USN | | | | | | | | | | |

IAT 3 – May 2018- Scheme and Solution

Sub:	Software Engineering					Sub Code:	15CS42	Branch:	CSE	
Date:	21/05/2018	Duration:	90 mins	Max Marks:	50	Sem/Sec:	4 (A,B,C)			OBE

Answer **FOUR** FULL questions selecting AT LEAST ONE question **FROM EACH PART**

MARKS | CO | RBT

PART A

1 (a) List and explain the Inspection Checklist. [10] CO4 L2

(Data faults-2M, Control faults-2M, I/O faults-2M, Interface faults-2M, Storage management faults-1M, Exception management faults-1M)

Inspection checklist

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?
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter type match? • Are parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?

Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?		
Exception management faults	• Have all possible error conditions been taken into account?		

OR

2 (a) Explain examples of product metrics. [10] CO4 L2

(Any 10 examples-10x1M)

Static software product metrics :-

(i) Fan-in / Fan-out

Fan-in is measure of no. of functions that call another function.

Fan-out is no. of functions that are called ~~out~~ by another function.

(ii) Length of Code

It is measure of size of a program

(iii) Cyclomatic complexity

It is measure of control complexity of program

(iv) Length of identifiers

It is measure of average length of identifiers (names for variables, classes, methods etc.)

(v) Depth of conditional nesting

This is a measure of the depth of nesting of if-statements in a program

(vi) Fog index

This is a measure of the average length of words and sentences in documents.

Chidamber and Kemerer's (CK) Suite of Object-oriented metrics :-

(i) WMC (Weighted methods per class)
No. of methods in each class, weighted by complexity of each method.

(ii) DIT (Depth of Inheritance tree)
No. of discrete levels in the inheritance tree where subclasses inherit attributes and methods from superclass

(iii) NOC (Number of children)
No. of immediate subclasses in a class.

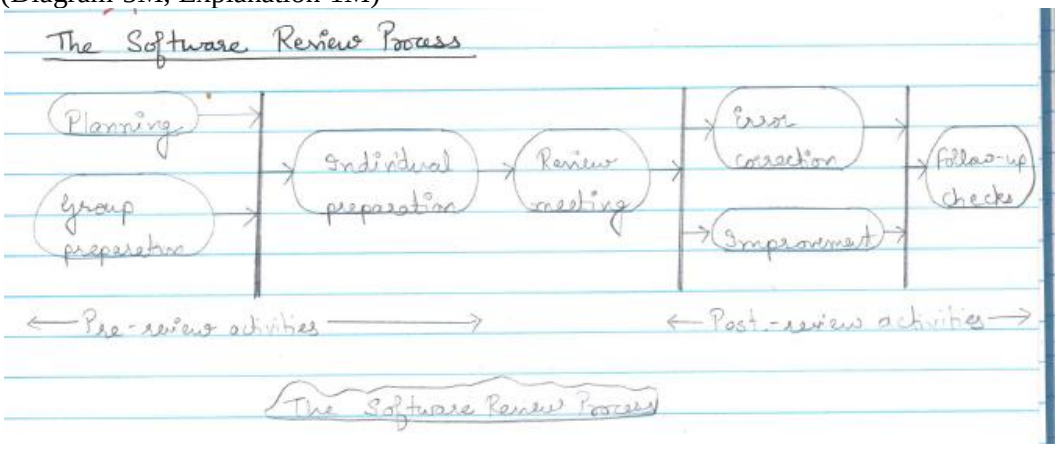
(iv) CBO (Coupling between object classes)
Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists.
A high value of CBO means classes are highly dependent

(v) RFC (Response for a Class)
No. of ~~messages~~ methods that could potentially be executed in response to a message received by an object of that Class

(vi) LCOM (Lack of cohesion in methods)
It is the difference between the number of method pairs without shared attributes and no. of method pairs with shared attributes.

