Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Object Oriented Designing and Modeling

[1]Sheetal K. Thakkar, [2]Kinjal K. Thakkar, [3]Nirav M. Satra
[1]Computer Technology Department, Shah & Anchor Kutchhi Polytechnic, Chembur, Mumbai, Maharashtra, India
[2]Systems Engineer, Infosys Limited, Phase - II, Pune, Maharashtra, India
[3]Computer Technology Department Shah & Anchor Kutchhi Engineering College, Chembur, Mumbai, Maharashtra, India

Abstract–System Designing and Modeling have been a vital part of Software Development Life Cycle (SDLC). There are various approaches for designing and modeling of a system. Two of the most common approaches are: Structured approach and Object-Oriented (OO) approach. The concepts of Structured and Object-Oriented analysis are not comparatively new but these approaches still play a very significant role in the present programming paradigm. Systems being developed presently are both complex and concurrent. Hence, in order to cope up with this, inclination towards Object-Oriented methodology in analysis and design of a system has aroused in recent years. This electronic document thus focuses on the unique features of structured analysis and OO analysis and designing. Besides, to best fit the OO concepts with the existing coherent discussion of structured approach, still faces a challenge. This paper also addresses some of the associated aspects of these design paradigms and offers depiction of Object-Oriented design methods used to develop a software system or any non-software system to replicate its functioning and behavior.

Keywords – Object-Oriented, Software Development Life Cycle (SDLC), Complexity, Unified Modeling Language (UML), Structured development.

I. INTRODUCTION

In this technical era, the systems (Software and Non-Software Systems) to be developed are turning to be complex. Due to this, designing and analysis have become a crucial part of Software Development. Modeling and designing (modeling and simulation) means getting information about something so as to know how it will behave without actually implementing and testing it in real life. It is a discipline on its own. Its many application domains often lead to the assumption that modeling and simulation is pure application. To ensure that the results of simulation are applicable to the real world, an engineer must understand the assumptions, conceptualizations, and implementation constraints of this emerging field [1].

Any complicated journey can be simplified if a map exists. Developing a system is also a complicated journey. Whether it is a landscaper, a bridge builder, an aeronautical engineer, a carpenter, or an architect, they work with models every day. Create a "sketch" of the thing so that it is easy to understand the big picture – what it will look like architecturally, how constituent parts fit together, and many other characteristics. An engineer does the same thing by creating models to better understand the requirements and the design that will achieve those requirements [2].

II. APPROACHES FOR DESIGNING AND ANALYSIS

The System Development Life Cycle (SDLC) can be defined as a process of understanding how an information system can support business needs or requirements of an organization, modeling the business processes, designing the system components, and building the system.

The two most common approaches for system modeling and analysis are:

A. Structured approach: Structured methodology is one of the traditional approaches of SDLC. It adopts a formal step by step flow. The structured approach looks at a system in a top-down view. In the context of traditional engineering, a component is a functional element of a program that incorporates processing logic, the internal data structures that are required to apply the processing logic, and an interface that enables the component to be invoked and data to be passed to it [2]. The center of the structured approach is the process model, which depicts the business processes of a system e.g. Waterfall model, Spiral model and etc. The primary model that presents the processes is Data-flow diagram (DFD). The DFDs and their allied data dictionary contain the information about the system components that need to be designed and ultimately built. This approach has been for many years and it is also known as traditional approach or procedural approach [3].

