

## CHAPTER: 3 SOFTWARE MODELLING & DESIGN

(14 Marks)

3.1 Translating Requirements model into Design model, Data Modelling.

3.2 Analysis Model, Elements of Analysis Model.

3.3 Design modelling concepts.

3.4 Design notations: DFD, Structured flowchart, decision tables.

3.5 Testing: Introduction, Black box testing, White box testing, Level of testing – Unit testing.

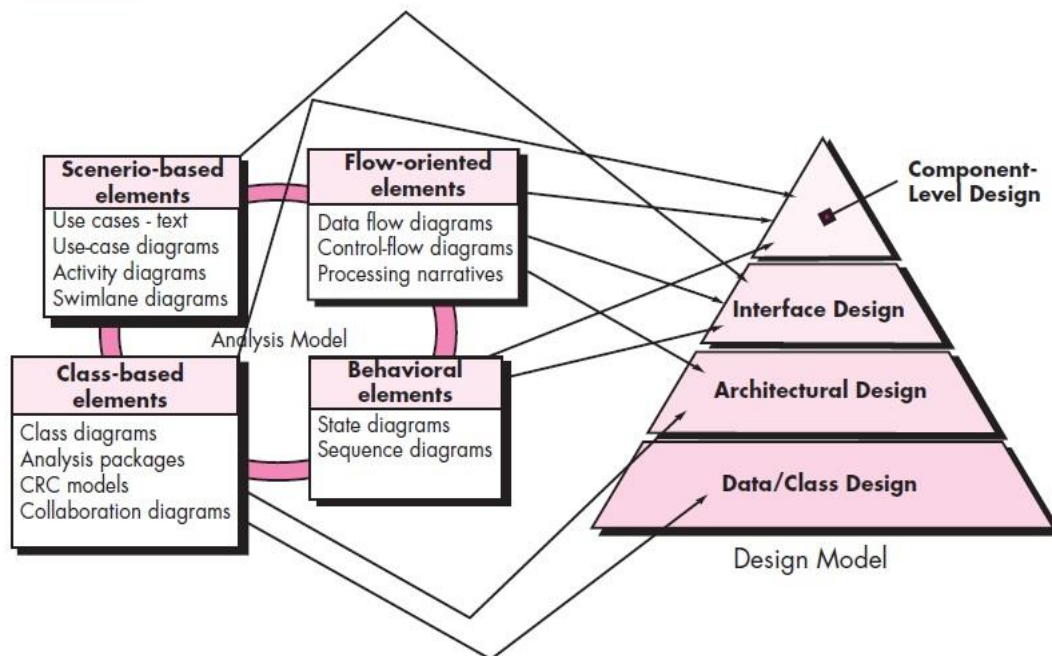
3.6 Test Documentation: Test case template, Test plan, Defect Report, Test Summary report.

---

### 3.1 Translating Requirements model into Design model:

- Once software requirements have been analyzed and modelled, software design is the last software engineering action within the modelling activity and sets the stage for **construction** (code generation and testing).
- Each of the elements of the requirements model provides information that is necessary to create the four design models required for a complete specification of design. The flow of information during software design is illustrated in Figure.

**FIGURE 8.1** Translating the requirements model into the design model



- The requirements model, expressed by scenario-based, class-based, flow-oriented, and behavioral elements, feed the design task.

- Using design notation and design methods, design produces a data/class design, an architectural design, an interface design, and a component design.
- Each element of the analysis model provides information that is necessary to create the four design models
  - The data/class design transforms analysis classes into design classes along with the data structures required to implement the software.
  - The architectural design defines the relationship between major structural elements of the software; architectural styles and design patterns help achieve the requirements defined for the system.
  - The interface design describes how the software communicates with systems that interoperate with it and with humans that use it.
  - The component-level design transforms structural elements of the software architecture into a procedural description of software components.

### 3.1.1 Data Modelling

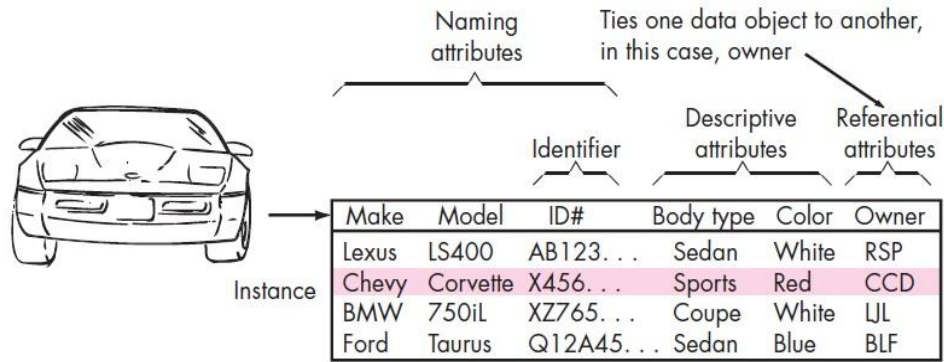
- If software requirements include the need to create, extend, or interface with a database or if complex data structures must be constructed and manipulated, then there is need to create a
- The data model is collection of 3 interrelated pieces or parts of information known as the data objects, the attribute & relationship.

#### 1. Data Objects:-

- A data object is a representation of composite information that must be understood by software.
- By composite information, I mean something that has a number of different properties or attributes.
- A data object encapsulates data only there is no reference within a data object to operations that act on the data
- E.g. **Car** is data object with properties name, color, & prize.

**2. Data attributes:** It define the properties of a data object and take on one of three different characteristics. They can be used to

- (1) Name an instance of the data object,
- (2) Describe the instance, or
- (3) Make reference to another instance in another table.



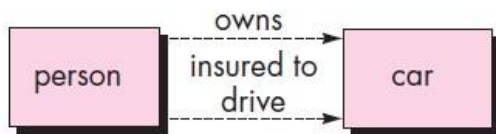
In above fig. **Car (Make, Model, ID, Body type, Color, Owner)**  
 Car is Data Object & remaining are data attributes.

### 3. Relationships

- Relationship indicates how data objects are related to one another.
- Data objects are connected to one another in different ways.
- Ex. Consider the two data objects, person and car



(a) A basic connection between data objects



(b) Relationships between data objects

- ☐ A person *owns* a car.
- ☐ A person *is insured to drive* a car.
- ☐ The relationship *owns* and *insured to drive* define the relevant connections between **person** and **car**.

### 4. Cardinality

- The *cardinality* represents "the number of occurrences of one object that can be related to the number of occurrences of another object."
- Cardinality can be expressed as:

#### 1. ONE TO ONE (1:1) :-

- ☐ It implies that ONE occurrences of object one is related to ONE occurrence of another object.
- ☐ E.g. Engg. College is having only one principal.

**2. ONE TO MANY (1:N):-** It implies that ONE occurrences of object one is related to MANY occurrence of another object.

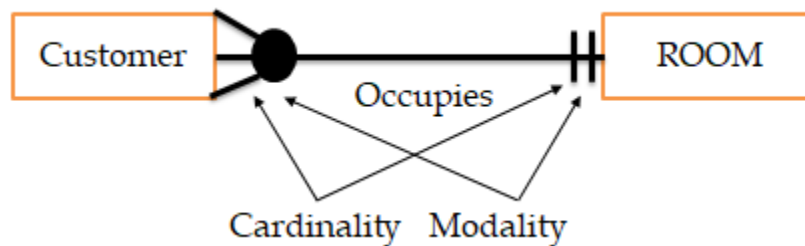
- ❑ E.g. one class may have many students.

### 3. MANY to MANY (N:N)

- ❑ It implies that many occurrences of object is related to MANY occurrence of another object.

### 5. Modality

- **The modality of a relationship is 0**
  - If an occurrence of the relationship is optional.
- **The modality of a relationship is 1**
  - If an occurrence of the relationship is mandatory (i.e. compulsory).
- Thus the modality specifies the minimum no. of relationship.

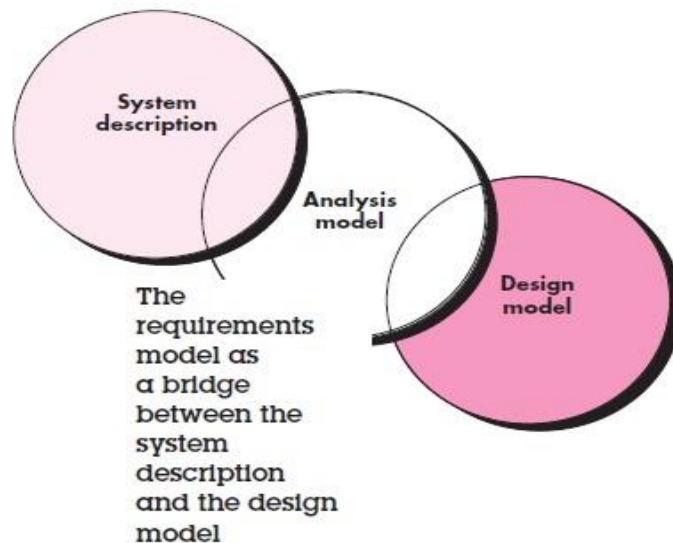


Here, the relationships shows that “exactly one room is occupied by zero or many customer”

## 3.2 Analysis Model

### 3.2.1 Analysis Modelling (AM):

- Analysis model make use of text & diagrammatic forms in a combination to represent requirements of data, functions & behavior for easy understanding of overall requirements of the s/w to be built.
- Basic aim of AM is to create the model that represents the information, functions, & behavior of the system to be built.



