PROJECT REPORT

MNNIT Allahabad, Prayagraj
B. Tech 6th Semester, Computer Science and Engineering
CS-1606 Software Engineering

Chart Generator Tool

By

Group - 6

Rahul Kumar, 20164164
Rahul Singh Adhikari, 20164121
Rajat Dipta Biswas, 20164114
Rakesh Singh, 20164111
Raman Sehgal, 20164130

Project coordinator:
Dr. Anoj Kumar
Associate Professor
anojk@mnnit.ac.in

April 2019

1 INTRODUCTION

Charts are an excellent way to condense large amounts of information into easy-to-understand formats that clearly highlight the points you'd like to make. Humans are visual creatures and it is much easier for humans to infer meaning from pictures than from text. Texts in a chart are generally used only to annotate the data.

Charts are often used to ease understanding of large quantities of data and the relationships between parts of the data. Charts can usually be read more quickly than the raw data.

This is why in project management visual aids like charts are invaluable to the planning of a project. Many different forms of visualisation and charts are used during project management which help in letting members know what to do or when the subtask should be handled.

2 PROBLEM STATEMENT

Any group or organisation making and maintaining a project seeks to complete the task at hand in the shortest possible time and as efficiently as possible. For this, it is necessary for team members to know when and how they should accomplish their task.

However, there are very few softwares that will help generate all the necessary charts that are necessary for project management. Most of them require making spreadsheets and don't have proper and simple input options for generating the charts.

In response to this problem, our team has decided to create a tool that will generate different charts for project management. This will help teams create visualisations quickly and without much hassle. Some of the charts to be implemented in the project include:

1. **Gantt Chart** - A type of bar chart that demonstrates the project schedule. It is best used in tracking progress. Moreover, project managers can plan, work out the practical aspects and potential problems, segregate the work to team members, and minimize the delivery time using this graphic.
2. **PERT Chart** - Program Evaluation Review Technique (PERT) Chart is known as the network diagram. PERT charts are used after a project has been planned and broken into smaller pieces, called tasks. They show how much time is needed to complete each task and they help project managers visualize

which tasks must be completed before others can begin. PERT charts also illustrate what tasks are critical (critical path) and which ones are less important. They portray more complex project tasks and analyze the tasks encompassed in the completion of the project. PERT charts also analyze the time needed to accomplish the task and recognize the minimum time required. They generally showcase he parallel activities.

3. **Work Breakdown Structure** - Work Breakdown Structure (WBS) is a project management diagram that illustrates an entire project broken down into components and then into smaller sub components. They are used to arrange a project team's work into manageable pieces and for estimating costs and time associated with each piece of work. This provides you the hierarchy of tasks that is part of the project. The manager can easily envision the components and their relation to execute the best planning.

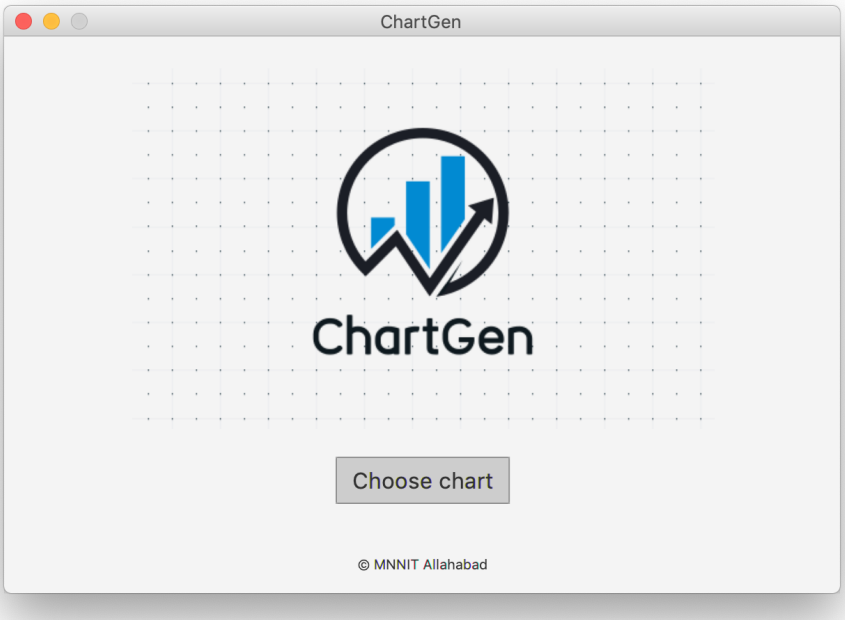
3 MODULES

Project Structure

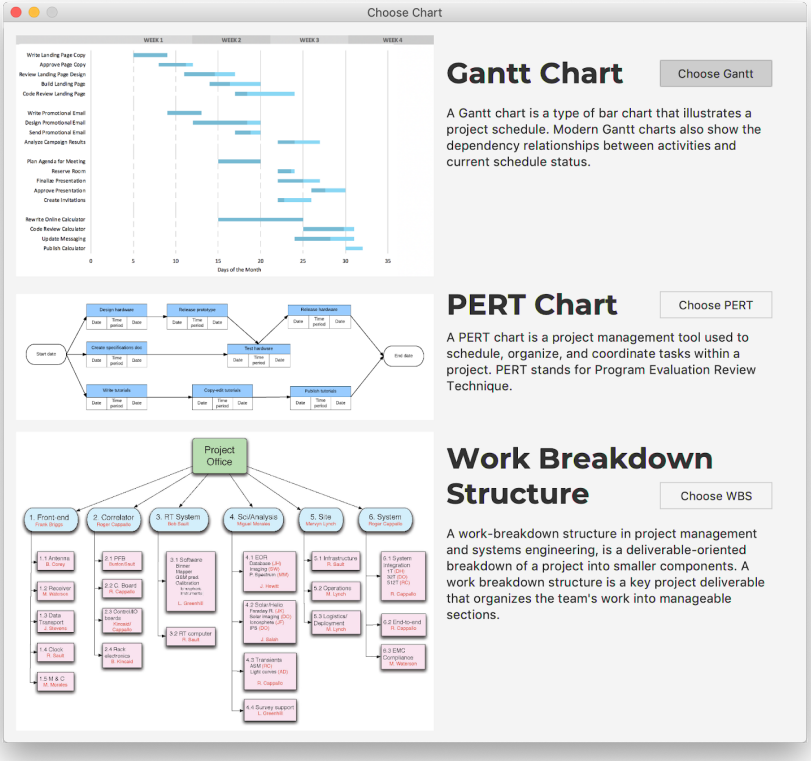
```
ChartGen
├── ChartGen.iml
├── histogram.png
├── out
│   └── production
│       └── ChartGen
│           ├── ChartGenerator
│           │   ├── ChooseChart.fxml
│           │   ├── ChooseChartController.class
│           │   ├── GanttChart.class
│           │   ├── GanttChart.fxml
│           │   ├── GanttChartController.class
│           │   ├── Main.class
│           │   ├── MainScreen.fxml
│           │   ├── MainScreenController.class
│           │   ├── PERTChart.fxml
│           │   ├── PERTChartController.class
│           │   ├── WorkBreakdown.fxml
│           │   └── WorkBreakdownController.class
│           ├── META-INF
│           │   └── ChartGen.kotlin_module
│           ├── PertChart
│           │   ├── Activity.class
│           │   ├── Calculations.class
│           │   ├── Histogram.class
│           │   ├── PertChart.class
│           │   ├── PertChartGUI$1.class
```

```
│           │   ├── PertChartGUI.class
│           │   ├── Project.class
│           │   └── Results.class
│           ├── images
│           │   ├── gantt.png
│           │   ├── logo.png
│           │   ├── pert.png
│           │   └── wbs.jpg
│           ├── jcommon-1.0.8.jar
│           ├── jfoenix-8.0.8.jar
│           ├── jfoenix-9.0.8.jar
│           ├── jfreechart-1.0.13.jar
│           └── jfreechart-1.5.0.jar
└── src
    ├── ChartGenerator
    │   ├── ChooseChart.fxml
    │   ├── ChooseChartController.java
    │   ├── GanttChart.fxml
    │   ├── GanttChart.java
    │   ├── GanttChartController.java
    │   ├── Main.java
    │   ├── MainScreen.fxml
    │   ├── MainScreenController.java
    │   ├── PERTChart.fxml
    │   ├── PERTChartController.java
    │   ├── WorkBreakdown.fxml
    │   └── WorkBreakdownController.java
    ├── PertChart
    │   ├── Activity.java
    │   ├── Calculations.java
    │   ├── Histogram.java
    │   ├── PertChart.java
    │   ├── PertChartGUI.java
    │   ├── Project.java
    │   └── Results.java
    ├── images
    │   ├── gantt.png
    │   ├── logo.png
    │   ├── pert.png
    │   └── wbs.jpg
    ├── jcommon-1.0.8.jar
    ├── jfoenix-8.0.8.jar
    ├── jfoenix-9.0.8.jar
    ├── jfreechart-1.0.13.jar
    └── jfreechart-1.5.0.jar

11 directories, 60 files
```

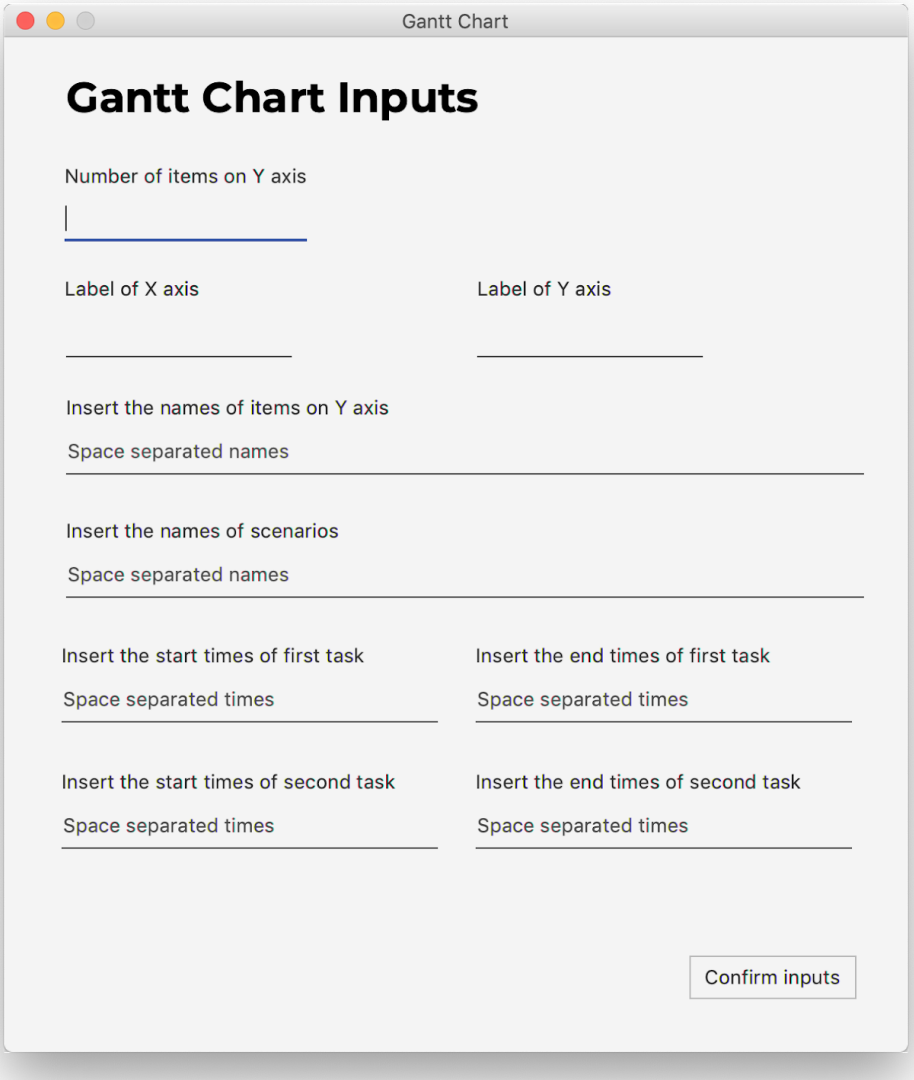
1. **Initial startup user interface** - This module will have the GUI for the start up page. The initial starting page provides the first impression of the software.