B. Object-Oriented Approach: As computer professionals, people strive to build system that are useful and that work; as software engineers, people come across the task of creating complex system with less computing and human resource. Besides, the present time is considered as a world of objects. These objects exist in nature, in man-made entities, in business, and in the products that we use. They can be categorized, described, organized, combined,

manipulated and created. Therefore, an object-oriented view has come into picture for creation of such systems. An object-oriented approach to the development of software was proposed in late 1960s.[7]

Over the past few years, object-oriented technology has evolved in diverse segments of the computer sciences as a means of managing the complexity inherent in many different kinds of systems. The object-oriented models have proven to be a very powerful and unifying concept. [4] It is most commonly used by software engineering professionals who deal with large and complex systems in domains such as aerospace, process control, Artificial Intelligence and etc. Object-Oriented development requires that object-oriented techniques be used for the analysis and implementation of the system. This methodology asks the analyst to find out what the objects of the system are, how they behave over the time or in response to events, and what responsibilities and relationships (links) an object has to other objects. Moreover, Object-oriented analysis demands inspection of all the objects in a system, their similarities, differences, and how the system needs to handle these objects.

The Object-Oriented methodology views a system as a bottom-up approach to system developments. To start with, it describes the system through a set of business processes that it performs as well as object classes that these processes deal with. It starts from the lowest module moving towards the main (root) component.

An object contains both data and methods (operations) that control data. The data represents the state of the object. A class describes an object and they also form hierarchy to visualize the real world system. The hierarchy is often termed as inheritance in the OO world [5].

In context of Object-Oriented software engineering, a component contains a set of collaborating classes. Each class within a component has been fully elaborated to include all attributes and operations that are appropriate to its implementation. As part of design elaboration, all interfaces that enable the classes to communicate and collaborate with other design classes must also be defined. To accomplish this, a person (engineer) often begins with the requirement model and elaborate analysis classes (for components that relate to the problem domain) and infrastructure classes (for components that provide support services for the problem domains) [2].

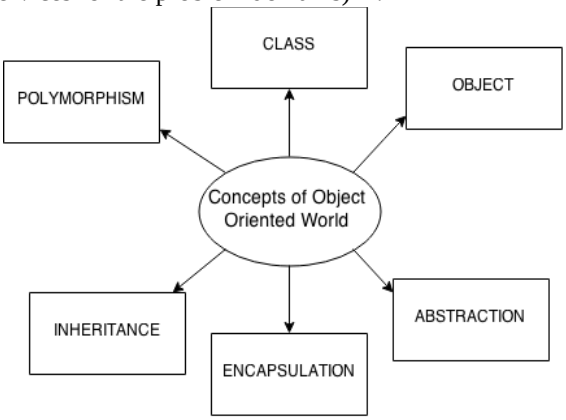

Figure 1. Basic concepts of Object-oriented world

III. BASIC TERMINOLOGIES OF OBJECT-ORIENTED WORLD

Following are some fundamental concepts related to the Object Oriented world:

A. Class: A class is a blue-print of an object. [5] It describes (defines) the features that an object of that type contains. It can also be defined as any uniquely identified abstraction of a set of logically related instance or objects that share similar characteristics. In short, a class is a collection of objects.

B. Object: Objects represent a real world entity or basic building blocks. Basically, Objects are representative or an instance of a class.

C. Abstraction: Abstraction represents the behavior of a real world entity. It means hiding the background details and explanation; and representing only essential details to the outside world.

Data abstraction is a methodology that enables us to isolate how a compound data object is used from the details of how it is constructed from more primitive data objects. Data abstraction is the enforcement of a clear separation between the abstract properties of a data type and the concrete details of its implementation. Data Abstraction is simplifying complex reality by modelling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem. Abstraction is also achieved through Composition. For example, a class Car would be made up of an Engine, Gearbox, Steering objects, and many more components. To build the Car class, one does not need to know how the different components work internally, but only how to interface with them, that is send messages to them, receive messages from them, and perhaps make the different objects composing the class interact with each other.[6]

D. Encapsulation: Encapsulation is the mechanism of binding the data together and hiding them from outside world. In other words, Encapsulation is the ability to package codes and data together in a place and hide (or prevent) that data from external contact thereby forcing anyone who wants to access it to pass through the associated code. Structured programming encourages code everywhere to deal directly with data structures. [6]

E. Inheritance: Inheritance is the mechanism of making new classes from existing one. Inheritance is when an object or class is based on another object or class, using the same implementation (inheriting from a class) or specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior). It is a mechanism

for code reuse and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a hierarchy.

