System Analysis and System Design

System Analysis and System Design are two stages of the software development life cycle. System Analysis is a process of collecting and analyzing the requirements of the system whereas System Design is a process of creating a design for the system to meet the requirements. Both are important stages as it helps to create an effective system with all the features and functions.

What is System Analysis?

System Analysis is a process of understanding the system requirements and its environment. It is one of the initial stages of the software development life cycle. System analysis is the process of breaking the system down into its individual components and understanding how each component interacts with the other components to accomplish the system's overall goal. In this process, the analyst collects the requirements of the system and documents them.

Characteristics

- It is the study of the existing system to identify the problem areas.
- It is a process of understanding the system requirements and its environment.
- It involves gathering and understanding the user's requirements.
- It involves analyzing the system in terms of its current and future needs.

Advantages

- It helps to identify the problems and their causes.
- It helps to understand the functional and non-functional requirements of the system.
- It helps to develop better solutions.
- It helps identify the areas of improvement.

Limitations

- It can be time-consuming.
- It can be costly.
- It can be difficult to get accurate information.

What is System Design?

System Design is the process of creating a design for the system to meet the requirements. System design is the process of designing the architecture, components, modules, interfaces, and data for a system to satisfy the specified requirements. It involves the design of the system architecture, components, modules, interfaces, and data.

Characteristics

- It is the process of creating a design for the system.
- It involves the design of the system architecture, components, modules, interfaces, and data.
- It involves identifying the modules and components of the system.
- It involves creating the user interface and database design.

Advantages

- It helps to create an efficient system.
- It helps identify the areas of improvement.

- It helps to reduce the development cost.
- It helps to create a better user experience.

Limitations

- It can be time-consuming.
- It can be costly.
- It can be difficult to get accurate information.

Differences between System Analysis and System Design

Factors	System Analysis	System Design
Purpose	System Analysis is the process of gathering and analyzing information to assess the suitability of a current system and to determine the requirements of a new system.	System Design is the process of specifying elements of a system such as modules, architecture, components, and their interfaces.
Approach	System Analysis is a top-down approach where the analyst looks at the big picture first and then delves into the details.	System Design is a bottom-up approach where the analyst starts with the details and moves up to the big picture.
Scope	System Analysis focuses on the needs of the user, the current system, and the business processes that the system must support.	System Design focuses on the design of the system, its architecture, and the components that make up the system.
Output	System Analysis produces the requirements document that describes the desired system.	System Design produces the design document that describes the architecture and components of the system.
Time	System Analysis is a one-time process that occurs at the beginning of the project.	System Design is an ongoing process that occurs throughout the project.
Methodology	System Analysis relies on a structured approach such as the Waterfall Model or the Agile Methodology.	System Design relies on an iterative approach such as the Spiral Model.