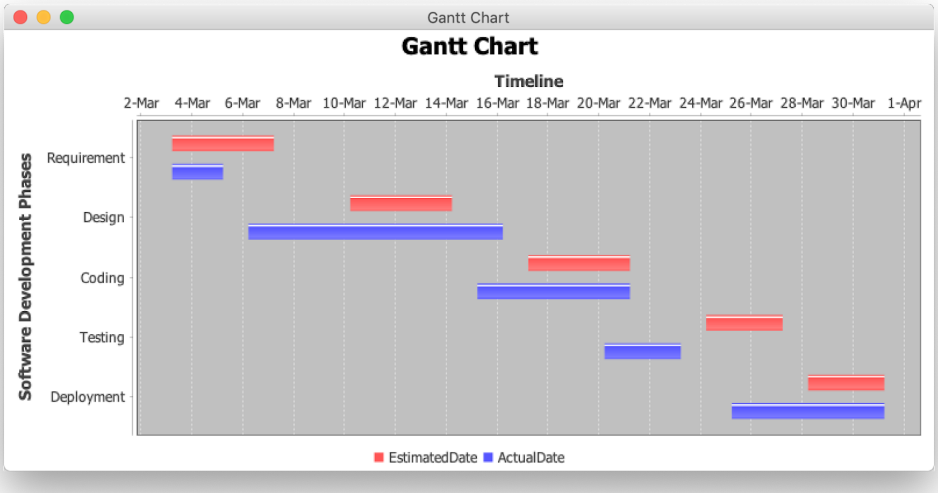
2. **Chart choosing** - The user is given the option of which chart to implement. The details of each of the charts is presented in brief.



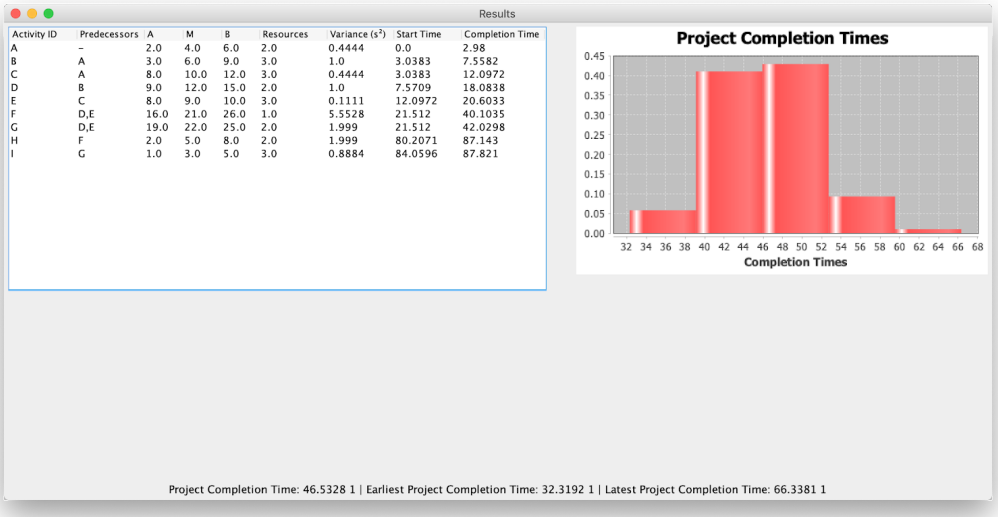
3. **Data entry module** - Once the chart is chosen, the user is given an user interface to input the data for each of the charts. Each chart will need to include a different form of data entry because not all charts will require the same inputs.



4. **Chart generating modules** - Each chart option will have a separate module to process the input data and generate the chart.



5. **Visualisation** - The output obtained will then be visualised. Each charting module visualises the output in a different way.



5 SPECIFICATIONS

5.1 Software Requirements

For best performance, the following software requirements must be met:

1. *Operating System:*

   1.1 Windows

   - Microsoft Windows 7, 64-bit
   - Microsoft Windows 8.1, 64-bit
   - Microsoft Windows 10, 64-bit (8u51 or later)

   1.2 Mac OS

   - macOS 10.13 High Sierra (and above)

   1.3 Linux

   - Oracle Linux 6.x (32-bit), 6.x (64-bit)
   - Oracle Linux 7.x (64-bit) (8u20 and above)
   - Red Hat Enterprise Linux 5.5+, 6.x (32-bit), 6.x (64-bit)
   - Red Hat Enterprise Linux 7.x (64-bit) (8u20 and above)
   - Suse Linux Enterprise Server 10 SP2+, 11.x
   - Suse Linux Enterprise Server 12.x (64-bit) (8u31 and above)
   - Ubuntu Linux 18.04 LTS Bionic Beaver (and above)

   NOTE: Must have Java (Version 8 Update 201) or later installed

5.2 Hardware Requirements

To maximize performance, the following hardware requirements must be met:

1. *Processor:*
Intel® Pentium IV (or an equivalent AMD processor - AMD Athlon XP) or better

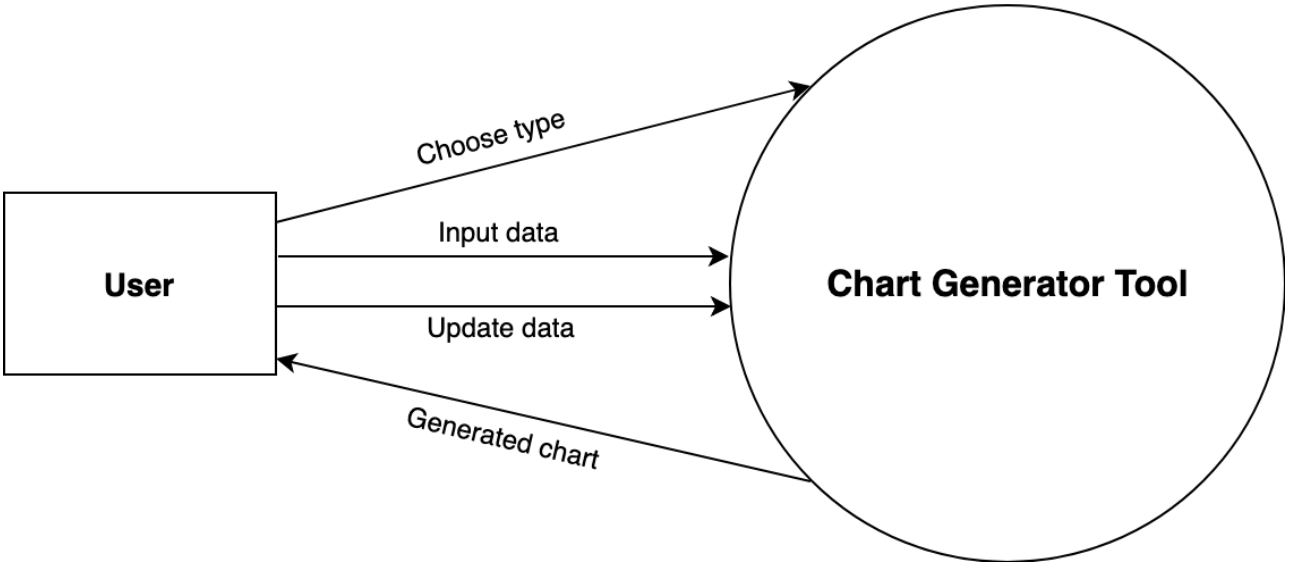
2. *RAM:*
   256 MB of RAM is sufficient for proper functionality of the software. It would be beneficial for the user to have RAM greater than that specified for better overall experience and enhanced multitasking.

3. *Graphics Card:*
   64 MB video card with DirectX 9.0 compatible drivers ("GeForce 3"/"Radeon 8500" or better with DirectX Texture Compression support)

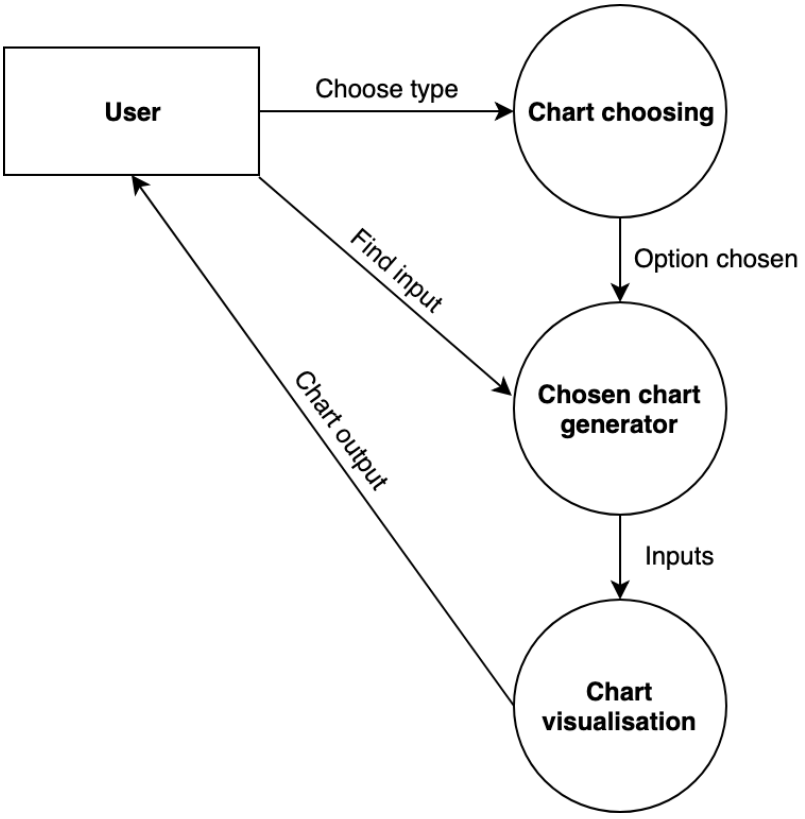
4. *Storage:*
   Hard Disk (5400 RPM or better)
   The storage space of the computer should also be sufficient enough for proper functionality of the application along with the other already installed softwares.