PART B

3 (a) Explain software review process with a neat diagram. [6]
(Diagram-5M, Explanation-1M)



The Software Review Process

CO4 L2

Review process has three phases :

(i) Pre-review activities – It contains review planning and preparation. Review planning involves setting up a review team, arranging a time and place for review, and distributing the documents to be reviewed. In review preparation, individual team members read and understand the software or documents and relevant standards to find errors, omissions, and departures from standards. If meeting can't be attended by someone, they can write comments.

(ii) The review meeting – Review should be relatively short – two hours almost. An author of document or program being reviewed should 'walk through' the document. One team member should chair the review and another should formally record all review decisions and actions to be taken. All written comments should be considered. The review chair should sign a record of comments and actions during the review.

(iii) Post review activities – Issues & problems raised during the review must be addressed (fixing bugs, refactoring, rewriting).

Sometimes, a further review will be required to check that changes made cover all the previous review comments.

(b) Explain pair programming and its advantages. [4] CO4 L2

(Definition-2M, Advantages-2M)

Pair Programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
- Programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all the team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.

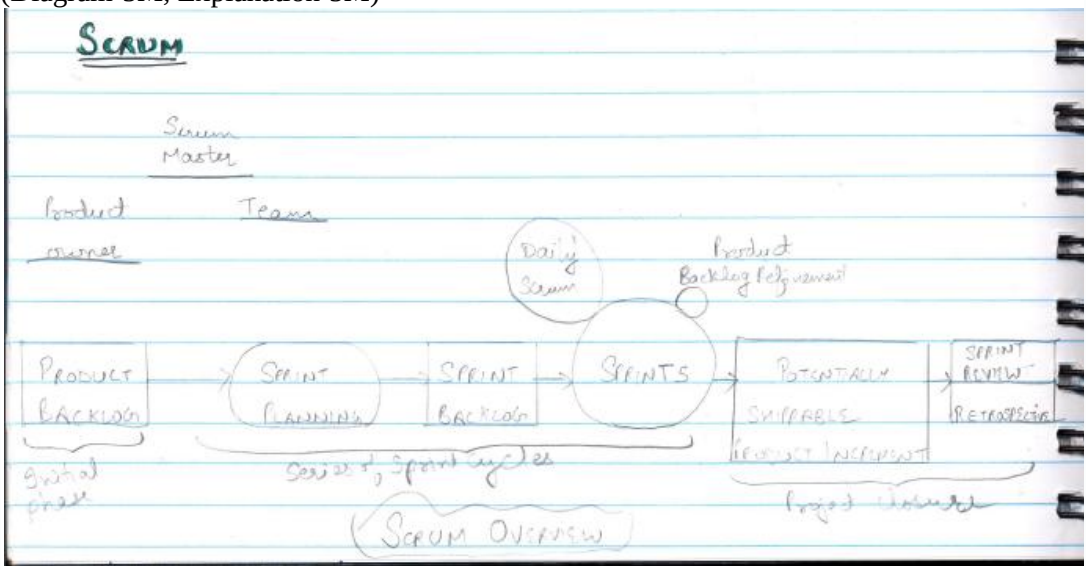
Advantages of pair programming:

- It supports the idea of collective ownership and responsibility for the system. Team has collective responsibility for resolving problems (not blaming).
- It acts as an informal review process because each line of code is looked at by atleast two people.
- It helps support refactoring, which is a process of software improvement.

OR

4 (a) Explain the Scrum process in detail. [10] CO1 L2

(Diagram-5M, Explanation-5M)



- The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.
- There are three main phases in Scrum:
  - Initial phase – It is an outline planning phase where you establish the general objectives for the project & design the s/w architecture.
  - Series of Sprint cycles – where each sprint cycle develops an increment of the system. The Sprints are time-boxed i.e., they end on a specific date whether work has been completed or not, and are never extended.
  - Project closure – It wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

- Detailed activities:-

→ Scrum Roles

- Product Owner – The person responsible for managing the Product Backlog so as to maximize the value of the product
- Team – A cross-functional group of people that is responsible for managing themselves to develop an increment of product every Sprint.
- Scrum master – He is responsible for the Scrum process, its correct implementation, and maximization of its benefits. He is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers & management outside of the team.

