Shadow Project

LTTS
GLOBAL
ENGINEERING
ACADEMY

L&T Technology Services

Details

SHADOW Project report

Name of Candidates	**Mohammed Sadiq Afreed Meeran**	**Kavya K**	**Akash Shetty**	**Ankita D Joil**
PS Numbers	**99002449**	**99002567**	**99002641**	**99002660**
Name of Mentor – Delivery	**Hrushi AC**		**PS No: 996171**	
Name of Mentor- GEA	**Rajesh Sola**		**PS No: 40010751**	
Project Period	**From: November 2020**		**To: December 2020**	
Date of Completion				

Contents

SHADOW Project report **2**
List of Figures **4**
List of Tables **5**
Product Title ***6***
Project Details ***6***
Requirements ***6***
Methodologies ***7***
Design ***10***
Git Link ***13***

List of Figures

Figure 1: MVC Architecture 7
Figure 2: MVVM Architecture 8
Figure 3: Use Case Diagram 11
Figure 4: Activity Diagram 12

List of Tables

Table 1: High Level Requirements 6
Table 2: Low Level Requirements 7

Product Title

Design and Develop a Qt/QML application for multi waveform drawing

Limited features Waveform drawing with given input data file, Waveform scaling, Waveform speed, Waveform color, Waveform thickness.

Project Details

- **AIM**:
  - Goal of this application development is to get an understanding of C++ features and Qt/QML features.

- **PROBLEM STATEMENT:**
  - Develop requirement
  - Come up with design (Class diagram and Flow diagrams)
  - Develop the code to have features like waveform drawing using C++, Qt/QML.

Requirements

High Level Requirements

HLR	High Level Requirement Details
HLR_01	User shall be able to draw single/multiple waveforms by providing input data file.
HLR_02	User shall be able to control the waveform sweep speed.
HLR_03	User shall be able to change the scale of the waveform.
HLR_04	User shall be able to change the color of the waveform.
HLR_05	User shall be able to change the thickness of the waveform.
HLR_06	User shall be able to apply localization for the textual contents.

Table 1: High Level Requirements

Low Level Requirements

LLR	Low Level Requirement Details
LLR_01	The application shall be able to portray single/multiple waveforms, provided with input data file.
LLR_02	The application shall portray waveforms with the different sweep speed.
LLR_03	The application shall be able to zoom-in/zoom-out waveform.
LLR_04	The application shall support localization of textual content in different languages.

LLR_05	The application shall be able to generate log file for unintended exceptions.
LLR_06	The application shall include singleton standard design pattern.

Table 2: Low Level Requirements

Methodologies

- **MVC (Model View Controller)**

Model **V**iew **C**ontroller as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts −

- **Model** − The lowest level of the pattern which is responsible for maintaining data.
- **View** − This is responsible for displaying all or a portion of the data to the user.
- **Controller** − Software Code that controls the interactions between the Model and View.

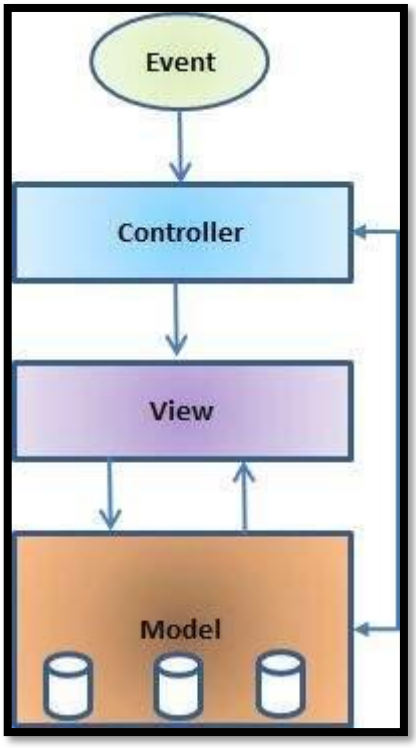

Figure 1: MVC Architecture

The Model:

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The View:

It means presentation of data in a particular format, triggered by a controller's decision to present the data. They are script-based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The Controller:

The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

- **MVVM (Model View View Model)**

MVVM architecture offers two-way data binding between view and view-model. It also helps you to automate the propagation of modifications inside View-Model to the view. The view-model makes use of observer pattern to make changes in the view-model.

