Software Reliability and Quality Management: Software Reliability, Statistical Testing, Software Quality, Software Quality Management System, ISO 9000, SEI Capability Maturity Model.
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

- Reliability of a software product is an important concern for most users. Users not only want the products they purchase to be highly reliable, but for certain categories of products they may even require a quantitative guarantee on the reliability of the product before making their buying decision.
- However, it is very difficult to accurately measure the reliability of any software product. One of the main problems encountered while quantitatively measuring the reliability of a software product is the fact that reliability is observer-dependent. That is, different groups of users may arrive at different reliability estimates for the same product.

5.1 SOFTWARE RELIABILITY

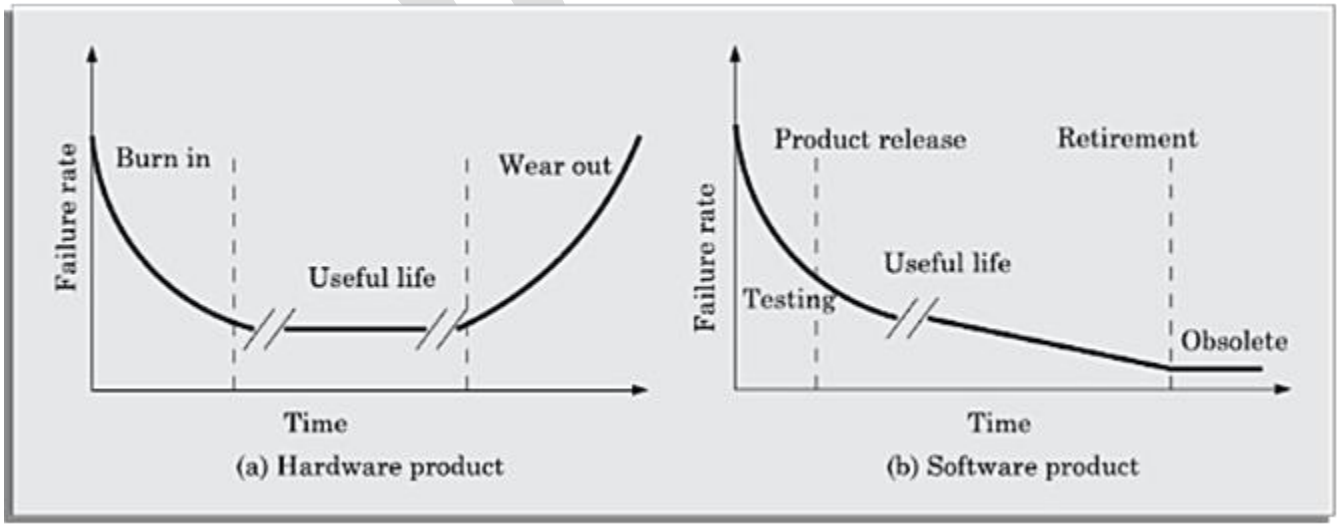
•The reliability of a software product essentially denotes its trustworthiness or dependability. Alternatively, the reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time.

- ✓ 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 observer-dependent.
- ✓ The reliability of a product keeps changing as errors are detected and fixed.

Hardware versus Software Reliability

•Hardware components fail due to very different reasons as compared to software components. Hardware components fail mostly due to wear and tear, whereas software components fail due to bugs.

- A logic gate may be stuck at 1 or 0, or a resistor might short circuit. To fix a hardware fault, one has to either replace or repair the failed part.
- In contrast, a software product would continue to fail until the error is tracked down and either the design or the code is changed to fix the bug.
- For this reason, when a hardware part is repaired its reliability would be 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 fix introduces new errors).
- A comparison of the changes in failure rate over the product life time for a typical hardware product as well as a software product are sketched in Figure11.1.