→ Product Backlog

The starting point of the planning is product backlog. It is a prioritized list of requirements with estimated times to turn them into completed product functionality.

→ Sprint planning

It is a selection phase that involves all of the project team who work with the customer to select features & functionality to be developed during the sprint.

→ Sprint backlog

It is a list of team's work for a sprint. It may be updated during sprint.

→ The Sprint cycle

- Sprints are fixed length, normally 2-4 weeks. They correspond to the development of a release of system.

- Daily Scrum is a short meeting held daily by each Team during which the team members inspect their work, synchronize their work and progress. They share information, describe their progress since last meeting, problems that have arisen and what is planned for the following day. This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

→ Sprint Review Meeting

A time-boxed two hour meeting (for a two week Sprint) at the end of every sprint where Team collaborates with Product Owner and stakeholders and they inspect the output from the Sprint. This usually starts with a review of completed Product Backlog items, a discussion of opportunities, constraints and risks, and a discussion of what might be the best things to do next (potentially resulting in Product Backlog changes). Only completed functionality can be demonstrated.

→ Sprint Retrospective Meeting

A meeting facilitated by the ScrumMaster at which the complete Team discusses the just-concluded Sprint & determines what could be changed that might make the next sprint more productive.

Scrum Benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments & gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

PART C

5 (a)	Explain principles of Agile development. (12 principles-10M)	[10]	CO1	L2

The Agile Manifesto: Values and Principles

① Customer involvement (work together)
Customers should be clearly involved throughout the development process. Their role is to provide and prioritize new system requirements & to evaluate the iterations of the system.

② Incremental delivery (shorter timescale - weeks to months)
The software is developed in increments with the customer specifying the requirements to be included in each increment.

③ People not process (motivated individuals)
The skills of development team should be recognised and exploited. Team members should be left to develop their own ways of working without prescriptive processes

④ Embrace change
Expect the system requirements to change and so design system to accommodate these changes

⑤ Maintain simplicity
Focus on simplicity in both software being developed and in the development process. Wherever possible, actively work to ~~elimin~~ eliminate complexity from the system.

⑥ Early and continuous delivery of valuable software

⑦ Most efficient and effective way of conveying information to and within a development team is face-to-face conversation.

⑧ Working software is primary measure of progress

⑨ Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

⑩ Continuous attention to technical excellence and good design enhances agility.

⑪ The best architectures, requirements, and designs emerge from self organizing teams.

⑫ At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

OR

6 (a) List and explain practices of extreme programming. [10] | CO1 | L2
(XP Practices- 10x1M)

Extreme programming practices

① Incremental planning

Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. Then developers break these stories into development 'Tasks'. See figures on next pages for story card & task.

② Small releases

The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

③ Simple design

Design is created only for the requirements to be implemented in current release.

④ Test-first development

An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented

⑤ Refactoring

All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

⑥ Pair programming

Developers work in pairs, checking each other's work and providing the support.

⑦ Collective ownership

The pairs of developers work on all areas of the system, so that no islands of expertise develop and all developers take responsibility for all of the code. Anyone can change anything

⑧ Continuous integration

As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.