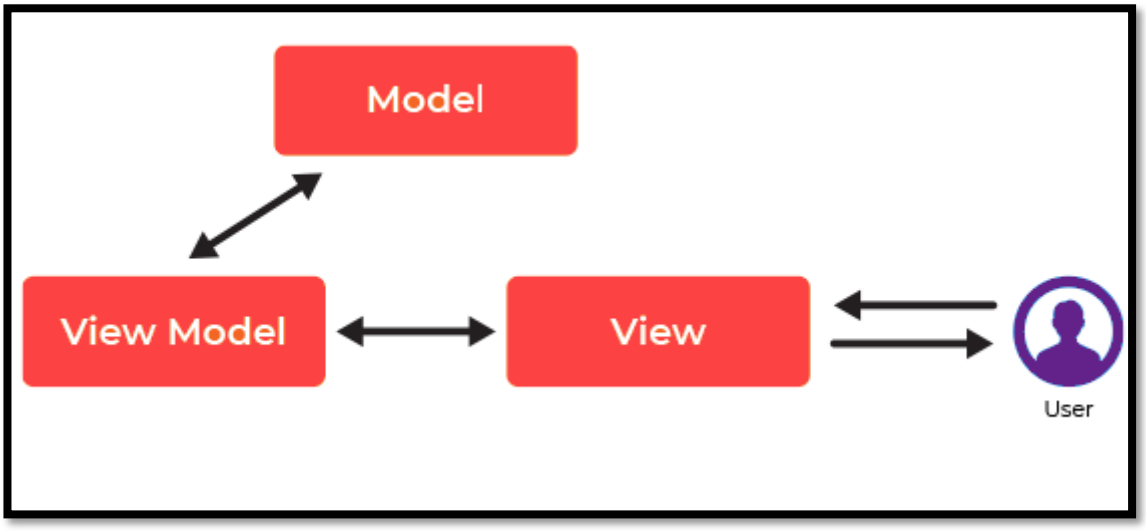

Figure 2: MVVM Architecture

Model

The model stores data and related logic. It represents data that is being transferred between controller components or any other related business logic.For example, a Controller object will retrieve the student info from the school database. It manipulates data and sends it back to the database or use it to render the same data.

View

The View stands for UI components like HTML, CSS, jQuery, etc. In MVVC pattern view is held responsible for displaying the data which is received from the Controller as an outcome. This View is also transformed Model (s) into the User Interface (UI).

View Model

The view model is responsible for presenting functions, commands, methods, to support the state of the View. It is also accountable to operate the model and activate the events in the View.

- **SOLID Principles**

SOLID is a **mnemonic device for 5 design principles** of object-oriented programs (OOP) that result in readable, adaptable, and scalable code. SOLID can be applied to any OOP program.

The 5 principles of SOLID are:

- **S**ingle-responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

- **S**ingle-responsibility principle:
    - The **single-responsibility principle** (SRP) states that each class, module, or function in your program should only do one job. In other words, each should have full responsibility for a single functionality of the program. The class should contain only variables and methods relevant to its functionality.
    - Classes can work together to complete larger complex tasks, but each class must complete a function from start to finish before it passes the output to another class.
- **O**pen-closed principle:
    - The **open-closed principle** (OCP) calls for entities that can be widely adapted but also remain unchanged. This leads us to create duplicate entities with specialized behavior through polymorphism.
    - Through polymorphism, we can extend our parent entity to suit the needs of the child entity while leaving the parent intact.
    - Our parent entity will serve as an abstract base class that can be reused with added specializations through inheritance. However, the original entity is locked to allow the program to be both open and closed.
- **L**iskov substitution principle
    - The **Liskov substitution principle** (LSP) is a specific definition of a subtyping relation created by Barbara Liskov and Jeannette Wing. The principle says that any class must be directly replaceable by any of its subclasses without error.
    - In other words, each subclass must maintain all behavior from the base class along with any new behaviors unique to the subclass. The child class must be able to process all the same requests and complete all the same tasks as its parent class.
- **I**nterface segregation principle
    - The **interface segregation principle** (ISP) requires that classes only be able to perform behaviors that are useful to achieve its end functionality. In other words, classes do not include behaviors they do not use.
    - This relates to our first SOLID principle in that together these two principles strip a class of all variables, methods, or behaviors that do not directly contribute to their role. Methods must contribute to the end goal in their entirety.

