PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)
END SEMESTER EXAMINATION
May 2023
BRANCH: Information Technology

SEM-VI

Subject:- Software Engineering and Project Management	**Time: 02.00 Hours**
Max. Marks: 60 **N.B 1. Q.1 is compulsory** **2. Attempt any two from the remaining three questions** **3. Each Question carry 20 marks.**	**Date: 02/05/2023** **Subject Code IT311**

Q.1.	**Attempt All**	**Marks**
a)	Explain and state different types of Coupling and Cohesion	**5**
b)	What is quality?Explain Mc Call's Quality factor.	**5**
c)	Draw DFD level 0 and level 1 for the Restaurant management system that includes food ordering,food delivering,Invoice creation and Payment subsystem.	**5**
d)	Explain PLC(Project life cycle)	**5**
Q.2.	**Attempt All**	
b)	Draw the activity diagram(Swim lanes) and Statechart diagram for ATM System	**10**
c)	Write a short note on SCM Process and explain how change control and version control is carried out in SCM.	**10**
Q.3.	**Attempt All**	
a)	What is Agile methodology? Explain SCRUM.	**10**
b)	Explain in detail 4P's of project management.	**10**
Q.4.	**Attempt All**	
a)	A Project manager and team came up with estimates as presented in the table given below.Draw an activity on node diagram based on the predecessors given,calculate expected duration for each activity and find the critical path. Table 1.1 Activity analysis for PERT	**5**

		Activity	Predecessor	Optimistic Estimates (days) a	Most likely Estimates (days) b	Pessimistic Estimates (days) c	
		A	None	1	2	4	
		B	A	3	5	8	
		C	B	2	4	5	
		D	B	2	3	6	
		E	B	1	1	1	
		F	C, D	2	4	6	
		G	D, E	2	3	4	
		H	F, G	1	2	5	
		I	G	4	5	9	
		J	H, I	0.5	1	3	
b)	What is software engineering? Explain RAD model with diagram						**5**
c)	Differentiate between Verification and validation						**5**
d)	Using the COCOMO II model to estimate the effort required to build a software that produces 10 screens and 8 reports with 70 3GL components. Assume that software has average complexity for screen-2,Reports-5 and difficult for 3GL components-10. Also average environmental maturity.This system has 50% reuse.						**5**

SOLUTION:

Q.1.	Attempt All	Marks
a)	Explain and state different types of Coupling and Cohesion	5

Ans:

Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

b)	What is quality?Explain Mc Call's Quality factor.	5

ANS:

McCall software quality model was introduced in 1977. This model is incorporated with many attributes, termed as software factors, which influence a software. The model distinguishes between two levels of quality attributes :

1. **Quality Factors** – The higher level quality attributes which can be accessed directly are called quality factors. These attributes are external attributes.The attributes in this level are given more importance by the users and managers.
2. **Quality Criteria** – The lower or second level quality attributes which can be accessed either subjectively or objectively are called Quality Criteria. These attributes are internal attributes. Each quality factor has many second level of quality attributes or quality criteria.
3. The following are the product quality factors –

1. Product Operation :

It includes five software quality factors, which are related with the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage and its correctness. These factors help in providing a better user experience.

a. **Correctness** – The extent to which a software meets its requirements specification.
b. **Efficiency** – The amount of hardware resources and code the software needs to perform a function.
c. **Integrity** – The extent to which the software can control an unauthorized person from accessing the data or software.
d. **Reliability** – The extent to which a software performs its intended functions without failure.
e. **Usability** –

The extent of effort required to learn, operate and understand the functions of the software.

4.

2. Product Revision :

It includes three software quality factors, which are required for testing and maintenance of the software. They provide ease of maintenance, flexibility and testing effort to support the software to be functional according to the needs and requirements of the user in the future.

a. **Maintainability** –
The effort required to detect and correct an error during the maintenance phase.

b. **Flexibility** –
The effort needed to improve an operational software program.

c. **Testability** –
The effort required to verify a software to ensure that it meets the specified requirements.

5.

3. Product Transition :

It includes three software quality factors, that allows the software to adapt to the change of environments in the new platform or technology from the previous.

a. **Portability** –
The effort required to transfer a program from one platform to another.

b. **Re-usability** –
The extent to which the program's code can be reused in other applications.

c. **Interoperability** –
The effort required to integrate two systems with one another.

c)	Draw DFD level 0 and level 1 for the Restaurant management system that includes food ordering,food delivering,Invoice creation and Payment subsystem.

Ans:

