##### **CHAPTER-1**

**1.1.INTRODUCTION**

In the current development of 5G networks, the protection of data stored in the network is highly important. Securing the network from malicious activity is highly demanded to secure the huge data in the cloud. Network hazards occuring in the massive network impact the system security with loop holes or wormholes in the system. The research effort developed an intrusion detection system that automatically identifies malware or other random activities by decoding packets and checking them each time they join the network. In this research, we provide an IDSS equipped with a sound-emitting alarm system that may be activated in response to any potentially malicious event. .

Various network anomalies happen to be occurring in the system that extracts the large scale of confidential data, information on transaction data, system activity logs, follow up details are hacked by the third party. These activities need to be monitored by the system intrusion detection models. Because of the flexible usage of cloud protocol, people are accessing the cloud for various reasons. The more frequent access to the cloud also found it important to track the activity of the user. The malicious user can get access in between the activity of similar logons made frequently.

The intrusion attacks can be protected through installing antivirus softwares, windows defenders, security guards etc. As a result, it is now easier for end users to upload data on a more frequent basis. Enable the port open to accept all inputs for a predetermined amount of time each time the user logs in to a particular network. Hackers and third-party users can exploit this key gap to gain access to the network grid's most sensitive data. Interruption recognition frameworks are little devices or programming that goes about as an entrance watch in the organization focuses to disregard the different action during weighty traffic. Performed a study effort on a revolutionary triple intrusion detection system, in which the IDPS assures secure connection with SDN-IOT in terms of failure analysis, accuracy, precision, latency, traffic crown sensing, and other such things; they have employed fuzzy based authentication scheme. They came to the conclusion that more enhancements were required with models that preserved privacy and used new strong authentication .

The IDSS or so called intrusion detection system offers enhanced protection to the computers to protect the system data from various third party attacks. grid level security is provided by the firewalls. the basic protection strategy followed by the system through antivirus installations, troubleshooting the threats, malware protection tools are employed in common. various classifications of intrusion detection systems are discussed. The common categories of intrusion detection systems are described below.

**NIDS: Network Intrusion detection system**

NIDS are models preprogrammed to be initiated for a node based anomaly detection framework. The system tracks all the incoming patterns of data and formulates the presence of anomaly patterns. Firewall protection is implemented in common where numerous anonymous software entries are mapped.

**System level Intrusion Detection**

The system level intrusion detection system are networks connected with the IoT devices. The IDS system runs independently in the network activity and alerts the administrator on entry of intrusion attacks. the system control takes over the attack scenario if the alert is acknowledged. In case of resilient attack entry, the system control is vanished by the slow attacks and a part of the anomaly code is dumped into the system hidden location. From the hidden location, the malwares takes control of the system. Each system independently has an intrusion detection framework.

**Protocol enabled Intrusions**

The systems and devices connected over the massive Internet of things (IoT) network have various agents connected over the server. the attacks enter over the communication protocols and happen to be attacked. The simple HTTP link from the malicious user can take over the control of the system. The regular screening of HTTP links are implemented by the system to prevent network attacks.

**Intrusion detection through Application**

Application specific modules are impacted by the network attacks. a specific application is affected by the malicious activity. The application is getting updated from the internet. During the updation of software applications, the malicious pattern also gets updated. This kind of application based system is keenly monitored.

Hybrid Network

In certain cases the combination of above explained network intrusion, host based intrusion, application based intrusion are employed as Hybrid network.

**Signature based IDSS (Start type from here)**

Signature based detection model is used to screen the external anomalies through affixed pattern called signature verification. artificial neural network algorithm to match the pattern between the miscellaneous activity and given input data. In case of pattern recognition required for intrusion detection system the trained model is helpful to make the alignment. The signature based authentication is helpful to keep the secure environment in cloud per message. The concept of top for signature based intersection detection system is similar to that of Malware detection in IoT networks and mobile devices.

Anomaly based in two deduction systems or helpful to detect the malware Attacks coming over the network routers. The continuous tracking scenario is utilized to make the suspicious attack pattern detection within the connected Network and protrude mission learning models or analysed to prove the actual origin of the problem.

* The study focuses on gathering a variety of evidence from the literature regarding the demand for intrusion detection systems, evaluating the drawbacks of existing models, coming up with a concept that would allow us to proceed with the research on implementations of intrusion detection systems, and developing a novel methodology that improvises from the existing system.
* Deep Intru-Net with Resistant Propagation is the framework used to create the proposed model for deep learning. The future upgrade and understandings on arrangements would be talked about as well.

**1.2.DETAIL DESCRIPTION OF THE PROJECT**

The Deep Intru-Net for anomaly detection in massive IoT networks is a highly sophisticated system designed to detect and prevent cyber attacks in large-scale IoT networks. The system is based on a deep neural network architecture that uses advanced machine learning algorithms to analyze network traffic and identify potential threats.

The project has a number of key requirements in order to achieve its goals. These requirements include:

1. **Scalability:** The system must be able to scale to support massive IoT networks with thousands or even millions of devices. This requires a highly distributed architecture that can handle large volumes of data in real-time.
2. **Real-time processing:** The system must be able to analyze network traffic in real-time, detecting anomalies and potential threats as soon as they occur. This requires a high degree of processing power and efficient algorithms that can quickly identify patterns in the data.
3. **Deep learning:** The system must use advanced machine learning techniques, such as deep neural networks, to analyze network traffic and identify potential threats. This requires a large amount of labeled data for training the models, as well as sophisticated algorithms for feature extraction and classification.
4. **Anomaly detection:** The system must be able to detect anomalous behavior in network traffic, such as unusual traffic patterns or unauthorized access attempts. This requires a high degree of accuracy and a low false positive rate to minimize the impact of false alarms.
5. **Network segmentation and isolation:** The system must be able to segment the network into smaller, more manageable sub-networks, and isolate potentially compromised devices to prevent the spread of malware or other attacks.
6. **Integration with existing security systems:** The system must be able to integrate with existing security systems, such as firewalls, intrusion prevention systems, and SIEM platforms. This requires standardized interfaces and protocols for data exchange and interoperability.
7. **Visualization and reporting:** The system must provide a powerful visualization and reporting engine that enables network administrators to easily monitor and analyze network data in real-time. This requires sophisticated visualization tools and a user-friendly interface.
8. **Customizability:** The system must be highly customizable to suit different types of IoT devices, protocols, and applications. This requires a flexible architecture that can be easily adapted to different use cases and industries.
9. **Privacy and data protection:** The system must protect the privacy and security of user data, ensuring that sensitive information is not exposed or compromised in the event of a cyber attack. This requires strong encryption and authentication mechanisms, as well as robust data protection policies and procedures.

**PROJECT REQUIREMENTS**

**SRS:**

SRS (Software Requirement Specification) is a document that completely

describes what the proposed should do, without describing how the software does it.

Performance Requirements

1) The operation time should be small and the throughput should be high..

2) It should produce timely and accurate result.

**Software Quality Attributes**

**i) Maintainability** – Since it is directly associated with the database, so

there is very little maintainability problem with this application.

**ii) Easy to Learn** – Since there are less number of forms, this application is

very easy to learn with user-friendly screens.

**iii) Flexibility –** This application is very much flexible for future

enhancements.

**Hardware Requirements**

• System : Pentium IV 2.4 GHz.

• Hard Disk : 40 GB.

