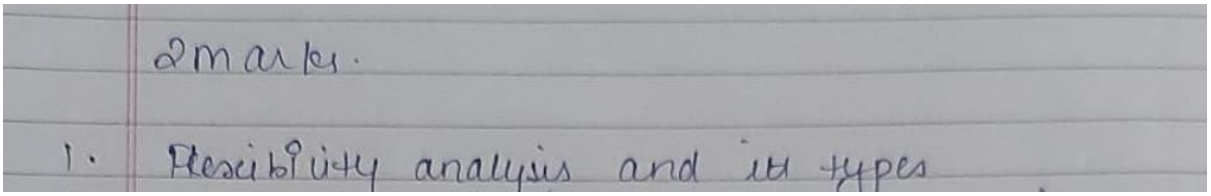
Software Engineering.

2 marks.

1. Flexibility analysis and its types
2. Different application of project scheduling
3. Justifying different requirement model in in software Engineering
4. Software project estimation and its example
5. Different types of cost benfit analysis and their example
6. Importance for requirement gathering
7. Function point analysis
8. Lines of code and example
9. Cohesion and coupling
10. Scenario based model
11. Advantage of design concept
12. Advantage of cocomo model.

5 marks.

1. Fish bone diagram and its example / steps / detail
2. Class diagram -
   Attendance management & airlines management
3. What is good design in software Engineering
4. Cohesion and coupling - difference
5. Use case diagram -
   Tourism application & e-commerce
6. COCOMO model.



Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system.

As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view.

1. **Technical Feasibility –**
   In Technical Feasibility current resources both hardware software along with required technology are analysed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development.
2. **Operational Feasibility –**
   In Operational Feasibility degree of providing service to requirements is analysed along with how much easy product will be to operate and maintenance after deployment.
3. **Economic Feasibility –**
   In Economic Feasibility study cost and benefit of the project is analysed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development .
4. **Legal Feasibility –**
   In Legal Feasibility study project is analysed in legality point of view. This includes analysing barriers of legal implementation of project, data protection acts or social media laws.
5. **Schedule Feasibility –**
   In Schedule Feasibility Study mainly timelines/deadlines is analysed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.

2. Different application of project scheduling

Applications of project scheduling :

1. Resource allocation: Project scheduling helps software engineers to allocate resources such as developers, testers, and project managers efficiently. With a well-planned schedule, team members can be assigned tasks according to their strengths and availability, ensuring that the project is completed on time and within budget.
2. Task sequencing: Project scheduling enables software engineers to define the order in which tasks are to be executed. This ensures that the project progresses in a logical and systematic manner, with each task being completed before the next one begins.
3. Time management: Project scheduling helps software engineers to manage their time effectively by defining deadlines for each task and tracking progress against those deadlines. This allows them to identify and address any delays or bottlenecks that may arise during the project.
4. Risk management: Project scheduling helps software engineers to identify potential risks and plan for contingencies in case they occur. This includes developing alternative schedules and resource plans to mitigate the impact of any risks that may arise.
5. Communication: Project scheduling helps software engineers to communicate project timelines and milestones to stakeholders such as clients, managers, and team members. This ensures that everyone is on the same page and that expectations are managed effectively.

3. Justifying different requirement model in in software Engineering

- **Requirements modeling** is the process of identifying the requirements this software solution must meet in order to be successful. Requirements modeling contains several sub-stages, typically: scenario-based modeling, flow-oriented modeling, data modeling, class-based modeling, and behavioral modeling.

- **Scenario-based Model :**
  Using a scenario-based approach, system is described from user's point of view. **For example**, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases.