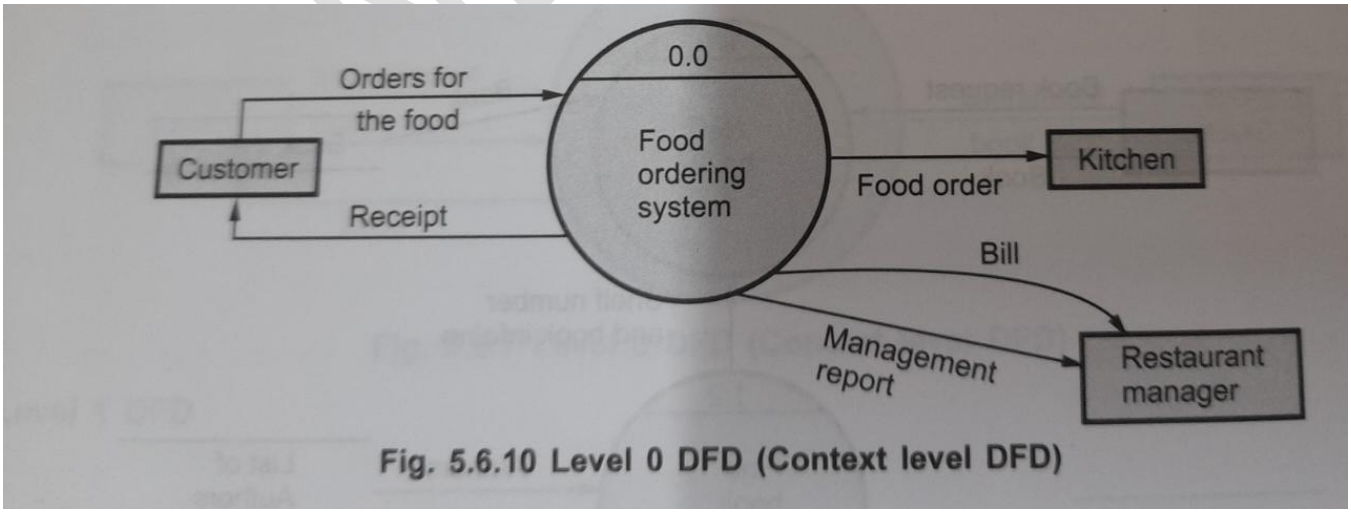


Fig. 5.6.10 Level 0 DFD (Context level DFD)

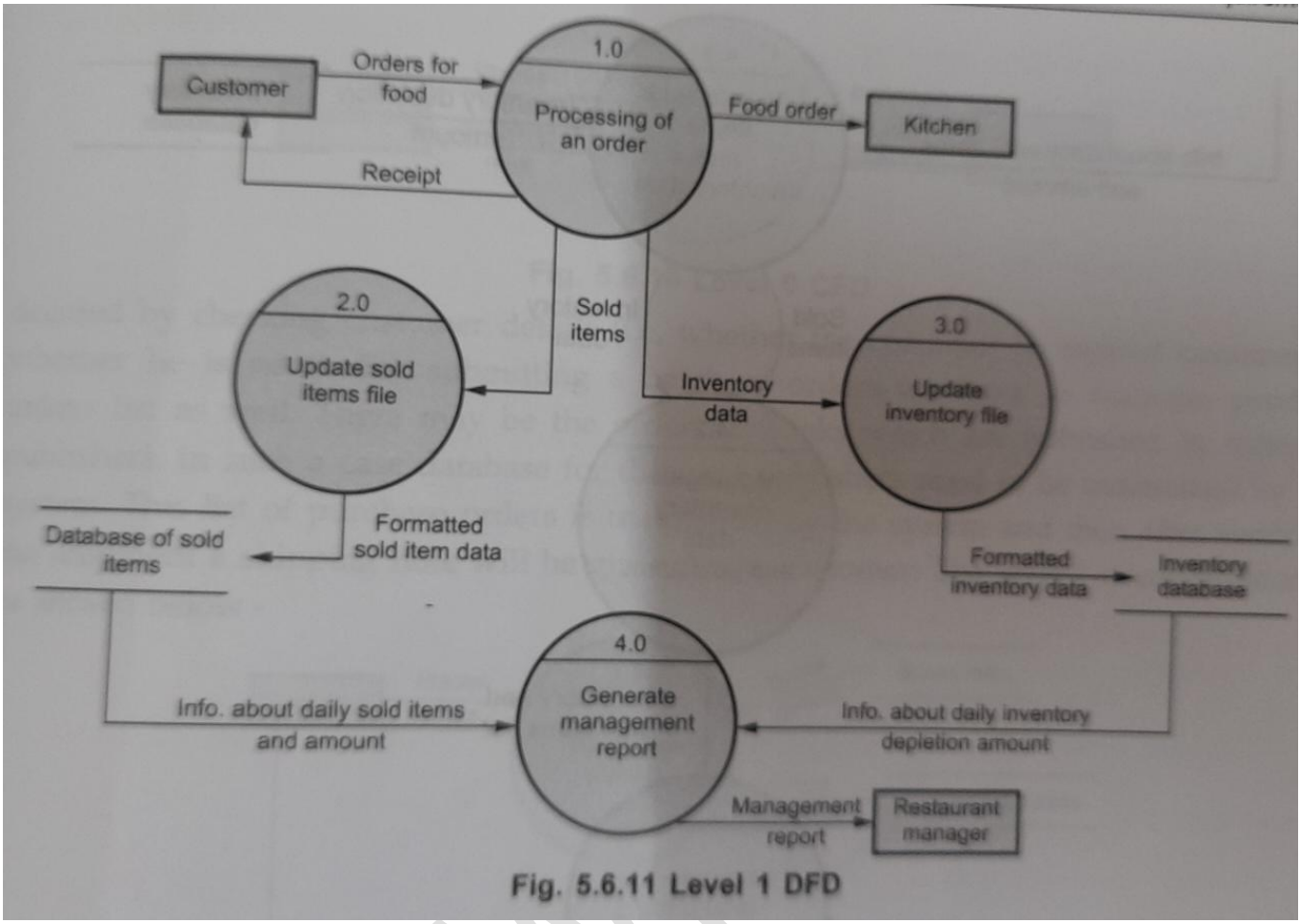
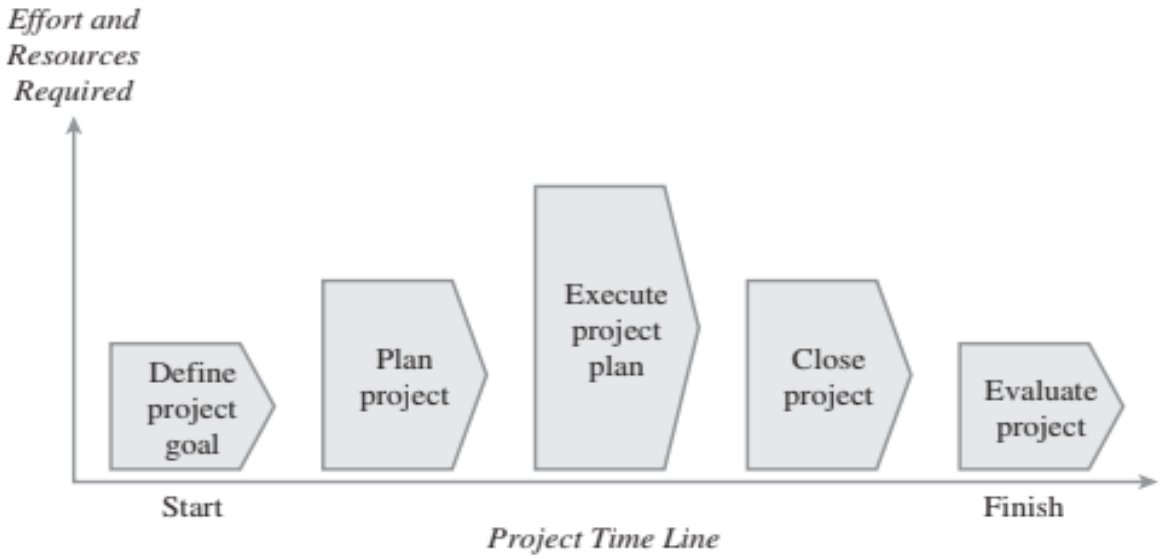


