USN

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 3 – November 2019

Sub:	Software Engineering				Sub Code:	18CS35	Branch:	ISE
Date:	16/11/19	Duration:	90 min's	Max Marks: 50	Sem / Sec:	III A,B,C		OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1 (a)	List and explain the 'Lehman's Laws' concerning system change.	[06]	CO1	L2
(b)	Explain software reengineering process with suitable diagram. State the activities of reengineering process and preventive maintenance by refactoring.	[04]	CO1	L2
2 (a)	Explain the four strategic options of legacy system Management.	[04]	CO1	L2
(b)	With appropriate block diagram, explain the system evolution process.	[06]	CO4	L2
3 (a)	What is software pricing? List and briefly explain the affecting software pricing.	[05]	CO1	L1
(b)	Explain the project scheduling using Activity bar chart and staff allocation chart.	[05]	CO1	L2
4 (a)	Write a note on software measurements and metrics with reference to predictor and control measurements	[05]	CO4	L1
(b)	Explain the different types of software maintenance and discuss why maintenance costs are generally expensive.	[05]	CO4	L2

P.T.O

USN

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 3 – November 2019

Sub:	Software Engineering				Sub Code:	18CS35	Branch:	ISE
Date:	16/11/19	Duration:	90 min's	Max Marks: 50	Sem / Sec:	III A,B,C		OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1 (a)	List and explain the 'Lehman's Law' concerning system change .	[06]	CO1	L2
(b)	Explain software reengineering process with suitable diagram. State the activities of reengineering process and preventive maintenance by refactoring .	[04]	CO1	L2
2 (a)	Explain the four strategic options of legacy system Management.	[04]	CO1	L2
(b)	With appropriate block diagram, explain the system evolution process.	[06]	CO4	L2
3 (a)	What is software pricing? List and briefly explain the affecting software pricing.	[05]	CO1	L1
(b)	Explain the project scheduling using Activity bar chart and staff allocation chart.	[05]	CO1	L2
4 (a)	Write a note on software measurements and metrics with reference to predictor and control measurements. .	[05]	CO5	L1
(b)	Explain the different types of software maintenance and discuss why maintenance costs are generally expensive.	[05]	CO4	L2

P.T.O

		MARKS	CO	RBT
5 (a)	Explain various inspection checklists for software inspection process.	[05]	CO5	L2
(b)	Explain test driven activities with block diagram and its benefits	[05]	CO1	L2
6	Explain the COCOMO – II estimation model with reference to its sub models	[10]	CO2	L2
7 (a)	Explain the ISO 9001 standards framework. Explain the core processes in the standards.	[06]	CO4	L2
(b)	Explain software quality attributes(external and internal) and also distinguish between product standards and process standards.	[04]	CO5	L2

		MARKS	CO	RBT
5 (a)	Explain various inspection checklists for fault classes and explain the review process in detail.	[05]	CO5	L2
(b)	What are product and process metrics. List down the product metrics you are familiar with , with reference to object oriented metrics.	[05]	CO1	L2
6	Explain the COCOMO – II estimation model with reference to its sub models.	[10]	CO2	L2
7 (a)	Explain the ISO 9001 standards framework. Explain the core processes in the standards.	[06]	CO4	L2
(b)	. Explain software quality attributes(external and internal) and also distinguish between product standards and process standards	[04]	CO4	L2

5 (a)	Explain various inspection checklists for fault classes and explain the review process in detail. b.) What are product and process metrics. List down the product metrics you are familiar with , with reference to object oriented metrics	[05]	CO5	L2

CMR
INSTIT UTE OF
TECHNLOGY

USN

Third Internal Test-November 2019

Sub:	**Software Engineering**				Sub Code:	18CS35	Branch:	**ISE**
Date:	16/11/2019	Duration:	90 min's	Max Marks: 50	Sem/Sec:	III A ,B,C		OBE