F. Polymorphism: It defines the mechanism to exist in different forms. [5]More specifically, it is the ability to redefine methods for derived classes. For example, given a base class shape, polymorphism enables the programmer to define different area methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the *area* method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).

IV. OBJECT ORIENTED PROCESS

The Object Oriented Methodology of building systems takes objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example in case of a Banking System, a customer is an object, a cheque book is an object, and even an account is an object. [7]

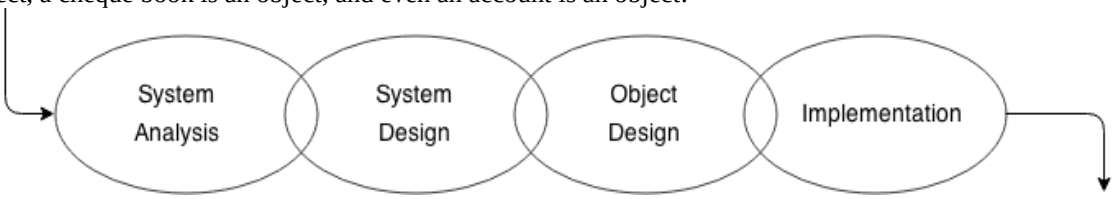

Figure 2. Object-Oriented process

In simple terms, Object Modeling is based on identifying the objects in a system and their inter-relationships and interactions with one another. Once this is done, the coding of the system is done. Object Modeling is somewhat similar to the traditional approach of system designing, in that it also follows a sequential process of system designing but with a different approach. The basic steps of system designing using Object Modeling may be explained as:

A. System Analysis: As in any other system development model, system analysis is the first phase of development in case of Object Modeling too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning. During object oriented analysis the most important purpose is to recognize objects and describing them in a proper way. If these objects are identified efficiently then the next job of design is easy. The objects should be identified with responsibilities. Responsibilities are the functions performed by the object. Each and every object has some type of tasks to be performed. When these responsibilities are collaborated the purpose of the system is fulfilled. [5]

Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized. [7]

B. System Design: System Design is the next progress stage where the overall architecture of the desired system is decided. The system is organized as a set of sub systems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user.

As the basic idea of Object-Oriented method of system analysis is to perceive the system as a set of interacting objects, a bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system as in the case of traditional Waterfall Model where the processes form the important part of the system.

C. Object Design: In this phase, the information of the system analysis and system design is implemented. The Objects identified in the system design phase are designed. Here the implementation of these objects is decided as the data structures get defined and also the inter-relationships between the objects are defined.

D. Implementation: The last phase is Object Oriented implementation. In this phase the design is implemented using Object Oriented languages like Java, C++ and etc.

V. IMPORTANCE OF OBJECT ORIENTED APPROACH

The purpose of Object-Oriented approach is explained below:

- Object Oriented Methodology closely represents the problem domain i.e. simplified mapping to real world. Because of this, it is easier to produce and understand designs. The objects in the system are resistant to requirement changes. Therefore, allows changes more easily.
- Object Oriented Methodology designs encourage more re-use. New applications can use the existing modules, thereby reduces the development cost and cycle time. It is the reuse of the artifacts or objects that are independent of the analysis method or programming language.
- Object Oriented Methodology approach is more natural. It provides nice structures for thinking and abstracting and leads to modular design.[7]
- Besides, it helps us to identify objects of a system to be designed and identify their relationship. Make a design that can be converted to executables using OOP language.

VI. UNIFIED MODELING LANGUAGE

UML is often a buzz word in the Object-Oriented world. Most people refer to the Unified Modelling Language as UML. The UML is an international industry standard graphical notation for describing software analysis and designs. UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. UML is different from the other common programming languages like C++, Java, COBOL and etc. UML is a pictorial language used to make system/software blue prints. So UML can be described as general purpose visual modelling language to visualize, specify, construct and document software system or any other system.