Fig. 5.6.11 Level 1 DFD

d)	Explain PLC(Project life cycle)	**5**

Ans:



Define Project Goal

Defining the project's overall goal should be the first step. This goal should focus on providing business value to the organization. A well defined goal gives the project team a clear focus and drives the other phases of the project. The project goal should also answer the question: How will we know if this project is successful given the time, money, and resources invested?

Plan Project

Once the project's goal has been defined, developing the project plan is a much easier task. A plan essentially answers the following questions:

- What are we going to do?
- What are we not going to do?
- Why are we going to do it?
- How are we going to do it?
- Who is going to be involved?
- How long will it take?
- How much will it cost?
- What can go wrong and what can we do about it?
- How will we know if we are successful?

In addition, the deliverables, tasks, resources, and time to complete each task must be defined for each phase of the project. The project plan defines the agreed upon scope, schedule, and budget and is used as a tool to gauge the project's performance throughout the life cycle.

Execute Project Plan

After the project's goal and plan have been defined, it's time to put the plan into action. As work on the project progresses, scope, schedule, budget, and people must be actively managed to ensure that the project achieves its goal. Progress must be documented and compared to the

THE PROJECT LIFE CYCLE AND IT DEVELOPMENT 33

baseline plan. In addition, project performance must be communicated to all of the stakeholders. At the end of this phase, the team implements or delivers a completed product, service, or information system to the organization.

Close Project

A project should have a definite beginning and end. The closing phase ensures that all of the work is completed as planned and as agreed to by the team and the sponsor. Therefore, there should be some kind of formal acknowledgment by the sponsor that they will accept the product delivered. This closure is often capped with a final project report and presentation to the client that documents that all promised deliverables have been completed as specified.

Evaluate Project

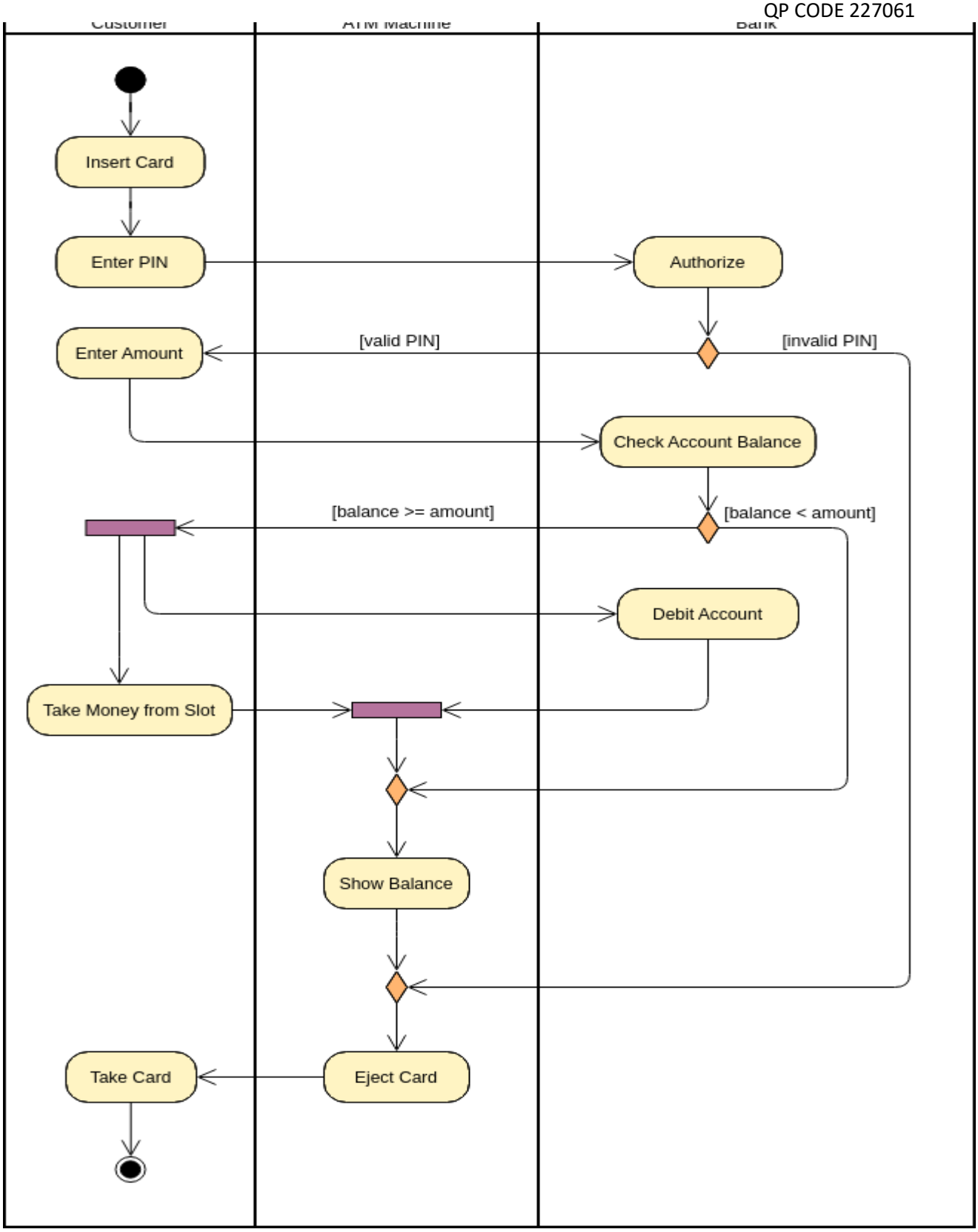
Sometimes the value of an IT project is not readily known when the product, service, or information system is implemented. For example, the goal of a project to develop an electronic