<u>Scheme and Solution</u>

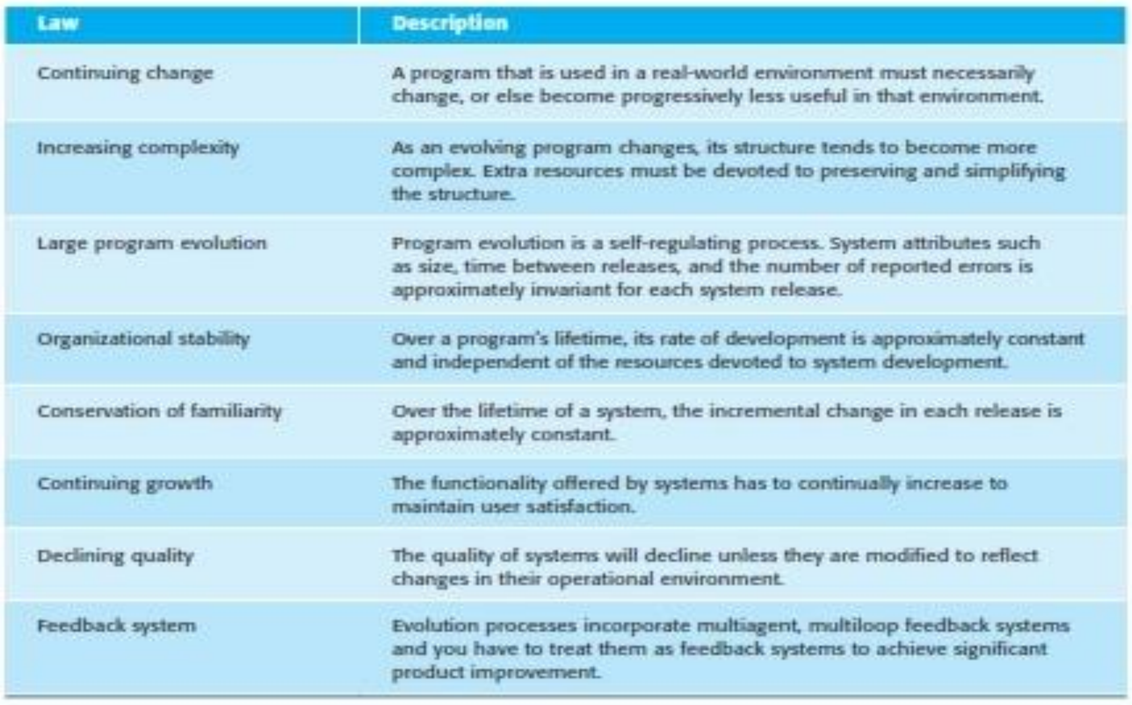
MARKS

1 (a)	List and explain the 'Lehman's Law' concerning system change .	[06]

Program evolution dynamics is the study of system change. In the 1970s and 1980s, Lehman and Belady (1985) carried out several empirical studies of system change with a view to understanding more about characteristics of software evolution. The work continued in the 1990s as Lehman and others investigated the significance of feedback in evolution processes (Lehman, 1996; Lehman et al., 1998; Lehman et al.,2001). From these studies, they proposed 'Lehman's laws' concerning system change.

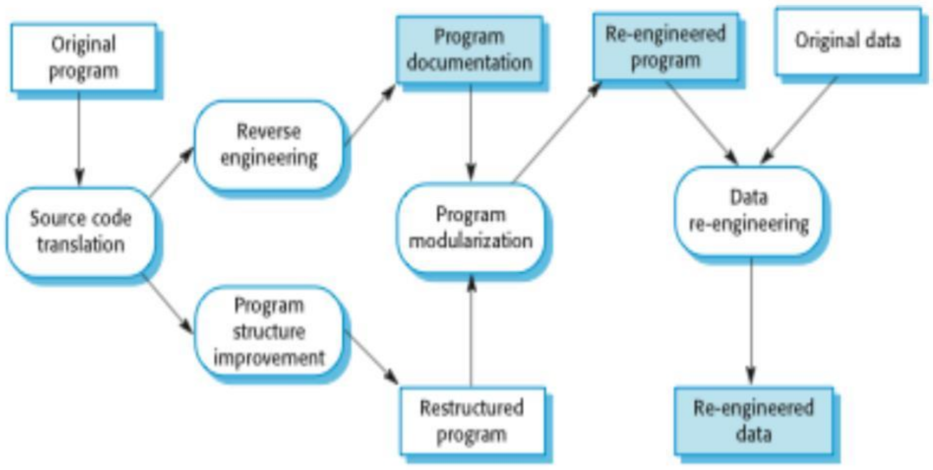
Lehman and Belady claim these laws are likely to be true for all types of large organizational software systems (what they call E-type systems). These are systems in which the requirements are changing to reflect changing business needs. New releases of the system are essential for the system to provide business value.