Class-based model :

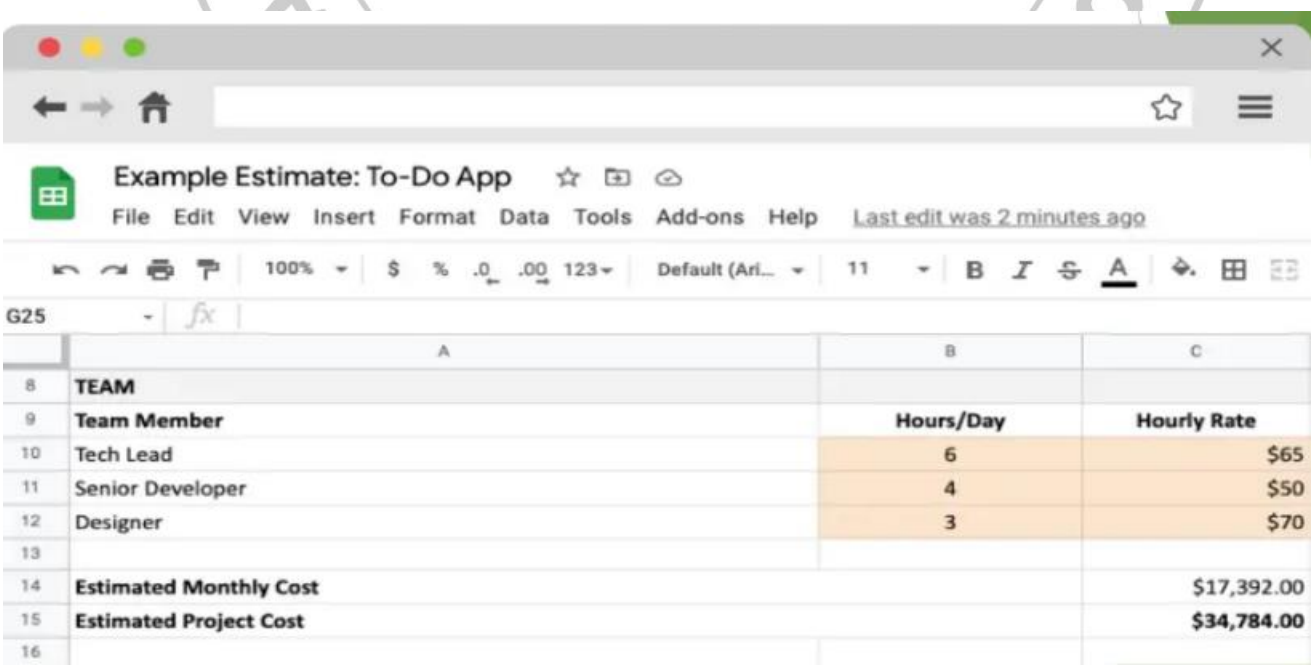
- Class-based modeling begins by identifying the classes in the use case.
- A collection of things that have similar attributes and common behaviors i.e., objects are categorized into classes. In addition to

- Behavioral Modeling
- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the

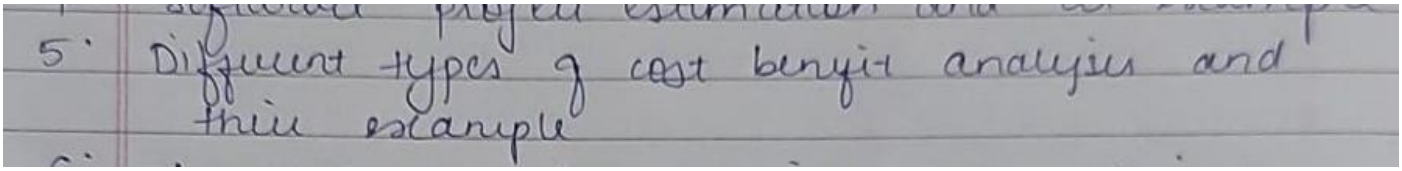
4. Software project estimation and its example

- *Estimating is, by definition, a guess about the future.*

- What is software project estimation?
- What is software project estimation? Literally, estimation is a process to predict the time and the cost that a project requires to be finished appropriately.

