An Overview of Structural UML Diagrams

Bhavik Bhatt[1], Muskaan Nandu[2]

[1]Bhavik Bhatt, Dept. of Information Technology, KJ Somaiya College of Engineering, Maharashtra, India
[2]Muskaan Nandu, Dept. of Information Technology, KJ Somaiya College of Engineering, Maharashtra, India

---

Abstract - *UML stands for Unified Modeling Language. UML diagrams are used to graphically visualize an abstract model of a system. The paper includes a brief introduction of UML diagrams and explains the two types of UML diagrams - Structural UML Diagrams and Behavioral UML Diagrams. This paper is an overview of Structural UML diagrams. We have discussed 5 Structural diagrams namely: Class diagram, Object Diagram, Component Diagram, Deployment diagram and Package Diagram. We have also presented the results of a survey which explains how the Class Diagram is the most important and most used Structural Diagram.*

***Key Words*: UML, Unified Modeling Language, Structural diagrams, Class Diagrams, Component Diagrams, Deployment Diagrams, Object Diagrams, Diagrams.**

1. INTRODUCTION

UML in software engineering stands for Unified Modelling Language. It is a specification language which is used for visualizing and creating an abstract model of the final product. These diagrams are a very important part of the modeling and design process step of any project. A study was offered in a paper which showed how different data tables, diagrams, graphical representations, maps and drawings help in visualizing and analyzing a subject better. The primary purpose of the study mentioned is to understand the use of visual inscriptions in science, by providing various analysis results [1].
Mainly, Unified Modeling Language has been used as a specification language or a general-purpose modeling language. Since, software development is a long process including several phases, it may get complex and hence representing each phase with different UML diagrams plays a vital role in easing the complexity of the implementation process. As we represent each phase with different diagrams, certain basic rules should be followed while drawing such diagrams to maintain the consistency between different UML diagrams. Certain rules have been proposed with relevant examples in Mohammad N, Alanzi's paper to build correct and consistent UML diagrams [2].
UML diagrams have found huge success for documentation of workflow of different business processes. UML helps in providing a visual representation or a blueprint of the steps of implementation to be performed in any large scale projects. This paper provides an overview of Structural UML diagrams. The types of UML diagrams are discussed in this paper followed by an in-depth overview of a few structural diagrams. The significance of using Structural UML diagrams is explained by presenting the results of a survey on using some of the popular structural diagrams.

2. TYPES OF UML DIAGRAMS

To write in a standard way, The UML diagrams give us a method to write a system's overview, defining all the phases involved in the business process like database schemas, the system functions, and in some cases even an overview of how classes are written in certain programming languages. The UML diagrams are divided into two types as shown in Fig 1.

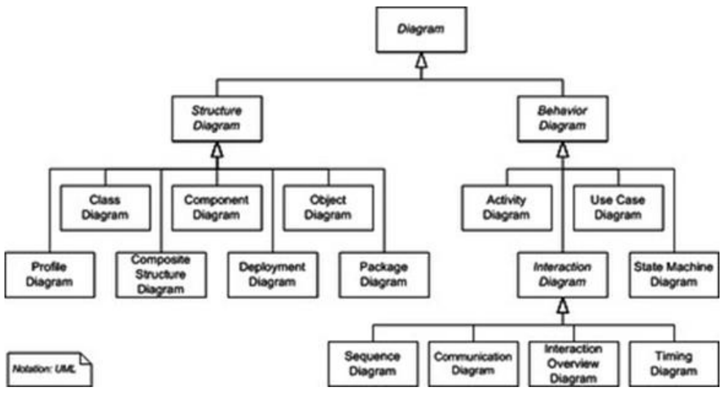


Fig -1: Types of UML Diagrams

The two types of UML diagram as explained in [3] are:

a. Structural UML Diagrams

Structural UML diagrams show the static aspect of modeling. These diagrams include: the composite structure diagram, Deployment diagram, Package Diagram, Profile Diagram, Class Diagram, Object Diagram and Component diagram. Some of these diagrams are explained in detail in this paper. These diagrams show us the main structure of the system using different elements of the system like its class, interfaces, objects, components and nodes. Thus, they show the things of a modeled system.

b. Behavioural UML diagrams

Behavioural diagrams represent the dynamic aspect of the system. They show how different elements of the system interact with each other. These diagrams include: State Machine diagrams, Communication diagrams, Usecase diagrams, Activity diagrams, Sequence diagrams, Timing diagram and Interaction diagrams. These diagrams mainly depict the changing parts of the system

This paper focuses on the Structural UML Diagrams.