'Lehman's Law'



Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

(b)	Explain software reengineering process with suitable diagram. State the activities of reengineering process and preventive maintenance by refactoring.		[04]



The activities in this reengineering process are as follows:

Source code translation: Using a translation tool, the program is converted from an old programming language to a more modern version of the same language or to a different language.

Reverse engineering: The program is analyzed and information extracted from it. This helps to document its organization and functionality. Again, this process is usually completely automated.

Program structure improvement: The control structure of the program is analyzed and modified to make it easier to read and understand. This can be partially automated but some manual intervention is usually required.

Program modularization: Related parts of the program are grouped together and, where appropriate, redundancy is removed. In some cases, this stage may involve architectural refactoring (e.g., a system that uses several different data stores may be to use a single repository). This is a manual process.

Data reengineering: The data processed by the program is changed to reflect program changes. This may mean redefining database schemas and converting existing databases to the new structure.

2 (a)	Explain the four strategic options of legacy system Management.		[04]

Most organizations usually have a portfolio of legacy systems that they use, with a limited budget for maintaining and upgrading these systems. They have to decide how to get the best return on their investment. This involves making a realistic assessment of their legacy systems and then deciding on the most appropriate strategy for evolving these systems. There are four strategic options:

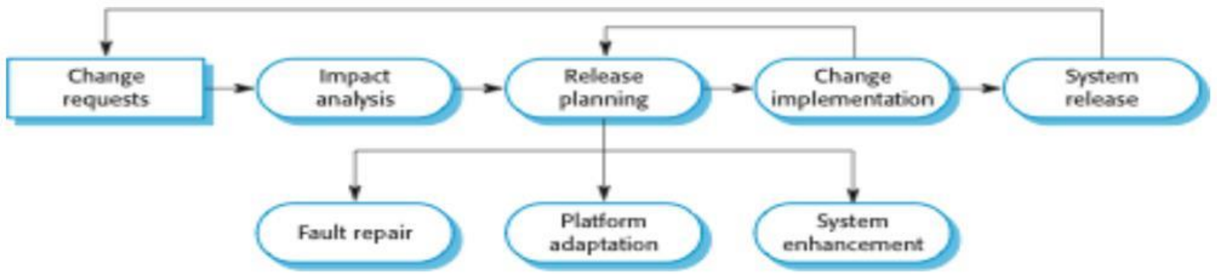
a. *Scrap the system completely* This option should be chosen when the system is not making an effective contribution to business processes. This commonly occurs when business processes have changed since the system was installed and are no longer reliant on the legacy system.

b. *Leave the system unchanged and continue with regular maintenance* This option should be chosen when the system is still required but is fairly stable and the system users make relatively few change requests.

c. *Reengineer the system to improve its maintainability* This option should be chosen when the system quality has been degraded by change and where a new change to the system is still being proposed. This process may include developing new interface components so that the original system can work with other, newer systems.

(b)	**With appropriate block diagram, explain the system evolution process.**		[06]

The process includes the fundamental activities of change analysis, release planning, system implementation, and releasing a system to customers.