One of the purposes of UML was to provide the development community with a stable and common design language that could be used to develop and build computer applications. UML brought forth a unified standard modelling notation that IT professionals had been wanting for years. Using UML, IT professionals could now read and disseminate system structure and design plans -- just as construction workers have been doing for years with blueprints of buildings.

It is now the twenty-first century -2015 to be specific and UML has gained grip in the technical field. In most of the Curriculum Vitae (CVs) there is a highlighting point claiming knowledge of UML. [8]

As mentioned, UML was intended to be a unifying language enabling IT professionals to model computer applications. The primary authors were Jim Rumbaugh, Ivar Jacobson, and Grady Booch, who originally had their own competing methods (OMT, OOSE, and Booch). Eventually, they joined forces and brought about an open standard. One reason UML has become a standard modelling language is that it is programming-language independent. (UML modelling tools from IBM Rational are used extensively in J2EE shops as well in .Net shops). Also, the UML notation set is a language and not a methodology. This is important, because a language, as opposed to a methodology, can easily fit into any company's way of conducting business without requiring change.

Since UML is not a methodology, it does not require any proper work products. Yet it does provide several types of diagrams that, when used within a given methodology, increase the ease of understanding an application under development. There is more in UML than these diagrams, but here, the diagrams propose a good introduction to the language and the principles behind its use. By placing standard UML diagrams in ones methodology's work products, this makes it easier for UML-proficient people to connect (join) your project and quickly become productive. [8]

UML focuses on various views of a system. They are described below:

A. Functional View: This view describes functional requirements of the system. Use case diagrams give static functional view for functions and static relationships. Activity diagrams give dynamic functional view.

B. Static Structural View: The class and object diagrams give the structural view of system.

C. Behavioural (dynamic structural) view: Interaction diagrams, collaboration (CRC) diagrams and sequence diagrams describe sequences of interactions between objects. State transition diagrams show state-based behaviour of objects.

D. Architectural View: This view describes logical and physical structure. Component diagrams and deployment diagrams are used in this view. [7]

VII. PURPOSE OF UML DIAGRAMS

A picture is worth a thousand words, this absolutely fits while discussing about UML. Object oriented concepts were introduced much earlier than UML. So at that time there were no standard methodologies to organize and strengthen the object oriented development. At that point of time UML came into picture.

There are a number of goals for developing UML but the most important is to define some general purpose modelling language which all modellers can use and also it needs to be made simple to understand and use.

UML diagrams are not only made for developers but also for business users, common people and anybody interested to understand the system. The system can be a software or non-software. So it must be clear that UML is not a development method rather it accompanies with processes to make a successful system.

At the conclusion the goal of UML can be defined as a simple modelling mechanism to model all possible practical systems in today's complex environment. [5]

VIII. UML DIAGRAMS

Diagrams are the heart of UML. These diagrams are broadly classified as structural and behavioural diagrams. Structural diagrams are consists of static diagrams like class diagram, object diagram etc. Behavioural diagrams are consists of dynamic diagrams like sequence diagram, collaboration diagram etc.

The static and dynamic nature of a system is visualized by using these diagrams. The different types of UML diagrams are:

A. Class Diagram: Class diagrams are the most popular UML diagrams used by the object oriented community. It describes the objects in a system and their relationships. Class diagram consists of attributes and functions. A single class diagram describes a specific aspect of the system and the collection of class diagrams represents the whole system. Basically the class diagram represents the static view of a system. [7] The class diagram shows how the different entities (people, things, and data) relate to each other; in other words, it shows the static structures of the system. A class diagram can be used to display logical classes, which are typically the kinds of things the business people in an organization talk about -- rock bands, CDs, radio play; or loans, home mortgages, car loans, and interest rates. Class diagrams can also be used to show implementation classes, which are the things that programmers typically deal with. An implementation class diagram will probably show some of the same classes as the logical classes' diagram. The execution class diagram won't be drawn with the same attributes, however, because it will most likely have references to things like Vectors and HashMaps.[8]

