**COMPUTATION AS SIMULATION**

The traditional programming model describes the behavior of a computer executing a program is a process state model. In this view, the computer acts like a data manager, following some pattern of instructions, wandering through memory, pulling out the values from different memory slots, processing them in some manner and pushing back the result back into other memory slots. By examining the values of memory slots, we can determine the state of machine or the result provided by a computation. But this model may be more or less accurate picture to know what is going inside the computer. It does little to help us understand how to solve problem using the computer and also it is not the way most people go about solving problems.

In contrast, in OOP framework we never mention memory addresses, variables, assignments, or any of the traditional programming terms. Instead we use objects, message and responsibility for some action. We have a universe (world) of well- behaved objects that politely ask each other to carry out their various desires. This view is similar in many ways to a style of computer simulation called “discrete event-driven simulation.” In brief, in a discrete event-driven simulation the user creates computer models of the various elements of the simulation, describes how they will interact with one another and sets them moving same as OO programs. Thus, in object-oriented programming, we have the view that computation is simulation.

**Coping with Complexity**

When computing was in its early stages, most programs were written in assembly language, by a single individual. As world technologically being developed, the program became more and more complex, programmer found it difficult to remember all the information they needed to know in order to develop or debug their software. Which values were contained in what register/memory slots? Did a new identifier name conflict with any previously defined name? What variable needed to be initialized before control could be transferred to another section of code?

The introduction of high-level language like FORTRAN, Cobol and Algol solved some difficulties while simultaneously raising people’s expectations of what a computer could do. As programmers attempted to solve ever more complex problems using a computer, tasks exceeding the grasp of even the best programmers became the norm. Thus, team of programmers working together on major programming efforts became common place. So that, by dividing the tasks into smaller units and allows the programmer groups to invest their development efforts unit-wise. This is called divide and conquer techniques, somehow provides the mechanism of coping with complexity.

**Abstraction Mechanism**

            If you see the map of the world in the atlas, obviously the map will show some significant feature of the earth such as oceans, continents and other extremely large structures. But small features such as the major cities, rivers, states of a country etc. are omitted in the map of the world. These things are mentioned in the map of the subsequent countries and again, the smaller things such roads, buildings etc. have been omitted even in them. So, in each level of maps in the atlas, some information’s are purposely omitted and some information are included.

Thus, abstraction is the purposeful suppression, or hiding, of some details of a process or artifact, in order to bring out more clearly other aspects, details, or structures.

            Information hiding is the purposeful omission of details in the development of an abstract representation. So, it describes the part of abstraction in which we intentionally choose to ignore some features so that we can concentrate on other.

**Layers of abstraction:**

            In an object-oriented program, we find the various levels of abstraction. At the highest level of abstraction, a program is viewed as a ‘community’ of objects where these objects interact with each other in order to achieve their common goal. In object-oriented development, there are two different form of community:

1. community of programmer: - In the community of programmer, the members interact with each other in the real world in order to produce their application.

            b) community of objects: - In the community of objects that programmers create, they interact with each other in a virtual universe in order to further their common goals.

Client and server are the two levels of abstraction which deal with the interaction between the two individual objects in the program. Server is the object which provides services to the other objects in the program whereas the client is the object which requests for the service to the server and gets them from server.

**Forms of abstraction**

**Is-a and Has-a Abstraction**

Generally, abstraction is divided into two forms: abstraction division into parts and abstraction division into specialization.

            The form of abstraction which is based on division into specialization is also known as ‘Is-a Abstraction’. For example, a car ‘is-a’ wheeled vehicle, which is turn ‘is-a’ means of transportation, dog ‘is-a’ type of animal etc.

            The form of abstraction which is based on division into parts is also known as ‘Has-a Abstraction’. For example, a car ‘has-a’ engine, a bicycle ‘has-a’ wheels., a dog ‘has-a’ tail etc.

**Nonlinear behavior of complexity**