**Objectives of AM:**

- AM acts as bridge between “system description” & the “design model”
- System description describes overall system functionality of the system including s/w, h/w, databases, human interfaces & other system elements.
- And s/w design mainly focuses on application architecture, user interface & component level design.
- The requirements model must achieve three primary objectives:
  - (1) To describe what the customer requires,
  - (2) To establish a basis for the creation of a software design,
  - (3) To define a set of requirements that can be validated once the software is built.

**3.2.2 Elements of Analysis Model.**

- Each element of the requirements model presents the problem from a different point of view.
- **Scenario-based modeling: -**
  - They represent the system in user point of view.
  - It depicts how the user interacts with the system and the specific sequence of activities that occur as the software is used.
- **Class-based modeling:-**
  - They define the objects, attributes & relationships.
  - Class-based elements model the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships (some hierarchical) between the objects, and the collaborations that occur between the classes that are defined.
- **Behavioral modeling: -**
  - They show the states & impact of the events on these states.
  - Behavioral elements depict how external events change the state of the system or the classes that reside within it.
- **Flow-oriented modeling:-**
  - They provide necessary information that how data objects are transformed by processing the functions.
  - It represent the system as information transform, depicting how data objects are transformed as they flow through various system functions.

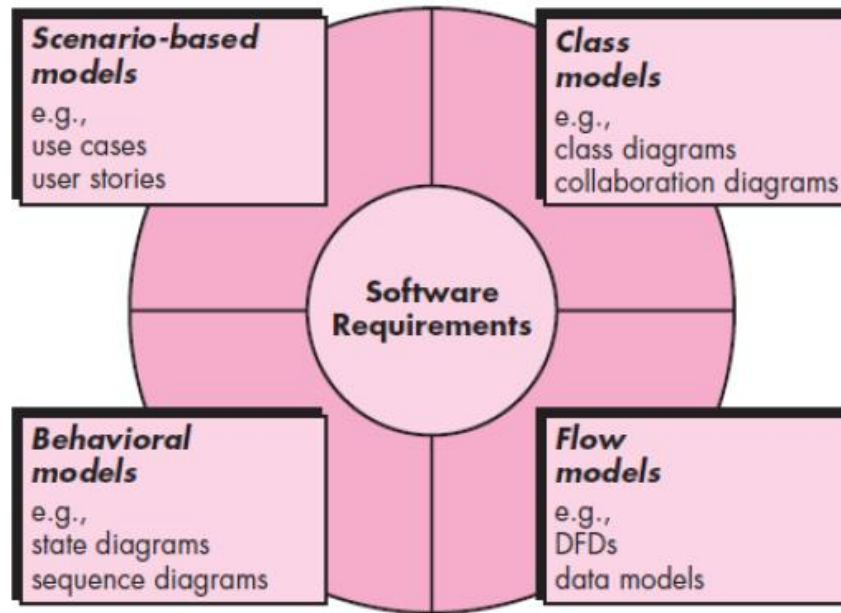


Fig. Four Elements of Analysis Model.

### 3.2.3 Analysis Modeling Approaches:

#### 1. Structured analysis

- Considers data and the processes that transform the data as separate entities.
- Data is modeled in terms of only attributes and relationships (but no operations).
- Processes are modeled to show the **1) input data, 2) the transformation that occurs on that data, and 3) the resulting output data.**

#### 2. Object-oriented analysis

- Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements.
- The basis of object oriented analysis is classes & member, objects & attributes, wholes & parts.
- The focus of object oriented analysis is objects.

### Importance of analysis modelling:

1. Designing gets easier to the designer
2. Better understanding of system can be accomplished
3. System feasibility can be determined
4. Defines data objects
5. Describes data attributes
6. Establishes data relationships
7. Identifies functions that transform data objects
8. Indicates different states of the system
9. Specifies events that cause the system to change state
10. Refines each model to represent lower levels of abstraction



11. Refines data objects

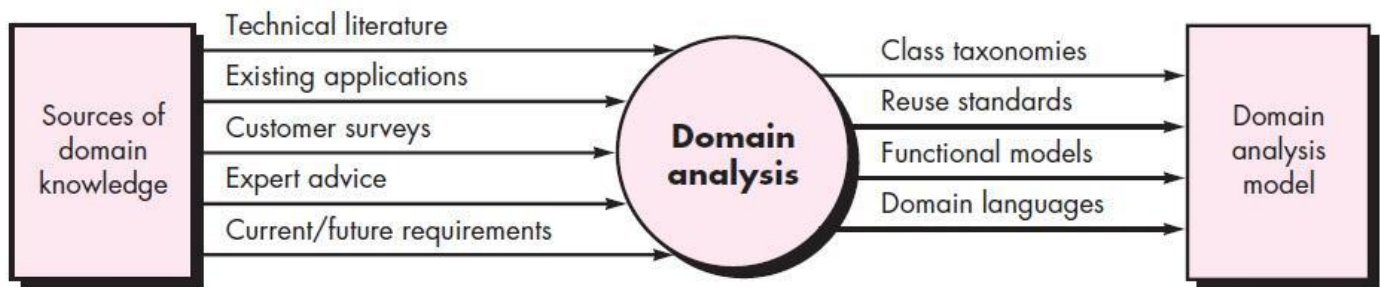
12. Relates a functional hierarchy represent behavior at different levels of detail

### Domain Analysis:

- Definition:-  
 “The identification, analysis, and specification of common, reusable capabilities within a specific application domain typically for reuse on multiple projects within that application Domain”
  - **Do this in terms of common objects, classes, subassemblies, and frameworks**

**FIGURE 6.2**

Input and output for domain analysis



- It takes i/p from source of knowledge, after domain analysis o/p is nothing but a domain analysis model.
- Software domain analysis can be defined as process of recognizing, analyzing & specifying of common requirements from a specific applications domain.
- It is a part of Analysis model that helps in building Analysis model by finding common requirements in the projects.
- A specific application domain, in terms of common objects, classes, subassemblies, and frameworks.
- E.g. application domain may be “Bus reservation system”

### The goal of domain analysis is straightforward:

- To find or create those analysis classes and/or analysis patterns that is broadly applicable so that they may be reused.
- Find out common requirements specification.
- To save the time.
- Reduce the repeated or duplicate work.

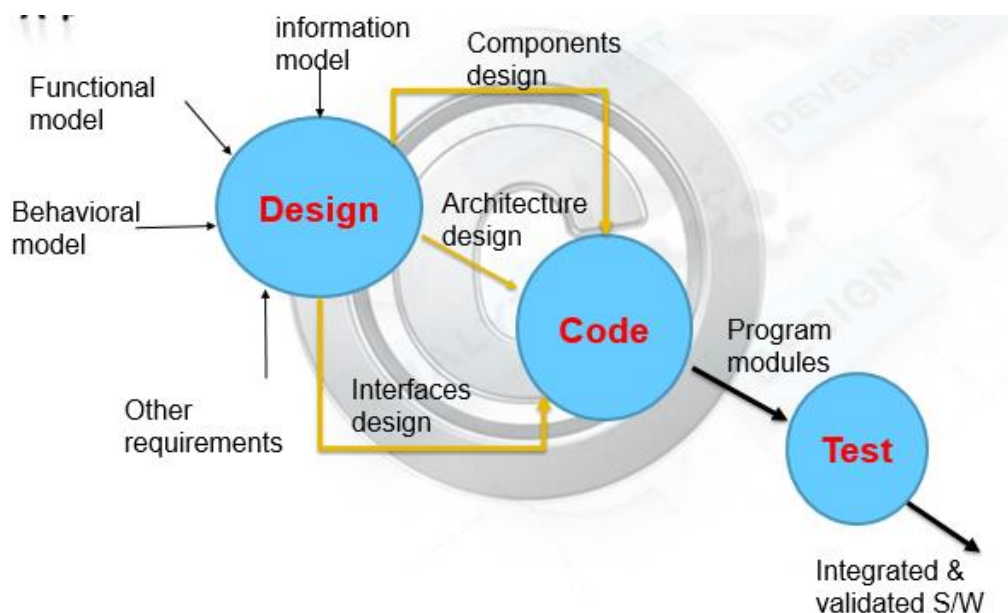
## 3.3 Design modelling concepts.

- **Design Modeling:** Software design is an iterative process through which requirements are translated into a blueprint for constructing the software.

- The design is representation at a high level of abstraction – data, functional, and behavioral requirements.
- As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction.
- **There are three characteristics that serve as a guide for the evaluation of a good design:**
  - The design must implement all of the explicit requirements contained in the requirements model, and it must accommodate all of the implicit requirements desired by stakeholders.
  - The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
  - The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

### 3.3.1 Design Modelling (DM): Purpose of Design

- Design is where customer requirements, business needs, and technical considerations all come together in the formulation of a product or system
- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software
- Design begins at a high level of abstraction that can be directly traced back to the data, functional, and behavioural requirements
- The design model provides detail about the software data structures, architecture, interfaces, and components
- The design model can be assessed for quality and be improved before code is generated and tests are conducted.
- It takes help of different diagrams & create complete picture of s/w before we actually develop it.
- **Software design phases are:-**





### 3.3.2 Design Concepts

- **A set of fundamental software design concepts has evolved over the history of software engineering.**
  1. Abstraction
  2. Architecture
  3. Patterns
  4. Modularity
  5. Information hiding
  6. Functional independence
  7. Refinement
  8. Refactoring
  9. Design classes

#### 1. Abstraction

- “Abstraction” can be defined as a view of a problem that extract the essential information relevant to a particular purpose & ignores the remainder of the information.
- It permits one to concentrate on a problem at some level of generalization.
- **Types of abstraction**
  1. **Procedural abstraction** – a sequence of instructions that have a specific and limited function
  2. **Data abstraction** – it is a named collection of data that describes a data object
  3. **Control abstraction** –implies a program control mechanism without specifying internal details.

