**CMR
INSTITUTE OF
TECHNOLOGY**

Internal Assessment Test 3 – Nov 2019

Solutions

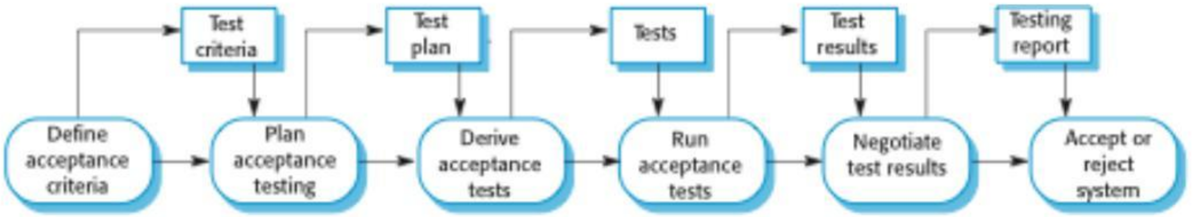
Sub:	Software Engineering							**Code:**	18CS35
Date:	16/ 11/2019	Duration:	90 mins	Max Marks:	50	**Sem:**	III	**Branch:**	CSE

Note: Answer any five questions:

1 **a) Explain the different stages of acceptance testing with a diagram - 6 M** **10M**

Six stages in the acceptance testing process

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system



Define acceptance criteria: This stage should, ideally, take place early in the process before the contract for the system is signed. The acceptance criteria should be part of the system contract and be agreed between the customer and the developer. Detailed requirements maynot be available and there may be significant requirements change during the development process.

Plan acceptance testing: This involves deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule. The acceptance test plan should also discuss the required coverage of the requirements and the order in which system features aretested. It should define risks to the testing process, such as system crashes and inadequate performance, and discuss how these risks can be mitigated.

Derive acceptance tests: Once acceptance criteria have been established, tests have to be designed to check whether or not a system is acceptable. Acceptance tests should aim to testboth the functional and non-functional characteristics (e.g., performance) of the system.

Run acceptance tests: The agreed acceptance tests are executed on the system. Ideally, this should take place in the actual environment where the system will be used, but this may be disruptive and impractical. Therefore, a user testing environment may have to be set up to run these tests. It is difficult to automate this process as part of the acceptance tests may involve testing the interactions between end-users and the system.

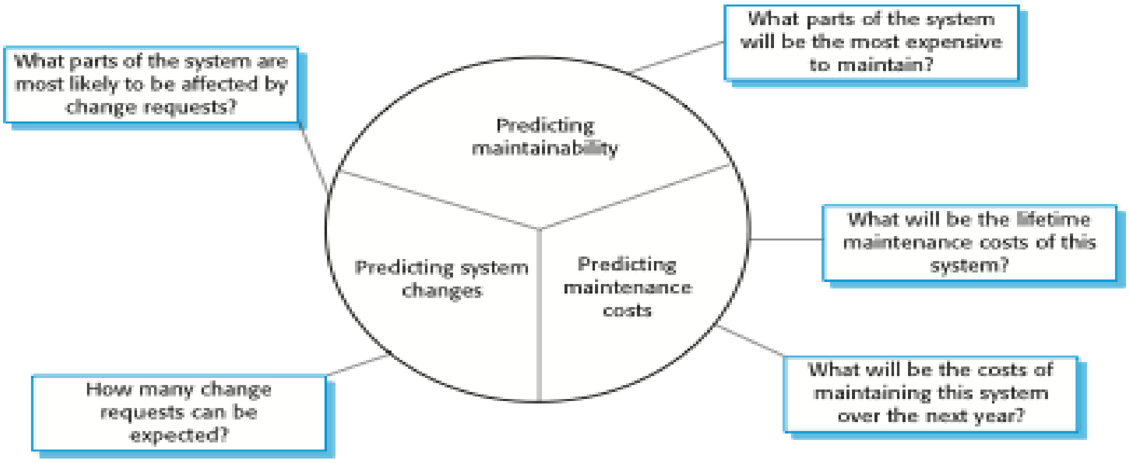
Negotiate test results: It is very unlikely that all of the defined acceptance tests will pass and that there will be no problems with the system. If this is the case, then acceptance testing is complete and the system can be handed over. More commonly, some problems will be discovered. In such cases, the developer and the customer have to negotiate to decide if the system is good enough to be put into use. They must also agree on the developer's response to identified problems.

