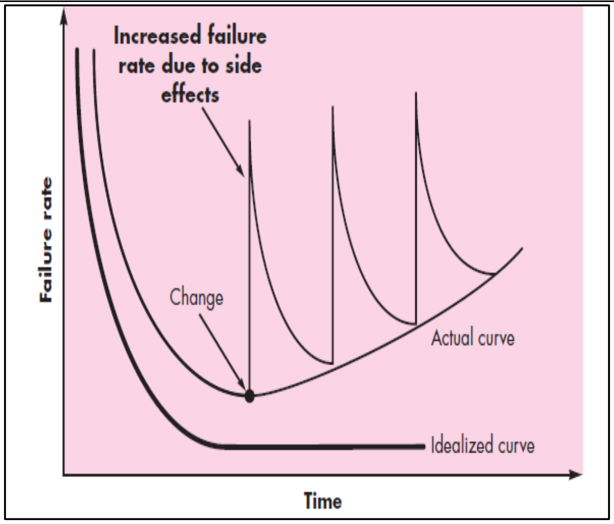
SUMMER – 19 EXAMINATION

Subject Name: Software Engineering **Model Answer** **Subject Code: 22413**

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1		**Attempt any Five of the following:**	**10 M**
	a	**Enlist and explain software characteristics (any two).**	**2 M**
	Ans	1. Software is developed or engineered; it is not manufactured in the classical sense. ▪ Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. ▪ In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are non-existent (or easily corrected) for software. ▪ Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different. ▪ Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects 2. Software doesn't "wear out."	Each Characteristics with explanation – 1M



- The idealized curve as shown in above figure is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does deteriorate!
- This contradiction can best be explained by considering the “actual curve” shown in Figure.
- During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Figure.
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

3. Although the industry is moving toward component-based construction, most software continues to be custom built.

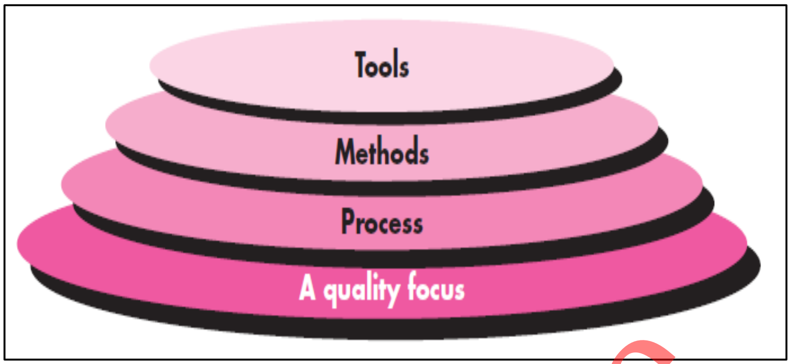
- The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new.
- In the software world, it is something that has only begun to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs
- A software component should be designed and implemented so that it can be reused in many different programs. Modern

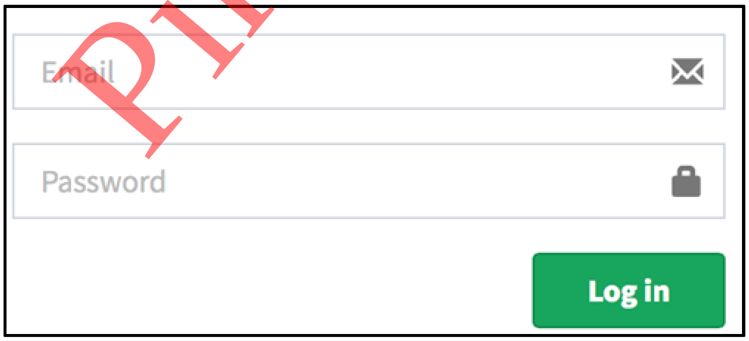
		reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts. ▪ For example, today's interactive user interfaces are built with reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.	
	b	**Define software on engineering.**	**2 M**
	Ans	Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.	Correct Definition-2M
	c	**State need of software requirement specification (SRS).**	**2 M**
	Ans	The need of SRS document is to provide • A detailed overview of software product, its parameters and goals. • The description regarding the project's target audience and its user interface hardware and software requirements. • How client, team and audience see the product and its functionality.	Any two points stating need of SRS-2M
	d	**Define Reactive Risk strategies.**	**2 M**
	Ans	A reactive risk strategy monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems. More commonly, the software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a fire-fighting mode. When this fails, "crisis management" takes over and the project is in real jeopardy.	Correct Definition-2M
	e	**Specify following cost directives of cocomo:** • **Product attributes (any two)** • **Hardware attributes (any two).**	**2 M**
	Ans	**Product attributes –** • Required software reliability extent • Size of the application database • The complexity of the product **Hardware attributes –**	Product attributes (any two)-1M, Hardware

		• Run-time performance constraints • Memory constraints • The volatility of the virtual machine environment • Required turnabout time		attributes (any two)-1M
	f	**Differentiate between Software Quality Management and Software Quality Assurance (any two points).**		**2 M**
	Ans	**Software Quality Assurance (QA)**	**Software Quality Control (QC)**	Each correct differentiation points- 1M
		• It is a procedure that focuses on providing assurance that quality requested will be achieved	• It is a procedure that focuses on fulfilling the quality requested.	
		• QA aims to prevent the defect	• QC aims to identify and fix defects	
		• It is a method to manage the quality- Verification	• It is a method to verify the quality-Validation	
		• It does not involve executing the program	• It always involves executing a program	
		• It's a Preventive technique	• It's a Corrective technique	
		• It's a Proactive measure	• It's a Reactive measure	
		• It is the procedure to create the deliverables	• It is the procedure to verify that deliverables	
		• QA involves in full software development life cycle	• QC involves in full software testing life cycle	
		• In order to meet the customer requirements,	• QC confirms that the standards are followed	