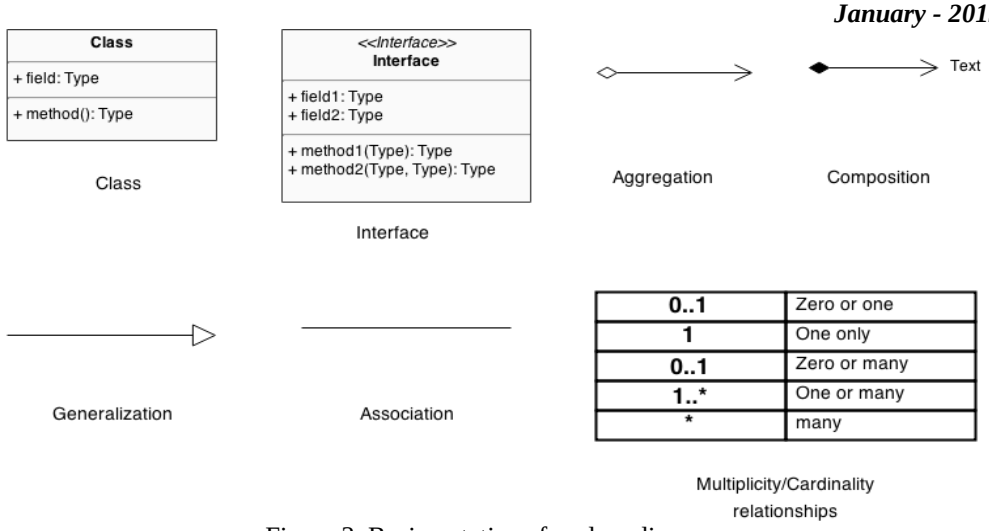

Figure 3. Basic notations for class diagram

A class is represented on the class diagram as a rectangle with three horizontal sections, as shown in Figure. The upper section shows the class's name; the middle section contains the class's attributes; and the lower section contains the class's operations (or "methods").

B. Object Diagram: As an object is an instance of a class similarly an object diagram is an instance of a class diagram. So the basic elements are similar to a class diagram. Object diagrams are consists of objects and links. It captures the instance of the system at a particular moment. Object diagrams are used for prototyping, reverse engineering and modeling practical scenarios. The object is represented in the same way as the class. The only difference is the name that is underlined.

C. Use Case Diagram: A use case illustrates a unit of functionality (basic functionality) provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use-case diagrams generally show groups of use cases -- either all use cases for the complete system, or a breakout of a particular group of use cases with related functionality (e.g., all security administration related use cases). To show a use case on a use-case diagram, you draw an oval in the middle of the diagram and put the name of the use case in the center of, or below, the oval. To draw an actor (indicating a system user) on a use-case diagram, you draw a stick person to the left or right of your diagram.

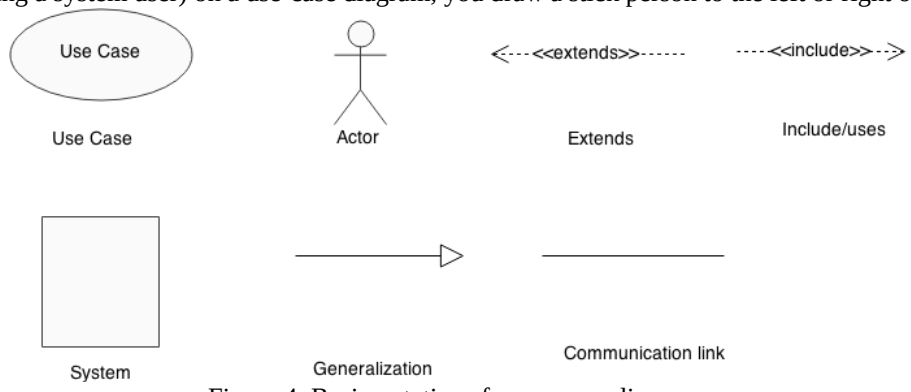

Figure 4. Basic notations for use case diagram

Use case diagram is used to capture the dynamic nature of a system. It consists of use cases, actors and their relationships (include, extend and etc.). Use case diagram is used at a high level design to capture the requirements of a system. So it represents the system functionalities and their flow. Although the use case diagrams are not a good candidate for forward and reverse engineering but still they are used in a slightly differently way to model it.