• Floppy Drive : 1.44 Mb.

• Monitor : 15 VGA Colour.

• Mouse : Logitech.

• Ram : 512 Mb.

**Software Requirements**

• Operating System: Windows

• Coding Language: Python 3.7

**OBJECTIVE OF THE PROJECT**

The objective of the Deep Intru-Net for anomaly detection in massive IoT networks is to develop a highly effective and scalable system for detecting and preventing cyber attacks in large-scale IoT networks. The system is designed to analyze network traffic in real-time using advanced machine learning techniques and detect potential threats as soon as they occur. This objective is driven by the need to secure IoT networks that are increasingly becoming ubiquitous in homes, industrial environments, and critical infrastructure. The Deep Intru-Net aims to provide a reliable and robust system that can detect and prevent attacks such as distributed denial of service (DDoS), malware propagation, and unauthorized access. By achieving this objective, the Deep Intru-Net will help ensure the safety and security of IoT networks, protecting them from attacks that can cause significant damage and disruption. Furthermore, the system aims to provide a scalable solution that can accommodate the growing number of devices and data generated by IoT networks. The ultimate objective of the Deep Intru-Net is to enable the deployment of IoT networks with confidence and trust, ensuring their ability to provide valuable services without being compromised by malicious actors.

**SCOPE OF THE PROJECT**

The scope of the Deep Intru-Net for anomaly detection in massive IoT networks is focused on providing an effective and scalable system for detecting and preventing cyber attacks in large-scale IoT networks. The scope includes several key areas, such as:

**Anomaly detection:**

The system is designed to identify anomalous behavior in network traffic, such as unusual traffic patterns or unauthorized access attempts.

**Deep learning:**

The system utilizes deep neural networks to analyze network traffic and identify potential threats, requiring a large amount of labeled data for training the models.

**Massive IoT networks:**

The system is designed to support large-scale IoT networks with thousands or even millions of devices, requiring a highly distributed architecture that can handle large volumes of data in real-time.

**Real-time processing:**

The system must be able to analyze network traffic in real-time, detecting anomalies and potential threats as soon as they occur.

**Network segmentation and isolation:**

The system is designed to segment the network into smaller, more manageable sub-networks and isolate potentially compromised devices to prevent the spread of malware or other attacks.

**Integration with existing security systems:**

The system must be able to integrate with existing security systems, such as firewalls, intrusion prevention systems, and SIEM platforms.

**Visualization and reporting:**

The system provides a powerful visualization and reporting engine that enables network administrators to easily monitor and analyze network data in real-time.

The scope of the project is driven by the need to provide a reliable and robust system for securing IoT networks, protecting them from attacks that can cause significant damage and disruption. By meeting the scope of the project, the Deep Intru-Net can provide a valuable solution for securing IoT networks, ensuring their ability to provide valuable services without being compromised by malicious actors.

**CHAPTER-2**

**2.1. SUPPORTING MATERIAL TO BASE PAPER**

The supporting material to the base paper for The Deep Intru-Net for anomaly detection in massive IoT networks includes various research papers, technical reports, and conference proceedings related to the topic.

One of the key supporting materials is the research paper "Deep Learning for IoT Big Data and Streaming Analytics: A Survey" by Shan Jiang et al. This paper provides an overview of the use of deep learning techniques for analyzing IoT big data and streaming analytics, including anomaly detection in network traffic.

Another important supporting material is the research paper "A Survey of Machine Learning Techniques for IoT Security" by Mohammed A. Al-Fayoumi et al. This paper provides an overview of various machine learning techniques that can be used for IoT security, including deep learning techniques for detecting anomalies in network traffic.

The conference proceedings of the IEEE International Conference on Internet of Things (IEEE IoT) also provide valuable supporting material for the Deep Intru-Net. The conference covers various topics related to IoT, including security and privacy, and provides a platform for researchers and practitioners to share their findings and insights.

Furthermore, technical reports and whitepapers from industry leaders such as Cisco, Intel, and IBM provide additional supporting material for the Deep Intru-Net. These reports highlight the challenges and opportunities in securing IoT networks and provide insights into the latest technologies and strategies for IoT security.

**2.2.DETAIL DESCRIPTION OF PAPER**

The paper "The Deep Intru-Net for Anomaly Detection in Massive IoT Networks" proposes a novel deep learning-based system for detecting anomalies in massive IoT networks. The paper provides a detailed description of the system architecture, which includes a distributed cluster of GPUs and deep neural networks for processing large volumes of network traffic in real-time.

The paper also presents a comprehensive evaluation of the system using real-world IoT datasets, demonstrating its effectiveness in detecting various types of attacks, including DDoS, botnets, and malware propagation. The evaluation includes a comparison with state-of-the-art anomaly detection systems, highlighting the superior performance of the Deep Intru-Net in terms of accuracy, precision, and recall.

Furthermore, the paper discusses the challenges and opportunities in securing massive IoT networks and highlights the potential of deep learning techniques in addressing these challenges. The paper emphasizes the need for an integrated and scalable system that can provide reliable and robust security for IoT networks, and demonstrates how the Deep Intru-Net can meet this need.

**CHAPTER-3**

**EXISTING SYSTEM**

**3.1 Detailed Description With System Architecture**

A firewall is a network security system that monitors and controls incoming and outgoing network traffic based on a set of predefined security rules. The system architecture of a firewall typically consists of several components, including:

1. **Network Interface:** The network interface component is responsible for receiving and transmitting network traffic between the firewall and the network.
2. **Packet Filtering:** The packet filtering component is responsible for analyzing incoming and outgoing packets and deciding whether to allow or block them based on predefined rules.
3. **Stateful Inspection:** The stateful inspection component is responsible for keeping track of the state of network connections and ensuring that only legitimate traffic is allowed to pass through the firewall.
4. **Application Proxy:** The application proxy component is responsible for inspecting traffic at the application layer and enforcing security policies based on the specific application being used.
5. **Logging and Monitoring:** The logging and monitoring component is responsible for logging and analyzing network traffic data to identify potential security threats and anomalies.

Firewalls can be deployed in various network topologies, such as as a perimeter firewall, internal firewall, or distributed firewall. In a perimeter firewall deployment, the firewall is placed at the network boundary to protect the internal network from external threats. In an internal firewall deployment, the firewall is placed inside the internal network to provide additional security and control between different network segments. In a distributed firewall deployment, multiple firewalls are deployed throughout the network to provide granular control and visibility of network traffic.

**3.2. DETAILED DESCRIPTION OF THE DRAWBACKS**

Despite their usefulness, firewalls also have some drawbacks that limit their effectiveness in some scenarios. Some of these drawbacks include:

1. **Inability to detect all types of threats:** Firewalls are primarily designed to block traffic based on a set of predefined rules. However, they may not be able to detect all types of threats, such as advanced persistent threats (APTs) or zero-day attacks.
2. **Limited visibility into encrypted traffic:** Firewalls cannot inspect encrypted traffic, which can allow malicious traffic to bypass detection.
3. **Inability to protect against insider threats:** Firewalls are designed to protect against external threats, but they may not be able to detect or prevent insider threats, such as malicious or unauthorized users within the network.
4. **Performance impact:** As firewalls need to inspect every packet that passes through them, they can sometimes cause performance issues in high-traffic networks.
5. **Complexity:** Firewalls can be complex to set up and manage, requiring skilled personnel to configure and maintain them effectively.
6. **False positives and negatives:** Firewalls may generate false positives or negatives, allowing malicious traffic to slip through undetected or blocking legitimate traffic.
7. **Lack of context:** Firewalls can only make decisions based on the information available to them at the time, and may not have access to the broader context of the network environment.