6 DATA FLOW DIAGRAM

Context Flow Diagram



Level-1 Data Flow Diagram



Level-2 Data Flow Diagram

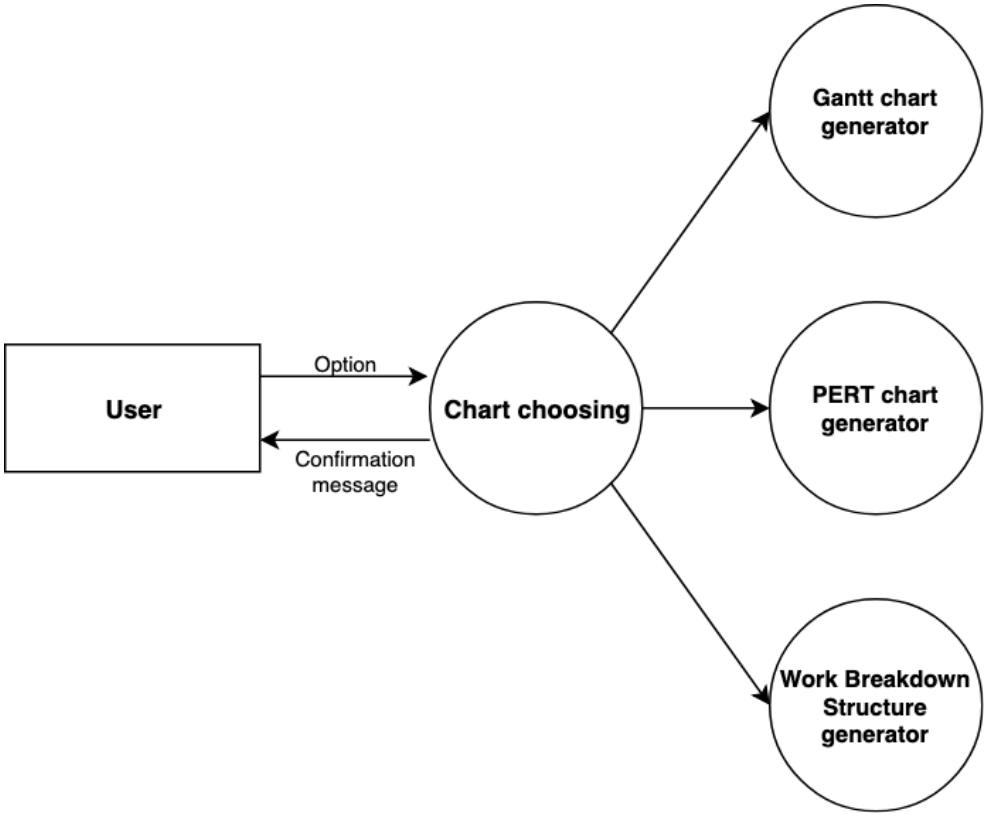


Chart Choosing Process

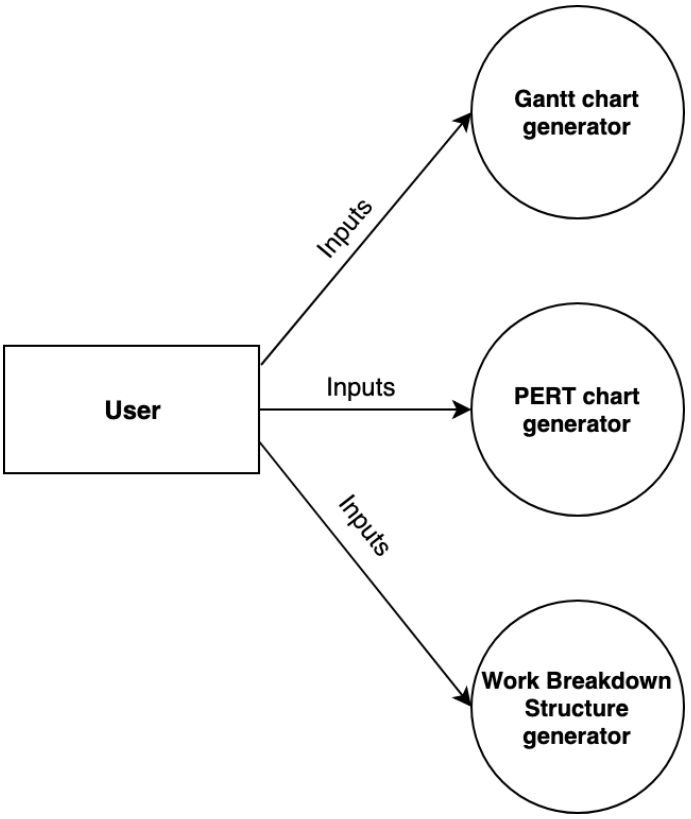


Chart Generator Process

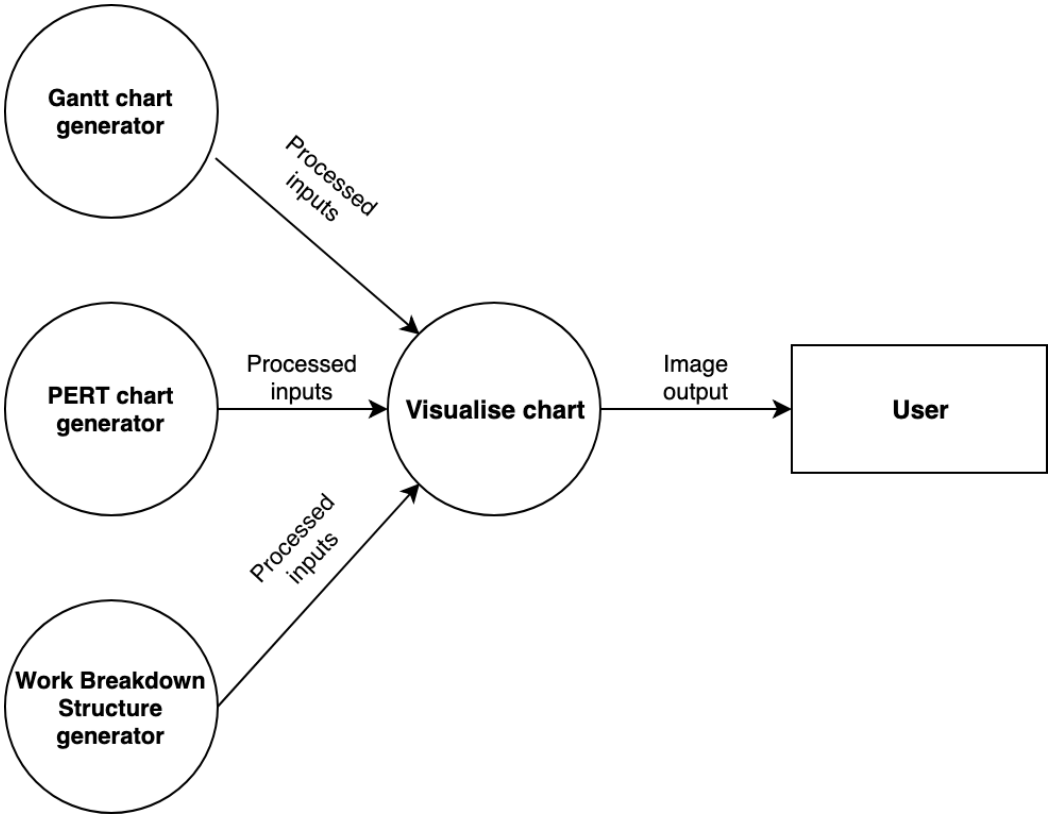


Chart Visualisation Process

7 CODING

Main.java

```
package ChartGenerator;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

   public static Stage MainStage, ChooseChartStage, GanttChartStage,
PertChartStage, WorkBreakdownStage;

   @Override
   public void start(Stage primaryStage) throws Exception
   {
       Parent root = FXMLLoader.load(getClass().getResource("MainScreen.fxml"));

       MainStage = primaryStage;
       primaryStage.setTitle("ChartGen");
       primaryStage.setResizable(false);
       primaryStage.setScene(new Scene(root));
       primaryStage.show();
   }

   public void showGanttChart() {

       try {
           Parent root =
FXMLLoader.load(getClass().getResource("GanttChart.fxml"));

           GanttChartStage = new Stage();
           GanttChartStage.setTitle("Gantt Chart");
           GanttChartStage.setResizable(false);
           GanttChartStage.setScene(new Scene(root));
           GanttChartStage.show();
       }
       catch (Exception e) {}
   }

   public void showPERTChart() {
```

```
        try {
            Parent root =
FXMLLoader.load(getClass().getResource("PERTChart.fxml"));

            PertChartStage = new Stage();
            PertChartStage.setTitle("PERT Chart");
            PertChartStage.setResizable(false);
            PertChartStage.setScene(new Scene(root));
            PertChartStage.show();
        }
        catch (Exception e) {}
    }

    public void showWBS() {

        try {
            Parent root =
FXMLLoader.load(getClass().getResource("WorkBreakdown.fxml"));

            WorkBreakdownStage = new Stage();
            WorkBreakdownStage.setTitle("Work Breakdown Structure");
            WorkBreakdownStage.setResizable(false);
            WorkBreakdownStage.setScene(new Scene(root));
            WorkBreakdownStage.show();
        }
        catch (Exception e) {}
    }

    public void showChooseChart() {

        try {
            Parent root =
FXMLLoader.load(getClass().getResource("ChooseChart.fxml"));

            ChooseChartStage = new Stage();
            ChooseChartStage.setTitle("Choose Chart");
            ChooseChartStage.setResizable(false);
            ChooseChartStage.setScene(new Scene(root));
            ChooseChartStage.show();
        }

        catch (Exception e) {}
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

MainScreen.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8.0.172-ea" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ChartGenerator.MainScreenController">
  <children>
     <ImageView fitHeight="259.0" fitWidth="418.0" layoutX="92.0" layoutY="23.0">
        <image>
           <Image url="@../images/logo.png" />
        </image>
     </ImageView>
     <JFXButton layoutX="238.0" layoutY="302.0" onAction="#chooseChart"
style="-fx-border-color: #aaa;" text="Choose chart">
        <font>
           <Font size="16.0" />
        </font>
        <opaqueInsets>
           <Insets />
        </opaqueInsets>
     </JFXButton>
     <Label layoutX="254.0" layoutY="373.0" text="(c) MNNIT Allahabad">
        <font>
           <Font size="10.0" />
        </font>
     </Label>
  </children>
</Pane>
```

MainScreenController.java

```
package ChartGenerator;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
```

```
import javafx.fxml.Initializable;

import java.net.URL;
import java.util.ResourceBundle;

public class MainScreenController implements Initializable {

    @Override
    public void initialize(URL location, ResourceBundle resources) {

    }

    @FXML
    void chooseChart(ActionEvent event) {
        new Main().showChooseChart();
        Main.MainStage.close();
    }
}
```

ChooseChart.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="800.0" prefWidth="825.0"
xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ChartGenerator.ChooseChartController">
   <children>
      <ImageView fitHeight="269.0" fitWidth="464.0" layoutX="14.0"
layoutY="14.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@../images/gantt.png" />
         </image>
      </ImageView>
      <ImageView fitHeight="144.0" fitWidth="464.0" layoutX="14.0"
layoutY="302.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@../images/pert.png" />
         </image>
      </ImageView>
```

```
      <ImageView fitHeight="334.0" fitWidth="464.0" layoutX="14.0"
layoutY="455.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@../images/wbs.jpg" />
         </image>
      </ImageView>
      <Label layoutX="492.0" layoutY="37.0" text="Gantt Chart">
         <font>
            <Font name="Montserrat Bold" size="32.0" />
         </font>
      </Label>
      <Label layoutX="492.0" layoutY="294.0" text="PERT Chart">
         <font>
            <Font name="Montserrat Bold" size="32.0" />
         </font>
      </Label>
      <Label layoutX="492.0" layoutY="454.0" prefHeight="99.0"
prefWidth="400.0" text="Work Breakdown Structure" wrapText="true">
         <font>
            <Font name="Montserrat Bold" size="32.0" />
         </font>
      </Label>
      <Label layoutX="492.0" layoutY="83.0" minHeight="48.0" minWidth="291.0"
prefHeight="90.0" prefWidth="363.0" text="A Gantt chart is a type of bar
chart that illustrates a project schedule. Modern Gantt charts also show the
dependency relationships between activities and current schedule status."
wrapText="true">
         <font>
            <Font size="14.0" />
         </font>
      </Label>
      <Label layoutX="492.0" layoutY="332.0" minHeight="38.0"
minWidth="291.0" prefHeight="90.0" prefWidth="363.0" text="A PERT chart is a
project management tool used to schedule, organize, and coordinate tasks
within a project. PERT stands for Program Evaluation Review Technique."
wrapText="true">
         <font>
            <Font size="14.0" />
         </font>
      </Label>
      <Label layoutX="492.0" layoutY="553.0" minHeight="38.0"
minWidth="291.0" prefHeight="119.0" prefWidth="363.0" text="A work-breakdown
structure in project management and systems engineering, is a
deliverable-oriented breakdown of a project into smaller components. A work
breakdown structure is a key project deliverable that organizes the team's
work into manageable sections." wrapText="true">
         <font>
```

```
            <Font size="14.0" />
         </font>
      </Label>
      <JFXButton layoutX="729.0" layoutY="42.0" onAction="#chooseGantt"
prefHeight="30.0" prefWidth="125.0" style="-fx-border-color: #ccc;"
text="Choose Gantt" />
      <JFXButton layoutX="729.0" layoutY="299.0" onAction="#choosePERT"
prefHeight="30.0" prefWidth="125.0" style="-fx-border-color: #ccc;"
text="Choose PERT" />
      <JFXButton layoutX="729.0" layoutY="511.0" onAction="#chooseWBS"
prefHeight="30.0" prefWidth="125.0" style="-fx-border-color: #ccc;"
text="Choose WBS" />
   </children>
</AnchorPane>
```

ChooseChartController.java