Reject/accept system: This stage involves a meeting between the developers and the customer to decide on whether the system should be accepted. If the system is not good enough for use, then further development is required to fix the identified problems. Once complete, the acceptance testing phase is repeated.

b) Explain about software maintenance prediction – 4 M

✧ Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs

- Change acceptance depends on the maintainability of the components affected by the change;
- Implementing changes degrades the system and reduces its maintainability;
- Maintenance costs depend on the number of changes and costs of change depend on maintainability.
- Predicting the number of changes requires and understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.
- Factors influencing this relationship are
- Number and complexity of system interfaces;
- Number of inherently volatile system requirements;
- The business processes where the system is used.


What parts of the system are most likely to be affected by change requests?
What parts of the system will be the most expensive to maintain?
Predicting maintainability
What will be the lifetime maintenance costs of this system?
Predicting system changes
Predicting maintenance costs
How many change requests can be expected?
What will be the costs of maintaining this system over the next year?

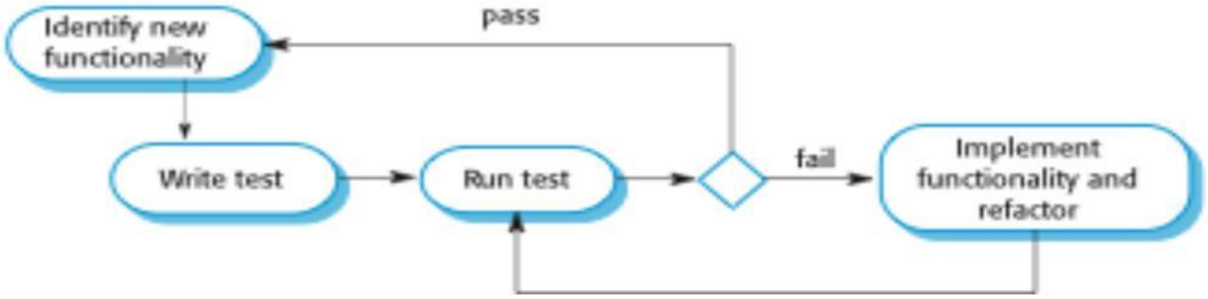
2

a)

10M

b) What is test driven development explain with a diagram also list its benefits. – 10 M

Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
• Tests are written before code and 'passing' the tests is the critical driver of development.
• TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.


Identify new functionality
pass
Write test
Run test
fail
Implement functionality and refactor

The fundamental TDD process is shown in Figure. The steps in the process are as follows:

- You start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- You write a test for this functionality and implement this as an automated test. This means that the test can be executed and will report whether or not it has passed or failed.
- You then run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail. This is deliberate as it shows that the test adds something to the test set.
- You then implement the functionality and re-run the test. This may involve refactoring existing code to improve it and add new code to what's already there.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.

Benefits of test-driven development are:

a. **Code coverage:** In principle, every code segment that you write should have at least one associated test. Therefore, you can be confident that all of the code in the system has actually been executed. Code is tested as it is written, so defects are discovered early in the development process.

b. **Regression testing:** A test suite is developed incrementally as a program is developed. Regression tests can be run to check that changes to the program have not introduced new bugs.

c. **Simplified debugging:** When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified. Debugging tools need not be used to locate the problem. Reports of the use of test-driven development suggest that it is hardly ever necessary to use an automated debugger in test-driven development