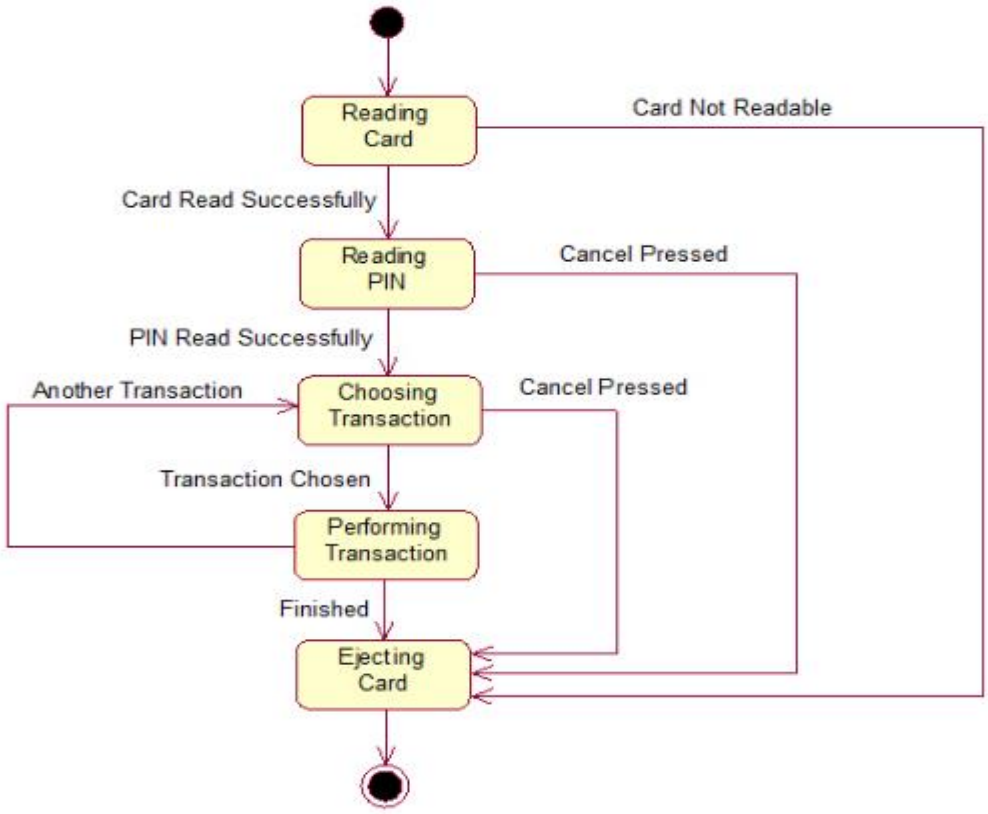
commerce site should be to make money—not to build or install hardware, software, and Web pages on a particular server platform. The technology and its subsequent implementation are only a means to an end. Therefore, the goal of the electronic commerce site may be to produce $250,000 in revenue within six months. As a result, evaluating whether the project met its goal can be made only after the system has been implemented.

However, the project can be evaluated in other ways as well. The project team should document its experiences in terms of lessons learned—those things that it would do the same and those things that it would do differently on the next project, based on its current project experiences. This post mortem should be documented, stored electronically, and shared throughout the organization. Subsequently, many of these experiences can be translated into best practices and integrated into future projects.

In addition, both the project team and the project itself should be evaluated at the end of the project. The project manager may evaluate each team member's performance in order to provide feedback and as part of the organization's established merit and pay raise processes and procedures. Often, however, an outside third party, such as a senior manager or partner, may audit the project to determine whether the project was well managed, provided the promised deliverables, followed established processes, and met specific quality standards. The team and project manager may also be evaluated in terms of whether they acted in a professional and ethical manner.

Q.2.	**Attempt All**	
a)	Draw the activity diagram(Swim lanes) and Statechart diagram for ATM System	**10**


Customer
ATM Machine
Bank
Insert Card
Enter PIN
Authorize
Enter Amount
[valid PIN]
[invalid PIN]
Check Account Balance
[balance >= amount]
[balance < amount]
Debit Account
Take Money from Slot
Show Balance
Take Card
Eject Card



b)	Write a short note on SCM Process and explain how change control and version control is carried out in SCM.	**10**

ANS:

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection.

SCM is also known as software control management. SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.

