual Responsibilities.** A quality system is actually the responsibility of the organization as a whole. However, every organization has a separate quality department to perform several quality system activities. The quality system of an organization should have support of the top management. Without support for the quality system at a high level in a company, few members of staff will take the quality system seriously.
• **Quality System Activities.** The quality system activities encompass the following:
- auditing of projects
- review of the quality system
- development of standards, procedures, and guidelines, etc.
- production of reports for the top management summarizing the effectiveness of the quality system in the organization.

ISO 9000:

ISO 9000 certification Introduction
Types of ISO 9000 quality standards
Software products vs. other products
Need for obtaining ISO 9000 certification
Salient features of ISO 9001 certification
Shortcomings of ISO 9000 certification

ISO 9000 certification Introduction:

ISO (International Standards Organization) is a consortium of 63 countries established to formulate and foster standardization. ISO published its 9000 series of standards in 1987. ISO certification serves as a reference for contract between independent parties. The ISO 9000

standard specifies the guidelines for maintaining a quality system. The ISO standard mainly addresses operational aspects and organizational aspects such as responsibilities, reporting, etc. In a nutshell, ISO 9000 specifies a set of guidelines for repeatable and high quality product development. It is important to realize that ISO 9000 standard is a set of guidelines for the production process and is not directly concerned about the product itself.

Types of ISO 9000 quality standards:

The types of industries to which the different ISO standards apply are as follows.

--***ISO 9001*** applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that is applicable to most software development organizations.

--***ISO 9002*** applies to those organizations which do not design products but are only involved in production. Examples of these category industries include steel and car manufacturing industries that buy the product and plant designs from external sources and are involved in only manufacturing those products. Therefore, ISO 9002 is not applicable to software development organizations.

--***ISO 9003*** applies to organizations that are involved only in installation and testing of the products.

Software products vs. other products:

- Software is intangible in nature and therefore difficult to control. It is very difficult to control and manage anything that is not seen. In contrast, any other industries such as car manufacturing industries where one can see a product being developed through various stages such as fitting engine, fitting doors, etc. Therefore, it is easy to accurately determine how much work has been completed and to estimate how much more time will it take.
- During software development, the only raw material consumed is data. In contrast, large quantities of raw materials are consumed during the development of any other product.

Need for obtaining ISO 9000 certification:

Some benefits that can be acquired to organizations by obtaining ISO certification are as follows:
• Confidence of customers in an organization increases when organization qualifies for ISO certification. This is especially true in the international market. In fact, many organizations awarding international software development contracts insist that the development organization have ISO 9000 certification.
• ISO 9000 requires a well-documented software production process to be in place. A well-documented software production process contributes to repeatable and higher quality of the developed software.
• ISO 9000 makes the development process focused, efficient, and cost effective.

• ISO 9000 certification points out the weak points of an organization and recommends remedial action.
• ISO 9000 sets the basic framework for the development of an optimal process and Total Quality Management (TQM).

Salient features of ISO 9001 certification:

The salient features of ISO 9001 are as follows:
• All documents concerned with the development of a software product should be properly managed, authorized, and controlled. This requires a configuration management system to be in place.
• Proper plans should be prepared and then progress against these plans should be monitored.
• Important documents should be independently checked and reviewed for effectiveness and correctness.
• The product should be tested against specification.
• Several organizational aspects should be addressed e.g., management reporting of the quality team.

Shortcomings of ISO 9000 certification:

ISO 9000 requires a software production process to be adhered to but does not guarantee the process to be of high quality. It also does not give any guideline for defining an appropriate process.
--ISO 9000 requires a software production process to be adhered to but does not guarantee the product to be of high quality.
--Most of companies have less funding available, therefore companies are finding difficulties to adopt ISO system.
--ISO 9000 registration need heavy document workload.
--ISO 9000 registration process require long time.
-- ISO 9000 certification process is not fool-proof and no international accreditation agency exists. Therefore it is likely that variations in the norms of awarding certificates can exist among the different accreditation agencies and also among the registrars.
--Organizations getting ISO 9000 certification often tend to downplay domain expertise. These organizations start to believe that since a good process is in place, any engineer is as effective as any other engineer in doing any particular activity relating to software development.
--ISO 9000 does not automatically lead to continuous process improvement, i.e. does not automatically lead to TQM.