- The cost and impact of these changes are assessed to see how much of the system is affected by the change and how much it might cost to implement the change.
- If the proposed changes are accepted, a new release of the system is planned.
- During release planning, all proposed changes (fault repair, adaptation, and new functionality) are considered.
- A decision is then made on which changes to implement in the next version of the system.
- The changes are implemented and validated, and a new version of the system is released.
- The process then iterates with a new set of changes proposed for the next release.
- Change implementation can be thought of as an iteration of the development process, where the revisions to the system are designed, implemented, and tested.
- However, a critical difference is that the first stage of change implementation may involve program understanding, especially if the original system developers are not responsible for change implementation.
- During this program understanding phase, it becomes necessary to understand how the program is structured, how it delivers functionality, and how the proposed change might affect the program.
- This understanding is required to make sure that the implemented change does not cause new problems when it is introduced into the existing system.
- The change implementation stage of this process should modify the system specification, design, and implementation to reflect the changes to the system
- During the evolution process, the requirements are analyzed in detail and implications of the changes emerge that were not apparent in the earlier change analysis process.
- This means that the proposed changes may be modified and further customer discussions may be required before they are implemented.

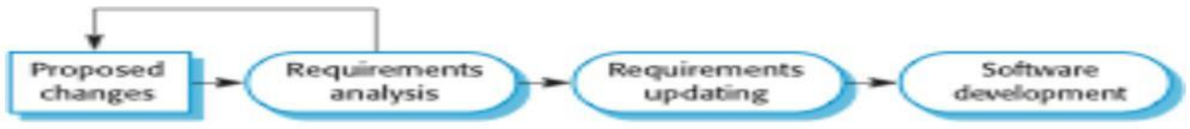


Fig Change Implementation

Change requests can arise for 3 reasons:
1. If a serious system fault occurs that has to be repaired to allow normal operation to continue.
2. If changes to the systems operating environment have unexpected effects that disrupt normal operation.
3. If there are unanticipated changes to the business running the system, such as the emergence of new competitors or the introduction of new legislation that affects the system.
• Rather than modify the requirements and design, it is possible to make an emergency fix to the program to solve the immediate problems.
• Problems may arise in situations in which there is a handover from a development team to a separate team responsible for evolution.

There are two potentially problematic situations:

1. Where the development team has used an agile approach but the evolution team is unfamiliar with agile methods and prefers a plan-based approach. The evolution team may expect detailed documentation to support evolution and this is rarely produced in agile processes.

2. Where a plan-based approach has been used for development but the evolution team prefers to use agile methods. In this case, the evolution team may have to start from scratch developing automated tests and the code in the system may not have been refactored and simplified as is expected in agile development.

3 (a)	**What is software pricing? List and briefly explain the affecting software pricing.**		[05]

In principle, the price of a software product to a customer is simply the cost of development plus profit for the developer. Figure above shows the factors affecting software pricing. It is essential to think about organizational concerns, the risks associated with the project, and the type of contract that will be used. These may cause the price to be adjusted upwards or downwards
Because of the organizational considerations involved, deciding on a project price should be a group activity involving marketing and sales staff, senior management, and project managers