Version Control:-

Software Version Control is a system or tool that captures the changes to a source code elements: files, folders, images or binaries.

A version control system (also known as a Revision Control System) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.

Version control systems are essential for any form of distributed, collaborative development. Whether it is the history of a wiki page or large software development project, the ability to track each change as it was made, and to reverse changes when necessary can make all the difference between a well managed and controlled process and

an uncontrolled ‘first come, first served’ system. It can also serve as a mechanism for due diligence for software projects.

• Combines procedures and tools to manage the different versions of configuration objects created during the software process.

• Version control systems require the following capabilities.

o Project repository – stores all relevant configuration objects.

o Version management capability – stores all versions of a configuration object (enables any version to be built from past versions)

o Make facility – enables collection of all relevant configuration objects and construct a specific software version.

o Issues (bug) tracking capability – enables team to record and track status of outstanding issues for each configuration object.

• Uses a system modeling approach (template – includes component hierarchy and component build order, construction rules, verification rules).

Change Control:-

Change control is a systematic approach to managing all changes made to a product or system. The purpose is to ensure that no unnecessary changes are made, that all changes are documented, that services are not unnecessarily disrupted and that resources are used efficiently.

Here's an example of a six-step process for a software change request:

1. Documenting the change request:

When the client requests the change, that request is categorized and recorded, along with informal assessments of the importance of that change and the difficulty of implementing it.

1. Formal assessment:

The justification for the change and risks and benefits of making/not making the change are evaluated. If the change request is accepted, a development team will be assigned. If the change request is rejected, that fact is documented and communicated to the client.

1. Planning:

The team responsible for the change creates a detailed plan for its design and implementation, as well as a plan for rolling back the change should it be deemed unsuccessful.

1. Designing and testing:

The team designs the program for the software change and tests it. If the change is deemed successful, the team

requests approval and a date for implementation.

1. Implementation and review:

The team implements the program and stakeholders review the change.

1. Final assessment:

If the client is satisfied that the change was implemented satisfactorily, the change request is closed. If the client is not satisfied, the project is reassessed and steps may be repeated.

Q.3.	**Attempt All**	
a)	What is Agile methodology? Explain SCRUM.	**10**

ANS:

What is Agile Scrum Methodology?

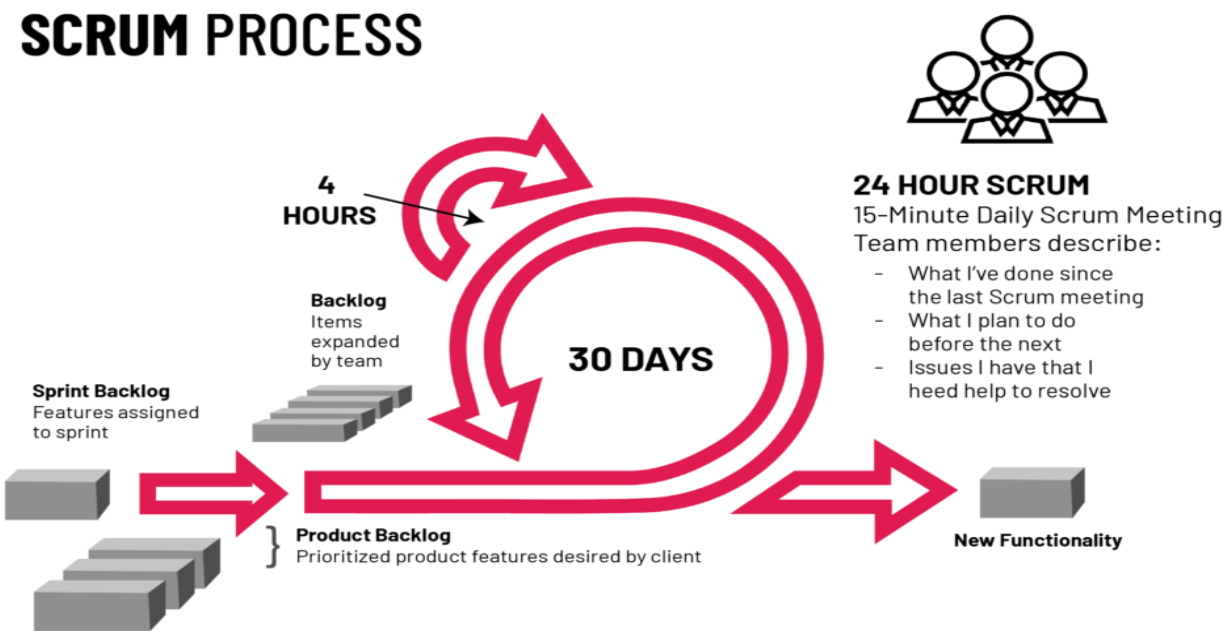
Agile scrum methodology is a project management system that relies on incremental development. Each iteration consists of two- to four-week sprints, where the goal of each sprint is to build the most important features first and come out with a Potentially Shippable Product.

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.

- A “process framework” is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.)
- “Lightweight” means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes. These and other terms used in Scrum are defined below. Scrum is most often used to manage complex software and product development, using iterative and incremental practices. Scrum significantly increases productivity and reduces time to benefits relative to classic “waterfall” processes. Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to

- Increase the quality of the deliverables
- Cope better with change (and expect the changes)
- Provide better estimates while spending less time creating them
- Be more in control of the project schedule and state



What are the Scrum requirements?

Scrum does not define just what form requirements are to take, but simply says that they are gathered into the Product Backlog, and referred to generically as "Product Backlog Items," or "PBIs" for short. Given the time-boxed nature of a Sprint, we can also infer that each set should require significantly less time to implement than the duration of the Sprint. Most Scrum projects borrow the "XP" (Extreme Programming) practice of describing a feature request as a "User Story," although a minority uses the older concept of a "Use Case." We will go with the majority view here, and describe three reasonably-standard requirements artifacts found in Product Backlogs.