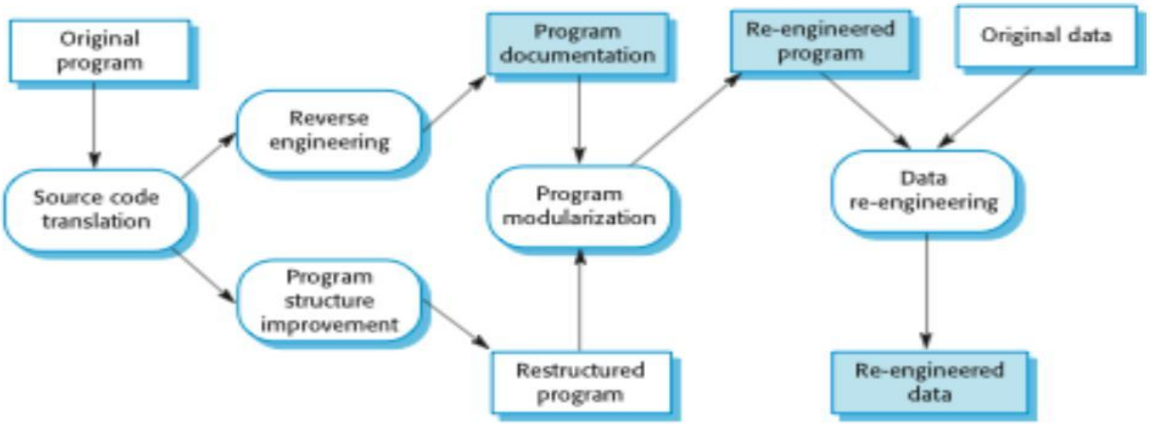
d. **System documentation:** The tests themselves act as a form of documentation that describes what the code should be doing. Reading the tests can make it easier to understand the code.

   Test-driven development is of most use in new software development where the functionality is either implemented in new code or by using well-tested standard libraries. If you are reusing large code components or legacy systems then you need to write tests for these systems as a whole. Test-driven development may also be ineffective with multi-threaded systems. The different threads may be interleaved at different times in different test runs, and so may produce different results.

3

a) Explain the process of software reengineering. Also mention the advantages – 6 M

10M



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.

b) Explain the four strategic options of legacy system management – 4 M

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.

d. *Replace all or part of the system with a new system* This option should be chosen when factors, such as new hardware, mean that the old system cannot continue in operation or where off-the-shelf systems would allow the new system to be developed at a reasonable cost. In many cases, an evolutionary replacement strategy can be adopted in which major system components are replaced by off the-shelf systems with other components reused wherever possible.

4

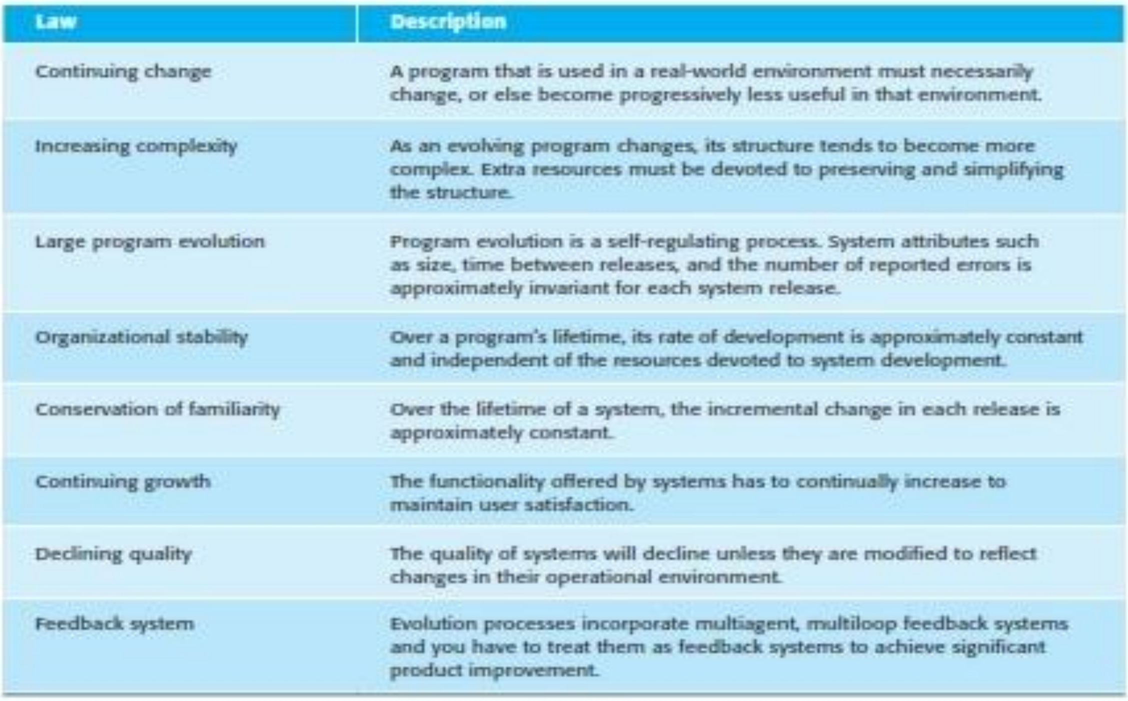
a) **Define "program evolution dynamics" and the different Lehman's laws concerning system change – 10M**