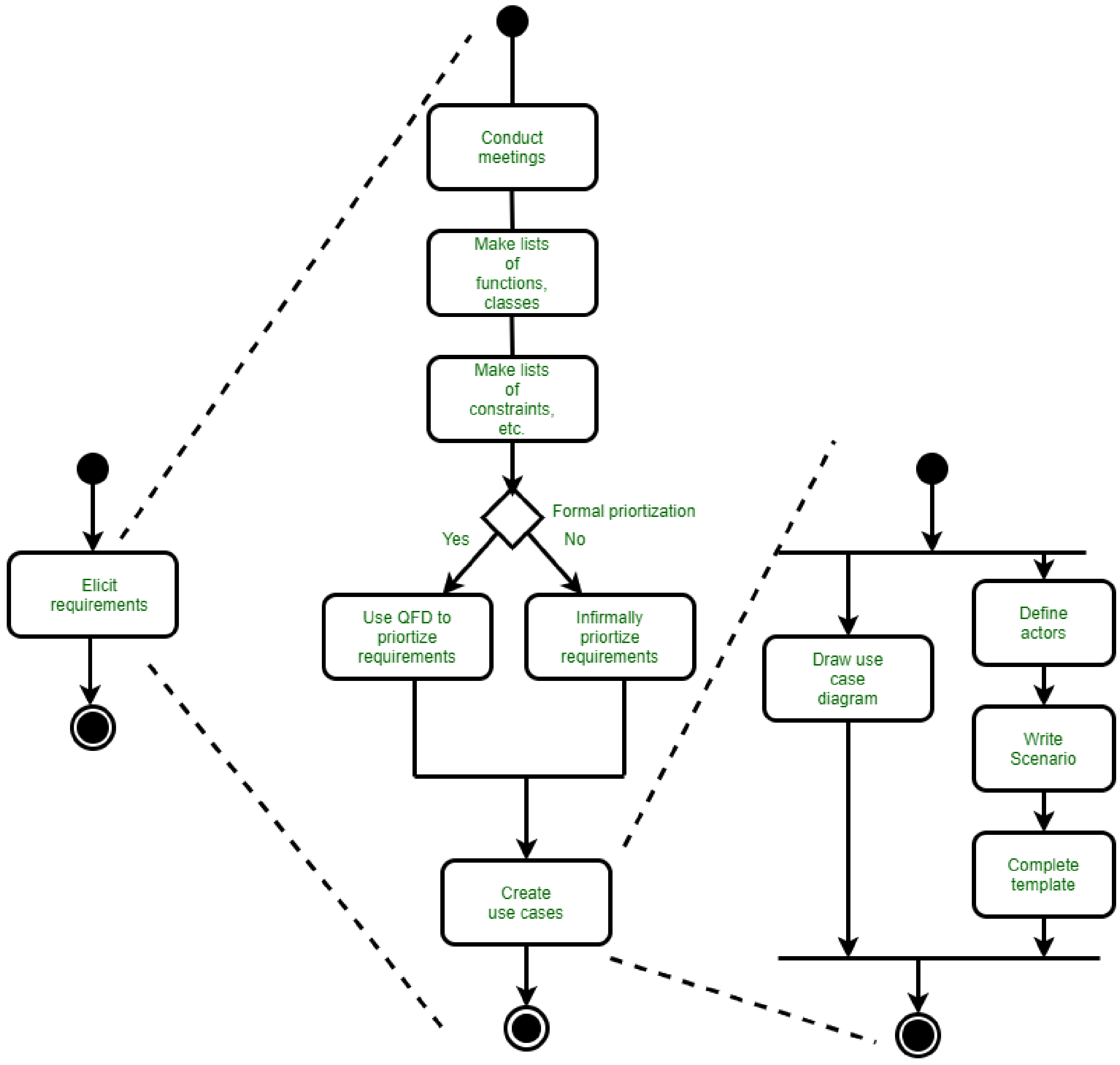
Factors	**System Analysis**	**System Design**
Tools	System Analysis utilizes tools such as interviews, surveys, questionnaires, and observation.	System Design utilizes tools such as data flow diagrams and object-oriented diagrams.
Process	System Analysis is the first step in the software development process.	System Design is the second step in the software development process.
Goals	The goal of System Analysis is to identify and understand the user requirements and the business processes that the system must support.	The goal of System Design is to create a design that meets the user requirements and supports the business processes.
Risk	System Analysis involves minimal risk.	System Design involves significant risk, as the design may not meet the user requirements or support the business processes.
Problem Solving	System Analysis focuses on problem identification and definition.	System Design focuses on problem-solving and finding solutions.

<u>Elements of the Requirements Model</u>

Requirements for a computer-based system can be seen in many different ways. Some software people argue that it's worth using a number of different modes of representation while others believe that it's best to select one mode of representation. The specific **elements of the requirements model** are dedicated to the analysis modeling method that is to be used.