#### 2. Architecture

- The overall structure of the software and the ways in which the structure provides conceptual integrity for a system
- Consists of components, connectors, and the relationship between them.

#### 3. Patterns

- A design structure that solves a particular design problem within a specific context.
- It provides a description that enables a designer to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns.

#### 4. Modularity

- It is represented by s/w architecture & design patterns.
- The meaning is s/w is divided into uniquely named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle).
- Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity.
- If the program is not modular that time program complicated & difficult to understand.

**5. Information hiding**

- The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules.
- This enforces access constraints to both procedural (i.e., implementation) detail and local data structures.

**6. Functional independence**

- The concept of functional independence is an outcome of modularity, abstraction & information hiding.
- Independent modules are easier to maintain & test.
- Independent modules are easier to develop because function may be compartmentalized & interfaces are simplified.

**7. Stepwise refinement**

- It is actually a process of elaboration.
- Development of a program by successively refining levels of procedure detail.
- Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details.

**8. Refactoring**

- A reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behaviour.
- Removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures.

**9. Design classes**





- Refines the analysis classes by providing design detail that will enable the classes to be implemented.
- Creates a new set of design classes that implement a software infrastructure to support the business solution.

**3.4 Design Notations:****3.4.1 DFD (Data flow diagrams)**

- DFD is also called as “bubble chart”
- This is a graphical technique that represents information flow & transformers those are applied when data moves from i/p to o/p.
- The DFD takes an input-process-output view of a system. That is, data objects flow into the software, are transformed by processing elements, and resultant data objects flow out of the software.
- DFD may be partitioned into different levels to show detailed information flow.

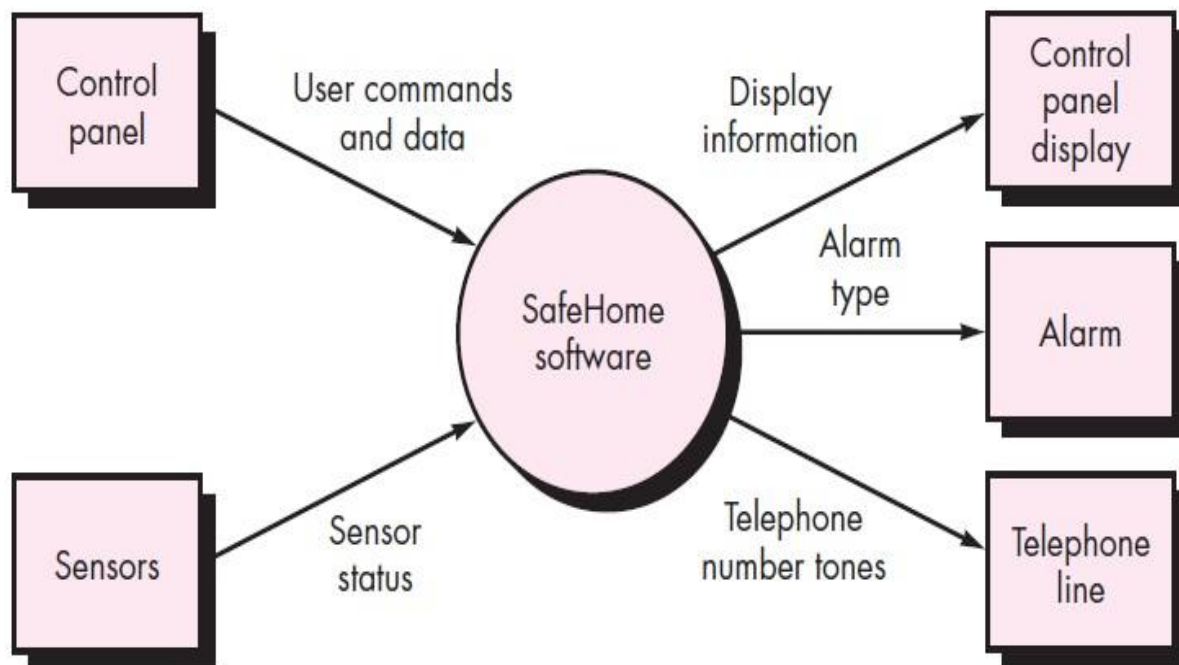
- E.g. level0, level1, level2 etc.

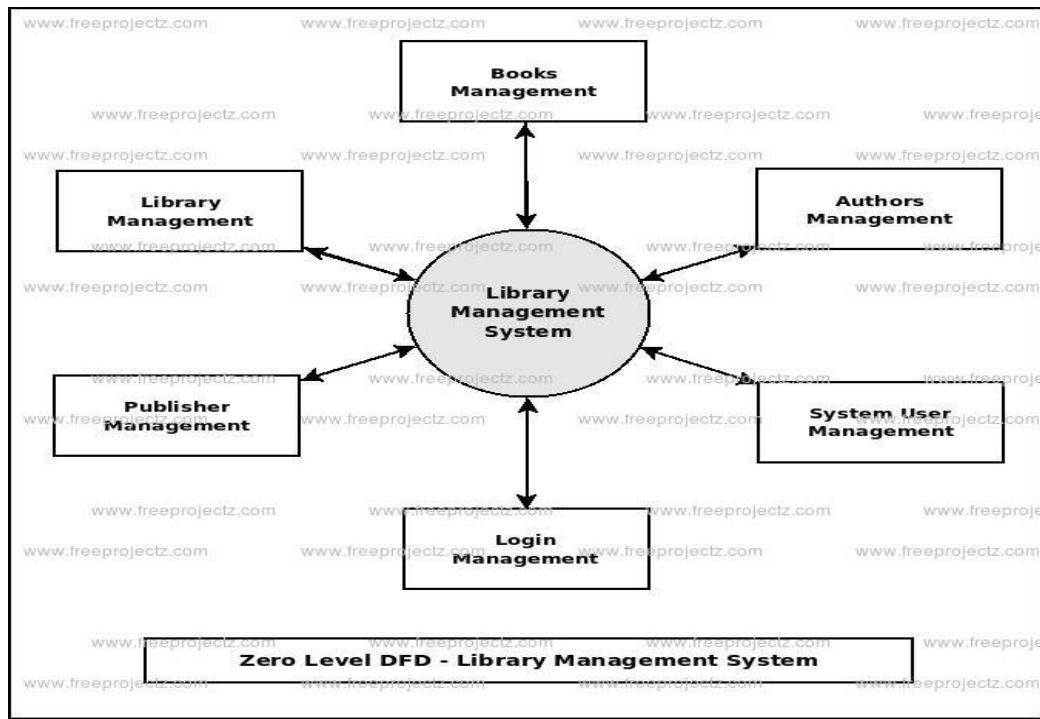
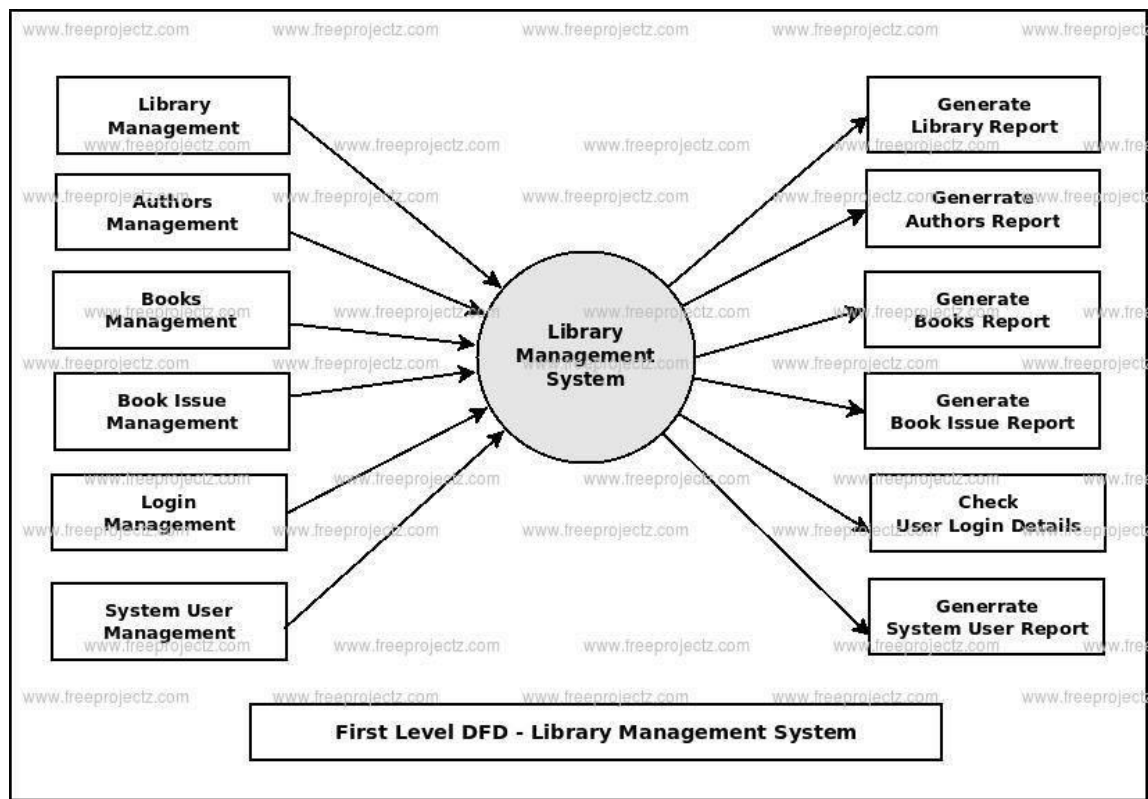
### DFD Notations

Sr. No.	Symbol	Description
1	 (Circle)	Process or transform that acts on data or control & changes it.
2	 (Arrow)	Arrow should be labeled they show direction of either input or output
3	 (double line)	A double line can be used to represent a data store
4	 (Rectangle)	Rectangular boxes, entities (things, persons, places etc.)