```
package ChartGenerator;

import PertChart.PertChartGUI;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;

public class ChooseChartController {

    @FXML
    void chooseGantt(ActionEvent event) {
        new Main().showGanttChart();

        Main.ChooseChartStage.close();
    }

    @FXML
    void choosePERT(ActionEvent event) {
        new PertChartGUI();
        Main.ChooseChartStage.close();
    }

    @FXML
    void chooseWBS(ActionEvent event) {
        new Main().showWBS();
        Main.ChooseChartStage.close();
    }

}
```

GanttChart.java

```
package ChartGenerator;

import java.time.LocalDate;
import java.time.ZoneOffset;
import java.util.Date;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.category.IntervalCategoryDataset;
import org.jfree.data.gantt.Task;
import org.jfree.data.gantt.TaskSeries;
import org.jfree.data.gantt.TaskSeriesCollection;

public class GanttChart extends JFrame {

    public GanttChart(String title, int n, String xLabel, String yLabel,
String[] yAxisItems, String[] scenarios, int[] start1, int[] end1, int[]
start2, int[] end2) {
        super(title);

        IntervalCategoryDataset dataset = getCategoryDataset(n, yAxisItems,
scenarios, start1, end1, start2, end2);

        // Create chart
        JFreeChart chart = ChartFactory.createGanttChart(
                "Gantt Chart", // Chart title
                yLabel,
                xLabel,
                dataset);

        ChartPanel panel = new ChartPanel(chart);
        setContentPane(panel);
    }

    private IntervalCategoryDataset getCategoryDataset(int n, String[]
yAxisItems, String[] scenarios, int[] start1, int[] end1, int[] start2, int[]
end2) {

        TaskSeries series1 = new TaskSeries(scenarios[0]);

        for(int i = 0; i<n; i++) {
```

```
            series1.add(new Task(yAxisItems[i],

Date.from(LocalDate.of(2019,3,start1[i]).atStartOfDay().toInstant(ZoneOffset.
UTC)),
                    Date.from(LocalDate.of(2019,
3,end1[i]).atStartOfDay().toInstant(ZoneOffset.UTC))
            ));
        }

        TaskSeries series2 = new TaskSeries(scenarios[1]);

        for(int i = 0; i<n; i++) {
            series2.add(new Task(yAxisItems[i],

Date.from(LocalDate.of(2019,3,start2[i]).atStartOfDay().toInstant(ZoneOffset.
UTC)),
                    Date.from(LocalDate.of(2019,
3,end2[i]).atStartOfDay().toInstant(ZoneOffset.UTC))
            ));
        }

        TaskSeriesCollection dataset = new TaskSeriesCollection();
        dataset.add(series1); dataset.add(series2);

        return dataset;
    }

}
```

GanttChart.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>


<AnchorPane prefHeight="675.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ChartGenerator.GanttChartController">
```

```
   <children>
      <JFXTextField fx:id="yAxisNumber" layoutX="40.0" layoutY="107.0"
prefHeight="27.0" prefWidth="161.0" />
      <Label layoutX="40.0" layoutY="84.0" text="Number of items on Y axis"
/>
      <Label layoutX="41.0" layoutY="238.0" text="Insert the names of items
on Y axis" />
      <Text layoutX="41.0" layoutY="50.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Gantt Chart Inputs">
         <font>
            <Font name="Montserrat Bold" size="28.0" />
         </font>
      </Text>
      <Label layoutX="38.0" layoutY="403.0" text="Insert the start times of
first task" />
      <Label layoutX="313.0" layoutY="403.0" text="Insert the end times of
first task" />
      <Label layoutX="40.0" layoutY="159.0" text="Label of X axis" />
      <Label layoutX="314.0" layoutY="159.0" text="Label of Y axis" />
      <JFXTextField layoutX="41.0" layoutY="184.0" prefHeight="20.0"
prefWidth="150.0" fx:id="xAxisLabel" />
      <JFXTextField fx:id="yAxisLabel" layoutX="314.0" layoutY="184.0"
prefHeight="20.0" prefWidth="150.0" />
      <JFXTextField fx:id="yAxisItemNames" layoutX="41.0" layoutY="262.0"
prefHeight="20.0" prefWidth="530.0" promptText="Space separated names" />
      <JFXTextField fx:id="firstStartTimes" layoutX="38.0" layoutY="427.0"
prefHeight="20.0" prefWidth="250.0" promptText="Space separated times" />
      <JFXTextField fx:id="firstEndTimes" layoutX="313.0" layoutY="427.0"
prefHeight="20.0" prefWidth="250.0" promptText="Space separated times" />
      <Label layoutX="38.0" layoutY="487.0" text="Insert the start times of
second task" />
      <Label layoutX="313.0" layoutY="487.0" text="Insert the end times of
second task" />
      <JFXTextField fx:id="secondStartTimes" layoutX="38.0" layoutY="511.0"
prefHeight="20.0" prefWidth="250.0" promptText="Space separated times" />
      <JFXTextField fx:id="secondEndTimes" layoutX="313.0" layoutY="511.0"
prefHeight="20.0" prefWidth="250.0" promptText="Space separated times" />
      <Label layoutX="41.0" layoutY="320.0" text="Insert the names of
scenarios" />
      <JFXTextField fx:id="scenarioNames" layoutX="41.0" layoutY="344.0"
prefHeight="20.0" prefWidth="530.0" promptText="Space separated names" />
      <JFXButton fx:id="confirmInputs" layoutX="455.0" layoutY="611.0"
onAction="#confirmInputOnClick" style="-fx-border-color: #bbb;" text="Confirm
inputs" />
   </children>
</AnchorPane>
```

GanttChartController.java

```
package ChartGenerator;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXTextField;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;

import javax.swing.*;

public class GanttChartController {

    @FXML private JFXTextField yAxisNumber;
    @FXML private JFXTextField xAxisLabel;
    @FXML private JFXTextField yAxisLabel;
    @FXML private JFXTextField yAxisItemNames;
    @FXML private JFXTextField scenarioNames;
    @FXML private JFXTextField firstStartTimes;
    @FXML private JFXTextField firstEndTimes;
    @FXML private JFXTextField secondStartTimes;
    @FXML private JFXTextField secondEndTimes;
    @FXML private JFXButton confirmInputs;

    @FXML
    void confirmInputOnClick(ActionEvent event) {

        int n = Integer.parseInt(yAxisNumber.getText());
        String xLabel = xAxisLabel.getText();
        String yLabel = yAxisLabel.getText();

        String[] yAxisItems = yAxisItemNames.getText().split(" ");
        String[] scenarios = scenarioNames.getText().split(" ");

        String[] temp = new String[16];

        temp = firstStartTimes.getText().split(" ");
        int[] start1 = new int[temp.length];

        for(int i = 0; i < temp.length; i++) {
            start1[i] = Integer.parseInt(temp[i]);
        }

        temp = firstEndTimes.getText().split(" ");
```

```
        int[] start2 = new int[temp.length];

        for(int i = 0; i < temp.length; i++) {
            start2[i] = Integer.parseInt(temp[i]);
        }

        temp = secondStartTimes.getText().split(" ");
        int[] end1 = new int[temp.length];

        for(int i = 0; i < temp.length; i++) {
            end1[i] = Integer.parseInt(temp[i]);
        }

        temp = secondEndTimes.getText().split(" ");
        int[] end2 = new int[temp.length];

        for(int i = 0; i < temp.length; i++) {
            end2[i] = Integer.parseInt(temp[i]);
        }

        GanttChart example = new GanttChart("Gantt Chart", n, xLabel, yLabel,
yAxisItems, scenarios, start1, start2, end1, end2);
        example.setSize(800, 400);
        example.setLocationRelativeTo(null);
        example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        example.setVisible(true);

    }

}
```

PertChart

PertChart.java

```
package PertChart;

public class PertChart {
    public static void main(String[] args) {
        PertChartGUI gui = new PertChartGUI();
    }
}
```

PertChartGUI.java

```
package PertChart;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

public class PertChartGUI extends JFrame implements ActionListener {

    //Data panel
    private JPanel projectPanel, tablePanel, resourcePanel, graph,
outerPanel, northPanel;
    private JLabel TotalResources, lbl_Early, lbl_Middle, lbl_Late,
lbl_Duration_Interval, projectInfo;
    private JTextField Resources, early, middle, late, durationInterval;
    private Calculations calculations;
    private JButton runSim, simulationGraph;
    private double earlyValue, middleValue, lateValue, durationValue,
resourcesValue;
    private DefaultTableModel preTableModel;
    private JTable preTable;
    private Activity activity;
    private String[] columns = {"Activity ID", "Optimistic", "Most Likely",
"Pessimistic", "Predecessors", "Resources"};
    private String[][] data = {{"", "", "", "", "", "", ""}};

    public PertChartGUI() {
        super("PERT Chart");
        //Create table panel
        tablePanel = new JPanel();
        JLabel activityInfo = new JLabel("Activity Information");
        tablePanel.add(activityInfo);
        preTableModel = new DefaultTableModel(data, columns);
        preTable = new JTable(preTableModel);
        preTable.setPreferredScrollableViewportSize(new Dimension(500, 300));
        preTable.setFillsViewportHeight(true);
        preTable.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                    preTableModel.addRow(new Object[]{"", "", "", "", "",
""});
                }
```

```
        }

        public void keyReleased(KeyEvent e) {
        }

        public void keyTyped(KeyEvent e) {
        }
    });

    tablePanel.add(new JScrollPane(preTable));
    add(tablePanel, BorderLayout.NORTH);

    outerPanel = new JPanel();

    northPanel = new JPanel();
    projectInfo = new JLabel("Project Information");
    northPanel.add(projectInfo);

    outerPanel.add(northPanel, BorderLayout.NORTH);

    //Create resource panel
    resourcePanel = new JPanel();


    lbl_Duration_Interval = new JLabel("Time Unit");
    durationInterval = new JTextField(5);
    resourcePanel.add(lbl_Duration_Interval);
    resourcePanel.add(durationInterval);

    outerPanel.add(resourcePanel, BorderLayout.CENTER);

    //Create project info panel
    projectPanel = new JPanel();
    TotalResources = new JLabel("Total Resources:");
    Resources = new JTextField(5);
    projectPanel.add(TotalResources);
    projectPanel.add(Resources);
    runSim = new JButton("Run Simulation");
    runSim.addActionListener(this);
    runSim.setToolTipText("Calculates an average completion time.");
    projectPanel.add(runSim);

    outerPanel.add(projectPanel, BorderLayout.SOUTH);

    add(outerPanel, BorderLayout.CENTER);
```

```
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        setSize(540, 500);
        //  pack();
        setLocationRelativeTo(null);
        setVisible(true);
    }

    @Override
    /**
     * Detect which button the user just pressed.
     */
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == runSim) {
            runSimulator();

        }

    }//actionPerformed

    /**
     * Using the data from the preTable and Resource textbox, this method
will
     * call the appropriate method to calculate a single Pert table. The
called
     * method should return the results in a 2d String array. The 2D string
     * array will be used in a new table to display the results.
     */
    public void runSimulator() {


        calculations = new Calculations(this.durationInterval.getText(),
Integer.parseInt(this.Resources.getText()));

        for (int i = 0; i < preTableModel.getRowCount(); ++i) {
            try {
                earlyValue = Double.parseDouble((String)
preTableModel.getValueAt(i, 1));
                middleValue = Double.parseDouble((String)
preTableModel.getValueAt(i, 2));
                lateValue = Double.parseDouble((String)
preTableModel.getValueAt(i, 3));
                resourcesValue = 0.0;
                String pred = "";
                if(preTableModel.getValueAt(i, 4) != null){
                    pred = (String) preTableModel.getValueAt(i, 4);
                }
```

```
                if(preTableModel.getValueAt(i, 5) != null &&
((String)preTableModel.getValueAt(i, 5)).length() > 0){
                    resourcesValue =
Double.parseDouble((String)preTableModel.getValueAt(i, 5));
                }

                activity = new Activity((String) preTableModel.getValueAt(i,
0), earlyValue, middleValue, lateValue, pred , resourcesValue);
                calculations.activities.add(activity);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Activity information has
been entered incorrectly", "Error", JOptionPane.ERROR_MESSAGE);
            }

        }

        calculations.run();

        Results results = new Results(calculations);
    }
}
```