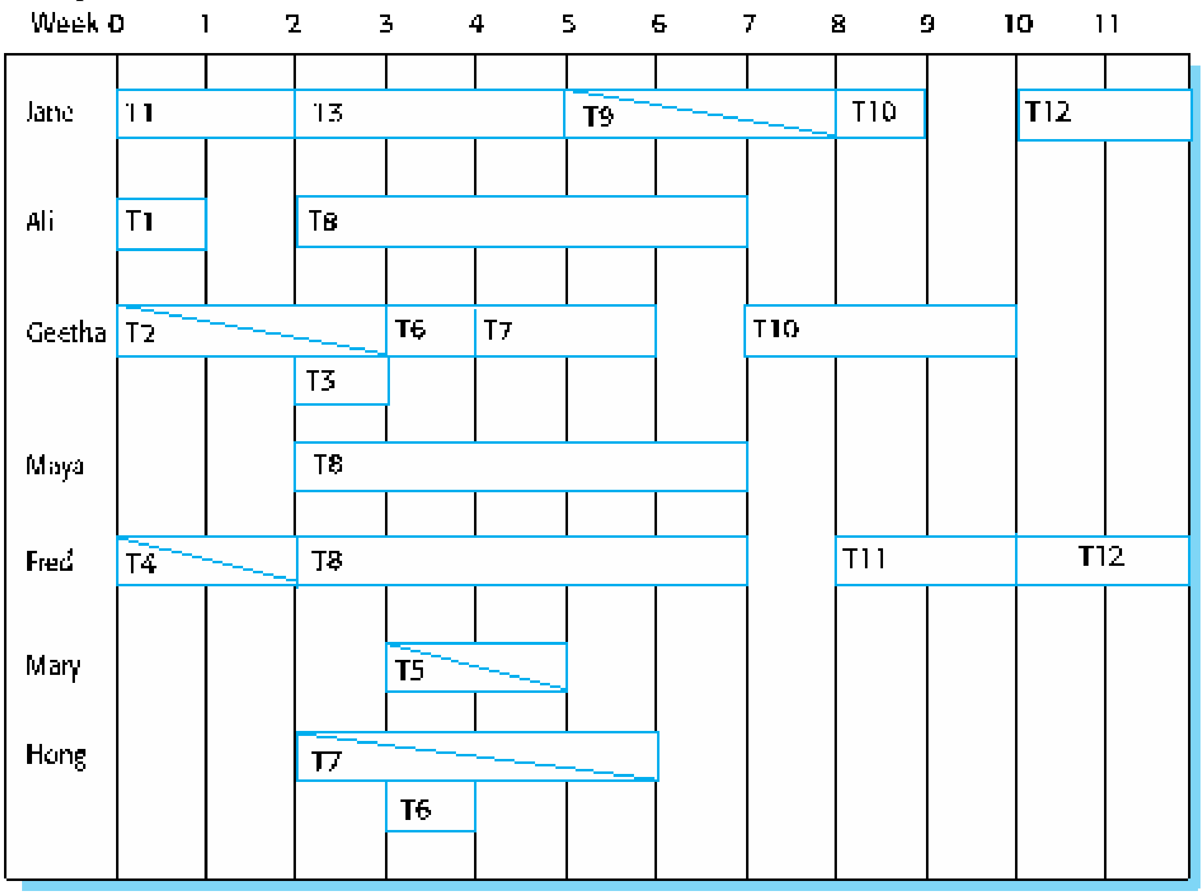
Factor	**Description**
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash

	flow is more important than profit in difficult economic times.
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

(b)	**Explain the project scheduling using Activity bar chart and staff allocation chart.**		[05]

- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Calendar-based: Bar charts are the most commonly used representation for project schedules. They show the schedule as activities or resources against time.
- Activity networks: Show task dependencies

- Example of a staff allocation chart



4 (a)	**Write a note on software measurements and metrics with reference to predictor and control measurements.** •		[05]

- Any type of measurement which relates to a software system, process or related documentation
- Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.

- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.
- *The time taken for a particular process to be completed*
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- *The resources required for a particular process*
  - Resources might include total effort in person-days, travel costs or computer resources.
- *The number of occurrences of a particular event*
  Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, the number of bug reports in a delivered system and the average number of lines of code modified in response to a requirements change

(b)	**Explain the different types of software maintenance and discuss why maintenance costs are generally expensive.**		[05]

- ✧ Modifying a program after it has been put into use.
- ✧ The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- ✧ Maintenance does not normally involve major changes to the system's architecture.
- ✧ Changes are implemented by modifying existing components and adding new components to the systemFault repairs
  - ▪ Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- ✧ Environmental adaptation
  - ▪ Maintenance to adapt software to a different operating environment
  - ▪ Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- ✧ Functionality addition and modification Modifying the system to satisfy new requirements.
  Usually greater than development costs (2* to 100* depending on the application). Affected by both technical and non-technical factors.
- ✧ Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- ✧ Ageing software can have high support costs (e.g. old languages, compilers etc.).
  It is usually more expensive to add new features to a system during maintenance than it is to add the same features during development
- ✧ A new team has to understand the programs being maintained
- ✧ Separating maintenance and development means there is no incentive for the development team to write maintainable software
- ✧ Program maintenance work is unpopular
- ✧ Maintenance staff are often inexperienced and have limited domain knowledge.
- ✧ As programs age, their structure degrades and they become harder to change

5 (a)	**Explain various inspection checklists for fault classes and explain the review process in detail.**		[06]

- These are peer reviews where engineers examine the source of a system with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.