**CHAPTER-4**

**PROPOSED SYSTEM**

**DESCRIPTION OF THE FUNCTIONALITIES OF THE PROPOSED SYSTEM**

The Deep Intru-Net for anomaly detection in massive IoT networks provides several key functionalities to enhance network security:

1. **Anomaly detection:** The system uses deep learning techniques to identify anomalous traffic patterns in the network, helping to detect potential security threats.
2. **Real-time monitoring:** The system continuously monitors network traffic in real-time, allowing it to quickly detect and respond to security incidents.
3. **Centralized management:** The system provides a centralized management interface, allowing network administrators to configure and monitor the system from a single location.
4. **Scalability:** The system is designed to scale to support large-scale IoT networks, providing effective security coverage even in complex and diverse network environments.
5. **Automatic rule generation:** The system can automatically generate rules based on network traffic patterns, reducing the need for manual configuration and management.
6. **Rapid response:** The system can quickly respond to detected security incidents, automatically blocking malicious traffic and alerting network administrators to potential threats.
7. **Flexibility:** The system is designed to be flexible and adaptable to different network environments, allowing it to provide effective security coverage in a wide range of scenarios.

**DESCRIPTION OF THE ADVANTAGES OF THE PROPOSED SYSTEM**

The Deep Intru-Net for anomaly detection in massive IoT networks offers several advantages over traditional network security solutions:

1. **Enhanced detection accuracy:** The system uses advanced deep learning techniques to accurately detect anomalies in network traffic, helping to identify potential security threats that may go unnoticed by other systems.
2. **Real-time monitoring:** The system provides real-time monitoring of network traffic, enabling rapid response to security incidents and minimizing the risk of data loss or network downtime.
3. **Centralized management:** The system provides a centralized management interface, making it easier for network administrators to configure, monitor, and respond to security incidents from a single location.
4. **Scalability:** The system is designed to be scalable, allowing it to provide effective security coverage even in large and complex IoT networks.
5. **Automatic rule generation:** The system can automatically generate rules based on network traffic patterns, reducing the need for manual configuration and management.
6. **Rapid response:** The system can quickly respond to detected security incidents, automatically blocking malicious traffic and alerting network administrators to potential threats.
7. **Flexibility:** The system is designed to be flexible and adaptable to different network environments, making it an effective solution for a wide range of scenarios.

**Software Design**

System design is the second step in the system life cycle, in which overall design of the system is achieved. The functionalities of the system is designed and studied in this phase. The first step is designing of program specification. This determines the various data inputs to the system, data flow and the format in which output is to be obtained.

Design phase is a transmission phase because it is a transition from user oriented document to computer data. The activity in the design phase is the allocation of functions to manual operations, equipment and computer programs. Flow charts are prepared in the study time and is decomposed until all functions in the system perform evidently.

Design is a multi-step process that focuses on data structures, software architecture, procedural details( algorithms etc) and links between the modules. The design process goes through logical and physical stages. In logical design reviews are made linking existing system and specification gathered. The physical plan specifies any hardware and software requirement, which satisfies the local design.

Modularization of task is made in this phase. The success of any integrated system depends on the planning of each and every fundamental module. Usually a project is revised in step by step sequence. Inter-phase management of such module is also important. Software design methodology changes continually as new methods, better analysis and broader understanding evolve.

**CHAPTER-5**

**CHAPTER-6**

**Design using UML diagrams :**

**Object Oriented Methodology:**

Object oriented methodology is a system development approach encouraging and facilitating re-use of software components. With this methodology, a computer can be developed on a component basis which enables the effective reuse of existing components and facilitates the sharing of its components by other systems. It employs international standard unified modelling language (UML) from the object management group (OMG) .

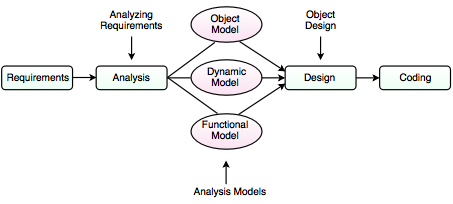
Using this methodology , a system can be developed on a component basis , which enables the effective reuse of existing components, it facilitates the sharing of its other system components. Objects oriented methodology asks the analyst to determine what the objects of the system are?, what responsibilities and relationships an object has to with the other objects? And how they behave over time?

**There are three types of object oriented methodologies:**

* Object Modelling Technique(OMT)
* Object Process Methodology(OPM)
* Rational Unified Process(RUP)

**Object Modelling Technique(OMT):**

It was one of the first object oriented methodologies and was introduced by RumBaugh in 1991. OMT uses that are combined in a way that are combined in a way that is analogous to the older structured methodologies.



**Analysis:**

The main goal of the analysis is to build models of the world. The requirements of the users, developers and managers provide the information needed to develop the initial problem statements.

**OMT MODELS**

**I. Object Model**

* It depicts the object classes and their relationships as a class diagram, which represents the static structure of the system.
* It observes all the objects as static and does not pay any attention to their dynamic nature.

**II. Dynamic Model**

* It captures the behaviour of the system over time and the flow control and events in the Event-Trace Diagrams and State Transition Diagrams.
* It portrays the changes occurring in the states of various objects with the events that might occur in the system.

**III. Functional Model**

* It describes the data transformations of the system.
* It describes the flow of data and the changes that occur to the data throughout the system.

**Design**

* It specifies all of the details needed to describe how the system will be implemented.
* In this phase, the details of the system analysis and system design are implemented.
* The objects identified in the system design phase are designed.

**Object Process Methodology (OPM):**

It is also called as second generation methodology. It was first introduced in 1995. It had only one diagram that is the object process diagram which is used for modelling the structure, function and behaviour of the system. It has a strong emphasis on modelling but has a weaker emphasis on process. It consists of three main process

**I. Initiating:** It determines high level requirements, the scope of the system and the resources that will be required.  
**II. Developing:** It involves the detailed analysis, design and implementation of the system.

**III. Deploying:** It introduces the system to the user and subsequent maintenance of the system.

**Rational Unified Process (RUP):**

It was developed in Rational Corporation in 1998. It consists of four phases which can be broken down into iterations.  
I. Inception  
II. Elaboration  
III. Construction  
IV. Transition

Each iteration consists of nine work areas called disciplines. A discipline depends on the phase in which the iteration is taking place. For each discipline, RUP defines a set of artefacts (work products), activities (work undertaken on the artefacts) and roles (the responsibilities of the members of the development team)

**Objectives of object oriented methodologies:**

* To encourage greater re-use.
* To produce a more detailed specification of system constraints.
* To have fewer problems with validation (Are we building the right product?).