User Story

A User Story describes a desired feature (functional requirement) in narrative form. User Stories are usually written by the Product Owner, and are the Product Owner's responsibility. The format is not standardized, but typically has a name, some descriptive text, references to external documents (such as screen shots), and information about how the implementation will be tested. For example, a Story might resemble the following:

> Name: Planner enters new contact into address book, so that one can contact the person later by postal or electronic mail
>
> Description: Planner enters standard contact information (first and last name, two street address lines, city, state, zip / postal code, country, etc.) into contact-entry screen. One clicks "Save" to keep the data, and "Cancel" to discard data and return to the previous screen.
>
> How to test: Tester enters and saves the data, finds the name in the address book, and clicks on it. One sees a read-only view of the contact-entry screen, with all data previously entered.

The elements in this User Story are:

1. Name: The Name is a descriptive phrase or sentence. The example uses a basic "Role-Action-Reason"

organization. Another common style, popularized by Mike Cohn, follows the template "As a <type of user>, I want <some goal> so that <some reason>." The choice of template is less important than having a workable standard of some kind.

2. Description: This is a high-level (low-detail) description of the need to be met. For functional (user-facing) requirements, the description is put in narrative form. For non-functional requirements, the description can be worded in any form that is easy to understand. In both cases, the key is that the level of detail is modest, because the fine details are worked out during the implementation phase, in discussions between team members, product owners, and anyone else who is involved. (This is one of the core concepts of Scrum: Requirements are specified at a level that allows rough estimation of the work required to implement them, not in detail.)
3. Screens and External Documents: If the Story requires user-interface changes (especially non-trivial ones), the Story should contain or link to a prototype of the changes. Any external documents required to implement the Story should also be listed.
4. How to test: The implementation of a Story is defined to be complete if, and only if, it passes all acceptance tests developed for it. This section provides a brief description of how the story will be tested. As for the feature itself, the description of testing methods is short, with the details to be worked out during implementation, but we need at least a summary to guide the estimation process.

There are two reasons for including the information about how to test the Story. The obvious reason is to guide development of test cases (acceptance tests) for the Story. The less-obvious, but important, reason, is that the Team will need this information in order to estimate how much work is required to implement the story (since test design and execution is part of the total work).

What are the Scrum roles?

The three roles defined in Scrum are the ScrumMaster, the Product Owner, and the Team (which consists of Team members). The people who fulfill these roles work together closely, on a daily basis, to ensure the smooth flow of information and the quick resolution of issues.

ScrumMaster

The ScrumMaster (sometimes written "Scrum Master," although the official term has no space after "Scrum") is the keeper of the process. The ScrumMaster is responsible for making the process run smoothly, for removing obstacles that impact productivity, and for organizing and facilitating the critical meetings. The ScrumMasters responsibilities include

- Teach the Product Owner how to maximize return on investment (ROI), and meet his/her objectives through

Scrum.

- Improve the lives of the development Team by facilitating creativity and empowerment.
- Improve the productivity of the development Team in any way possible.
- Improve the engineering practices and tools so that each increment of functionality is potentially shippable.
- Keep information about the Team's progress up to date and visible to all parties.

In practical terms, the ScrumMaster needs to understand Scrum well enough to train and mentor the other roles, and educate and assist other stakeholders who are involved in the process. The ScrumMaster should maintain a constant awareness of the status of the project (its progress to date) relative to the expected progress, investigate and facilitate resolution of any roadblocks that hold back progress, and generally be flexible enough to identify and deal with any issues that arise, in any way that is required. The ScrumMaster must protect the Team from disturbance from other people by acting as the interface between the two. The ScrumMaster does not assign tasks to Team members, as task assignment is a Team responsibility. The ScrumMaster's general approach towards the Team is to encourage and facilitate their decision-making and problem-solving capabilities, so that they can work with increasing efficiency and decreasing need for supervision. The goal is to have a team that is not only empowered to make important decisions, but does so well and routinely.

b)	Explain in detail 4P's of project management.	**10**

ANS:

People

The Stakeholders

The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies:

1. Senior managers who define the business issues that often have a significant influence on the project.
2. Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
3. Practitioners who deliver the technical skills that are necessary to engineer a product or application.
4. Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. End users who interact with the software once it is released for production use.

Team Leaders

Project management is a people-intensive activity, and for this reason, competent

practitioners often make poor team leaders.

- Motivation. The ability to encourage (by "push or pull") technical people to produce to their best ability.
- Organization. The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- Ideas or innovation. The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

The Software Team:The "best" team structure depends on the management style of your organization,the number of people who will populate the team and their skill levels, and the overall problem difficulty.

• Difficulty of the problem to be solved

• "Size" of the resultant program(s) in lines of code or function points

• Time that the team will stay together (team lifetime)

• Degree to which the problem can be modularized

• Required quality and reliability of the system to be built

• Rigidity of the delivery date

• Degree of sociability (communication) required for the project

The Product:

Software Scope

The first software project management activity is the determination of software

scope. Scope is defined by answering the following questions:

- Context. How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?
- Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input?
- Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Problem Decomposition

Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis.During the scoping activity no attempt is made to fully decompose the problem. Rather, decomposition is applied in two major areas: (1) the functionality and content (information) that must be delivered and (2) the process that will be used to deliver it.

The Process:

1. Review the customer request.

2. Plan and schedule a formal, facilitated meeting with all stakeholders.

3. Conduct research to specify the proposed solution and existing approaches.

4. Prepare a "working document" and an agenda for the formal meeting.

5. Conduct the meeting.

6. Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.