3. STRUCTURAL DIAGRAMS

3.1 Class Diagrams

Class diagrams are static diagrams which help a user in constructing the executable code of the software [3]. The main purpose of class diagrams is to describe responsibilities of the system, to analyze and design the view of the application, forms a basis for component and deployment diagrams and also plays an important role in forward and reverse engineering. Class diagrams are primarily used to show how different types of elements which include a class, an interface, a data type and a component, are being modeled within the system. Class diagrams are a boon because developers might think the class diagram was created specifically for them but other team members like business analysts might find them useful too. [4]

Class diagrams can be mapped to structural coding; thus, one should ensure that each element and their relationships are identified in advance. The responsibilities must be well distributed. In class diagrams the three essential elements are: class name, attributes and operations. The classes are defined using rectangles with their name written on the top-center of the rectangle. This class name should be meaningful. This rectangle is divided into half, where the upper half contains all the attributes of the class and the lower half contains all the operations as shown in Fig 2.

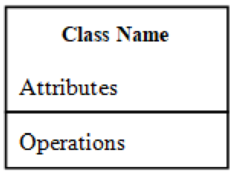


Fig -2: Structure of Class in Class Diagrams

There are different types of relationships or associations between classes offered by the UML class diagram which serve as a rudimentary stage for the making of several other statements of object-oriented philosophy. Class visibility, usage, aggregation, dependency, generalization, etc. are included in these associations.

Class Diagrams depict the system's elements in order to have an idea about the main components of the system and how they interact with each other to fulfill a common goal. In order to extract the coding structure, Class Diagrams should be drawn without any errors. The attributes and functions should be defined properly as per the guidelines. A tool is mentioned in Sadia Sadaf, Ali Athar and Farooque Azam's paper: "Evaluation of FED-CASE - A Tool to Convert Class Diagram into Structural Coding", which has two main modules of forward and reverse engineering [5]. When talking about converting a class diagram to structural coding, component mapping is to be done. The entity name in class diagram is mapped to class name in structural coding, attributes in class diagrams are mapped to variables in coding, operations in class diagrams to Functions in coding.

An example is shown using the figures below where the Example Class in Fig 3 is an entity in the class diagram which is mapped to its equivalents in structural coding in Fig 4 [6].

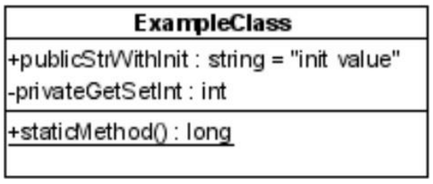
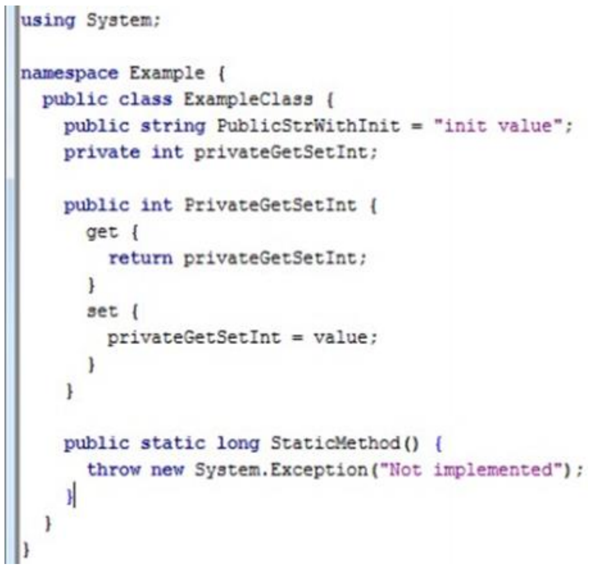


Fig -3: Example Class in Modeling



```
using System;

namespace Example {
  public class ExampleClass {
    public string PublicStrWithInit = "init value";
    private int privateGetSetInt;

    public int PrivateGetSetInt {
      get {
        return privateGetSetInt;
      }
      set {
        privateGetSetInt = value;
      }
    }

    public static long StaticMethod() {
      throw new System.Exception("Not implemented");
    }
  }
}
```

Fig -4: Example Class in Programming language

Thus, class diagrams are beneficial for both stakeholders and developers as it provides a graphical semantic of an application for its better understanding. An example of a class diagram for the following problem statement is shown below in Figure 5 [7].