- **D**ependency inversion principle

  The **dependency inversion principle** (DIP) has two parts:

  1. High-level modules should not depend on low-level modules. Instead, both should depend on abstractions (interfaces)
  2. Abstractions should not depend on details. Details (like concrete implementations) should depend on abstractions.

Design

Structural Diagrams

UML Structural diagrams depict the elements of a system that are independent of time and that convey the concepts of a system and how they relate to each other. The elements in these diagrams resemble the nouns in a natural language, and the relationships that connect them are structural or semantic relationships.

These static parts are represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are −

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Behavioral Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as complete when both the aspects are fully covered.

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

UML has the following five types of behavioral diagrams −

- Use case diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram

Use Case Diagram

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system. A use case represents a functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.

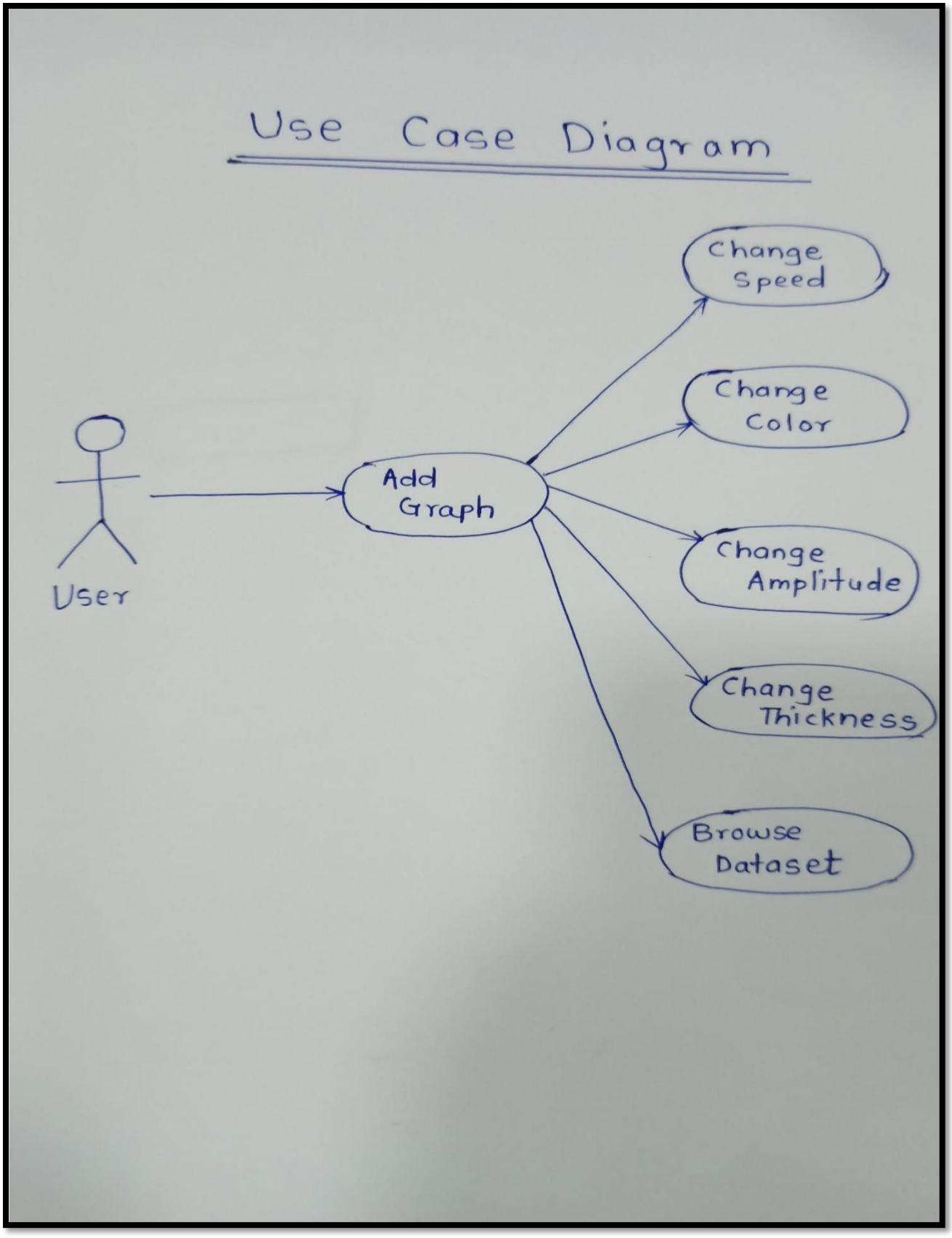

Figure 3: Use Case Diagram

Activity Diagram

Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched. Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