10M

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.

5

a) Explain various inspection checklist for software inspection process – 6M

10M

- 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) Explain different types of software standards and mention their importance – 4M

Standards define the required attributes of a product or process. They play an important role in quality management. Standards may be international, national, and organizational or project standards. They may not be seen as relevant and up-to-date by software engineers. They often involve too much bureaucratic form filling.

If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards. Involve practitioners in development. Engineers should understand the rationale underlying a standard.

Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners .Detailed standards should have specialized tool support. Excessive clerical work is the most significant complaint against standards.

- Web-based forms are not good enough.

✧ *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.

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Importance of standards

✧ Encapsulation of best practice- avoids repetition of past mistakes.

✧ They are a framework for defining what quality means in a particular setting i.e. that organization’s view of quality.

✧ They provide continuity - new staff can understand the organisation by understanding the standards that are used.

6

a) List and explain the factors affecting software pricing – 5M

10M

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 sou code and reuse it in other projects. The price charged may then be less than if th 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 b to make a smaller than normal profit or break even than to go out of business. C 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 i new segment of the software market. Accepting a low profit on one project may the organization the opportunity to make a greater profit later. The experience g may also help it develop new products.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to v contract. After the contract is awarded, high prices can be charged for changes t requirements.

b)Explain the plan driven development approach to software engineering – 5M

Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail. Plan-driven development is based on engineering project management techniques and is the 'traditional' way of managing large software development projects.A project plan is created that records the work to be done, who will do it, the development schedule and the work products. Managers use the plan to support project decision making and as a way of measuring progress. The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway. The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.

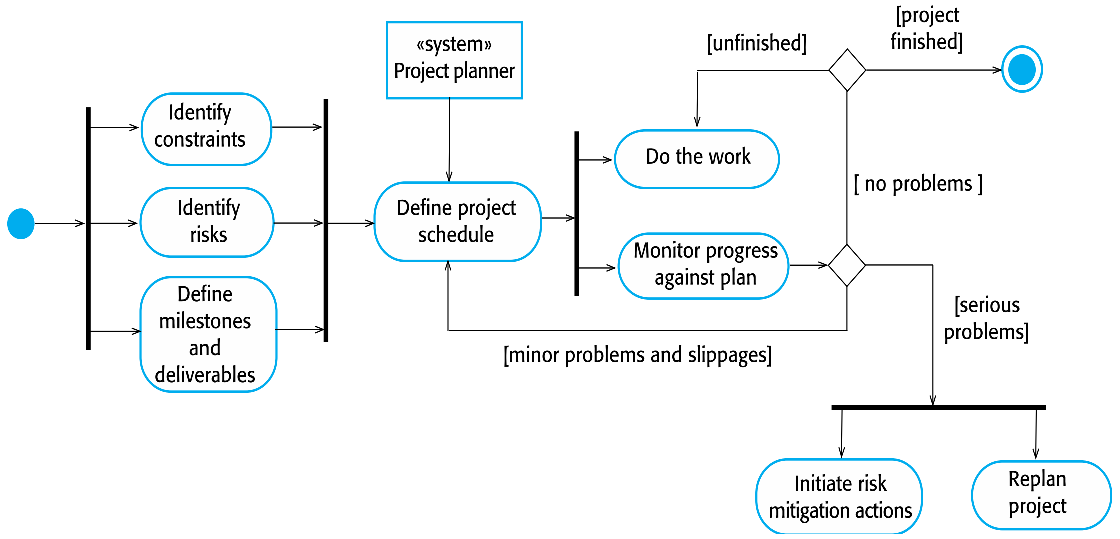
Plan sections are the following

- Introduction
- Project organization
- Risk analysis
- Hardware and software resource requirements
- Work breakdown

- Project schedule
- Monitoring and reporting mechanisms

Project plan supplements

Plan	Description
Configuration management plan	Describes the configuration management procedures and structures to be used.
Deployment plan	Describes how the software and associated hardware (if required) will be deployed in the customer's environment. This should include a plan for migrating data from existing systems.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.