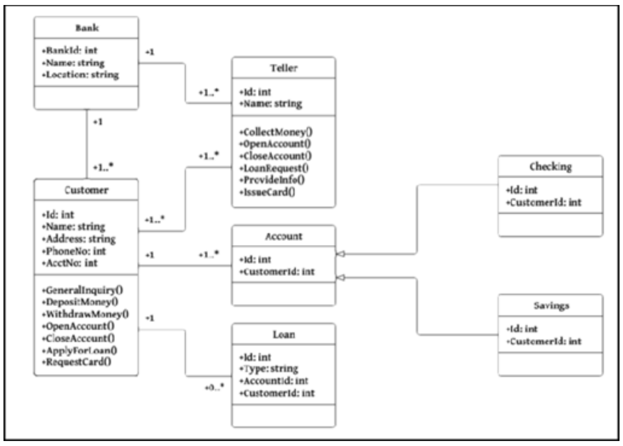


Fig -5: Example of Class Diagram for Banking System

3.2 Object Diagram

The Object UML diagram, also known as Object-Oriented diagram, is used to represent the static view of a system and portray an instance of a UML class diagram. To make objects and their relationships as an instance, Object diagrams are used. It represents structural and behavioural aspects of the system with respect to an actor. Rudimentarily object diagrams and class diagrams are similar to each other.

However, as mentioned earlier Object diagram represents an instance at a particular point in time which is solid in nature whereas the Class diagram represents an abstract model including classes and their relationships. Intuitively object diagrams are very close to the actual system behaviour.
The basic purposes of object diagrams are as follows:
- Forward and reverse engineering
- Static View of an interaction
- Object relationships of a system
- Understand from practical perspective the object behaviour and their relationships

Shown below is an example of Object diagram of an order management system

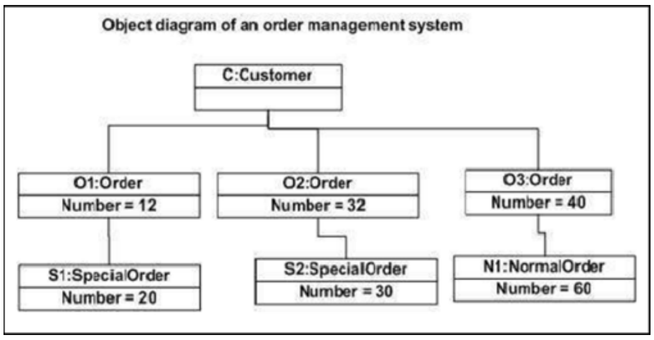


Fig -6: Object Diagram of Order Management System

The above diagram is an instance of the system which has the following objects:

- Customer
- Order
- SpecialOrder
- NormalOrder

The object C (Customer) is linked to three O (order) objects (O1,O2,O3). These O objects are linked with special and normal order objects (S1,S2 and N1 respectively).

The customer has 3 orders with numbers 12,32 and 40 for the particular instance considered. These numbers will differ from time instances. The same is true with respect to Special and Normal orders which have numbers 20,30 and 60. At a different instance of time, these numbers will also be different. The difference between class and object diagram is shown in Table-1 as explained in [8].

Table -1: Difference between Class Diagram and Object Diagram

Class diagram	Object diagram
Entire business process represented in one class diagram	Entire business process represented in number of object diagrams
Higher level diagram	Higher level diagram
State information implicitly present	State information explicitly present
Implicitly gives information for composite structure diagram	Explicitly gives information for composite structure diagram
All methods correspond to class participated	Intaglio subset of methods participated
Visibility: private, public, protected	Visibility: public, protected
Subclasses present once in class diagram	Same subclasses can present in many object diagram
Many boundary object classes may present	Scope to identify path between source boundary object class and destination boundary object class
No scope to identify usecase from class diagram	Scope to identify usecase diagram
No scope to identify usecase hierarchies identified in object diagram	Scope to identify usecase hierarchies identified in object diagram

3.3 Component Diagram

To model the physical features of a system, Component UML diagrams are used. Physical aspects include elements like libraries, executables, files, documents etc. which are placed in a node. A node is defined as a physical element that exists during runtime. Also, a component does not have its own features. These diagrams are used to to render executable systems and also showcase the relationships and organization among components of a system. Component diagrams are used to describe the components utilized to execute a functionality and not to describe the functionality as a whole. There are 3 elements in UML component diagrams: components, interfaces, dependencies. A basic component diagram is shown below in Figure-7 [9].