Reliability Metrics of Software Products

- it is very difficult to formulate a metric using which precise reliability measurement would be possible. In the absence of such measures, we discuss six metrics that correlate with reliability as follows:
- Rate of occurrence of failure (ROCOF): ROCOF measures the frequency of occurrence of 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 calculating the ROCOF value as the ratio of the total number of failures observed and the duration of observation.
- Mean time to failure (MTTF): MTTF is the time between two successive failures, averaged over a large number of failures. To measure MTTF, we can record the failure data for n failures. Let the failures occur at the time.

instants t1, t2, ..., tn. Then, MTTF can be calculated as

.

$$\sum_{i=1}^{n} \frac{t_{i+1}-t_i}{(n-1)}.$$

It is important to note that only run time is considered in the time measurements. That is, the time for which the system is down to fix the error, the boot time, etc. are not taken into account in the time measurements and the clock is stopped at these times.

- 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 (MTBF): The MTTF and MTTR metrics can be combined to get the MTBF 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, the 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 would 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.

Shortcomings of reliability metrics of software products

- In order to estimate the reliability of a software product more accurately, it is necessary to classify various types of failures.
- Transient: Transient failures occur only for certain input values while invoking a function of the system.
- Permanent: Permanent failures occur for all input values while invoking a function of the system.
- Recoverable: When a recoverable failure occurs, the system can recover without having to shutdown and restart the system (with or without operator intervention).
- Unrecoverable: In unrecoverable failures, the system may need to be restarted.
- Cosmetic: These classes of failures cause only minor irritations, and do not lead to incorrect results. An example of a cosmetic failure is the situation where the mouse button has to be clicked twice instead of once to invoke a given function through the graphical user interface.

Reliability Growth Modelling

- A reliability growth model is a mathematical model of how software reliability improves as errors are detected and repaired.
- A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained. Thus, reliability growth modelling can be used to determine when to stop testing to attain a given reliability level'

Belinsky and Morando 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 Figure11.2.

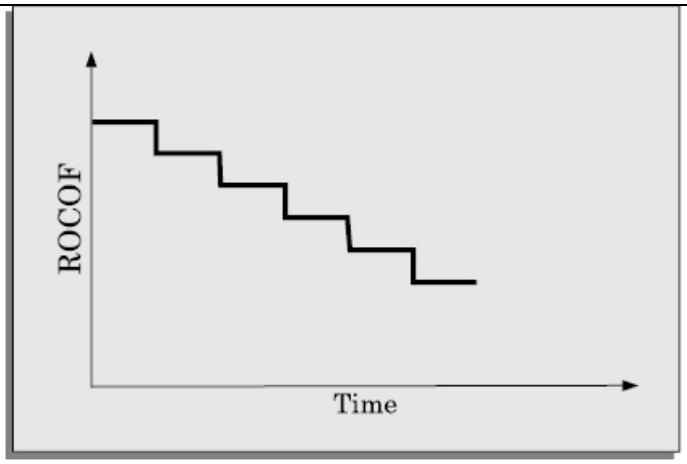


Figure 11.2: Step function model of reliability growth.

Littlewood and Veeral'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 to the product reliability per repair decreases.
- It treats an error's contribution to reliability improvement to be an independent random variable having Gamma distribution.

5.2 STATISTICAL TESTING

- Statistical testing is a testing process whose objective is to determine the reliability of the product rather than discovering errors.
- To carry out statistical testing, we need to first define the operation profile of the product.
- Operation profile: Different categories of users may use a software product for very different purposes.
- Formally, we can define the operation profile of a software as the probability of a user selecting the different functionalities of the software.

  How to define the operation profile for a product?
- We need to divide the input data into a number of input classes. For example, for a graphical editor software, we might divide the input into data associated with the edit, print, and file operations. We then need to assign a probability value to each input class; to signify the probability for an input value from that class to be selected.
- The operation profile of a software product can be determined by observing and analyzing the usage pattern of the software by a number of users.

  Steps in Statistical Testing
- The first step 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.

  Pros and cons 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 can find to be more reliable (than actually it is!).

5.3 SOFTWARE QUALITY

- The modern view of a quality associates with a software product several quality factors (or attributes) such as the following:

- Portability: A software product is said to be portable, if it can be easily made to work in different hardware and operating system environments, and easily interface with external hardware devices and software products.
- 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.