**Benefits of object oriented methodologies:**

* It represents the problem domain, because it is easier to produce and understand designs.
* It allows changes more easily.
* It provides nice structure for thinking abstracting and leads to modular design.
* Simplicity
* Reusability
* Increased quality
* Maintainable
* Scalable
* Modularity

**Objectives**

After going through this unit, you should be able to:

* Find the importance of OO approach.
* Define the basic concepts of OO approach.
* Differentiate between object and procedure-oriented approaches.
* Know about various OO languages. Object Oriented Technology and Java
* Describe the applications of OOP, and
* Understand the benefits of OO approach

**Evolution of OO Methodology**

The earliest computers were programmed in machine language using 0 and 1. The mechanical switches were used to load programs. Then, to provide convenience to the programmer, assembly language was introduced where programmers use pneumonic for various instructions to write programs. But it was a tedious job to remember so many pneumonic codes for various instructions. Other major problem with the assembly languages is that they are machine architecture dependent. To overcome the difficulties of Assembly language, high-level languages came into existence. Programmers could write a series of English-like instructions that a compiler or interpreter could translate into the binary language of computers directly. These languages are simple in design and easy to use because programs at that time were relatively simple tasks like any arithmetic calculations. As a result, programs were pretty short, limited to about a few hundred line of source code. As the capacity and capability of computers increased, so did the scope to develop more complex computer programs. However, these languages suffered the limitations of reusability, flow control (only goto statements), difficulty due to global variables, understanding.

**Structured Programming**

When the program becomes larger, a single list of instructions becomes unwieldy. It is difficult for a programmer to comprehend a large program unless it is broken down into smaller units. For this reason languages used the concept of functions (or subroutines, procedures, subprogram) to make programs more comprehensible. A program is divided into functions or subroutines where each function has a clearly defined purpose and a defined interface to the other functions in the program. Further, a number of functions are grouped together into larger entity called a module, but the principle remains the same, i.e. a grouping of components that carryout specific tasks.

Dividing a program into functions and modules is one of the major characteristics of structured programming. By dividing the whole program using functions, a structured program minimizes the chance that one function will affect another. Structured programming helps the programmer to write an error free code and maintain control over each function. This makes the development and maintenance of the code faster and efficient. Structured programming remained the leading approach for almost two decades. With the emergence of new applications of computers the demand for software arose with many new features such as GUI (Graphical user interface). The complexity of such programs increased multi-fold and this approach started showing new problems.

The problems arose due to the fundamental principle of this paradigm. The whole emphasis is on doing things. Functions do some activity, maybe a complex one, but the emphasis is still on doing. Data are given a lower status. For example in banking application, more emphasis is given to the function which collects the correct data in a desired format or the function which processes it by doing some summation, manipulation etc. or a function which displays it in the desired format or creates a report. But you will also agree that the important part is the data itself. The major drawback with structured programming are its primary components, i.e., functions and data structures. But unfortunately functions and data structures do not model the real world very well. Basically to model a real world situation data should be given more importance. Therefore, a new approach emerges with which we can express solutions in terms of real world entities and give due importance to data.

**Object Oriented Programming**

The world and its applications are not organized as functions and values separate from one another. The problem solvers do not think about the world in this manner. They always deal with their problems by concentrating on the objects, their characteristics and behavior.

The world is Object Oriented, and Object Oriented programming expresses programs in the ways that model how people perceive the world. It shows different real world objects around us which we often use for performing different functions. This shows that problem solving using the objects oriented approach is very close to our real life problem solving techniques.

The basic difference in Object Oriented programming (OOP) is that the program is organized around the data being operated upon rather than the operations performed. The basic idea behind OOP is to combine both, data and its functions that operate on the data into a single unit called object. Now in our next section, we will learn about the basic concepts used extensively in the Object Oriented approach.

**Basic Concepts of OO Approach**

Object Oriented methods are favored because many experts agree that Object Oriented techniques are more disciplined than conventional structured techniques. (Martin and Odell 1992)

The main components of Object Oriented technology are ‘objects and classes’, ‘data abstraction and encapsulation’, ‘inheritance’ and ‘polymorphism’. It is very important for you to understand these concepts. Further, in this unit you can find the details of these concepts.

**Objects**

Let’s start with “Object”. The first thing that we should do in the Object Oriented approach is to start thinking in terms of Objects. The problem to be solved is divided into objects. Start analyzing the problem in terms of objects and the nature of communication between them. Program object should be chosen such that they match closely with real-world objects. Let’s start creating objects using real-life things, for example, the dog. You can create an object representing a dog, It would have data like How hungry is it? How happy is it? Where is it? Now think what are the different functions you can perform on a dog, like eat, bark, run and dig. Similarly, the following can be treated as objects in different programming problems:

• Customers and accounts in a banking system

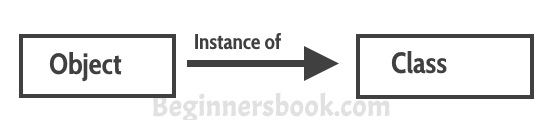
• Salesman, products, customers in a sales tracking system

• Data structures like linked lists, stacks, etc.

• Hardware devices like magnetic tape drive, keyboard, printer etc.

• GUI elements like windows, menus, events, etc. in any window-based application.

Each object contains data and the functions that operate on the data. Objects can interact without having to know details of each other’s data or functions. It is sufficient to know the type of message accepted and the type of response returned by the object. For example, in the banking system, customer object may send a message named as check balance to the account object to get the response, i.e. bank balance. An Object Oriented system can be considered as network of cooperating objects which interact by sending messages to each other. Let us see the below figure, how objects interact by sending messages to one another.



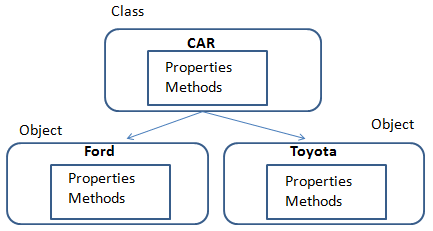
**Classes**

Objects of the similar type can be grouped together to form a class. Can you tell to which class dog belongs? Yes, of course, it belongs to the animal class. Now, let us concentrate on the creation of objects. This can be easily answered if we look at the way of creating any variable in common programming languages. Almost all computer languages have built-in data types, for example integer, character, real, Boolean , etc. One can declare as many variables of any built-in type as needed in any problem solution. In the similar way one can define many objects of the same class. You can take a class as a type created by a programmer.

A class serves as a plan or template. The programmer has to specify the entire set of data and functions for various operations on the data for an object as a user-defined type in the form of a class. In other words, the programmer defines the object format and behavior by defining a class. The compiler of that language does not know about this user-defined data type. The programmer has to define the data and functionality associated with it by designing a class.

Finally, defining the class doesn’t create an object just as the existence of a built-in type integer doesn’t create any variable.

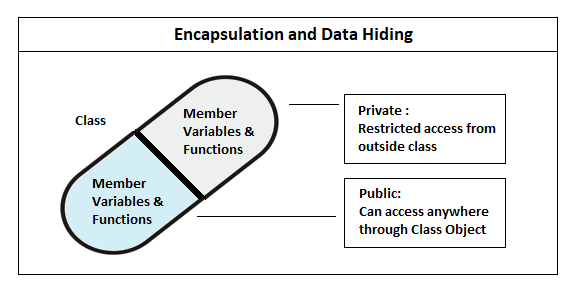
A class is thus a collection of objects of similar type. For example, in a collection of potatoes each individual potato is an object and belongs to the class potato. Similarly, each individual car running on the road is an object, Collectively these cars are known as cars.



**Data abstraction and encapsulation**

The wrapping up of data and functions into a single unit is known as encapsulation. This is one of the strong features of the object oriented approach. The data is not directly accessible to the outside world and only the functions, which are wrapped in the class, can access it. Functions are accessible to the outside world. These functions provide the interface to access data. If one wants to modify the data of an object, s/he should know exactly what functions are available to interact with it. This insulation of the data from direct access by the program is known as data hiding.