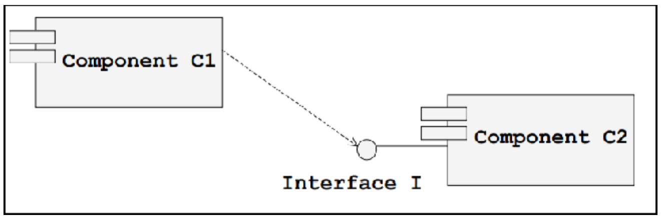


Fig -7: Basic Component Diagram

There are a few different interpretations of this diagram. Firstly, Classes in C1 utilise an instance of a class that implements I, and I is defined in C2.
Or Secondly, A class in C1 implements I, and I is defined in C2.
Or Lastly, Classes in C1 will use an instance of a class in C2 that implements I.
In addition to this, component diagrams include a Requires and Provides Port which is beyond the scope of this paper. Also, there are two ways to represent component diagrams which are described below in figure 8 [10].

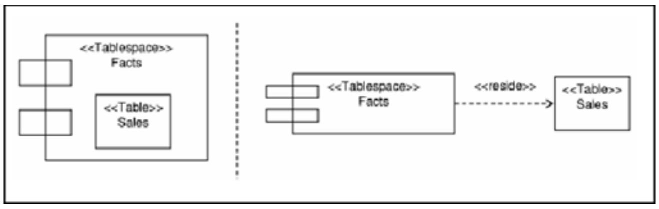


Fig -8: Different Component Representations in a Component Diagram

● LHS figure: the class (Sales) that resides on the component (Facts) is shown as nested inside the component (this indicates residence and not ownership)
● RHS figure: the class is connected to the component by a reside dependency

A sample of a Component Diagram of a Order Management System is shown below in figure 9. The files are the artifacts hence the figure shows the files in the application along with their relations. Artifacts can also include libraries, folders etc.

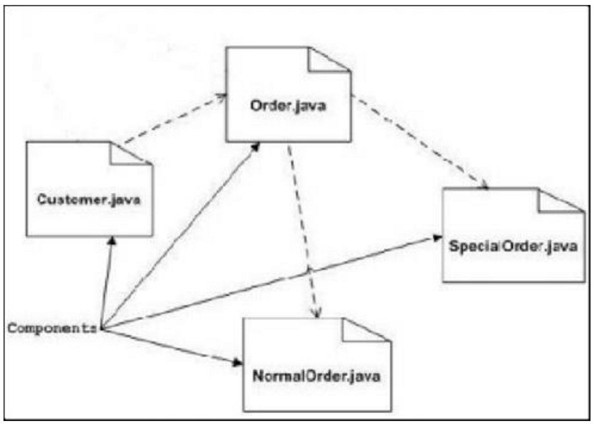


Fig -9: Component Diagram of an Order Management System

3.4 Package Diagram

Package diagram is a subtype Structural UML Diagram in which model elements are organized in various middle to large scale projects [11]. By grouping classes into packages complex Class diagrams can be simplified using Package diagrams. A package in a package diagram is basically a collection of different logically associated UML elements. These packages are represented as file folders which we can use in different UML diagrams.

Packages are rectangular with small tabs on the top. While labeling these packages, the class stereotype in a package should be considered too. The package name is either on the top or inside the rectangle. The label on the top is to fix the package's type but this isn't enough as developers are still not able to acquire enough information about the system or package. That's why there is a need to take some part of the program for displaying the content of a package. These packages are interdependent on each other. Here we decide that the class's stereotype and their distribution in the packages are similar. These classes have some design intent, which shows the main function as well as purpose of this package. And in this part, the description is displayed and represented in a table.

Package diagrams have the structure of nested packages. While using package names cannot be the same for a system. Packages can be represented as shown in Figure 10 and Figure 11.