D. Sequence Diagram: Sequence diagrams show a detailed flow for a specific use case or even just part of a specific use case. They are almost self-explanatory; they show the calls between the different objects in their sequence and can show, at a detailed level, different calls to different objects.A sequence diagram has two dimensions: The vertical dimension shows the sequence of messages/calls in the time order that they occur; the horizontal dimension shows the object instances to which the messages are sent.

A sequence diagram is not much difficult to draw. Across the top of the diagram, identify the class instances (objects) by putting each class instance inside a box. In the box, put the class instance name and class name separated by a space/colon/space ": " (e.g. mango:Fruit). If a class instance sends a message to another class instance, draw a line with an open arrowhead pointing to the receiving class instance; place the name of the message/method above the line. Optionally, for important messages, you can draw a dotted line with an arrowhead pointing back to the originating class instance; label the return value above the dotted line.

E. Collaboration Diagram: A collaboration diagram, also called a communication diagram or interaction diagram, is a representation of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.
A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing. Collaboration diagrams are best suited to the portrayal of simple interactions among fairly small numbers of objects. As the number of objects and messages grows, a collaboration diagram can become difficult to read. Several vendors offer software for creating and editing collaboration diagrams. [9]

F. Statechart Diagrams: The statechart diagram depicts the different states that a class can be in and how that class transitions from state to state. It can be argued that every class has a state, but that every class shouldn't have a statechart diagram. Only classes with "interesting" states -that is, classes with three or more potential states during system activity should be modeled.

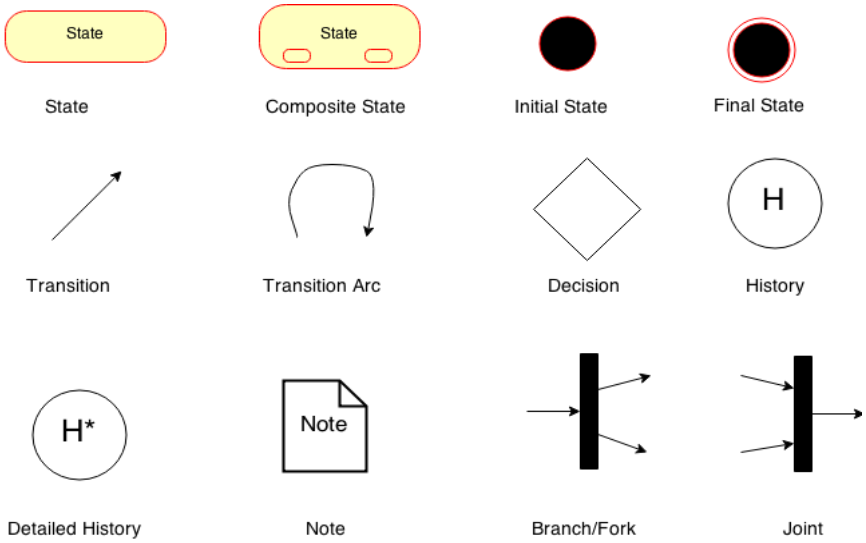

Figure 5. Basic notations for statechart diagram

The notation set of the statechart diagram has five basic elements: the initial starting point, which is drawn using a solid circle; a transition between states, which is drawn using a line with an open arrowhead; a state, which is drawn using a rectangle with rounded corners; a decision point; and one or more termination points, which are drawn using a circle with a solid circle inside it. To draw a statechart diagram, begin with a starting point and a transition line pointing to the initial state of the class. Draw the states themselves anywhere on the diagram, and then simply connect them using the state transition lines.