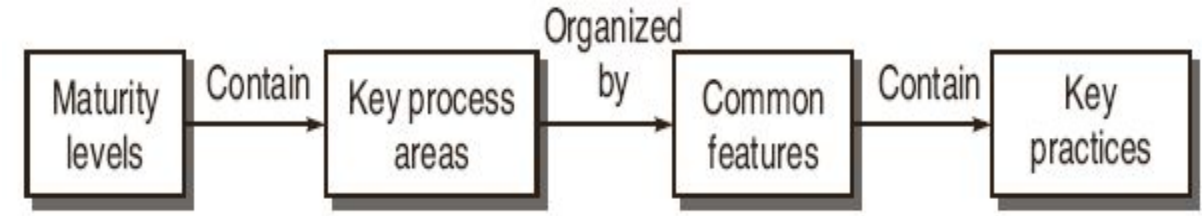
SEI Capability Maturity Model:

Capability Maturity Model is a bench-mark for measuring the maturity of an organization's software process. It is a methodology used to develop and refine an organization's software development process. CMM can be used to assess an organization against a scale of five process maturity levels based on certain Key Process Areas (KPA). It describes the maturity of the company based upon the project

the company is dealing with and the clients. Each level ranks the organization according to its standardization of processes in the subject area being assessed.

A maturity model provides:
--A place to start
--The benefit of a community's prior experiences
--A common language and a shared vision
--A framework for prioritizing actions
--A way to define what improvement means for your organization

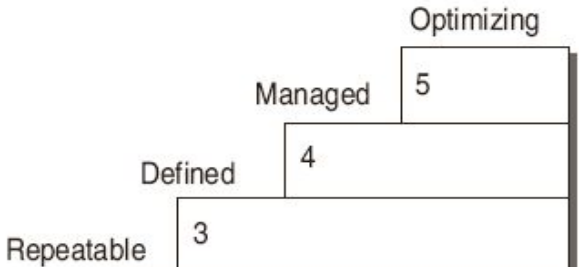
CMM Structure:



Maturity levels	Indicate	Process capability
Key process areas	Achieve	Goals
Common features	Address	Implementation/Institutionalization
Key practices	Describe	Infrastructure/Activities

Maturity Levels:It is a layered framework providing a progression to the discipline needed to engage in continuous improvement
Key Process Areas: A Key Process Area (KPA) identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important.
Common Features: Common features include practices that implement and institutionalize a key process area. These five types of common features include: Commitment to Perform, Ability to Perform, Activities Performed, Measurement and Analysis, and Verifying Implementation.
Key Practices: The key practices describe the elements of infrastructure and practice that contribute most effectively to the implementation and institutionalization of the key process areas.

Maturity Levels:



***Initial*:** It is a basis for comparison with the next levels. In an organization at the initial level, conditions are not stable for the development of quality software.

***Repeatable*:** At this level, project management technologies have been introduced in a company. That project planning and management is based on accumulated experience and there are standards for produced software (these standards are documented) and there is a special quality management group.

***Defined*:** Here, standards for the processes of software development and maintenance are introduced and documented (including project management). During the introduction of standards, a transition to more effective technologies occurs.

***Managed*:** There are quantitative indices (for both software and process as a whole) established in the organization. Better project management is achieved due to the decrease of digression in different project indices.

***Optimizing*:** Improvement procedures are carried out not only for existing processes, but also for evaluation of the efficiency of newly introduced innovative technologies. The main goal of an organization on this level is permanent improvement of existing processes.

Maturity Level	**Process Capability**
1	-----
2	Disciplined Process
3	Standard Consistent Process

4	Predictable Process
5	Continously Improving Proces

Computer Aided Software Engineering

Case and its Scope:

A CASE (Computer Aided Software Engineering) tool is a generic term used to denote any form of automated support for software engineering. In a more restrictive sense, a CASE tool means any tool used to automate some activity associated with software development. Many CASE tools are available. Some of these CASE tools assist in phase related tasks such as specification, structured analysis, design, coding, testing, etc.; and others to non-phase activities such as project management and configuration management.

The primary reasons for using a CASE tool are:

– To increase productivity

– To help produce better quality software at lower cost

CASE environment:

Introduction
Benifits

Introduction:

Although individual CASE tools are useful, the true power of a tool set can be realized only when these set of tools are integrated into a common framework or environment. CASE tools are characterized by the stage or stages of software development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository to have a consistent view of information associated with the software development artifacts. This central repository is usually a data dictionary containing the definition of all composite and elementary data items.

Through the central repository all the CASE tools in a CASE environment share common information among themselves. Thus a CASE environment facilities the automation of the step-by-step methodologies for software development. A schematic representation of a CASE environment is shown in fig. 15.1.

Benefits of CASE:

Several benefits accrue from the use of a CASE environment or even isolated CASE tools. Some of those benefits are:

-- A key benefit arising out of the use of a CASE environment is cost saving through all development phases. Different studies carry out to measure the impact of CASE put the effort reduction between 30% to 40%.

– Use of CASE tools leads to considerable improvements to quality. This is mainly due to the facts that one can effortlessly iterate through the different phases of software development and the chances of human error are considerably reduced.