⑨ Sustainable pace

Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity

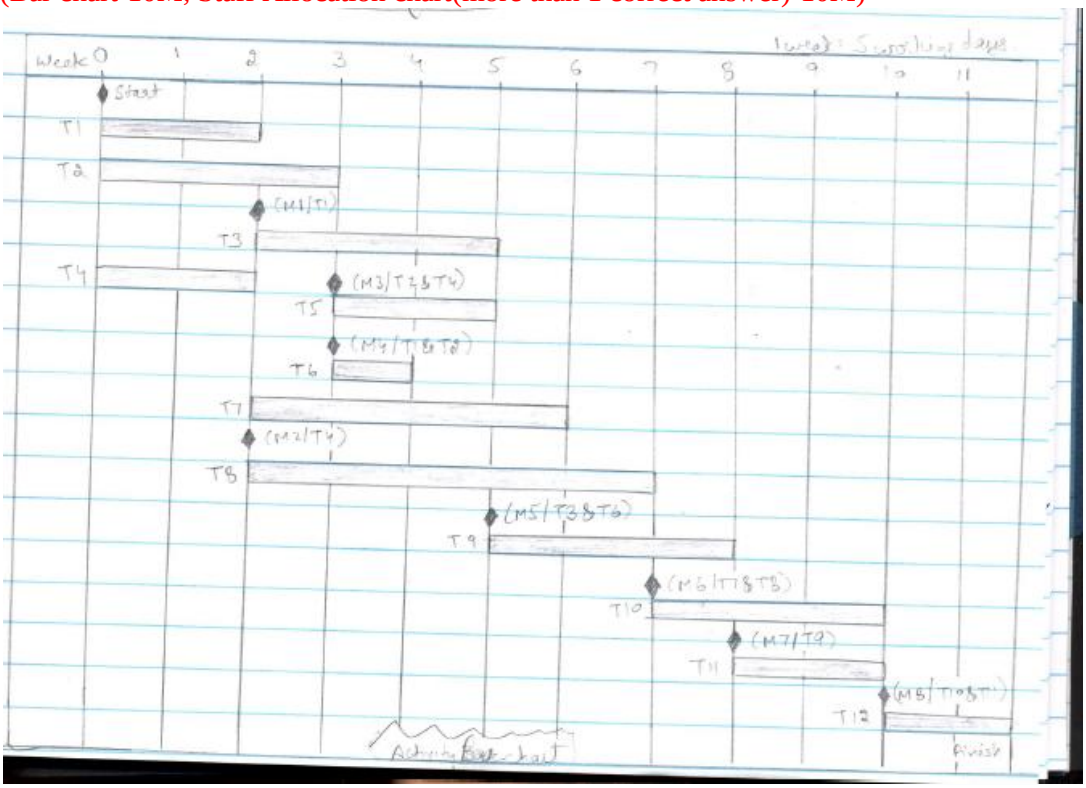
⑩ On-site customer

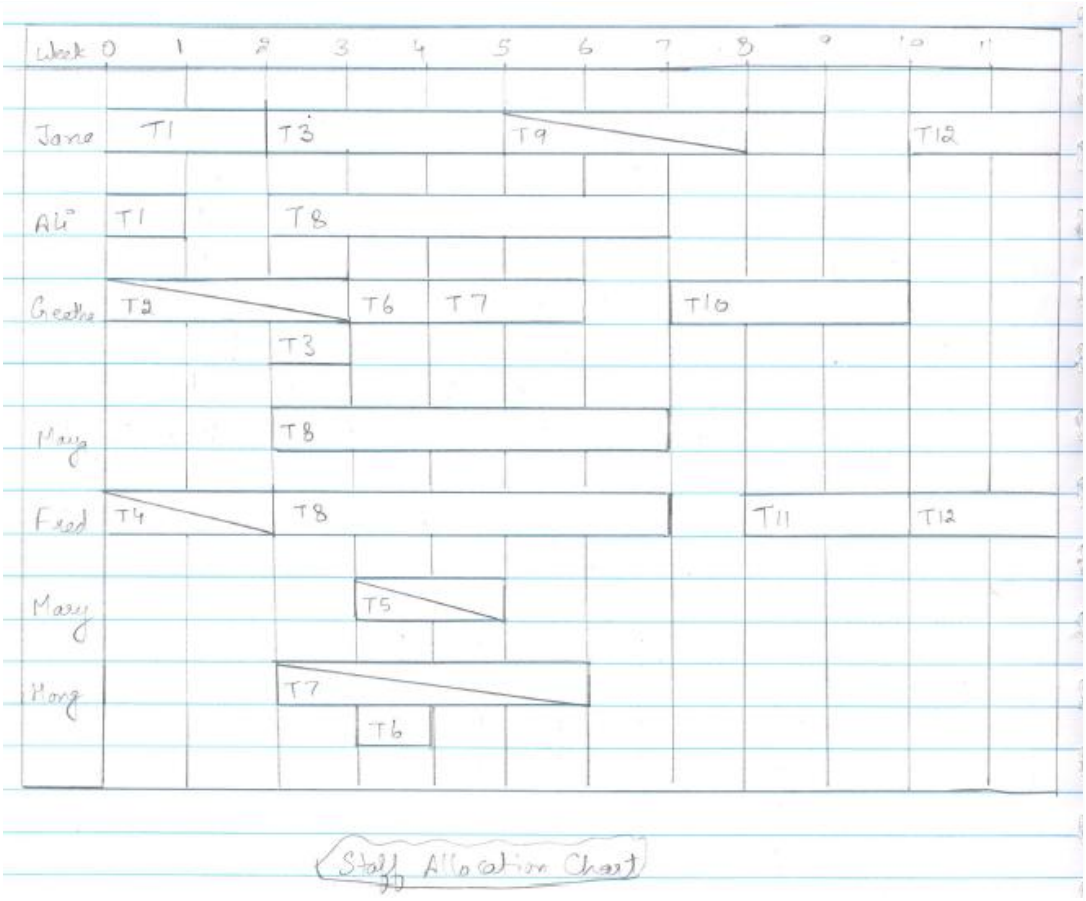
A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. Customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