Abstraction refers to the act of representing essential features without including the background details to distinguish objects/ functions from other objects/functions. In case of structured programming, functional abstraction was provided by telling, which task is performed by function and hiding how that task is performed. A step further, in the Object Oriented approach, classes use the concept of data abstraction. With data abstraction, data structures can be used without having to be concerned about the exact details of implementation. As in case of built-in data types like integer, floating point, etc. The programmer only knows about the various operations which can be performed on these data types, but how these operations are carried out by the hardware or software is hidden from the programmer. Similarly in Object Oriented approach, classes act as abstract data types. Classes are defined as a set of attributes and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created.



**Inheritance**

Inheritance is the process by which objects of one class acquire the properties of objects of another class in the hierarchy. For example, the scooter is a type of the class two-wheelers, which is again a type of (or kind of) the class motor vehicles. As shown in the Figure 5 the principle behind it is that the derived class shares common characteristics with the class from which it is derived.

New classes can be built from the existing classes. It means that we can add additional features to an existing class without modifying it. The new class is referred as derived class or sub class and the original class is known as base class or super class. Therefore, the concept of inheritance provides the idea of reusability. This inheritance mechanism allows the programmer to reuse a class that is made almost, but not exactly, similar to the required one by adding a few more features to it.

Feature A and Feature B of the base class are inherited in all the three derived classes. Also, each derived class has added its own features according to the requirement. Therefore, new classes use the concept of reusability and extend their functionality.

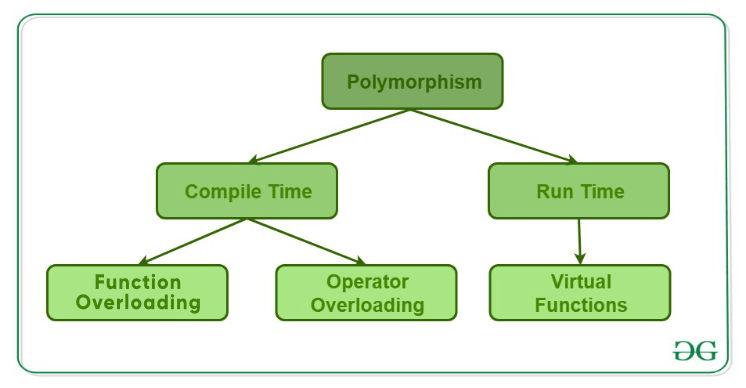


**Polymorphism**

Polymorphism means the ability to take more than one form of the same property. For example, consider an addition operation. It shows a different behavior in different types of data. For two numbers, it will generate a sum. The numbers may integers or float. Thus the addition for integers is different from the addition to floats.

An example is shown in Figure 6, where single function name, i.e. draw can be used to draw different shapes. The name is the same in all the classes but the functionality differs. This is known as function overriding, which is a type of polymorphism. We will discuss it in detail in our next unit.

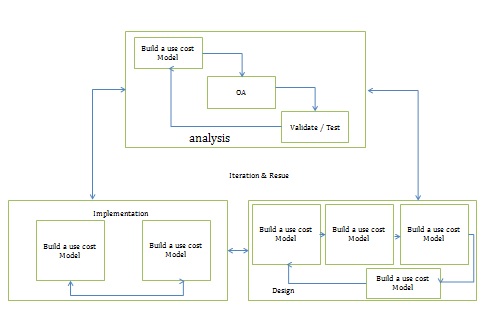
In our example, we also used a function “area” which was inherited by all the three derived classes, i.e. triangle, circle and rectangle. But in the cases of the circle and the triangle, we override the function area because the data types and number of parameters varies.



**Object oriented approach:**

The major factor, which leads to the development of this new approach i.e, Object Oriented approach is to resolve many problems encountered earlier in the procedural approach.

In this approach, we decompose a problem into a number of entities called objects and then build data and functions around these entities. The notion of “Object” comes into the picture. ‘A collection of data and its operations is referred to as an object’. Data is a vital element in the program development. Data is local to an object. This is encapsulated within an object and is not accessible directly from outside the object. These objects know how to interact with another object through the interface (a set of operations). The organization of data and functions in Object Oriented programs is



**The salient features of Object Oriented programming are:**

• More emphasis is on data rather than procedure.

• Programs are modularized into entities called objects.

• Data structures methods characterize the objects of the problem.

• Since the data is not global, there is no question of any operations other than those defined within the object, accessing the data. Therefore, there is no scope of accidental modification of data.

• It is easier to maintain programs. The manner in which an object implements its operations is internal to it. Therefore, any change within the object would not affect external objects. Therefore, systems built using objects are resilient to change.

• Object reusability, which can save many human hours of effort, is possible. An application developer can use objects like ‘array’, ‘list’, ‘windows’, ‘menus’, ‘event’ and many other components, which were developed by other programmers, in her program and thus reduce program development time.

• It employs bottom-up approach in program design.

**Benefits of OOP:**

OOP offers several benefits to both the program developer and the user. The new technology provides greater programmer productivity, better quality of software and lesser maintenance cost. The major benefits are:

• Ease in division of job: Since it is possible to map objects of the problem domain to those objects in the program, the work can be easily partitioned based on objects.

• Reduce complexity: Software complexity can be easily managed.

• Provide extensibility: Object Oriented systems can be easily upgraded from small to large system.

• Eliminate redundancy: Through inheritance we can eliminate redundant code and extend the use of existing classes.

• Saves development time and increases productivity: Instead of writing code from scratch, solutions can be built by using standard working modules.

• Allows building secure programs: Data hiding principle helps programmer to build secure programs that cannot be accessed by code in other parts of the program.

• Allows designing simpler interfaces: Message passing techniques between objects allows making simpler interface descriptions with external systems.

**Software Architecture with UML Technology:**

What makes software so complex and so difficult to grasp is the fact that the number of information loaded onto a single person is vastly exceeding the capabilities of the human mind. We are not able to handle thousands of pieces of information at any given time as the human short-term memory is quite limited in that respect. The 7+2 rule is a well-known example. This rule states that the human mind can usually only handle 7 distinct things (plus or minus 2) at the same time. Given that restriction, one obvious question that arise is how have we managed even the simplest piece of technology that consists of more (much more) than 9 pieces. The answer to that is probably the concept of modelling. If we encounter too much information ,we consciously and/or unconsciously group this information in some way which makes it easier for us to recall it later on.

The field of Software Engineering uses this human capability of abstraction to create "theories, methods, and tools which are needed to develop software [1]." Those theories, methods, and tools in turn use further abstractions until we have pieces which are small enough for us to comprehend without any further abstraction. Moreover, this need for abstraction is clear in where it emphasizes: "it is not the number of details, as such, that contributes to complexity, but the number of details of which we have to be aware off at the same time." This is the very reason systems engineers have been modelling systems since the inception of the discipline. Specifically, the reasons for modelling systems include among others: definition of the work to be done (requirements), allocation of responsibilities, documentation of the completed system, and communication with the various disciplines who will create, operate, and maintain the system.