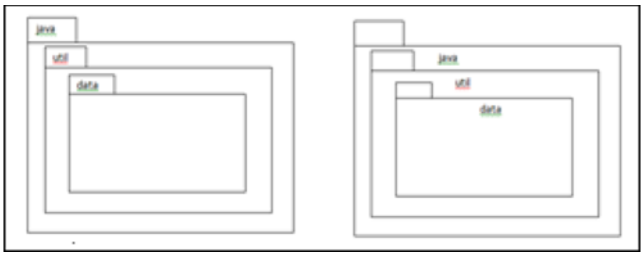


Fig -10: Structure of a Package

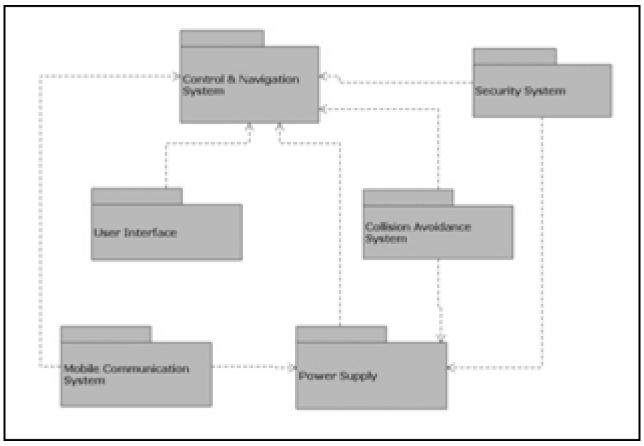

Control & Navigation System
Security System
User Interface
Collision Avoidance System
Mobile Communication System
Power Supply

Fig -11: Generalized form of a Package Diagram

Complex systems are usually structured using package diagrams. There two subtypes of dependencies involved in a package diagram that is <<import>> and <<access>>.

a. Import Command

By using this command one package can import the functionalities of another package. The imported element gets added to the namespace. Import is a public command. Figure 12 is a diagram of import command.

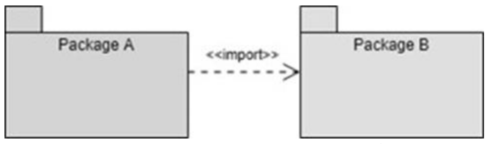


Fig -12: Import Command

b. Access Command

Access command is used when one package needs functionalities of the other package. Access is basically a private form of import. This is explained below in figure 13

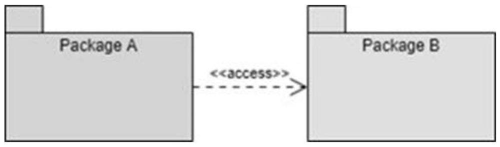


Fig -13: Access Command

3.5 Deployment Diagram

Deployment diagram is another type of structural UML diagram representing the hardware based upon which the software is going to run for performing some functionality. Deployment diagrams are made of different UML shapes. The 3D boxes (nodes) represent the basic hardware or software components. Lines represent the relationship between nodes and the small shapes in the boxes represent deployed artifacts.

Deployment diagrams have a wide range of applications. We can use it to check which hardware component has deployed which software component. Also, we can illustrate runtime processing for the hardware using deployment diagrams. We can also provide hardware system's topology view by using deployment diagrams. Artifact is an information generated by the software.

Deployment diagram shows the interaction between software and hardware to complete the functionality. Let us see an example of a deployment diagram of a web application as shown in Figure 14.

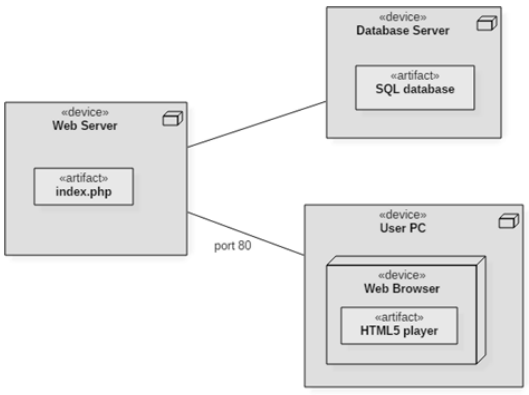


Fig -14: Deployment Diagram Example

Artifact is basically the information generated by the software. An artifact is a very important element in creating a deployment diagram. An artifact is shown in Figure 15.