McCall's quality factors

- McCall distinguishes two levels of quality attributes [McCall]. The higher- level attributes, known as quality factor s or external attributes can only be measured indirectly.
- The second-level quality attributes are called quality criteria. Quality criteria can be measured directly, either objectively or subjectively. By combining the ratings of several criteria, we can either obtain a rating for the quality factors, or the extent to which they are satisfied.
- For example, the reliability cannot be measured directly, but by measuring the number of defects encountered over a period of time. Thus, reliability is a higher-level quality factor and number of defects is a low-level quality factor.

ISO 9126

- ISO 9126 defines a set of hierarchical quality characteristics. Each sub characteristic in this is related to exactly one quality characteristic.
- This is in contrast to the McCall's quality attributes that are heavily interrelated. Another difference is that the ISO characteristic strictly refers to a software product, whereas McCall's attributes capture process quality issues as well.

5.4 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.
- we briefly discuss some of the important issues associated with a quality system:

Managerial structure and individual responsibilities

- A quality system is 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 the full 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 to check if the processes are being followed.
- ✓ Collect process and product metrics and analyses them to check if quality goals are being met.
- ✓ Review of the quality system to make it more effective. Development of standards, procedures, and guidelines.
- ✓ Produce reports for the top management summarizing the effectiveness of the quality system in the organization.
- ✓ A good quality system must be well documented. Without a properly documented quality system, the application of quality controls and procedures become ad hoc,

Evolution of Quality Systems

- Quality systems of organizations have undergone four stages of evolution as shown in Figure 11.3. The initial product inspection method gave way to quality control (QC) principles.
- Quality control (QC) focuses not only on detecting the defective products and eliminating them, but also on determining the causes behind the defects, so that the product rejection rate can be reduced.

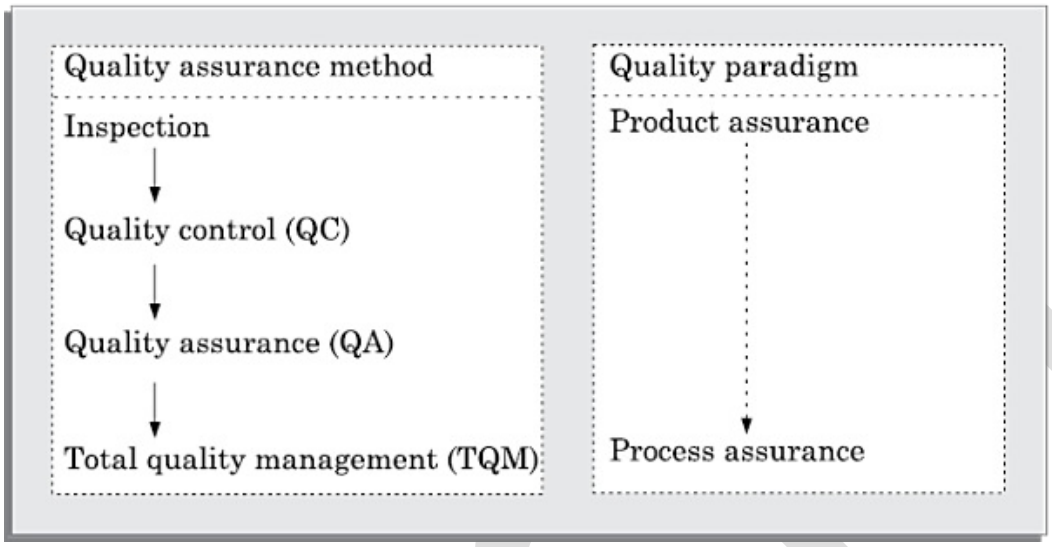


Figure 11.3: Evolution of quality system and corresponding shift in the quality paradigm.