7

a) Explain COCOMO II estimation model – 8M

An empirical model based on project experience. Well-documented, ‘independent’ model which is not tied to a specific software vendor. Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
COCOMO 2 takes into account different approaches to software development, reuse, etc.
COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
The sub-models in COCOMO 2 are:

a. **Application composition model**. Used when software is composed from existing parts. Supports prototyping projects and projects where there is extensive reuse. Based on standard estimates of developer productivity in application (object) points/month. Takes software tool use into account.
Formula is
PM = (NAP ´ (1 - %reuse/100)) / PROD
PM is the effort in person-months, NAP is the number of application points and PROD is the productivity.

Developer’s experience and capability	**Very low**	**Low**	**Nominal**	**High**	**Very high**
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

b. **Early design model.** Used when requirements are available but design has not yet started.Estimates can be made after the requirements have been agreed.Based on a standard formula for algorithmic models
PM = A ´ Size^B ´ M where
M = PERS ´ RCPX ´ RUSE ´ PDIF ´ PREX ´ FCIL ´ SCED;
A = 2.94 in initial calibration,
Size in KLOC,
B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

✧ Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

- RCPX - product reliability and complexity;
- RUSE - the reuse required;
- PDIF - platform difficulty;
- PREX - personnel experience;
- PERS - personnel capability;
- SCED - required schedule;
- FCIL - the team support facilities.

c. **Reuse model**. Used to compute the effort of integrating reusable components. Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code. There are two versions:

- Black-box reuse where code is not modified. An effort estimate (PM) is computed.
- White-box reuse where code is modified. A size estimate equivalent to the

number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates 1 -For generated code:

PM = (ASLOC * AT/100)/ATPROD

- ASLOC is the number of lines of generated code
- AT is the percentage of code automatically generated.
- ATPROD is the productivity of engineers in integrating this code.

Reuse model estimates 2 - When code has to be understood and integrated:

ESLOC = ASLOC * (1-AT/100) * AAM.

- ASLOC and AT as before.
- AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.

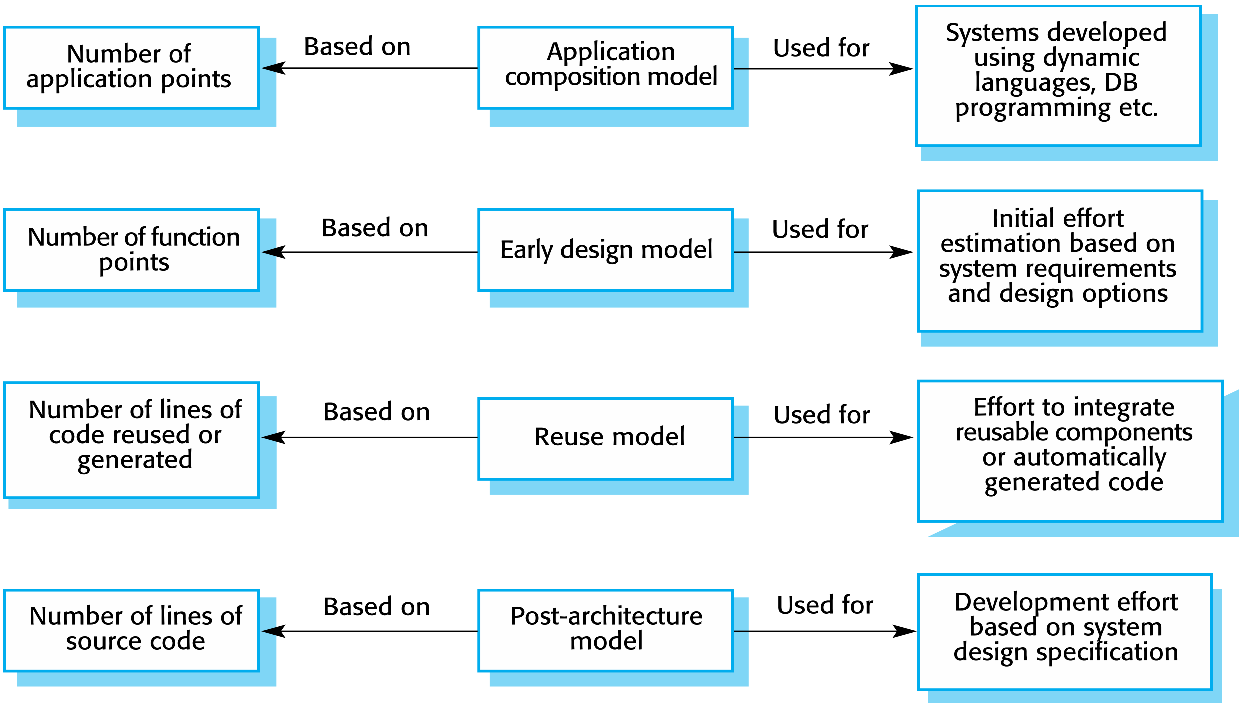
d. **Post-architecture model.** Used once the system architecture has been designed and more information about the system is available.

- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.
- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.
  - Precedenteness - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low .(5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)
- Scale factor is therefore 1.17.

Scale factor	**Explanation**
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Process maturity	Reflects the process maturity of the organization. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.

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

b) What are the key stages in component measurement process/ analysis – 2M

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.
- The diagram bellows the stages in process of product measurement

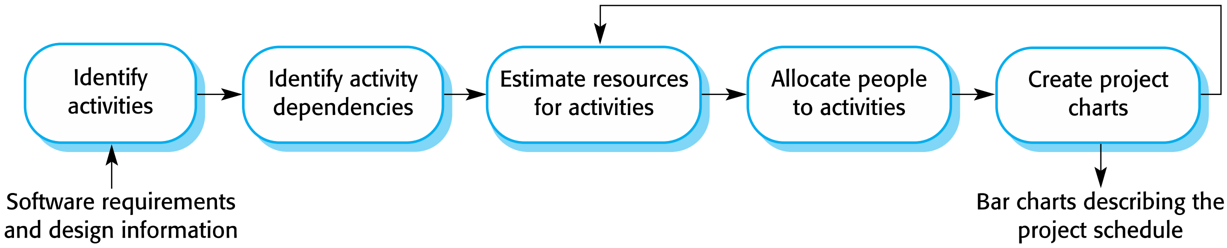
8 a) Explain the project scheduling using Activity bar chart and staff allocation chart. 10M

Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed. You estimate the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified. You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

Project scheduling activities

- ✧ Split project into tasks and estimate time and resources required to complete each task.
- ✧ Organize tasks concurrently to make optimal use of workforce.
- ✧ Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- ✧ Dependent on project managers intuition and experience.