QA defines standards and methodologies	while working on the product
• It is performed before Quality Control	• It is performed only after QA activity is done
• It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	• It is a High-Level Activity, it can identify an error that QA cannot
• Its main motive is to prevent defects in the system. It is a less time-consuming activity	• Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
• QA ensures that everything is executed in the right way, and that is why it falls under verification activity	• QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity
• It requires the involvement of the whole team	• It requires the involvement of the Testing team
• The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)	• The statistical technique applied to QC is known as SQC or Statistical Quality Control

	g	**Define Software Quality Assurance.**	**2 M**
	Ans	• Quality assurance consists of the auditing and reporting functions of management. • The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.	Correct Definition-2M

2.		**Attempt any THREE of the following:**	**12M**
	a	**Explain Software Engineering as layered technology approach.**	**4 M**
	Ans	Software engineering is a layered technology. The layers of software engineering as shown in the above diagram are:-  1. **A Quality Focus:** Any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management, six sigma and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering. The bedrock that supports software engineering is a quality focus. 2. **Process Layer:** The foundation for software engineering is the process layer. Software Engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology. The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, works products (models, documents, data, reports, forms etc.) are produced, milestones are established, quantity is ensured and change is properly managed. 3.**Methods:**	Correct Diagram -1M, explanation - 3M

		Software Engineering methods provide the technical —how to building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing and support. 4.**Tools:** Software Engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer–aided software engineering is established.	
	b	**Explain with example Decision table**	**4 M**
	Ans	• Decision table is a software testing technique used to test system behaviour for different input combinations. • This is a systematic approach where the different input combinations and their corresponding system behaviour (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage. • **Example 1: Decision Base Table for Login Screen**  • The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.	Explanation-2 M, Example of Decision table- 2 M

Decision Table

Conditions	Rule 1	Rule2	Rule3	Rule 4
Username(T/F)	F	T	F	T
Password(T/F)	F	F	T	T
Output(E/H)	E	E	E	H

- **Legend:**

  T – Correct username/password

  F – Wrong username/password

  E – Error message is displayed

  H – Home screen is displayed

- **Interpretation:**
  - Case 1 – Username and password both were wrong. The user is shown an error message.
  - Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
  - Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
  - Case 4 – Username and password both were correct, and the user navigated to homepage.

c **Explain following elements of management spectrum:** **4 M**

i. **People**
ii. **Process**
iii. **Product**
iv. **Project**

Ans **The management Spectrum: 4p's**

Effective software project management focuses on the four P's: people, product, process, and project.

The People:

Explanation each element of management spectrum – 1M

1. The “people factor” is so important that the Software Engineering Institute has developed a People Capability Maturity Model (People-CMM) to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives.
2. The people capability maturity model defines the following key practice areas for software people:

a. Staffing
b. communication and coordination
c. work environment
d. performance management
e. Training, compensation, competency analysis and development, career development, workgroup development, team/culture development and others.

3. Organizations that achieve high levels of People-CMM maturity have higher likelihood of implementing effective software project management practices.

The Product:

1. Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered and technical and management constraints should be identified.
2. Without this information, it is impossible to define reasonable (and accurate) estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress.
3. Objectives identify the overall goals for the product (from the stakeholders’ points of view) without considering how these goals will be achieved.
4. Scope identifies the primary data, functions, and behaviors that characterize the product
5. The alternatives enable managers and practitioners to select a “best” approach, given the constraints imposed by delivery deadlines, budgetary restrictions, personnel availability, technical interfaces, and other factors.

The Process:

		1. A software process provides the framework from which a comprehensive plan for software development can be established. 2. A small number of framework activities are applicable to all software projects, regardless of their size or complexity. 3. A number of different task sets—tasks, milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. 4. Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement occur throughout the process. **The Project:** 1. To manage complexity, we conduct planned and controlled software projects. 2. The success rate for present-day software projects may have improved but our project failure rate remains much higher than it should be. 3. To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a common-sense approach for planning, monitoring, and controlling the project.	
	d	**List and explain basic principles of project scheduling.**	**4 M**
	Ans	**Basic Principles** • **Compartmentalization:** The project must be compartmentalized into a number of manageable activities and tasks. • **Interdependency:** The interdependency of each compartmentalized activity or task must be determined. • **Time allocation:** Each task to be scheduled must be allocated some number of work units. • **Effort validation:** Every project has a defined number of staff members. • **Defined responsibilities:** Every task that is scheduled should be assigned to a specific team member. • **Defined outcomes:** Every task that is scheduled should have a defined outcome.	Correct listing – 2M, explanation – 2M

- **Defined milestones:** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality.