Activity.java

```
package PertChart;

import java.util.ArrayList;

public class Activity {


    // user input
    private String activityId;
    private double a;
    private double m;
    private double b;
    private double resources;

    private String predecessorsInput;
    private ArrayList<Activity> predecessors;
    private ArrayList<Activity> successors;

    //To be calculated from doing the CPM portio
    private float totalSlack;
    private float freeSlack;
```

```
    private float interferingSlack;
    private float independentSlack;
    private double startTime;
    private double completionTime;
    private double expectedTime;
    private double averageStartTime;
    private double averageCompletionTime;
    private double averageVariance;

    Activity(String aId, double a, double m, double b, String p, double r) {
        this.activityId = aId;
        this.a = a;
        this.m = m;
        this.b = b;
        this.predecessorsInput = p;
        this.resources = r;
        this.predecessors = new ArrayList();
        this.successors = new ArrayList();
    }

    public Double getA() {
        return a;
    }

    public void setA(double a) {
        this.a = a;
    }

    public Double getM() {
        return m;
    }

    public void setM(double m) {
        this.m = m;
    }

    public Double getB() {
        return b;
    }

    public void setB(double b) {
        this.b = b;
    }

    public Double getStartTime() {
        return this.startTime;
    }
```

```
public void setStartTime(double a) {
    this.startTime = a;
}

public Double getCompletionTime() {
    return this.completionTime;
}

public void setCompletionTime(double a) {
    this.completionTime = a;
}

public ArrayList<Activity> getPredecessors(){
    return predecessors;
}

public ArrayList<Activity> getSuccessors(){
    return successors;
}

public Double getResources() {
    return resources;
}

public void setResources(double r) {
    this.resources = r;
}

public String getPredecessorsInput() {
    return this.predecessorsInput;
}

public void setPredecessorsInput(String p) {
    this.predecessorsInput = p;
}

public String getActivityId() {
    return this.activityId;
}

public void setActivityId(String id) {
    this.activityId = id;
}

public float getTotalSlack() {
    return this.totalSlack;
```

```
    }

    public void setTotalSlack(float s) {
        this.totalSlack = s;
    }

    public float getFreeSlack() {
        return this.freeSlack;
    }

    public void setFreeSlack(float s) {
        this.freeSlack = s;
    }

    public float getInterferingSlack() {
        return this.interferingSlack;
    }

    public void setInterferingSlack(float s) {
        this.interferingSlack = s;
    }

    public float getIndependentSlack() {
        return this.independentSlack;
    }

    public void setIndependentSlack(float s) {
        this.independentSlack = s;
    }

    public double getExpectedTime() {
        return this.expectedTime;
    }

    public void setExpectedTime(double d) {
        this.expectedTime = d;
    }

    public double getAverageStartTime() {
        return this.averageStartTime;
    }

    public void setAverageStartTime(double d) {
        this.averageStartTime = d;
    }

    public double getAverageCompletionTime() {
```

```
        return this.averageCompletionTime;
    }

    public void setAverageCompletionTime(double d) {
        this.averageCompletionTime = d;
    }

    public double getAverageVariance() {
        return this.averageVariance;
    }

    public void setAverageVariance(double d) {
        this.averageVariance = d;
    }
}
```

Calculations.java

```
package PertChart;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;
import javax.swing.JOptionPane;

public class Calculations {

    protected ArrayList<Activity> activities;
    private Project project;
    private final ArrayList<Activity> completedActvities =  new ArrayList();
    private final ArrayList<Activity> activitiesQueue = new ArrayList();
    private Random random = new Random();
    private ArrayList<Activity> activityResults;
    private HashMap<String, Double[]> activityMap = new HashMap<>();
    private ArrayList<Activity> simActivities;
    private String timeUnit;
    private double totalResources;
    private double averageProjectCompletionTime = 0.0;
    private double simLength = 1000.0; // 5 million runs
    private int arraySize = (int)this.simLength;
    private double[] projectCompletionTimes = new
double[(int)this.simLength];
    private double maxProjectCompletionTime;
    private double minProjectCompletionTime;
```

```
    public Calculations(String di, int tr) {
        this.timeUnit = di;
        this.totalResources = tr;
        this.project = new Project(di, tr);
        this.activities = new ArrayList();
    }

    public void run() {
        determinePredecessors();
        determineSuccessors();
        generateHashMap();

        // run simulation
        //System.out.print("Simulation running...");
        //int scaler = 0;
        for (int i = 0; i < this.simLength; i++) {
            /*if (((double)i + 1.0) % (this.simLength / 10.0) == 0) {
                if (scaler + 1 < 10) {
                    scaler = scaler + 1;
                    System.out.print((10 * scaler)  + "%...");
                }
                else {
                    System.out.print(100 + "%!!!");
                    System.out.println("");
                }
            }*/
            this.simActivities = this.activities;
            calculateActvityCompletionTime();
            calculateProjectCompletionTime(i);
        }

        // calculate the final results
        Iterator it = this.activityMap.entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry a = (Map.Entry)it.next();
            Double[] d = (Double[])a.getValue();
            d[0] = d[0] / this.simLength;
            d[1] = d[1] / this.simLength;
            d[2] = d[2] / this.simLength;
            a.setValue(d);
        }

        // max and min project completion times
        this.averageProjectCompletionTime = this.averageProjectCompletionTime
/ this.simLength;
        int pl = this.projectCompletionTimes.length;
```

```
        double minTime = this.averageProjectCompletionTime;
        double maxTime = this.averageProjectCompletionTime;
        for (int i = 0; i < pl; i++) {
            double pTime = this.projectCompletionTimes[i];
            if (maxTime < pTime) {
                maxTime = pTime;
            }
            if (minTime > pTime && pTime > 0.0) {
                minTime = pTime;
            }
        }
        this.maxProjectCompletionTime = maxTime;
        this.minProjectCompletionTime = minTime;
    }

    public void generateHashMap() {
        for (Activity a: this.activities) {
            Double[] d = {0.0, 0.0, 0.0};
            this.activityMap.put(a.getActivityId(), d);
        }
    }

    public Activity getActivityById(String s) {
        Activity a = null;
        for (Activity b: this.activities) {
            if (s.equalsIgnoreCase(b.getActivityId()) == false) {
               continue;
            }
            else {
                a = b;
                break;
            }
        }
        return a;
    }

    public void determinePredecessors() {
        for (Activity a: this.activities) {
                String p = a.getPredecessorsInput();
                String[] pList = p.split(",");
                for (String s: pList) {
                    if (s.equalsIgnoreCase(a.getActivityId()) == false) {
                        Activity pa = getActivityById(s);
                        if (pa != null) {
                            a.getPredecessors().add(pa);
                        }
                    }
```

```
                }
            }
        }

        public void determineSuccessors() {
            for (Activity a: this.activities) {
                for (Activity b: this.activities) {
                    ArrayList<Activity> pList  = b.getPredecessors();
                    for (Activity c: pList) {
                        if
(c.getActivityId().equalsIgnoreCase(a.getActivityId())) {
                            a.getSuccessors().add(b);
                            break;
                        }
                    }
                }
            }
        }

        public void calculateActvityCompletionTime() {
            for (Activity a: this.simActivities) {
                double amb = (a.getB() + (4.0 * a.getM()) + a.getA()) / 6.0;
                double lateTimeRange = amb;
                double earlyTimeRange = amb;
                if (amb < a.getB()) {
                    lateTimeRange = (amb + a.getB()) / 2.0;
                }
                if (amb > a.getA()) {
                    earlyTimeRange = (amb + a.getA()) / 2.0;
                }
                double range = lateTimeRange - earlyTimeRange;
                double scaled = random.nextGaussian() * range;
                double expectedTime = scaled + earlyTimeRange;
                a.setExpectedTime(expectedTime);

                // calculate variance
                double v = Math.pow((a.getB() - a.getA()), 2) / 36.0;
                a.setAverageVariance(v);
            }
        }

        public void calculateProjectCompletionTime(int currentSimCycle) {
            double projectCompletionTime = 0.0;
            // push all activities that have no predecessors into the queue
            for (Activity a: this.simActivities) {
                ArrayList<Activity> pList  = a.getPredecessors();
                if (pList.size() > 0) {
```

```
                continue;
            }
            else {
                a.setStartTime(0.0);
                activitiesQueue.add(a);
            }
        }
        while (activitiesQueue.size() > 0) {
            Activity a = activitiesQueue.get(0);
            if (a.getResources() <= this.totalResources) {
                this.totalResources = this.totalResources - a.getResources();
            }
            else {
                JOptionPane.showMessageDialog(null, "There are not enough
resources for this project.");
                System.exit(0);
            }
            double completionTime = a.getStartTime() + a.getExpectedTime();
            a.setCompletionTime(completionTime);
            completedActvities.add(a);

            // update our hash map for the final results
            double numOfPredecessors = a.getPredecessors().size();
            Double[] d = this.activityMap.get(a.getActivityId());
            d[0] = d[0] + a.getAverageVariance();
            if (numOfPredecessors > 1) {
                d[1] = d[1] + (a.getStartTime() / numOfPredecessors);
                d[2] = d[2] + (completionTime / numOfPredecessors);
            }
            else {
                d[1] = d[1] + a.getStartTime();
                d[2] = d[2] + completionTime;
            }
            this.activityMap.put(a.getActivityId(), d);

            // check if this activities completion time is higher than the
project's
            if (completionTime > projectCompletionTime) {
                projectCompletionTime = completionTime;
            }

            // release the resources
            this.totalResources = this.totalResources + a.getResources();

            // push next activities into queue whose successors have finished
            for (Activity successor: a.getSuccessors()) {
                ArrayList<Activity> predecessorList =
```

```
successor.getPredecessors();
                int pListL = predecessorList.size();
                int completedPredecessors = 0;
                double maxPredecessorCompletionTime = 0.0;
                for (int i = 0; i < pListL; i++) {
                    Activity predecessor = predecessorList.get(i);
                    if
(activityHasBeenCompleted(predecessor.getActivityId())) {
                        completedPredecessors = completedPredecessors + 1;
                        if (maxPredecessorCompletionTime <
predecessor.getCompletionTime()) {
                            maxPredecessorCompletionTime =
predecessor.getCompletionTime();
                        }
                    }
                }

                // if all predecessors have been completed then push the
successor into the queue
                if (completedPredecessors == pListL) {
                    // if there are enough resources available
                    if (successor.getResources() <= this.totalResources) {
                        successor.setStartTime(maxPredecessorCompletionTime);
                        activitiesQueue.add(successor);
                    }
                }
            }
            // its now safe to remove the activity
            activitiesQueue.remove(0);
        }
        this.averageProjectCompletionTime = this.averageProjectCompletionTime
+ projectCompletionTime;
        projectCompletionTimes[currentSimCycle] = projectCompletionTime;
        project.setCompletionTime(projectCompletionTime);
    }

    public boolean activityHasBeenCompleted(String id) {
        boolean completed = false;
        int len = completedActvities.size();
        for (int i = 0; i < len; i++) {
            Activity completedActivity = completedActvities.get(i);
            if (completedActivity.getActivityId().equalsIgnoreCase(id) ==
false) {
                continue;
            }
            else {
                completed = true;
```

```
                break;
            }
        }
        return completed;
    }

    public HashMap<String, Double[]> getActivityResults() {
        return this.activityMap;
    }

    public Double getProjectCompletionTime() {
        return this.averageProjectCompletionTime;
    }

    public double[] getProjectCompletionTimes() {
        return this.projectCompletionTimes;
    }

    public Double getMaxProjectCompletionTime() {
        return this.maxProjectCompletionTime;
    }

    public Double getMinProjectCompletionTime() {
        return this.minProjectCompletionTime;
    }

    public String getTimeUnit() {
        return this.timeUnit;
    }
}
```