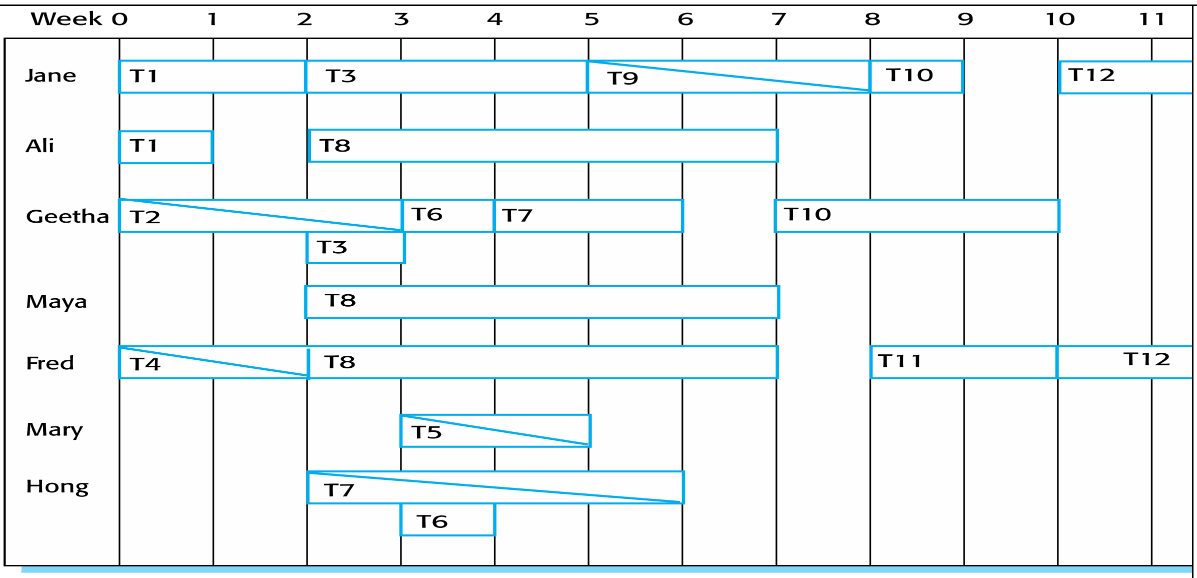
The project scheduling process


Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software requirements and design information
Bar charts describing the project schedule

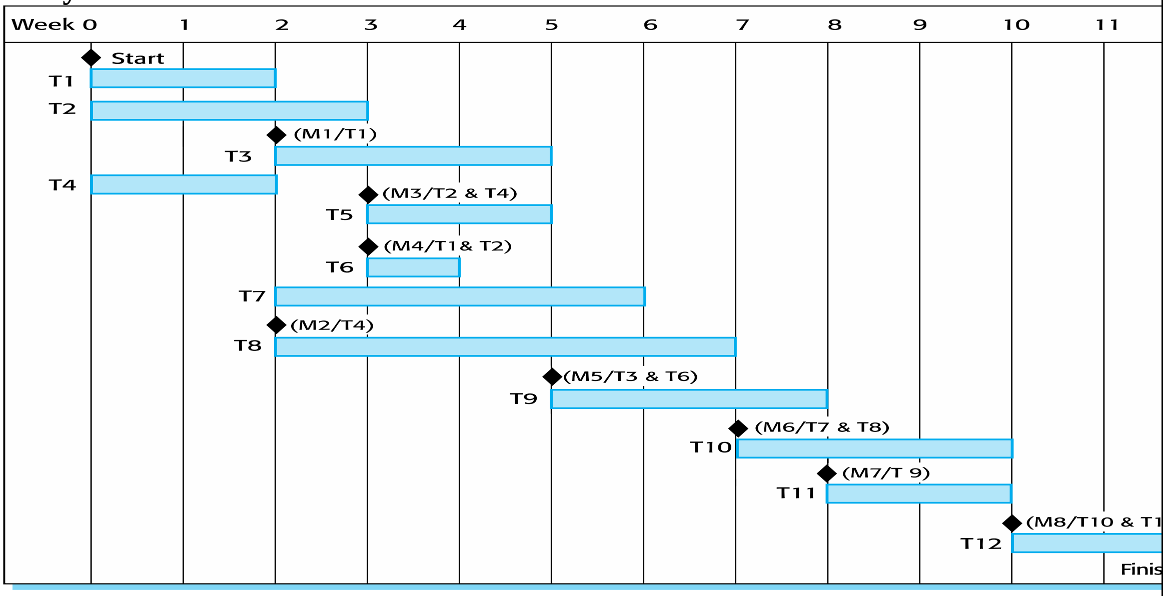
- ✧ Estimating the difficulty of problems and hence the cost of developing a solution is hard. Productivity is not proportional to the number of people working on a task. Adding people to a late project makes it later because of communication overheads. The unexpected always happens. Always allow contingency in planning.

  - ○ 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



Activity bar chart


Week 0
Start
T1
T2
(M1/T1)
T3
T4
(M3/T2 & T4)
T5
(M4/T1& T2)
T6
T7
(M2/T4)
T8
(M5/T3 & T6)
T9
(M6/T7 & T8)
T10
(M7/T 9)
T11
(M8/T10 & T1
T12
Finis