7. Review each mini-spec or use case for correctness, consistency, and lack of ambiguity.

8. Assemble the mini-specs into a scoping document.

9. Review the scoping document or collection of use cases with all concerned.

10. Modify the scoping document or use cases as required.

The Project:

In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided. In an excellent paper on software projects, defines 10 signs that indicate that an information systems project is in jeopardy:

1. Software people don't understand their customer's needs.

2. The product scope is poorly defined.

3. Changes are managed poorly.

4. The chosen technology changes.

5. Business needs change [or are ill defined].

6. Deadlines are unrealistic.

7. Users are resistant.

8. Sponsorship is lost [or was never properly obtained].

9. The project team lacks people with appropriate skills.

10. Managers [and practitioners] avoid best practices and lessons learned.

Q.4.	**Attempt All**	
a)	A Project manager and team came up with estimates as presented in the table given below.Draw an activity on node diagram based on the predecessors given,calculate expected duration for each activity and find the critical path. Table 1.1 Activity analysis for PERT	5

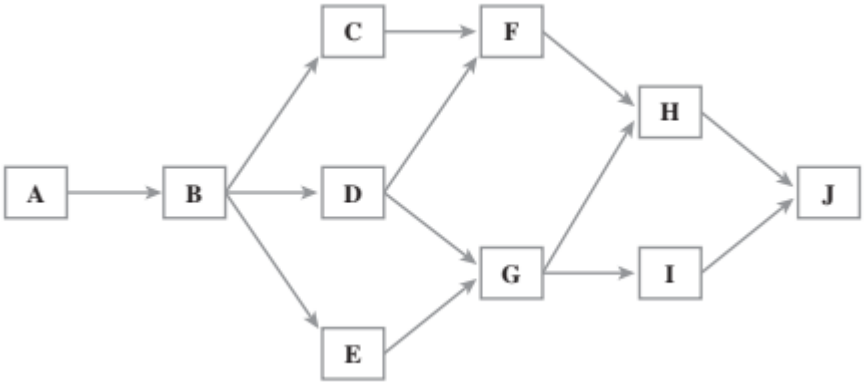
Activity	Predecessor	Optimistic	Most likely	Pessimistic

		Estimates (days) a	Estimates (days) b	Estimates (days) c
A	None	1	2	4
B	A	3	5	8
C	B	2	4	5
D	B	2	3	6
E	B	1	1	1
F	C, D	2	4	6
G	D, E	2	3	4
H	F, G	1	2	5
I	G	4	5	9
J	H, I	0.5	1	3

ANS:

PERT The program evaluation and review technique (PERT) was developed in the late 1950s to help manage the Polaris submarine project. At about the same time, the critical path method (CPM) was developed. The two methods are often combined and called PERT/CPM. PERT uses the project network diagramming technique to create a visual representation of the scheduled activities that expresses both their logical sequence and interrelationships. PERT also uses a statistical distribution that provides probability for estimating when the project and its associated activities will be completed. This probabilistic estimate is derived by using three estimates for each activity: optimistic, most likely, and pessimistic. An optimistic estimate is the minimum time in which an activity or task can be completed. This is a best-case scenario where everything goes well and there is little or no chance of finishing earlier. A most likely estimate, as the name implies, is the normally expected time required to complete the task or activity. A pessimistic estimate is a worst-case scenario and is viewed as the maximum time in which an activity can or should be completed.One can use the following equation to compute a mean or weighted average for each individual activity that will become the PERT estimate:

Activity Estimate = Optimistic Time + (4 × Most Likely Time) + Pessimistic Time/6



The total expected time to complete the project can be easily found by summing each of individual activity estimates or:

$$\text{Total Expected Time of Project} = \sum_{i=1}^{n} \text{Activity Estimates}$$

For example, on our project used earlier, a project manager and team came up with the estimates presented in Table

Activity	*Predecessor*	*Optimistic Estimates (Days) a*	*Most Likely Estimates (Days) b*	*Pessimistic Estimates (Days) c*	*Expected Duration (a + 4b + c)/6*
A	None	1	2	4	2.2
B	A	3	5	8	5.2
C	B	2	4	5	3.8
D	B	2	3	6	3.3
E	B	1	1	1	1.0
F	C, D	2	4	6	4.0
G	D, E	2	3	4	3.0
H	F, G	1	2	5	2.3
I	G	4	5	9	5.5
J	H, I	.5	1	3	1.3

Possible Paths	*Path*	*Total*
Path 1	A + B + C + F + H + J	18.8
	2.2 + 5.2 + 3.8 + 4.0 + 2.3 + 1.3	
Path 2	A + B + D + F + H + J	18.3
	2.2 + 5.2 + 3.3 + 4.0 + 2.3 + 1.3	
Path 3	A + B + D + G + H + J	18.6
	2.2 + 5.2 + 3.3 + 4.0 + 2.3 + 1.3	
Path 4	A + B + D + G + I + J	20.5*
	2.2 + 5.2 + 3.3 + 3.0 + 5.5 + 1.3	
Path 5	A + B + E + G + I + J	18.2
	2.2 + 5.2 + 1.0 + 3.0 + 5.5 + 1.3	

*Critical Path

Analyzing the various paths using PERT provides the critical paths presented in Table .As can be seen in Table, the critical path is still Path 4 and the expected completion date of the project is 20.5, or 21 days if we round up.