Histogram.java

```
package PertChart;

import java.io.File;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.statistics.HistogramDataset;
import org.jfree.data.statistics.HistogramType;
import org.jfree.ui.ApplicationFrame;

/**
 * An example found here:
```

```
 * http://java2s.com/Code/Java/Chart/JFreeChartXYSeriesDemo3.htm
 */
public class Histogram extends ApplicationFrame {

    private double[] data;
    private int bins;

    public Histogram(final String title, double[] data, int bins) {
        super(title);
        this.data = data;
        this.bins = bins;
    }

    public JFreeChart createHistorgram(double[] data) {

        HistogramDataset dataset = new HistogramDataset();
        dataset.setType(HistogramType.RELATIVE_FREQUENCY);
        dataset.addSeries("Histogram", data, this.bins);
        String plotTitle = "Project Completion Times";
        String xaxis = "Completion Times";
        String yaxis = "";
        PlotOrientation orientation = PlotOrientation.VERTICAL;
        boolean show = false;
        boolean toolTips = false;
        boolean urls = false;
        JFreeChart chart = ChartFactory.createHistogram(plotTitle, xaxis,
yaxis,
                    dataset, orientation, show, toolTips, urls);
        int width = 500;
        int height = 300;
        try {
            ChartUtilities.saveChartAsPNG(new File("histogram.png"), chart,
width, height);
        } catch (Exception e) {
            System.out.println(e);
        }
        return chart;
    }

    public void saveChart() {
        JFreeChart chart = createHistorgram(this.data);
        final ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 300));
        setContentPane(chartPanel);
    }
}
```

Project.java

```
package PertChart;

public class Project {
    // user input
    private String durationInterval;
    private double resources;

    private double completionTime;
    private double probabilityByDeadline;
    private float totalMinSlack;

    private double specifiedDeadline;
    private double probabilityOnTime;
    private double expectedDuration;
    private double expectedDurationStandardDeviation;

    private double averageCompletionTime;

    Project(String di, double resources) {
        this.resources = resources;
        this.durationInterval = di;
    }


    public double getResources() {
        return resources;
    }

    public void setResources(double resources) {
        this.resources = resources;
    }

    public String getDurationInterval() {
        return durationInterval;
    }

    public void setDurationInterval(String durationInterval) {
        this.durationInterval = durationInterval;
    }

    public Double getCompletionTime() {
        return this.completionTime;
    }
```

```
public void setCompletionTime(Double t) {
    this.completionTime = t;
}

public Double getProbabilityOnTime() {
    return this.probabilityOnTime;
}

public void setProbabilityOnTime(Double t) {
    this.probabilityOnTime = t;
}

public Double getProbabilityByDeadline() {
    return this.probabilityByDeadline;
}

public void setProbabilityByDeadline(Double t) {
    this.probabilityByDeadline = t;
}

public Float getTotalMinSlack() {
    return this.totalMinSlack;
}

public void setTotalMinSlack(Float s) {
    this.totalMinSlack = s;
}

public Double getSpecifiedDeadline() {
    return this.specifiedDeadline;
}

public void setSpecifiedDeadline(Double t) {
    this.specifiedDeadline = t;
}

public Double getExpectedDuration() {
    return this.expectedDuration;
}

public void setExpectedDuration(Double t) {
    this.expectedDuration = t;
}

public Double getExpectedDurationStandardDeviation() {
    return this.expectedDurationStandardDeviation;
```

```
    }

    public void setExpectedDurationStandardDeviation(Double t) {
        this.expectedDurationStandardDeviation = t;
    }

    public Double getAverageCompletionTime() {
        return this.averageCompletionTime;
    }

    public void setAverageCompletionTime(Double t) {
        this.averageCompletionTime = t;
    }
}
```

Results.java

```
package PertChart;

import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import javax.swing.*;
import javax.swing.table.*;

public class Results extends JFrame implements ActionListener
{
        //Data panel
    private JPanel resourcePanel, timePanel, tablePanel;
    private JLabel TotalResources, resNum, TotalTime, timeNum;

    //private JButton single, runSim;
    private DecimalFormat df;
    private DefaultTableModel resultsModel;
    private JTable resultsTable;
    //private String[] columns = {"Activity ID", "Predecessors", "A", "M",
"B", "Variance", "Start Time", "Completion Time"};


    Results(Calculations calculations)
    {
        super("Results");
```

```
        HashMap<String, Double[]> map = calculations.getActivityResults();

        Double projectTime = calculations.getProjectCompletionTime();

        DefaultTableColumnModel columns = new DefaultTableColumnModel();
        //create and define columns
        TableColumn column1 = new TableColumn(0);
        column1.setHeaderValue("Activity ID"); // set column name
        column1.setPreferredWidth(50); //set column width

        TableColumn column2 = new TableColumn(1);
        column2.setHeaderValue("Predecessors"); // set column name
        column2.setPreferredWidth(50); //set column width

        TableColumn column3 = new TableColumn(2);
        column3.setHeaderValue("A"); // set column name
        column3.setPreferredWidth(15); //set column width

        TableColumn column4 = new TableColumn(3);
        column4.setHeaderValue("M"); // set column name
        column4.setPreferredWidth(15); //set column width

        TableColumn column5 = new TableColumn(4);
        column5.setHeaderValue("B"); // set column name
        column5.setPreferredWidth(15); //set column width

        TableColumn resourceColumn = new TableColumn(5);
        resourceColumn.setHeaderValue("Resources"); // set column name
        resourceColumn.setPreferredWidth(50); //set column width

        TableColumn column6 = new TableColumn(6);
        column6.setHeaderValue("Variance (s\u00B2)"); // set column name
        column6.setPreferredWidth(50); //set column width

        TableColumn column7 = new TableColumn(7);
        column7.setHeaderValue("Start Time"); // set column name
        column7.setPreferredWidth(50); //set column width

        TableColumn column8 = new TableColumn(8);
        column8.setHeaderValue("Completion Time"); // set column name
        column8.setPreferredWidth(70); //set column width

        columns.addColumn(column1);
        columns.addColumn(column2);
        columns.addColumn(column3);
        columns.addColumn(column4);
        columns.addColumn(column5);
```

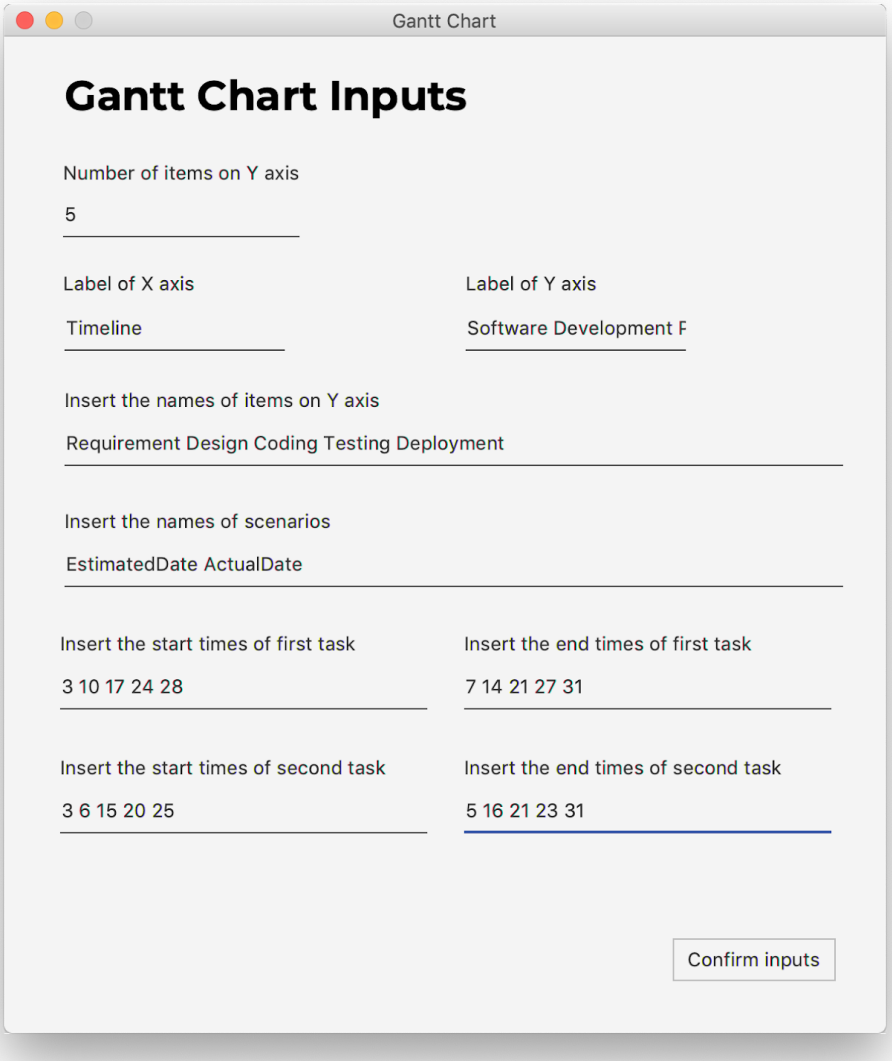
```
        columns.addColumn(resourceColumn);
        columns.addColumn(column6);
        columns.addColumn(column7);
        columns.addColumn(column8);

        resultsModel = new DefaultTableModel(0 , 9);

        int l = calculations.activities.size();
        for (int i = 0; i < l; i++) {
            Activity a = calculations.activities.get(i);
            Iterator it = map.entrySet().iterator();
            while (it.hasNext()) {
                Map.Entry e = (Map.Entry)it.next();
                if (a.getActivityId() == e.getKey()) {
                    Double[] d = (Double[])e.getValue();
                    df = new DecimalFormat("####.####");
                    double var = Double.parseDouble(df.format(d[0]));
                    double start = Double.parseDouble(df.format(d[1]));
                    double end = Double.parseDouble(df.format(d[2]));
                    Object[] row = {a.getActivityId(),
a.getPredecessorsInput(), a.getA(), a.getM(), a.getB(), a.getResources(),
var, start, end};
                    resultsModel.addRow(row);
                }
            }
        }

        //Create table panel
        tablePanel = new JPanel();
        tablePanel.setSize(550, 300);

        resultsTable = new JTable(resultsModel, columns);
        resultsTable.setPreferredScrollableViewportSize(new Dimension(650,
300));
        resultsTable.setFillsViewportHeight(true);

        tablePanel.add(new JScrollPane(resultsTable));
        add(tablePanel, BorderLayout.WEST);


        timePanel = new JPanel();
        String totalTimes = "Project Completion Time: ";
        Double formatedProjectTime =
Double.parseDouble(df.format(projectTime));
        totalTimes = totalTimes + formatedProjectTime + " " +
calculations.getTimeUnit();
```