Unified Modelling Language has recently been used by Systems Engineers to model systems [12-14]. Many papers and articles have been written on various aspects of systems modelling with UML, most especially Use Cases, but little attention has been focused to develop the system and software architecture of embedded real-time systems. Papers that address development and modelling of system and software architecture do so with a "one size fies all philosophy, by mandating a prescriptive structure that does not address the true needs a many practical system's development. The following of such approaches also may some time result into a badly architectured system which can be a serious handicap to the development of a system.

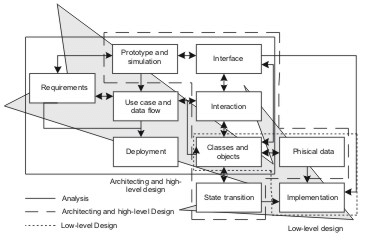
A. Basics of Model In the engineering of software, 'model' is used in many different contexts. But in essence, a model is an abstraction. It abstracts away all kinds of details that are not relevant. The quality of the model is judged from the purpose of the model. There are different model like "system model', 'process model', 'mathematical model", "platform models", "meta model [15] etc used in software engineering for different purpose. A system model is a presentation of particular aspects of (software) systems and their application context that is defined by predefined description techniques based on text, graphics and tables.

**A .Architectural Views and Styles in UML:**

Architecting in UML is represented by different views of UML, which is shown in Fig. 2. It shows the analysis, architecture, and design of a software system, implementation and testing, which often are part of the major development stages (from the developers point of view). The arrows depict the dependencies of the views onto information in other views. Architecture is the appropriate level of abstraction at which rules of a compositional style (i.e., an architectural style) can be exploited and should be elaborated. Doing so results in a set of heuristics that if followed will guarantee a resulting system with certain desirable properties. In many literature UML Views are used to mainly represent the following architecture of a system.

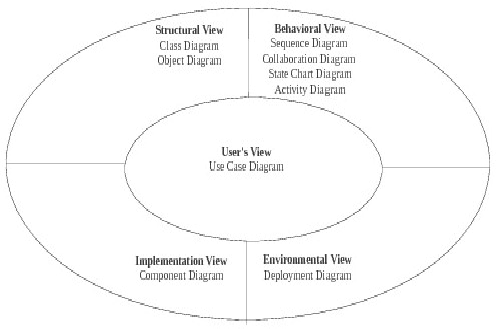
Function Architecture in which the system is structured into a family of sub functions also called services or features.

Logical Component Architecture in which the system is decomposed into sets of components forming architecture.



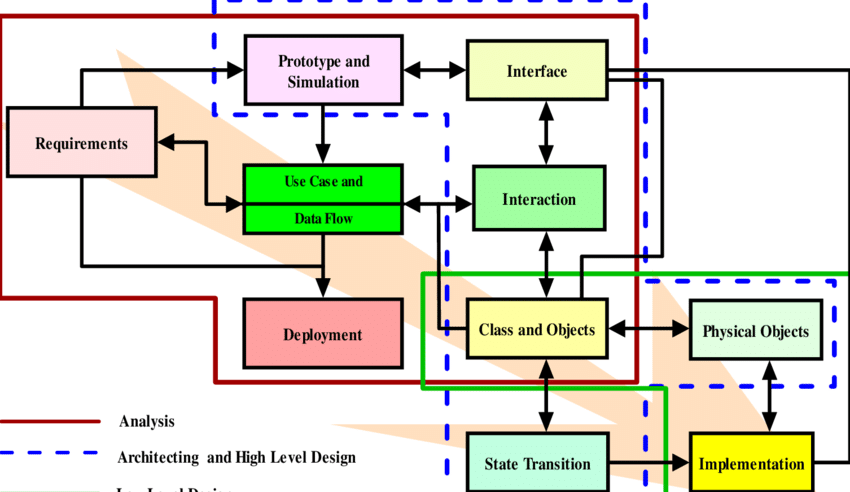
Code architecture: Each state machine is realized by families of modules or classes which form code architecture.

Deployment architecture that maps the components onto a network or hardware systems connected by communication links or bus



**B. Approaches to Architectural Modelling:**

Using UML The suitability of UML as an ADL has been conceived by many as it provides a common platform and notation from architecture through design to implementation. Like any ADLs, as UML also satisfies the requirement of an ADL with the advantage that it supports more views of representation of the system from different perspectives, this paper considers its suitable extension for architectural modeling. The basic promise of software architecture research is that better software systems can result from modeling their important aspects during, and especially early in the development. Choosing which aspects to model and how to evaluate them are two decisions that frame software architecture research. The four-layer metamodeling architecture of UML suggests three possible approaches for modeling software architectures in UML: Using UML as it is, constrain the UML meta model using UML's built in extension mechanisms, and augment the UML meta model to directly support the needed architectural concepts. Each approach has certain potential advantages and disadvantages for forward and reverse engineering, discussed below



**ARCHITECTURAL METHODS**

The modal driven development process incorporates modelling in the development process at suitable stages that plays a defined role for a given modelling technique as shown. For instance the development of the architecture of a system can be obtained by the use of different modelling techniques for the architecture. The different milestone for architecting the system can be achieved by Function Architecture, Logical Component Architecture, Code architecture etc which give the essential structures of a software system in its different phases using architecting methods as described in the following subsection.

Artifacts Driven In this method of architectural design, the starting point is the textual requirements that are used to identify the artifacts types in the process and procedure to the objectives. The identified artifacts are grouped into subsystems to represent the architectural components and subsequently, the relation between these subsystems are defines elaborate the architecture and architectural boundaries. Although the methodology is relatively simpler, however, obstacles often arise as textual requirements are imprecise and less useful as a of architectural abstraction.

Domain Driven Most software systems can be classified according to the business area and the kind of tasks they support, eg, airline reservation systems, medical record systems, portfolio management systems, order processing systems, inventory management systems, etc. Similarly, we can also classify parts of software systems according to their functionality, e.g. database systems synchronization packages, work flow systems, GUI libraries, numerical code libraries, etc. We refer to areas organized around classes of systems or parts of systems as domains. Obviously, specific systems or components within a domain share many characteristics since they also share many requirements. Therefore, an organization, which has built a number of systems or components in a particular domain, can take advantage of the acquired knowledge when building subsequent systems or components in the same domain.

By capturing the acquired domain knowledge in the form of reusable assets and by reusing these assets in the development of new products, the organization will be able to deliver the new products in a shorter time and at a lower cost. Domain Engineering collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e. reusable work products), as well as providing an adequate means for reusing these assets ( i.e. retrieval, qualification, dissemination, adaptation, assembly, etc.) when building new systems. The purpose of Domain Analysis is to select and define the domain of focus and collect relevant domain information and integrate it into a coherent domain model.

Pattern Driven Experienced developers find when they approach a new problem to solve, that the situation usually has something in common with a solution they have already either created or seen. The problems are not identical and the identical solution will rarely solve the new problems, but the problems are still similar, so a similar solution will probably work. The "similar solution" generalized and formalized, is called a design pattern. Creating design patterns is a problem of abstracting the similarities of the two problems and the solution so that the generic aspects of the original solution can be applied to the new problem at hand. Design patterns codify design expertise and have been used in an intuitive way as long as design has been a recognized activity. The intentional use of design patterns allows us to think about our designs using a vastly richer vocabulary and to this think critically about the trade offs we make during the design process resulting in better optimized, more robust systems. A design pattern is "a generalized solution to a commonly occurring problem."