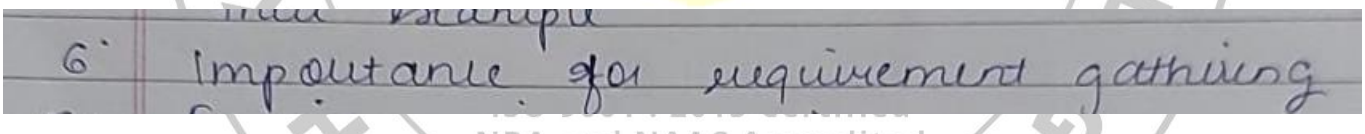


	A	B	C
8	**TEAM**		
9	**Team Member**	**Hours/Day**	**Hourly Rate**
10	Tech Lead	6	$65
11	Senior Developer	4	$50
12	Designer	3	$70
13			
14	**Estimated Monthly Cost**		$17,392.00
15	**Estimated Project Cost**		**$34,784.00**
16			



Cost-Benefit Analysis (CBA) is an important tool in software engineering used to evaluate the potential costs and benefits of a software development project. There are several types of cost-benefit analysis that can be used in software engineering, including:

1. Economic Cost-Benefit Analysis: This type of CBA evaluates the economic costs and benefits of a software development project, such as the costs of development, maintenance, and support, as well as the potential benefits of increased revenue, productivity, or market share.
2. Social Cost-Benefit Analysis: This type of CBA evaluates the social costs and benefits of a software development project, such as the impact on the environment, public health, or societal well-being.
3. Technical Cost-Benefit Analysis: This type of CBA evaluates the technical costs and benefits of a software development project, such as the costs of acquiring hardware and software, implementing new technologies, or training personnel.
4. Risk-Based Cost-Benefit Analysis: This type of CBA evaluates the potential risks and benefits of a software development project, such as the risk of project failure, cost overruns, or security breaches, and weighs these risks against potential benefits.
5. Sensitivity Analysis: This type of CBA evaluates the sensitivity of the costs and benefits of a software development project to changes in key assumptions, such as changes in market demand, technological obsolescence, or labor costs.



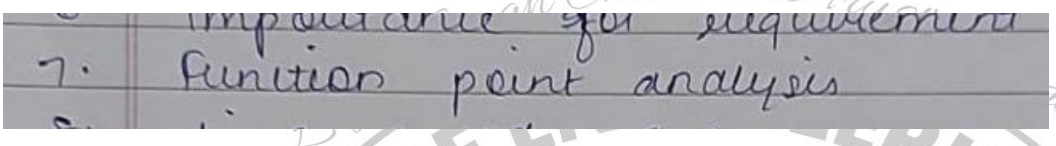
1. Helps to define project scope: Requirement gathering helps to define the scope of the project and identify the features and functionalities that the system needs to have. This ensures that the project stays on track and meets the desired outcomes.

2. Reduces development costs: Proper requirement gathering ensures that the development team has a clear understanding of what is required, which helps to avoid costly rework and delays in the development process.

3. Improves communication and collaboration: Requirement gathering involves communication and collaboration between stakeholders, users, and the development team. This helps to ensure that everyone is on the same page and that there is a shared understanding of the project goals and requirements.

4. Enhances user satisfaction: Requirement gathering ensures that the final product meets the needs and expectations of the users. This enhances user satisfaction and increases the chances of user adoption and acceptance.

5. Mitigates project risks: Proper requirement gathering helps to identify potential risks and issues early in the project, which allows for appropriate risk mitigation strategies to be put in place. This helps to avoid project delays, cost overruns, and other negative impacts.



- Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement.
- It assesses the functionality delivered to its users, based on the user's external view of the functional requirements.
- It measures the logical view of an application, not the physically implemented view or the internal technical view.
- The Function Point Analysis technique is used to analyze the functionality delivered by software and Unadjusted Function Point (UFP) is the unit of measurement.

Objectives of FPA:

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.