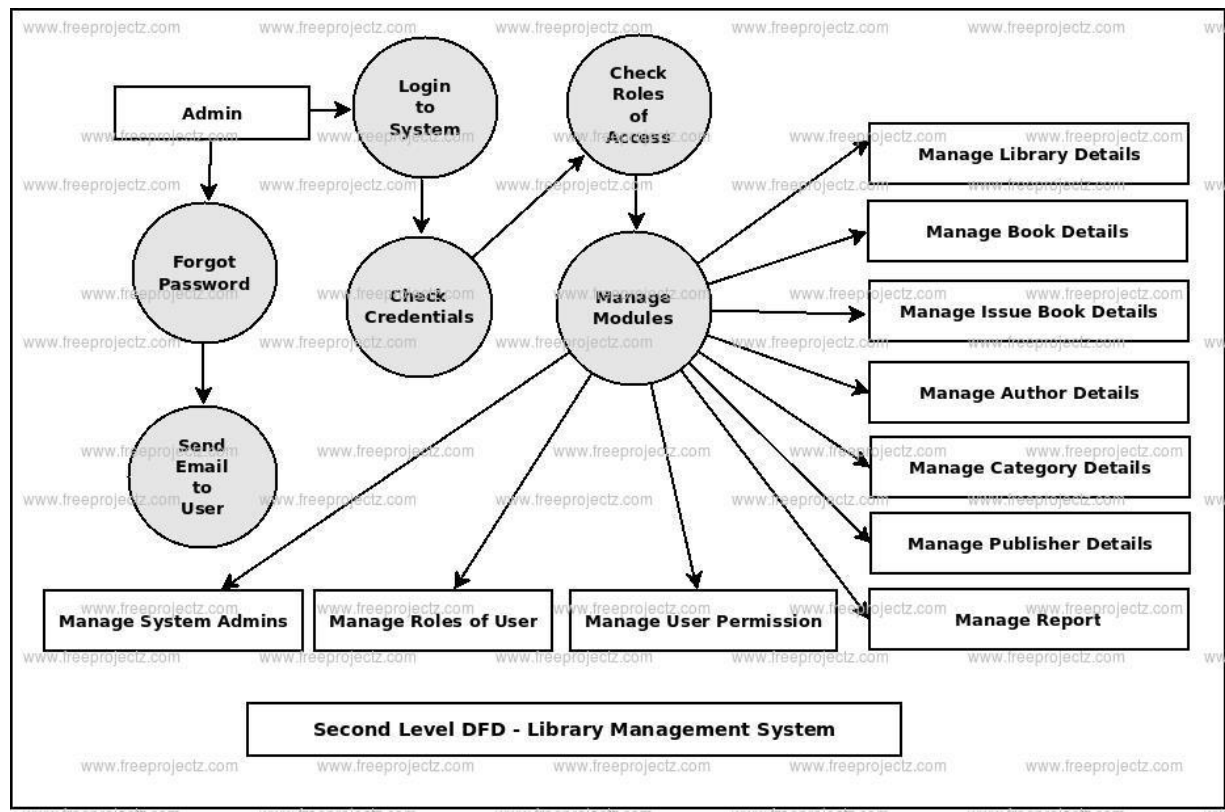
### Examples of DFD

- Example1: Safe Home software

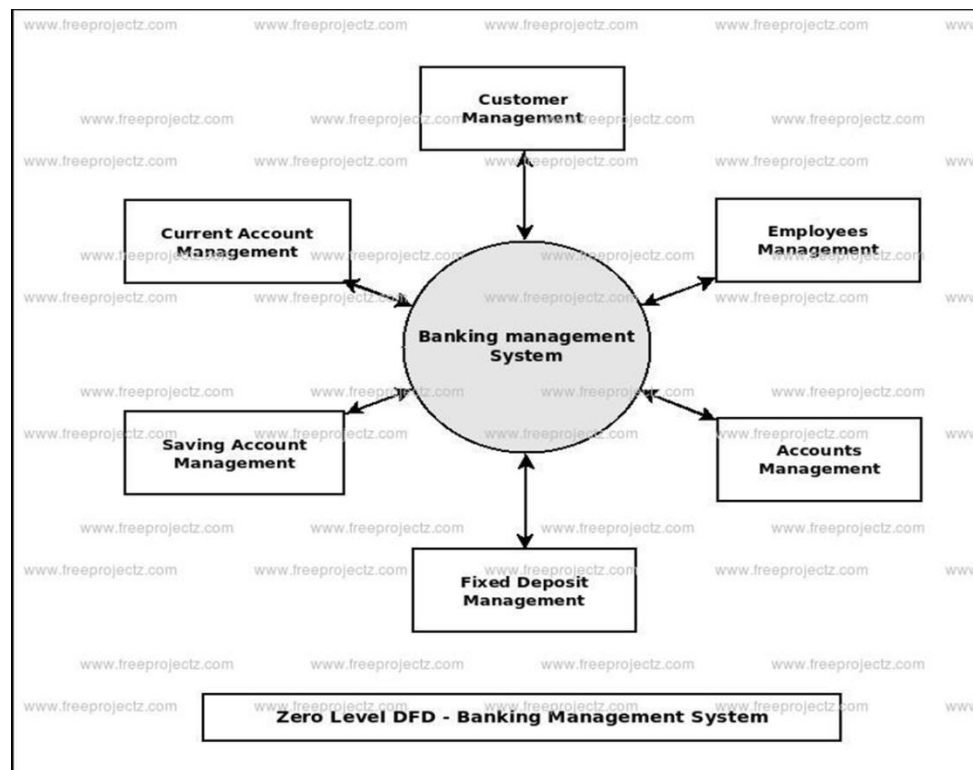


**Example2: Library Management System****• DFD Level 0:-****• DFD Level 1:**

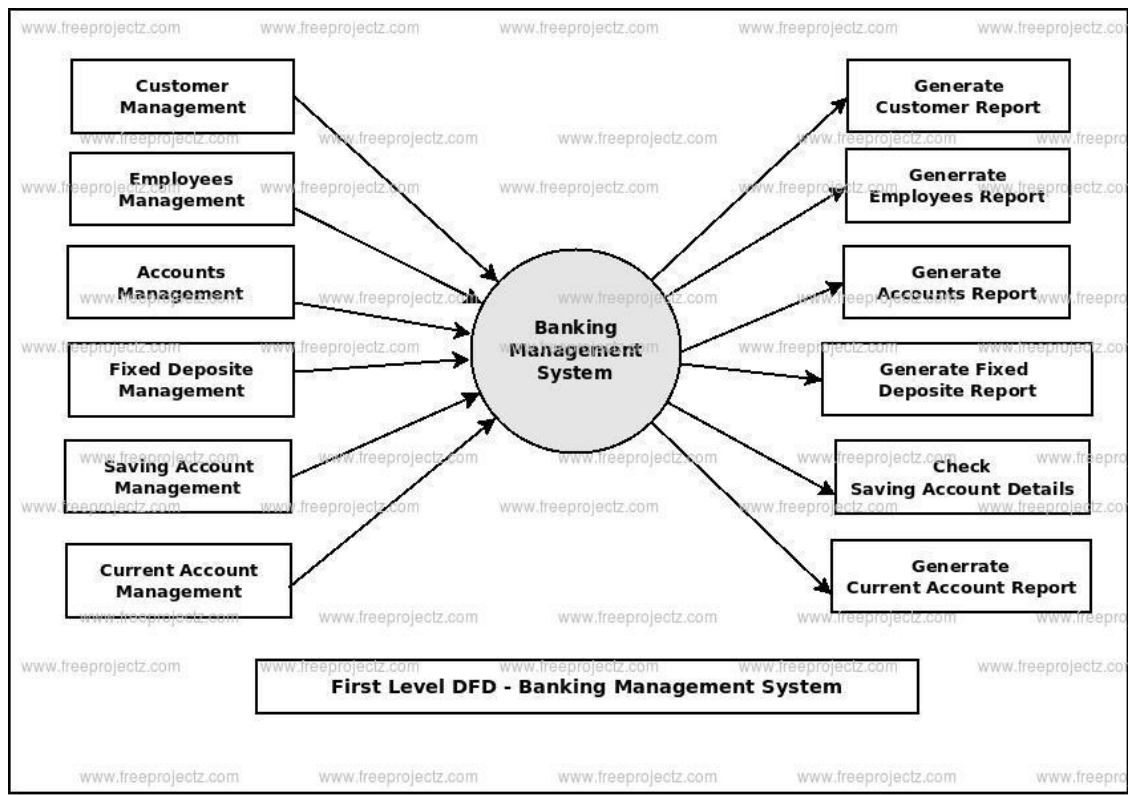
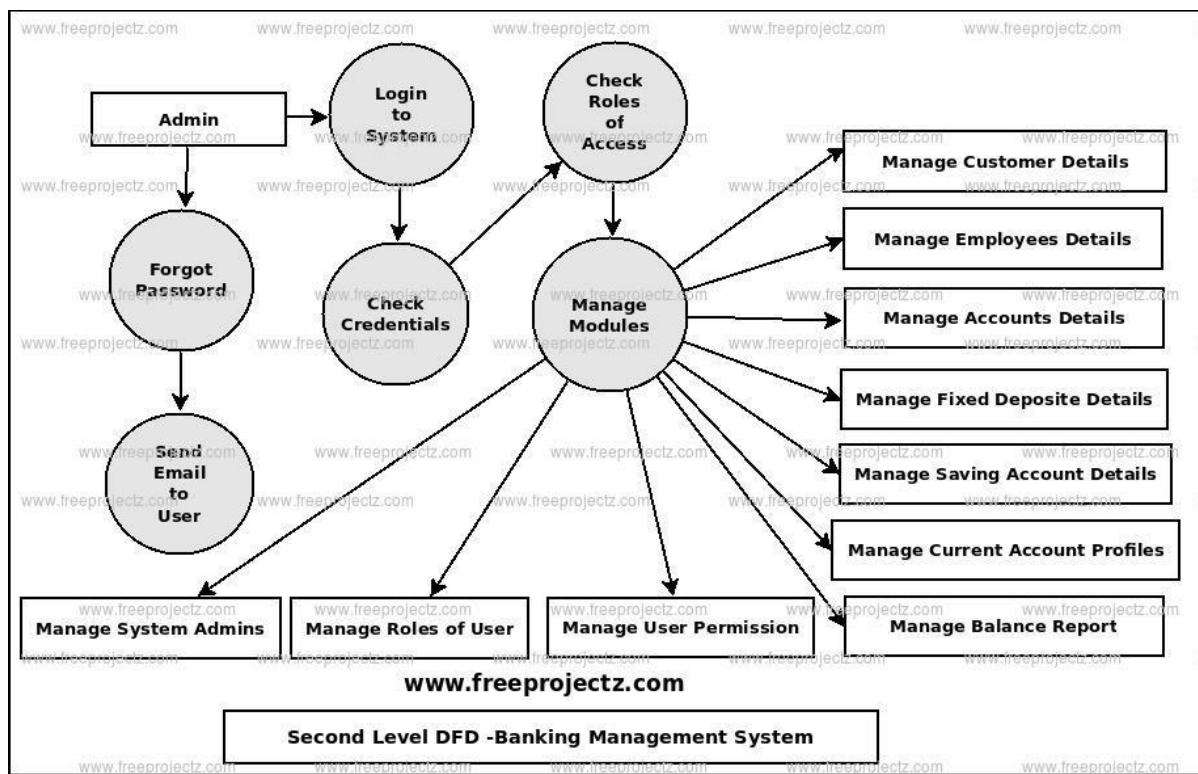
- DFD Level 2:



### Example3: Bank Management System DFD Level 0:-





**DFD Level 1:****DFD Level 2:-**



Q. Draw proper labelled “LEVEL 1 Data Flow Diagram” (DFD) for student attendance system(S-19)

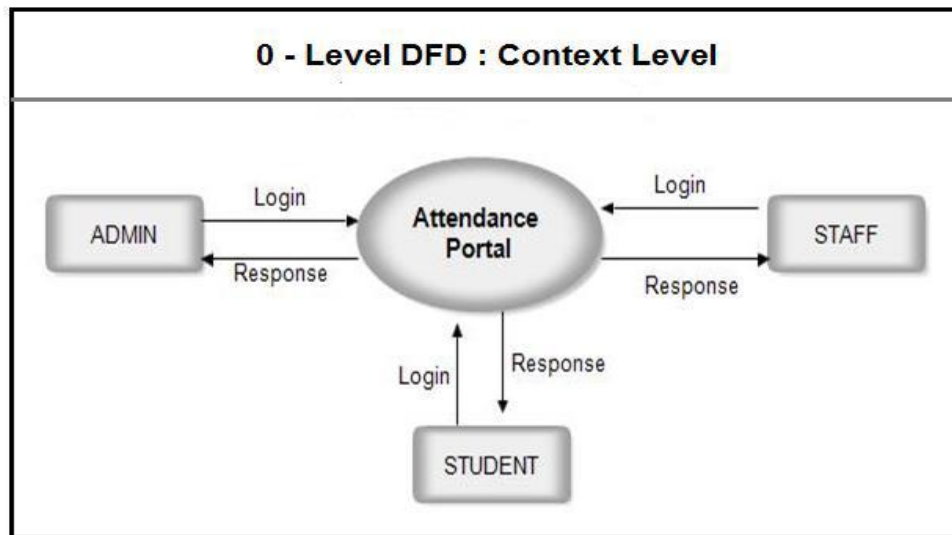


Fig. DFD Level 0

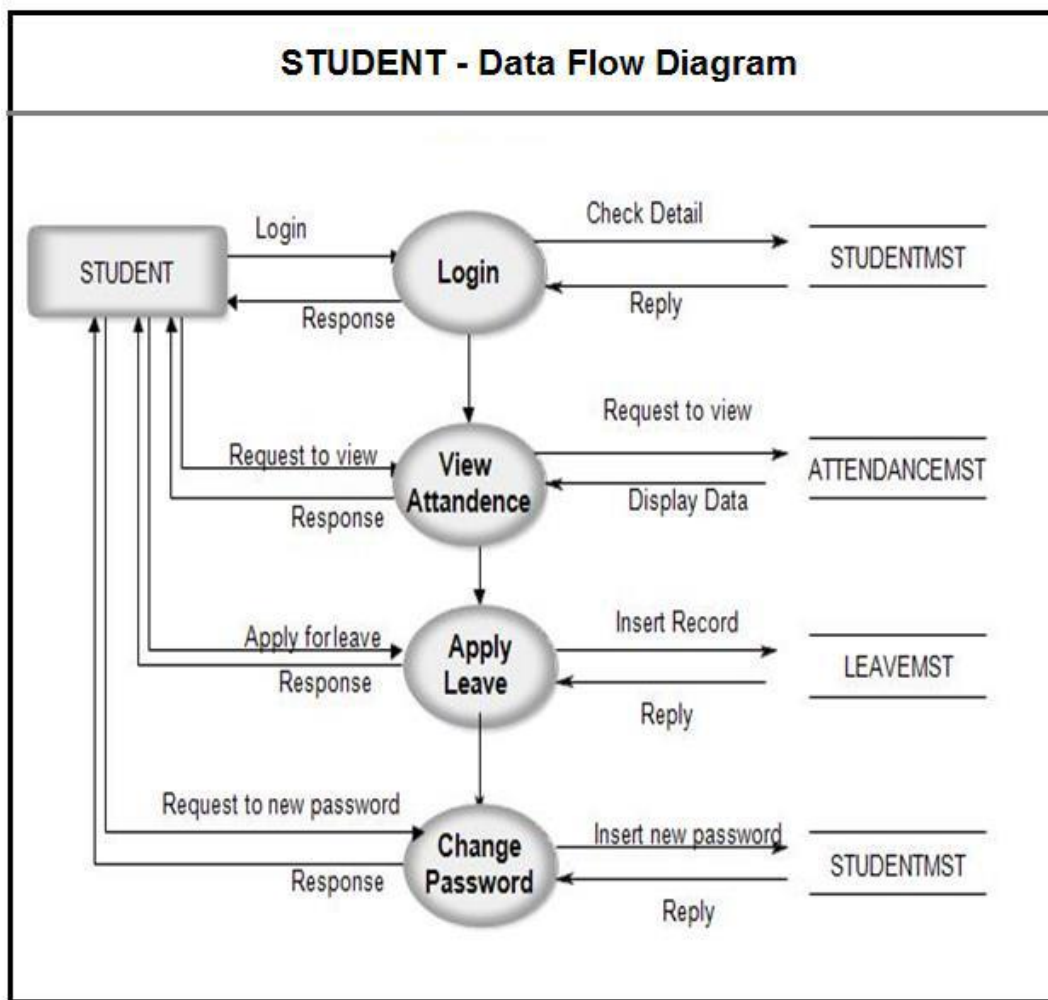


Fig. DFD Level 1 for Student

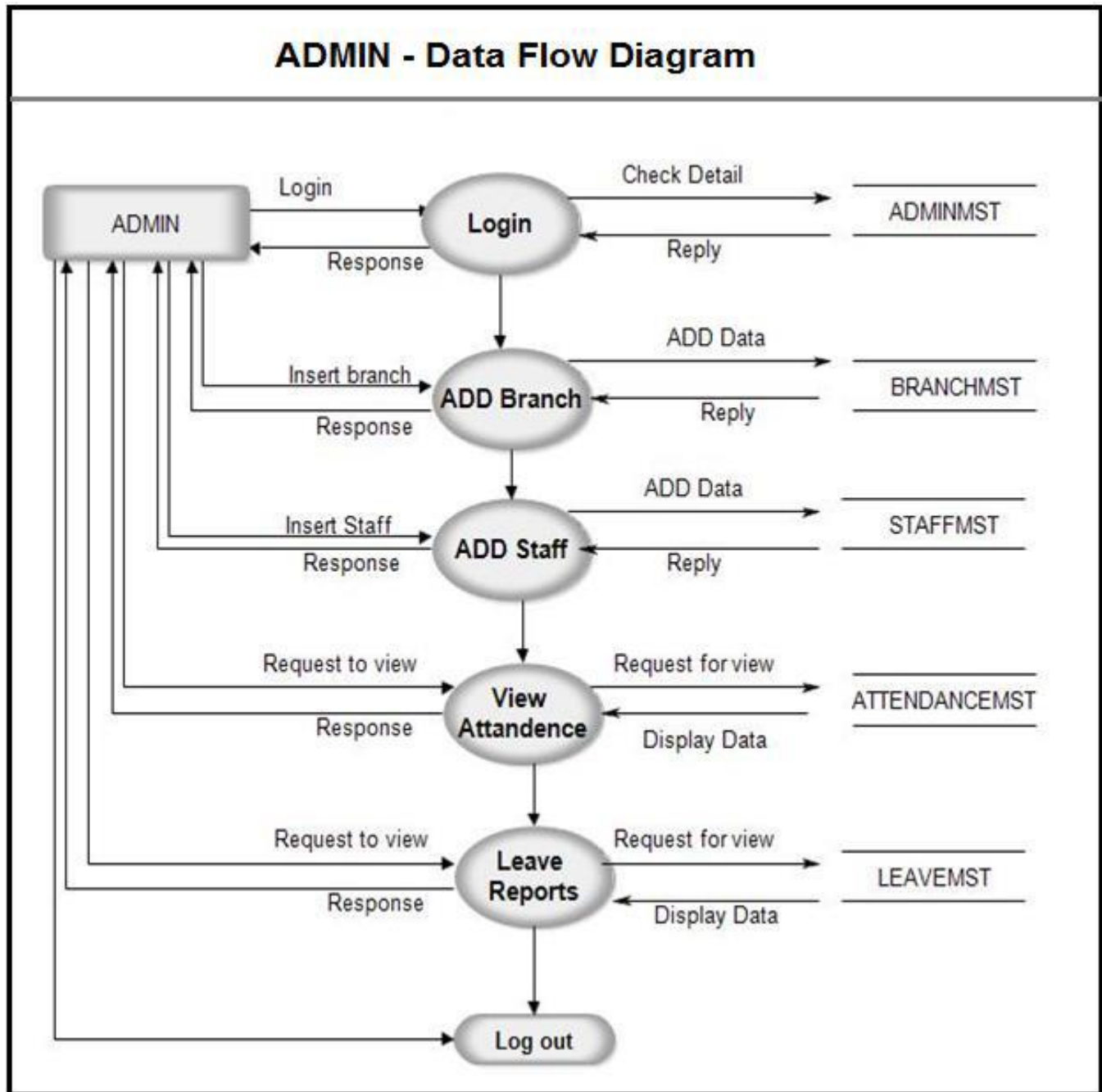


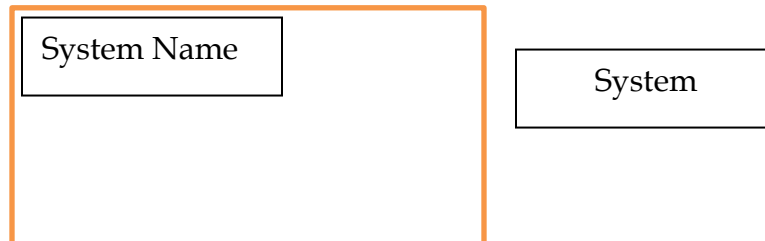
Fig. DFD Level 1 for Admin

Q. Draw and explain Level 1 DFD for railway reservation system. (Winter 2019)

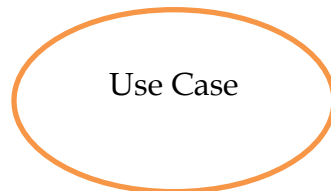
### 3.4.2 Use Case Diagram

#### Basic Use Case Diagram Symbols and Notations

**1. System:** Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



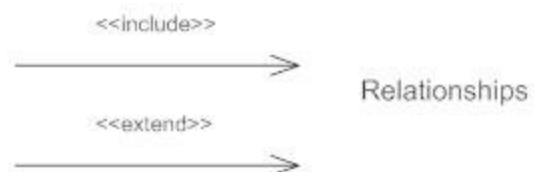
**2. Use Case:** Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.