«artifact»
configuration.php

Fig -15: Artifacts

Node is a resource or a medium on which an artifact is developed for execution. The size of a node in the deployment diagram varies as the size of project. The connection of two nodes represents the exchange of information between two nodes in any direction. A node is shown in Figure 16.

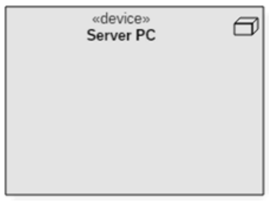


Fig -16: Node

4. RESULTS

The Results in this paper is an analysis of the survey taken by "Mr.Yashwant Waykar", included in his paper [12]. Different people working in software development projects took part in this survey where the participants rated the class diagram, component diagram and deployment diagram by selecting one out of four following options. Each option were assigned points as shown below. Choices given were:

1. Mandatory (most important) - 4 points
2. Important - 3 points
3. Less Important - 2 points
4. Optional - 1 point

Suppose if 10 participants selected option 1, i.e. Mandatory for class diagram then 10 times 4 is equal to 40 (10*4=40) points will be considered. Thus, diagram which receives the maximum points will be considered the most important diagram.
The results found are as follows:

4.1 Class Diagram

Table -2: Class Diagram Statistics from Survey

Options	No. Of Respond - ents	No. Of Respond- ents (%)	Points
Mandatory	18	69%	18 x 4 = 72
Important	4	15%	4 x 3 = 12
Less Imp	4	15%	4 x 2 = 8

Optional	0	0%	0

Total Points = 92

Thus, from the results, 69% people found Class diagram important and nobody found it least important.

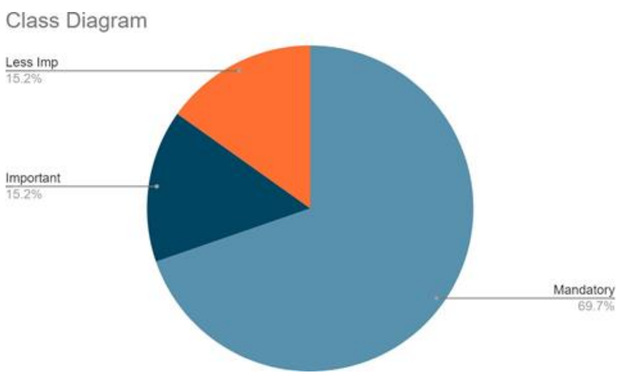


Chart -1: Importance of Class Diagram

4.2 Component Diagram

Table -3: Component Diagram Statistics from Survey

Options	**No. Of Respond - ents**	**No. Of Respond-ents (%)**	**Points**
Mandatory	6	23%	6 x 4 = 24
Important	10	38%	10 x 3 = 30
Less Imp	8	31%	8 x 2 = 16
Optional	2	8%	2 x 1 = 2

Total Points = 72

Thus, from the results, 23% people found Component diagram important and 8% people found it least important.

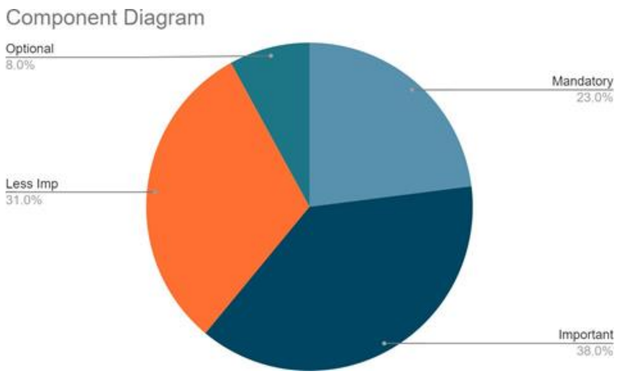


Chart -2: Importance of Component Diagram

4.3 Deployment Diagram

Table -4: Deployment Diagram Statistics from Survey

Options	**No. Of Respond - ents**	**No. Of Respond-ents (%)**	**Points**
Mandatory	8	31%	8 x 4 = 32
Important	6	23%	6 x 3 = 18
Less Imp	8	31%	8 x 2 = 16
Optional	4	15%	4 x 1 = 4

Total Points = 70

Thus, from the results, 31% people found Deployment diagram important and 15% people found it least important.