8. Lines of code and example

- A **line of code (LOC)** is any line of text in a code that is not a comment or blank line, and also header lines.
- LOC clearly consists of all lines containing the declaration of any variable, and executable and non-executable statements.
- As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

```
void main()
{
    int fN, sN, tN;
    cout << "Enter the 2
integers: ";
    cin >> fN >> sN;
    // sum of two numbers in
stored in variable sum
    sum = fN + sN;
    // Prints sum
    cout << fN << " + " <<
sN << " = " << sum;
    return 0;
}
```

- Here also, If LOC is simply a count of the numbers of lines then the above function shown contains 11 lines of code (LOC).
- But when comments and blank lines are ignored, the function shown above contains 9 lines of code (LOC).

9. Cohesion and coupling

Cohesion refers to the degree to which elements within a module work together to fulfil a single, well-defined purpose. High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.

Coupling refers to the degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules. Low coupling means that modules are independent and changes in one module have little impact on other modules.

10. Scenario based model

Using a scenario-based approach, system is described from user's point of view. For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases.

- For the building of analysis and design models, it is essential for software engineers to understand how end users and other actors want to interact with the system
- Analysis Modeling with UML begins with the creation of scenarios in the form of use-cases, activity diagrams and swim-lane diagrams
- Use cases can be represented as a text narrative or as a ordered sequence of user actions or by using a template

11. Advantage of design concept

Some of the advantages of using design concepts in software engineering include:

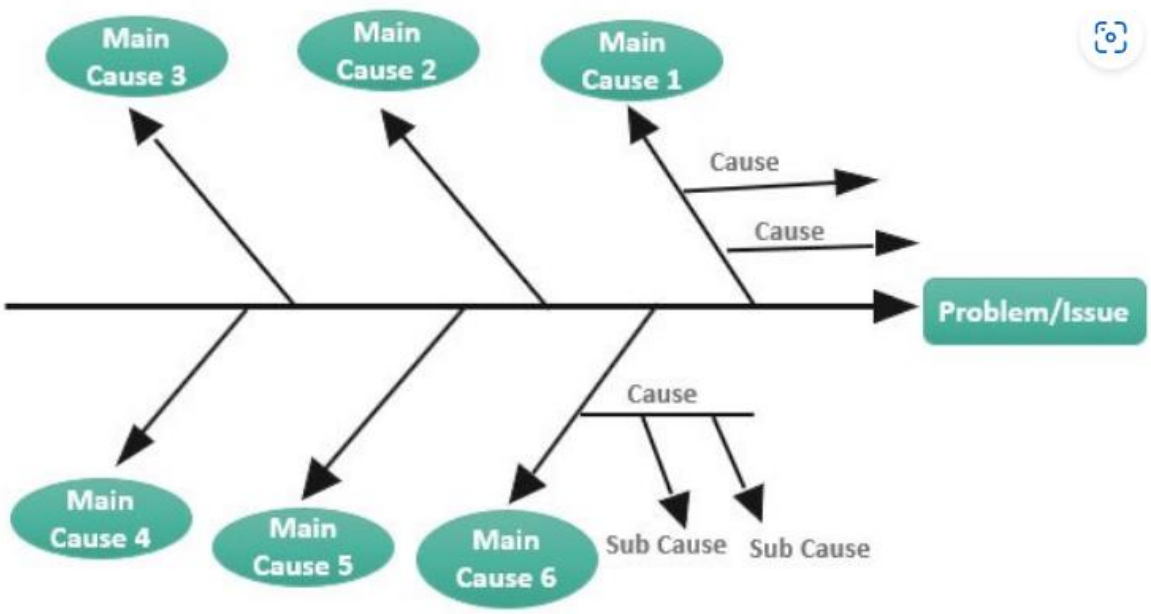
1. Modularity: Design concepts promote modular software design, which breaks down complex systems into smaller, manageable components. This makes it easier to maintain and update the system over time, as changes can be made to individual modules without affecting the entire system.
2. Scalability: Design concepts promote scalability, which allows software systems to grow and adapt to changing user needs and business requirements. Scalable software design enables the addition of new features, the handling of larger data volumes, and the support of more users as the system evolves.
3. Reusability: Design concepts promote the development of reusable software components, which can be used in multiple applications or systems. This reduces development time and costs and improves software quality by eliminating the need to write new code from scratch for each project.
4. Flexibility: Design concepts promote flexible software design, which allows software systems to be adapted and customized to meet specific user needs. This enables software developers to create software systems that can be easily customized to meet different business requirements, without requiring major code changes.
5. Maintainability: Design concepts promote maintainable software design, which reduces the cost and effort required to maintain and update software systems over time. This makes it easier to identify and fix bugs, improve performance, and update the system with new features or capabilities as needed.