- The modern quality assurance paradigm includes guidance for recognizing, defining, analyzing, and improving the production process.
- Total quality management (TQM) advocates that the process followed by an organization must continuously be improved through process measurements.
- TQM goes a step further than quality assurance and aims at continuous process improvement. TQM goes beyond documenting processes to optimizing them through redesign.
- A term related to TQM is business process re-engineering (BPR), which is aims at re-engineering the way business is carried out in an organization, whereas our focus in this text is re-engineering of the software development process.

Product Metrics versus Process Metrics

- Product metrics help measure the characteristics of a product being developed, whereas process metrics help measures how a process is performing.
- Examples of product metrics are LOC and function point to measure size, PM (person- month) to measure the effort required to develop it, months to measure the time required to develop the product, time complexity of the algorithms, etc.
- Examples of process metrics are review effectiveness, average number of defects found per hour of inspection, average defect correction time, productivity, average number of failures detected during testing per LOC, number of latent defects per line of code in the developed product.

5.5 ISO 9000

- International standards organization (ISO) is a consortium of 63 countries established to formulate and foster standardization. ISO published its 9000 series of standards in 1987.

What is ISO 9000 Certification?

- ISO 9000 certification serves as a reference for contract between independent parties.

- 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.
- The types of software companies to which the different ISO standards apply are as follows:
- ISO 9001: This standard 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: This standard applies to those organizations which do not design products but are only involved in production. Examples of this category of industries include steel and car manufacturing industries who 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: This standard applies to organizations involved only in installation and testing of products.

ISO 9000 for Software Industry

- Two major differences between software development and development of other kinds of products are as follows:
- Software is intangible and therefore difficult to control. It means that software would not be visible to the user until the development is complete and the software is up and running. It is difficult to control and manage anything that you cannot see and feel. In contrast, in any other type of product manufacturing such as car manufacturing, you can see a product being developed through various stages such as fitting engine, fitting doors, etc. Therefore, it becomes 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. As an example, consider a steel making company. The company would consume large amounts of raw material such as iron-ore, coal, lime, manganese, etc. Not surprisingly then, many clauses of ISO 9000 standards are concerned with raw material control. These clauses are obviously not relevant for software development organisations.
- Due to such radical differences between software and other types of product development, it was difficult to interpret various clauses of the original ISO standard in the context of software industry. Therefore, ISO released a separate document called ISO 9000 part-3 in 1991 to help interpret the ISO standard for software industry.

Why Get ISO 9000 Certification?

- Let us examine some of the benefits that accrue to organisations obtaining ISO certification.
- Confidence of customers in an organisation increases when the organisation qualifies for ISO 9001 certification. This is especially true in the international market
- 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 organisations and recommends remedial action.
- ISO 9000 sets the basic framework for the development of an optimal process and TQM.

How to Get ISO 9000 Certification?

The ISO 9000 registration process consists of the following stages:

- **Application stage:** Once an organisation decides to go for ISO 9000 certification, it applies to a registrar for registration.
- **Pre-assessment:** During this stage the registrar makes a rough assessment of the organisation.

- **Document review and adequacy audit:** During this stage, the registrar reviews the documents submitted by the organisation and makes suggestions for possible improvements.
- **Compliance audit:** During this stage, the registrar checks whether the suggestions made by it during review have been complied to by the organisation or not.
- **Registration:** The registrar awards the ISO 9000 certificate after successful completion of all previous phases.
- **Continued surveillance:** The registrar continues monitoring the organisation periodically.

Summary of ISO 9001 Requirements

- A summary of the main requirements of ISO 9001 as they relate of software development are as follows:
- Section numbers in brackets correspond to those in the standard itself:

Management responsibility (4.1)

- The management must have an effective quality policy.
- The responsibility and authority of all those whose work affects quality must be defined and documented.
- A management representative, independent of the development process, must be responsible for the quality system.
- This requirement probably has been put down so that the person responsible for the quality system can work in an unbiased manner.
- The effectiveness of the quality system must be periodically reviewed by audits.

Quality system (4.2)

- A quality system must be maintained and documented.

Contract reviews (4.3)

- Before entering into a contract, an organisation must review the contract to ensure that it is understood, and that the organisation has the necessary capability for carrying out its obligations.