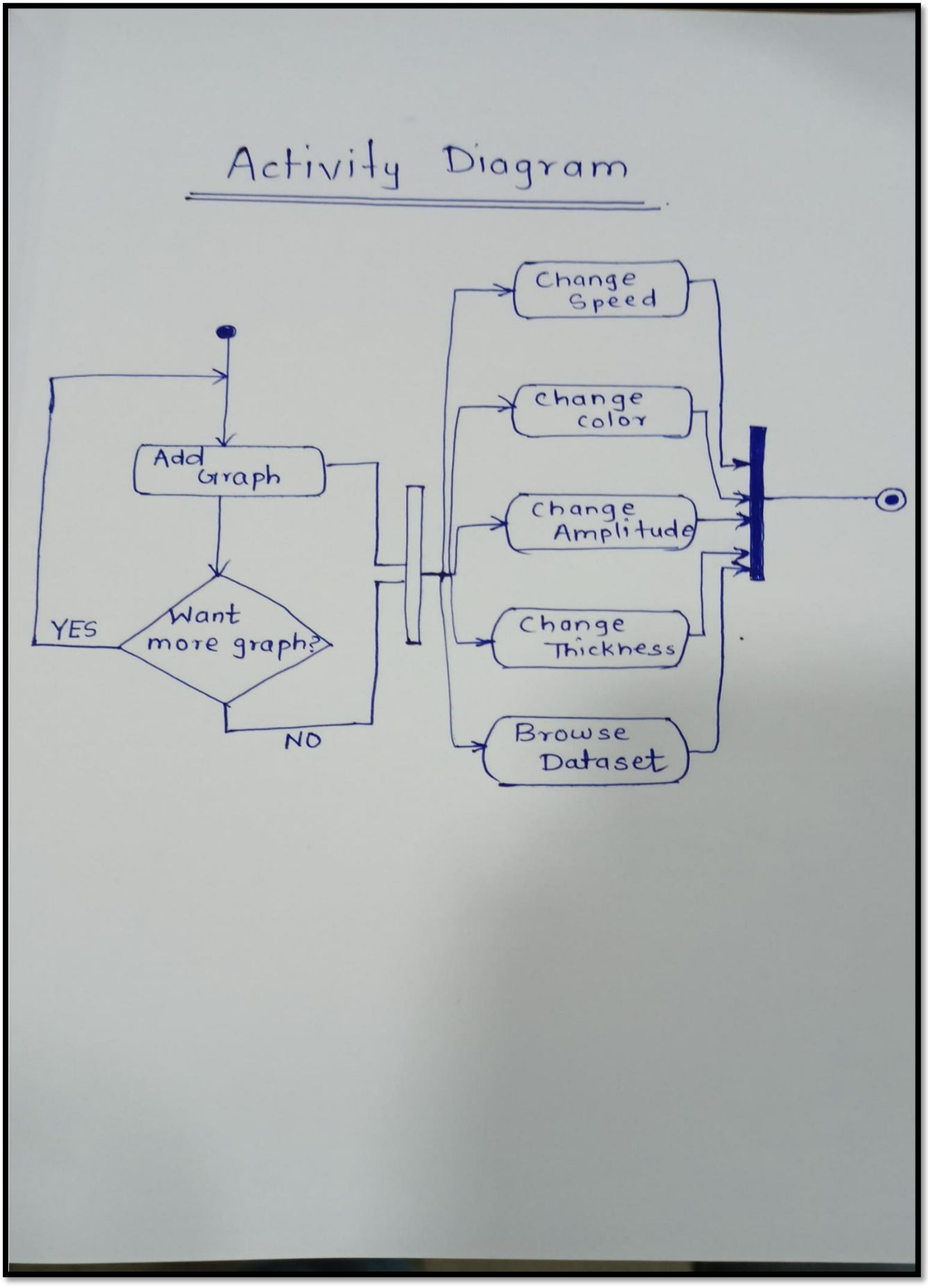

Figure 4: Activity Diagram

Git Link

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers.

Git Link- https://github.com/99002449/Shadow_Project