To be a pattern, the problem must recur often enough to be usefully generalizable. The solution must also be general enough to be applied in a wide set of application domains. If it only applies to a single application domain, then it is probably an analysis pattern. An analysis pattern is similar to a design pattern but applies to a specific application domain such as finance or aerospace. Analysis patterns define ways for organizing problem-specific object analysis models within a single application domain. For some examples of domain-specific analysis patterns.

Use Case Driven The use case driven approach has been the widely accepted architecting technique used by the OO development community to define the business and application functionality from user point of view; this is why the technique is so popular. In this method. the architecture shows what the user wants to do, and then design and build an application to help the user do this. Although, there are number of variants of the same concept, however, the main theme to derive an architectural representation by gathering the business requirements in enough details such that the risk to the project is minimized remains the same

UML is not a programming vernacular yet rather instruments can be utilized to make code in different tongues utilizing UML graphs. UML has an incite relationship with question composed examination and outline, UML expect a fundamental part in portraying trade viewpoints of a structure

**Use case Diagram:**

The use case graph is for demonstrating the direct of the structure. This chart contains the course of action of use cases, performing pros and their relationship. This chart might be utilized to address the static perspective of the structure. The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart ) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows

* Used to gather the requirements of a system.
* Used to get an outside view of a system.
* Identify the external and internal factors influencing the system.
* Show the interaction among the requirements are actors.

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.

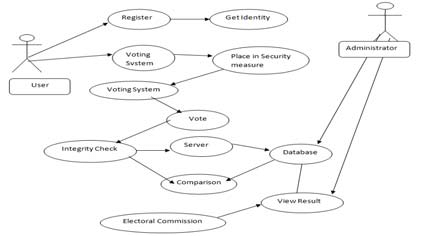
We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

* Functionalities to be represented as use case
* Relationships among the use cases and actors.
* Actors

In the below use case diagram . use cases are used to register, voting system, Get identity, Place in security measure, Database, View results, Server, Integrity check, comparison, Election commission and the actors are the User, Administrator who involved in the below use case diagram

**Use case diagram for online voting system:**



**Class Diagram:**

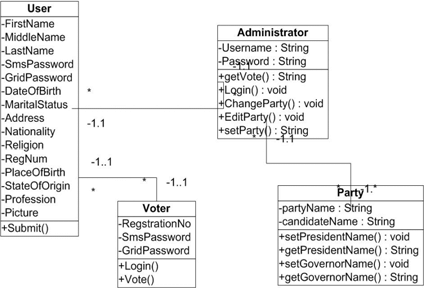
The class graph is the most normally pulled in layout UML. It addresses the static course of action perspective of the structure. It solidifies the strategy of classes, interfaces, joint attempts and their affiliations.

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

* Analysis and design of the static view of an application
* Describe responsibilities of a system.
* Base for component and deployment diagrams.
* Forward and reverse engineering



In the above class diagram, the relationship that is dependence between each one of the classes is sketched out, Additionally, even the operations performed in each and every class is similarly appeared...

**Sequence Diagram:**

This is a cooperation design which tends to the time requesting of messages. It includes set of parts and the messages sent and gotten by the instance of parts. This chart is utilized to address the dynamic perspective of the structure.

A succession outline indicates question communications masterminded in time arrangement. in the above graph, there are five articles cooperating with each other. Each protest has a vertical dashed line which speaks to the presence of a question over some undefined time frame. This graph has additionally a tall, thin rectangle which is called center of control that demonstrates the timeframe amid which a protest is playing out an activity, either specifically or through a subordinate system.

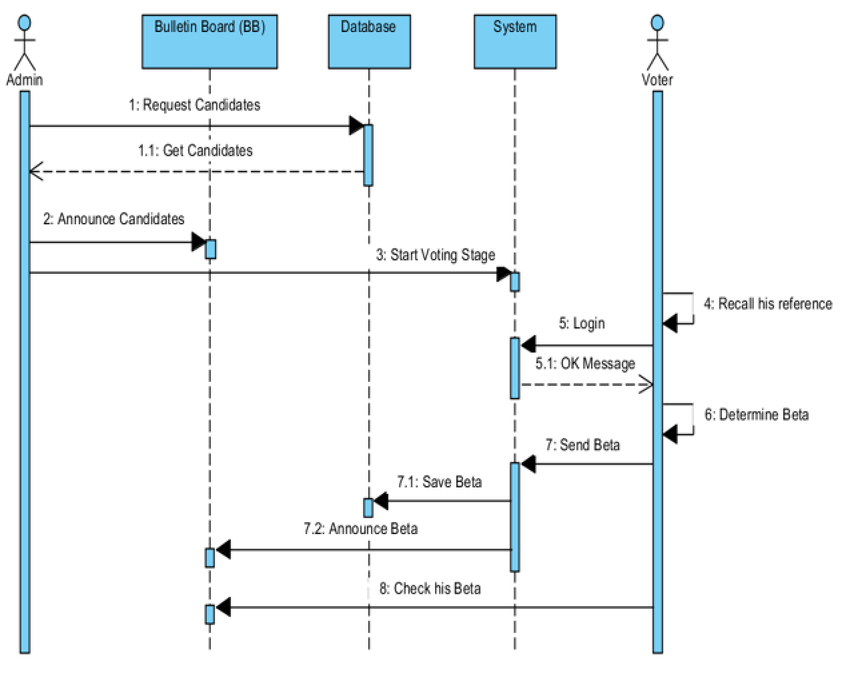
The following scenarios are ideal for using a sequence diagram:

**Usage scenario:** A usage scenario is a diagram of how your system could potentially be used. It's a great way to make sure that you have worked through the logic of every usage scenario for the system.

**Method logic:** Just as you might use a UML sequence diagram to explore the logic of a use case, you can use it to explore the logic of any function, procedure, or complex process.

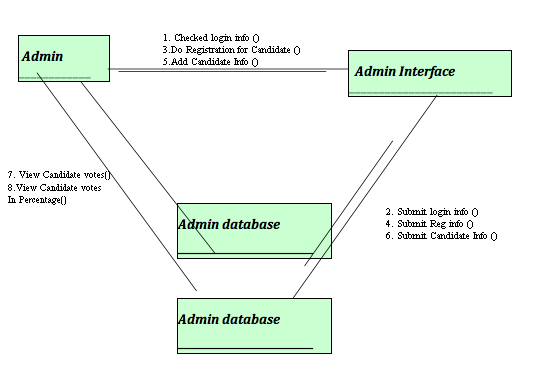
**Service logic:** If you consider a service to be a high-level method used by different clients, a sequence diagram is an ideal way to map that out.

**Sequence diagram Visio** - Any sequence diagram that you create with Visio can also be uploaded into Lucid chart. Lucid chart supports .vsd and .vdx file import and is a great Microsoft Visio alternative. Almost all of the images you see in the UML section of this site were generated using Lucid chart.



In the above sequence diagram messages passed from one object to other objects which starts from the object Admin announces the voting day on that the admin send a request a message for the candidates to login to start their vote. After that voter will login with his username and password then voter enters into candidates page at there voter will make their voter for their candidate in this list of candidates. Once voter submit the information of their voting information is stored in the database.

**Collaboration Diagram:**



This is a support format, which tends to the principal relationship of articles that send and get messages. It incorporates set of parts, connectors that interface the parts and the messages sent and get by those parts. This graph is utilized to address the dynamic perspective of the framework.