Design control (4.4)

- The design process must be properly controlled, this includes controlling coding also. This requirement means that a good configuration control system must be in place.
- Design inputs must be verified as adequate. Design must be verified.
- Design output must be of required quality. Design changes must be controlled

Document control (4.5)

- There must be proper procedures for document approval, issue and removal.
- Document changes must be controlled. Thus, use of some configuration management tools is necessary.

Purchasing (4.6)

- Purchased material, including bought-in software must be checked for conforming to requirements.

Purchaser supplied product (4.7)

- Material supplied by a purchaser, for example, client-provided software must be properly managed and checked.

Product identification (4.8)

- The product must be identifiable at all stages of the process. In software terms this means configuration management.

Process control (4.9)

- The development must be properly managed.
- Quality requirement must be identified in a quality plan.

Inspection and testing (4.10)

- In software terms this requires effective testing i.e., unit testing, integration testing and system testing. Test records must be maintained.
- Inspection, measuring and test equipment (4.11)

If integration, measuring, and test equipments are used, they must be properly maintained and calibrated.

Inspection and test status (4.12)

- The status of an item must be identified. In software terms this implies configuration management and release control.

Control of non-conforming product (4.13)

- In software terms, this means keeping untested or faulty software out of the released product, or other places whether it might cause damage.

Corrective action (4.14)

- This requirement is both about correcting errors when found, and also investigating why the errors occurred and improving the process to prevent occurrences. If an error occurs despite the quality system, the system needs improvement.

Handling (4.15)

- This clause deals with the storage, packing, and delivery of the software product.

Quality records (4.16)

- Recording the steps taken to control the quality of the process is essential in order to be able to confirm that they have actually taken place.

Quality audits (4.17)

- Audits of the quality system must be carried out to ensure that it is effective.

Training (4.18)

- Training needs must be identified and met.
- Various ISO 9001 requirements are largely common sense.
- Official guidance on the interpretation of ISO 9001 is inadequate at the present time, and taking expert advice is usually worthwhile.

Salient Features of ISO 9001 Requirements

- We can summarise the salient features all the the requirements as follows:
- Document control: All documents concerned with the development of a software product should be properly managed, authorised, and controlled. This requires a configuration management system to be in place.
- Planning: Proper plans should be prepared and then progress against these plans should be monitored.
- Review: Important documents across all phases should be independently checked and reviewed for effectiveness and correctness.
- Testing: The product should be tested against specification.
- Organisational aspects: Several organisational aspects should be addressed e.g., management reporting of the quality team.

ISO 9000-2000

- ISO revised the quality standards in the year 2000 to fine tune the standards. The major changes include a mechanism for continuous process improvement. There is also an increased emphasis on the role of the top management, including establishing a measurable objectives for various roles and levels of the organisation. The new standard recognises that there can be many processes in an organisation.

Shortcomings of ISO 9000 Certification

- Even though ISO 9000 is widely being used for setting up an effective quality system in an organisation, it suffers from several shortcomings. Some of these shortcoming of the ISO 9000 certification process are the following:
- 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 certification process is not fool-proof and no international accredition agency exists. Therefore it is likely that variations in the norms of awarding certificates can exist among the different accredition agencies and also among the registrars.

- Organisations getting ISO 9000 certification often tend to downplay domain expertise and the ingenuity of the developers.
- ISO 9000 does not automatically lead to continuous process improvement. In other words, it does not automatically lead to TQM.