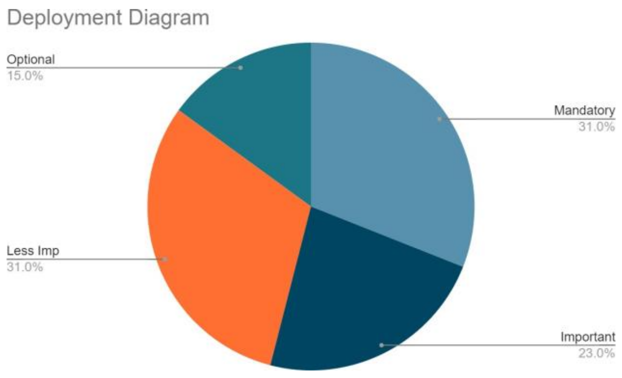


Chart -3: Importance of Deployment Diagram

Thus, from the above tables and charts it is inferred that Class diagram is the most used Diagram having 92 points.

5. CONCLUSIONS

In this paper we have introduced five of the most used and important Structural UML diagrams. From the results presented above, Class diagram as mentioned earlier is a static diagram and it is used to model the unchanging view of a system and is the most used or the most important Structural UML diagram. It is also considered as the foundation for Component and Deployment diagrams. Modeling the deployment aspects of software applications can reduce the complexities of software development, thus deployment diagrams are used. Component diagrams explain the components necessary to execute software functionalities. As explained earlier, an object diagram shows an instance of the system at a particular moment in time, thus it is solid in nature. Package diagrams are used to group classes into packages and also to detangle complex class diagrams. Throughout the software development process, the UML model developing is an essential element and demands that the diagrams are correct and consistent to each other, to ensure a smooth software development process.

REFERENCES

[1] L.P. Fanjoy, A.L.MacNeill, and L.A. Best: "The Use of Diagrams in Science" in Conference: Proceedings of the 7th international conference on Diagrammatic Representation and Inference, July 2012.

[2] Mohammad N, Alanzi: "Basic Rules to Build Correct UML Diagrams", in International Conference on New Trends in Information and Service Science, 2009

[3] Er.D.Singh and Dr.J.S.Sidhu: "A SCRUTINY STUDY OF VARIOUS UNIFIED MODELING LANGUAGE (UML) DIAGRAMS, SOFTWARE METRICS TOOL AND PROGRAM SLICING TECHNIQUE", in JETIR June 2018, Volume 5, Issue 6

[4] D. Bell, IT Architect, IBM Corporation: "UML basics: The class diagram", 15 September 2004.

[5] Sadia Sadaf, Ali Athar and Farooque Azam "Evaluation of FED-CASE - A Tool to Convert Class Diagram into Structural Coding" In Islamabad Pakistan.

[6] Oskana Nikiforova and Janice Sejans "Role of UML Class Diagram in Object-Oriented Software Development", January 2011.

[7] Salma, Medium, "UML Class Diagrams Tutorial, Step by Step", September 2017. [Accessed 15th July,2021].

[8] S.M. Handigund, S. Sajjanar; Arunakumari B. N, "Resuscitation of syllogism within unified modeling language levels through the renovation of object diagram",In Proc. International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2015.

[9] D.S.Rosenblum and C.Ler, "UML Component Diagrams and Software Architecture - Experiences from the Wren Project",Workshop at the 23rd International Conference on Software Engineering, Toronto, Canada, 2001

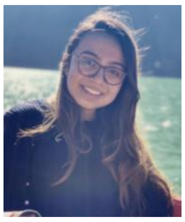
[10] S.Lujan-Mora and J.Tujillo, "Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams: Design and Implementation Issues", In Proc.IEEE conference, 2005.

[11] Li Jiang, Xiaobing Sun, Yun L and, Xiangyue Liu "Automatic Generation of Package Diagram to Understand Java Packages" June4-6 2014.

[12] Mr.Yashwant Waykar, "A Study of Importance of UML diagrams: With Special Reference to Very Large-sized Projects", in International Conference on Reinventing Thinking beyond boundaries to ExcelAt: FARIDABAD, INDIA, March 2013.

BIOGRAPHIES

Bhavik Bhatt


Muskaan Nandu