3.		**Attempt any THREE of the following:**	**12 M**
	a	**Prescriptive process model and agile process model.**	**4 M**
	Ans		1 M for each Difference ,Any Four Difference

Prescriptive process model	**agile process mode**
Prescriptive process models stress detailed definition, identification, and application of process activates and tasks.	Agile process models emphasize project “agility” and follow a set of principles that lead to a more informal approach to software process.
A prescriptive model also describes how each of these elements are related to one another.	Agile methods note that not only do the software requirements change, but so do team members, the technology being used.
It is Process oriented.	It is people oriented.
It follows Life cycle model (waterfall, spiral) development model.	It follows Iterative and Incremental development model.
Documentation required is to be comprehensive and constant.	Documentation required is to be minimal and evolving.
Predictive planning is required	Adaptive planning is required.
Customers role is important.	Customers role is critical.
Formal communication is required.	Informal communication is required.
To maintain quality heavy planning and strict control with late heavy testing is required.	To maintain quality continuous control of requirements and

		development with continuous testing is required.	
	b	**Describe any four principles of communication for software engineering :**	**4 M**
	Ans	**Principle 1 Listen**: • Try to focus on the speaker‘s words, rather than formulating your response to those words. • Ask for clarification if something is unclear, but avoid constant interruptions. • Never become contentious in your words or actions (e.g., rolling your eyes or shaking your head) as a person is talking. **Principle 2 Prepare before you communicate:** • Spend the time to understand the problem before you meet with others. If necessary, perform some research to understand business domain. • If you have responsibility for conducting a meeting, prepare an agenda in advance of the meeting. **Principle 3 someone should facilitate the activity:** • Every communication meeting should have a leader (a facilitator) • To keep the conversation moving in a productive direction, • To mediate any conflict that does occur, and • To ensure that other principles are followed. **Principle 4 Face-to-face communication is best:** • It usually works better when some other representation of the relevant information is present. • For example, a participant may create a drawing /document that serve as a focus for discussion. **Principle 5 Take notes and document decisions:**	1M for one principle, Any four princple

- Someone participating in the communication should serve as a recorder and write down all important points and decisions.

Principle 6 Strive for collaboration:

- Collaboration occurs when the collective knowledge of members of the team is used to describe product or system functions or features.
- Each small collaboration builds trust among team members and creates a common goal for the team.

Principle 7 Stay focused; modularize your discussion:

- The more people involved in any communication, the more likely that discussion will bounce from one topic to the next.
- The facilitator should keep the conversation modular; leaving one topic only after it has been resolved.

Principle 8 If something is unclear, draw a picture:

- Verbal communication goes only so far.
- A sketch or drawing can often provide clarity when words fail to do the job.

Principle 9

(a) Once you agree to something, move on.

(b) If you can’t agree to something, move on.

(c) If a feature or function is unclear and cannot be clarified at the moment,

move on.

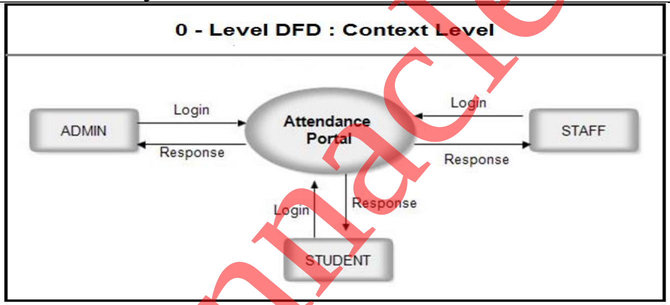
<table>
<tr><td></td><td></td><td>

- The people who participate in communication should recognize that many topics require discussion and that moving on is sometimes the best way to achieve communication agility.

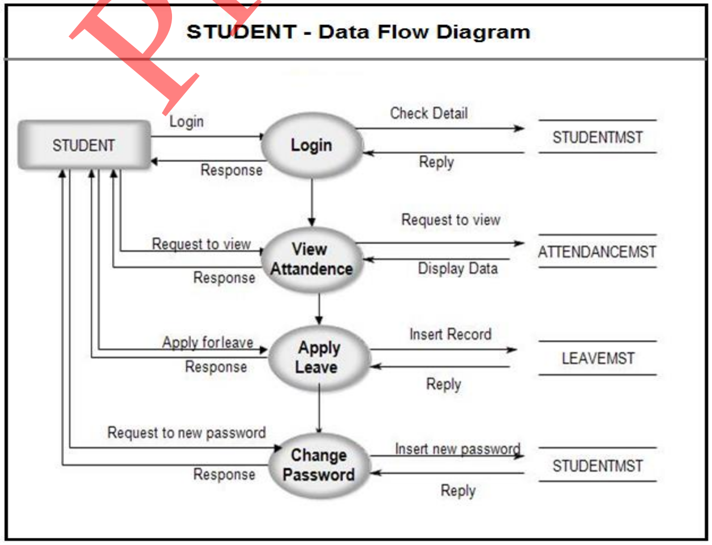
Principle 10 Negotiation is not a contest or a game: It works best when both parties win.

- There are many instances in which you and other stakeholders must negotiate functions and features, priorities, and delivery dates.
- If the team has collaborated well, all parties have a common goal. Still, negotiation will demand compromise from all parties.

</td><td></td></tr>
<tr><td></td><td>c</td><td><b>Draw proper labelled “LEVEL 1 Data Flow Diagram” (DFD) for student attendance system</b></td><td><b>4 M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>