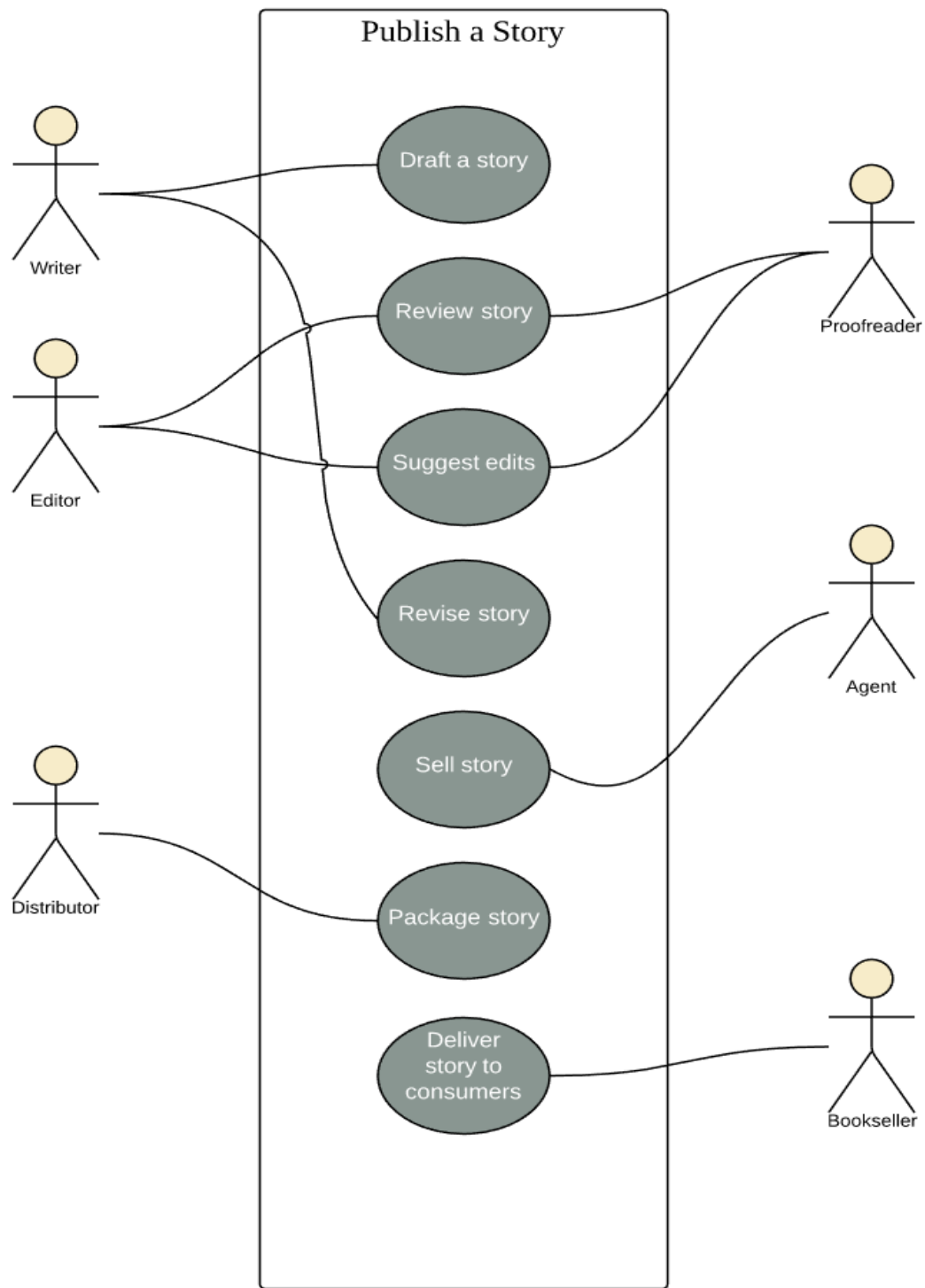


**3. Actors:** Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

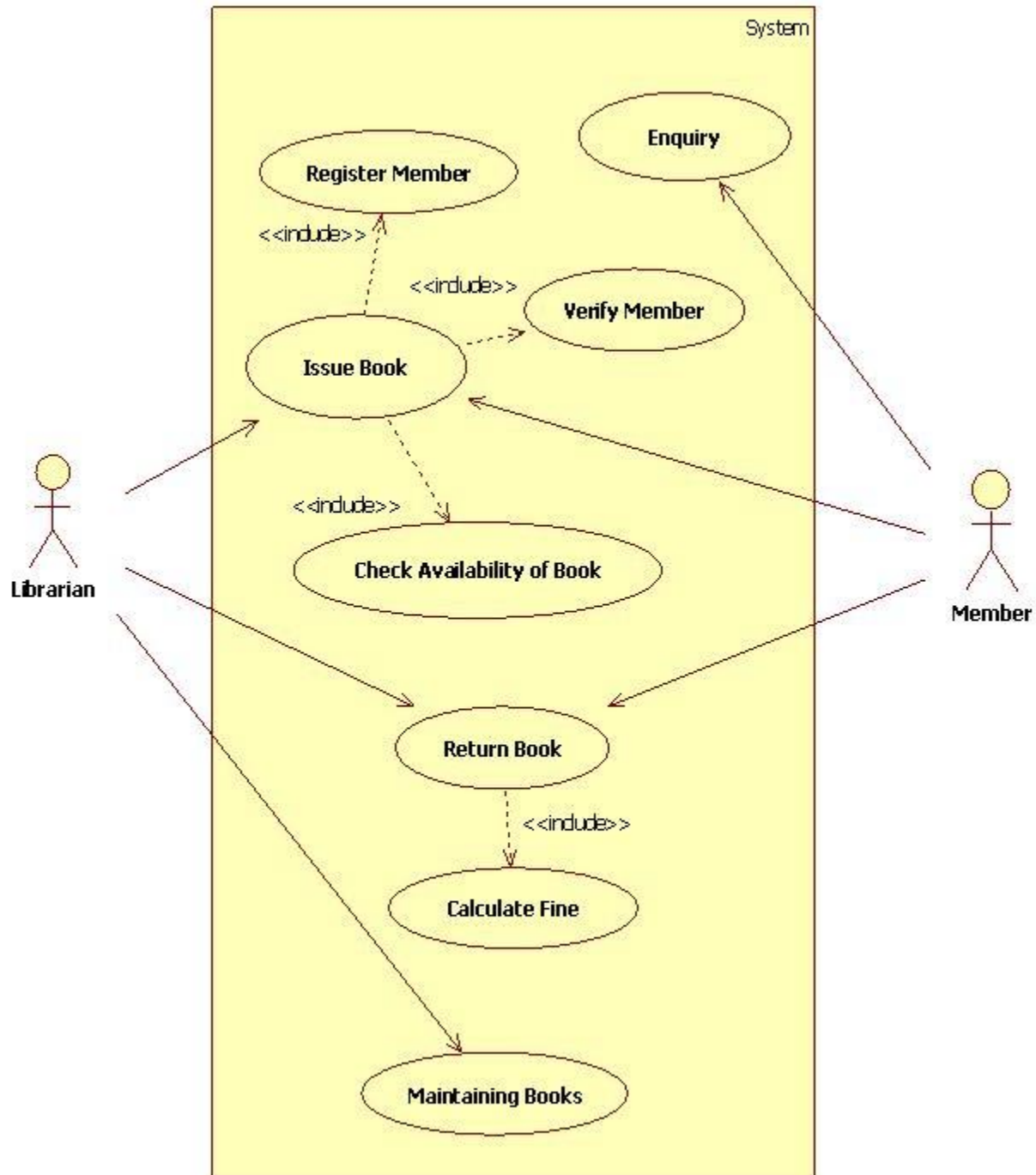


**4. Relationships:** - Illustrate relationships between an actor and a use case with a simple line.



Examples:**Ex1:- Book publishing use case diagram example**

Q. Sketch a use case diagram for library management system with minimum four use cases and two actors.

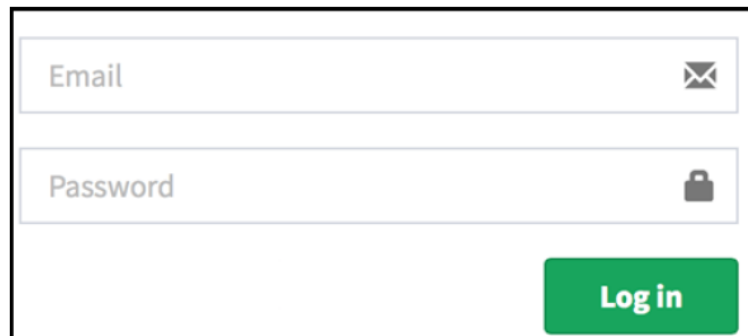


### 3.4.2 Structured flowchart

- Flow chart gives pictorial representation of all elements of structured programming. Like the activity diagram it allows you to represent sequence, condition, and repetition all elements of structured programming.
- A box is used to indicate a processing step. A diamond represents a logical condition, and arrows show the flow of control.
- Structured flowchart consist of following structured constructs:
  - The *sequence* is represented as two processing boxes connected by a line (arrow) of control.
  - *Condition*, also called *if-then-else*, is depicted as a decision diamond that, if true, causes *then-part* processing to occur, and if false, invokes *else-part* processing.
  - *Repetition* is represented using two slightly different forms. The *do while* tests a condition and executes a loop task repetitively as long as the condition holds true. A *repeat until* executes the loop task first and then tests a condition and repeats the task until the condition fails.
  - The selection (or select-case) construct shown in the figure is actually an extension of the if-then-else. A parameter is tested by successive decisions until a true condition occurs and a case part processing path is executed.

### 3.4.3 Decision table

- Decision table is a software testing technique used to test system behavior for different input combinations.
- This is a systematic approach where the different input combinations and their corresponding system behavior (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.
- Example1:- **Decision Base Table for Login Screen**



The image shows a login form with two input fields. The first field is labeled 'Email' and has an envelope icon on the right. The second field is labeled 'Password' and has a lock icon on the right. Below these fields is a green button with the text 'Log in' in white.

- 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.