5.6 SEI CAPABILITY MATURITY MODEL

- SEI capability maturity model (SEI CMM) was proposed by Software Engineering Institute of the Carnegie Mellon University, USA.
- In simple words, CMM is a reference model for apprising the software process maturity into different levels.
- This can be used to predict the most likely outcome to be expected from the next project that the organisation undertakes.
- SEI CMM classifies software development industries into the following five maturity levels:
  Level 1: Initial
- A software development organisation at this level is characterised by ad hoc activities. Very few or no processes are defined and followed.
- Since software production processes are not defined, different engineers follow their own process and as a result development efforts become chaotic.
- The success of projects depend on individual efforts and heroics.
  Level 2: Repeatable
- At this level, the basic project management practices such as tracking cost and schedule are established. Configuration management tools are used on items identified for configuration control. Size and cost estimation techniques such as function point analysis, COCOMO, etc., are used.
- The necessary process discipline is in place to repeat earlier success on projects with similar applications. Though there is a rough understanding among the developers about the process being followed, the process is not documented.
- Configuration management practices are used for all project deliverables. Please remember that opportunity to repeat a process exists only when a company produces a family of products. Since the products are very similar, the success story on development of one product can repeated for another.
- In a non- repeatable software development organisation, a software product development project becomes successful primarily due to the initiative, effort, brilliance, or enthusiasm displayed by certain individuals.
- On the other hand, in a non-repeatable software development organisation, the chances of successful completion of a software project is to a great extent depends on who the team members are.
- For this reason, the successful development of one product by such an organisation does not automatically imply that the next product development will be successful.
  Level 3: Defined
- At this level, the processes for both management and development activities are defined and documented. There is a common organisation-wide understanding of activities, roles, and responsibilities.
- The processes though defined, the process and product qualities are not measured. At this level, the organisation builds up the capabilities of its employees through periodic training programs
  Level 4: Managed
- At this level, the focus is on software metrics. Both process and product metrics are collected. Quantitative quality goals are set for the products and at the time of completion of development it was checked whether the quantitative quality goals for the product are met.
- Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to check if a project performed satisfactorily. Thus, the results of process measurements are used to evaluate project performance rather than improve the process.
- Level 5: Optimising

- At this stage, process and product metrics are collected. Process and product measurement data are analysed for continuous process improvement.
- For example, if from an analysis of the process measurement results, it is found that the code reviews are not very effective and a large number of errors are detected only during the unit testing, then the process would be fine tuned to make the review more effective.
- Also, the lessons learned from specific projects are incorporated into the process. Continuous process improvement is achieved both by carefully analysing the quantitative feedback from the process measurements and also from application of innovative ideas and technologies.
- At CMM level 5, an organisation would identify the best software engineering practices and innovations (which may be tools, methods, or processes) and would transfer these organisation- wide.
- Level 5 organisations usually have a department whose sole responsibility is to assimilate latest tools and technologies and propagate them organisation-wide. Since the process changes continuously, it becomes necessary to effectively manage a changing process. Therefore, level 5 organisations use configuration management techniques to manage process changes.
- The focus of each level and the corresponding key process areas are shown in the Table 11.1:

Table 11.1 Focus areas of CMM levels and Key Process Areas

CMM Level	*Focus*	*Key Process Areas (KPAs)*
Initial	Competent people	
Repeatable	Project management	Software project planning Software configuration management
Defined	Definition of processes	Process definition Training program Peer reviews
Managed	Product and process quality	Quantitative process metrics Software quality management
Optimising	Continuous process improvement	Defect prevention Process change management Technology change management

- CMM Shortcomings: CMM does suffer from several shortcomings.
The important among these are the following:
- The most frequent complaint by organisations while trying out the CMM-based process improvement initiative is that they understand what is needed to be improved, but they need more guidance about how to improve it.
- Another shortcoming (that is common to ISO 9000) is that thicker documents, more detailed information, and longer meetings are considered to be better. This is in contrast to the principles of software economics—reducing complexity and keeping the documentation to theminimum without sacrificing the relevant details.
- Getting an accurate measure of an organisation's current maturity level is also an issue. The CMM takes an activity-based approach to measuring maturity; if you do the prescribed set of activities then you are at a certain level. There is nothing that characterises or quantifies whether you do these activities well enough to deliver the intended results.

Comparison Between ISO 9000 Certification and SEI/CMM

- ISO 9000 is awarded by an international standards body. Therefore, ISO 9000 certification can be quoted by an organisation in official documents, communication with external parties, and in tender quotations. However, SEI CMM assessment is purely for internal use.
- SEI CMM was developed specifically for software industry and therefore addresses many issues which are specific to software industry alone.
- SEI CMM goes beyond quality assurance and prepares an organisation to ultimately achieve TQM. In fact, ISO 9001 aims at level 3 of SEI CMM model.