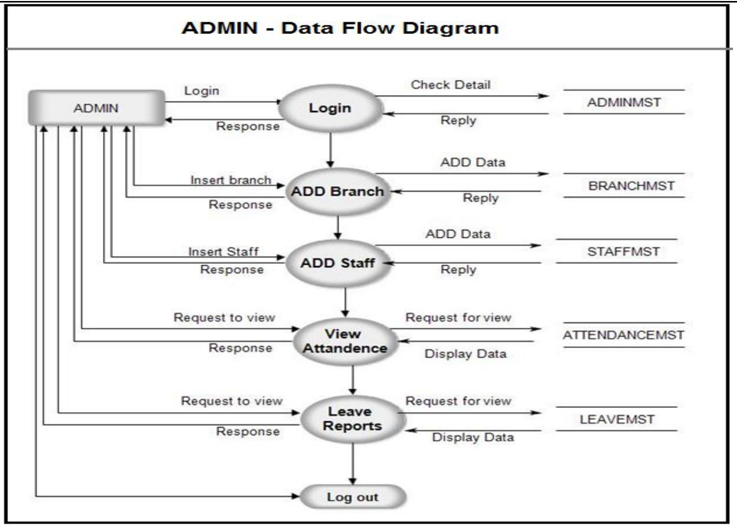


Level 0 Context Level



Level 1 DFD student

</td><td>1 M for level 0 and 3 M for level 1 DFD</td></tr>
</table>

		 ADMIN - Data Flow Diagram ADMIN Login Response Login Check Detail Reply ADMINMST Insert branch Response ADD Branch ADD Data Reply BRANCHMST Insert Staff Response ADD Staff ADD Data Reply STAFFMST Request to view Response View Attandence Request for view Display Data ATTENDANCEMST Request to view Response Leave Reports Request for view Display Data LEAVEMST Log out **Level 1 for admin**	
	d	**State importance of "Function point " and "lines of code" in concerned with project estimation**	**4 M**
	Ans	Currently two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). **Lines of Code (LOC)** LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored. **Function Point (FP):** The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different Functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number	2 M for function point and 2 M for lines of code

		of features. Each function when invoked reads some input data and transforms it to the corresponding output data. For example, the issue book feature (as shown in figure) of a Library Automation Software takes the name of the book as input and displays its location and the number of copies available. Thus, a computation of the number of input and the output data values to a system gives some indication of the number of functions supported by the system. Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent on the number of files and the number of interfaces.	
4.		**Attempt any THREE of the following:**	**12 M**
	a	**Describe Extreme programming with proper diagram**	**4 M**
	Ans	Extreme programming is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software. eXtreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements. Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where- • Each practice is simple and self-complete. • Combination of practices produces more complex and emergent behavior. Extreme Programming is based on the following values- • Communication • Simplicity • Feedback • Courage • Respect	1 M for Diagram and 3 M for explanation

Extreme Programming involves-

- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.

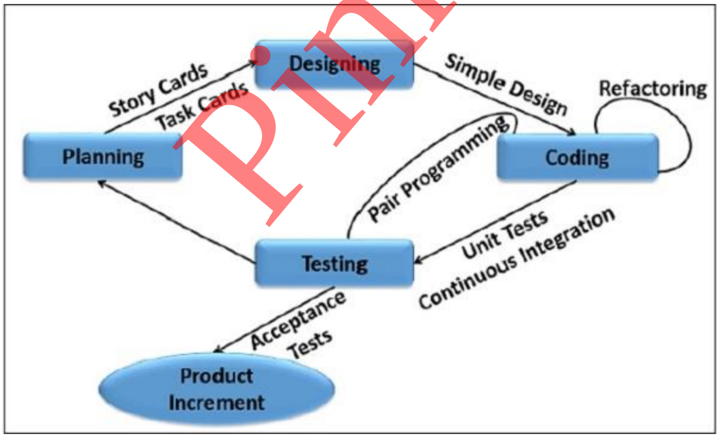
- Starting with a simple design just enough to code the features at hand and redesigning when required.

- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.

- Integrating and testing the whole system several times a day.

- Putting a minimal working system into the production quickly and upgrading it whenever required.

- Keeping the customer involved all the time and obtaining constant feedback. Iterating facilitates the accommodating changes as the software evolves with the changing requirements.



Extreme Programming solves the following problems often faced in the software development projects-

- Slipped schedules: Short and achievable development cycles ensure timely deliveries.

		• Cancelled projects: Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues. • Costs incurred in changes: Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected. • Production and post-delivery defects: Emphasis is on the unit tests to detect and fix the defects early. • Misunderstanding the business and/or domain: Making the customer a part of the team ensures constant communication and clarifications. • Business changes: Changes are considered to be inevitable and are accommodated at any point of time. • Staff turnover: Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit Extreme Programming takes the effective principles and practices to extreme levels. Extreme Programming • Code reviews are effective as the code is reviewed all the time. • Testing is effective as there is continuous regression and testing. • Design is effective as everybody needs to do refactoring daily. • Integration testing is important as integrate and test several times a day. • Short iterations are effective as the planning game for release planning and iteration planning.	
	b	**List and explain any four principles of "Core Principles" of Software Engineering.**	**4 M**
	Ans	**The First Principle: The Reason It All Exists**	1 M for one principle and explanation

- A software system exists for one reason: to provide value to its users. All decisions should be made with this in mind.
- Before specifying a system requirement, system functionality, before determining the hardware platforms, first determine, whether it adds value to the system.

The Second Principle: KISS (Keep It Simple, Stupid!)

- All design should be as simple as possible, but no simpler. This facilitates having a more easily understood and easily maintained system.
- It doesn't mean that features should be discarded in the name of simplicity.
- Simple also does not mean "quick and dirty." In fact, it often takes a lot of thought and work over multiple iterations to simplify.

The Third Principle: Maintain the Vision

- A clear vision is essential to the success of a software project.
- If you make compromise in the architectural vision of a software system, it will weaken and will eventually break even the well-designed systems.
- Having a powerful architect who can hold the vision helps to ensure a very successful software project.

The Fourth Principle: What You Produce, Others Will Consume

- Always specify, design, and implement by keeping in mind that someone else will have to understand what you are doing.
- The audience for any product of software development is potentially large.
- Design (make design), keeping the implementers (programmers) in mind. Code (program) with concern for those who will maintain and extend the system.
- Someone may have to debug the code you write, and that makes them a user of your code.