The joint effort outline contains articles, way and arrangement number. In the above graph, there are five questions specifically customer, client, framework, Python and server. These items are connected to each other utilizing a way. A succession number show the time request

**State chart Diagram:**

The state graph contains the game-plan of states, occasions and exercises. This graph is noteworthy for tending to the lead of the interface, class and made effort. The key centralization of state outline is to show the occasion sort out lead of the request. The state follows diagram the dynamic perspective of the framework.

A state outline graph contains two components called states and progress. States speak to circumstances amid the life of a question. We can without much of a stretch outline a state in Smart Draw by utilizing a rectangle with adjusted corners. Change is a strong bolt speaks to the way between various conditions of a question. Name the change with the occasion that activated it and the activity those outcomes from it. State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

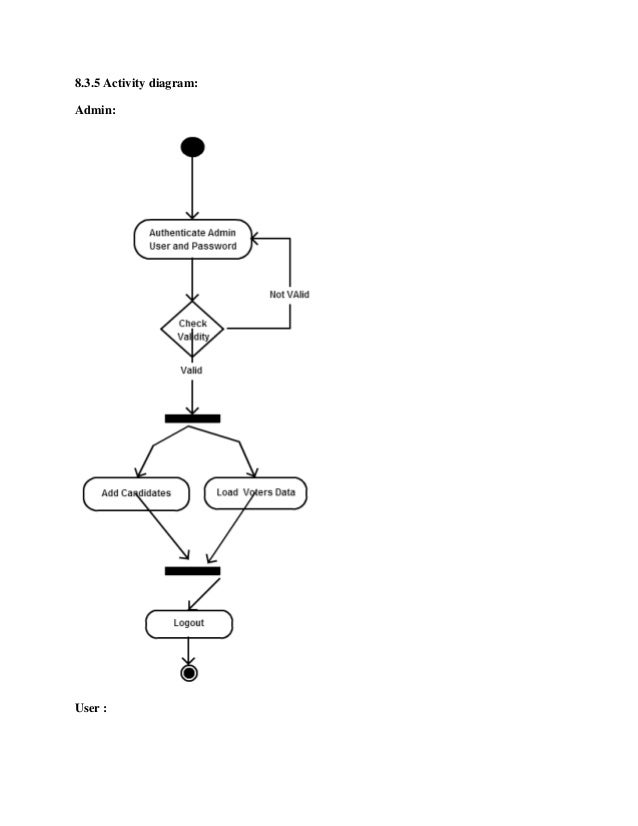
State chart diagram describes the flow of control from one state to another state. States are defined as a condition in w of State chart object exists when some event is triggered. The most important purpose creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using Statechart diagrams –

* To model the dynamic aspect of a system.
* To model the life time of a reactive system
* To describe different states of an object during its life time.
* Define a state machine to model the states of an object.

**Statechart Diagram:**



**Component Diagram :**

The imperative portion of part format is segment. This diagram demonstrates within parts, connectors and ports that understand the piece. Precisely when section is instantiated, duplicates of inside parts are besides instantiated.

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

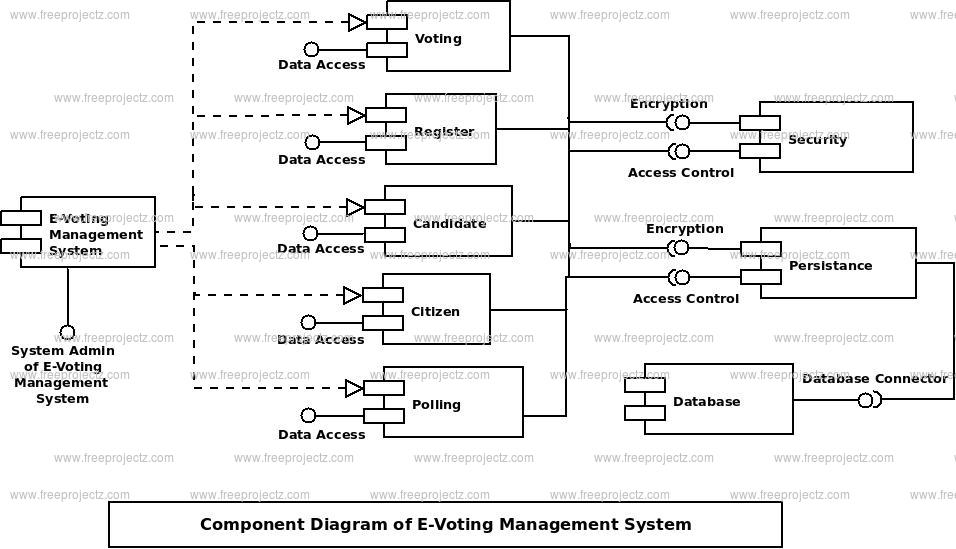
Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as –

* Visualize the components of a system.
* Construct executables by using forward and reverse engineering.
* Describe the organization and relationships of the components.



A part outline is spoken to utilizing segment. A part is a physical building piece of the framework. It is spoken to as a rectangle with tab. Part outline portrays the inward handling of the venture. The information sent to the Python where sqoop is utilized for information cleaning and the reports are produced utilizing hive.

**Deployment Diagram:**

The fundamental fragment in game-plan layout is a middle point. The strategy of focus focuses and their relationship with other is tended to utilizing sending plot. The sending outline is identified with the area diagram, that is one focus purpose obviously of activity format frequently includes no short of what one sections. This outline is in like way critical for tending to the static perspective of the framework.

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

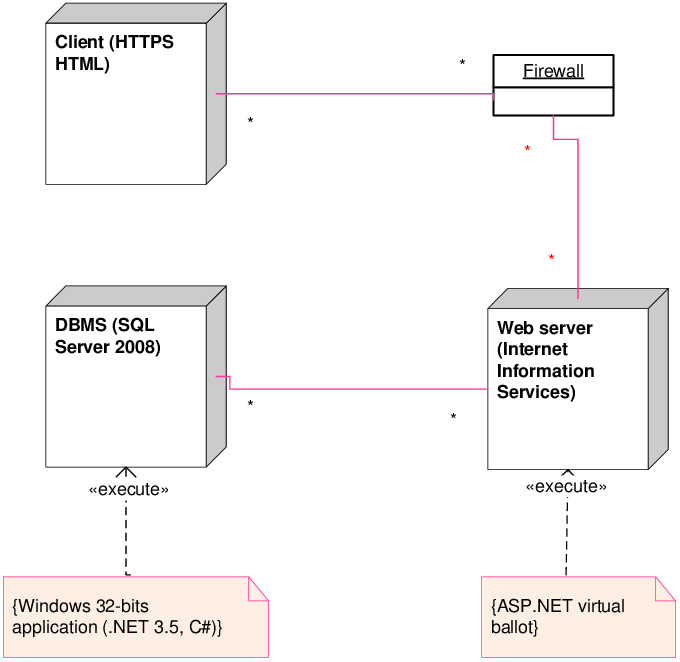
UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UMI. diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as –

* Visualize the hardware topology of a system.
* Describe the hardware components used to deploy software components.
* Describe the runtime processing nodes.

From the below Deployment diagram we can see there is a client and web server and DBMS server which are protected by the firewall. We can execute the our project with windows 32 and other higher version. For user interface we are using HTML, CSS, Javascript and for back end we are using PHP and Mysql for information storage which we are collecting from the user’s and their related information



**CHAPTER-7**