12. Advantage of cocomo model.

- It works on historical data and provides more accurate details.
- Easy to implement with various factors. One can easily understand how it works.
- Easy to estimate the total cost of the project.
- The drivers are very helpful to understand the impact of the different factors that affect project crises.

1. Fish bone diagram and its example/steps/detail

The Fishbone diagram or Ishikawa diagram is a modern quality management tool that explains the cause and effect relationship for any quality issue that has arisen or that may arise. It provides the visual representation of all the possible causes of a problem to analyze and find out the root cause. It is a tool that can be used both proactively and reactively.



Follow the steps to create a fishbone diagram.

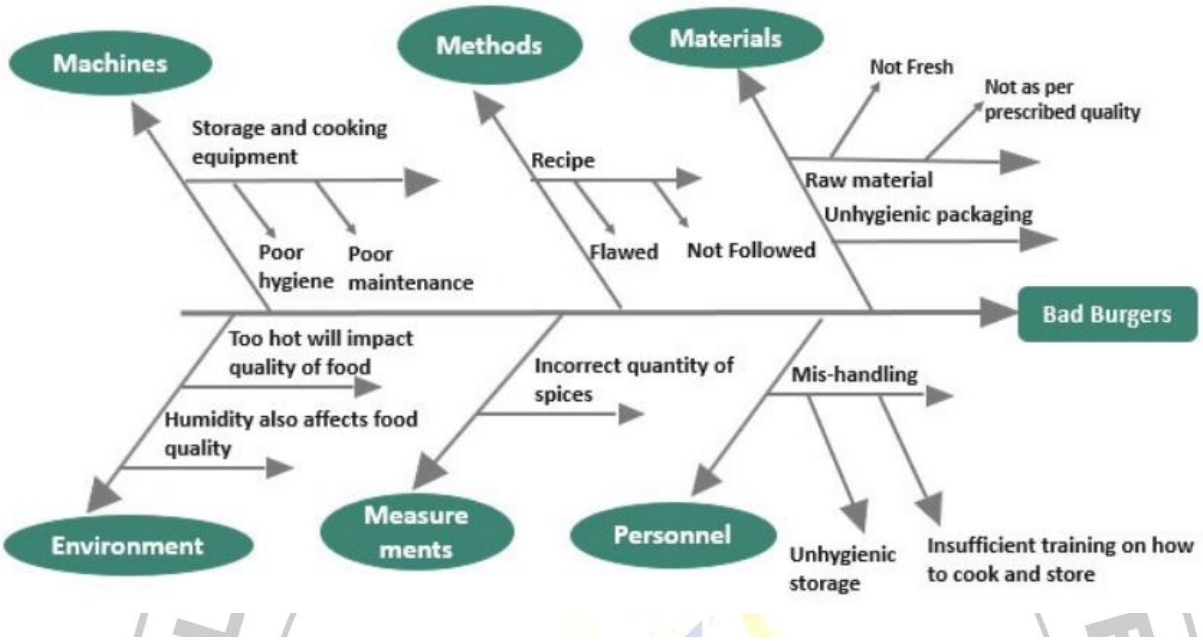
1. **Make the head of the fish on the right. Here we mention the subject that needs our attention.**

2. **Draw a backbone on the left.**

3. **Draw branches to the backbones that will list the main causes. List four to eight main causes.**

4. **Under these main causes are listed the causes and sub-causes. These can be identified by organizing a brainstorming session or minutely following the whole process and identifying all the possible reasons that can lead to quality damage.**

Let's take one more example. Looking at how well we addressed the above problem, we have hired consultants from a burger store to identify the reasons for preparing a bad-quality burger. The question sounds tasty. I mean engaging.