The Fifth Principle: Be Open to the Future

- A system with a long lifetime has more value.
- True "industrial-strength" software systems must last for longer.

		• To do this successfully, these systems must be ready to adapt changes. • Always ask "what if," and prepare for all possible answers by creating systems that solve the general problem. **The Sixth Principle: Plan Ahead for Reuse** • Reuse saves time and effort. • The reuse of code and designs has a major benefit of using object-oriented technologies. • Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated. **The Seventh principle: Think!** • Placing clear, complete thought before action almost always produces better results. • When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again. • If you do think about something and still do it wrong, it becomes a valuable experience. • Applying the first six principles requires intense thought, for which the potential rewards are enormous.	
	c	**Explain RMMM plan with example .**	**4 M**
	Ans	A risk management plan or plan risk management is a document that a prepares to foresee risks, estimate impacts, and define responses to risks. It also contains a risk matrix. A risk is "an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives." Risk is inherent with any and project manager should assess risks continually and develop plans to address them. The risk management plan contains an analysis of likely risks with both high and low impact, as well as mitigation strategies to help the project avoid being derailed should common problems arise. Risk management plans should be periodically reviewed by the project team to avoid having the analysis become stale and not reflective of actual potential project risks. Most critically, risk management plans include a risk strategy. There are two characteristics of risk i.e. uncertainty and loss. Risk Mitigation, Monitoring and Management (RMMM)	1 M for introduction to risk and 3 M for RMMM plan example

Risk analysis support the project team in constructing a strategy to deal with risks.

There are three important issues considered in developing an effective strategy:

Risk avoidance or mitigation - It is the primary strategy which is fulfilled through a plan.

Risk monitoring - The project manager monitors the factors and gives an indication whether the risk is becoming more or less.

Risk management and planning - It assumes that the mitigation effort failed and the risk is a reality.

RMMM PlanIt is a part of the software development plan or a separate document.

The RMMM plan documents all work executed as a part of risk analysis and used by the project manager as a part of the overall project plan.

The risk mitigation and monitoring starts after the project is started and the documentation of RMMM is completed.

Risk :Computer Crash

Mitigation :

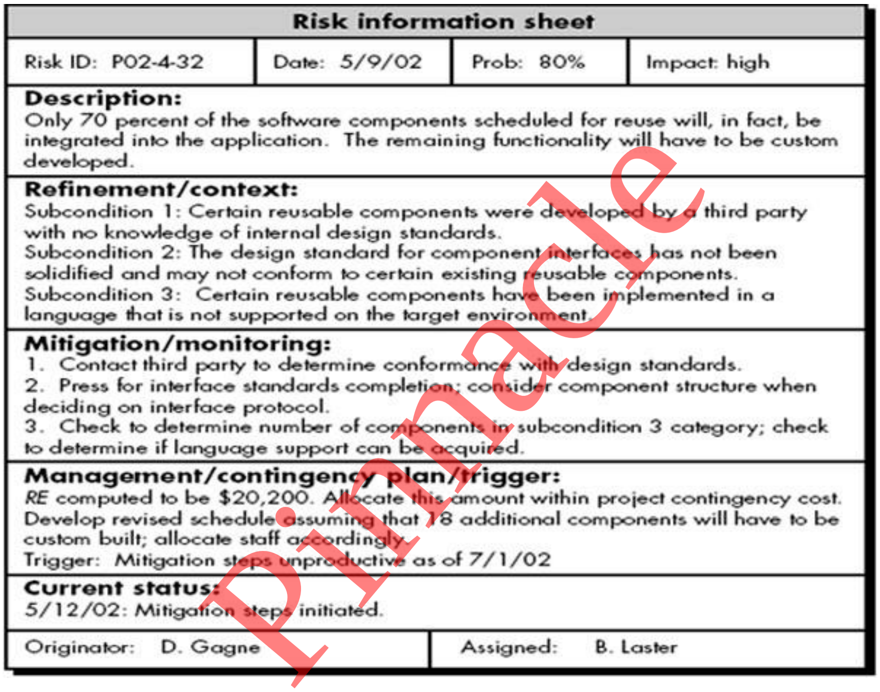
The cost associated with a computer crash resulting in a loss of data is crucial. A computer crash itself is not crucial, but rather the loss of data. A loss of data will result in not being able to deliver the product to the customer. This will result in a not receiving a letter of acceptance from the customer. Without the letter of acceptance, the group will receive a failing grade for the course. As a result the organization is taking steps to make multiple backup copies of the software in development and all documentation associated with it, in multiple locations. ·

Monitoring :

When working on the product or documentation, the staff member should always be aware of the stability of the computing environment they're working in. Any changes in the stability of the environment should be recognized and taken seriously. ·

Management :

The lack of a stable-computing environment is extremely hazardous to a software development team. In the event that the computing environment is found unstable, the development team should cease work on that system until the environment is made stable again, or should move to a system that is stable and continue working there.



Risk information sheet

Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/02

Current status:
5/12/02: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

d **Explain any one project cost estimation approach.** **4 M**

Ans **(i) Heuristic**

Any one approach -Explanation 4 M

Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: single variable model and the multi variable model.

Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size. A single variable estimation model takes the following form:

Estimated Parameter = $c_1 * e_1{}^{d}{}_{1}$

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). Estimated Parameter is the dependent parameter to be estimated. The dependent parameter to be estimated could be effort, project duration, staff size, etc. c1 and d1 are constants. The values of the constants c1 and d1 are usually determined using data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.