- SEI CMM model provides a list of key process areas (KPAs) on which an organisation at any maturity level needs to concentrate to take it from one maturity level to the next. Thus, it provides a way for achieving gradual quality improvement. In contrast, an organisation adopting ISO 9000 either qualifies for it or does not qualify.

5.7 COMPUTER AIDED SOFTWARE ENGINEERING

•Software is becoming the costliest component in any computer installation.

•CASE tools promise effort and cost reduction in software development and maintenance

5.8 CASE AND ITS SCOPE

•A CASE tool is a generic term used to denote any form of automated support for software engineering

•A CASE tool can mean any tool used to automate some activity associated with software development.

•Some of these 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 objectives in using any CASE tool are:

- ✓ To increase productivity.
- ✓ To help produce better quality software at lower cost

5.9 CASE ENVIRONMENT

•The true power of a tool set can be realized only when these set of tools are integrated into a common framework or environment.

• If the different CASE tools are not integrated, then the data generated by one tool would have to input to the other tools.

•This may also involve format conversions as the tools developed by different vendors are likely to use different formats. This results in additional effort of exporting data from one tool and importing to another.

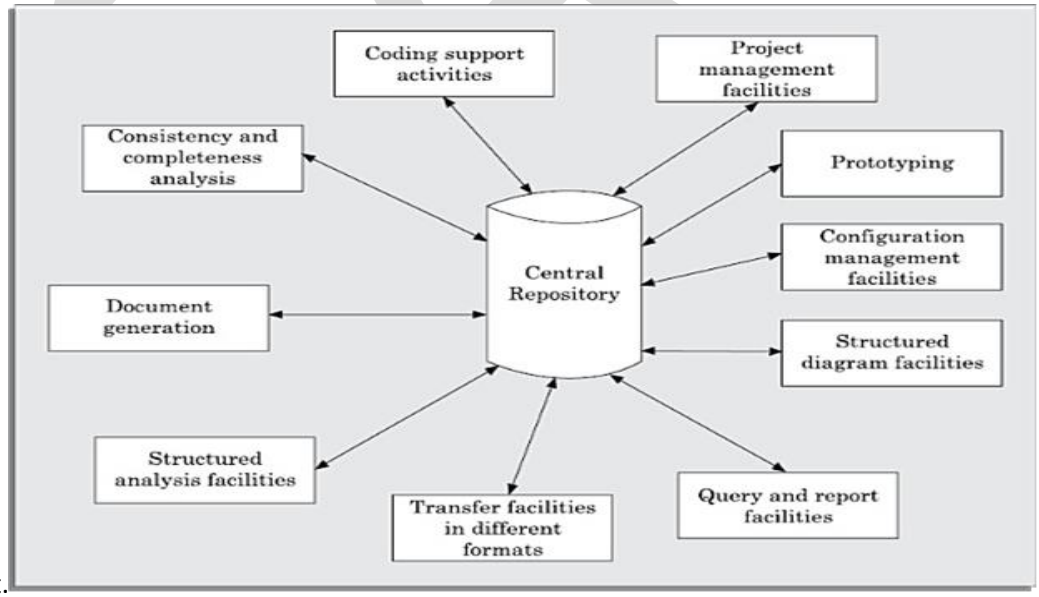
•CASE tools are characterized by the stages of software development life cycle on which they focus.

•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.

•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 facilitates the automation of the step-by-step methodologies for software development.



•A programming environment is an integrated collection of tools to support only the coding phase of software development.

•The tools commonly integrated in a programming environment are a text editor, a compiler, and a debugger.

•The different tools are integrated to the extent that once the compiler detects an error; the editor takes automatically goes to the statements in error and the error statements are highlighted. Examples of popular programming environments are Turbo C environment, Visual Basic, Visual C++, etc.