– CASE tools help produce high quality and consistent documents. Since the important data relating to a software product are maintained in a central repository, redundancy in the stored data is reduced and therefore chances of inconsistent documentation is reduced to a great extent.

– CASE tools take out most of the drudgery in a software engineer's work. For example, they need not check meticulously the balancing of the DFDs but can do it effortlessly through the press of a button.

– CASE tools have led to revolutionary cost saving in software maintenance efforts. This arises not only due to the tremendous value of a CASE environment in traceability and consistency checks, but also due to the systematic information capture during the various phases of software development as a result of adhering to a CASE environment.

– Introduction of a CASE environment has an impact on the style of working of a company, and makes it oriented towards the structured and orderly approach.

Case Support in Software Life Cycle:

Prototyping support
Structured Analysis and Design
Code Generation
Test case generator

Features of a good prototyping CASE tool:

A good prototyping tool should support the following features:

– Since one of the main uses of a prototyping CASE tool is graphical user interface (GUI) development, prototyping CASE tool should support the user to create a GUI using a graphics editor.

• It should integrate with the data dictionary of a CASE environment.

• If possible, it should be able to integrate with external user defined modules written in C or some popular high level programming languages.

Structured analysis and design with CASE tools

Several diagramming techniques are used for structured analysis and structured design. The following supports might be available from CASE tools.

• A CASE tool should support one or more of the structured analysis and design techniques.

• It should support effortlessly drawing analysis and design diagrams.

• It should support drawing for fairly complex diagrams, preferably through a hierarchy of levels.

• The CASE tool should provide easy navigation through the different levels and through the design and analysis.

Code generation and CASE tools

– The CASE tool should support generation of module skeletons or templates in one or more popular languages.

• The tool should generate records, structures, class definition automatically from the contents of the data dictionary in one or more popular languages.

– It should generate database tables for relational database management systems.

– The tool should generate code for user interface from prototype definiton for X window and MS window based applications.

Test case generation CASE tool

• It should support both design and requirement testing.

• It should generate test set reports in ASCII format which can be directly imported into the test plan document.

Other Characteristics of Case Tools:

Hardware and environmental requirements
Documentation support
Project management support
External interface
Reverse engineering
Data dictionary interface

Hardware and environmental requirements:

In most cases, it is the existing hardware that would place constraints upon the CASE tool selection. Thus, instead of defining hardware requirements for a CASE tool, the task at hand becomes to fit in an optimal configuration of CASE tool in the existing hardware capabilities.

The heterogeneous network is one instance of distributed environment and this can be chosen for illustration as it is more popular due to its machine independent features. The CASE tool implementation in heterogeneous network makes use of client-server paradigm.

Documentation support:

The deliverable documents should be organized graphically and should be able to incorporate text and diagrams from the central repository. This helps in producing up-to-date documentation.

Project management support:

The CASE tool should support collecting, storing, and analyzing information on the software project's progress such as the estimated task duration, scheduled and actual task start, completion date, dates and results of the reviews, etc.

External interface:

The CASE tool should allow exchange of information for reusability of design. The information which is to be exported by the CASE tool should be preferably in ASCII format and support open architecture.

Reverse engineering:

If the tool is used for re-engineering information systems, it should contain conversion tool from indexed sequential file structure, hierarchical and network database to relational database systems.

Data dictionary interface:

The data dictionary interface should provide view and update access to the entities and relations stored in it. It should have print facility to obtain hard copy of the viewed screens.

Towards Second Generation CASE Tool:

The second-generation CASE tools have following features:

– ***Intelligent diagramming support:*** The fact that diagramming techniques are useful for system analysis and design is well established. The future CASE tools would provide help to aesthetically and automatically lay out the diagrams.

– ***Integration with implementation environment:*** The CASE tools should provide integration between design and implementation.

– ***Data dictionary standards:*** The user should be allowed to integrate many development tools into one environment. This is possibly only if some standard on data dictionary emerges.

– ***Customization support:*** The user should be allowed to define new types of objects and connections.

Architecture of a Case Environment

The architecture of a typical modern CASE environment is shown diagrammatically in fig. 15.2. The important components of a modern CASE environment are user interface, tool set, object management system (OMS), and a repository.

User Interface:

The user interface provides a consistent framework for accessing the different tools thus making it easier for the users to interact with the different tools and reducing the overhead of learning how the different tools are used.

Object Management System (OMS) and Repository:

Different case tools represent the software product as a set of entities such as specification, design, text data, project plan, etc. The object management system maps these logical entities such into the underlying storage management system (repository). Thus the object management system takes care of appropriately mapping into the underlying storage management system.