A multivariable cost estimation model takes the following form:

Estimated Resource = $c_1 * e_1{}^{d}{}_{1} + c\,_2 * e_2\,{}^{d}{}_{2}\ + ...$

Where e1, e2, … are the basic (independent) characteristics of the software already estimated, and c1, c2, d1, d2, … are constants.

(ii) Analytical

Halstead’s Software Science – An Analytical Technique
Halstead’s software science is an analytical technique to measure size, development effort, and development cost of software products. Halstead used a few primitive program parameters to develop the expressions for over all program length, potential minimum value, actual volume, effort, and development time.

Example:

Let us consider the following C program:

```
main( )
{
      int a, b, c, avg;

      scanf(“%d %d %d”, &a, &b, &c);
      avg = (a+b+c)/3;
      printf(“avg = %d”, avg);
}
```

The unique operators are:
main,(),{},int,scanf,&,“,”,“;”,=,+,/, printf

The unique operands are:
a, b, c, &a, &b, &c, a+b+c, avg, 3, “%d %d %d”, “avg = %d”

Therefore,
n1 = 12, n2 = 11

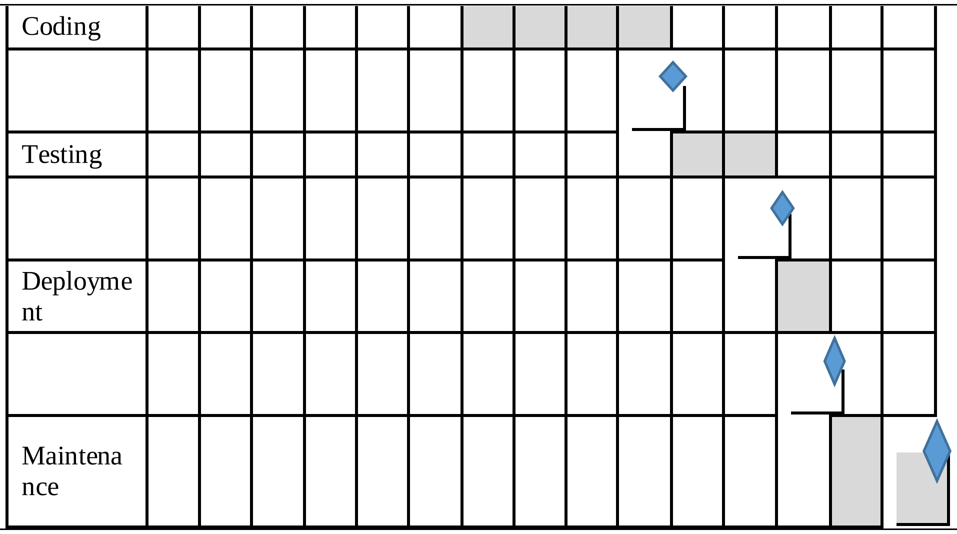
Estimated Length = (12*log12 + 11*log11)
= (12*3.58 + 11*3.45)
= (43+38) = 81

Volume = Length*log(23)
= 81*4.52
= 366

	e	**Draw time chart for Libraray management system System (5 days a week). Consider broad phases of SDLC.**	**4 M**

Ans

	Week 1					Week 2					Week 3				
	D 1	D 2	D 3	D 4	D 5	D 1	D 2	D 3	D 4	D 5	D 1	D 2	D 3	D 4	D 5
Ananlysis	▨	▨	▨												
			◆												
Design				▨	▨	▨									
						◆									



5 **Attempt any TWO of the following:** **12 M**

a **Enlist requirement Gathering and Analysis for web based project for registering candidates for contest** **6 M**

Ans

Requirement gathering includes suggestions and ideas for ways to best capture the different types of requirement (functional, system, technical, etc.) during the gathering process.

6M – 1M for 1 point

1. **Functional requirements**

The functional requirements are the requirements that will enable solving the real world problem. The web based project must be able to register the candidates for contest.

2. **Non-functional requirements**

These requirements aim at providing support, security and facilitate user interaction segment of the website.

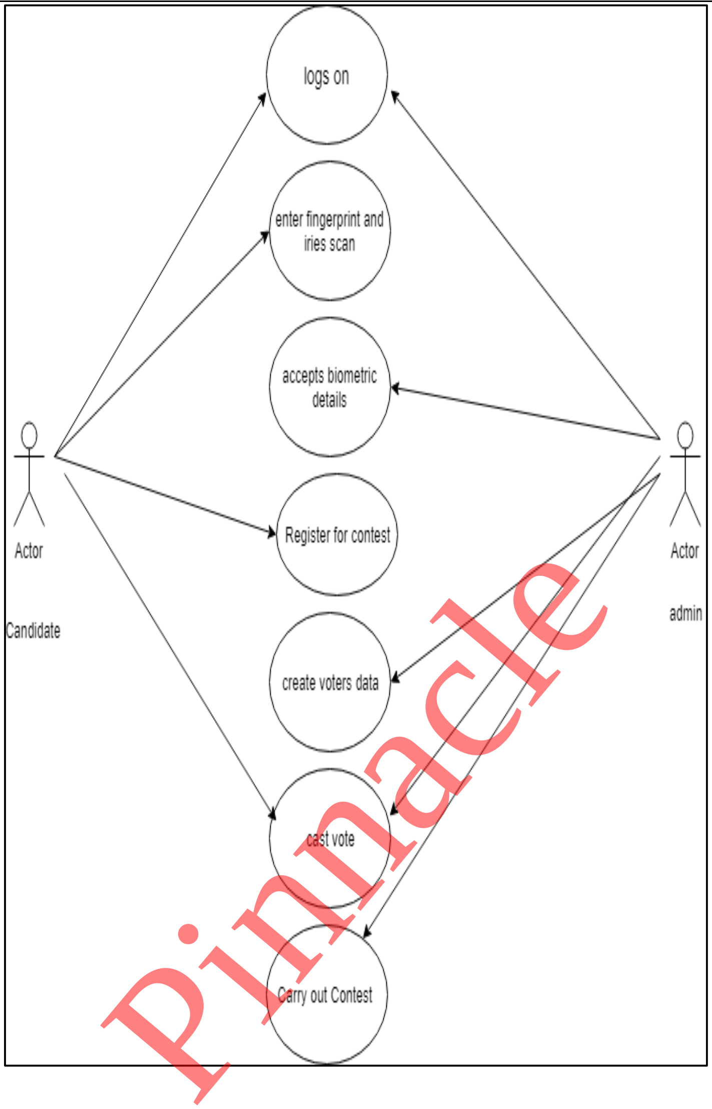
- The project must enable the candidates to safely enter their passwords and other biometric information.
- There must be no repetition in registration of candidates i.e the candidates must be registered only once.