- They may be applied to any representation of the system (requirements, design,configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.
- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

Fault class	**Inspection check**
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?

(b)	**What are product and process metrics. List down the product metrics you are familiar with , with reference to object oriented metrics.**		[04]

Product Quality Metrics

This metrics include the following −

- Mean Time to Failure
- Defect Density
- Customer Problems
- Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

Process Metrics:

Cost of quality: It is a measure of the performance of quality initiatives in an organization. It's expressed in monetary terms.

1. **Cost of poor quality**: It is the cost of implementing imperfect processes and products.

Cost of poor quality = rework effort/ total effort x 100.

2. **Defect density**: It is the number of defects detected in the software during development divided by the size of the software (typically in KLOC or FP)

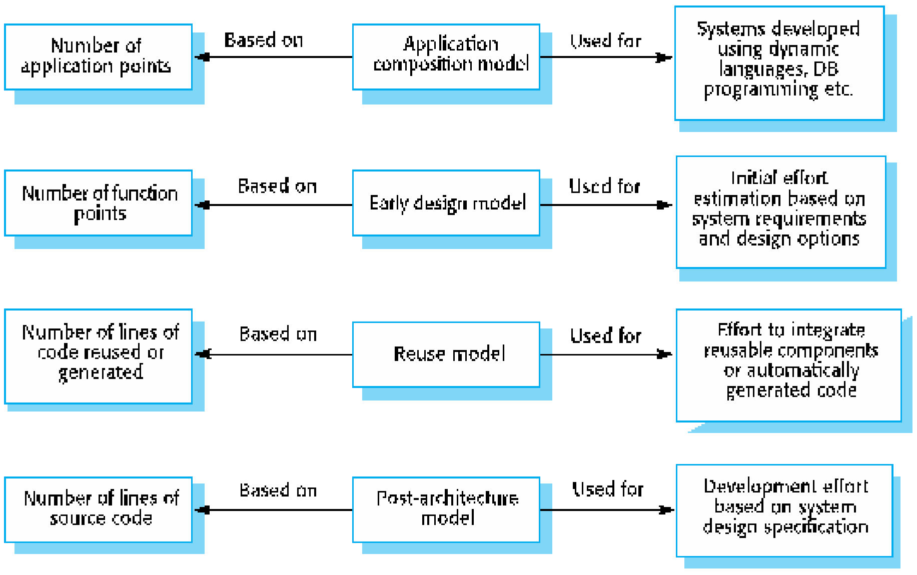
Defect density for a project = Total number of defects/ project size in KLOC or FP

3. **Review efficiency**: defined as the efficiency in harnessing/ detecting review defects in the verification stage.

6	**Explain the COCOMO – II estimation model with reference to its sub models.**		[10]

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
- Application composition model. Used when software is composed from existing parts.
- Early design model. Used when requirements are available but design has not yet started.
- Reuse model. Used to compute the effort of integrating reusable components.
- Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