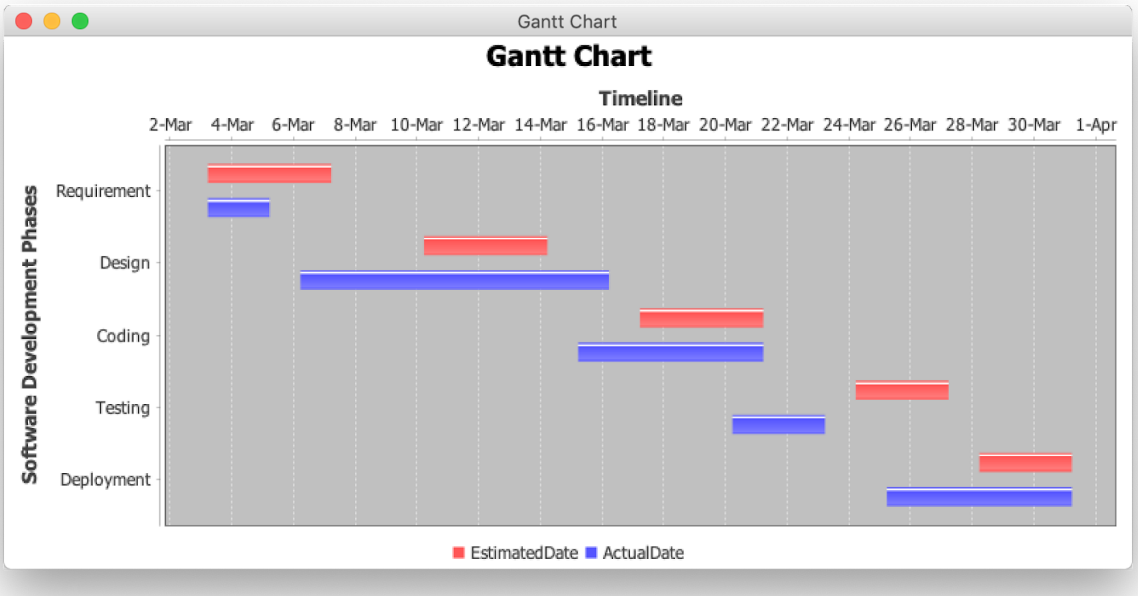
```
        String earliestTime = " | Earliest Project Completion Time: ";
        Double formatedEarliestTime =
Double.parseDouble(df.format(calculations.getMinProjectCompletionTime()));
        earliestTime = earliestTime + formatedEarliestTime + " " +
calculations.getTimeUnit();

        String latestTime = " | Latest Project Completion Time: ";
        Double formatedLatestTime =
Double.parseDouble(df.format(calculations.getMaxProjectCompletionTime()));
        latestTime = latestTime + formatedLatestTime + " " +
calculations.getTimeUnit();

        String resultsReport = totalTimes + earliestTime + latestTime;
        JLabel timesLabel = new JLabel(resultsReport);
        timePanel.add(timesLabel);
        add(timePanel, BorderLayout.SOUTH);

        JPanel graphPanel = new JPanel();
        graphPanel.setSize(550, 300);
        Histogram h = new Histogram("Project Histogram",
calculations.getProjectCompletionTimes(), 5);
        h.saveChart();
        graphPanel.add(h.getContentPane());
        add(graphPanel, BorderLayout.EAST);

        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        setSize(1200,600);
        //pack();
        setVisible(true);
    }


    @Override
    /**
     * Detect which button the user just pressed.
     */
    public void actionPerformed(ActionEvent e)
    {

    }//actionPerformed
}
```

8 TESTING

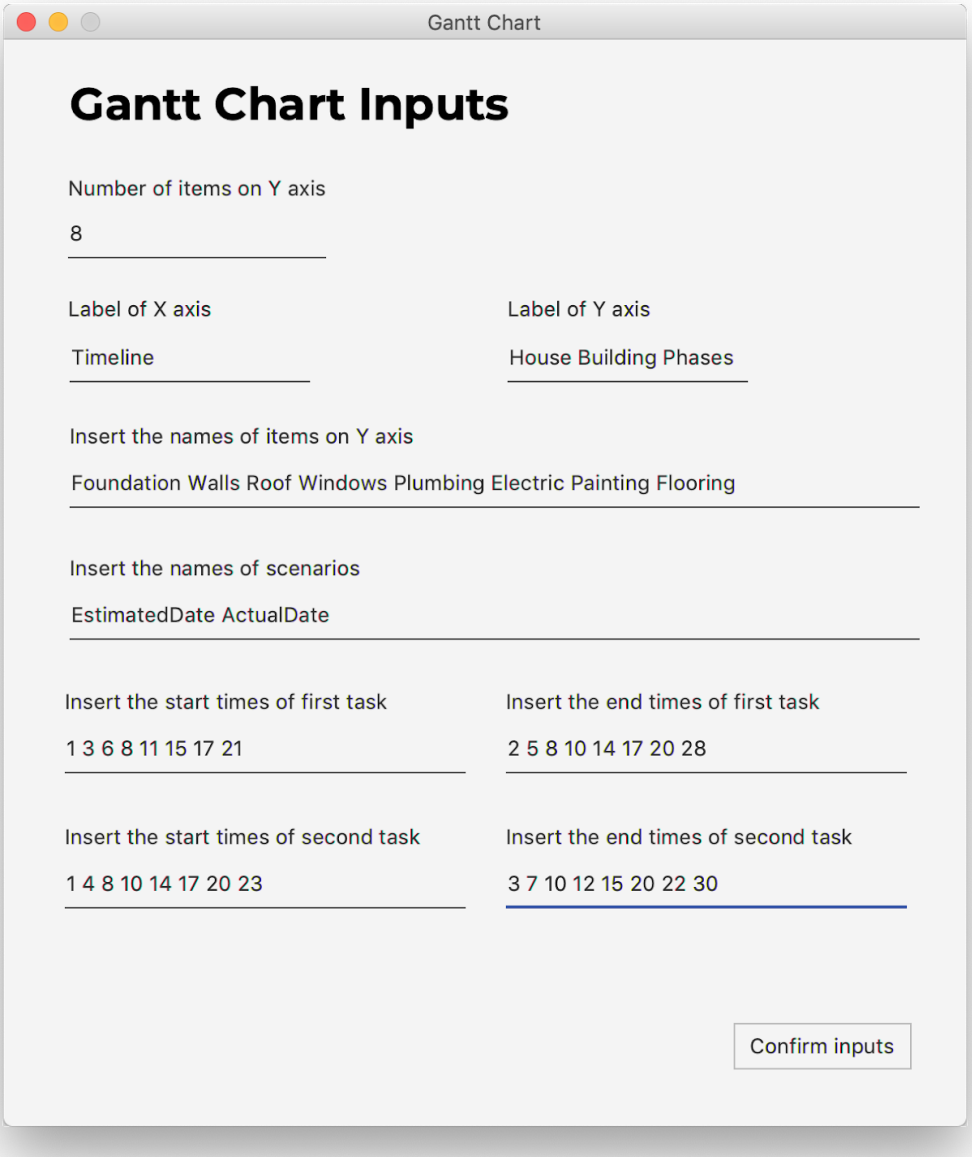
8.1 Gantt Chart Module Testing

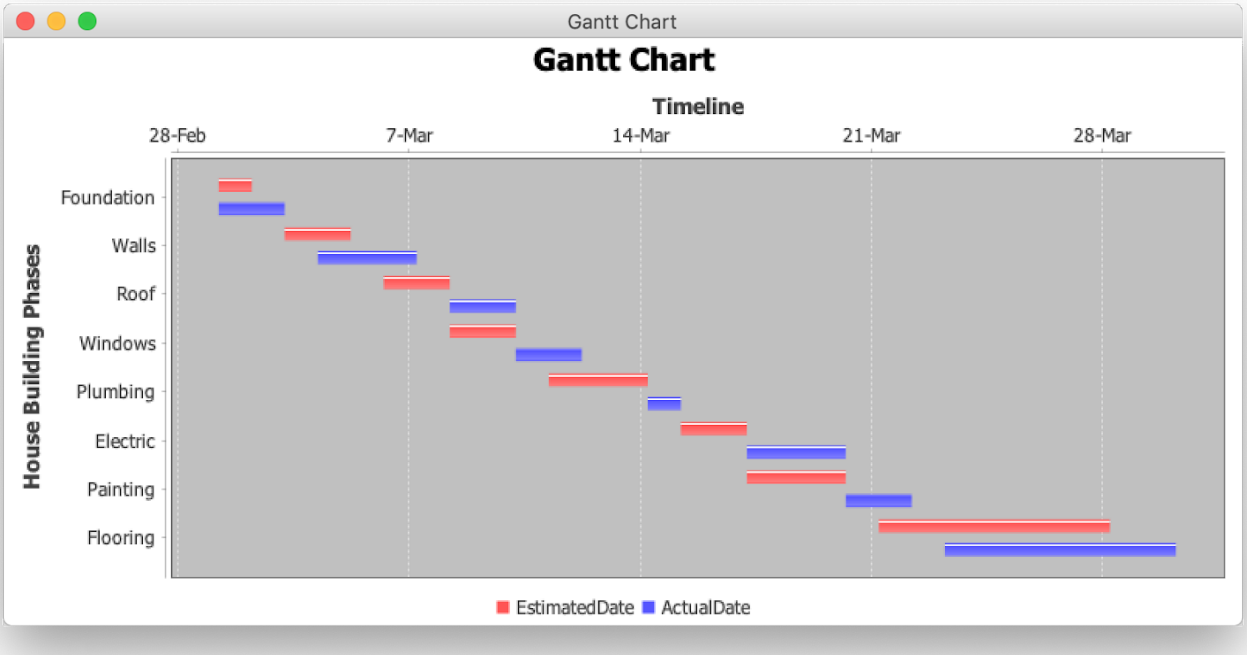
8.1.1 Test 1




Gantt Chart
Gantt Chart
Timeline
2-Mar
4-Mar
6-Mar
8-Mar
10-Mar
12-Mar
14-Mar
16-Mar
18-Mar
20-Mar
22-Mar
24-Mar
26-Mar
28-Mar
30-Mar
1-Apr
Software Development Phases
Requirement
Design
Coding
Testing
Deployment
EstimatedDate
ActualDate