		MARKS	CO	RBT
	PART D			
7 (a) (b)	From the information given in the table, draw Bar chart/Gantt chart Staff allocation chart	[20]	CO3	L3

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	-
T2	8	15	-
T3	20	15	T1 (M1)
T4	5	10	-
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

(Bar chart-10M, Staff Allocation chart(more than 1 correct answer)-10M)



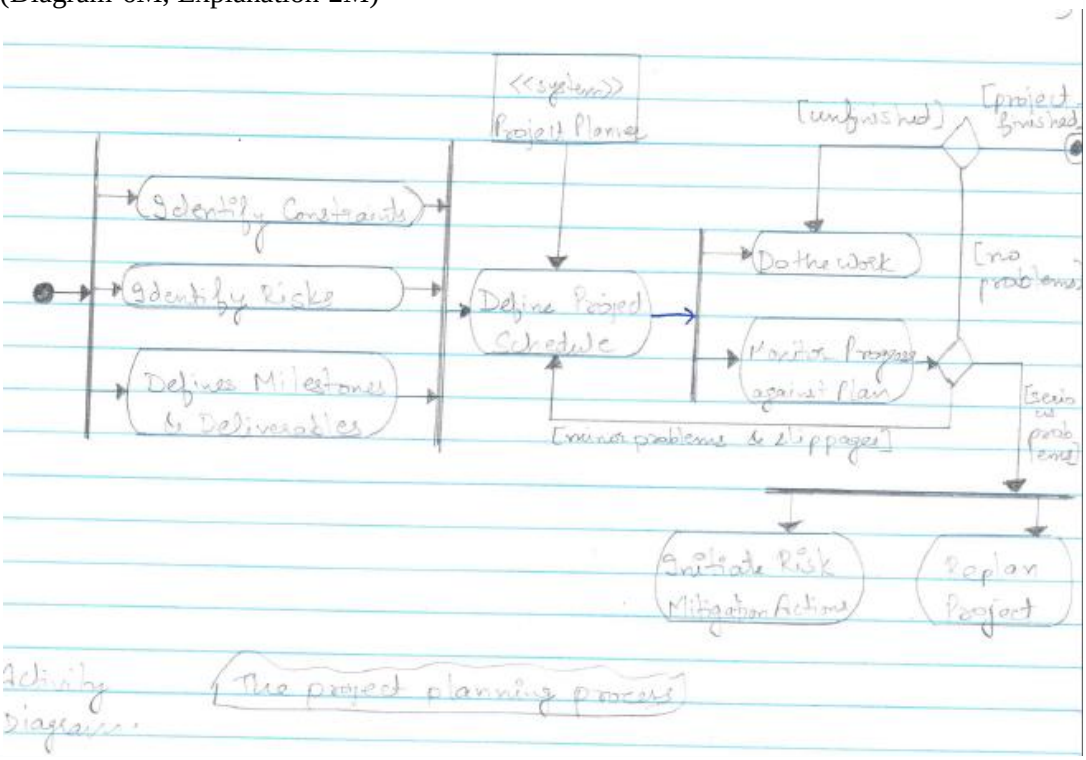


Staff Allocation Chart

OR

8 (a) With a neat diagram, explain planning process. [8]

(Diagram-6M, Explanation-2M)



Activity Diagram

The project planning process

CO3	L2

- Assess constraints (required delivery date, staff available, overall budget, available tools etc), identify risks, milestones and deliverables.
- Draw up estimated schedule and start activities.
- Monitor progress against plan and make modifications to project schedule based on minor problems and slippage.
- If no problems arise, keep doing work & monitoring progress until project is finished.
- If some serious problems arise, initiate the risk mitigation actions and replan the project.

(b) Explain COCOMO-II. [12]

(Diagram-6M, Exaplantion-4x1.5M)

V.IMP

The COCOMO-II Model.

- This is an empirical model that was derived by collecting data from a large number of software projects. The data were analyzed to discover the formulae that were the best fit to the observations.
- It is well-documented and non-proprietary estimation model and takes into account modern approaches to software development.

Number of Application Points	Based on	Application Composition Model	Used for	Systems developed using Dynamic Languages, OO Programming etc.
Number of Function Points	Based on	Early Design Model	Used for	Initial effort estimation based on system requirements & design options
Number of LOC 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

CO4 L2

(i) Application Composition Model

- It is used when software is composed from existing parts. It supports prototyping projects and extensive reuse.