3. **Business requirements**: They are high-level requirements that are taken from the business case from the projects.

For eg:-

Qualifying criteria	Allowed/Disallowed	
Indian Nationality Registration	Allowed	
Age>18	Allowed	
No criminal record	Allowed	

4. **Architectural and Design requirements**: These requirements are more detailed than business requirements. It determines the overall design required to implement the business requirement.
    - The web based project must be supported by different operating systems , PC and mobile compatibility etc.
    - The hardware must be integrated so as to accept the fingerprint details of a candidate and register him in the system.
    - The database of the project must be updated.
5. **System and Integration requirements**: At the lowest level, we have system and integration requirements. It is detailed description of each and every requirement. It can be in form of user stories which is really describing everyday business language. The requirements are in abundant details so that developers can begin coding.



6. **Documenting the requirement using traceability matrix**
A Traceability Matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.It is used to track the requirements and to check the current project requirements are met.

Req no	Description	Test case ID	Status
1	Login	TC1	TC1 Pass
2	Feed in biometric details	TC2	TC2 Pass

	b	**Differentiate between White box and Black Box Testing.**	**6 M**

Ans

Sr.no	White box testing	Black Box Testing
1	The tester needs to have the knowledge of internal code or program.	This technique is used to test the software without the knowledge of internal code or program.
2	It aims at testing the structure of the item being tested.	It aims at testing the functionality of the software.
3	It is also called structural testing, clear box testing, code-based testing, or glass box testing.	It also knowns as data-driven, box testing, data-, and functional testing.
4	Testing is best suited for a lower level of testing like Unit Testing, Integration testing.	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing.
5	Statement Coverage, Branch coverage, and Path coverage are White Box testing technique.	Equivalence partitioning, Boundary value analysis are Black Box testing technique
6	Can be based on detailed design documents.	Can be based on Requirement specification document.

6M- 1M for 1point

	c	**Describe COCOMO II model for evaluating size of software project with any three parameters in detail**	**6 M**
	Ans	COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.	3M for Description, 3M for parameters

COCOMO II provides the following three-stage series of models for estimation of Application Generator, System Integration, and Infrastructure software projects:

End User Programming	Application Generators and composition aids	Infrastructure
	Application Composition	
	System Integration	

- The Application Composition Model

  This model involves prototyping efforts to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity. The costs of this type of effort are best estimated by the Applications Composition model. It is suitable for projects built with modern GUI-builder tools. It is based on new Object Points.

- The Early Design Model

  The Early Design model involves exploration of alternative software/system architectures and concepts of operation. It uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KSLOC.

- The Post-Architecture Model

The Post-Architecture model involves the actual development and maintenance of a software product

Estimates

In COCOMO II effort is expressed as Person Months (PM). The inputs are the Size of software development, a constant, A, and a scale factor, B. The size is in units of thousands of source lines of code (KSLOC). The constant, A, is used to capture the multiplicative effects on effort with projects of increasing size.

The parameters used in COCOMO II are described below:-

a. **Person month**- A person month is the amount of time one person spends working on the software development project for one month. The nominal effort for a given size project and expressed as person months (PM) is given by Equation 1.

$PM_{nominal} = A^{*} (Size)^{B}$

Where

A- constant

$B = 0.91 + 0.01 \sum(\text{exponent driver ratings})$

- B ranges from 0.91 to 1.23

- 5 drivers; 6 rating levels each

b. **Maintenance size** is the amount of project code that is change. It is calculated as below:-

Size=[(BaseCodeSize) *MCF] *MAF

COCOMO II uses the reuse model for maintenance when the amount of added or changed base source code is less than or equal to 20% or the new code being developed. Base code is source code that already exists and is being changed for use in the current project. For maintenance projects that involve more than 20% change in the existing base code (relative to new code being developed) COCOMO II uses maintenance size.

c. Maintenance Change Factor MCF

The percentage of change to the base code is called the Maintenance Change Factor (MCF).