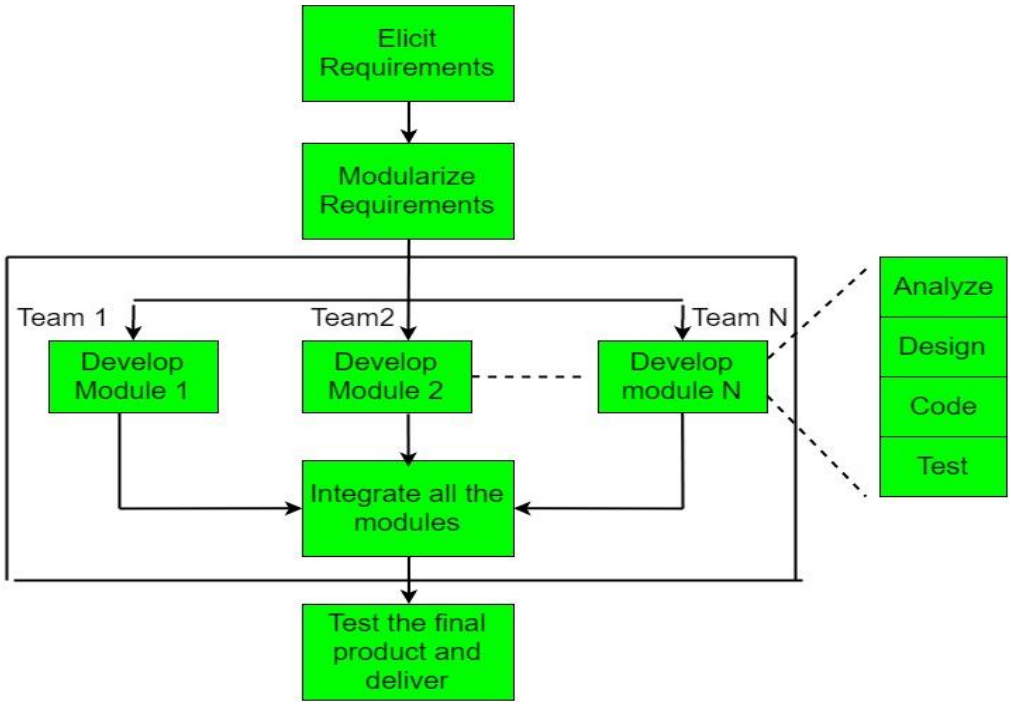
In this case, the deadline increased from 19 days using the AON method to 21 days using the statistical technique associated with PERT.In the first case, the most likely estimates were used, while PERT took into account not only the most likely estimates, but optimistic and pessimistic estimates as well. PERT is well suited for developing simulations whereby the project manager can conduct a sensitivity analysis for schedule planning and risk analysis. But, like any planning and scheduling tool, its usefulness is highly correlated to the quality of the estimates used.

b)	What is software engineering? Explain RAD model with diagram	**5**

ANS:

RAD Model or Rapid Application Development model is a software development process based on prototyping without any specific planning. In RAD model, there is less attention paid to the planning and more priority is given to the development tasks. It targets at developing software in a short span of time.

The Rapid Application Development Model was first proposed by IBM in the 1980s. The critical feature of this model is the use of powerful development tools and techniques. A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product. Development of each module involves the various basic steps as in the waterfall model i.e analyzing, designing, coding, and then testing, etc. as shown in the figure. Another striking feature of this model is a short time span i.e the time frame for delivery(time-box) is generally 60-90 days.



The use of powerful developer tools such as JAVA, C++, Visual BASIC, XML, etc. is also an integral part of the projects. This model consists of 4 basic phases:

1. Requirements Planning – It involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.
2. User Description – This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.
3. Construction – In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform process and data models into the final working product. All the required modifications and enhancements are too done in this phase.
4. Cutover – All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

The process involves building a rapid prototype, delivering it to the customer, and taking feedback. After validation by the customer, the SRS document is developed and the design is finalized.

Advantages:

- The use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at the initial stages.
- Reduced costs as fewer developers are required.
- The use of powerful development tools results in better quality products in comparatively shorter time spans.
- The progress and development of the project can be measured through the various stages.
- It is easier to accommodate changing requirements due to the short iteration time spans.

Disadvantages:

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to the failure of the project.
- The team leader must work closely with the developers and customers to close the project in time.
- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small-scale projects as in such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.

c)	Differentiate between Verification and validation	5

ANS:

Verification	Validation
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

d)	Using the COCOMO II model to estimate the effort required to build a software that produces 10 screens and 8 reports with 70 3GL components. Assume that software has average complexity for screen-2,Reports-5 and difficult for 3GL components-10. Also average environmental maturity.This system has 50% reuse.	5

ANS:

Step-1: Access Object counts

Estimate the number of screens, reports and 3GL components that will comprise this application. There are 10 screens, 8 reports and 70 3GL components.

Step-2: Classify complexity levels of each object

We have to classify each object instance into simple, medium and difficult complexity level depending on values of its characteristics.

Average complexity for screen and reports whereas difficult for 3GL components.

Step-3: Assign complexity weights to each object

The weights are used for three object types i.e, screens, reports and 3GL components. Complexity weight are assigned according to object's complexity level using following table

Object Type	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Components	-	-	10

Complexity Weight

Step-4: Determine Object Points

SCREEN= 2*10=20

REPORT=8*5=40

3GL COMPONENTS=70*10=700

AFTER ADDING TO GET THE TOTAL COUNT=20+40+700=760

Step-5: Compute New Object Points (NOP)

We have to estimate the %reuse to be achieved in a project.

Depending on %reuse

```
NOP = [(object points) * (100 - %reuse)]/100

   = [(760)*(100-50)]/100 =380
```

Step-6: Calculate Productivity rate (PROD)

Productivity rate **is calculated on the basis of information given about developer's experience and capability.**

For calculating it, we use following table

Developers experience & capability	Productivity (PROD)
Very Low	4
Low	7
Nominal	13
High	25
High	50

Productivity Rate

Step-7: Compute the estimated Effort

Effort **to develop a project can be calculated as**

```
Effort = NOP/PROD
       =380/13=29.23 pm
```