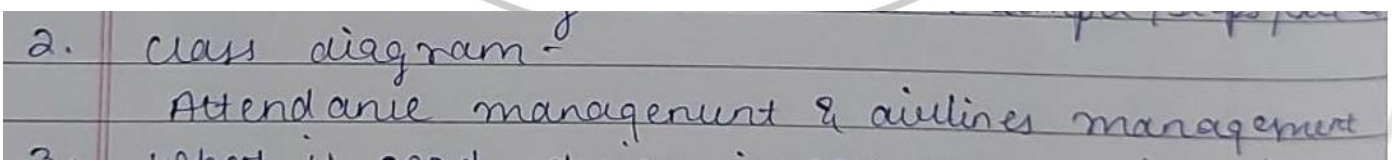


Benefits

- This technique is widely used in product design, quality improvement, and defect minimization.
- Instead of pointing out just one reason, this technique gives us a gamut of all the possible reasons that assist not only in identifying the root cause of the current problem and avoiding any future mishappening.

Limitations

This technique should not be used as a one-time activity. Instead, this activity should be undertaken continuously to be proactive in determining any possible loophole.

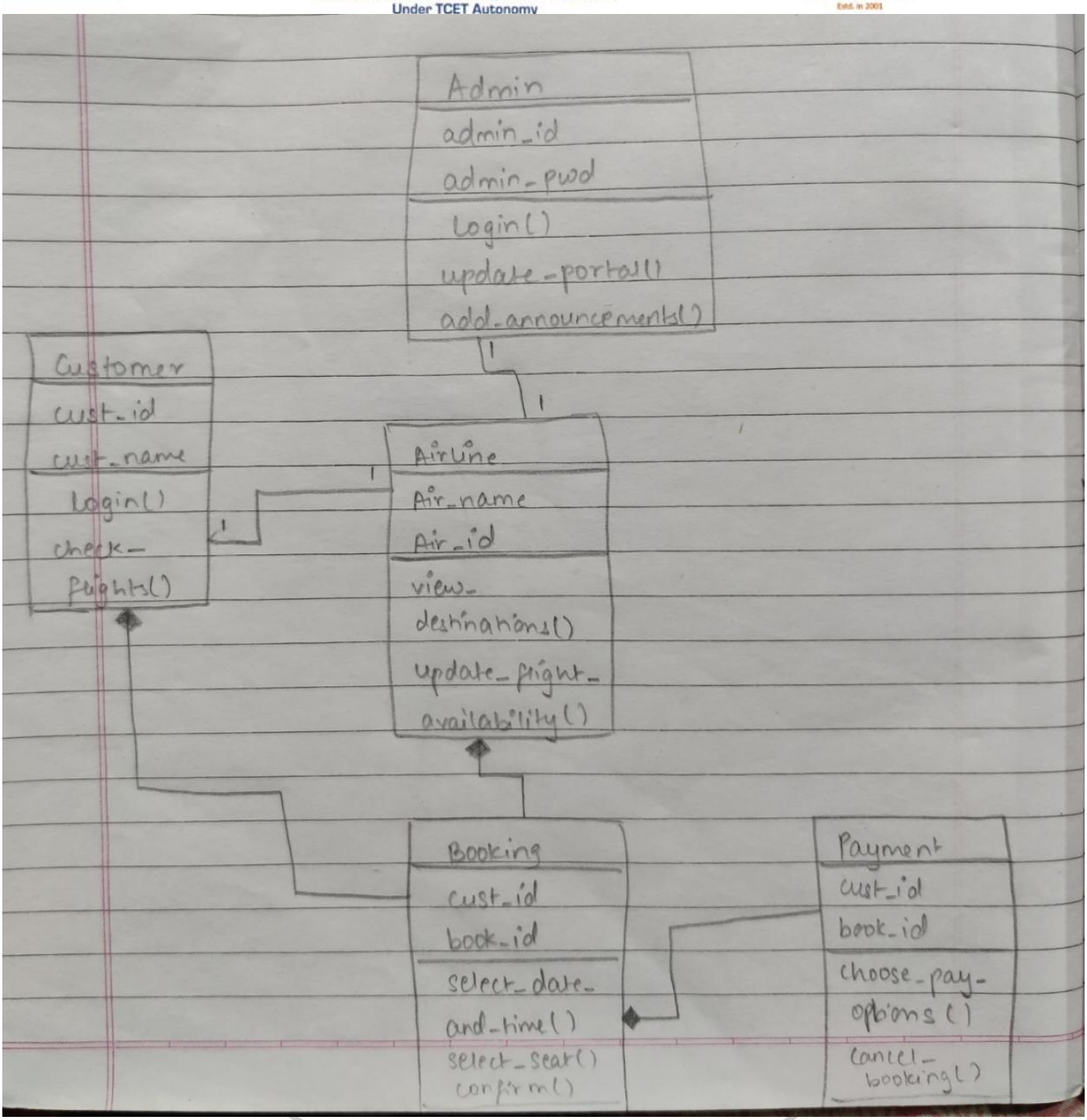


The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

Vital components of a Class Diagram

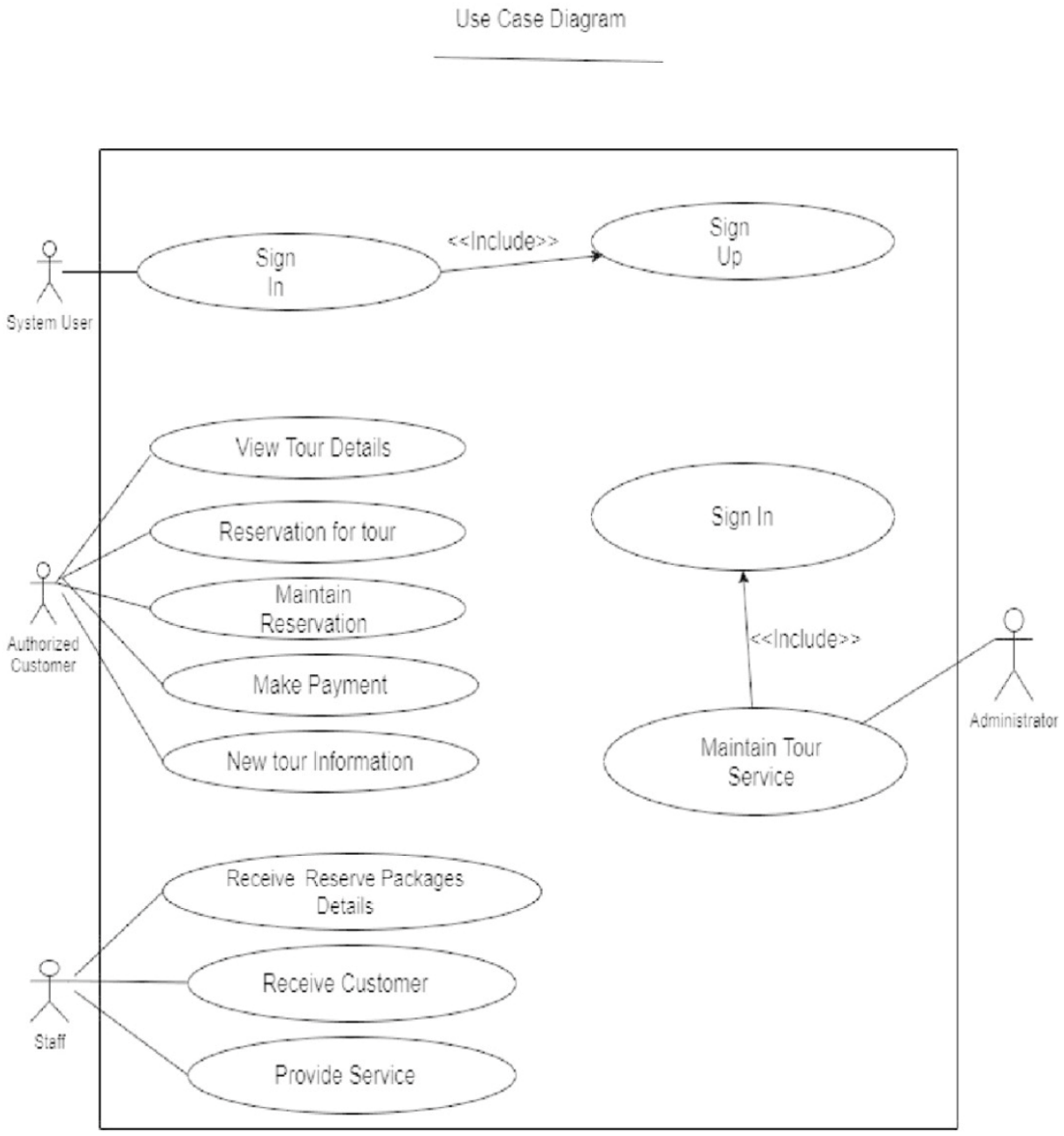
The class diagram is made up of three sections:

- **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
  a. Capitalize the initial letter of the class name.
  b. Place the class name in the center of the upper section.
  c. A class name must be written in bold format.
  d. The name of the abstract class should be written in italics format.
- **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:
  a. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
  b. The accessibility of an attribute class is illustrated by the visibility factors.
  c. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.
- **Lower Section:** The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.


Admin
admin_id
admin_pwd
login()
update_portal()
add_announcements()
Customer
cust_id
cust_name
login()
check_flights()
AirLine
Air_name
Air_id
view_destinations()
update_flight_availability()
Booking
cust_id
book_id
select_date_and_time()
select_seat()
confirm()
Payment
cust_id
book_id
choose_pay_options()
cancel_booking()
1
1
1
1

TOURISM USE CASE DIAGRAM:

A use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.



- Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
- Actors: Stick figures that represent the people actually employing the use cases.
- Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.

Include relationship: This is used when a use case is always or frequently included within another use case. The included use case is not complete on its own and is always invoked by the base use case. The included use case represents a part of the functionality of the base use case. The include relationship is shown as a dotted arrow with an open arrowhead pointing to the included use case.

Extend relationship: This is used when a use case is optional and extends the behavior of another use case. The extended use case is optional and is invoked only under certain conditions or scenarios. The extended use case adds new functionality to the base use case, but does not affect its basic behavior. The extend relationship is shown as a dotted arrow with an open arrowhead pointing to the extending use case.

	Cohesion	Coupling
Definition	Measures how closely related elements within a module are	Measures how dependent two modules are on each other
Purpose	Determines how well a module performs a single, well-defined task	Determines how well modules work together to achieve a common goal
Effect on Maintainability	High cohesion leads to better maintainability	high coupling leads to poor maintainability
Effect on Modifiability	High cohesion makes it easy to modify a module without affecting other parts of the system	High coupling makes it difficult to modify one module without affecting another module
Internal/External Property	Cohesion is an internal property of a module	Coupling is an external property of a module
Risk of Errors/Bugs	~~Low~~ High cohesion reduces the risk of errors and bugs	High coupling increases the risk of errors and bugs
Focus	Cohesion focuses on the internal structure of a module	Coupling focuses on the interactions between modules
Measurement	Cohesion is measured using metrics such as function points or cyclomatic complexity	Coupling is measured using metrics such as fan-in or fan-out
Types	Cohesion can be of types such as functional, sequential, communicational, and procedural cohesion	Coupling can be of types such as content, common, control, stamp, and data coupling