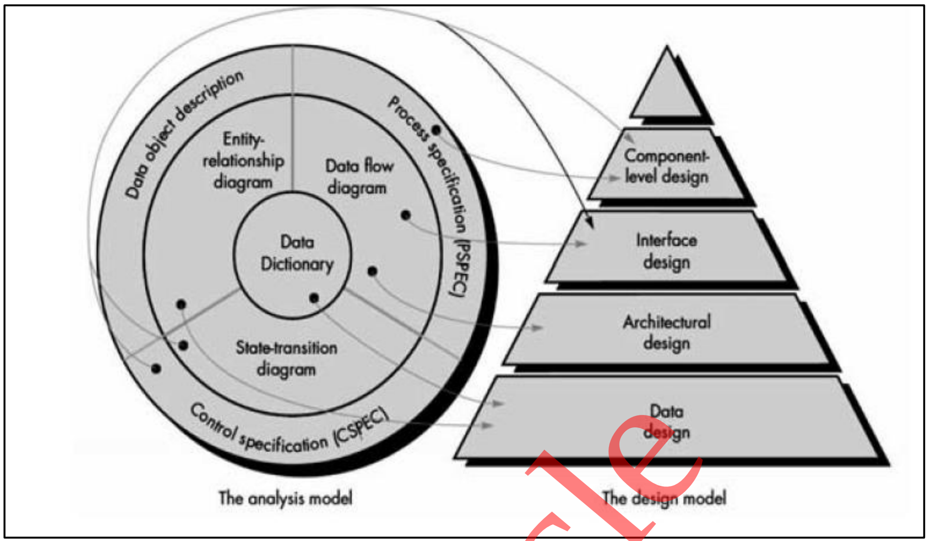
MCF= (SizeAdded +SizeModified)/BaseCodeSize

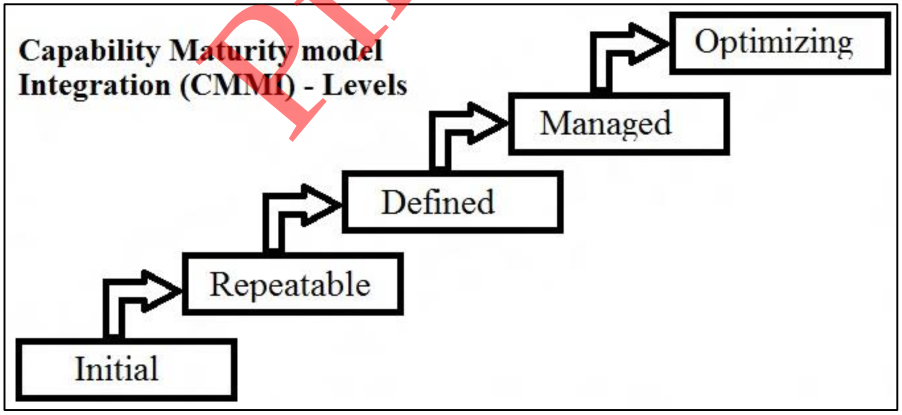
d. Maintenance effort (MAF)

COCOMO II instead used the Software Understanding (SU) and Programmer Unfamiliarity (UNFM) factors from its reuse model to model the effects of well or poorly structured/understandable software on maintenance effort.

MAF=1+ (SU.01*UNFM)

6		**Attempt any TWO of the following:**	**12 M**

	a	**Draw and explain Transition diagram from requirement model to design model**	**6 M**
	Ans	**Transition diagram from requirement model to design model**  Data object description Entity-relationship diagram Data flow diagram Process specification (PSPEC) Data Dictionary State-transition diagram Control specification (CSPEC) The analysis model Component-level design Interface design Architectural design Data design The design model Software requirements, manifested by the data, functional, and behavioural models, feed the design task. Using one of a number of design methods, the design task produces a data design, an architectural design, an interface design, and a component design. Each of the elements of the analysis model provides information that is necessary to create the four design models required for a complete specification of design. Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for ―good‖ design. In the software engineering context, design focuses on four major areas of concern: data, architecture, interfaces, and components Design begins with the requirements model. The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design activity. Part of data design may occur in conjunction with the design of software architecture. More detailed data design occurs as each software component is designed. The architectural design defines the relationship between major structural elements of the software, the design pattern that can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which architectural design patterns can be applied.	2M –diiagram, 4M – explanation

		The architectural design representation the framework of a computer-based system can be derived from the system specification, the analysis model, and the interaction of subsystems defined within the analysis model. The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it. An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, data and control flow diagrams provide much of the information required for interface design. The component-level design transforms structural elements of the software architecture into a procedural description of software components. Information obtained from the PSPEC, CSPEC, and STD serve as the basis for component design.	
	b	**Describe CMMI. Give significance of each level.**	**6 M**
	Ans	**The Capability Maturity Model Integration (CMMI)**, a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity. The CMMI represents a process meta-model in two different ways: (1) Continuous model and (2) Staged model. The continuous CMMI meta-model describes a process in two dimensions. Each process area (e.g. project planning or requirements management) is formally assessed against specific goals and practices and is rated according to the following capability levels:  Capability Maturity model Integration (CMMI) - Levels Optimizing Managed Defined Repeatable Initial **Level 1: Initial**. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.	1M- diagram , 5M- 5 points

		Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. **Level 3: Defined**. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2 **Level 4: Managed**. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3 **Level 5: Optimizing**. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.	
	c	**Identify and enlist requirement for given modules of employee management software**	**6 M**
	Ans	i. Employee detail ii. Employee salary iii.Employee performance This is with perspective of employee management software. Requirements for following modules will be as i. Employee details a. Getting information about the customer b. Updation of employee details (department, change of address, emp_code etc) c. Assignment of tasks , duties and responsibilities. d. Recording of employee attendance. ii. Employee salary a. Salary calculation	2 M for employee detail, salary, performance each

		b. Allowances, special bonus calculation and approval c. Tax statement/certificate d. Apply loan/approvals iii. Performance a. Recording annual performance b. Details about parameters for performance appraisal c. Analysis performance and determining hike in payment.	