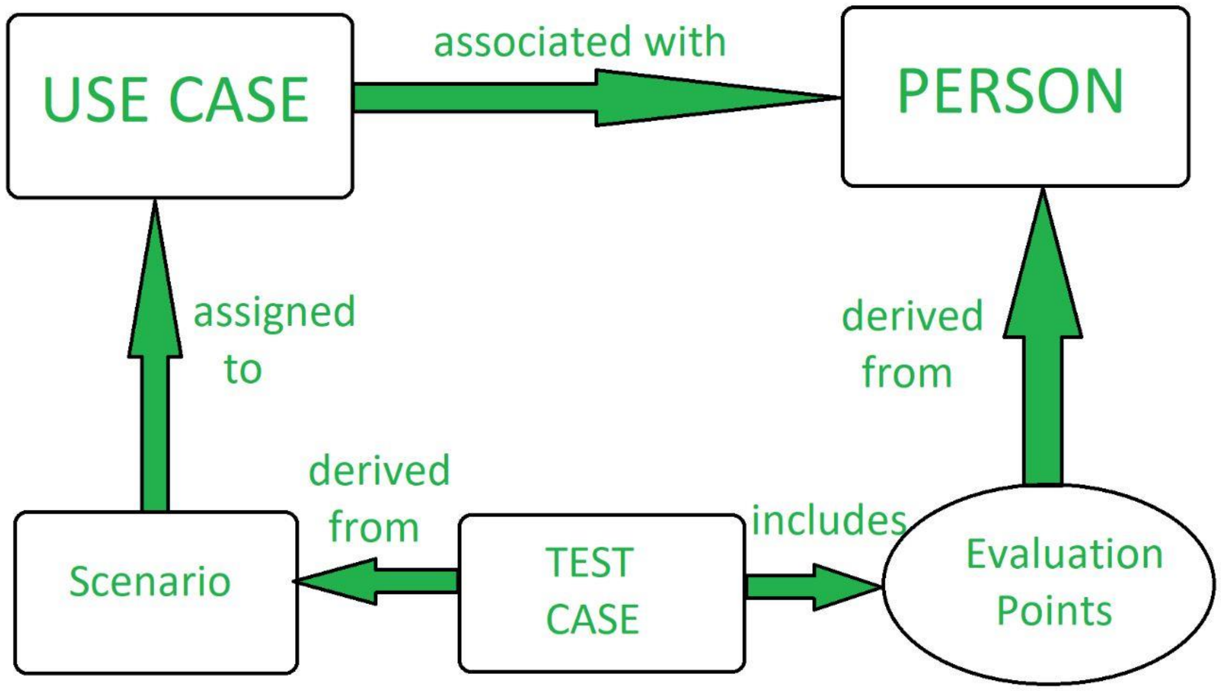
- **Scenario-based elements :** Using a scenario-based approach, system is described from user's point of view. **For example**, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases. Figure 1(a) depicts a UML activity diagram for eliciting requirements and representing them using use cases. There are three levels of elaboration.

- **Class-based elements :** A collection of things that have similar attributes and common behaviors i.e., objects are categorized into classes. **For example**, a UML case diagram can be used to depict a Sensor class for the SafeHome security function. Note that diagram lists attributes of sensors and operations that can be applied to modify these attributes. In addition to class diagrams, other analysis modeling elements depict manner in which classes collaborate with one another and relationships and interactions between classes.
  - **Behavioral elements :** Effect of behavior of computer-based system can be seen on design that is chosen and implementation approach that is applied. Modeling elements that depict behavior must be provided by requirements model.
  - Method for representing behavior of a system by depicting its states and events that cause system to change state is state diagram. A state is an externally observable mode of behavior. In addition, state diagram indicates actions taken as a consequence of a particular event. To illustrate use of a state diagram, consider software embedded within safe Home control panel that is responsible for reading user input.
  - **Flow-oriented elements :** As it flows through a computer-based system information is transformed. System accepts input, applies functions to transform it, and produces output in a various forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. Transform may compromise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system. Output produce a 200-page report or may light a single LED. In effect, we can create a flow model for any computer-based system, regardless of size and complexity.



Scenario Testing – Software Testing

Scenario testing helps testers to know how the software will exactly work when end user will use it. As the scenario testing tests the business process flow of the software so it helps in figure out a lot of defects which cannot be found with the help of other testing. Scenario testing is carried out by creating test scenarios which copy the end users usage. In this article, we'll learn about it's characteristics, process, methods and risks.



What is Scenario Testing?