<u>Conditions</u>	<u>Rule 1</u>	<u>Rule 2</u>	<u>Rule 3</u>	<u>Rule 4</u>
<u>Username(T/F)</u>	<u>F</u>	<u>T</u>	<u>F</u>	<u>T</u>
<u>Password(T/F)</u>	<u>F</u>	<u>F</u>	<u>T</u>	<u>T</u>
<u>Output(E/H)</u>	<u>E</u>	<u>E</u>	<u>E</u>	<u>H</u>

T - Correct username/password  
 F - Wrong username/password  
 E - Error message is displayed  
 H - Home screen is displayed

### Interpretation:

**Case1:** Username and password both were wrong. The user is shown an error message.

**Case2:** Username was correct, but the password was wrong. The user is shown an error message.

**Case3:** 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.

## 3.5 Testing: Introduction

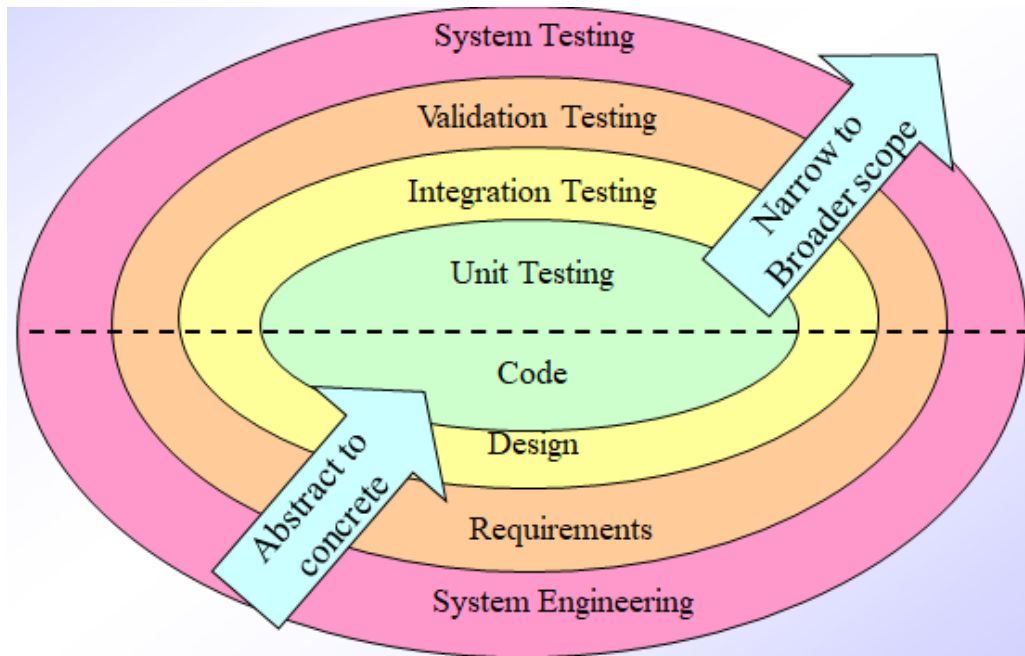
- Testing is an activity that is used to discover errors and correct them, so that we are able to create a defect free product for the customer.
- Software testing is a process of executing a program or software with the intent of finding an error.
- Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- **Definition of software Testing:**
  - **The process of executing a program (application) with the intent of finding errors.”**  
Or
  - **“Testing is the process to detect (find) the defects (error) & minimize the risk associated in Software”**

### Objectives of s/w testing

- Find out error & fix them afterwards.
- Good testing has highly probability to find out error which could not be yet discovered.
- Good test is able to find out undiscovered error.
- To fulfill user/ customer requirement regarding s/w product.
- To enforce standard while developing s/w product.

- To support s/w quality assurance.
- Check operability & flexibility of s/w.
- Check reliability of s/w product.

### 3.5.1 Level of testing

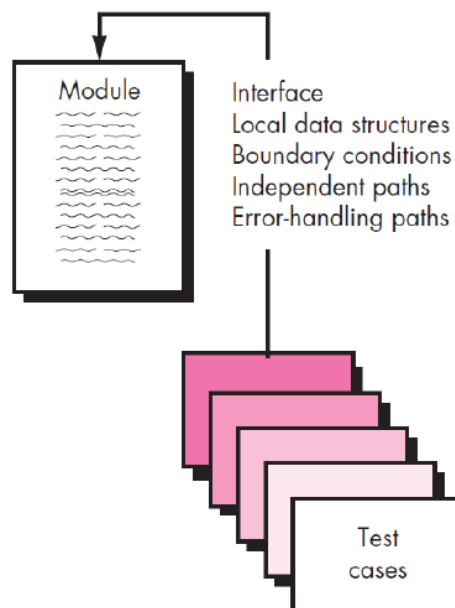


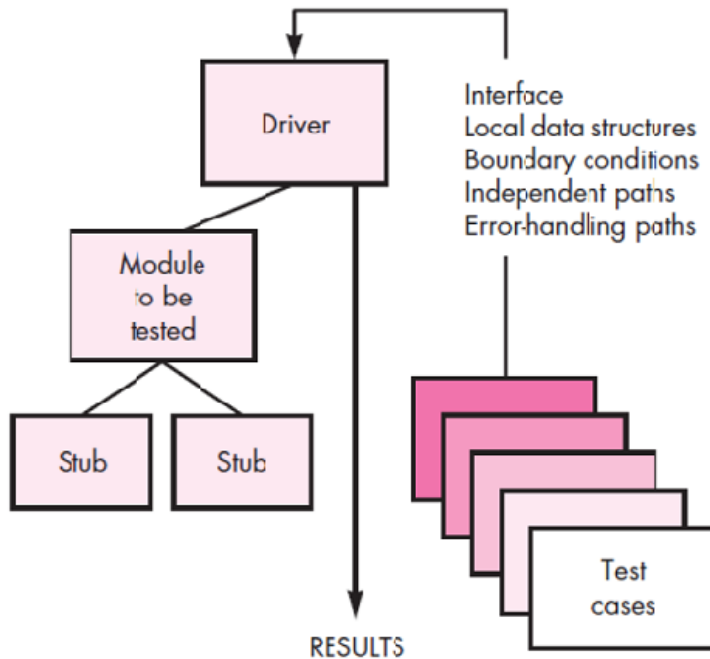
- **Unit testing**
  - Exercises specific paths in a component's control structure to ensure complete coverage and maximum error detection
  - Components are then assembled and integrated.
- **Integration testing**
  - Focuses on inputs and outputs, and how well the components fit together and work together.
- **Validation testing**
  - Provides final assurance that the software meets all functional, behavioral, and performance requirements.
- **System testing**
  - Verifies that all system elements (software, hardware, people, databases) mesh properly and that overall system function and performance is achieved.