- Effort is estimated from software size which is estimated based on application points. (Application points in a program is a weighted estimate of no. of separate screens displayed, no. of reports produced, no. of modules in imperative programming languages, no. of lines of scripting language or database programming code).

$$PM = \frac{\left(NAP \times \left(1 - \frac{\%reuse}{100}\right)\right)}{PROD}$$

PM = Effort in person-months
NAP = no. of application points
PROD = productivity.

(ii) Early design model

- Used when requirements are available but design has not started.

$$PM = A \times Size^{B} \times M$$

$$= 2.94 \times Size^{(1.1-1.24)} \times M$$

where $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCILXSCED$

- PERS: personnel capability
- RCPX: product reliability & complexity
- RUSE: reuse required
- PDIF: platform difficulty
- PREX: personnel experience
- FCILXSCED: support facilities, schedule

(iii) Reuse Model

- Used to compute the effort of integrating reusable components.
- Black-box reuse - when code is not modified (generated)
  White-box reuse - where code is modified (understood & integrated)

Black-box reuse :

$$PM = \frac{\left(\frac{ASLOC * AT}{100}\right)}{ATPROD}$$

ASLOC - no. of lines of generated code
AT - % of code automatically generated
ATPROD - productivity of engineers in integrating this code

White-box reuse :

$$ESLOC = ASLOC * \left(1 - \frac{AT}{100}\right) * AAM$$

AAM = Adaptation Adjustment Multiplier.

(iv) Post-architectural level

- Used once system architecture has been designed and more information about the system is available.
- Uses same formula as early design model but with 17 rather than 7 associated multipliers

$$PM = A \times Size^{B} \times M$$

Code size is estimated as:

→ No. of lines of new code to be developed.
→ Estimate of equivalent no. of lines of new code computed using reuse model
→ Estimate of no. of lines of code that have to be modified according to requirements changes.