Scenario Testing is a Software Testing Technique that uses scenarios i.e. speculative stories to help the tester work through a complicated problem or test system. The ideal scenario test is a reliable, complicated, convincing or motivating story the outcome of which is easy to assess. It is performed to ensure that the end to end functioning of software and all the process flow of the software are working properly.
In scenario testing:

1. The testers assume themselves to be the end users and find the real world scenarios or use cases which can be carried out on the software by the end user.
2. The testers take help from clients, stakeholders and developers to create test scenarios. A test scenario is a story which describes the usage of the software by an end user.

Characteristics of Scenario Testing
A scenario test has five key characteristics:
1. Story
Scenario tests are sometimes given as stories or narratives that outline a certain circumstance or environment in which the application is expected to function. Stakeholders may more easily relate to the testing method and understand how the product will function in real-world scenarios when stories are used.
2. Motivating
Scenario tests need to inspire and relate to stakeholders or end users. A compelling scenario motivates stakeholders to actively participate, which improves teamwork and results in a greater understanding of user requirements and expectations.
3. Credible

Since stakeholders can see that the scenarios are meaningful and representative of actual circumstances, credible scenarios boost stakeholder's trust in the testing process.

4. Complex

Since they include a variety of inputs, conditions and interactions, scenario tests are intended to be complex. The complicated nature of the scenarios guarantees a deeper analysis of the software's capabilities and it's capacity to manage complex, multiple circumstances.

5. Easy to evaluate

Even if situations could be complex, they should be designed so that evaluation is simple. Simple evaluation speeds up decision-making and feedback, enabling effective problem-solving and identification.

Scenario Testing Process

Methods in Scenario Testing

There are various methods in scenario testing:

1. **System scenarios:** Scenario tests used in this method are only those sets of realistic, user activities that cover various components in the system.
2. **Use-case and role-based scenarios** In the use-case and role-based scenario method the focus is specifically on how the system is used by a user with different roles and environment.
3. **Recovery Scenarios:** Test scenarios for data backup, restoration and recovery are called recovery scenarios. Additionally, it assesses how the system would function in the case of a server or component failure.
4. **Positive Scenarios:** Examining the system in conditions that are common and expected.
5. **Negative Scenarios:** Assessing the way the system responds to incorrect or unexpected inputs and circumstances.
6. **Boundary Scenarios:** Testing the system at the boundaries of its inputs and outputs is known as boundary scenario.
7. **Error scenarios:** These involve generating error scenarios and testing that the system reacts correctly.

Risks of Scenario Testing

1. **Limited Scenario Reporting:** Inadequate coverage may result from not identifying or testing every case that could arise.
2. **Little In-depth Analysis of Edge Cases:** If edge situations or extreme possibilities are disregarded, there could be problems when using them in the actual world.
3. **High Maintenance Costs**: Maintaining a big number of situations can get expensive and time-consuming.
4. **Dependency on Data:** Since scenarios could rely heavily on particular data sets, simulating differences in real life might be difficult.
5. **A Lost Feeling of Security:** If other crucial testing kinds, like unit or integration testing, are disregarded, relying just on scenario testing could provide a false sense of security.

6. **Excessive Focus on Positive Situations:** Over focusing on favorable possibilities can cause one to overlook bad or extraordinary scenarios, which are more likely to cause problems for the system.
7. **The complexity of executing a scenario:** Complex scenario execution might be prone to errors, making it challenging to replicate problems or interpret test findings.

Class based modeling in Software Engineering

Class Modeling focuses on static system structure in terms of classes (Class, Data Type, Interface and Signal items), Associations and on characteristics of Classes (Operations and Attributes).

Class Modeling focuses on static system structure in terms of classes (Class, Data Type, Interface and Signal items), Associations and on characteristics of Classes (Operations and Attributes).
Modeler provides Class Diagrams and Composite Structure Diagrams to support the definition of the Class Model:

- The Class Diagram is the primary diagram for defining Classes and their Attributes, Operations and relationships. The Class Diagram notation is based on the Unified Modeling Language (UML).
- The Composite Structure Diagram defines the structure of Classes, in particular, showing how Class parts and ports connect with each other. The Composite Structure Diagram notation is based on the UML 2.0 notation, with the addition of SysML IO Flows.