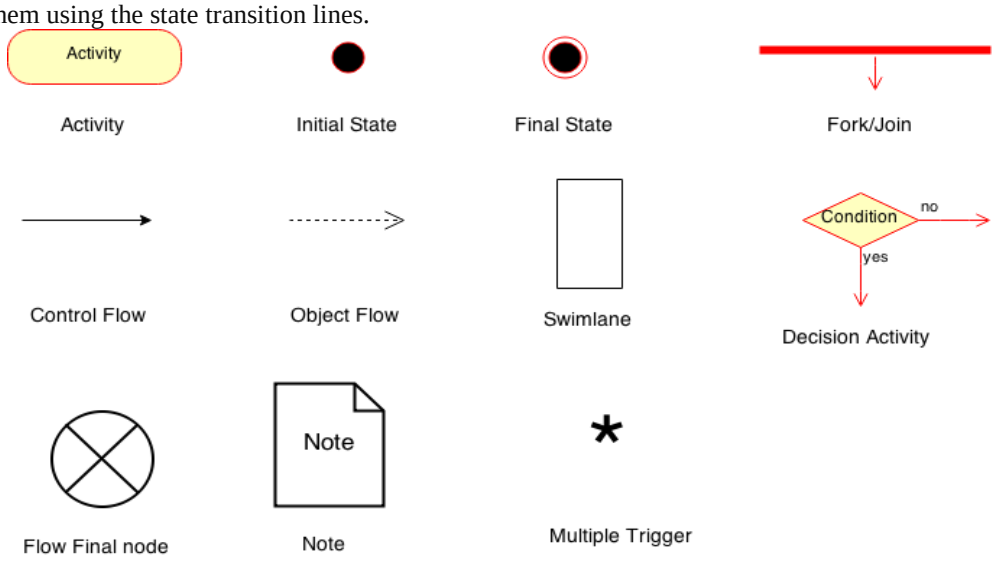

Figure 6. Basic notations for activity diagram

G. Activity Diagrams: Activity diagrams show the procedural flow of control between two or more class objects while processing an activity. Activity diagrams can be used to model higher-level business process at the business unit level, or to model low-level internal class actions. In my experience, activity diagrams are best used to model higher-

level processes, such as how the company is currently doing business, or how it would like to do business. This is because activity diagrams are "not as much of technical" in appearance, compared to sequence diagrams, and business-minded people tend to understand them more quickly. An activity diagram's notation set is similar to that used in a statechart diagram. Like a statechart diagram, the activity diagram starts with a solid circle connected to the initial activity.

The activity is displayed by drawing a rectangle with rounded edges, enclosing the activity's name. Activities can be connected to other activities through transition lines, or to decision points that connect to different activities guarded by conditions of the decision point. Activities that terminate the modelled process are connected to a termination point (just as in a statechart diagram). [7]Activity diagrams can also be classified as simple activity diagram and swimlane diagram.

H. Component Diagram: A component diagram provides a physical view of the system. Its purpose is to show the dependencies that the software has on the other software (e.g., software libraries) in the system. The diagram can be shown at a very high level, with just the large-grain components, or it can be shown at the component package level.[10]

I. Deployment Diagram: The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other. Since the diagram models the physical runtime, a system's production staff will make considerable use of this diagram.

The notation in a deployment diagram includes the notation elements used in a component diagram, with a couple of additions, including the concept of a node. A node represents either a physical machine or a virtual machine node (e.g., a mainframe node). To model a node, simply draw a three-dimensional cubewith the name of the node at the top of the cube. Use the naming convention used in sequence diagrams: [instance_name]:[instance_type](e.g.,"OOA:Book").