  The figure below shows the COCOMO estimation models

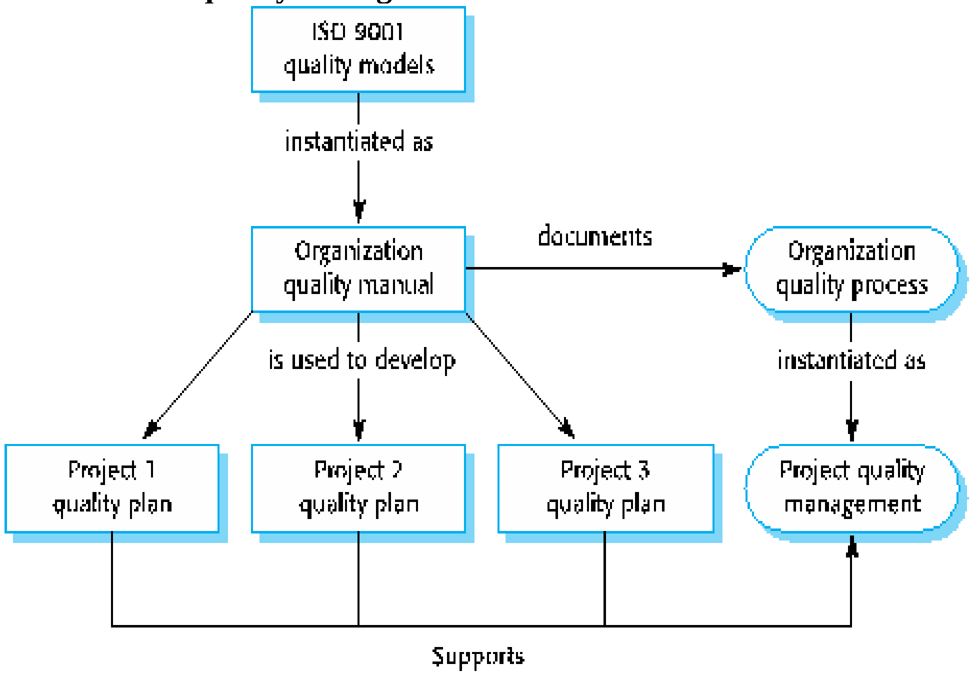


Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

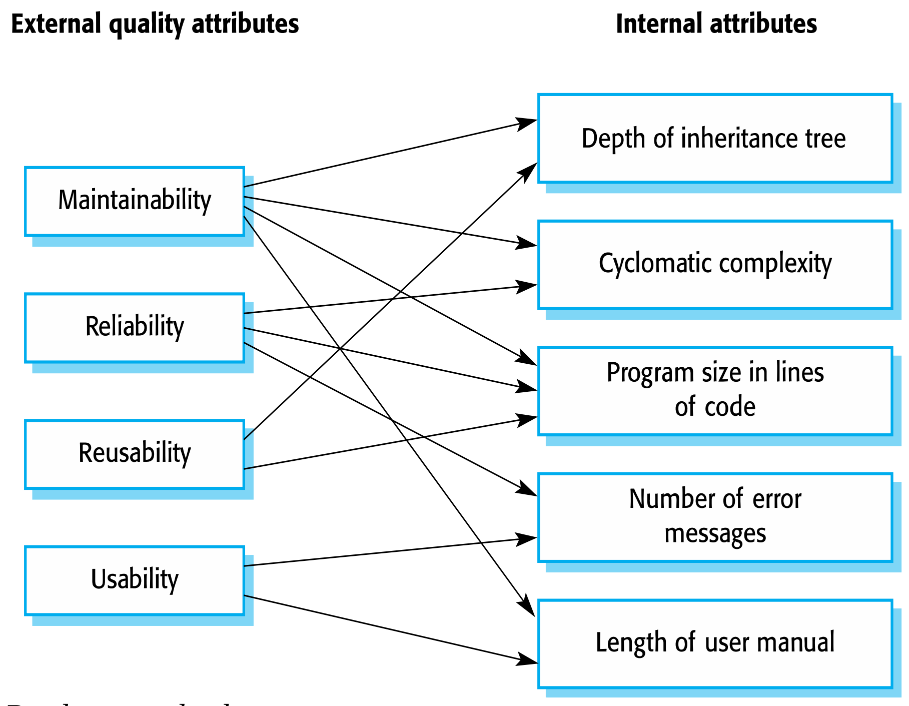
7 (a)	Explain the ISO 9001 standards framework. Explain the core processes in the standards.		[06]

- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
  It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined.
  These should be documented in an organizational quality manual.

ISO 9001 and quality management



(b)	Explain software quality attributes(external and internal) and also distinguish between product standards and process standards		[04]



- *Product standards*
  - Apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.
- *Process standards*
  - These define the processes that should be followed during software development. Process standards may include definitions of specification, design and validation processes, process support tools and a description of the documents that should be written during these processes.