Flow-oriented modeling in software engineering

What is Flow Oriented Modeling ?

The flow oriented modeling represents how data objects are transformed at they move through the system. Derived from structured analysis, flow models use the data flow diagram, a modeling notation that depicts how input is transformed into output as data objects move through the system. Each software function that transforms data is described by a process specification or narrative. In addition to data flow, this modeling element also depicts control flow.

.

Data Flow Diagram :

The data flow diagram represents the flows of data between different process in a business. It is a graphical technique that depicts information flow and transforms that are applied as data from input to output. It provides a simple, intuitive method for describing business processes without focusing on the details of computer systems. DFDs are attractive techniques because they provide what users do rather than what computers do. In DFD, there are four symbols are used :

1. Process :
The circle represents the process. An activity that changes or transforms data flows. Since they transform incoming data to outgoing data, all processes must have inputs and outputs on a DFD.
2. Data Flow :
The labeled arrows indicate incoming and outgoing data flow. Movement of data between external entities, processes and data stores is represented with an arrow symbol, which indicates the direction of flow.
3. Data Store :
The rectangle represents an external entity. A data store does not generate any operations but simply holds data for later access.
4. External Entity :
In Data Flow Diagrams external entities produce and consume data that flows between the entity and the system being diagrammed.

<u>Behavioral Model</u>

Overall behavior of a system can be fully understood by Behavioral model. **Behavioral Model** is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system. So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.

Example : Consider an Elevator. This elevator is for n number of floors and has n number of buttons one for each floor. Elevator's working can be explained as follows :

1. **Elevator buttons** are type of set of buttons which is there on elevator. For reaching a particular floor you want to visit, "elevator buttons" for that particular floor is pressed. Pressing, will cause illumination and elevator will start moving towards that particular floor for which you pressed "elevator buttons". As soon as elevator reaches that particular floor, illumination gets canceled.

2. **Floor buttons** are another type of set of buttons on elevator. If a person is on a particular floor and he wants to go on another floor, then elevator button for that floor is pressed. Then, process will be same as given above. Pressing, will cause illumination and elevator to start moving, and when it reaches on desired floor, illumination gets canceled.
3. When there is no request for elevator, it remains closed on current floor.State Transition Diagram for an elevator system is shown below –**Advantages :**

- Behavior and working of a system can easily be understood without any effort.
- Results are more accurate by using this model.
- This model requires less cost for development as cost of resources can be minimal.
- It focuses on behavior of a system rather than theories.

Disadvantages :

- This model does not have any theory, so trainee is not able to fully understand basic principle and major concept of modeling.
- This modeling cannot be fully automated.
- Sometimes, it's not easy to understand overall result.
- Does not achieve maximum productivity due to some technical issues or any errors.

Software Design process

Software Design is the process of transforming user requirements into a suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, this phase aims to transform the SRS document into a design document.

The following items are designed and documented during the design phase:

- Different modules are required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms are required to be implemented among the individual modules.

Objectives of Software Design:

1. **Correctness:**
   A good design should be correct i.e., it should correctly implement all the functionalities of the system.
2. **Efficiency:**
   A good software design should address the resources, time, and cost optimization issues.
3. **Flexibility:**
   A good software design should have the ability to adapt and accommodate changes easily. It includes designing the software in a way, that allows for modifications,

enhancements, and scalability without requiring significant rework or causing major disruptions to the existing functionality.

4. **Understandability:**
   A good design should be easily understandable, it should be modular, and all the modules are arranged in layers.
5. **Completeness:**
   The design should have all the components like data structures, modules, and external interfaces, etc.
6. **Maintainability:**
   A good software design aims to create a system that is easy to understand, modify, and maintain over time. This involves using modular and well-structured design principles e.g.,(employing appropriate naming conventions and providing clear documentation). Maintainability in software Design also enables developers to fix bugs, enhance features, and adapt the software to changing requirements without excessive effort or introducing new issues.

Software Design Concepts:

Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software.