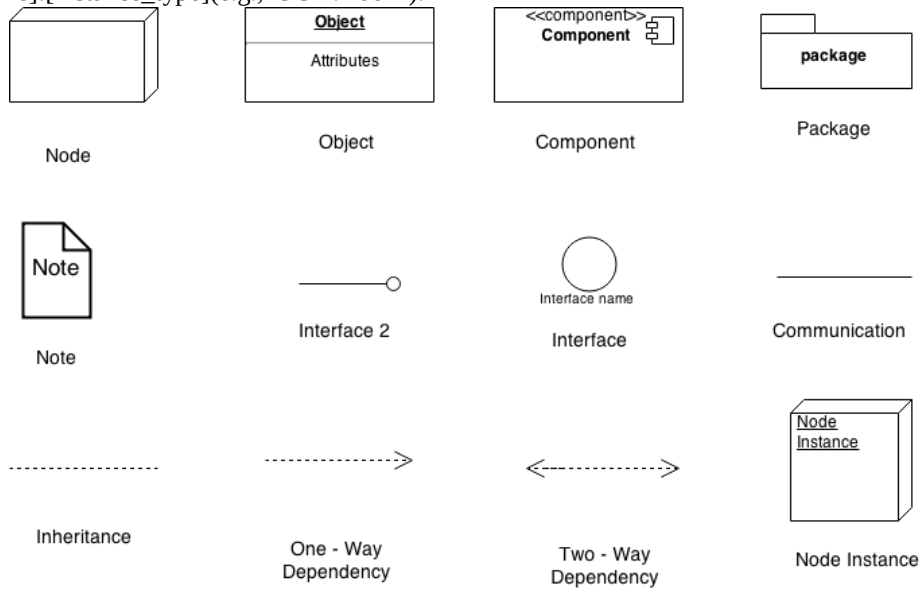

Figure 7. Basic notations for deployment diagram

IX. CONCLUSION

Hence by this, it can be concluded that Designing and modeling simplify the task of developing a system. Object-Oriented approach for SDLC has gained importance in the Engineering field. Although in most of the resumes, there is a highlighting point claiming knowledge of UML. However, it becomes clear that they do not actually know much about UML, after discussing with a majority of these job applicants. Typically, some these candidates either use it as a buzz word, or have had a gist to UML. Besides, even employers look for graduates (engineers) having familiarity with Object-Oriented concepts. These all things have made a necessity to learn Object-Oriented approach for people in the engineering field.

There are a number of software tools that help you to incorporate UML diagrams into your software development process; even without automated tools it is not a tough task to utilize Object-Oriented methodology in the development of any proposed system.

REFERENCES

[1] http://en.wikipedia.org/wiki/Modeling_and_simulation
[2] Roger S. Pressman, “Software Engineering a practitioner’s approach”, McGraw-Hill, 7th edition.
[3] https://mis.uhcl.edu/rob/Professional/Publications/Dilemma%20of%20Structured%20and%20OO%20Methodology.pdf
[4] Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, "Object-Oriented Analysis and Design with Applications", Pearson Education.Inc, 2nd edition.

[5] http://www.tutorialspoint.com/uml/uml_tutorial.pdf
[6] http://www.bioline.org.br/pdf?ja08064.pdf
[7] http://www.mu.ac.in/myweb_test/MCA%20study%20material/M.C.A.(Sem%20_%20IV)%20Object%20Oriented%20Modeling%20&%20Design%20Using%20UML
[8] http://www.nyu.edu/classes/jcf/g22.2440-001_sp09/handouts/UMLBasics.pdf
[9] http://searchsoftwarequality.techtarget.com/definition/collaboration-diagram
[10] The phrase component package level is a programming language-neutral way of referring to class container levels such as .NET's namespaces (e.g., System.Web.UI) or Java's packages (e.g., java.util).
[11] eprints.eemcs.utwente.nl/10667/01/object-oriented-analysis-structured.pdf
[12] cc.ee.ntu.edu.tw/~fom/courses/BCC/NTUEE/2012.spring/uml_tutorial.pdf