Benefits of CASE

•A key benefit arising out of the use of a CASE environment is cost saving through all developmental phases. Different studies carry out to measure the impact of CASE, put the effort reduction between 30 per cent and 40 per cent.

•Use of CASE tools leads to considerable improvements in 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 is considerably reduced.

•CASE tools help produces 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 are reduced to a great extent.

•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.

CASE SUPPORT IN SOFTWARE LIFE CYCLE

Prototyping Support

•The prototyping CASE tool's requirements are as follows:

1. Define user interaction.
2. Define the system control flow.
3. Store and retrieve data required by the system.
4. Incorporate some processing logic.

•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, a prototyping CASE tool should support the user to create a GUI using a graphics editor. The user should be allowed to define all data entry forms, menus and controls.

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

•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

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

•The CASE tool should support effortlessly drawing analysis and design diagrams.

•The CASE tool should support drawing fairly complex diagrams and preferably through a hierarchy of levels.

•It should provide easy navigation through different levels and through design and analysis.

•The tool must support completeness and consistency checking across the design and analysis and through all levels of analysis hierarchy

Code Generation

•The CASE tool should support generation of module skeletons or templates in one or more popular languages. It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format.

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

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

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

Test Case Generator

The CASE tool for test case generation should have the following features:

•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

•Optimal configuration of CASE tool in the existing hardware capabilities

Documentation Support

•It should be possible to export text, graphics, tables, data dictionary reports to the DTP package in standard forms such as PostScript.

Project Management

•It 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 tool should allow exchange of information for reusability of design.

•Reverse Engineering Support

•The tool should support generation of structure charts and data dictionaries from the existing source codes.

Tutorial and Help

•The tutorial should cover all techniques and facilities through logically classified sections

Data Dictionary Interface

•The data dictionary interface should provide view and update access to the entities and relations stored in it.

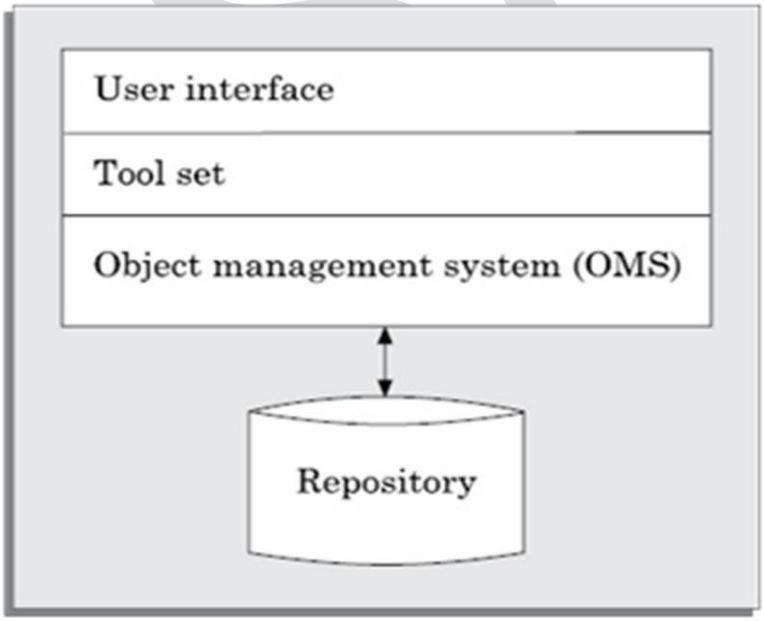
TOWARDS SECOND GENERATION CASE TOOL

- Intelligent diagramming support
- Integration with implementation environment
- Data dictionary standards
- Customization support:

ARCHITECTURE OF A CASE ENVIRONMENT

•The architecture of a typical modern CASE environment is shown diagrammatically in Figure 12.2.

•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 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 into the underlying storage management system (repository).

The commercial relational database management systems are geared towards supporting large volumes of information structured as simple relatively short records.

There are a few types of entities but large number of instances. By contrast, CASE tools create a large number of entity and relation types with perhaps a few instances of each. Thus, the object management system takes care of appropriately mapping these entities into the underlying storage management system.