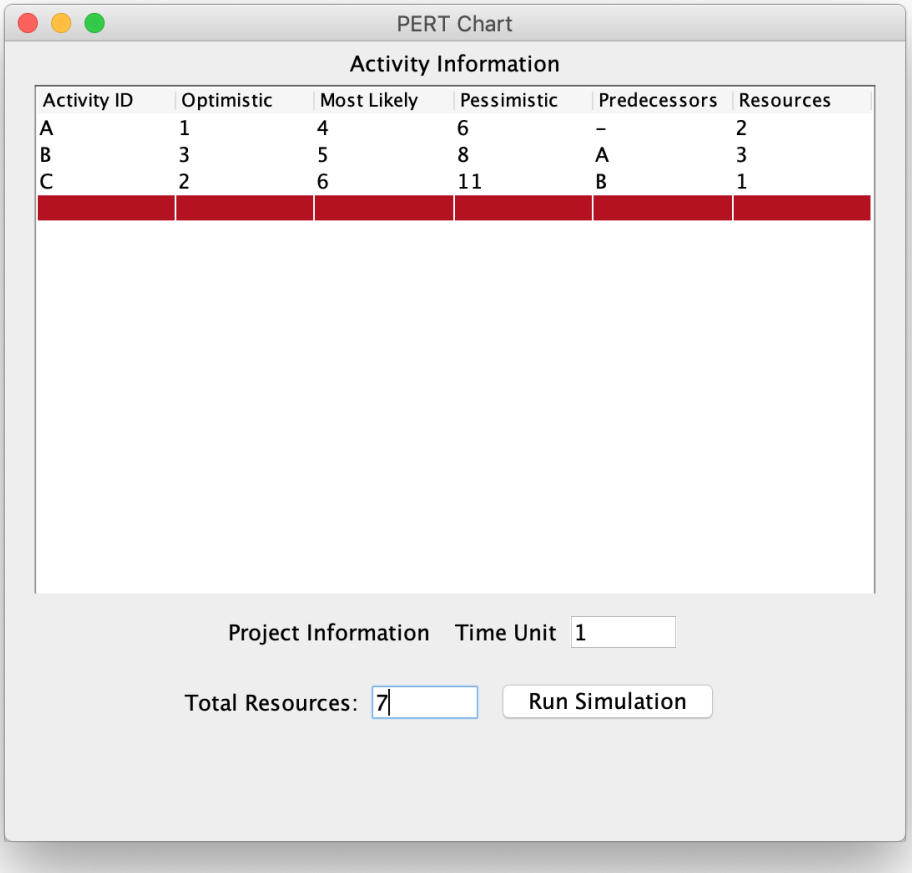
8.1.2 Test 2

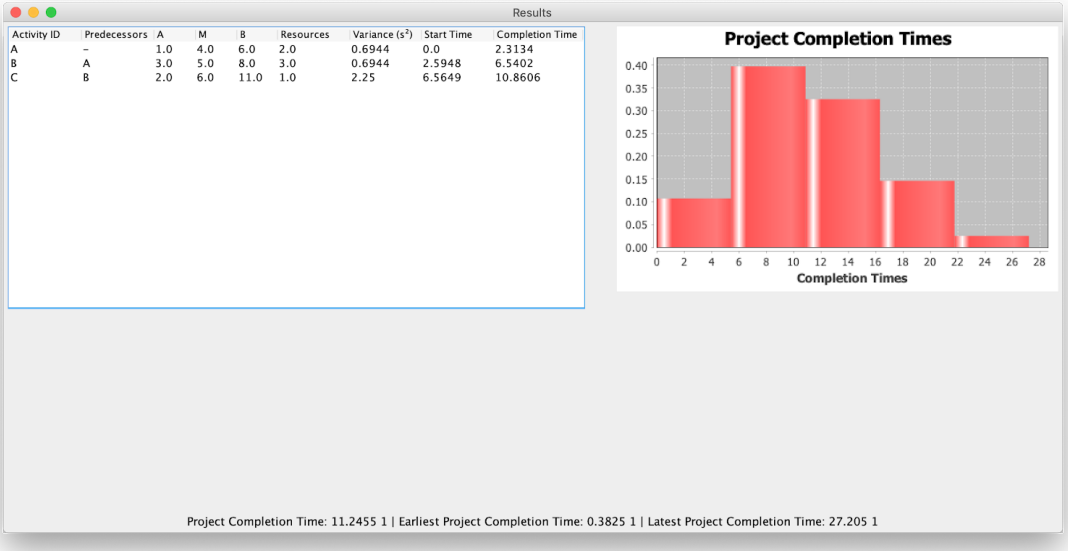

Gantt Chart
Gantt Chart Inputs
Number of items on Y axis
8
Label of X axis
Timeline
Label of Y axis
House Building Phases
Insert the names of items on Y axis
Foundation Walls Roof Windows Plumbing Electric Painting Flooring
Insert the names of scenarios
EstimatedDate ActualDate
Insert the start times of first task
1 3 6 8 11 15 17 21
Insert the end times of first task
2 5 8 10 14 17 20 28
Insert the start times of second task
1 4 8 10 14 17 20 23
Insert the end times of second task
3 7 10 12 15 20 22 30
Confirm inputs


Gantt Chart
Gantt Chart
Timeline
28-Feb
7-Mar
14-Mar
21-Mar
28-Mar
House Building Phases
Foundation
Walls
Roof
Windows
Plumbing
Electric
Painting
Flooring
EstimatedDate
ActualDate

8.2 PERT Chart Module Testing

8.2.1 Test 1





8.2.2 Test 2

PERT Chart

Activity Information

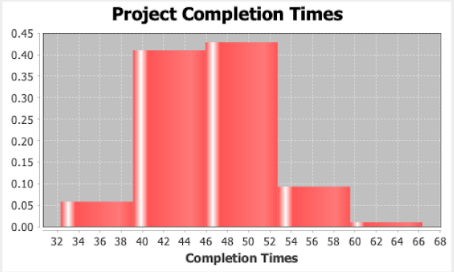
Activity ID	Optimistic	Most Likely	Pessimistic	Predecessors	Resources
A	2	4	6	–	2
B	3	6	9	A	3
C	8	10	12	A	3
D	9	12	15	B	2
E	8	9	10	C	3
F	16	21	26	D,E	1
G	19	22	25	D,E	2
H	2	5	8	F	2
I	1	3	5	G	3

Project Information Time Unit 1

Total Resources: 22 Run Simulation

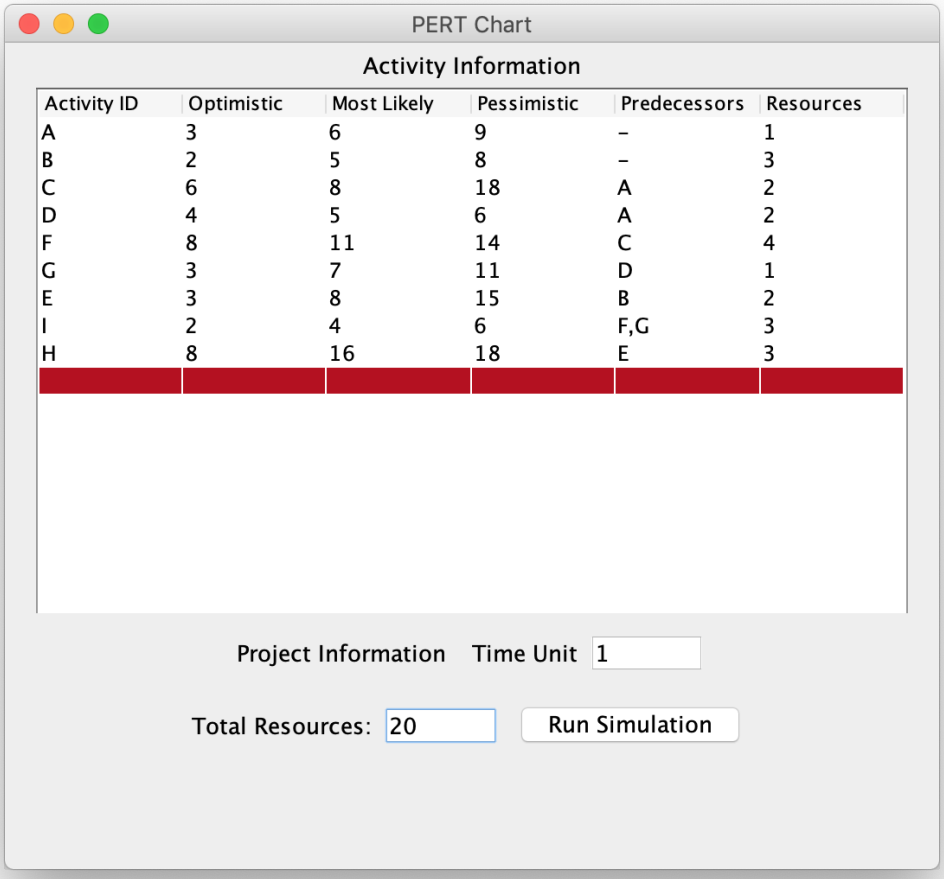
Results

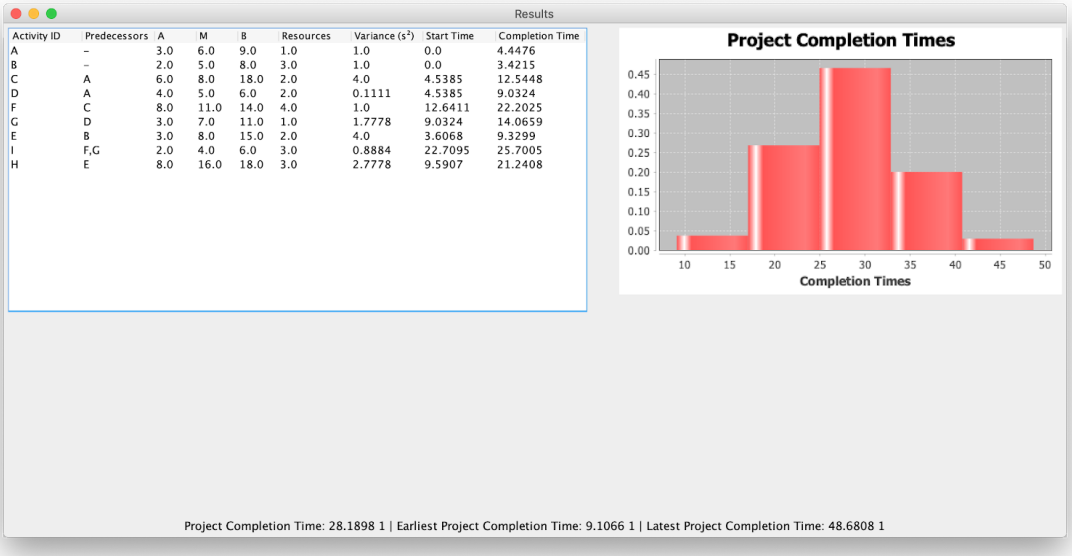
Activity ID	Predecessors	A	M	B	Resources	Variance (s^2)	Start Time	Completion Time
A	–	2.0	4.0	6.0	2.0	0.4444	0.0	2.98
B	A	3.0	6.0	9.0	3.0	1.0	3.0383	7.5582
C	A	8.0	10.0	12.0	3.0	0.4444	3.0383	12.0972
D	B	9.0	12.0	15.0	2.0	1.0	7.5709	18.0838
E	C	8.0	9.0	10.0	3.0	0.1111	12.0972	20.6033
F	D,E	16.0	21.0	26.0	1.0	5.5528	21.512	40.1035
G	D,E	19.0	22.0	25.0	2.0	1.999	21.512	42.0298
H	F	2.0	5.0	8.0	2.0	1.999	80.2071	87.143
I	G	1.0	3.0	5.0	3.0	0.8884	84.0596	87.821



Project Completion Time: 46.5328 1 | Earliest Project Completion Time: 32.3192 1 | Latest Project Completion Time: 66.3381 1

8.2.3 Test 3


PERT Chart
Activity Information
Activity ID | Optimistic | Most Likely | Pessimistic | Predecessors | Resources
A | 3 | 6 | 9 | – | 1
B | 2 | 5 | 8 | – | 3
C | 6 | 8 | 18 | A | 2
D | 4 | 5 | 6 | A | 2
F | 8 | 11 | 14 | C | 4
G | 3 | 7 | 11 | D | 1
E | 3 | 8 | 15 | B | 2
I | 2 | 4 | 6 | F,G | 3
H | 8 | 16 | 18 | E | 3
Project Information
Time Unit 1
Total Resources: 20
Run Simulation


Results
Activity ID | Predecessors | A | M | B | Resources | Variance (s²) | Start Time | Completion Time
A | – | 3.0 | 6.0 | 9.0 | 1.0 | 1.0 | 0.0 | 4.4476
B | – | 2.0 | 5.0 | 8.0 | 3.0 | 1.0 | 0.0 | 3.4215
C | A | 6.0 | 8.0 | 18.0 | 2.0 | 4.0 | 4.5385 | 12.5448
D | A | 4.0 | 5.0 | 6.0 | 2.0 | 0.1111 | 4.5385 | 9.0324
F | C | 8.0 | 11.0 | 14.0 | 4.0 | 1.0 | 12.6411 | 22.2025
G | D | 3.0 | 7.0 | 11.0 | 1.0 | 1.7778 | 9.0324 | 14.0659
E | B | 3.0 | 8.0 | 15.0 | 2.0 | 4.0 | 3.6068 | 9.3299
I | F,G | 2.0 | 4.0 | 6.0 | 3.0 | 0.8884 | 22.7095 | 25.7005
H | E | 8.0 | 16.0 | 18.0 | 3.0 | 2.7778 | 9.5907 | 21.2408
Project Completion Times
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10 15 20 25 30 35 40 45 50
Completion Times
Project Completion Time: 28.1898 1 | Earliest Project Completion Time: 9.1066 1 | Latest Project Completion Time: 48.6808 1

9 RESULT ANALYSIS

The project has been completed and can easily be incorporated in personal projects or commercial projects. The results given by the chart modules help understand and visualise the deadlines and help create an efficient schedule for the whole task.

This will lead to a more efficient and quick project management.

10 CONCLUSION

The result of visualising different project management charts has been completed. Work Breakdown Structure was not implementable by our team. Future work may be done to generate a Work Breakdown Structure charts.

11 REFERENCES

1. **Microsoft Excel** - Has a wide range of charting tools that we used to understand the requirements.
2. **JFreeChart** - The most widely used charting library among Java developers.
3. **Bright Hub PM, brighthubpm.com** - Overview of charts used in project management.
4. **GraphStream** - A Java library for the modeling and analysis of dynamic graphs. Can be used to generate, import, export, measure and layout the graphs.
5. **JGraphX** - A Java Swing diagramming (graph visualisation) library.
6. **graphviz** - Java wrapper around the Ecmascripten Javascript compile of graphviz.
7. **Office TIMELINE, officetimeline.com** - Project management charts and other visualisation tools for project managers.
8. **Google Charts** - JavaScript based charting library meant to enhance web applications by adding interactive charting capability.
9. **Stackoverflow** - Question and answer site for professional and enthusiast programmers.