### 3.5.2 Unit testing:-

- Focuses testing on the function or software module.
- Concentrates on the internal processing logic and data structures.
- Is simplified when a module is designed with high cohesion.

- Reduces the number of test cases.
  - Allows errors to be more easily predicted and uncovered.
- Concentrates on critical modules and those with high cyclomatic complexity when testing resources are limited.
- In unit testing it is possible that the output produced by one unit become output for another unit.
- If in incorrect output produced by one unit is provided as input to second unit then it also produce wrong output.
- If this process is not corrected the entire software may produce unexpected output.
- To avoid this all the unit in the software is tested independently using unit testing.
- **Targets for Unit Test Cases**
  - **Module interface**
    - Ensure that information flows properly into and out of the module.
  - **Local data structures**
    - Ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution.
  - **Boundary conditions**
    - Ensure that the module operates properly at boundary values established to limit or restrict processing.
  - **Independent paths (basis paths)**
    - Paths are exercised to ensure that all statements in a module have been executed at least once.
  - **Error handling paths**
    - Ensure that the algorithms respond correctly to specific error conditions.





- **Driver(main program)**
  - A simple **main program** that accepts test case data, passes such data to the component being tested, and prints the returned results.
- **Stubs**
  - It acts as the sub modules called by the test modules.
  - It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing

### 3.5.3 Black box testing & White box testing

#### White-box Testing:-

- It is a way of testing the software in which the tester has knowledge about the internal structure the code or the program of the software.
- It is also called structural testing, clear box testing, code-based testing, or glass box test
- Testing is best suited for a lower level of testing like Unit Testing, Integration testing.
- It is mostly done by software developers.
- Knowledge of implementation is required.
- It is structural test of the software.
- Testing can start after preparing for Detail design document.
- **Various types of method applicable for white box testing are:-**

#### **1. Basis Path Testing**

- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

#### **2. Independent Program Paths**

- Defined as a path through the program from the start node until the end node that introduces at least one new set of processing statements or a new condition (i.e., new nodes).

#### **3. Cyclomatic Complexity**

- Provides a quantitative measure of the logical complexity of a program.

**4. Loop Testing – General**

- A white-box testing technique that focuses exclusively on the validity of loop constructs
- Four different classes of loops exist
  - Simple loops
  - Nested loops
  - Concatenated loops
  - Unstructured loops
- Testing occurs by varying the loop boundary values
- Examples:

```
for (i = 0; i < MAX_INDEX; i++)
```

```
while (currentTemp >= MINIMUM_TEMPERATURE)
```

**5. Control structure testing:-**

- It is used to test various control structure present in the program.

**6. Mutation testing.**

- This type of testing where errors are purposely inserted into the program to verify whether the existing test cases are able to detect the error.

**Black-box Testing:-**

- Complements white-box testing by uncovering different classes of errors
- It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.
- It also known as data-driven, box testing, data-, and functional testing.
  - Not concerned with internal logical structure of the software
- Focuses on the functional requirements and the information domain of the software.
- Used during the later stages of testing after white box testing has been performed
- It is mostly done by software testers.
- No knowledge of implementation is needed.
- Incorrect or missing functions
- Interface errors
- Errors in data structures or external data base access
- Behavior or performance errors
- Initialization and termination errors

**Techniques used:**

**1. Equivalence partitioning:** Equivalence partitioning divides input values into valid and invalid partitions and selecting corresponding values from each partition of the test data.

**2. Boundary value analysis:**

- Checks boundaries for input values.

White box Testing	Black box testing
To test the internal structure of s/w.	To test the functionality of s/w.
Testing is application with coding & programming knowledge.	Testing is application without coding & programming knowledge.
It is done by developer	It is done by separate testing team.
We cannot test the performance of the application	We can test the performance of the application
Performed in the early stages of testing	Performed in the later stages of testing
The most time consuming type of testing	The least time consuming type of testing
Data domains & internal boundaries can be better tested	This can only be done by trial & error method.
Address flow & control structure of a program	Address validity, behavior & performance of software.

### 3.6 Test Documentation: Test case template, Test plan, Defect Report, Test Summary report.

#### 3.6.1 Test Plan

- Test plan tells “What to Test”.
- It is document that contains the strategy that will be used to verify as well as make sure that a product or s/m meets its design specification & other requirements.
- Test plan describes how testing would be accomplished.
- It is document that specifies the purpose, scope & method of s/w testing.
- Testing engineer includes significant input for the s/w in a test plan.
- Test plan format varies organization to organization.
- Important aspect is that contain 3 important element like test coverage, test methods & test responsibilities.
- **Components of Test Plan:-**
  - Responsibilities, assumptions, Test, communication, risk analysis, defect reporting, environment.



### 3.6.2 Test Cases

- Test case can be defined as set of “actions”, “input data”, & “expected results.”
- It is a set of conditions under which a tester will determine whether a functionality of s/w is working correctly or not.
- **Test cases can be broadly classified into 3 categories as follows:**
  1. Formal test cases: - it is a test where i/p is known & o/p is an Expected, which is worked out before the test is executed.
  2. Informal test cases:-it mainly used in scenario testing which are generally of multiple steps & not written down like formal tests.
  3. Typical written test case contains the test case data documented in detail.

### 3.6.3 Test case template

(Name of the product)
<ul style="list-style-type: none"><li>• <b>Prepared by:</b> (Name of Prepares)</li></ul>
<b>Date:</b> __
<b>Test Plan Template Table of Contents (TOC)</b>
<b>1. Introduction</b>
1.1. Test Plan Objectives
<b>2. Scope</b>
2.1. Data Entry
2.2. Reports File Transfer
2.3. File Transfer
2.4. Security
<b>3. Test Strategy</b>
3.1. System Test
3.2. Performance Test
3.3. Security Test
3.4. Automated Test
3.5. Stress and Volume Test
3.6. Recovery Test
3.7. Documentation Test
3.8. Beta Test
3.9. User Acceptance Test
<b>4. Environment Requirements</b>
4.1. Data Entry Workstations
4.2 Main Frame
<b>5. Test Schedule</b>
<b>6. Control Procedures</b>

6.1 Reviews
6.2 Bug Review Meetings
6.3 Change Request
6.4 Defect Reporting
<b>7. Functions To Be Tested</b>
<b>8. Resources and Responsibilities</b>
8.1. Resources
8.2. Responsibilities
<b>9. Deliverables</b>
<b>10. Suspension / Exit Criteria</b>
<b>11. Resumption Criteria</b>
<b>12. Dependencies</b>
12.1 Personnel Dependencies
12.2 Software Dependencies
12.3 Hardware Dependencies
12.3 Test Data & Database
<b>13. Risks</b>
13.1. Schedule
13.2. Technical
13.3. Management
13.4. Personnel
13.5 Requirements
<b>14. Tools</b>
<b>15. Documentation</b>
<b>16. Approvals</b>

### **3.6.4 Defect report in software testing**

- DEFECT REPORT is a document that identifies and describes a defect detected by a tester.
- The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

### **Bug Report**

While reporting the bug to developer, your Bug Report should contain the following information

- **Defect\_ID** - Unique identification number for the defect.
- **Defect Description** - Detailed description of the Defect including information about the module in which Defect was found.
- **Version** - Version of the application in which defect was found.
- **Steps** - Detailed steps along with screenshots with which the developer can reproduce the defects.
- **Date Raised** - Date when the defect is raised

- **Reference**- where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect
- **Detected By** - Name/ID of the tester who raised the defect
- **Status** - Status of the defect , more on this later
- **Fixed by** - Name/ID of the developer who fixed it
- **Date Closed** - Date when the defect is closed
- **Severity** which describes the impact of the defect on the application
- **Priority** which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively.

### 3.6.5 Test Summary Report

- Offers information about the various activities performed during the testing process.
- Summarizes the results of all testing efforts.
- Prepared by the test leads or test managers.
- Test managers prepare this document at the end of the testing project.

Test Report	
Project Name	Guru99 Bank
Test Type	Performance Test
Pass	250
Fail	30
Not Executed	30
Total	310

**Q. Identify and enlist requirement for given modules of employee management software (Summer-2019, 6 Marks.)**

**Ans:-**

- Employee detail
  - Employee salary
  - Employee performance
- This is with perspective of employee management software. Requirements for following modules will be as
    - Employee details
      - Getting information about the customer
      - Updation of employee details (department, change of address, emp\_code etc)
      - Assignment of tasks, duties and responsibilities.
      - Recording of employee attendance.
    - Employee salary

- a. Salary calculation
  - 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.

-----End of Chapter-----