Software Design process

Points should be considered while Designing Software:

1. **Abstraction- (Hide Irrelevant data)**
   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.
2. **Modularity- (subdivide the system)**
   Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays, there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a

trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

3. **Architecture- (design a structure of something)**
   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.
4. **Refinement- (removes impurities)**
   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.
5. **Pattern- (a Repeated form)**
   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
6. **Information Hiding – Hide the Information**
   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules.
7. **Refactoring-(Reconstruct something)**
   Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t impact the behavior of the design and improves the internal structure”.

Different levels of Software Design:
There are three different levels of software design. They are:

1. **Architectural Design:**
   The architecture of a system can be viewed as the overall structure of the system & the way in which structure provides conceptual integrity of the system. The architectural design identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of the proposed solution domain.

2. **Preliminary or high-level design:**
   Here the problem is decomposed into a set of modules, the control relationship among various modules identified, and also the interfaces among various modules are identified. The outcome of this stage is called the program architecture. Design representation techniques used in this stage are structure chart and UML.

3. **Detailed design:**
   Once the high-level design is complete, a detailed design is undertaken. In detailed design, each module is examined carefully to design the data structure and algorithms. The stage outcome is documented in the form of a module specification document.

Data Design Elements

Following are the types of design elements:

1. Data design elements

- The data design element produced a model of data that represent a high level of abstraction.
- This model is then more refined into more implementation specific representation which is processed by the computer based system.
- The structure of data is the most important part of the software design.

2. Architectural design elements

- The architecture design elements provides us overall view of the system.
- The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

The architecture model is derived from following sources:

- The information about the application domain to built the software.
- Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
- The architectural style and pattern as per availability.

3. Interface design elements

- The interface design elements for software represents the information flow within it and out of the system.
- They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

1. The user interface
2. The external interface to the other systems, networks etc.

3. The internal interface between various components.

4. Component level diagram elements

- The component level design for software is similar to the set of detailed specification of each room in a house.
- The component level design for the software completely describes the internal details of the each software component.
- The processing of data structure occurs in a component and an interface which allows all the component operations.

- In a context of object-oriented software engineering, a component shown in a UML diagram.
- The UML diagram is used to represent the processing logic.

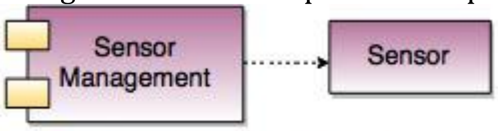


Fig. - UML component diagram for sensor managemnet

5. Deployment level design elements

- The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.
- Following figure shows three computing environment as shown. These are the personal computer, the CPI server and the Control panel.

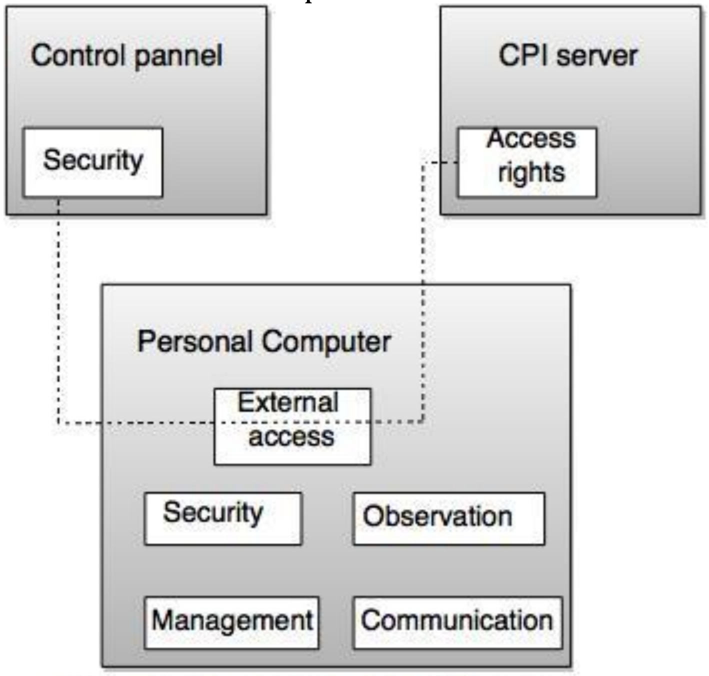


Fig. - Deployment level diagram